МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Курганский государственный университет»

Кафедра программного обеспечения автоматизированных систем

АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ

Методические указания
к выполнению курсовых работ
для студентов направления 231000.62

Курган 2014

Кафедра: «Программное обеспечение автоматизированных систем»

Дисциплина: «Программная инженерия»
(направление 231000.62).

Составил: канд. техн. наук, доцент А.М. Семахин.

Утверждены на заседании кафедры «30» января 2014 г.

Рекомендованы методическим советом университета « 8 » апреля 2014 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Современная методология программирования предполагает, что выбор структур данных и запись алгоритма на языке программирования важны и заслуживают одинакового внимания. Решение о том, как представлять данные, невозможно принимать без понимания того, какие алгоритмы будут к ним применяться, и наоборот, выбор алгоритма зависит от строения данных, к которым он применяется. Квалифицированный разработчик программных средств должен владеть современными технологиями создания, знать несколько языков программирования, уметь выбирать для решения задачи подходящие структуры данных и наилучшие алгоритмы.

При разработке компьютерной программы реализуется метод, который был разработан для решения задачи. Метод не зависит от конкретного компьютера и будет пригодным для разных компьютеров и языков программирования. Для определения способа решения задачи необходимо исследовать метод, а не программу. Термин «алгоритм» используется для описания метода решения задачи, пригодного для реализации в виде компьютерной программы.

Алгоритмы касаются методов организации данных, участвующих в вычислениях. Созданные таким образом объекты называются структурами данных. Они существуют в качестве субпродуктов или конечных продуктов алгоритмов и являются центральными объектами изучения в компьютерных науках. Простые алгоритмы могут порождать сложные структуры данных, и наоборот, сложные алгоритмы могут использовать простые структуры данных.

Существует множество подходов при решении задач на ЭВМ. В случае решения простых задач выбор подхода не имеет особого значения. При решении сложных задач требуются методы, при которых время или память используются с максимальной эффективностью.

Изучение алгоритмов необходимо для обеспечения экономии ресурсов. В программных приложениях, обрабатывающих миллионы объектов, оказывается возможным ускорить работу программы в миллионы раз, используя хорошо разработанный алгоритм. Вложение дополнительных денежных средств или времени для приобретения и установки нового компьютера позволяет ускорить работу программы в 10-100 раз. Тщательно разработанный алгоритм – эффективная часть процесса решения сложной задачи в любой области применения.

При разработке большой или сложной компьютерной программы значительные усилия затрачиваются на выяснение и определение задачи, которая должна быть решена, осознание ее сложности и разбиение ее на простые подзадачи, решение которых можно легко реализовать. Реализация многих из алгоритмов, требующихся после разбиения, тривиальна. Однако в большинстве случаев существует несколько алгоритмов, выбор которых критичен, поскольку для их выполнения требуется большая часть системных ресурсов.

Использование готовых программ в компьютерных системах становится все более распространенным. Например, библиотека стандартных шаблонов (Standard Template Library, STL) C++ содержит реализации множества базовых алгоритмов. Однако реализация простых версий основных алгоритмов позволит лучше понять и, следовательно, эффективнее использовать и настраивать более совершенные библиотечные версии.

Компьютерные программы чрезмерно оптимизированы. Обеспечение наиболее эффективной реализации алгоритма может не стоить затраченных усилий, если только алгоритм не должен использоваться для решения сложной задачи или многократно. В противном случае вполне достаточно качественной, сравнительно простой реализации: в худшем случае она будет работать в 5-10 раз медленнее эффективной версии, что означает увеличение времени выполнения на несколько дополнительных секунд. Правильный выбор алгоритма может ускорить работу в 100-1000 и более раз, что может вылиться во время выполнения в экономию минут и часов.

Выбор наилучшего алгоритма выполнения задачи – сложный процесс, требующий сложного математического анализа. Анализ алгоритмов – определение ресурсов для выполнения алгоритма решения задачи /1/.

Характеристиками алгоритмов являются временная и ёмкостная сложности алгоритма. Оценивание алгоритмов производится с помощью функций сложности: О-большое, омега, тэта и о-малое.

Для оценивания сложности алгоритмов применяются математические методы.

Без знания алгоритмов и структур данных невозможно создать качественный программный продукт.

Задачей методических указаний к выполнению курсовых работ является закрепление теоретических знаний и приобретение практических навыков в разработке и анализе алгоритмов решения задач.

Методическое указание включает описание и анализ алгоритмов, описание математических методов оценивания алгоритмов, темы курсовых работ, требования к разработке программного алгоритма и оформлению документации курсовых работ и пример курсовой работы.

1 Требования к курсовой работе

1.1 Назначение, цели и задачи курсовой работы

Проектируемое приложение курсовой работы предназначено для формализации и анализа алгоритмов решения задачи.

Основная учебная цель выполнения курсовой работы – закрепление теоретических знаний и приобретение практических навыков в программной реализации и анализе сложных структур данных и алгоритмов их обработки.

Основные задачи, решаемые студентом в процессе выполнения курсовой работы:

- программная реализация алгоритма решения задачи;
- анализ программного алгоритма;
- проведение и анализ экспериментального исследования алгоритма;
- документирование курсовой работы в соответствии требованиями.

1.2 Требования к функциональным характеристикам

Проектируемая система должна обеспечивать выполнение следующих основных функций:

- ввод исходных данных задачи;
- расчет параметров;
- оценка программного алгоритма по временному и объемному параметрам;
- проведение и анализ экспериментального исследования программного алгоритма;
- хранение исходных данных с возможностью их загрузки для повторной обработки;
- хранение результатов решения с возможностью их повторной визуализации;
- вывод результирующих данных;

1.3 Требования к эксплуатационным характеристикам

- Модульность.
- Расширяемость.

1.4 Требования к программному обеспечению

- Среда разработки – MS Visual C++ версии не ниже 6.0.

1.5 Требования к содержанию курсовой работы

К защите курсовой работы должен быть представлен альбом,

включающий следующие проектные, программные и эксплуатационные документы:

- опись альбома;
- пояснительная записка, включающая разделы:
    - аналитический обзор;
    - описание алгоритмов решения задачи;
    - описание структуры программного комплекса;
    - описание структур данных;
    - анализ алгоритма;
    - описание методики проведения экспериментального исследования;
    - описание и анализ результатов проведенного исследования;
    - выводы по результатам проведенного анализа;
- спецификация;
- описание программы;
- текст программы (на машинном носителе);
- руководство пользователя.

Пояснительная записка оформляется в соответствии с требованиями методического указания к оформлению документации курсовых и дипломных проектов /10/.

2 Анализ алгоритмов

Анализ алгоритма – определение объема ресурсов (памяти и времени выполнения), требуемых алгоритмом для успешной обработки данных.

Для оценки качества алгоритма вводятся понятия сложность и эффективность алгоритма. Чем большее время и объем памяти требуются для реализации алгоритма, тем больше сложность и ниже эффективность.

2.1 Характеристики алгоритма

Алгоритм обладает набором численных характеристик, сравнивая значения которых можно произвести выбор наилучшего алгоритма. Сложность алгоритма делится на временную и ёмкостную, практическую и теоретическую.

Временная (вычислительная) сложность – критерий, характеризующий временные затраты на реализацию алгоритма. Практическая временная сложность оценивается во временных единицах (милисекунды, секунды, количество временных тактов процессора, количество выполнения циклов).

Емкостная сложность – объем памяти компьютера, требуемый для реализации алгоритма. Практическая ёмкостная сложность выражается в битах, байтах, словах.

Вычислительная и емкостная сложности позволяют проводить анализ алгоритмов /3, 6/.

2.2 Функции сложности алгоритмов

Эффективность программы является важной характеристикой. Пространственная эффективность измеряется количеством памяти, требуемой для выполнения программы. Временная эффективность программы определяется временем, необходимым для выполнения.

Сравнение алгоритмов производится сопоставлением порядков сложности. Порядок сложности – это функция, доминирующая над точным выражением временной сложности. Функция $f(n)$ имеет порядок $O(g(n))$, если имеются константа k и счетчик n_0, такие, что выполняются условия $f(n) \le k * g(n)$ для любых $n > n_0$. Например, точное время обработки массива определяется по формуле

$$T = N^2 + 5 * N + 100,$$

где N – размер массива.

Приближенное время определяется функцией

$$T = 1.1 * N^2.$$

Функция сложности – это функция, которая определяет количество сравнений, перестановок, временные и объемные затраты на реализацию алгоритма.

Функция сложности O выражает относительную скорость алгоритма в зависимости от переменных. Для функции сложности сформулированы три правила:

1) $O(k * f) = O(f)$. Постоянные множители не имеют значения для определения порядка сложности, например, $O(1.5 * N) = O(N)$;

2) $O(f * g) = O(f) * O(g)$, $O(f / g) = O(f) / O(g)$. Порядок сложности произведения двух функций равен произведению их сложностей, например, $O(15 * N * N) = O(15 * N) * O(N) = O(N) * O(N) = O(N^2)$;

3) $O(f + g)$ равна доминанте $O(f)$ и $O(g)$. Порядок сложности суммы функций определяется как порядок доминанты первого и второго слагаемых (выбирается наибольший порядок), например, $O(N^5 + N^2) = O(N^5)$,

где k – константа, f и g – функции.

2.2.1 Виды функции сложности алгоритмов

Время выполнения алгоритма T зависит от объема входных данных N (рисунок 2.1)

$$T = f(N),$$

где T – время выполнения алгоритма, мс;

N – объем входных данных.

Для оценивания трудоемкости алгоритмов введена специальная система обозначений – О-нотация.

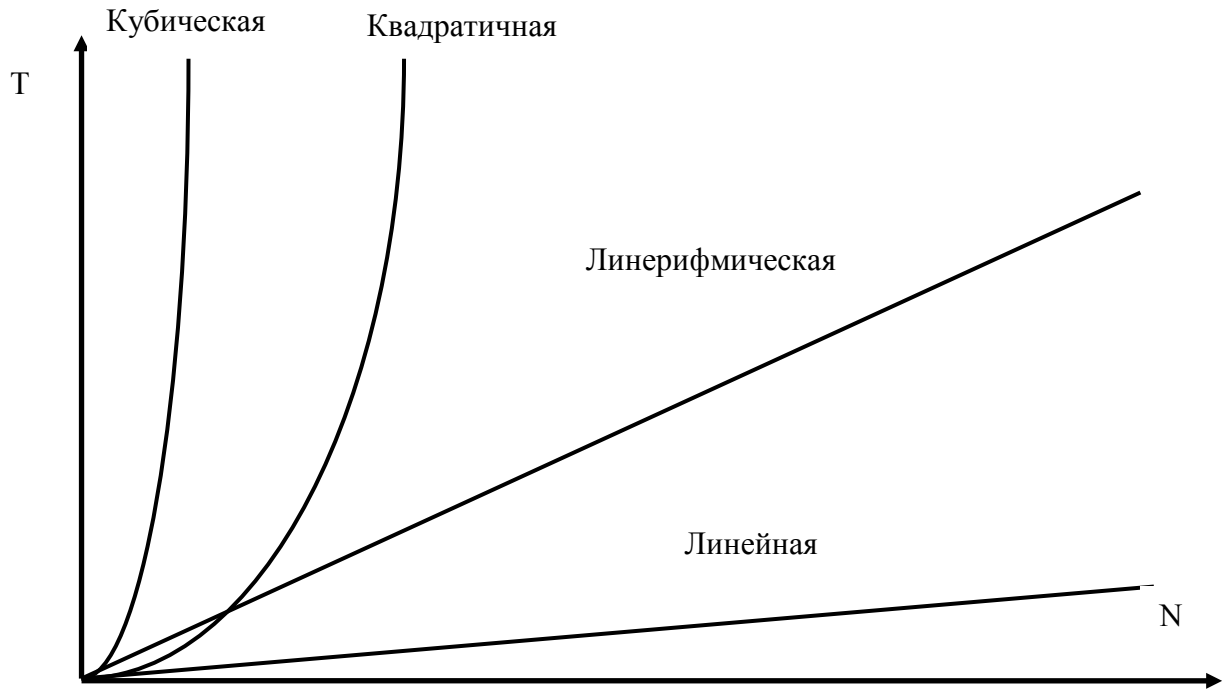


Рисунок 2.1 – Графики функций сложности алгоритмов

О-нотация позволяет учитывать в функции f (n) значимые элементы, отбрасывая второстепенные:

1) кубическая функция – функция f(n), старший член которой содержит N^3. О-нотация имеет вид $O(N^3)$;

2) квадратическая функция – функция f(n), старший член которой содержит N^2**.** О-нотация имеет вид $O(N^2)$;

3) линейная функция – функция f(n), старший член которой содержит N**.** О-нотация имеет вид $O(N)$;

4) функция линерифмическая – функция, старший член которой равен N логарифмов N. О-нотация имеет вид $O(N \log N)$.

Например, в функции $f(n) = 5 * n^2 + 3 * n + 2$ при больших n компонента n^2 превосходит остальные слагаемые. Поведение функции определяется компонентой n^2. Остальные компоненты отбрасываются. Функция $f(n) = 5 * n^2 + 3 * n + 2$ имеет оценку поведения (скорость роста значений) $O(n^2)$.

О-оценивание позволяет описывать характер поведения функции f(n) с ростом n: насколько быстро или медленно растет эта функция.

О-оценка разбивает функции сложности на группы в зависимости от скорости роста:

1) постоянные функции $O(1)$, которые с ростом n не растут;

2) функции с логарифмической скоростью роста $O(\log_2 n)$;

3) функции с линейной скоростью роста $O(n)$;

4) функции с линейно-логарифмической скоростью роста $O(n * \log_2 n)$;

5) функции с квадратичной скоростью роста $O(n^2)$;

6) функции со степенной скоростью роста $O(n^a)$ при a>2;

7) функции с показательной или экспоненциальной скоростью роста $O(2^n)$;

8) функции с факториальной степенью роста $O(n!)$ /9/.

Описание О-нотаций приведено в таблице 2.1.

Таблица 2.1 – Описание О-нотаций

О-нотация	Описание
O (1)	Инструкции программы запускаются независимо от n. Время выполнения программы постоянно (помещение в стек). Операции в программе выполняются один или несколько раз. Алгоритм независимо от размера данных требует одно и тоже время.
O (n)	Время выполнения программы линейно и зависит от n. Входной элемент обрабатывается линейное число раз.
O (n^2)	Время выполнения программы является квадратичным. Алгоритмы используются для небольших n (цикл двойного уровня вложенности, сортировки выбором, вставками)
O (n^3)	Алгоритм программы имеет кубическое время выполнения (цикл тройного уровня вложенности). Применяется для небольших задач. (умножение матриц)
O (logn)	Логарифмическая зависимость. С ростом n программа работает медленнее. Время характерно для программ, которые сводят большую задачу к набору меньших подзадач, уменьшая на каждом шаге размер подзадачи на постоянный коэффициент. Общее решение находится в одной из подзадач (бинарный поиск)
O (n*logn)	Линерифмическая зависимость. Время выполнения программы пропорционально n*logn. Возникает, когда алгоритм решает задачу, разбивая ее на меньшие подзадачи, решает независимо и затем объединяет решения подзадач (комбинация, сортировки быстрая, слиянием)

Продолжение таблицы 2.1

O (2^n)	Экспоненциальная зависимость. Прямое решение задач (перебор и сравнение различных решений). Для комбинаторных задач нереализуемы. Сведение к приближенному алгоритму с приближенным значением.

2.2.2 Сравнение асимптотического поведения функций

Математические обозначения для сравнения асимптотического поведения функций имеют вид:

1) определение О большое. Функция $T(n) = O(f(n))$, если имеются положительные константы c и n_0 такие, что $T(n) \le c * f(n)$, когда $n \ge n_0$;

2) определение Омега большое. Функция $T(n) = \Omega(f(n))$, если имеются положительные константы c и n_0 такие, что $T(n) \ge c * f(n)$, когда $n \ge n_0$;

3) определение Тета большое. Функция $T(n) = \theta(f(n))$, только и если только $T(n) = O(f(n))$ и $T(n) = \Omega(f(n))$;

4) определение о малое. Функция $T(n) = o(f(n))$, если и только если $T(n) = O(f(n))$ и $T(n) = \theta(f(n))$.

Определение О большое устанавливает, что имеется такая точка n_0, что для всех значений n за этой точкой значения $T(n)$ ограничены значениями $f(n)$, умноженными на константу. Например, время выполнения алгоритма $T(n) = O(n^2)$. В некоторой точке время выполнения ограничивается квадратичной функцией. Константы игнорируются.

Определение Омега большое утверждает, что скорость роста функции $T(n)$ больше или равна скорости функции $f(n)$. Например, алгоритм, основанный на анализе подпоследовательности в задаче определения максимальной суммы непрерывной подпоследовательности требует время $\Omega(n^2)$, потому что возможно наблюдение квадратичного числа подпоследовательностей. Это нижняя оценка, используемая в более точном анализе.

Определение Тета большое утверждает, что скорость роста функции $T(n)$ равна скорости функции $f(n)$. Например, алгоритм максимальной подпоследовательности выполняется за время $\theta(n^2)$. Время выполнения ограничено квадратичной функцией. Граница не может быть улучшена, потому что она ограничена снизу другой квадратичной функцией. Тета большое предоставляет верхнюю границу алгоритма и гарантирует точность анализа.

Определение о малое утверждает, что скорость роста функции $T(n)$ строго меньше, чем скорость роста функции $f(n)$. Функция о малое отлична

от О большое, так как О большое допускает возможность одинаковых скоростей роста. Например, если время выполнения алгоритма составляет $o(n^2)$, оно гарантированно растет медленнее, чем квадратичная функция. Имеем подквадратичный алгоритм. Граница $o(n^2)$ ниже, чем $\theta(n^2)$ /9/.

Сравнение асимптотического поведения функций приведено в таблице 2.2.

Таблица 2.2 – Относительная скорость роста

Математическое выражение	Относительные скорости роста
$T(n) = O(f(n))$	Скорость роста $T(n) \leq$ скорости роста $f(n)$
$T(n) = \Omega(f(n))$	Скорость роста $T(n) \geq$ скорости роста $f(n)$
$T(n) = \theta(f(n))$	Скорость роста $T(n) =$ скорости роста $f(n)$
$T(n) = o(f(n))$	Скорость роста $T(n) <$ скорости роста $f(n)$

2.2.3 Базовое правило использования О большого

Базовое правило формулируется следующим образом: время выполнения цикла не превышает времени выполнения операторов, входящих в цикл (включая операторы условия), умноженное на число итераций цикла.

Время выполнения операторов внутри группы вложенных циклов равно времени выполнения операторов, умноженному на число итераций во всех циклах. Время выполнения последовательности циклов, следующих друг за другом, равно времени выполнения доминантного цикла. Первый случай описывается квадратичной функцией. Второй случай линеен.

2.3 Анализ функции сложности

Определение сложности алгоритма сводится к анализу циклов и рекурсивных вызовов. Алгоритмы без циклов и рекурсивных вызовов имеют константную сложность.

Программный алгоритм создания и вывода целочисленного одномерного массива приведен в листинге 2.1.

Листинг 2.1 Одномерный целочисленный массив. Функция сложности алгоритма O(N).

```
#include "stdafx.h"
#include <iostream>
#include <ctime>
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{       setlocale(LC_ALL, "russian");
```

```
int N;
cout<<"Введите размер массива "; cin>>N; cout<<endl;
int *masiv=new int [N];          //Создание динамического массива
srand(time(NULL));                    //Автоматизация рандомизации
for(int i=0; i<N; i++)
        masiv[i]=2+rand()%15;//ГСЧ
                for(int i=0; i<N; i++)
                        cout<<"Элемент                    masiv["<<i<<"]--
>"<<masiv[i]<<endl;
system("pause");
return 0;
}
```

Результат работы программы, приведенной в Листинге 2.1, изображен на рисунке 2.2.

