МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
“СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ”

ТЕОРИЯ АЛГОРИТМОВ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
к выполнению лабораторных работ по дисциплине
для студентов направлений подготовки
09.03.02 – «Информационные системы и технологии» и
09.03.03 – «Прикладная информатика»
всех форм обучения

Севастополь
2023

УДК 519.85:004.421(076.5)
ББК 22.12я73
Т338

Рецензенты:
Чернега В. С., кандидат техн. наук, доцент кафедры Информационных систем:
Козлова Е. В., кандидат техн. наук, доцент кафедры Информационных технологий и компьютерных систем.
Составители: Е. Н. Заикина, В. Ю. Карлусов.

Т338 Теория алгоритмов : методические указания к выполнению лабораторных работ по дисциплине для студентов направлений подготовки 09.03.02 – «Информационные системы и технологии» и 09.03.03 – «Прикладная информатика» всех форм обучения / Севастопольский государственный университет ; сост.: Е. Н. Заикина, В. Ю. Карлусов. – Севастополь : СевГУ, 2023. – 63 с.

Цель методических указаний: обеспечить студентов дидактическим материалом для вырабатывания практических навыков и освоения методов анализа алгоритмов с различными классами зависимости функций трудоёмкости от входных данных.

УДК 519.85:004.421(076.5)
ББК 22.12я73

Методические указания рассмотрены и утверждены на заседании кафедры Информационных систем, протокол № *04* от *17 ноября* 20*22* г.

Методическое пособие рассмотрено и рекомендовано к изданию на заседании Учёного Совета Института информационных технологий, протокол № *03* от *27 декабря* 20*22* г. года.

Ответственный за выпуск: заведующий кафедрой Информационных систем, канд. физ.-мат. наук, доцент И. П. Шумейко

Издательский номер №№ /23

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Теория алгоритмов, бесспорно, является фундаментальной научной дисциплиной, лежащей в основании принципов современного устройства средств компьютерной техники, организации вычислительных и информационных процессов.

Однако, несмотря на теоретическое изящество и мощь логических построений, лишь отдельные аспекты дисциплины могут быть в первозданном, неадаптированном виде использованы в повседневной деятельности программиста-практика, являясь, по сути, предметом его компетенций.

Из этих соображений в настоящее методическое пособие включены разделы, касающиеся методов и приёмов анализа вычислительной сложности в приложении к обще употребляемым и фундаментальным алгоритмам.

Освоение указанных навыков может принести несомненную пользу в производственной деятельности на ниве программирования.

Лабораторные работы основываются на имитационном моделировании и проведении вычислительных экспериментов, поэтому качестве лабораторной установки для исследований используется персональный компьютер с соответствующим программным обеспечением.

ЛАБОРАТОРНАЯ РАБОТА № 1.
“ИССЛЕДОВАНИЕ КОЛИЧЕСТВЕННО-ЗАВИСИМЫХ АЛГОРИТМОВ МЕТОДАМИ АСИМПТОТИЧЕСКОГО АНАЛИЗА”

1. ЦЕЛЬ РАБОТЫ

- Исследовать поведение функций трудоемкости количественно-зависимых алгоритмов в реальных интервалах значений мощности множества исходных данных с использованием аппарата интервального анализа.
- На основании этих исследований сделать предпочтительный выбор того или иного алгоритма из двух предложенных.
- Приобретение практических навыков проведения асимптотического анализа.

2. КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

2.1. Основные определения асимптотического анализа алгоритмов

При использовании алгоритмов для решения практических задач сталкиваются с проблемой рационального выбора алгоритма решения задачи. Решение проблемы выбора связано с построением системы сравнительных

оценок, которая в свою очередь существенно опирается на формальную модель алгоритма.

Под *трудоёмкостью* алгоритма для данного конкретного входа – $F_a(N)$, понимают количество «элементарных» операций, совершаемых алгоритмом для решения конкретной проблемы в данной формальной системе. При анализе поведения функции трудоемкости алгоритма часто используют принятые в математике асимптотические обозначения, позволяющие показать *скорость роста функции*, маскируя при этом конкретные коэффициенты.

Такая оценка функции трудоемкости алгоритма называется *сложностью алгоритма* и позволяет определить предпочтения в использовании того или иного алгоритма для больших значений размерности исходных данных.

В асимптотическом анализе приняты следующие обозначения:

1. Оценка Θ (тэта)

Пусть $f(n)$ и $g(n)$ – положительные функции положительного аргумента, $n \geq 1$ (количество объектов на входе и количество операций – положительные числа), тогда: $f(n) = \Theta(g(n))$, если существуют положительные c_1, c_2, n_0, такие, что: $c_1 * g(n) \leq f(n) \leq c_2 * g(n)$, *при* $n > n_0$.

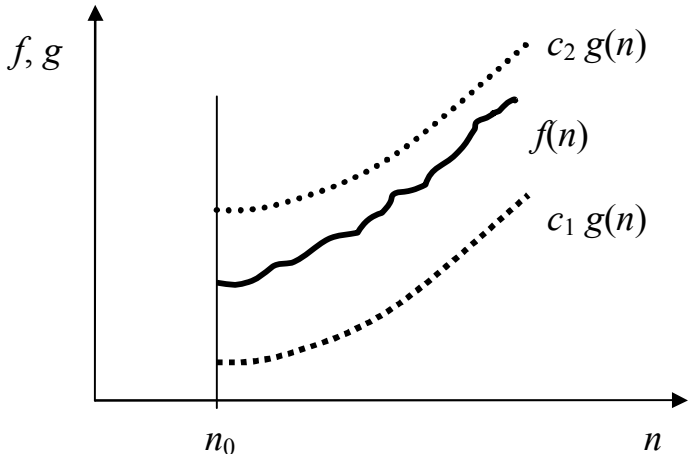


Рисунок 1 – Оценка Θ

Обычно говорят, что при этом функция $g(n)$ является асимптотически точной оценкой функции $f(n)$, т.к. по определению функция $f(n)$ не отличается от функции $g(n)$ с точностью до постоянного множителя.

Указанная оценка симметрична: $f(n) = \Theta(g(n)) \Leftrightarrow g(n) = \Theta(f(n))$.

Примеры:

1) $f(n)=4n^2+n\ln N+174$ – $f(n)= \Theta(n^2)$;

2) $f(n)=\Theta(1)$ – запись означает, что $f(n)$ или равна константе, не равной нулю, или $f(n)$ ограничена константой на ∞ : $f(n) = 7+1/n = \Theta(1)$.

2) Оценка O (O большое)

В отличие от оценки Θ, оценка O требует, чтобы функция $f(n)$ не превышала $g(n)$ начиная с $n > n_0$, с точностью до постоянного множителя:

$\exists\ c > 0,\ n_0 > 0;\ 0 \leq f(n) \leq c * g(n),\ \forall n > n0.$

Выбор того или иного алгоритма для решения заданной задачи требует исследования претендентов не только для оценки сложности, но и для определения размерности и иных характеристик множества реальных исходных данных задачи. Это позволяет рационально выбрать алгоритм.

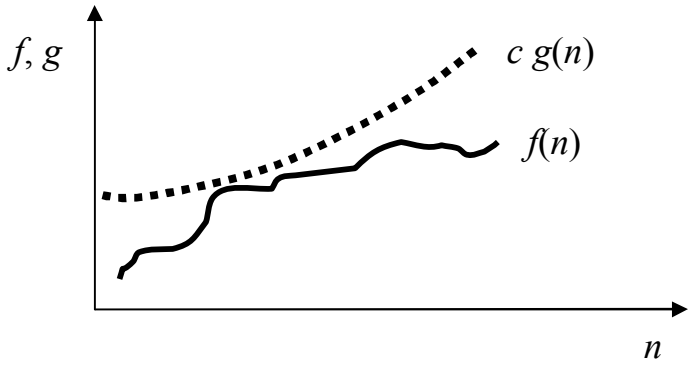


Рисунок 2 – Оценка *O* (О большое)

2. Оценка Ω (Омега большое, существует обозначение Δ, дельта большое)

Эта оценка определяет, в отличие от оценки *O*, определяет функции, с точностью до постоянного множителя, доминируемые функцией $f(n)$. Определение этой оценки выглядит так: $\exists\ c > 0,\ n_0 > 0 : 0 \le c * g(n) \le f(n)$.

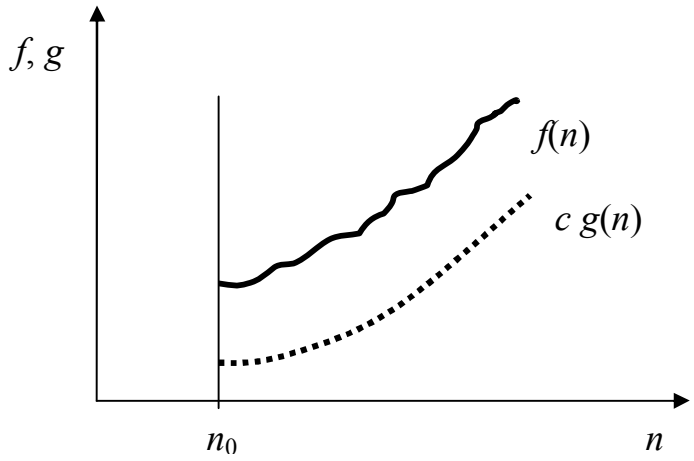


Рисунок 3 – Оценка Ω (Омега)

Например, запись $\Omega(n*\mathrm{Ln}(n))$ обозначает класс функций, которые растут не медленнее, чем $g(n) = n*\mathrm{Ln}(n)$, в этот класс попадают все полиномы со степенью большей единицы, равно как и все степенные функции с основанием большим единицы.

В отличие от оценки *O*, оценка Ω является оценкой снизу – т.е. определяет класс функций, которые растут не медленнее, чем $g(n)$ с точностью до постоянного множителя:

Не всегда алгоритм, имеющий асимптотически оптимальную трудоемкость (в смысле оценок O или Θ), будет иметь лучшие показатели для реального множества исходных данных из-за существенного различия констант, входящих в асимптотические оценки.

Для окончательного выбора потребуется выполнение практического сравнительного анализа алгоритмов, для чего осуществляется:

- детальный анализ трудоемкости алгоритмов, получение в явном виде функции трудоемкости;
- сравнительный анализ функций трудоемкости претендующих алгоритмов для выбора рационального алгоритма решения данной задачи при реальных ограничениях множества исходных данных.

Реализация второго этапа – сравнительного анализа функций трудоемкости – предполагает определение характеристических значений реального множества исходных данных задачи D_A, при которых предпочтение может быть отдано одному из анализируемых алгоритмов. Можно предположить, что таким характеристическим значением является размерность множества исходных данных – n, а трудоемкость алгоритма в основном определяется количеством исходных данных: $f(D_A) = f(\mathrm{n})$.