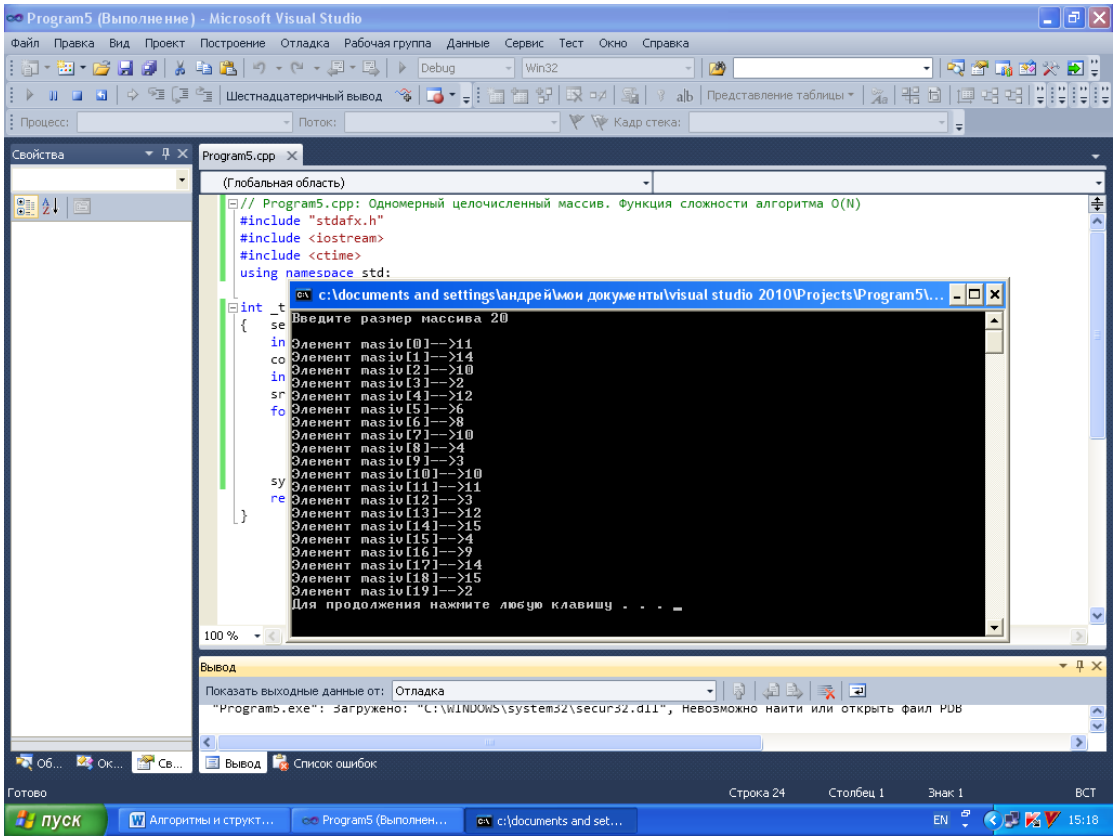


Рисунок 2.2 – Вывод одномерного целочисленного массива

Сложность алгоритма $O(2*N)=O(N)$ (правило 1). Программный алгоритм содержит два последовательно N раз выполняющихся циклов. Сложность тела циклов равна O(1).

Программный алгоритм создания и вывода целочисленного двумерного массива приведен в листинге 2.2.

Листинг 2.2 Двумерный целочисленный массив. Сложность алгоритма $O(N^2)$

```
#include "stdafx.h"
#include <iostream>
#include <ctime>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{       setlocale(LC_ALL, "russian");
int N,M;
cout<<"Введите количество строк матрицы "; cin>>N; cout<<endl;
cout<<"Введите количество столбцов матрицы "; cin>>M; cout<<endl;
int **mas=new int *[N];
for(int i=0; i<N; i++)
        mas[i]=new int [M];
srand(time(NULL));
for(int i=0; i<N; i++)
        for(int j=0; j<M; j++)
                mas[i][j]=1+rand()%25;
for(int i=0; i<N; i++) {
        for(int j=0; j<M; j++)
                cout<<mas[i][j]<<'\t';
                        cout<<endl;
}
system("pause");
return 0;
}
```

Результат работы программы, приведенной в Листинге 2.2, изображен на рисунке 2.3.

Сложность алгоритма $O(N + 2 * N * M) = O(N^2)$ (правило 2, 3). Программный алгоритм содержит три последовательно выполняющихся циклов. Первый цикл выполняется N раз. Второй и третий циклы – вложенные циклы. Они выполняются N*M раз. Сложность тела циклов равна O(1).

Существуют два способа анализа сложности алгоритма: восходящий (от внутренних управляющих структур к внешним) и нисходящий (от внешних структур к внутренним). Вычисления составляют около 10% времени работы программы, 90% времени работы программы составляют повторения. Повторения организованы в виде вложенных циклов или вложенной рекурсии. Для повышения эффективности программы сокращают глубину вложенности повторений, количество операторов в циклах с наибольшей глубиной вложенности /6/.

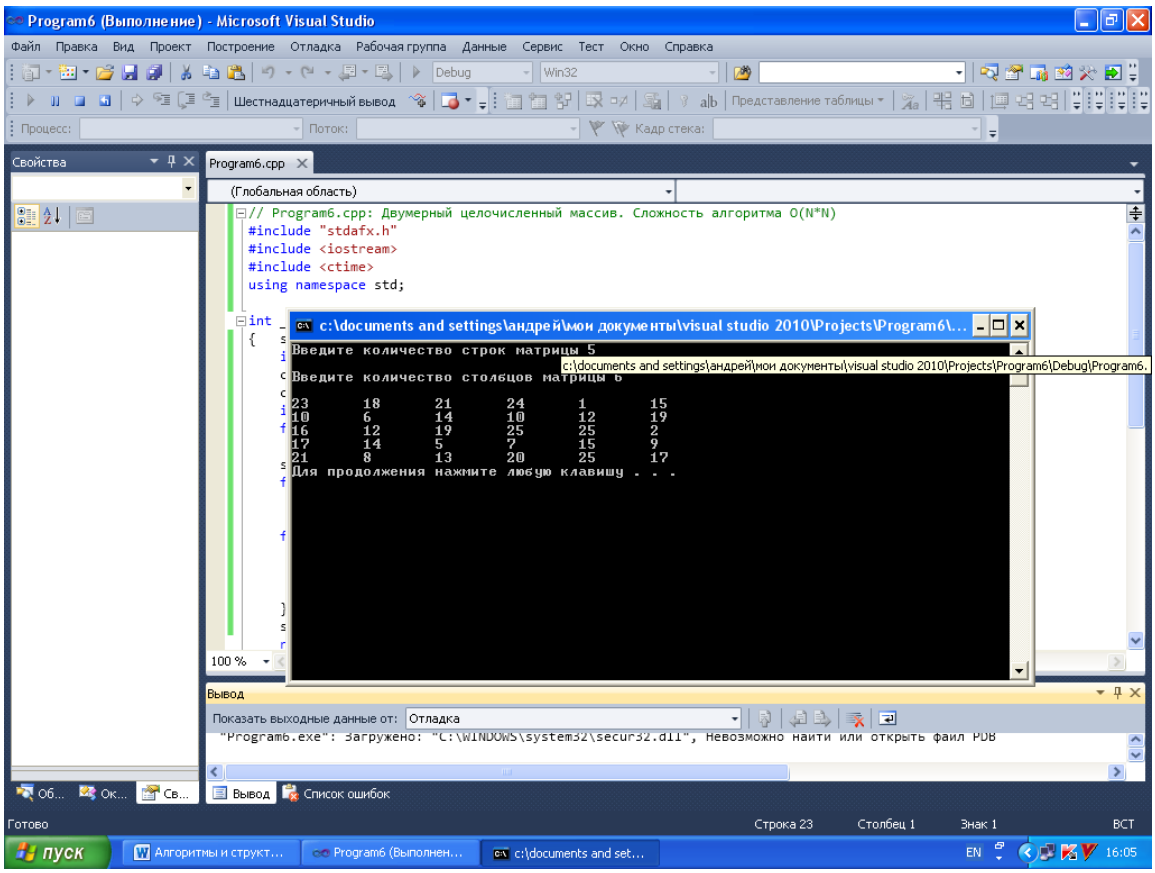


Рисунок 2.3 – Двумерный целочисленный массив

2.3.1 Проверка анализа алгоритма

Проверка анализа алгоритма заключается в проверке правильности алгоритма и оценке скорости алгоритма. Для этого определяются:

1) насколько эмпирически наблюдаемое время выполнения соответствует величине, предсказанной в результате анализа. Если n увеличивается в 10 раз, время выполнения возрастает в 10 раз для линейных программ, в 100 раз для квадратичных и в 1000 раз для кубических программ. Для линерифмических программ время возрастает чуть больше, чем в 10 раз;

2) значение $T(n)/F(n)$ в определенном диапазоне значений n (разделены между собой в отношении кратном 2). Время $T(n)$ – эмпирически определяемое время выполнения. Если $f(n)$ является точной оценкой времени выполнения, то вычисленные значения будут сходиться к положительной константе. Если $f(n)$ дает завышенную оценку, значения стремятся к нулю. Если $f(n)$ занижена и, следовательно, неверна, значения будут увеличиваться.

Например, программа, выполняющая n случайных поисков с использованием алгоритма двоичного поиска. Поиск дает логарифмическое время. Полное время выполнения программы соответствует критерию $O(n\log n)$. Фактические значения времени выполнения программы на ПЭВМ при различных объемах входных данных приведены в таблице 2.3.

Таблица 2.3 – Фактические значения времени выполнения программы на ПЭВМ

N	Время процессора, мс	T/n	T/n^2	$T/(n\log n)$
10000	100	0,01000000	0,00000100	0,00075257
20000	200	0,01000000	0,00000050	0,00069990
40000	440	0,01100000	0,00000027	0,00071953
80000	930	0,01162500	0,00000015	0,00071373
160000	1960	0,01225000	0,00000008	0,00070860
320000	4170	0,01303125	0,00000004	0,00071257
640000	8770	0,01370313	0,00000002	0,00071046

Значения $T/(n\log n)$ в таблице 2.3 сходятся, поэтому программа соответствует критерию $O(n\log n)$. Значения T/n в таблице 2.3 возрастают, следовательно, оценка $O(n)$ дает заниженный результат. Значения T/n^2 в таблице 2.3 стремятся к нулю, следовательно, оценка $O(n^2)$ дает завышенный результат /9/.

2.3.2 Ограничения анализа посредством О большого

Анализ по критерию О большого является эффективным методом.

Не рекомендуется применять в случаях, когда:

1) малы объемы входных данных;

2) недостаточно памяти.

При небольших объемах данных применяется линейный алгоритм.

2.3.3 Анализ алгоритма программы «Тройки Пифагора»

Пифагорова тройка – набор из трех натуральных чисел (x, y, z), удовлетворяющих однородному уравнению $x^2 + y^2 = z^2$.

Пифагорова тройка называется примитивной, если числа (x, y, z) являются взаимно-простыми.

Пифагоровы тройки (примитивные выделены): **(3,4,5)**, (6,8,10), **(5,12,13)**, (9,12,15), **(8,15,17)**, (12,16,20), (15,20,25), **(7,24,25)**, (10,24,26), **(20,21,29)**, (18,24,30), (16,30,34), (21,28,35), **(12,35,37)**, (15,36,39), (24,32,40), **(9,40,41)**, (27,36,45), (14,48,50), (30,40,50), …/6/.

Задача. Разработать программу «Тройки Пифагора». Оценить сложность программы.

Программный алгоритм решения задачи приведен в листинге 2.3.

Листинг 2.3 Тройки Пифагора

// Pifagor.cpp

```
#include "stdafx.h"
#include <iostream>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{   setlocale(LC_ALL, "russian");
    int i,x,y,z,n;
A   cout<<"Введите ограничение "; cin>>n; cout<<endl;
    x=1;
    while(x<n)
B   {   y=x;
        while(y<=n)
        {   z=y;
G       C   while(z<=n)
        {   if((z<=x*2 && y<=x*2) && (z*z==x*x+y*y))
                {   cout<<"Число x-->"<<x<<endl;
            F   cout<<"Число y-->"<<y<<endl;
        I   H   cout<<"Число z-->"<<z<<endl;
D               }
            J   z+=1;
            }
E       K   y+=1;
        }
L       x+=1;
    }
    system("pause");
M   return 0;
}
```

На рисунке 2.4 приведен результат работы программы «Тройки Пифагора».

Оценим сложность алгоритма программы «Тройки Пифагора».

$$O(H) = O(1) + O(1) + O(1) = O(1);$$

$$O(I) = O(N) * (O(F) + O(J)) = O(N) * O(\text{доминанты условия}) = O(N);$$

$$O(G) = O(N) * (O(C) + O(I) + O(K)) = O(N) * (O(1) + O(N) + O(1)) = O(N^2);$$

$$O(E) = O(N) * (O(B) + O(G) + O(L)) = O(N) * O(N^2) = O(N^3);$$

$$O(D) = O(A) + O(E) + O(M) = O(1) + O(N^3) + O(1) = O(N^3)$$

Сложность алгоритма программы «Тройки Пифагора» $O(N^3)$ /6/.

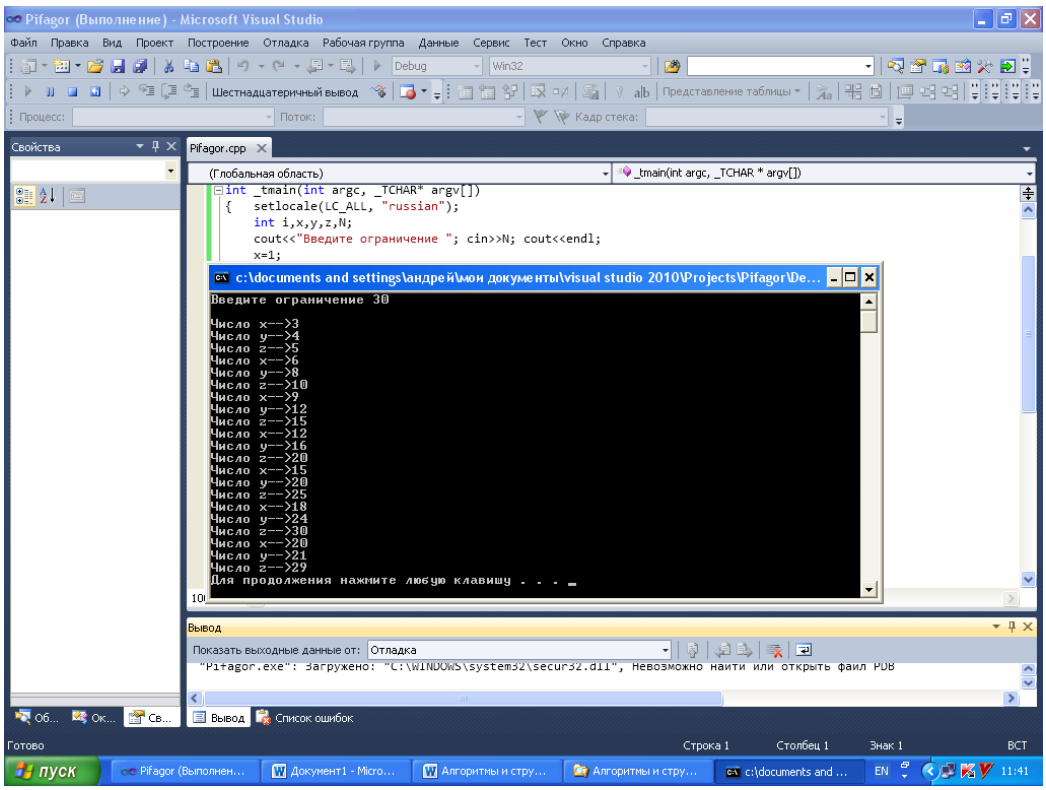


Рисунок 2.4 – Тройки Пифагора

2.4 Анализ алгоритма максимальной суммы непрерывной подпоследовательности

2.4.1 Постановка задачи

Даны целые числа $A_1, A_2, ..., A_n$ (возможно отрицательные). Определить максимальное значение $\sum_{k=i}^{j} A_k$. Максимальная сумма равна нулю, если все целые числа отрицательны /9/.

2.4.2 Кубический алгоритм. Функция $T(n) = O(n^3)$

Кубический алгоритм – алгоритм грубой силы (листинг 2.4)

Листинг 2.4 Алгоритм грубой силы.

```
//Кубический алгоритм максимальной суммы непрерывной
//подпоследовательности. SeqStart и SeqEnd представляют наилучшую
//последовательность.
template <class T>
T maxSubsequenceSum(const vector<T> & a,
                    int & seqStart, int & seqEnd)
{       int n=a.size();
```

```
        T maxSum=0;
        for(int j=i; j<n; j++) { T thisSum=0;
                for(int k=i; k<=j; k++)
                        thisSum+=a[k];
                        if(thisSum>maxSum) { maxSum=thisSum;
                                                seqStart=i;
                                                seqEnd=j; }}
        return maxSum;
}
```

Время выполнения алгоритма подчиняется функции $T(n) = O(n^3)$.

Достоинство кубического алгоритма – простота.

Недостаток кубического алгоритма – малая эффективность /9/.

2.4.3 Квадратичный алгоритм. Функция $T(n) = O(n^2)$

Низкая эффективность кубического алгоритма объясняется затратными вычислениями во внутреннем цикле for. Улучшенный алгоритм использует выражение $\sum_{k=i}^{j} A_k = A_j + \sum_{k=i}^{j-1} A_k$ (листинг 2.5).

Листинг 2.5 Квадратичный алгоритм.

```
//Квадратичный алгоритм максимальной суммы непрерывной
//подпоследовательности. SeqStart и SeqEnd представляют наилучшую
//последовательность.
template <class T>
T maxSubsequenceSum(const vector<T> & a,
int & seqStart, int & seqEnd)
{       int n=a.size();
T maxSum=0;
for(int i=0; i<n; i++) { T thisSum=0;
for(int j=i; j<=n; j++) {
thisSum+=a[j];
if(thisSum>maxSum) { maxSum=thisSum;
seqStart=i;
seqEnd=j; }}}
return maxSum;
}
```

В улучшенном алгоритме используются не три, а два вложенных цикла.

Время выполнения квадратичного алгоритма подчиняется функции $T(n) = O(n^2)$ /9/.

2.4.4 Линейный алгоритм. Функция $T(n) = O(n)$

В линейном алгоритме используется один цикл (листинг 2.6).

Листинг 2.6 Линейный алгоритм.

```
//Линейный алгоритм максимальной суммы непрерывной
//подпоследовательности. SeqStart и SeqEnd представляют наилучшую
//последовательность.
template <class T>
T maxSubsequenceSum(const vector<T> & a,
                    int & seqStart, int & seqEnd)
{       int n=a.size();
        T thisSum=0;
        T maxSum=0;
        for(int i=0, j=0; j<n; j++) { thisSum+=a[j];
                        if(thisSum>maxSum)
                        { maxSum=thisSum; seqStart=i; seqEnd=j; }
                else if (thisSum<0) { i=j+1;
                thisSum=0; }}
                system("pause");
                return maxSum;
                }
```

Время выполнения линейного алгоритма подчиняется функции $T(n) = O(n)$ /9/.

2.4.5 Сравнение алгоритмов

Сравнение алгоритмов определения максимальной суммы непрерывной подпоследовательности приведено в таблице 2.6.

Таблица 2.6 – Сравнение алгоритмов

N	Кубический алгоритм, $T(n) = O(n^3)$	Квадратичный алгоритм, $T(n) = O(n^2)$	Линейный алгоритм, $T(n) = O(n)$
10	0,000009	0,000004	0,000003
100	0,002580	0,000109	0,000006
1000	2,281013	0,010203	0,000031
10000	Слишком велико	1,2329	0,000317
100000	Слишком велико	135	0,003206

Время кубического алгоритма возрастает в 1000 раз, если количество входных данных увеличивается в 10 раз.

Время квадратичного алгоритма возрастает в 100 раз, если количество входных данных увеличивается в 10 раз.

Время линейного алгоритма возрастает в 10 раз, если количество входных данных увеличивается в 10 раз /9/.

3 Математические методы оценивания времени выполнения алгоритмов

Цель эксперимента – определение уравнения связи между временем выполнения программного алгоритма и размером задачи. В зависимости от постановки задачи разрабатывается регрессионная модель парной корреляции или регрессионная модель множественной корреляции. Например, в случае внешней сортировки данных для установления связи между временем выполнения программного алгоритма и количеством элементов исходного массива разрабатывается классическая регрессионная модель парной корреляции. В случае определения критического пути графа разрабатывается классическая регрессионная модель множественной корреляции, устанавливающая связь между временем работы программного алгоритма и входными переменными: количеством вершин графа и количеством дуг.

Пусть оценки состоятельные, несмещенные и эффективные.

3.1 Классическая регрессионная модель парной корреляции

Классическая регрессионная модель парной корреляции (КРМПК) устанавливает связь между эндогенной переменной и одной экзогенной переменной. Уравнение связи имеет вид

$$\hat{y} = a_0 + a_1 * x, \qquad (3.1)$$

где $\hat{y}$ – эндогенная переменная;

x – экзогенная переменная;

a_0 – свободный коэффициент;

a_1 – выборочный коэффициент.

Корреляция – теснота (сила) связи. Регрессия – форма (уравнение) связи.

КРМПК используется в задачах сортировки массивов. Эндогенная переменная – время сортировки, экзогенная переменная размер массива (количество элементов).

Экспериментальное исследование программного алгоритма решения задачи включает этапы:

Этап 1. *Получение выборки статистических данных в ходе проведения эксперимента на ЭВМ.*

Количество наблюдений m=10 … 15. Результаты проведения эксперимента представляются в таблице 3.1.

Таблица 3.1 – Исходные статистические данные

Номер наблюдения i	Время выполнения программы, мс, y_i	Размер задачи, x_i	x_i^2	$x_i * y_i$
1	y_1	x_1	x_1^2	$x_1 * y_1$
2	y_2	x_2	x_2^2	$x_2 * y_2$
3	y_3	x_3	x_3^2	$x_3 * y_3$
4	y_4	x_4	x_4^2	$x_4 * y_4$
…	…	…	…	…
M	y_m	x_m	x_m^2	$x_m * y_m$
Сумма	$\sum_{i=1}^{m} y_i$	$\sum_{i=1}^{m} x_i$	$\sum_{i=1}^{m} x_i^2$	$\sum_{i=1}^{m} x_i * y_i$

Этап 2. *Определение уравнения связи в общем виде.*

По результатам проведения эксперимента на ЭВМ строится и анализируется точечный график зависимости $y_i = f(x_i)$, $i = \overline{1,m}$.

На рисунке 3.1 представлен точечный график $y = f(x)$ зависимости времени y выполнения сортировки от количества x элементов массива. Анализ точечного графика позволяет сделать вывод о линейной тенденции. Уравнение связи в общем виде имеет вид (3.1).

Этап 3. *Оценивание коэффициентов уравнения связи.*

Оценивание коэффициентов уравнения связи – приближенный расчет значений коэффициентов. Оценивание коэффициентов a_0 и a_1 производится методом наименьших квадратов (МНК).

Определение. Сумма квадратов разности между наблюденным и расчетным значениями эндогенной переменной минимальна.

Математическое описание МНК имеет вид.

$$S_{кв} = \sum_{i=1}^{m} (y_i - \hat{y}_i)^2 -> \min, \qquad (3.2)$$

где y_i – наблюденное значение эндогенной переменной;

$\hat{y}_i$ – расчетное значение эндогенной переменной;

i – номер наблюдения, $i = \overline{1,m}$,

m – объем выборки.

Оценивание коэффициентов a_0 и a_1 производится по шагам.

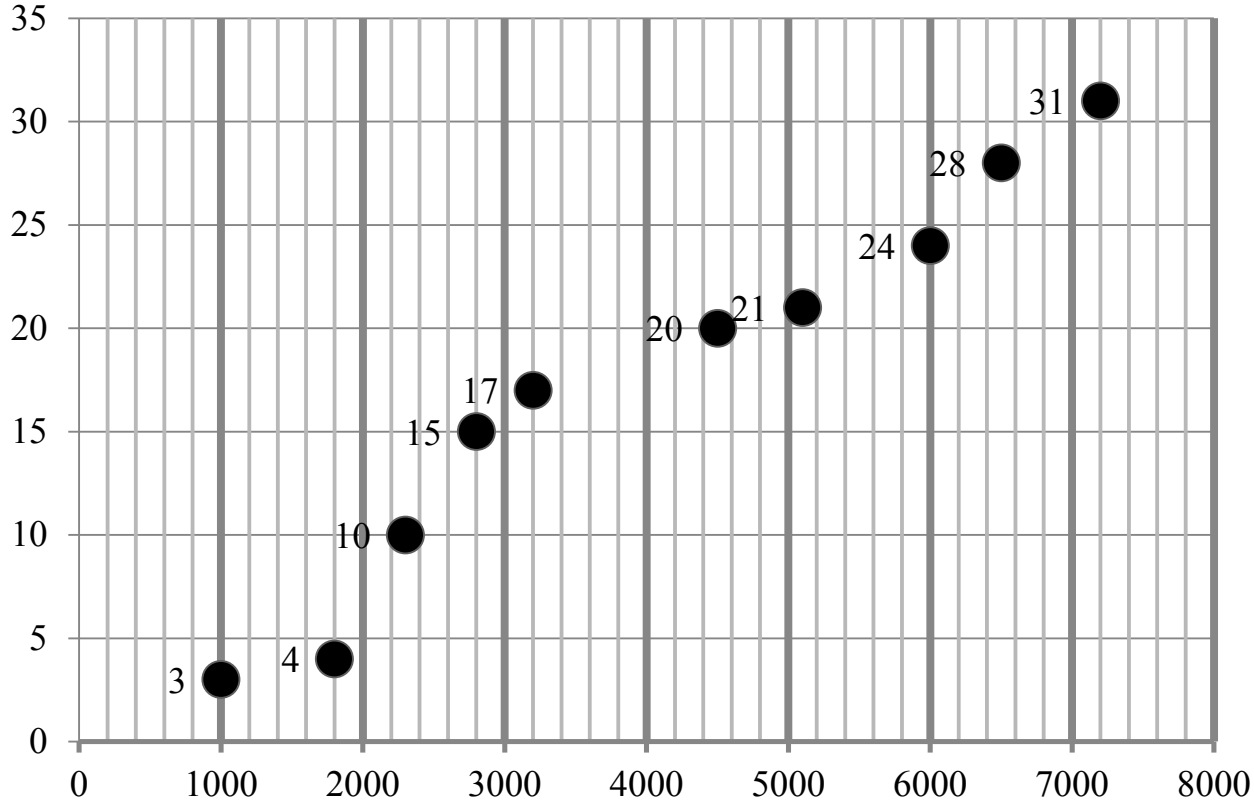


Рисунок 3.1 – Точечный график зависимости времени сортировки от количества элементов массива

Шаг 1. Подставим в математическое описание (3.2) МНК вместо $\hat{y}_i$ правую часть уравнения связи в общем виде (3.1). Математическое описание МНК будет иметь вид

$$S_{кв} = \sum_{i=1}^{m} (y_i - a_0 - a_1 * x)^2 -> \min . \qquad (3.3)$$

Шаг 2. Определим математический инструментарий (система нормальных уравнений, СНУ) для оценивания коэффициентов a_0 и a_1 уравнения связи.

Определение. Производной функции $y = f(x)$ в точке x_0 называется предел отношения приращения функции $\Delta y = f(x_0 + \Delta x) - f(x_0)$ к приращению аргумента Δx при произвольном стремлении Δx к нулю, если такой предел существует.

$$f'(x_0) = \lim_{\Delta x -> 0} \frac{f(x + \Delta x) - f(x_0)}{\Delta x}. \qquad (3.4)$$

Определение. Функция $y = f(x)$ называется непрерывной, если она не имеет точек разрыва.

Достаточный признак существования экстремума. Если непрерывная функция $y = f(x)$ имеет производную $f'(x)$ во всех точках интервала, содержащего стационарную точку $x = c$, и если производная $f'(x)$ при переходе аргумента слева направо через стационарную точку $x = c$ меняет знак

с плюса на минус, то стационарная точка $x = c$ есть точка максимума, при перемене знака с минуса на плюс – точка минимума.

Точки минимума и максимума называются точками экстремума.

Необходимый признак существования экстремума. Если функция $f(x)$ имеет в точке $x = c$ экстремум и дифференцируема в этой точке, то $f'(c) = 0$.

Произведем вывод СНУ для оценивания коэффициентов a_0 и a_1 уравнения связи

$$\begin{cases} \dfrac{\partial S_{кв}}{\partial a_0} = 0 \\ \dfrac{\partial S_{кв}}{\partial a_1} = 0 \end{cases} \tag{3.5}$$

$$\begin{cases} \dfrac{\partial \sum_{i=1}^{m} (y_i - a_0 - a_1 * x)^2}{\partial a_0} = 0 \\ \dfrac{\partial \sum_{i=1}^{m} (y_i - a_0 - a_1 * x)^2}{\partial a_1} = 0 \end{cases} \tag{3.6}$$

Производная степенной функции рассчитывается по формуле (3.7)

$$(X^n)' = n * X^{(n-1)} * (X)'.$$

$$\begin{cases} -2 * \sum_{i=1}^{m} (y_i - a_0 - a_1 * x) = 0 \\ -2 * \sum_{i=1}^{m} (y_i - a_0 - a_1) * x = 0 \end{cases} \tag{3.8}$$

Выполним преобразования: сократим на (-2) левую и правую части, раскроем скобки, сгруппируем слагаемые, заменим аддитивную операцию $\underbrace{a_0 + a_0 + ... a_0}_{m}$ на мультипликативную операцию $m * a_0$. СНУ имеет вид

$$\begin{cases} m * a_0 + a_1 * \sum_{i=1}^{m} x_i = \sum_{i=1}^{m} y_i \\ a_0 * \sum_{i=1}^{m} x_i + a_1 * \sum_{i=1}^{m} x_i^2 = \sum_{i=1}^{m} y_i * x_i \end{cases} \tag{3.9}$$

СНУ решается методом Гаусса.

Этап 4. *Анализ уравнения связи и расчет доверительного интервала.*

Анализ уравнения связи включает расчет коэффициентов парной корреляции, детерминации, эластичности и бэта-коэффициента.

Коэффициент парной корреляции показывает тесноту (силу) связи между выходной переменной y и входной переменной x. Коэффициент парной корреляции рассчитывается по формуле

$$r_{\hat{yx}} = \frac{m * \sum_{i=1}^{m} x_i * y_i - \sum_{i=1}^{m} x_i * \sum_{i=1}^{m} y_i}{\sqrt{(m * \sum_{i=1}^{m} x_i^2 - \sum_{i=1}^{m} x_i * \sum_{i=1}^{m} x_i) * (m * \sum_{i=1}^{m} y_i^2 - \sum_{i=1}^{m} y_i * \sum_{i=1}^{m} y_i)}}. \quad (3.10)$$

Коэффициент корреляции $r_{\hat{yx}} \in [-1,+1]$. Если $r_{\hat{yx}} = 0$, то связь отсутствует, $0 < |r_{\hat{yx}}| \le 0.5$ – связь слабая, $0.5 < |r_{\hat{yx}}| \le 0.8$ – связь умеренная, $0.8 < |r_{\hat{yx}}| \le 1$ – связь сильная.

Коэффициент детерминации $r_{\hat{yx}}^2$ показывает значимость входной переменной: долю существенного влияния на выходную переменную y выбранной входной переменной x. Коэффициент детерминации $r_{\hat{yx}}^2 \in [0,+1]$.

Коэффициент эластичности показывает на сколько процентов в среднем изменится эндогенная переменная y от своего среднего значения при изменении экзогенной переменной x на 1 процент своей средней величины. Коэффициент эластичности рассчитывается по формуле

$$Э_{\hat{yx}} = \frac{a_1 * \bar{x}}{\bar{y}}, \quad (3.11)$$

где a_1 – выборочный коэффициент;

$\bar{x}$ – среднее арифметическое значение объема выборки X;

$\bar{y}$ – среднее арифметическое значение объема выборки Y.

Среднее арифметическое значение объема выборки X рассчитывается по формуле

$$\bar{x} = \frac{\sum_{i=1}^{m} x_i}{m}. \quad (3.12)$$

Среднее арифметическое значение объема выборки Y рассчитывается по формуле

$$\bar{y} = \frac{\sum_{i=1}^{m} y_i}{m}. \quad (3.13)$$

Бета-коэффициент показывает на какую часть величины своего среднего квадратического отклонения изменится в среднем значение эндогенной переменной y при изменении экзогенной переменной x на величину среднеквадратического отклонения. Бета-коэффициент рассчитывается по формуле

$$\hat{\beta}_{yx} = \frac{a_1 * S_x}{S_y}, \tag{3.14}$$

где a_1 – выборочный коэффициент;

S_x – средняя квадратическая ошибка (отклонение) объема выборки X;

S_y – средняя квадратическая ошибка (отклонение) объема выборки Y.

Средняя квадратическая ошибка объема выборки X рассчитывается по формуле

$$S_x = \sqrt{\frac{\sum_{i=1}^{m}(x_i - \bar{x})^2}{m}}, \tag{3.15}$$

где x_i – экзогенная переменная;

$\bar{x}$ – среднее арифметическое значение объема выборки X;

m – объем выборки.

Средняя квадратическая ошибка выборки Y рассчитывается по формуле

$$S_y = \sqrt{\frac{\sum_{i=1}^{m}(y_i - \bar{y})^2}{m}}, \tag{3.16}$$

где y_i – экзогенная переменная;

$\bar{y}$ – среднее арифметическое значение объема выборки Y.

Доверительный интервал функции регрессии рассчитывается по формуле

$$\Delta = t_{f,\alpha} * \frac{S_y}{\sqrt{m}}, \tag{3.17}$$

где $t_{f,\alpha}$ – коэффициент Стьюдента при уровне значимости α и числе степеней свободы f;

S_y – средняя квадратическая ошибка (отклонение) объема выборки Y;

m - объем выборки.

Число степеней свободы рассчитывается по формуле

$$f = m - g, \tag{3.18}$$

где m – объем выборки;

g – число коэффициентов регрессии.

Число коэффициентов регрессии рассчитывается по формуле

$$g = k + 1, \tag{3.19}$$

где k – число экзогенных переменных.

Уравнение регрессии примет вид

$$y = a_0 + a_1 * x \pm \Delta. \tag{3.20}$$

3.2 Классическая регрессионная модель множественной корреляции

Классическая регрессионная модель множественной корреляции (КРММК) устанавливает связь между эндогенной переменной и двумя и более экзогенными переменными. Уравнение связи имеет вид

$$\hat{y} = a_0 + a_1 * x + a_2 * x_2 + ... + a_n * x_n, \qquad (3.21)$$

где $\hat{y}$ – эндогенная переменная;

x_j – экзогенная переменная, $j = \overline{1,n}$;

a_0 – свободный коэффициент;

a_j – выборочные коэффициенты;

n – количество экзогенных переменных;

m – объем выборки, $i = \overline{1,m}$.

КРММК используется в задачах определения минимального остовного дерева графа, критического пути. Эндогенная переменная y – время выполнения программного алгоритма решения задачи, экзогенные переменные x_1 – количество вершин графа, x_2 – количество дуг.

Экспериментальное исследование программного алгоритма решения задачи включает этапы:

Этап 1. *Получение выборки статистических данных в ходе проведения эксперимента на ЭВМ.*

Количество наблюдений m=10 … 15. Результаты проведения эксперимента представляются в таблице 3.2.

Таблица 3.2 – Исходные статистические данные

Номер наблюдения i	Время выполнения программы, мс, y_i	Количество вершин, x_{1i}	Количество дуг, x_{2i}	x_{1i}^2	x_{2i}^2	$x_{1i} * x_{2i}$	$x_{1i} * y_i$	$x_{2i} * y_i$
1	y_1	x_{11}	x_{21}	x_{1i}^2	x_{2i}^2	$x_{11} * x_{21}$	$x_{11} * y_1$	$x_{21} * y_1$
2	y_2	x_{12}	x_{22}	x_{1i}^2	x_{2i}^2	$x_{12} * x_{22}$	$x_{12} * y_2$	$x_{22} * y_2$
3	y_3	x_{13}	x_{23}	x_{1i}^2	x_{2i}^2	$x_{13} * x_{23}$	$x_{13} * y_3$	$x_{23} * y_3$
4	y_4	x_{14}	x_{24}	x_{1i}^2	x_{2i}^2	$x_{14} * x_{24}$	$x_{14} * y_4$	$x_{24} * y_4$
…	…	…	…	…	…	…	…	…
M	y_m	x_{1m}	x_{2m}	x_{1i}^2	x_{2i}^2	$x_{1m} * x_{2m}$	$x_{1m} * y_m$	$x_{2m} * y_m$

Продолжение таблицы 3.2

Сумма	$\sum_{i=1}^{m} y_m$	$\sum_{i=1}^{m} x_{1i}$	$\sum_{i=1}^{m} x_{2i}$	$\sum_{i=1}^{m} x_{1i}^2$	$\sum_{i=1}^{m} x_{2i}^2$	$\sum_{i=1}^{m} x_{1i} * x_2$	$\sum_{i=1}^{m} x_{1i} * y_i$	$\sum_{i=1}^{m} x_{2i} * y_i$

Этап 2. *Определение уравнения связи в общем виде.*

По результатам проведения эксперимента на ЭВМ строятся и анализируются точечные графики зависимости y от x_j, $j=\overline{1,n}$: $y=f(x_1)$ и $y=f(x_2)$.

Анализ точечных графиков позволяет сделать вывод о линейной тенденции. Уравнение связи в общем виде имеет вид

$$\hat{y}=a_0+a_1*x_1+a_2*x_2, \tag{3.22}$$

где $\hat{y}$ – время выполнения программного алгоритма;

x_1 – количество вершин графа;

x_2 – количество дуг графа;

a_0 – свободный коэффициент;

a_1, a_2 – выборочные коэффициенты.

Этап 3. *Оценивание коэффициентов уравнения связи.*

Оценивание коэффициентов a_0, a_1 и a_2 производится МНК (3.2) по шагам.

Шаг 1. Подставим в математическое описание (3.2) МНК вместо $\hat{y}_i$ правую часть уравнения связи в общем виде (3.22). Математическое описание МНК будет иметь вид

$$S_{кв}=\sum_{i=1}^{m}(y_i-a_0-a_1*x_1-a_2*x_2)^2->\min. \tag{3.23}$$

Шаг 2. Определим СНУ для оценивания коэффициентов a_0, a_1 и a_2 уравнения связи.

$$\begin{cases}\dfrac{\partial S_{кв}}{\partial a_0}=0\\ \dfrac{\partial S_{кв}}{\partial a_1}=0\\ \dfrac{\partial S_{кв}}{\partial a_2}=0\end{cases} \tag{3.24}$$

$$\begin{cases} \dfrac{\partial \sum_{i=1}^{m} (y_i - a_0 - a_1 * x_{1i} - a_2 * x_{2i})^2}{\partial a_0} = 0 \\ \dfrac{\partial \sum_{i=1}^{m} (y_i - a_0 - a_1 * x_{1i} - a_2 * x_{2i})^2}{\partial a_1} = 0 \\ \dfrac{\partial \sum_{i=1}^{m} (y_i - a_0 - a_1 * x_{1i} - a_2 * x_{2i})^2}{\partial a_2} = 0 \end{cases} \quad (3.25)$$

$$\begin{cases} -2 * \sum_{i=1}^{m} (y_i - a_0 - a_1 * x_{1i} - a_2 * x_{2i}) = 0 \\ -2 * \sum_{i=1}^{m} (y_i - a_0 - a_1 * x_{1i} - a_2 * x_{2i}) * x_{1i} = 0 \\ -2 * \sum_{i=1}^{m} (y_i - a_0 - a_1 * x_{1i} - a_2 * x_{2i}) * x_{2i} = 0 \end{cases} \quad (3.26)$$

Выполним преобразования: сократим на (-2) левую и правую части, раскроем скобки, сгруппируем слагаемые, заменим аддитивную операцию $\underbrace{a_0 + a_0 + ... a_0}_{m}$ на мультипликативную операцию $m * a_0$. СНУ имеет вид

$$\begin{cases} m * a_0 + a_1 * \sum_{i=1}^{m} x_{1i} + a_2 * \sum_{i=1}^{m} x_{2i} = \sum_{i=1}^{m} y_i \\ a_0 * \sum_{i=1}^{m} x_{1i} + a_1 * \sum_{i=1}^{m} x_{1i}^2 + a_2 * \sum_{i=1}^{m} x_{2i} * x_{1i} = \sum_{i=1}^{m} y_i * x_{1i} \\ a_0 * \sum_{i=1}^{m} x_{2i} + a_1 * \sum_{i=1}^{m} x_{1i} * x_{2i} + a_2 * \sum_{i=1}^{m} x_{2i}^2 = \sum_{i=1}^{m} y_i * x_{2i} \end{cases} \quad (3.27)$$

СНУ решается методом Гаусса.

Этап 4. *Анализ уравнения связи и расчет доверительного интервала.*

Анализ уравнения связи включает расчет коэффициента множественной корреляции, совокупного коэффициента детерминации, частных коэффициентов корреляции, частных коэффициентов детерминации, частных коэффициентов эластичности и частных бэта-коэффициентов.

Коэффициент множественной корреляции рассчитывается по формуле

$$R_{y x_1 x_2}^{\wedge} = \sqrt{\frac{r_{yx_1}^2 + r_{yx_2}^2 - 2 * r_{yx_1} * r_{yx_2} * r_{x_1 x_2}}{1 - r_{x_1 x_2}^2}}, \quad (3.28)$$

где r_{yx_1} – парный коэффициент корреляции между y и x_1;

r_{yx_2} – парный коэффициент корреляции между y и x_2;

$r_{x_1x_2}$ – парный коэффициент корреляции между x_1 и x_2.