Таким образом, сравнительный анализ функций трудоемкости предполагает *определение тех интервалов аргумента функций трудоемкости, при которых данный алгоритм может быть выбран в качестве рационального*. В связи с этим предлагаемый ниже метод получил название *интервального анализом функций* {трудоемкости алгоритмов}.

Исходными данными для интервального анализа функций являются *известные функции трудоемкости алгоритмов*, а *результат*ами – *предпочтительные интервалы значений множества исходных данных* задачи для рационального применения того или иного алгоритма.

Выбор одного из алгоритмов в качестве предпочтительного, может быть произведен даже тогда, когда его трудоемкость на данном интервале несколько хуже (более, чем на величину порога φ), чем трудоемкость участников сравнения. Возможно эквивалентное, с расхождением, не более чем на φ, использование на этом интервале несколько алгоритмов. Это приводит к необходимости введения понятий меры в аппарат интервального анализа.

2.2. Общие допущения, принятые в интервальном анализе функций

Пусть заданы:

- трудоемкости двух алгоритмов в виде функций $f(n)$ и $g(n)$ соответственно;
- интервал (a, b) значений n;
- мера $\pi(f(n), g(n))$ расхождения функций при данном значении n.
- порог φ допустимого расхождения значений функций на интервале.

Будем полагать, что:

а) $f(n) = \Omega_{\varphi}(g(n))$, если $\min\limits_{n\in(a,b)} \{\pi(f(n), g(n))\} \geq \varphi$;

б) $f(n) = \Theta_{\varphi}(g(n))$, если $\max\limits_{n\in(a,b)} |\pi(f(n), g(n))| < \varphi$;

в) $f(n) = O_{\varphi}(g(n))$, если $\max\limits_{n\in(a,b)} \{\pi(f(n), g(n))\} \leq -\varphi$.

2.3. Метод интервального анализа

Метод интервального анализа функций трудоемкости алгоритмов, включает этапы:

- Определение для данной конкретной задачи допустимого интервала изменения размерности множества исходных данных (a, b);
- Получение в явном виде функций трудоемкости анализируемых алгоритмов $f(n)$ и $g(n)$;
- Разбиение интервала (a, b) на подынтервалы, в каждой целочисленной точке которых явно выполняется одно из следующих соотношений для принятой меры $\pi(f(n), g(n))$:
  1. если $\varphi - \pi(f(n), g(n)) < 0$, то $f(n) = \Omega_{\varphi}(g(n))$ и предпочтительным на данном подынтервале является алгоритм, имеющий функцию трудоемкости $g(n)$.
  2. если $|\pi(f(n), g(n))| - \varphi < 0$, то $f(n) = \Theta_{\varphi}(g(n))$, и оба алгоритма, с точностью до порога φ, могут быть использованы на этом подынтервале.
  3. если $\pi(f(n), g(n)) + \varphi < 0$, то $f(n) = O_{\varphi}(g(n))$, и предпочтительным на данном подынтервале является алгоритм, имеющий функцию трудоемкости $f(n)$.

2.4. Мера $\pi(f(n), g(n))$

Мера расхождения значений функций $f(n)$ и $g(n)$ при фиксированном n должна удовлетворять следующим требованиям:

- $\pi(f(n), g(n)) = -\pi(f(n), g(n))$;
- $\pi(f(n), g(n)) = 0$, если $f(n) = g(n)$.

На практике может быть определено несколько различных мер, удовлетворяющих данным требованиям. В работе предлагается использовать следующую функцию меры, которая может быть интерпретирована, как угловое расхождение значений аргументов:

$$\pi\big(f(n), g(n)\big) = arctg\frac{f(n)}{g(n)} - arctg\frac{g(n)}{f(n)}.$$

На рисунке 4 показана графическая интерпретация метода интервального анализа функций трудоемкости с использованием меры π.

Поскольку значения функций трудоемкости положительны, то при таком определении меры значения $\pi(f(n), g(n))$ ограничены между $\pi/2$ и $-\pi/2$. Пороговое значение φ может быть интерпретировано, как максимальное значение угла расхождения, для которого алгоритмы могут быть признаны равно применимыми на интервале.

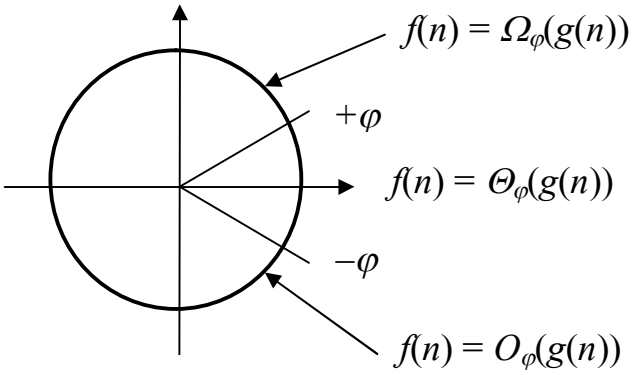


Рисунок 4 – Угловая интерпретация меры $\pi(f(n), g(n))$

3. ПРОГРАММА ВЫПОЛНЕНИЯ ИССЛЕДОВАНИЙ

1. Ознакомиться с теоретическим разделом настоящих методических указаний и повторить соответствующий лекционный материал.