Парный коэффициент корреляции между y и x_1 рассчитывается по формуле

$$r_{\hat{yx_1}} = \frac{m*\sum_{i=1}^{m} x_{1i} * y_i - \sum_{i=1}^{m} x_{1i} * \sum_{i=1}^{m} y_i}{\sqrt{(m*\sum_{i=1}^{m} x_{1i}^2 - \sum_{i=1}^{m} x_{1i} * \sum_{i=1}^{m} x_{1i}) * (m*\sum_{i=1}^{m} y_i^2 - \sum_{i=1}^{m} y_i * \sum_{i=1}^{m} y_i)}}. \quad (3.29)$$

Парный коэффициент корреляции между y и x_2 рассчитывается по формуле

$$r_{\hat{yx_2}} = \frac{m*\sum_{i=1}^{m} x_{2i} * y_i - \sum_{i=1}^{m} x_{2i} * \sum_{i=1}^{m} y_i}{\sqrt{(m*\sum_{i=1}^{m} x_{2i}^2 - \sum_{i=1}^{m} x_{2i} * \sum_{i=1}^{m} x_{2i}) * (m*\sum_{i=1}^{m} y_i^2 - \sum_{i=1}^{m} y_i * \sum_{i=1}^{m} y_i)}}. \quad (3.30)$$

Парный коэффициент корреляции между x_1 и x_2 рассчитывается по формуле

$$r_{x_1x_2} = \frac{m*\sum_{i=1}^{m} x_{1i} * x_{2i} - \sum_{i=1}^{m} x_{1i} * \sum_{i=1}^{m} x_{2i}}{\sqrt{(m*\sum_{i=1}^{m} x_{2i}^2 - \sum_{i=1}^{m} x_{2i} * \sum_{i=1}^{m} x_{2i}) * (m*\sum_{i=1}^{m} x_{1i}^2 - \sum_{i=1}^{m} x_{1i} * \sum_{i=1}^{m} x_{1i})}}, \quad (3.31)$$

где $R^2_{\hat{y x_1 x_2}}$ – совокупный коэффициент детерминации.

Частные коэффициенты корреляции позволяют провести анализ тесноты связи между эндогенной переменной y и одной из экзогенных переменных при неизменных других.

Частный коэффициент корреляции между y и x_1 при неизменной x_2 рассчитывается по формуле

$$r_{yx_1(x_2)} = \frac{r_{yx_1} - r_{yx_2} * r_{x_1x_2}}{\sqrt{(1-r_{yx_2}^2) * (1-r_{x_1x_2}^2)}}. \quad (3.32)$$

Частный коэффициент корреляции между y и x_2 при неизменной x_1 рассчитывается по формуле

$$r_{y(x_1)x_2} = \frac{r_{yx_2} - r_{yx_1} * r_{x_1x_2}}{\sqrt{(1-r_{yx_1}^2) * (1-r_{x_1x_2}^2)}}, \quad (3.33)$$

где $r^2{}_{yx_1(x_2)}$, $r^2{}_{y(x_1)x_2}$ – частные коэффициенты детерминации.

Частные коэффициенты эластичности рассчитывают по формулам

$$Э^{\wedge}_{y x_1(x_2)} = \frac{a_1 * \overline{x_1}}{\overline{y}}, \quad (3.34)$$

где $\overline{x_1}$ – среднее арифметическое значение объема выборки X_1.

$$Э^{\wedge}_{y(x_1) x_2} = \frac{a_2 * \overline{x_2}}{\overline{y}}, \quad (3.35)$$

где $\overline{x_2}$ – среднее арифметическое значение объема выборки X_2.

Частные β-коэффициенты рассчитываются по формулам

$$\beta^{\wedge}_{y x_1(x_2)} = \frac{a_1 * S_{x_1}}{S_y}, \quad (3.36)$$

где a_1 – выборочный коэффициент;

S_{x_1} – средняя квадратическая ошибка (отклонение) объема выборки X_1;

S_y – средняя квадратическая ошибка (отклонение) объема выборки Y.

$$\beta^{\wedge}_{y(x_1) x_2} = \frac{a_1 * S_{x_2}}{S_y}, \quad (3.37)$$

где S_{x_2} – средняя квадратическая ошибка (отклонение) объема выборки X_2.

Средняя квадратическая ошибка объема выборки X_1 рассчитывается по формуле

$$S_{x_1} = \sqrt{\frac{\sum_{i=1}^{m} (x_{1i} - \overline{x_1})^2}{m}}. \quad (3.38)$$

Средняя квадратическая ошибка объема выборки X_2 рассчитывается по формуле

$$S_{x_2} = \sqrt{\frac{\sum_{i=1}^{m} (x_{2i} - \overline{x_2})^2}{m}}. \quad (3.39)$$

Доверительный интервал функции регрессии рассчитывается по формуле (3.17).

3.3 Нелинейная регрессионная модель

Уравнение связи – нелинейная функция (гиперболическая, показательная, степенная, логарифмическая и т.д.). Существует два вида нелинейных регрессионных моделей:

1) нелинейная относительно экзогенных переменных, но линейная относительно оцениваемых параметров регрессии;

2) нелинейные относительно экзогенных переменных и относительно оцениваемых параметров.

Решение основано на линеаризации нелинейного уравнения связи.

Линеаризация – процесс преобразования нелинейной функции к линейной.

Линеаризация производится заменой или логарифмированием:

1) гиперболическая функция связи $y = a_0 + \frac{a_1}{x}$. Модель нелинейна относительно экзогенной переменной x. Линеаризация выполняется заменой переменной $x' = \frac{1}{x}$. Исходное уравнение связи принимает вид $y = a_0 + a_1 * x'$;

2) показательная функция связи $y = a_0 * e^{a_1 * x}$. Модель независима относительно экзогенной переменной x и относительно оцениваемого параметра – коэффициента a_1. Линеаризация выполняется логарифмированием выражения: $\ln y = \ln a_0 + a_1 * x$. Выполним замену переменных: $y' = \ln y$, $a_0' = \ln a_0$. Исходное уравнение связи принимает вид $y' = a_0' + a_1 * x'$;

3) степенная функция связи $y = a_0 * x^{a_1}$. Модель нелинейна относительно оцениваемого параметра – коэффициента a_0. Линеаризация выполняется логарифмированием выражения: $\ln y = \ln a_0 + a_1 * \ln x$. Выполним замену переменных: $y' = \ln y$, $a_0' = \ln a_0$, $x' = \ln x$. Исходное уравнение связи принимает вид $y' = a_0' + a_1 * x'$;

4) логарифмическая функция связи $y = a_0 + a_1 * \ln x$. Модель нелинейна относительно экзогенной переменной x. Линеаризуется заменой $x' = \ln x$. Исходное уравнение связи принимает вид $y = a_0 + a_1 * x'$.

3.3.1 Метод нелинейного оценивания параметров

Метод наименьших квадратов применяется к нелинейным регрессионным моделям, если возможна линеаризация, т.е. они являются нелинейными по экзогенным переменным или нелинейны по оцениваемым параметрам, но внутренне линейны.

3.3.1.1 Оценивание параметров уравнения параболической зависимости МНК

Уравнение связи $\hat{y} = a_0 + a_1 * x + a_2 * x^2$ – полином второго порядка является нелинейным относительно экзогенной переменной x. Для определения коэффициентов a_0, a_1, a_2 минимизируется

$$S_{кв} = \sum_{i=1}^{m} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{m} (y_i - a_0 - a_1 * x - a_2 * x^2)^2 -> \min . \qquad (3.40)$$

Процесс минимизации сводится к вычислению частных производных функции $S_{кв}$ по каждому из оцениваемых параметров a_0, a_1, a_2.

$$\begin{cases} \dfrac{\partial S_{кв}}{\partial a_0} = -2 * \sum_{i=1}^{m} (y_i - a_0 - a_1 * x_i - a_2 * x_i^2) = 0 \\ \dfrac{\partial S_{кв}}{\partial a_1} = -2 * \sum_{i=1}^{m} (y_i - a_0 - a_1 * x_i - a_2 * x_i^2) * x_i = 0 \\ \dfrac{\partial S_{кв}}{\partial a_2} = -2 * \sum_{i=1}^{m} (y_i - a_0 - a_1 * x_i - a_2 * x_i) * x_i^2 = 0 \end{cases} \quad (3.41)$$

СНУ имеет вид

$$\begin{cases} m * a_0 + a_1 * \sum_{i=1}^{m} x_i + a_2 * \sum_{i=1}^{m} x_i^2 = \sum_{i=1}^{m} y_i \\ a_0 * \sum_{i=1}^{m} x_i + a_1 * \sum_{i=1}^{m} x_i^2 + a_2 * \sum_{i=1}^{m} x_i^3 = \sum_{i=1}^{m} y_i * x_i \\ a_0 * \sum_{i=1}^{m} x_i^2 + a_1 * \sum_{i=1}^{m} x_i^3 + a_2 * \sum_{i=1}^{m} x_i^4 = \sum_{i=1}^{m} y_i * x_i^2 \end{cases} \quad (3.42)$$

Коэффициенты a_0, a_1, a_2 определяются методом Гаусса, если свести СНУ к линейному виду с помощью метода замен.

В общем случае полином m-ой степени имеет вид

$$\hat{y} = a_0 + a_1 * x + a_2 * x^2 + ...a_n * x^m. \quad (3.43)$$

Минимизируется сумма квадратов

$$S_{кв} = \sum_{i=1}^{m} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{m} (y_i - a_0 - a_1 * x_i - a_2 * x_i^2 - ... - a_n * x_i^m)^2 -> \min. \quad (3.44)$$

СНУ имеет вид

$$\begin{cases} m * a_0 + a_1 * \sum_{i=1}^{m} x_i + a_2 * \sum_{i=1}^{m} x_i^2 + ... + a_n * \sum_{i=1}^{m} x_i^m = \sum_{i=1}^{m} y_i \\ a_0 * \sum_{i=1}^{m} x_i + a_1 * \sum_{i=1}^{m} x_i^2 + a_2 * \sum_{i=1}^{m} x_i^3 + ... + a_n \sum_{i=1}^{m} x_i^{m+1} = \sum_{i=1}^{m} y_i * x_i \\ ..... \\ a_0 * \sum_{i=1}^{m} x_i^{m-1} + a_1 * \sum_{i=1}^{m} x_i^m + a_2 * \sum_{i=1}^{m} x_i^{m+1} + ... + a_n * \sum_{i=1}^{m} x_i^{2*m-1} = \sum_{i=1}^{m} y_i * x_i^{m-1} \\ a_0 * \sum_{i=1}^{m} x_i^m + a_1 * \sum_{i=1}^{m} x_i^{m+1} + a_2 * \sum_{i=1}^{m} x_i^{m+2} + ... + a_n * \sum_{i=1}^{m} x_i^{2*m} = \sum_{i=1}^{m} y_i * x_i^m \end{cases} \quad (3.45)$$

Решением системы (3.45) являются оценки коэффициентов регрессионной зависимости, выраженной полиномом m-го порядка.

3.3.1.2 Оценивание параметров уравнения показательной зависимости МНК

Показательная функция $\hat{y} = a_0 * a_1^{x_i}$ – нелинейная функция по оцениваемому параметр a_1, но внутренне линейная. Для линеаризации применяется логарифмирование

$$\log y_i = \log a_0 + x_i * \log a_1 . \quad (3.46)$$

Применим метод замены: $\log y_i = y_i'$, $\log a_0' = a_0'$, $\log a_1' = a_1'$. Преобразованная показательная функция имеет вид

$$y_i' = a_0' + a_1' * x_i . \quad (3.47)$$

Для определения оценок параметров минимизируется сумма квадратов

$$S_{кв} = \sum_{i=1}^{m} (\log y_i - \log \hat{y}_i)^2 -> \min . \quad (3.48)$$

СНУ имеет вид

$$\begin{cases} m * a_0' + a_1' * \sum\limits_{i=1}^{m} x_i = \sum\limits_{i=1}^{m} y_i = \sum\limits_{i=1}^{m} \log y_i \\ a_0' * \sum\limits_{i=1}^{m} x_i + a_1' * \sum\limits_{i=1}^{m} x_i^2 = \sum\limits_{i=1}^{m} y_i' * x_i = \sum\limits_{i=1}^{m} x_i * \log y_i \end{cases} \quad (3.49)$$

МНК-оценки параметров для нелинейных регрессионных моделей, сводимых к линейному виду, не обладают свойством несмещенности.

3.3.1.3 Анализ нелинейной регрессионной модели

Качество нелинейной регрессионной модели определяется с помощью индекса корреляции для нелинейных форм, индекса детерминации и коэффициента эластичности.

Индекс корреляции для нелинейных форм связи вычисляется по формуле

$$r_{\hat{y}x} = \sqrt{1 - \frac{\sum\limits_{i=1}^{m} (y_i - \hat{y}_i)^2}{\sum\limits_{i=1}^{m} (y_i - \overline{y_i})^2}} , \quad (3.50)$$

где y_i – наблюденная эндогенная переменная;

$\hat{y}_i$ – расчетная эндогенная переменная;

$\overline{y}_i$ – среднее арифметическое значение объема выборки Y;

m – объем выборки, $i = \overline{1,m}$.

Индекс корреляции для нелинейной формы связи изменяется в пределах $r_{\hat{y}x} \in [0,+1]$. Чем ближе его значение к 1, тем сильнее взаимосвязь между эндогенной и экзогенной переменными.

Индекс детерминации $r^{2\hat{}}_{yx}$ для нелинейной формы связи рассчитывается как квадрат индекса корреляции для нелинейной формы связи.

Индекс детерминации $r^{2\hat{}}_{yx}$ для нелинейной формы связи по характеристикам аналогичен множественному коэффициенту детерминации. Чем больше значение $r^{2\hat{}}_{yx}$, тем лучше уравнение регрессии описывает взаимосвязь.

Общий коэффициент эластичности показывает на сколько процентов изменится y при изменении x на 1%. Общий коэффициент эластичности рассчитывается по формуле

$$\Э_{\hat{yx}} = y'_x \frac{x}{y}, \qquad (3.51)$$

где y'_x – первая производная эндогенной переменной по экзогенной переменной.

4 Варианты заданий курсовой работы

1 Транспортная задача. Метод потенциалов /5/.

2 Эвристические методы поиска решения на графах. Метод минимальной стоимости /7/.

3 Динамическое программирование. Метод обратной прогонки /5/.

4 Сортировка файлов простым слиянием /2, 4/.

5 Сортировка файлов естественным слиянием /2, 4/.

6 Транспортная задача. Распределительный метод /5/.

7 Остовное дерево наименьшей стоимости. Алгоритм Прима /4/.

8 Остовное дерево наименьшей стоимости. Алгоритм Борувки /1/.

9 Алгоритм определения максимального потока сети /5/.

10 Алгоритм минимизации стоимости потока в сети с ограниченной пропускной способностью /5/.

11 Алгоритм определения критического пути /5/.

12 Методы поиска решения на графах. Методы поиска в ширину /4, 8/.

13 Методы поиска решения на графах. Методы поиска в глубину /4, 8/.

14 Эвристические методы поиска решения на графах. Метод экстремума /7/.

15 Алгоритм поиска оптимального кода. Алгоритм Хаффмана /4/.

16 Динамическое программирование. Метод прямой прогонки /4, 5/.

17 Алгоритм поиска оптимального кода. Алгоритм Шеннона-Фено /4/.

18 Алгоритм сжатия данных. Алгоритм «арифметическое кодирование» /4/.

19 Структуры данных и алгоритмы для внешней памяти. Внешние деревья поиска. B дерево (B tree) /3, 8/.

20 Алгоритм метода ветвей и границ. Задача расшифровки криптограмм /6/.

21 Остовное дерево наименьшей стоимости. Алгоритм Крускала /4, 6/.

22 Алгоритмы внешней сортировки. Многофазная сортировка /3, 4/.

23 Алгоритмы внешней сортировки. Каскадная сортировка /3, 4/.

24 Алгоритм внешней сортировки. Сбалансированное многопутевое слияние /4/.

25 Алгоритмы на графах. Задача поиска кратчайшего отрицательно взвешенного пути при одном источнике (алгоритм Беллмана-Форда) /9/.

26 Алгоритм Йена /6/.

27 Алгоритм Форда /6/.

28 Оптимальное двоичное дерево поиска. Алгоритм Гарсия-Воча /3/.

29 Поиск в деревьях. Красно-черные деревья (RB tree) /3, 8, 9/.

30 Структуры данных и алгоритмы для внешней памяти. Внешние деревья поиска. B+ дерево (B+ tree) /3, 4, 8/.

31 Алгоритм Джонсона нахождения всех пар кратчайших путей /9/.

32 Алгоритм с возвратом. Задача Гамильтона /4/.

33 Алгоритм метода ветвей и границ. Задача коммивояжера /6/.

34 Алгоритм Литтла. Задача коммивояжера /6/.

ЗАКЛЮЧЕНИЕ

Изучение алгоритмов и структур данных позволяет создавать качественные эффективные программные продукты. Хорошо разработанный алгоритм и правильно подобранные структуры данных позволяют ускорить работу программы, снизить финансовые затраты.

Выбор наилучшего алгоритма и структур данных – сложный процесс, требующий знаний и навыков.

Выполнение курсовой работы позволяет закрепить теоретические знания по дисциплине алгоритмы и структуры данных и приобрести практические навыки в разработке и анализе программных алгоритмов.

В методических указаниях приведены описания алгоритмов решения задач, варианты заданий для выполнения курсовых работ, анализ программных алгоритмов и математические методы оценивания коэффициентов уравнений связи между временем выполнения программного алгоритма и размером.

Приведен список литературных первоисточников и ссылки на них для каждой темы курсовой работы.

СПИСОК ЛИТЕРАТУРЫ

1 Сэджвик, Р. Фундаментальные алгоритмы на C++ [Текст] / Р. Сэджвик. – М. : «DiaSoft», 2001. – 688 с. – Ч. 1-5.

2 Давыдов, В. Г. Технологии программирования C++ [Текст] / В. Г. Давыдов. – СПб. : БХВ-Петербург, 2005. – 672 с.

3 Анашкина, Н. В. Технологии и методы программирования [Текст] : учеб. пособие для студ. учреждений высш. проф. образования / Н. В. Анашкина, Н. Н. Петухова, В. Ю. Смольянинов. – М. : Издательский центр «Академия», 2012. – 384 с.

4 Хусаинов, Б. С. Структуры и алгоритмы обработки данных. Примеры на языке Си (+CD) [Текст] : учеб. Пособие / Б. С. Хусаинов. – М. : Финансы и статистика, 2004. – 464 с.

5 Таха Хемди А. Введение в исследование операций [Текст] / Хемди А. Таха : Пер. с англ. – 7-е издание. – М. : Издательский дом «Вильямс», 2005. – 912 с.

6 Гагарина, Л. Г. Алгоритмы и структуры данных [Текст] : учеб. пособие / Л. Г. Гагарина, В. Д. Колдаев. – М. : Финансы и статистика, ИНФРА-М, 2009. – 304 с.

7 Шилдт, Г. Искусство программирования на C++ [Текст] / Г. Шилдт. – СПб. : БХВ – Петербург, 2005. – 496 с.

8 Романенко, Т. А. Структуры и алгоритмы обработки данных. ЭУМК. [Электронный ресурс]. – Новосибирск, Издательство НГТУ, 2012. URL: http://edu.nstu.ru/courses/saod/file_structure.htm (дата обращения: 30 января 2014).

9 Уайс, М. А. Организация структур данных и решение задач на C++ [Текст] / М. А. Уайс; пер. с англ. – М. : ЭКОМ Паблишерз, 2008. – 896 с.

10 Дик, Д. И. Требования к оформлению текстовой документации курсовых и дипломных проектов (работ) [Текст] : Методические указания для студентов направлений (специальностей) 230000 (230105), 090000 (090105) Д. И. Дик. – Курган : Изд-во КГУ, 2008. – 40 с.

Приложение А. Пример курсовой работы.

1.1 Титульный лист

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГБОУ ВПО «КУРГАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Кафедра «Программное обеспечение автоматизированных систем»

Курсовая работа
по дисциплине
«Алгоритмы и структуры данных»

РФ. КГУ. КР.231000.62. 201078

Разработал студент гр. Т - 20012 _____/ Е.Б.Белинский/

Руководитель,
канд. техн. наук, доцент _____/ А.М. Семахин /

Члены комиссии _____/А.М. Семахин/

_____/А.Г. Кокин/

Проект защищен с оценкой _____ «___» _____ 2013 г.

Курган 2013

1.2. Техническое задание

Курганский Государственный университет
Кафедра программного обеспечения автоматизированных систем

УТВЕРЖДЕНО
Распоряжением по кафедре ПОАС
№ 1 от 1 сентября 2013 г.

ТЕХНИЧЕСКОЕ ЗАДАНИЕ
на выполнение курсовой работы
по дисциплине «Алгоритмы и структуры данных»
Специальность 231000.62 – Программная инженерия
Тема: ***«Транспортная задача. Метод потенциалов».***

1. **Назначение, цели и задачи разработки**

Проектируемая программная система **предназначена для** *анализа качества алгоритмов поиска путей перевозки продукции от поставщиков потребителям на сетевом графике.*

Основная учебная цель выполнения разработки – повышение уровня квалификации разработчика в области проектирования, программной реализации и анализа сложных структур данных и алгоритмов их обработки.

Основные задачи, решаемые разработчиком в процессе выполнения курсовой работы:

программная реализация одного из приближенных методов получения опорного плана:

1) Метод северо-западного угла.
2) Метод минимального элемента.
3) Метод Фогеля.

программная реализация метода потенциалов;
разработка программы для оценки качества реализованного алгоритма;
проведение экспериментального исследования алгоритма и анализ его результатов;
документирование проекта в соответствии с установленными требованиями.