2. Для указанной в варианте задания пары функций трудоемкости, заданных целочисленных интервалов {(20;50), (100;120), (500, 540)} и фиксированных значений $\varphi = \{\pi/32, \pi/24, \pi/18\}$, определить, каково соотношение между функциями трудоемкости на заданном интервале. Код программы, выполняющей расчёты функций, приведён в приложении А. Результаты работы программы следует сохранять в текстовые файлы. Сведения об основных алгебраических функциях и основных конструкциях языка С можно получить либо из системы помощи интегрированной среды разработки, либо из методических указаний, посвященных программированию на языке С.

3. Путем подбора значений аргумента определить интервалы, на которых выполняется соотношение: $f(n) = \Theta_\varphi(g(n))$ для разных значений φ.

4. Построить графики заданных функций на указанном интервале (вручную или с помощью любого программного обеспечения) в масштабе, позволяющем их адекватное восприятие.

5. Сделать выводы по работе, оформить отчет, подготовить ответы на контрольные вопросы.

4. ВАРИАНТЫ ЗАДАНИЙ

№	$f(n)$	$g(n)$
1	$28 \bullet n + 300$	$n^2 + 100$
2	$575 \bullet n^2 + 2 \bullet n + 1981$	$5 \bullet n^3 + 3 \bullet n + 2028$
3	$550 \bullet n^3 + n + 755$	$8 \bullet n^4 + 2 \bullet n^3 + 3 \bullet n + 4550$
4	$2 \bullet n^4 + n + 3114$	$142 \bullet n^3 + n + 2789$
5	$43050 \bullet n + 250$	$4 \bullet n^3 + 70$
6	$7.2e+05 \bullet n^2 + 100 \bullet n + 1833$	$6 \bullet n^5 + n^3 + 2 \bullet n^2 + n + 3031$
7	$795675 \bullet n^2 + 5 \bullet n + 2001$	$5 \bullet n^4 + 2 \bullet n^3 + n + 750$
8	$5476 \bullet n^3 + n^2 + 1024$	$4 \bullet n^5 + n^2 + 3 \bullet n + 6754$
9	$7 \bullet n^4 + 4 \bullet n + 1300$	$56700 \bullet n^2 + 3 \bullet n + 1150$
10	$5 \bullet n^5 + 7 \bullet n^4 + n^2 + 1017$	$2.9e+11 \bullet n + 5050$
11	$1.4e+08 \bullet n + 1250$	$3 \bullet n^4 + 670$
12	$73710 \bullet n^2 + 5 \bullet n + 150$	$3 \bullet n^5 + 2 \bullet n^3 + n + 459$
13	$n^5 + n^3 + 3 \bullet n + 5022$	$16e+05 \bullet n^2 + 3 \bullet n + 1115$
14	$7 \bullet n^5 + n^2 + n + 2810$	$1.5e+08 \bullet n + 4760$
15	$50 \bullet n^2 + n + 1205$	$5 \bullet n^3 + n^2 + 2 \bullet n + 4200$
16	$1.5e+06 \bullet n^3 + 9 \bullet n^2 + 3 \bullet n + 1212$	$6 \bullet n^5 + n^3 + 2 \bullet n^2 + n + 3031$
17	$n^2 + 2.2e+11 \bullet n + 1430$	$4 \bullet n^5 + 2 \bullet n + 1257$
18	$109375 \bullet n + 25705$	$3 \bullet n^5 + 310$
19	$2 \bullet n^5 + n^2 + 2 \bullet n + 450$	$2.9e+10 \bullet n + 1330$
20	$1.32e+10 \bullet n + 1911$	$2 \bullet n^5 + n^2 + n + 2215$

5. СОДЕРЖАНИЕ ОТЧЁТА

1. Цель работы.
2. Вариант задания.
3. Текст программы, реализующей расчеты по соответствующему варианту.
4. Анализ результатов работы программы в виде таблицы результатов и графиков.
5. Развернутый вывод по работе.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Дайте определение алгоритма по Маркову.
2. Дайте определение алгоритма по Колмогорову.
3. Обоснуйте основные требования, предъявляемые к алгоритму.
4. В чём состоит понятие о трудоемкости алгоритма?
5. Что является целью асимптотического анализа алгоритмов?
6. Каковы асимптотические обозначения, принятые для анализа функций

трудоемкости алгоритмов?

7. Дайте определение сложности алгоритма.

8. В чём состоит смысл основных оценок в асимптотическом анализе алгоритмов?

9. Перечислите основные моменты интервального анализа функций трудоемкости алгоритмов

10. Каковы свойства, предъявляемые к мере?

11. Как выглядит графическая интерпретация меры π?

ЛАБОРАТОРНАЯ РАБОТА № 2. “ЭКСПЕРИМЕНТАЛЬНАЯ ОЦЕНКА СРЕДНЕГО КОЛИЧЕСТВА ОПЕРАЦИЙ ПЕРЕПРИСВАИВАНИЯ В АЛГОРИТМЕ ПОИСКА МИНИМУМА”

1. ЦЕЛЬ РАБОТЫ

- В ходе ряда экспериментов иссследовать соответствие теоретической оценки трудоемкости алгоритма поиска минимума, основанной на вычислении гармонического среднего.
- Получить практические навыки применения генераторов случайных чисел при проведении численных экспериментов.

2. КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

2.1. Теоретическая оценка среднего количества операций переприсваивания в алгоритме поиска минимума

Под переприсваиванием в данной лабораторной работе будем понимать операцию присвоения переменной нового значения вместо старого при выполнении определённого условия.

Алгоритм поиска минимума последовательно перебирает элементы массива, сравнивая текущий элемент массива с текущим значением минимума. На очередном шаге, когда просматривается k-й элемент массива, переприсваивание минимума произойдет, если в подмассиве из первых k элементов минимальным элементом является последний. В случае равномерного распределения исходных данных вероятность того, что максимальный из k элементов расположен в определенной (последней) позиции равна $1/k$.

Тогда в массиве из N элементов среднее количество операций переприсваивания максимума определяется как:

$$H_n = \sum_{i=1}^{N} i^{-1} \approx \ln N + \gamma, \qquad \gamma = 0{,}57.$$

Величина H_n называется n-м гармоническим числом. Таким образом, среднее значение (математическое ожидание) среднего количества операций присваивания в алгоритме поиска минимума в массиве из N элементов определяется величиной H_n (для бесконечно большого количества испытаний).

2.2. Понятие о генераторе псевдослучайных чисел

Для тестирования программ часто необходимо использовать наборы случайных чисел. Такие числа могут быть использованы и для реализации тех или иных алгоритмов. Для создания таких чисел используется генератор случайных чисел. В общем случае алгоритм называется рандомизированным (*randomized*), если его поведение определяется не только набором входных величин, но и значениями, которые выдает генератор случайных чисел.

На практике большинство сред программирования предоставляют в распоряжение программиста генератор псевдослучайных чисел, т.е. детерминированный алгоритм, который возвращающий числа, которые ведут себя при статистическом анализе как случайные.

Текст программы, выполняющей подсчёт числа операций переприсваивания, помещён в приложении А.

3. ПРОГРАММА ВЫПОЛНЕНИЯ ИССЛЕДОВАНИЙ

1. Ознакомиться с теоретическим разделом настоящих методических указаний и повторить соответствующий лекционный материал. При проведении исследований необходимо использовать генератор псевдослучайных чисел для создания наборов исходных данных
2. Изучить функции, использующиеся для генерации псевдослучайных чисел, привести их описание в отчете (*rand*, *random*, *randomize* и т. п.).
3. Составить структурную схему и написать программу поиска минимума в массиве сгенерированных псевдослучайных чисел.
4. Написать программу подсчета n-го гармонического числа.
5. Подсчитать количество операций переприсваивания для программной реализации поиска минимума в массиве случайных чисел. Внести изменения в соответствующую программу. Длину массива и максимальное случайное число в последовательности взять в соответствии с вариантом.
6. Сравнить практически полученное значение с теоретическим n-м гармоническим числом. Примеры основных функций приведены в приложении.

7. Сделать выводы по работе, оформить отчет, подготовить ответы на контрольные вопросы.

4. ВАРИАНТЫ ЗАДАНИЙ

Вариант	Наибольшее случайное число в последовательности	Количество элементов в массиве псевдослучайных чисел
1	100	100, 1000, 5000
2	250	150, 550, 1000
3	150	100, 500, 1000
4	950	250, 300, 400
5	200	200, 300, 500
6	800	100, 1000, 2000
7	250	300, 400, 1400
8	850	300, 450, 1000
9	200	150, 200, 1000
10	350	100, 500, 1500
11	100	300, 900, 3000
12	400	100, 200, 500
13	500	100, 1000, 1500
14	950	250, 500, 750
15	150	100, 1000, 5000
16	200	150, 550, 1000
17	750	100, 500, 1000
18	550	250, 300, 400
19	150	200, 300, 500
20	200	100, 1000, 2000

5. СОДЕРЖАНИЕ ОТЧЁТА

1. Цель работы.
2. Вариант задания.
3. Тексты программы, реализующих расчеты по соответствующему варианту, описания функций, используемых для генерации наборов исходных данных, структурные схемы основных функций, используемых в программе.
4. Результаты работы программы.
5. Развернутый вывод по работе.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. В чём состоит понятие трудоемкости алгоритма?

2. В чём состоит понятие о сложности алгоритма?

3. Дайте определение детерминированного алгоритма.

4. Дайте определение рандомизированного алгоритма.

5. Почему псевдослучайные числа так называются: каков смысл приставки “псевдо” и в чём случайность?

6. При каких условиях справедлива оценка числа операций переприсваивания в виде гармонического среднего?

7. Какие аспекты и допущения принимаются во внимание при анализе алгоритмов?

8. Какие классы функций трудоёмкости по виду входных данных Вам известны?

9. Формула вычисления гармонического среднего посредством суммирования, по сути – определение математического ожидания случайной величины. Чему, по Вашему мнению, равны, в этом случае, величины p_i и x_i?

ЛАБОРАТОРНАЯ РАБОТА № 3. “ЭКСПЕРИМЕНТАЛЬНОЕ ОПРЕДЕЛЕНИЕ КОЛИЧЕСТВА ЭЛЕМЕНТАРНЫХ ОПЕРАЦИЙ ЯЗЫКА ВЫСОКОГО УРОВНЯ В ПРОГРАММНОЙ РЕАЛИЗАЦИИ АЛГОРИТМА”

1. ЦЕЛЬ РАБОТЫ

- Экспериментальные исследования теоретически полученной функции трудоемкости для алгоритма точного решения задачи о сумме методом полного перебора.
- Получение практических навыков пооперационного анализа и построения функции трудоёмкости.

2. КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

2. 1. Понятие элементарных операций в формальной системе языка высокого уровня

Будем понимать под **элементарными** операциями языка высокого уровня (на примере языка C++) те, которые могут быть представлены в виде **элементарных конструкций** данного языка (но не обязательно в виде одной машинной команды). Так, за одну элементарную операцию можно считать следующие конструкции языков высокого уровня:

1)	операцию присваивания	$a \leftarrow b$
2)	операцию индексации массива	$a[i]$
3)	арифметические операции	*, /, –,+
4)	операции сравнения	$a < b$
5)	логические операции	*or*, *and*, *not*

Неявно в операцию сравнения входит машинная команда перехода в конструкции if-then-else.

Цикл не является элементарной операцией, т.к. может быть представлен в виде:

for $i \leftarrow 1$ *to* N ⇔ $i \leftarrow 1$; *if* $i \le N$; тело; $i \leftarrow i+1$; ⟵ *end;*

Таким образом, конструкция цикла требует $1 + 3 \times N$ элементарных операций для его организации и поддержания.

В качестве примера построения функции трудоемкости и расстановки элементарных операций рассмотрим задачу суммирования квадратного массива на псевдоязыке:

Программный код	**Элементарных операций в строке**
SumM(A,N;Sum);	
Sum←0	1
for i←1 *to* N	$1 + 3 \times N$
for j←1 *to* N	$1 + 3 \times N$
Sum←*Sum*+*A*[*i*,*j*];	1(←)+1(+)+1(i)+1(j)
end for j	
end for i	
Return (*Sum*);	
End;	

Таким образом, $f_A(N) = 1+1+N\times(3+1+N\times(3+4)) = 7N^2+4N+2 = \Theta(N^2)$.

2.2. Формулировка задачи о сумме

Словесно задача о сумме формулируется как задача нахождения такого множества чисел из данной совокупности, которые в сумме дают заданное число.

В терминах языка высокого уровня задача формулируется, как задача определения таких элементов исходного массива из N чисел, которые в сумме дают число V (задача относится к классу *NPC*).

Дано: Массив $S[i]$, $i=1, N$ и число V.

Требуется: определить S_j: $\sum S_j = V$

Получим **асимптотическую** оценку сложности решения данной задачи при использовании прямого перебора вариантов:

Поскольку исходный массив содержит N чисел, то проверке на V подлежат варианты:

- V содержит 1 слагаемое $\Rightarrow C_N^1 = N$;
- V содержит 2 слагаемых $\Rightarrow C_N^2 = (N*(N-1))/(1*2)$;
- V содержит 3 слагаемых $\Rightarrow C_N^3 = (N*(N-1)*(N-2))/(1*2*3)$;
- и т.д. до N слагаемых включительно.

Т.е. необходимо рассмотреть сумму всех возможных **сочетаний** элементов массива из N чисел.

Любое подмножество из k элементов множества, содержащего n элементов, называется **сочетанием** из n элементов по k.

Число всех различных сочетаний из n элементов по k равно

$$C_n^k = \frac{n!}{k!(n-k)!}.$$

Для всех действительных чисел a, b, отличных от нуля, и для всех натуральных чисел n справедливо следующее соотношение

$$(a+b)^n = \sum_{k=0}^{n} C_n^k a^{n-k} b^k = C_n^0 a^n b^0 + C_n^1 a^{n-1} b^1 + \ldots + C_n^n a^0 b^n .$$

Если в последней формуле заменить b на $-b$, то получим

$$(a-b)^n = \sum_{k=0}^{n} (-1)^k C_n^k a^{n-k} b^k .$$

Биномиальные коэффициенты формулы C_n^k составляют в треугольнике Паскаля строку с номером n.

Каждый коэффициент образуется путем сложения двух стоящих над ним (слева и справа). Крайние значения известны для любого n: $C_n^0 = C_n^n = 1$. В строке с номером n слева направо стоят значения C_n^0, C_n^1, C_n^2, … , C_n^n.

Свойства биномиальных коэффициентов. Для целых $n \geq 0$, $k \geq 0$ справедливо свойство симметрии: $C_n^k = C_n^{k-1}$.

При $a = b = 1$ получаем $\sum_{k=0}^{n} C_n^k = 2^n$, а при $a = 1$, $b = -1$ будет $\sum_{k=0}^{n} (-1)^k C_n^k = 0$.

Поскольку сумма биномиальных коэффициентов равна $(1+1)^n = \sum c_n^k = 2^N$ и для каждого варианта необходимо выполнить суммирование, то оценка сложности алгоритма решения задачи о сумме имеет вид: $f_A(N, V) = O(N*2^N)$.

Таблица 1 – Биномиальные коэффициенты в треугольнике Паскаля

n	C_n^k
1	1
2	1 2 1
3	1 3 3 1
4	1 4 6 4 1
5	1 5 10 10 5 1
6	1 6 15 20 15 6 1
7	1 7 21 35 35 21 7 1
…	……

2.3. Точное решение задачи о сумме

Определим вспомогательный массив *Cnt*, хранящий текущее сочетание исходных чисел, подлежащее проверке на формирование заданной суммы V следующим образом

if Cnt[*j*] = 1 *then*
элемент *S*[*j*] участвует в операции суммирования
else
элемент *S*[*j*] не участвует в операции суммирования

Одно из возможных решений будет получено, если $V = \sum_{j=1}^{N} S[j] \cdot Cnt[j]$

Условия существования решения имеют вид: $\min(S_i) \le V \le \sum S_i$

Могут быть предложены следующие две реализации полного перебора:

- перебор с генерацией непосредственно сочетаний C_N^k числом $2^N - 1$
- перебор по двоичному счётчику, реализованному в массиве *Cnt*:

Последняя алгоритмически более проста и сводится к задаче выполнения инкремента двоичного счётчика, моделируемого массивом *Cnt*.