2. **Характер разработки:** прикладная квалификационная работа.
3. **Основания для разработки**

Учебный план специальности 231000.62 – Программная инженерия
Рабочая программа дисциплины " Алгоритмы и структуры данных ".
Распоряжение по кафедре ПОАС №__1_ от 01.09.2013 г.

4. **Плановые сроки выполнения –** осенний семестр 2013/14 учебного года:

Начало - 01.09.2013 г.
Окончание - 28.12.2013 г.

5. **Требования к проектируемой системе**

5.1 Требования к функциональным характеристикам

Проектируемая система должна обеспечивать выполнение следующих основных функций:

ввод исходных данных и представление сетевого графика постановки задачи;
преобразование сетевого графика в распределительную таблицу и проверка условий для решения задачи;
получение опорного плана и проверка на невырожденность;
преобразование распределительной таблицы в сетевой график;
вывод промежуточного решения по итерациям и оптимального решения;
оценка стоимости реализации алгоритма по временным и объемным параметрам;
хранение исходных данных с возможностью их загрузки для повторной обработки;

хранение результатов решения с возможностью их повторной визуализации.

5.2 Требования к эксплуатационным характеристикам

модульность;

расширяемость

5.3 Требования к программному обеспечению:

среда разработки – MS Visual C++ версии не ниже 6.0

6. Стадии и этапы разработки

6.1 Эскизный проект (ЭП)

Обзор приложений задачи поиска путей перевозки продукции от поставщиков к потребителям с наименьшими транспортными затратами на сетевом графике;

Разработка (описание) алгоритмов поиска оптимального плана на сетевом графике;

Разработка методики проведения экспериментального исследования;

Подготовка проектной документации.

6.2 Технический проект (ТП)

Разработка структур и форм представления данных;

Разработка структуры программного комплекса;

Подготовка проектной документации.

6.3 Рабочий проект (РП)

Программная реализация;

Тестирование и отладка программы;

Подготовка программной и эксплуатационной документации.

6.4 Эксплуатация (Э)

Описание и анализ результатов проведения проведенного исследования.

7. Требования к документированию работы

7.1 К защите курсовой работы должен быть представлен *альбом*, включающий следующие проектные, программные и эксплуатационные документы:

7.1.1 Опись альбома

7.1.2 Пояснительная записка (состав основных разделов документа)*:*

Аналитический обзор

Описание алгоритма решения задачи

Описание структуры программного комплекса

Описание структур данных

Анализ алгоритма решения задачи

Описание методики проведения экспериментального исследования

Описание и анализ результатов проведенного исследования

Выводы по результатам проведенного анализа

7.1.3 Спецификация

7.1.4 Описание программы

7.1.5 Текст программы (на машинном носителе)

7.1.6 Руководство пользователя

7.1.7 Руководство программиста

7.2 Требования к структуре документов определены соответствующими стандартами ЕСПД.

7.3 Требования к оформлению определены соответствующими методическими указаниями.

8. Порядок контроля и приемки

8.1 Контроль выполнения курсовой работы проводится руководителем поэтапно в соответствии с утвержденным графиком выполнения проекта.

8.2 На завершающем этапе руководитель осуществляет нормоконтроль представленной исполнителем документации и принимает решение о допуске

(недопуске) проекта к защите.

8.3 Защита курсовой работы проводится комиссией в составе не менее двух человек, включая руководителя работы.

8.4 В процессе защиты работы исполнитель представляет документацию, делает краткое сообщение по теме разработки и демонстрирует ее программную реализацию.

8.5 При выставлении оценки учитывается:

- степень соответствия представленной разработки требованиям технического задания;
- качество программной реализации, документации и доклада;
- соблюдение исполнителем графика выполнения курсовой работы.

График выполнения курсовой работы				
Стадия проекта	Содержание работ, отчетная документация	Контроль выполнения		
		Плановая Дата	Фактическая дата	Подпись руководителя
ЭП	1.Обзор приложений и анализ методов поиска оптимального плана на сетевом графике	16.09.13		
	2.Описание алгоритмов	24.09.13		
	3. Методика исследования качества алгоритмов	2.10.13		
ТП	1. Разработка структуры программного комплекса	9.10.13		
	2. Разработка структур и форм представления данных	16.10.13		
РП	1. Результаты тестирования программных модулей	31.10.13		
	2. Сборка и тестирование программного комплекса	06.11.13		
	3. Текст программы	13.11.13		
	4. Описание программы	30.11.13		
	5. Руководство пользователя	13.12.13		
	6. Нормоконтроль документации	20.12.13		
Э	1. Результаты экспериментального исследования	27.12.13		
Защита курсового проекта		18.12.13-28.12.13		

Исполнитель: Студент гр. Т-20012 _____ Белинский Е.Б.

Руководитель: к.т.н., доцент _____ Семахин А.М.

1.3 Опись альбома

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГБОУ ВПО «КУРГАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Кафедра «Программное обеспечение автоматизированных систем»

Курсовая работа

По дисциплине «Алгоритмы и структуры данных»

ОПИСЬ АЛЬБОМА

РФ КГУ 231000.62.КР.201078 01

Разработал студент гр. Т-20012 _____ / Е.Б.Белинский /

Руководитель,

канд. техн. наук, доцент _____ /А.М.Семахин /

Проект защищен с оценкой ____ _____ «___» _____ 2013 г.

Члены комиссии

_____/ А.М.Семахин /

_____ / А.Г.Кокин /

Курган 2013

№ с-ки	Обозначение	Наименование
	Программные документы	
1	РФ КГУ 231000.62.КР.201078 02	Пояснительная записка
2	РФ КГУ 231000.62.КР. 201078 03	Спецификация
3	РФ КГУ 231000.62.КР. 201078 04	Описание программы

4	РФ КГУ 231000.62.КР. 201078 05	Руководство пользователя
5	РФ КГУ 231000.62.КР. 201078 06	Руководство программиста

1.4 Пояснительная записка

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГБОУ ВПО «КУРГАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Кафедра «Программное обеспечение автоматизированных систем»

Курсовая работа

По дисциплине «Алгоритмы и структуры данных»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Листов 29

РФ КГУ 231000.62.КР.201078 02

Разработал студент гр. Т-20012 _____ / Е.Б.Белинский /

Руководитель,

канд. техн. наук, доцент _____ /А.М.Семахин /

Проект защищен с оценкой _____ «___» _____ 2013 г.

Члены комиссии _____ /А.М. Семахин/

_____ /А.Г. Кокин/

Курган 2013

АННОТАЦИЯ

Документ содержит общие сведения о программе, ее описании, структуре и вариантах использования, а также входных и выходных данных.

СОДЕРЖАНИЕ

1 АНАЛИТИЧЕСКИЙ ОБЗОР

Данное приложение формализует алгоритм решения транспортной задачи методом потенциалов. Визуальное приложение разработано с использованием объектно-ориентированного метода программирования. В программе обеспечен ввод исходных данных (размер задачи, распределительная таблица), поиск начального опорного плана методом минимального элемента, поиск оптимального плана, проведение эксперимента и вывод данных в форме таблицы и графика.

Приложение написано на языке C++ в программной среде Microsoft Visual Studio 2012 Ultimate.

2 ОПИСАНИЕ ИСХОДНЫХ ДАННЫХ

Исходными данными в программе являются:

1) размер задачи;
2) распределительная таблица.

3 ОПИСАНИЕ АЛГОРИТМА

Транспортная задача (ТЗ) – специальный класс задач линейного программирования. Она описывают перевозку груза из пункта отправления в пункт назначения. Назначение ТЗ – определение объемов перевозок из пунктов отправления в пункты назначения с минимальной суммарной стоимостью перевозок. При этом должны учитываться ограничения, накладываемые на объемы грузов, имеющихся в пунктах отправления (предложение), и ограничения, учитывающие потребность грузов в пунктах назначения (спрос). В ТЗ предполагается, что стоимость перевозки по какому-либо маршруту прямо пропорциональна объему груза, перевозимого по этому маршруту. В общем случае модель ТЗ можно применять для описания ситуаций, связанных с управлением запасами, управлением движением капиталов, составлением расписаний, назначением персонала.

3.1 Постановка транспортной задачи

Транспортная задача – частный случай задачи линейного программирования. **Общая постановка транспортной задачи** состоит в определении оптимального плана перевозок груза из m пунктов отправления $A_1,...,A_m,(i=\overline{1,m})$ в n пунктов назначения $B_1,...B_n$ $(j=\overline{1,n})$. Критерий оптимальности – минимальная стоимость перевозок всего груза, либо минимальное время доставки.

Математическая постановка транспортной задачи. Пусть $A_1, A_2,...A_m$ - пункты отправления груза, $i=\overline{1,m}$. a_i - запасы груза в i пункте отправления. $B_1, B_2,..., B_n$ -пункты назначения груза, $j=\overline{1,n}$. b_j - потребности в грузе в j пункте назначения. c_{ij} - тарифы перевозки единицы груза из i пункта отправления в j пункт назначения. X_{ij} - количество единиц груза, перевозимого из i пункта отправления в j пункт назначения.

Необходимо определить минимальную стоимость перевозки:

$$Min \leftarrow Z = \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij} * X_{ij}\ \textit{Целевая функция}$$

при ограничениях

$$\begin{cases} \sum\limits_{i=1}^{m} X_{ij} = b_j,\ j=\overline{1,n}, & (1) \\ \sum\limits_{j=1}^{n} X_{ij} = a_i, i=\overline{1,m} & (2)\ \text{Ограничения} \\ X_{ij} \geq 0, i=\overline{1,m},\ j=\overline{1,n} & (3) \end{cases}$$

1) обеспечивается доставка необходимого количества груза в пункт назначения;
2) обеспечивается вывоз имеющегося груза из всех пунктов отправления;
3) условие неотрицательности.

Исходные данные транспортной задачи записывают в виде распределительной таблицы. Распределительная таблица приведена в таблице 1.

Матрица $C=(c_{ij})_{m*n}$ – матрица тарифов, а числа c_{ij} - тарифы.

Матрица $X=(x_{ij})_{m*n}$ – матрица перевозок, числа x_{ij} обозначают количество единиц груза, которое надо доставить из i пункта отправления в j пункт назначения.

Общее наличие груза у поставщиков – $\sum\limits_{i=1}^{m} a_i$

Таблица 1 – Распределительная таблица транспортной задачи

Поставщик	Потребитель				Запасы груза
	B_1	B_2	…	B_n	
A_1	X_{11} C_{11}	X_{12} C_{12}	…	X_{1n} C_{1n}	a_1
A_2	X_{21} C_{21}	X_{22} C_{22}	…	X_{2n} C_{2n}	a_2
…	…	…	…		
A_m	X_{m1} C_{m1}	X_{m2} C_{m2}	…	X_{mn} C_{mn}	a_m

Потребность в грузе	b_1	b_2	…	b_n	$\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j$

Общая потребность в грузе в пунктах назначения $\sum_{j=1}^{n} b_j,\ j=\overline{1,n},\ i=\overline{1,m}.$

Модель закрытая, если общая потребность в грузе в пунктах назначения равна запасу в пунктах отправления, т. е. $\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j,\ i=\overline{1,m},\ j=\overline{1,n}.$ Модель открытая, если $\sum_{i=1}^{m} a_i > \sum_{j=1}^{n} b_j$, или $\sum_{j=1}^{n} b_j > \sum_{i=1}^{m} a_i$. Для разрешимости транспортной задачи необходимо и достаточно, чтобы запасы груза в пунктах отправления были равны потребностям в грузе в пунктах назначения, т.е. чтобы выполнялось условие $\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j,\ i=\overline{1,m},\ j=\overline{1,n}.$ В случае превышения запаса над потребностью, т.е. $\sum_{i=1}^{m} a_i > \sum_{j=1}^{n} b_j$, вводится фиктивный $(n+1)$ пункт назначения с потребностью $b_{n+1} = \sum_{i=1}^{m} a_i - \sum_{j=1}^{n} b_j$ и соответствующие тарифы считаются равными нулю: $C_{in+1} = 0, i=\overline{1,m}.$ Аналогично, при $\sum_{i=1}^{m} a_i < \sum_{j=1}^{n} b_j$ вводится фиктивный $(m+1)$ пункт отправления с запасом груза $a_{m+1} = \sum_{j=1}^{n} b_j - \sum_{i=1}^{m} a_i$ и тарифы полагаются равными нулю: $C_{m+1\,j} = 0,\ j=\overline{1,n}.$ На рисунке 1 показано представление ТЗ в виде графа с m пунктами отправления и n пунктами назначения, которые показаны в виде вершин графа. Дуги, соединяющие узлы сети, соответствуют маршрутам, связывающим пункты отправления и назначения. С дугой (i, j), соединяющий пункт отправления i с пунктом назначения j, соотносятся два вида данных: стоимость c_{ij} перевозки единицы груза из пункта i в пункт j и количество перевозимого груза x_{ij}. Объем грузов в пункте отправления i равен a_i, а объем грузов в пункте назначения j - b_j. Задача состоит в определении неизвестных величин x_{ij}, минимизирующих суммарные транспортные расходы и удовлетворяющих ограничениям, накладываемым на объемы грузов в пунктах отправления (предложения) и пунктах назначения (спрос).

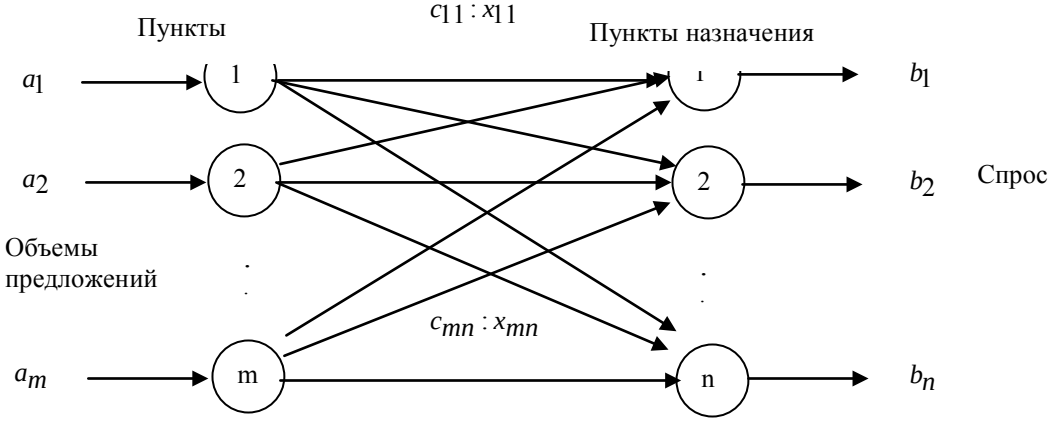


Рисунок 1 – Представление ТЗ в виде графа

3.2. Алгоритм решения транспортной задачи

Алгоритм состоит из четырех этапов.

Этап 1. Представление данных в форме стандартной таблицы и поиск допустимого распределения ресурсов (опорного плана транспортной задачи). Допустимым называется такое распределение ресурсов, которое позволяет удовлетворить весь спрос в пунктах назначения и вывезти весь запас продуктов из пунктов производства.

Этап 2. Проверка полученного распределения ресурсов на оптимальность.

Этап 3. Если распределение не оптимально, то ресурсы перераспределяются, снижая стоимость транспортировки.

Этап 4. Повторная проверка оптимальности полученного распределения ресурсов.

Итеративный процесс повторяется до тех пор, пока не будет получено оптимальное решение.

Применение алгоритма требует условия:

1) даны стоимости перевозки единицы продукции из каждого пункта производства в каждый пункт назначения;
2) даны запасы продуктов в пункте производства;
3) даны потребности в продуктах в каждом пункте потребления;
4) общее предложение равно общему спросу.

3.3 Методы решения транспортной задачи

Методы получения опорного плана:

1) метод северо-западного угла;
2) метод минимального элемента;
3) метод Фогеля.

Методы получения оптимального плана:

1) распределительный метод;
2) метод потенциалов.

Метод северо-западного угла. Заполняют таблицу, начиная с левого верхнего (северо-западного) угла, двигаясь по строке вправо или по столбцу вниз. В клетку (1,1) заносят меньшее из чисел a_1 и b_1, т.е. $x_{11} = \min(a_1; b_1)$. Если $a_1 > b_1$, то $x_{11} = b_1$ и первый столбец "закрыт" для заполнения остальных его клеток, т.е. $x_{i1} = 0,\ \overline{i = 2, m}.$ Потребности первого потребителя удовлетворены полностью. Двигаясь далее по первой строке, записываем в соседнюю клетку $(1, 2)$ меньшее из чисел $a_1 - b_1, b_2$, т.е. $x_{12} = \min(a_1 - b_1, b_2)$. Если $b_2 > a_1$, то аналогично "закрывается" первая строка, т.е. $x_{1j} = 0,\ \overline{j = 2, n}.$ Переходим к заполнению соседней клетки $(2,1)$, куда заносим $x_{21} = \min(a_2; b_1 - a_1)$. Заполнив вторую клетку $(1, 2)$ или $(2,1)$, переходим к заполнению следующей, третьей клетки, либо по второй строке, либо по второму столбцу. Процесс продолжается до полного исчерпания груза у поставщиков или полного удовлетворения потребителей. Последняя заполненная клетка (m, n), окажется лежащей в последнем n столбце и последней m строке. План, полученный по методу северо-западного угла, будет опорным планом системы ограничений задачи.

Метод минимального элемента. Метод находит лучшее начальное решение, чем метод северо-западного угла, поскольку выбирает переменные, которым соответствуют наименьшие стоимости. Заполнение таблицы начинают с клетки с минимальным тарифом c_{ij}. Остаток по столбцу или строке помещают в клетку того же столбца или строки, которой соответствует следующее по величине значение c_{ij} и т. д. На каждом шаге осуществляется максимально возможная поставка в клетку с минимальным тарифом c_{ij} Иными словами,

последовательность заполняемых клеток определяется по величине c_{ij}, а помещаемые в этих клетках величины X_{ij} как и в методе северо-западного угла.

Метод минимального элемента позволяет найти опорный план транспортной задачи, при котором общая стоимость перевозок груза меньше, чем общая стоимость перевозок при плане, найденном методом северо-западного угла.

Метод аппроксимации Фогеля. По каждой строке и каждому столбцу определяем разность между двумя наименьшими тарифами и записываем ее. Из этих разностей выбираем наибольшую и отмечаем знаком . В строке или столбце, где имеется наибольшая разность заносим в клетку с минимальным тарифом максимально возможную поставку. После этого записываем остаток груза по строкам и столбцам. В строках и столбцах с нулевыми остатками груза прочеркиваем все незанятые клетки. Занятые и прочеркнутые клетки не учитываются на следующих этапах. Все делается в одной таблице. Полученный план близок к оптимальному (обычно оптимальный). Метод Фогеля применяется для получения плана, приближенного к оптимальному, и используется при решении задач вручную, когда число поставщиков и потребителей незначительно.

Распределительный метод. Распределительный метод представляет собой разновидность симплексного метода, так как основан на аналогичном общем принципе расчета: вначале строится исходный опорный план перевозок, затем последовательно производится его улучшение до получения оптимального. Для этого для каждой свободной клетки строят замкнутый цикл. Набор клеток матрицы перевозок, в котором две и только две клетки расположены в одной строке или одном столбце и последняя клетка набора лежит в той же строке или столбце, что и первая, называется замкнутым циклом.

Если замкнутый цикл имеет вид (i,j)→(k,j)→(k,l)→(t,l)→…→(u,v)→(i,v),

то $S_{ij} = C_{ij} - C_{kj} + C_{kl} - C_{tl} + ... + C_{uv} - C_{iv}$

Если алгебраическая сумма S_{ij} отрицательна, то путем изменения значений, стоящих в клетках замкнутого цикла, можно получить план с меньшим значением линейной формы.

Критерием оптимальности при нахождении минимума функций служит неотрицательность алгебраических сумм S_{ij}. Если указанное требование не соблюдено, план не оптимален и подлежит улучшению.

Вычисления при решении транспортной задачи распределительным методом ведутся по алгоритму:

1) исходные данные задачи располагают в распределительной таблице;

2) строят исходный опорный план по методу «северо-западного угла» или по методу «минимального элемента» или методом Фогеля. При этом должны оказаться занятыми $r = m + n - 1$ клеток;

3) производят оценку первой свободной клетки путем построения замкнутого цикла и вычисления по этому циклу величины S_{ij}. Если $S_{ij} < 0$, то переходят к следующему пункту алгоритма;

4) перемещают по циклу количество груза, равное наименьшему из чисел, размещенных в четных клетках цикла. Далее возвращаются к пункту 3. Если $S_{ij} \geq 0$, то оценивают следующую свободную клетку, и т. д., пока не обнаружат клетку с отрицательной оценкой. Среди всех клеток с оценкой меньше нуля нужно найти клетку с наибольшим нарушением оптимальности.

Если, наконец, оценки всех свободных клеток окажутся неотрицательными, то оптимальное решение найдено.

Замечания: 1 При появлении вырожденных опорных решений для сохранения числа занятых клеток $r = m + n - 1$ неизменным оставляют свободной только одну клетку, а во всех остальных освободившихся клетках помещают “базисные” нули, полагая в дальнейшем эти клетки занятыми.