Вариант программы для выполнения исследований алгоритма решения задачи о сумме, помещён в приложении А.

2.4 Функция трудоемкости процедуры *TaskSum*

а) В лучшем случае $f\check{}(S, N, V) = \Theta(N)$, если $V = S[N]$;

б) В худшем случае $f\hat{}(S, N, V) = \Theta(\mathrm{N}\times 2^N)$, если решения вообще нет

$$f\hat{}_A(N) = 8 \times \mathrm{N} \times 2^{\mathrm{N}} + 16 \times 2^{\mathrm{N}} - 3 \times N - 12,$$

что согласуется с асимптотической оценкой.

3. ПРОГРАММА ВЫПОЛНЕНИЯ ИССЛЕДОВАНИЙ

1. Изучить теоретический раздел настоящих методических указаний и повторить соответствующий лекционный материал.

2. Ознакомится с программой, реализующей алгоритм точного решения задачи о сумме – Приложение А. Составить структурную схему алгоритма точного решения задачи о сумме.

3. Дополнить функцию *TaskSum* конструкциями подсчёта элементарных операций в соответствии с изложенной ранее методикой.

4. Для заданных значений по варианту использовать два способа построения массива: заполнения вручную и заполнения с помощью генератора псевдослучайных чисел (см. лабораторную работу №2). При использовании генератора обеспечить смену псевдослучайных последовательностей. Провести порядка 50 экспериментов.

5. Результаты экспериментов, проведённых с использованием генератора псевдослучайных чисел, представить таблицей следующего вида:

Таблица 2 – Оформление результатов вычислительных экспериментов

№ эксперимента	Экспериментальные данные		В сравнении с теорией	
	Количество элементов, образующих сумму	Использованное количество элементарных операций	Процент от теоретически наихудшего случая $f\check{}(S, N, V)$	Процент от теоретически наихудшего случая $f\hat{}(S, N, V)$
1				
…	…	…	…	…
50				
Среднее по проведённым экспериментам				

6. Сделать выводы по работе, оформить отчет, подготовить ответы на контрольные вопросы.

4. ВАРИАНТЫ ЗАДАНИЙ

Вариант	Размерность вектора случайных чисел	Максимальное случайное число в векторе	Значение суммы (V)
1	10	100	10
2	11	80	15
3	12	70	20
4	13	90	25
5	9	25	60
6	8	35	55
7	10	40	50
8	15	60	45
9	14	50	40
10	13	45	30
11	12	55	35
12	11	100	58
13	9	65	48
14	7	85	38
15	9	75	68
16	8	100	17
17	12	90	29
18	11	50	37
19	10	40	41
20	14	30	43

5. СОДЕРЖАНИЕ ОТЧЁТА

1. Цель работы.
2. Вариант задания.
3. Тексты программы, реализующих расчеты по соответствующему варианту, описания функций, используемых для генерации наборов исходных данных, структурные схемы основных функций, используемых в программе.
4. Результаты работы программы.
5. Развернутый вывод по работе.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. В чём состоит процедура определение трудоемкости алгоритма?
2. Как обозначаются лучший, худший, средний случай в анализе трудоемкости алгоритма?

3. Как выглядит классификация алгоритмов по виду функции трудоемкости?

4. Перечень элементарных операций языка высокого уровня?

5. В чём состоит формулировка задачи о сумме?

6. Каковы основные формулы комбинаторики, использующиеся в задаче о сумме?

7. К какому классу по данным относится функция трудоёмкости алгоритма решения задачи о сумме?

8. Как Вам представляется наилучший вид данных, который может встретиться при решении задачи о сумме?

9. Как Вам представляется наихудший вид данных, который может встретиться при решении задачи о сумме?

10. Как связано число комбинаций, генерируемых в массиве *Cnt* с мощностью множества входных данных?

ЛАБОРАТОРНАЯ РАБОТА № 4. “ЭКСПЕРИМЕНТАЛЬНОЕ ОПРЕДЕЛЕНИЕ СРЕДНЕГО ВРЕМЕНИ ВЫПОЛНЕНИЯ ОБОБЩЕННОЙ ЭЛЕМЕНТАРНОЙ ОПЕРАЦИИ МЕТОДОМ ТЕСТОВЫХ ПРОГОНОВ”

1. ЦЕЛЬ РАБОТЫ

- Экспериментальное определение среднего времени выполнения обобщенной элементарной операции в языке высокого уровня.
- Получения прогноза времени выполнения программы для больших размерностей множества исходных данных в зависимости от типов данных (целые, длинные целые, действительные).
- Получение практических навыков экспериментальной оценки среднего времени выполнения элементарных операций и программной реализации алгоритмов в целом.

2. КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

2.1. Понятие элементарных операций в формальной системе языка высокого уровня

Анализ трудоемкости алгоритмов позволяет получить функциональную зависимость между параметрами исходной задачи (мощностью множества

исходных данных) и количеством выполненных элементарных операций в заданной среде реализации.