2 На каждом этапе построения нового опорного плана при выполнении пункта 3 алгоритма целесообразно испытывать не все свободные клетки, а выбрать для оценки клетки, имеющие относительно малые тарифы.

Метод потенциалов. Сущность метода потенциалов (модифицированный распределительный метод, метод МОДИ) в следующем: после нахождения опорного плана перевозок, каждому поставщику A_i $(i=\overline{1,m})$ (каждой строке) ставится в соответствие число U_i , а каждому потребителю B_j $(j=\overline{1,n})$ (каждому столбцу) – некоторое число V_j . Числа U_i и V_j называются потенциалами поставщика A_i $(i=\overline{1,m})$ и потребителя B_j $(j=\overline{1,n})$ и выбираются так, чтобы в любой загруженной клетке их сумма равнялась тарифу этой клетки, т. е. $C_{ij}=U_i+V_j$. Для определения чисел U_i и V_j решают систему из m+n-1 уравнений $C_{ij}=U_i+V_j$ с m+n неизвестными.

Одной из неизвестных придают произвольное значение, тогда система имеет однозначное решение. Для проверки оптимальности плана просматривают свободные клетки (i;j) и для каждой из них вычисляют разность $S_{ij}=C_{ij}-(U_i+V_j)$. План оптимален, когда для каждой свободной клетки (i;j) разность $S_{ij}=C_{ij}-(U_i+V_j)\geq 0.$ Полученные разности называются оценками (характеристиками) свободных клеток. Отрицательные оценки указывают на перспективность клеток, загрузка их приведет к улучшению плана. Положительные и нулевые оценки исключают возможность улучшения полученного плана. Переход к новому плану осуществляется по общим правилам распределительного метода. Для наиболее перспективной клетки строится замкнутый контур, вершинам которого приписываются чередующиеся знаки (свободной клетке приписывается положительный знак). В клетках с отрицательным знаком отыскивается наименьший груз, которой “перемещается” по клеткам замкнутого цикла, т. е. прибавляется к клеткам со знаком плюс, включая свободную, и вычитается из клеток со знаком минус. В новом плане вновь определяются потенциалы строк и столбцов и вычисляются оценки для всех свободных клеток. Когда среди оценок не окажется отрицательных, полученный план будет оптимальным.

Алгоритм метода потенциалов содержит этапы:

1) построить опорный план перевозок;

2) Вычислить потенциалы U_i и V_j соответственно поставщиков и потребителей;

3) Вычислить суммы потенциалов (косвенные тарифы) для свободных клеток $U_i+V_j=C_{ij}^{'}$;

4) Проверить разность $S_{ij}=C_{ij}-C_{ij}^{'}$.Если все $S_{ij}\geq 0$ для свободных клеток, полученный план оптимальный. Если хотя бы одна оценка $S_{ij}<0,$ в число занятых вводят клетку для которой оценка минимальна, и получают новый план перевозок. Процесс продолжается до тех пор, пока не будет получен план, для которого все оценки $S_{ij}\geq 0.$

Алгебраическая сумма S_{ij} стоимостей по циклу пересчета свободных клеток (i;j) равна разности между стоимостью C_{ij} и суммой потенциалов U_i и V_j , т. е.

$S_{ij}=C_{ij}-C_{ij}^{'}=C_{ij}-(U_i+V_j)$.

3.4 Оценка сложности алгоритма.

Под **вычислительной сложностью** (или **трудоемкостью**) алгоритма понимают количество операций, необходимых для его выполнения. Она характеризует время работы алгоритма и является некоторой функцией *T(n),* где *n*-размер задачи.

Под **емкостной сложностью** алгоритма понимают объем памяти компьютера, требуемый для реализации алгоритма.

Для оценивания трудоемкости алгоритмов была введена специальная система обозначений – так называемая ***O*-нотация**. Эта нотация позволяет учитывать в функции *f(n)* лишь наиболее значимые элементы, отбрасывая второстепенные.

O-нотация при оценке функции вычислительной сложности алгоритма используется по двум основным причинам:

- чтобы не учитывать вклад малых слагаемых в математических формулах;
- чтобы классифицировать алгоритмы согласно верхней границе их общего времени выполнения.

Важность *O*-оценивания состоит в том, что оно позволяет описывать характер поведения функции *f(n)* с ростом n: насколько быстро или медленно растет эта функция.

O-оценка позволяет разбить все основные функции на ряд групп в зависимости от скорости их роста:

Постоянные функции типа *O*(1), которые с ростом *n* НЕ растут

- функции с логарифмической скоростью роста *O(log n);*
- функции с линейной скоростью роста *O(n);*
- функции с линейно–логарифмической скоростью роста *O(n*log n);*
- функции с квадратичной скоростью роста $O(n^2)$;
- функции со степенной скоростью роста $O(n^a)$ при *a>2;*
- функции с показательной или экспоненциальной скоростью роста $O(2^n)$;
- функции с факториальной степенью роста *O(n!).*

Виды функции сложности алгоритмов

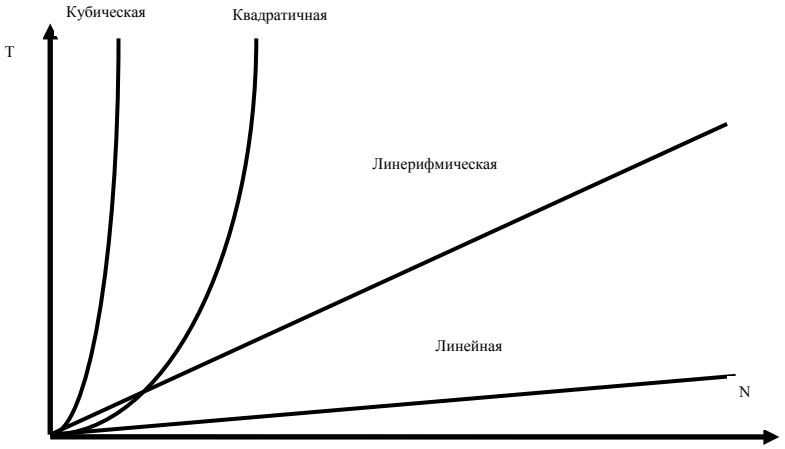
Время выполнения алгоритма T зависит от объема входных данных N

$$T = f(N),$$

где T - время выполнения алгоритма, мс.

N – объем входных данных.

Для оценивания трудоемкости алгоритмов введена специальная система обозначений



– О-нотация.

Рисунок 2 – Графики функций сложности алгоритмов

О-нотация позволяет учитывать в функции f (n) значимые элементы, отбрасывая второстепенные:

1 Кубическая функция – функция f(n), старший член которой содержит N^3. О-нотация имеет вид $O(N^3)$.

2 Квадратическая функция – функция f(n), старший член которой содержит N^2**.** О-нотация имеет вид $O(N^2)$.

3 Линейная функция – функция f(n), старший член которой содержит N**.** О-нотация имеет вид $O(N)$.

4 Функция линерифмическая - функция, старший член которой равен N логарифмов N. О-нотация имеет вид $O(N \log N)$.

Например, в функции $f(n) = 5*n^2 + 3*n + 2$ при больших n компонента n^2 превосходит остальные слагаемые. Поведение функции определяется компонентой n^2. Остальные компоненты отбрасываются. Функция $f(n) = 5*n^2 + 3*n + 2$ имеет оценку поведения (скорость роста значений) $O(n^2)$.

Описание О-нотаций приведено в таблице 2.

Таблица 2 – Описание О-нотаций

О-нотация	Описание
O (1)	Инструкции программы запускаются независимо от n. Время выполнения программы постоянно. (помещение в стек). Операции в программе выполняются один или несколько раз. Алгоритм независимо от размера данных требует одно и тоже время.
O (n)	Время выполнения программы линейно и зависит от n. Входной элемент обрабатывается линейное число раз.
O (n^2)	Время выполнения программы является квадратичным. Алгоритмы используются для небольших n (цикл двойного уровня вложенности, сортировки выбором, вставками)
O (n^3)	Алгоритм программы имеет кубическое время выполнения (цикл тройного уровня вложенности). Применяется для небольших задач. (умножение матриц)
O (logn)	Логарифмическая зависимость. С ростом n программа работает медленнее. Время характерно для программ, которые сводят большую задачу к набору меньших подзадач, уменьшая на каждом шаге размер подзадачи на постоянный коэффициент. Общее решение находится в одной из подзадач (бинарный поиск)
O (n*logn)	Линерифмическая зависимость. Время выполнения программы пропорционально n*logn. Возникает, когда алгоритм решает задачу, разбивая ее на меньшие подзадачи, решает независимо и затем объединяет решения подзадач (комбинация, сортировки быстрая, слиянием)
O (2^n)	Экспоненциальная зависимость. прямое решение задач. (перебор и сравнение различных решений) Для комбинаторных задач нереализуемы. Сведение к прибли-женному алгоритму с приближенным значением.

О-оценивание позволяет описывать характер поведения функции f(n) с ростом n: насколько быстро или медленно растет эта функция.

О-оценка разбивает функции сложности на группы в зависимости от скорости роста:

1 Постоянные функции $O(1)$, которые с ростом n не растут.

2 Функции с логарифмической скоростью роста $O(\log_2 n)$.

3 Функции с линейной скоростью роста $O(n)$.

4 Функции с линейно–логарифмической скоростью роста $O(n*\log_2 n)$.

5 Функции с квадратичной скоростью роста $O(n^2)$.

6 Функции со степенной скоростью роста $O(n^a)$ при a>2.

7 Функции с показательной или экспоненциальной скоростью роста $O(2^n)$.

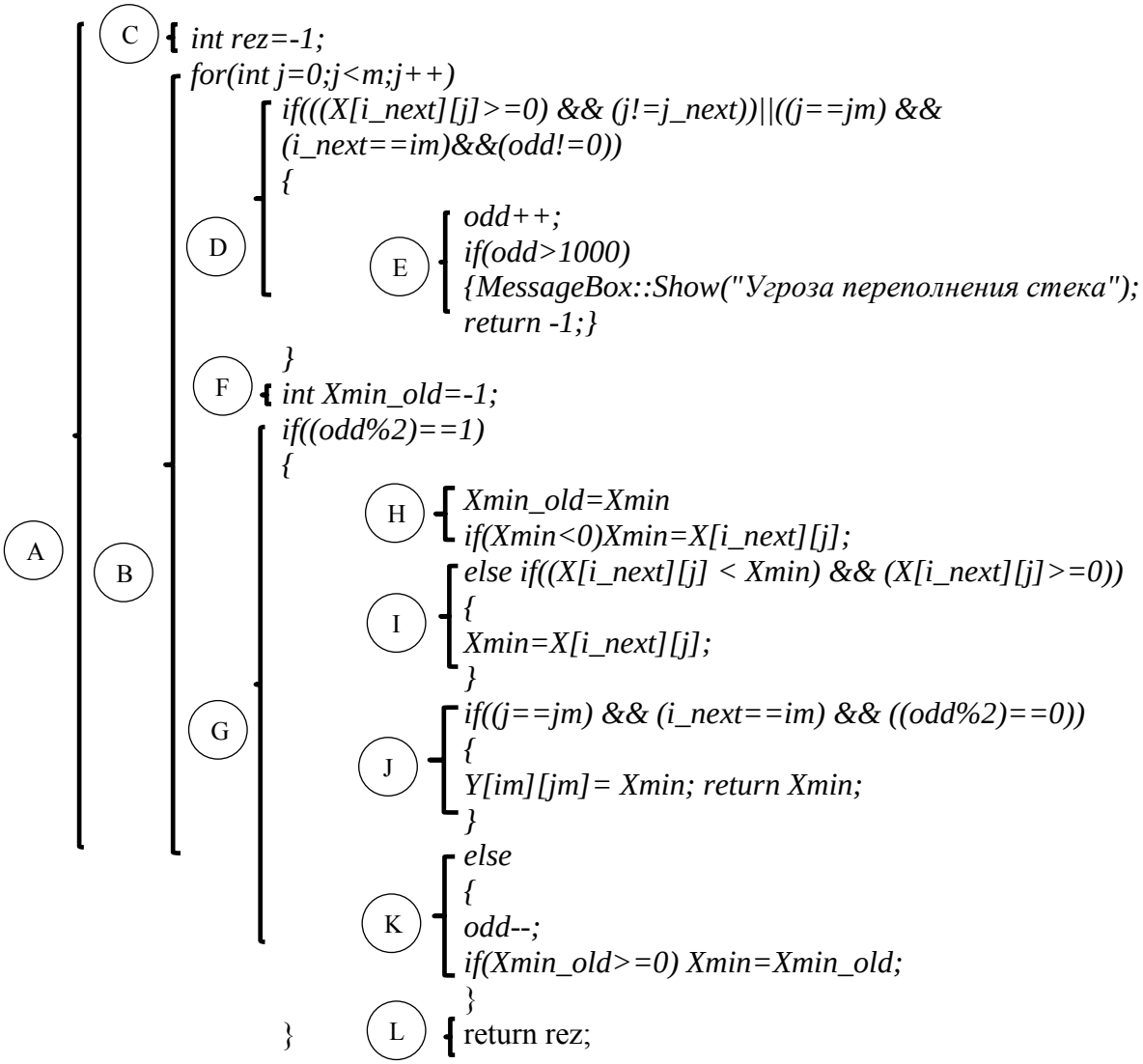
8 Функции с факториальной степенью роста $O(n!)$.

Базовое правило использования О большого

Базовое правило формулируется следующим образом: время выполнения цикла не превышает времени выполнения операторов, входящих в цикл (включая операторы условия), умноженное на число итераций цикла.

Время выполнения операторов внутри группы вложенных циклов равно времени выполнения операторов, умноженному на число итераций во всех циклах. Время выполнения последовательности циклов, следующих друг за другом, равно времени выполнения доминантного цикла. Первый случай описывается квадратичной функцией. Второй случай линеен.

Оценим сложность алгоритма нахождения замкнутой цепи:



O(E) = O(1) O(F) = O(1)
O(H) = O(1) O(I) = O(1)
O(J) = O(1) O(K) = O(1)
O(L) = O(1) O(D) = O(1)
O(F) = O(1) O(G) = O(1)
O(B) = O(N)*(O(D)+O(F)+O(G)) = O(N)
O(A) = O(C)+O(B)+O(L) = O(N). Сложность алгоритма равна O(N).

4 ЭТАПЫ ПРОВЕДЕНИЯ ЭКСПЕРИМЕНТА

Этап 1. Получение выборки статистических данных в ходе проведения эксперимента на ЭВМ.

Таблица 3–Результаты эксперимента

Номер наблюдения , i	Количество дуг, x_{2i}	Количество вершин, x_{1i}	Время выполнения программы, мс, y_i	x_{1i}^2	x_{2i}^2	$x_{1i} * x_{2i}$	$x_{1i} * y_i$	$x_{2i} * y_i$
1	1	2	30	1	4	2	30	60
2	2	3	30	4	9	6	60	90
3	3	4	100	9	16	12	300	400
4	4	6	1560	16	36	24	6240	9360
5	5	7	4070	25	49	35	20350	28490
6	6	8	4220	36	64	48	25320	33760
7	7	9	15091	49	81	63	105637	135819
Сумма	28	39	25101	140	259	190	157937	207979

В ходе проведения экспериментальной части курсовой работы получены исходные статистические данные (таблица 3).

Этап 2. Определение уравнения связи в общем виде

По результатам проведения эксперимента на ЭВМ строятся и анализируются точечные графики зависимости y от x_j, $j=\overline{1,n}$: $y=f(x_1)$ и $y=f(x_2)$

Анализ точечных графиков позволяет сделать вывод о линейной тенденции. Уравнение связи в общем виде имеет вид

$$\hat{y} = a_0 + a_1 * x_1 + a_2 * x_2, \quad (1)$$

где $\hat{y}$ – время выполнения программного алгоритма;

x_1 – количество вершин графа;

x_2 – количество дуг графа;

a_0 – свободный коэффициент;

a_1, a_2 – выборочные коэффициенты.

Этап 3. Оценивание коэффициентов уравнения связи

Оценивание коэффициентов a_0, a_1 и a_2 производится МНК по шагам.

Шаг 1. Подставим в математическое описание МНК вместо $\hat{y}_i$ правую часть уравнения связи в общем виде. Математическое описание МНК будет иметь вид

$$S_{кв} = \sum_{i=1}^{m} (y_i - a_0 - a_1 * x_1 - a_2 * x_2)^2 -> \min . \quad (2)$$

Шаг 2. Определим СНУ для оценивания коэффициентов a_0, a_1 и a_2 уравнения связи.

$$\begin{cases} \dfrac{\partial S_{кв}}{\partial a_0} = 0 \\ \dfrac{\partial S_{кв}}{\partial a_1} = 0 \\ \dfrac{\partial S_{кв}}{\partial a_2} = 0 \end{cases} \quad (3)$$

$$\begin{cases} \dfrac{\partial \sum_{i=1}^{m} (y_i - a_0 - a_1 * x_{1i} - a_2 * x_{2i})^2}{\partial a_0} = 0 \\ \dfrac{\partial \sum_{i=1}^{m} (y_i - a_0 - a_1 * x_{1i} - a_2 * x_{2i})^2}{\partial a_1} = 0 \\ \dfrac{\partial \sum_{i=1}^{m} (y_i - a_0 - a_1 * x_{1i} - a_2 * x_{2i})^2}{\partial a_2} = 0 \end{cases} \quad (4)$$

$$\begin{cases} -2 * \sum_{i=1}^{m} (y_i - a_0 - a_1 * x_{1i} - a_2 * x_{2i}) = 0 \\ -2 * \sum_{i=1}^{m} (y_i - a_0 - a_1 * x_{1i} - a_2 * x_{2i}) * x_{1i} = 0 \\ -2 * \sum_{i=1}^{m} (y_i - a_0 - a_1 * x_{1i} - a_2 * x_{2i}) * x_{2i} = 0 \end{cases} \quad (5)$$

Выполним преобразования: сократим на (-2) левую и правую части, раскроем скобки, сгруппируем слагаемые, заменим аддитивную операцию $\underbrace{a_0 + a_0 + ... a_0}_{m}$ на мультипликативную операцию $m * a_0$. СНУ имеет вид

$$\begin{cases} m * a_0 + a_1 * \sum_{i=1}^{m} x_{1i} + a_2 * \sum_{i=1}^{m} x_{2i} = \sum_{i=1}^{m} y_i \\ a_0 * \sum_{i=1}^{m} x_{1i} + a_1 * \sum_{i=1}^{m} x_{1i}^2 + a_2 * \sum_{i=1}^{m} x_{2i} * x_{1i} = \sum_{i=1}^{m} y_i * x_{1i} \\ a_0 * \sum_{i=1}^{m} x_{2i} + a_1 * \sum_{i=1}^{m} x_{1i} * x_{2i} + a_2 * \sum_{i=1}^{m} x_{2i}^2 = \sum_{i=1}^{m} y_i * x_{2i} \end{cases} \quad (6)$$

СНУ решается методом Гаусса.

Этап 4. Анализ уравнения связи и расчет доверительного интервала

Анализ уравнения связи включает расчет коэффициента множественной корреляции, совокупного коэффициента детерминации, частных коэффициентов корреляции, частных коэффициентов детерминации, частных коэффициентов эластичности и частных β-коэффициентов.

Коэффициент множественной корреляции рассчитывается по формуле

$$R^{\wedge}_{y x_1 x_2} = \sqrt{\frac{r^2_{yx_1} + r^2_{yx_2} - 2 * r_{yx_1} * r_{yx_2} * r_{x_1 x_2}}{1 - r^2_{x_1 x_2}}}, \quad (7)$$

где r_{yx_1} – парный коэффициент корреляции между y и x_1;

r_{yx_2} – парный коэффициент корреляции между y и x_2;

$r_{x_1 x_2}$ – парный коэффициент корреляции между x_1 и x_2.

Парный коэффициент корреляции между y и x_1 рассчитывается по формуле

$$r^{\wedge}_{yx_1} = \frac{m * \sum_{i=1}^{m} x_{1i} * y_i - \sum_{i=1}^{m} x_{1i} * \sum_{i=1}^{m} y_i}{\sqrt{(m * \sum_{i=1}^{m} x^2_{1i} - \sum_{i=1}^{m} x_{1i} * \sum_{i=1}^{m} x_{1i}) * (m * \sum_{i=1}^{m} y^2_i - \sum_{i=1}^{m} y_i * \sum_{i=1}^{m} y_i)}}. \quad (8)$$

Парный коэффициент корреляции между y и x_2 рассчитывается по формуле

$$r^{\wedge}_{yx_2} = \frac{m * \sum_{i=1}^{m} x_{2i} * y_i - \sum_{i=1}^{m} x_{2i} * \sum_{i=1}^{m} y_i}{\sqrt{(m * \sum_{i=1}^{m} x^2_{2i} - \sum_{i=1}^{m} x_{2i} * \sum_{i=1}^{m} x_{2i}) * (m * \sum_{i=1}^{m} y^2_i - \sum_{i=1}^{m} y_i * \sum_{i=1}^{m} y_i)}}. \quad (9)$$

Парный коэффициент корреляции между x_1 и x_2 рассчитывается по формуле

$$r_{x_1 x_2} = \frac{m * \sum_{i=1}^{m} x_{1i} * x_{2i} - \sum_{i=1}^{m} x_{1i} * \sum_{i=1}^{m} x_{2i}}{\sqrt{(m * \sum_{i=1}^{m} x^2_{2i} - \sum_{i=1}^{m} x_{2i} * \sum_{i=1}^{m} x_{2i}) * (m * \sum_{i=1}^{m} x^2_{1i} - \sum_{i=1}^{m} x_{1i} * \sum_{i=1}^{m} x_{1i})}}. \quad (10)$$

$R^2_{\wedge\, y x_1 x_2}$ - совокупный коэффициент детерминации.

Частные коэффициенты корреляции позволяют провести анализ тесноты связи между эндогенной переменной y и одной из экзогенных переменных при неизменных других.

Частный коэффициент корреляции между y и x_1 при неизменной x_2 рассчитывается по формуле

$$r^{\wedge}_{y x_1 (x_2)} = \frac{r_{yx_1} - r_{yx_2} * r_{x_1 x_2}}{\sqrt{(1 - r^2_{yx_2}) * (1 - r^2_{x_1 x_2})}}. \quad (11)$$

Частный коэффициент корреляции между y и x_2 при неизменной x_1 рассчитывается по формуле

$$r^{\wedge}_{y(x_1) x_2} = \frac{r_{yx_2} - r_{yx_1} * r_{x_1 x_2}}{\sqrt{(1 - r^2_{yx_1}) * (1 - r^2_{x_1 x_2})}}, \quad (12)$$

где $r^2_{\wedge\, y(x_1) x_2}$, $r^2_{\wedge\, y x_1 (x_2)}$ – частные коэффициенты детерминации.

Частные коэффициенты эластичности рассчитывают по формулам

$$Э^{\wedge}_{y x_1(x_2)} = \frac{a_1 * \overline{x_1}}{\overline{y}}, \quad (13)$$

где $\overline{x_1}$ – среднее арифметическое значение объема выборки X_1.

$$Э^{\wedge}_{y(x_1) x_2} = \frac{a_2 * \overline{x_2}}{\overline{y}}, \quad (14)$$

где $\overline{x_2}$ – среднее арифметическое значение объема выборки X_2.

Частные бэта-коэффициенты рассчитываются по формулам

$$\beta^{\wedge}_{y x_1(x_2)} = \frac{a_1 * S_{x_1}}{S_y}, \quad (15)$$

где a_1 – выборочный коэффициент;

S_{x_1} – средняя квадратическая ошибка (отклонение) объема выборки X_1;

S_y – средняя квадратическая ошибка (отклонение) объема выборки Y.

$$\beta^{\wedge}_{y(x_1) x_2} = \frac{a_1 * S_{x_2}}{S_y}, \quad (16)$$

где S_{x_2} – средняя квадратическая ошибка (отклонение) объема выборки X_2.

Средняя квадратическая ошибка объема выборки X_1 рассчитывается по формуле

$$S_{x_1} = \sqrt{\frac{\sum_{i=1}^{m} (x_{1i} - \overline{x_1})^2}{m}}. \quad (17)$$

Средняя квадратическая ошибка объема выборки X_2 рассчитывается по формуле

$$S_{x_2} = \sqrt{\frac{\sum_{i=1}^{m} (x_{2i} - \overline{x_2})^2}{m}}. \quad (18)$$

5 ОПИСАНИЕ СТРУКТУРЫ ПРОГРАММНОГО КОМПЛЕКСА

На рисунках 3-4 представлены скриншоты программы.

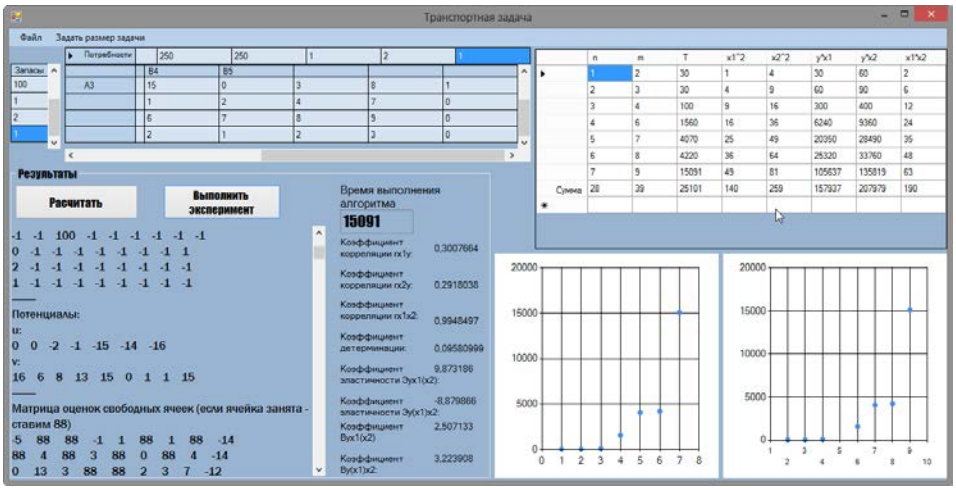


Рисунок 3–Интерфейс программы

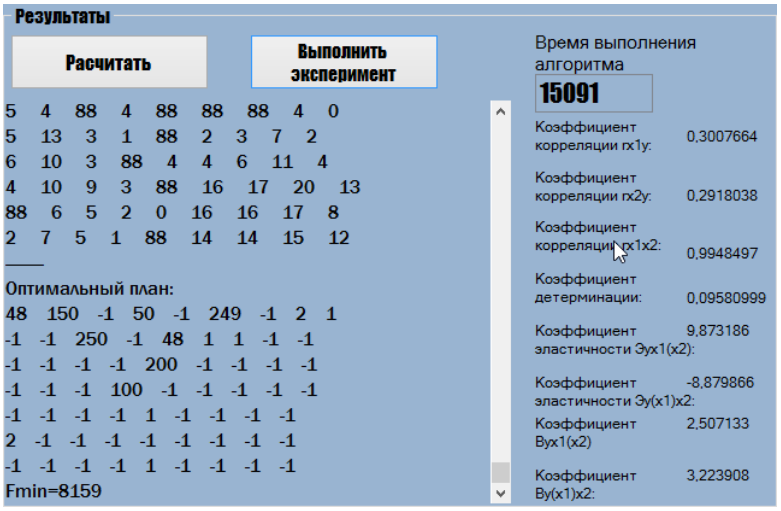


Рисунок 4 – Результат выполнения алгоритма и эксперимента

6 ОПИСАНИЕ ФУНКЦИЙ

Описания функций приведены в таблице 4.

Таблица 4–Описания функций

Описание метода	**Функция метода**
Void button2_Click()	Выполнение алгоритма решения задачи
float find_gor(int **i_next**,int **j_next**,int **im**,int **jm**,int **n**,int **m**,float** **X**,float** **Y**,int **odd**,float **Xmin**);	Поиск замкнутого цикла по строкам распределительной таблицы
float find_ver(int **i_next**,int **j_next**,int **im**,int **jm**,int **n**,int **m**,float** **X**,float** **Y**,int **odd**,float **Xmin**);	Поиск замкнутого цикла по столбцам распределительной таблицы
System::*Void* открытьToolStripMenuItem_Click(System::Object^ sender, System::*EventArgs*^ e)	Открытие задачи из файла

7 Выводы по результатам проектирования

В ходе курсового проектирования разработано визуальное объектно-ориентированное приложение C++, реализующее алгоритм решения транспортной задачи методом потенциалов. Проведен анализ оценки сложности алгоритма. Проведен эксперимент, устанавливающий время зависимости от количества элементов в массиве в виде уравнения связи и анализ полученных данных.

Использование данного программного обеспечения позволяет исследовать алгоритм решения транспортной задачи, определить время выполнения алгоритма в зависимости от входных значений.

В результате курсового проектирования повышены теоретические знания и приобретены практические навыки в разработке приложений, исследован и проанализирован алгоритм решения транспортной задачи методом потенциалов.

1.5 Спецификация

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГБОУ ВПО «КУРГАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Кафедра «Программное обеспечение автоматизированных систем»

Курсовая работа

По дисциплине «Алгоритмы и структуры данных»

СПЕЦИФИКАЦИЯ

РФ КГУ 231000.62.КР.201078 03

Разработал студент гр. Т-20012 _____ / Е.Б.Белинский /

Руководитель,

канд. техн. наук, доцент _____ /А.М.Семахин /

Проект защищен с оценкой ____ _____ «___» _____ 2013 г.

Члены комиссии

_____ / А.М.Семахин /

_____ / А.Г.Кокин /

Курган 2013

Обозначение	**Наименование**	**Примечание**
Компоненты		
	Исполняемый файл	Транспортная

	программы	задача_курс.exe
Документация		
РФ КГУ 231000.62.КР.201078 01	Опись альбома	Опись альбома.doc
РФ КГУ 231000.62.КР.201078 02	Пояснительная записка	Пояснительная записка.doc
РФ КГУ 231000.62.КР.201078 03	Спецификация	Спецификация.doc
РФ КГУ 231000.62.КР.201078 04	Описание программы	Описание программы.doc
РФ КГУ 231000.62.КР.201078 05	Руководство пользователя	Руководство пользователя.doc
РФ КГУ 231000.62.КР.201078 06	Руководство программиста	Руководство программиста.doc

1.6 Описание программы

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГБОУ ВПО «КУРГАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Кафедра «Программное обеспечение автоматизированных систем»

Курсовая работа

По дисциплине «Алгоритмы и структуры данных»

ОПИСАНИЕ ПРОГРАММЫ

Листов 7

РФ КГУ 231000.62.КР.201078 04

Разработал студент гр. Т-20012 _____ / Е.Б.Белинский /

Руководитель,

канд. техн. наук, доцент _____ /А.М.Семахин /

Проект защищен с оценкой ____ _____ «___» _____ 2013 г.

Члены комиссии

_____ / А.М.Семахин /

_____ / А.Г.Кокин /

Курган 2013

АННОТАЦИЯ

Документ содержит общие сведения о программе, ее функциональном назначении; описание структуры, используемых технических средств, входных и выходных данных.

СОДЕРЖАНИЕ

1 ОБЩИЕ СВЕДЕНИЯ

1.1 Программное обеспечение

Данная программа выполняется под управлением операционной системы Windows XP-8. Для запуска программы необходимо наличие исполняемого файла Транспортная задача_курс.exe.

1.2 Языки программирования

Для реализации программного продукта выбрана среда разработки MS Visual Studio 2012 и язык программирования C++.

2 ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ

Данное приложение формализует алгоритм решения транспортной задачи методом потенциалов. Приложение разработано с использованием объектно-ориентированного метода программирования. В программе обеспечен ввод исходных данных (размер задачи, распределительная таблица) и вывод данных в форме таблицы и графика.

3 ОПИСАНИЕ СТРУКТУРЫ

Программа основана на классе TransportTask. В нем реализованы следующие возможности программы: построение графика, нахождение оптимального решения транспортной задачи, расчет коэффициентов, проведение эксперимента, решение системы уравнений методом Гаусса.

4 ВХОДНЫЕ И ВЫХОДНЫЕ ДАННЫЕ

Исходными данными в программе являются:
1) размер задачи.;
2) запасы;
3) потребности;
4) тарифы.

Выходными данными в программе являются:
1) начальный план;
2) решение задачи на каждой итерации;
3) потенциалы;
4) оптимальный план;
5) график зависимости размерности задачи от времени выполнения;
6) коэффициенты корреляции, детерминации, эластичности, бэта-коэффициенты.

1.7 Руководство пользователя

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГБОУ ВПО «КУРГАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Кафедра «Программное обеспечение автоматизированных систем»

Курсовая работа
По дисциплине «Алгоритмы и структуры данных»

РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ
Листов 7

РФ КГУ 231000.62.КР.201078 05

Разработал студент гр. Т-20012 / Е.Б.Белинский /

Руководитель,
канд. техн. наук, доцент /А.М.Семахин /

Проект защищен с оценкой _____ «___» _____ 2013 г.

Члены комиссии

_____ / А.М.Семахин /

_____/ А.Г.Кокин /

Курган 2013

АННОТАЦИЯ

Документ предназначен для конечного пользователя и содержит сведения о том, каким образом необходимо работать с программным продуктом, обеспечивать его работоспособность.

СОДЕРЖАНИЕ

1 НАЗНАЧЕНИЕ ПРОГРАММЫ

Данное приложение формализует алгоритм решения транспортной задачи методом потенциалов. Приложение разработано с использованием объектно-ориентированного метода программирования. В программе обеспечен ввод исходных данных (размер задачи, распределительная таблица) и вывод данных в форме таблицы и графика.

1.2 УСЛОВИЯ ВЫПОЛНЕНИЯ ПРОГРАММЫ

2.1 Программное обеспечение

- ОС Windows версии XP, VISTA, 7, 8;
- .NET Framework 4.5.

2.2 Требования к аппаратуре

- компьютер IBM PC или другой совместимый с операционной системой Windows.

2.3 ВЫПОЛНЕНИЕ ПРОГРАММЫ

3.1 Запуск программы

Для запуска программы необходим файл ТранспортнаяЗадача_курс.exe.

3.2 Работа с программой

При запуске программы появляется главное окно программы.

Необходимо заполнить распределительную таблицу или открыть ее из текстового файла. После этого нужно нажать кнопку «Решить». В поле, которое находится под таблицей, будет помещено решение задачи. Главное окно приведено на рисунке 1.

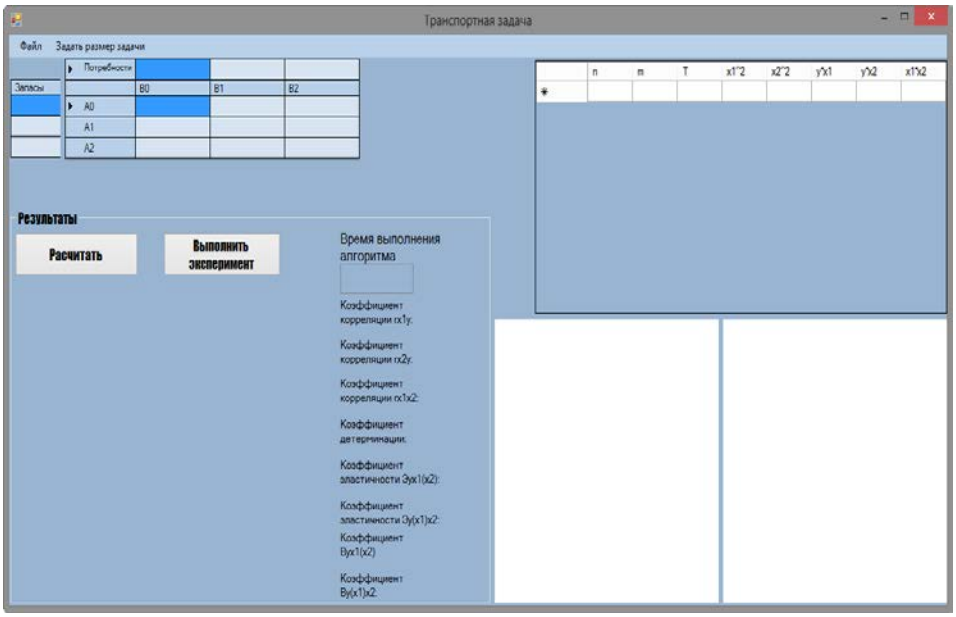


Рисунок 1 – Главное окно программы

Чтобы провести эксперимент нужно выполнить алгоритм несколько раз. При этом будут заноситься результаты выполнения алгоритма в таблицу в правой части окна. После того, как эти действия произведены нужно нажать кнопку «Выполнить эксперимент». В результате будут отображены результаты эксперимента(график, коэффициенты). Результат работы программы приведен на рисунке 2.

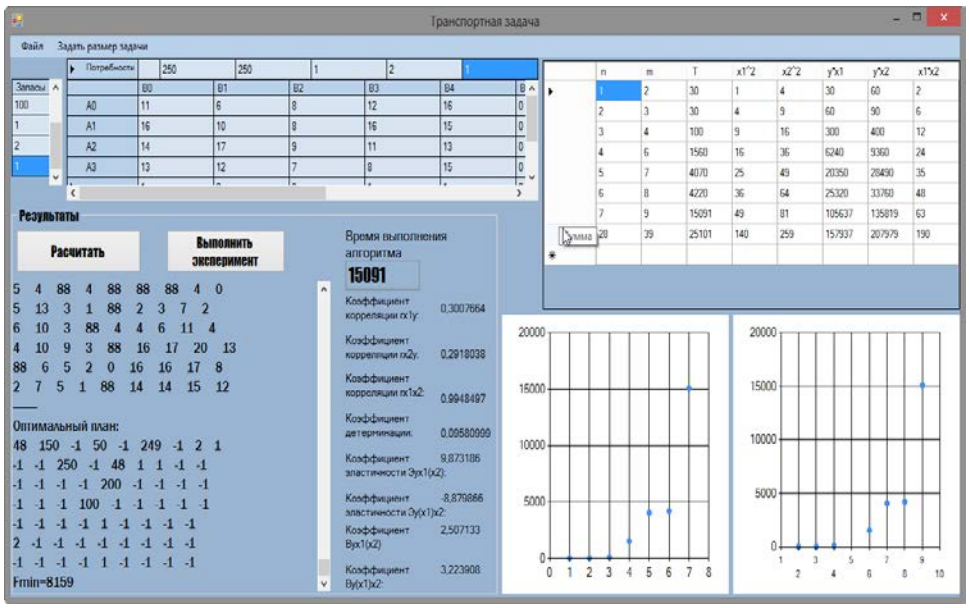


Рисунок 2 – Результат работы программы

1.8 Руководство программиста

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГБОУ ВПО «КУРГАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Кафедра «Программное обеспечение автоматизированных систем»

Курсовая работа

По дисциплине «Алгоритмы и структуры данных»

РУКОВОДСТВО ПРОГРАММИСТА

Листов 27

РФ КГУ 231000.62.КР.201078 06

Разработал студент гр. Т-20012 _____ / Е.Б.Белинский /

Руководитель,

канд. техн. наук, доцент _____ /А.М.Семахин /

Проект защищен с оценкой ____ _____ «___» _____ 2013 г.

Члены комиссии

_____ / А.М.Семахин /

_____ / А.Г.Кокин /

Курган 2013

АННОТАЦИЯ

Документ содержит общие сведения о программе, функциональное назначение, описание логической структуры, используемых технических средств, входных и выходных данных.

Предназначен для программистов.

СОДЕРЖАНИЕ

ОБЩИЕ СВЕДЕНИЯ

Программа, моделирующая работу алгоритма сортировки внешних файлов методом естественного слияния, позволяет провести серию сортировок и на основе данных полученных в результате каждой сортировки сделать анализ всего алгоритма.

1.1 Программное обеспечение

Данная программа выполняется под управлением операционной системы Windows XP-8. Для запуска программы необходимо наличие исполняемого файла ТранспортнаяЗадача_курс.exe.

1.2 Языки программирования

Для реализации программного продукта выбрана среда разработки MS Visual Studio 2012 и язык программирования C++.

ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ

Данное приложение формализует алгоритм решения транспортной задачи методом потенциалов. Приложение разработано с использованием объектно-ориентированного метода программирования. В программе обеспечен ввод исходных данных (размер задачи, распределительная таблица) и вывод данных в форме таблицы и графика.

3 ОПИСАНИЕ ЛОГИЧЕСКОЙ СТРУКТУРЫ

Программа основана на классе TransportTask. В нем реализованы следующие возможности программы: построение графика, нахождение оптимального решения транспортной задачи, расчет коэффициентов, проведение эксперимента, решение системы уравнений методом Гаусса.

4 ВХОДНЫЕ И ВЫХОДНЫЕ ДАННЫЕ

Исходными данными в программе являются:
1) размер задачи;
2) запасы;
3) потребности;
4) тарифы.
Выходными данными в программе являются:
1) начальный план;
2) решение задачи на каждой итерации;
3) потенциалы;
4) оптимальный план;
5) график зависимости размерности задачи от времени выполнения;
6) коэффициенты корреляции, детерминации, эластичности, бэта-коэффициенты.

5 ЛИСТИНГ АЛГОРИТМА

```
int N,M,Mm,Nn;//Размерность задачи
        float *a;  //адрес массива запасов поставщиков
        float *b;  //адрес массива потребностей потребителей
        float **C; //адрес массива(двумерного) стоимости перевозки
        float **X;//адрес массива(двумерного) плана доставки
        float *u;//адрес массива потенциалов поставщиков
        float *v;//адрес массива потенциалов потребителей
        float **D;//адрес массива(двумерного) оценок свободных ячеек таблицы
        bool stop;//признак достижения оптимального плана
        bool **T;//массив будет хранить коодинаты ячеек, в коорые уже вписывались
        //нулевые поставки при попытках устранить вырожденность плана
        bool ok;        array<double,2>^A; // массив коэффициентов системы
уравнений
        array<double>^L;   // массив свободных членов системы уравнений
// эта функция осуществляет решение задачи
private: System::Void button2_Click(System::Object^ sender, System::EventArgs^ e) {
richTextBox1->Clear(); int n= dataGridView1->RowCount;
int m= dataGridView1->ColumnCount; int Sa=0; int Sb=0; int time1 = clock();
try {  for(int i=0; i<n; i++)  Sa = Sa + Convert::ToInt32(dataGridView2[0,i]->Value);
 for(int i=0; i<m; i++) Sb = Sb + Convert::ToInt32(dataGridView3[i,0]->Value);
if(Sa == Sb) richTextBox1->Text += "Транспортная задача - закрытая\n\n";
```

```
else {  richTextBox1->Text += "Транспортная задача - открытая\n\n";
 if(Sa>Sb) {    / MessageBox::Show("Преобразуем в закрытую добавлением
фиктивного потребителя");
richTextBox1->Text += "Преобразуем в закрытую путем добавлением фиктивного
потребителя\n\n";
m++; dataGridView1->ColumnCount = m;
dataGridView1->Columns[m-1]->HeaderText = "B"+Convert::ToString(m-1);
for(int i=0;i<n;i++)
dataGridView1[m-1,i]->Value = 0;//стоимость перевозки фиктивному потребителю
равна нулю
dataGridView3->ColumnCount++;
dataGridView3[m-1,0]->Value = Sa-Sb;//спрос фиктивного потребителя
} else { //MessageBox::Show("Преобразуем закрытую задачу добавлением фиктивного
поставщика
 n++; dataGridView1->RowCount = n; for(int i=0;i<m;i++)
dataGridView1[i,n-1]->Value = 0;//стоимость перевозки от фиктивного поставщика
равна нулю
dataGridView2->RowCount++; dataGridView2[0,n-1]->Value = Sb-Sa;//запас
фиктивного поставщика
 }   } } catch(...)
 {  MessageBox::Show("Ошибка ввода данных\n Проверьте правильность заполнения
таблиц");  }
//-----Инициализация динамических массивов: /запоминаем размерность в глобальных
переменных:
N=n;/*кол-во строк (поставщиков)*/      M=m;//кол-во столбцов
(потребителей)
//выделение памяти под динамические массивы и их заполнение:
a=new float[N]; //массив для Запасов
for(int i=0;i<N;i++)      a[i]=Convert::ToInt32(dataGridView2[0,i]->Value);
b=new float[M];//Массив для Потребностей
for(int i=0;i<M;i++)      b[i]=Convert::ToInt32(dataGridView3[i,0]->Value);
//Двумерный массив для Стоимости:
C=new float*[N];//выделяем память под массив адресов начала строк
for(int i=0;i<N;i++)      C[i]=new float[M];//выделяем память для каждой
строки
for(int i=0;i<N;i++)   for(int j=0;j<M;j++) //Заполняем массив значениями из
таблицы на форме:
 C[i][j]=Convert::ToInt32(dataGridView1[j,i]->Value);   //Двумерный массив для
Доставки:
X=new float*[N+1];//выделяем память под массив адресов начала строк
for(int i=0;i<N+1;i++)      X[i]=new float[M+1];//выделяем память для
каждой строки
/*      В последней строке(столбце) массива X будем записывать
сумму заполненных клеток в соответствующем столбце(строке)      */
for(int i=0;i<N+1;i++)  for(int j=0;j<M+1;j++) { X[i][j]=-1;//вначале все клетки
не заполнены
  if(i==N)X[i][j]=0;//сумма заполненных клеток в j-м столбце
  if(j==M)X[i][j]=0;//сумма заполненных клеток в i-й строке      }
 //-----------------Метод минимального элемента:
 float Sij=0;  do { int im;      int jm;      int Cmin=-1;  for(int i=0;i<N;i++)   for(int
j=0;j<M;j++)
  if(X[N][j]!=b[j])//если не исчерпана Потребность Bj
```

```
                    if(X[i][M]!=a[i])//если не исчерпан Запас Ai
                    if(X[i][j]<0)//если клетка ещё не заполнена
                        {  if(Cmin==-1)//если это первая подходящая ячейка
                    {  Cmin=C[i][j];  im=i;  jm=j; }  else //если это не первая
подходящая ячейка
        if(C[i][j]<Cmin)//если в ячейке меньше,чем уже найдено
        { Cmin=C[i][j];  im=i;  jm=j; } }  X[im][jm]=min(a[im]-X[im][M],b[jm]-
X[N][jm]);//выбираем поставку
       X[N][jm]=X[N][jm]+X[im][jm];//добавляем поставку jm-му потребителю
       X[im][M]=X[im][M]+X[im][jm];//добавляем поставку im-му поставщику
       Sij=Sij+X[im][jm]; //Подсчёт суммы добавленых поставок
       }while(Sij<max(Sa,Sb));//условие продолжения
       int L=0;  for(int i=0;i<N;i++)  for(int j=0;j<M;j++)  if(X[i][j]>=0)L++;//подсчёт
заполненных ячеек
        int d=M+N-1-L;//если d>0,то задача - вырожденная,придётся добавлять d нулевых
поставок
        int d1=d; /*запоминаем значение d */  richTextBox1->Text += "Начальный опорный
план:\n";
        int F=0;  for(int i=0;i<N;i++)  for(int j=0;j<M;j++)
        {  richTextBox1->Text=richTextBox1->Text+Convert::ToString(X[i][j])+"   ";
        if(X[i][j]>0)F=F+X[i][j]*C[i][j];  if(j==M-1)richTextBox1->Text += "\n";  }
        richTextBox1->Text +="--------";  richTextBox1->Text +="F="+Convert::ToString(F);
              // ~~~~~~~~~~~ метод потенциалов ~~~~~~~~~~~~~
        T=new bool*[N];  for(int i=0;i<N;i++)  T[i]=new bool[M];  for(int i=0;i<N;i++)
for(int j=0;j<M;j++)
        T[i][j]=false;  do{//цикл поиска оптимального решения
        stop=true;//признак оптимальности плана(после проверки может стать false)
        u=new float[N];//выделение массивов под значения потециалов
        v=new float[M];  bool *ub=new bool[N];//вспомогательные массвы
        bool *vb=new bool[M];  for(int i=0;i<N;i++)ub[i]=false;
       for(int i=0;i<M;i++)vb[i]=false;  //цикл расчёта потенциалов (несколько избыточен):
       u[0]=0;//значение одного потенциала выбираем произвольно
       ub[0]=true;  int count=1;  int tmp=0;  do{  for(int i=0;i<N;i++)  if(ub[i]==true)  for(int
j=0;j<M;j++)
       if(X[i][j]>=0)  if(vb[j]==false)  {   v[j]=C[i][j]-u[i];   vb[j]=true;   count++;  } for(int
j=0;j<M;j++)
       if(vb[j]==true)  for(int i=0;i<N;i++)  if(X[i][j]>=0)  if(ub[i]==false)  {  u[i]=C[i][j]-
v[j];  ub[i]=true;  count++;  }  tmp++;  }while((count<(M+N-d*2))&&(tmp<M*N));
richTextBox1->Text += "--------\n";
       //цикл добавления нулевых поставок(в случае вырожденности):
        bool t=false;  if((d>0)||ok==false)t=true;//цикл начинается, если d>0
        while(t)//цикл продолжается до тех пор, пока все потенциалы не будут найдены
        {  for(int i=0;(i<N);i++)//просматриваем потенциалы поставщиков
        if(ub[i]==false)//если среди них не заполненный потенциал
        for(int j=0;(j<M);j++)  if(vb[j]==true)  {  if(d>0)  if(T[i][j]==false)
       //если раньше не пытались использовать
        {  X[i][j]=0;//добавляем нулевую поставку
        d--;//уменьшаем кол-во требуемых добавлений нулевых поставок
       T[i][j]=true;//отмечаем, что эту ячейку уже использовали
       } if(X[i][j]>=0) {  u[i]=C[i][j]-v[j];//дозаполняем потенциалы
        ub[i]=true; } } for(int j=0;(j<M);j++)//просматриваем потенциалы потребителей
       if(vb[j]==false)//если среди них не заполненный потенциал
```

```
for(int i=0;(i<N);i++) if(ub[i]==true) {  if(d>0)  if(T[i][j]==false)//если не пытались использовать
{  X[i][j]=0;//добавляем нулевую поставку
d--;//уменьшаем кол-во требуемых добавлений нулевых поставок
T[i][j]=true;//отмечаем, что эту ячейку уже использовали
} if(X[i][j]>=0) {  v[j]=C[i][j]-u[i];//дозаполняем потенциалы
vb[j]=true; } }  t=false; //проверяем, все ли потенциалы найдены
for(int i=0;i<N;i++)   if(ub[i]==false)t=true; for(int j=0;j<M;j++) if(vb[j]==false)t=true;
if(t==false) {   richTextBox1->Text += "Опорный план после устранения вырожденности: \n";
for(int i=0;i<N;i++)   for(int j=0;j<M;j++) { richTextBox1->Text += Convert::ToString(X[i][j])+"    ";
if(j==M-1)richTextBox1->Text += "\n"; } richTextBox1->Text += "--------\n"; } }
richTextBox1->Text += "Потенциалы:\n";  richTextBox1->Text += "u: \n";   for(int i=0;i<N;i++)
richTextBox1->Text += Convert::ToString(u[i])+"    ";   richTextBox1->Text += "\nv: \n";
for(int i=0;i<M;i++)   richTextBox1->Text += Convert::ToString(v[i])+"    ";
richTextBox1->Text += "\n--------\n";
D=new float*[N];//выделяем память под массив оценок свободных ячеек
for(int i=0;i<N;i++)  D[i]=new float[M];  for(int i=0;i<N;i++)  for(int j=0;j<M;j++)
{   if(X[i][j]>=0)//если ячейка не свободна
D[i][j]=88;//Заполняем любыми положительными числами
else  /* если ячейка свободна */   D[i][j]=C[i][j]-u[i]-v[j];//находим оценку
if(D[i][j]<0) {   stop=false;//признак того, что план не оптимальный
} } richTextBox1->Text += "Матрица оценок свободных ячеек (если ячейка занята - ставим 88)\n";
for(int i=0;i<N;i++)  for(int j=0;j<M;j++)  {  richTextBox1->Text += Convert::ToString(D[i][j])+"     ";
if(j==M-1)richTextBox1->Text += "\n";  }
richTextBox1->Text += "--------\n";  if(stop==false)//если план не оптимальный
{  float **Y=new float*[N];//массив для хранения цикла перераспределения поставок
for(int i=0;i<N;i++)  Y[i]=new float[M];
float find1,find2;//величина перераспределения поставки для цикла
float best1=0;//наилучшая оценка улучшения среди всех допустимых перераспределений
float best2=0;  int ib1=-1;  int jb1=-1; int ib2=-1; int jb2=-1;
//Ищем наилучший цикл перераспределения поставок:
for(int i=0;i<N;i++)  for(int j=0;j<M;j++)
if(D[i][j]<0)//Идём по ВСЕМ ячейкам с отрицательной оценкой
{  //и ищем допустимые циклы перераспределения ДЛЯ КАЖДОЙ такой ячейки.
/Обнуляем матрицу Y:
for(int i=0;i<N;i++) for(int j=0;j<M;j++) Y[i][j]=0; //Ищем цикл для ячейки с оценкой D[i][j]:
find1=find_gor(i,j,i,j,N,M,X,Y,0,-1);//Начинаем идти по горизонтали. Обнуляем матрицу Y:
for(int i=0;i<N;i++)  for(int j=0;j<M;j++)  Y[i][j]=0;
find2=find_ver(i,j,i,j,N,M,X,Y,0,-1);//Начинаем по вертикали
if(find1>0) if(best1>D[i][j]*find1) {  best1=D[i][j]*find1;//наилучшая оценка
ib1=i;//запомминаем координаты ячейки
jb1=j;//цикл из которой даёт наибольшее улучшение
```

```
        } if(find2>0) if(best2>D[i][j]*find2) {  best2=D[i][j]*find2;//наилучшая оценка
        ib2=i;//запомминаем координаты ячейки
        jb2=j;//цикл из которой даёт наибольшее улучшение
        } } if((best1==0)&&(best2==0)) {   ok=false;  d=d1;//откат назад
        for(int i=0;i<N;i++)  for(int j=0;j<M;j++)  if(X[i][j]==0)X[i][j]=-1;  continue;
        } else {   /*Обнуляем матрицу Y:*/  for(int i=0;i<N;i++) for(int j=0;j<M;j++)
Y[i][j]=0;
        //возвращаемся к вычислению цикла с наилучшим перераспределением:
        int ib,jb; if(best1<best2) {   find_gor(ib1,jb1,ib1,jb1,N,M,X,Y,0,-1); ib=ib1; jb=jb1; } else {
        find_ver(ib2,jb2,ib2,jb2,N,M,X,Y,0,-1);  ib=ib2;  jb=jb2; } for(int i=0;i<N;i++) { for(int
j=0;j<M;j++)
        { if( (X[i][j]==0)&&(Y[i][j]<0) ) {  stop=true;  ok=false; d=d1;//откат назад
        richTextBox1->Text += "\nПопытка отрицательной поставки!\n";
        } X[i][j]= X[i][j]+Y[i][j];//перераспределяем поставки
        if((i==ib)&&(j==jb))X[i][j]=X[i][j]+1;//добавляем 1 (т.к. до этого было -1 )
        if((Y[i][j]<=0)&&(X[i][j]==0))X[i][j]=-1;//если ячейка обнулилась, то выбрасываем её
из рассмотрения
        } } } richTextBox1->Text += "Матрица цикла перерасчёта:\n";
        for(int i=0;i<N;i++)  for(int j=0;j<M;j++) {  richTextBox1->Text +=
Convert::ToString(Y[i][j])+"    ";
        if(j==M-1)richTextBox1->Text += "\n"; }  richTextBox1->Text += "\n--------\n";
        richTextBox1->Text += "Новый план:\n"; float F=0; for(int i=0;i<N;i++)  for(int
j=0;j<M;j++)
        { richTextBox1->Text += Convert::ToString(X[i][j])+"    ";
if(X[i][j]>0)F=F+X[i][j]*C[i][j];
        if(j==M-1)richTextBox1->Text += "\n"; } richTextBox1->Text +=
"F="+Convert::ToString(F);
        richTextBox1->Text += "\n--------\n"; ok=true; for(int i=0;i<N;i++) for(int j=0;j<M;j++)
T[i][j]=false;
        delete []Y; //проверка на вырожденность: (?)
        L=0; for(int i=0;i<N;i++)  for(int j=0;j<M;j++)  if(X[i][j]>=0)L++;//подсчёт
заполненных ячеек
        d=M+N-1-L;//если d>0,то задача - вырожденная
        d1=d; if(d>0)ok=false; }  delete []u;  delete []v;  delete []ub; delete []vb; delete []D;
}while(stop==false);
        richTextBox1->Text += "Оптимальный план:\n"; float Fmin=0; for(int i=0;i<N;i++)
for(int j=0;j<M;j++)
        { richTextBox1->Text += Convert::ToString(X[i][j])+"    ";
if(X[i][j]>0)Fmin=Fmin+X[i][j]*C[i][j];
        if(j==M-1)richTextBox1->Text += "\n"; } richTextBox1->Text +=
"Fmin="+Convert::ToString(Fmin);
        float timenew = clock() - time1;  delete []X;  delete []C; delete []a; delete []b; listView1-
>Items->Clear();
        listView1->Items->Add(System::Convert::ToString(timenew));
        dataGridView4->Rows->Add(dataGridView1->RowCount,dataGridView1->ColumnCount ,
        timenew,Math::Pow(N,2),Math::Pow(M,2),N*timenew,M*timenew, M*N);                }
        //Функуция поиска ячеек,подлежащих циклу перераспределения (вдоль строки)
        float find_gor(int i_next,int j_next,int im,int jm,int n,int m,float** X,float** Y,int odd,float
Xmin)
        {  int rez=-1;  for(int j=0;j<m;j++)//идём вдоль строки, на которой стоим
        //ищем заполненную ячейку(кроме той,где стоим) или начальная ячейка(но уже в
конце цикла:odd!=0 )
```

```
if(((X[i_next][j]>=0)&&(j!=j_next))||((j==jm)&&(i_next==im)&&(odd!=0)))
{  odd++;//номер ячейки в цикле перерасчёта(начало с нуля)
 if(odd>1000) {  MessageBox::Show("Угроза переполнения стека");  return -1;  } int
Xmin_old=-1;
if((odd%2)==1)//если ячейка нечётная в цикле ( начальная- нулевая ) {
Xmin_old=Xmin;//Запоминаем значение минимальной поставки в цикле (на случай
отката назад)
if(Xmin<0)Xmin=X[i_next][j];//если это первая встреченная ненулевая ячейка
 else if((X[i_next][j]<Xmin)&&(X[i_next][j]>=0)) {  Xmin=X[i_next][j];
} } if((j==jm)&&(i_next==im)&&((odd%2)==0))//если замкнулся круг и цикл имеет
чётное число ячеек
{  Y[im][jm]= Xmin;//Значение минимальной поставки, на величину которой будем
изменять
return Xmin; } //если круг еще не замкнулся - переходим к поиску по вертикали:
else rez=find_ver(i_next,j,im,jm,n,m,X,Y,odd,Xmin);//рекурсивный вызов
if(rez>=0)//как бы обратный ход рекурсии(в случае если круг замкнулся)
{ //для каждой ячейки цикла заполняем матрицу перерасчёта поставок:
 if(odd%2==0)Y[i_next][j]=Y[im][jm];//в чётных узлах прибавляем
 else  Y[i_next][j]=-Y[im][jm];//в нечётных узлах вычитаем
break; } else //откат назад в случае неудачи(круг не замкнулся):
{ odd--; if(Xmin_old>=0)//если мы изменяли Xmin на этой итерации
Xmin=Xmin_old; } } return rez;//если круг замкнулся (вернулись в исходную за чётное
число шагов) -
// возвращает найденное минимальное значение поставки в нечётных ячейках цикла,
// если круг не замкнулся, то возвращает -1.
} //Функуция поиска ячеек,подлежащих циклу перераспределения (вдоль столбца)
float find_ver(int i_next,int j_next,int im,int jm,int n,int m,float** X,float** Y,int odd,float
Xmin)
 {  int rez=-1;  int i;  for(i=0;i<n;i++)//идём вдоль столбца, на котором стоим
//ищем заполненную ячейку(кроме той,где стоим) или начальная ячейка(но уже в
конце цикла:odd!=0 )
if(((X[i][j_next]>=0))&&(i!=i_next)||((j_next==jm)&&(i==im)&&(odd!=0)))
{  odd++;//номер ячейки в цикле перерасчёта(начало с нуля)
 if(odd>1000)  {        MessageBox::Show("Угроза переполнения стека");        return -1;
}
 int Xmin_old=-1; if((odd%2)==1)//если ячейка нечётная в цикле ( начальная- нулевая )
{
 Xmin_old=Xmin;//Запоминаем значение минимальной поставки в цикле (на случай
отката назад)
if(Xmin<0)Xmin=X[i][j_next];//если это первая встреченная ненулевая ячейка
else if((X[i][j_next]<Xmin)&&(X[i][j_next]>=0))  Xmin=X[i][j_next]; }
if((i==im)&&(j_next==jm)&&((odd%2)==0))//если замкнулся круг и цикл имеет
чётное число ячеек
{ Y[im][jm]= Xmin;//Значение минимальной поставки, на величину которой будем
изменять
return Xmin; } //если круг еще не замкнулся - переходим к поиску по горизонтали:
else rez=find_gor(i,j_next,im,jm,n,m,X,Y,odd,Xmin);//- рекурсивный вызов
if(rez>=0)//как бы обратный ход (в случае если круг замкнулся)
{ //для каждой ячейки цикла заполняем матрицу перерасчёта поставок:
 if(odd%2==0)Y[i][j_next]=Y[im][jm];//эти прибавляются
else  Y[i][j_next]=-Y[im][jm];//эти вычитаются
 break; } else //откат назад в случае неудачи (круг не замкнулся):
```

*{ **odd**--; if(**Xmin_old**>=0)//если мы изменяли Xmin на этой итерации*
***Xmin**=**Xmin_old**; } } return **rez**; }*

Семахин Андрей Михайлович

АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ

Методические указания
к выполнению курсовых работ
для студентов направления 231000.62

Редактор Е. А. Могутова

Подписано в печать19.06.14	Формат 60x84 1/16	Бумага 65 г/м^2
Печать цифровая	Усл. печ. л. 4,75	Уч.-изд. л. 4,75
Заказ 198	Тираж 30	Не для продажи

РИЦ Курганского государственного университета.
640000, г. Курган, ул. Советская, 63/4.
Курганский государственный университет.