При этом переход от функции трудоемкости ко времени выполнения программы для алгоритма с известной трудоемкостью зависит от:

1) времени выполнения элементарных операций в среде языка реализации алгоритма.

2) специфики реализации алгоритма (квалификация программиста, структуры данных и др.) на данном языке программирования.

Таким образом, имея аналитическую оценку трудоемкости данного алгоритма – $f_A(N)$ и ее экспериментальное подтверждение и зная среднее время выполнения обобщенной элементарной операции для данного языка и данного процессора можно получить оценку времени выполнения программы:

$$T_A(N) = \sum_{i \subset R} F_A^{(i)}(N) \cdot \bar{t}_i, \qquad (1)$$

где N – объём входных данных; i – множество “элементарных” операций, используемых в реализации алгоритма; R – множество “элементарных” операций, предоставляемых вычислительным устройством; $F_A^{(i)}(N)$ – число операций i–го типа; $\bar{t}_i$ – среднее время выполнения i–ой “элементарной” операции.

Построение (определение) оценки $\bar{t}_i$ связано с решением следующих трех задач:

1. Определение времени выполнения элементарных операций в среде языка реализации для различных типов данных (типы операций - целые, действительные).

2. Построение средневзвешенной оценки для типов операций, на основе предположений или экспериментальных оценок по частотной встречаемости операции в рамках данного типа.

3. Построение на этой основе оценки $\bar{t}_i$ с использованием весов, характеризующих особенности данного класса задач

При решении задачи (1) можно воспользоваться техническими данными процессора, только в том случае, когда языком реализации является Ассемблер. В случае алгоритмического языка высокого уровня, придётся использовать метод пооперационного анализа, элементы которого использовались при выполнении лабораторной работы № 3, и воспользоваться обобщающей формулой

$$t_A(N) = t_{cp} * f_A(N), \qquad (2)$$

где t_{cp} – среднее время выполнения прогона программы, $f_A(N)$ – суммарное число элементарных операций.

3. ПРОГРАММА ВЫПОЛНЕНИЯ ИССЛЕДОВАНИЙ

1. Ознакомиться с теоретическим разделом настоящих методических указаний и повторить соответствующий лекционный материал.

2. Используя программную реализацию алгоритма поиска минимума в массиве случайных чисел, внести в неё изменения, позволяющие подсчитывать элементарные операции по видам и общее их число, а также зафиксировать время выполнения программы.

3. Длину массива и максимальное случайное число в последовательности взять в соответствии с вариантом.

4. Выполнить тестовые прогоны не менее 20 для каждого числа элементов в массиве, при этом обеспечить смену псевдослучайных последовательностей. Данные, полученные в ходе эксперимента, оформить в виде таблицы

№	N	Число элементарных операций по видам					$t_{вып}$
		:=	[ ]	+	<	$\sum$	
…	…	…	…	…	…	…	
Средние						$f_A(N)$	$t_{ср}$

5. В последнюю строку таблицы поместить средние наблюдаемые значения, и выполнить расчёт, воспользовавшись 2.

6. Сделать выводы по работе, оформить отчет, подготовить ответы на контрольные вопросы.

4. ВАРИАНТЫ ЗАДАНИЙ

Вариант	Наибольшее случайное число в последовательности	Количество элементов в массиве случайных чисел
1	100	100, 1000, 5000
2	250	150, 550, 1000
3	150	100, 500, 1000
4	950	250, 300, 400
5	200	200, 300, 500
6	800	100, 1000, 2000
7	250	300, 400, 1400
8	850	300, 450, 1000
9	200	150, 200, 1000
10	350	100, 500, 1500
11	100	300, 900, 3000
12	400	100, 200, 500
13	500	100, 1000, 1500
14	950	250, 500, 750
15	150	100, 1000, 5000
16	200	150, 550, 1000

Вариант	Наибольшее случайное число в последовательности	Количество элементов в массиве случайных чисел
17	750	100, 500, 1000
18	550	250, 300, 400
19	150	200, 300, 500
20	200	100, 1000, 2000

6. СОДЕРЖАНИЕ ОТЧЁТА

1. Цель работы.
2. Вариант задания.
3. Тексты программы, реализующих расчеты по соответствующему варианту, описания функций, используемых для генерации наборов исходных данных, структурные схемы основных функций, используемых в программе.
4. Результаты работы программы.
5. Развернутый вывод по работе.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Сформулируйте задачу построения временной оценки трудоёмкости.
2. Перечислите этапы метода пооперационного анализа.
3. Как реализуется метод прямого определения среднего времени.
4. Поясните, насколько применим метод пооперационного анализа при рассмотрении параметрических и порядково-зависимых алгоритмов?
5. Зачем в методе, предложенном Гиббсоном, выполняется классификация задач по типам?
6. Какие задачи включены в класс *P*?
7. Какими свойствами обладает класс *P*?
8. Что означает термин “полиномиально проверяемые задачи”?
9. В чём состоит идея алгоритмической сводимости одной задачи к другой?
10. Приведите примеры *NPC*-задач.

ЛАБОРАТОРНАЯ РАБОТА № 5.
“СРАВНИТЕЛЬНОЕ ИССЛЕДОВАНИЕ АЛГОРИТМОВ ПОИСКА КРАТЧАЙШИХ ПУТЕЙ НА ГРАФАХ”

1 ЦЕЛЬ РАБОТЫ

- Исследование фундаментальных алгоритмов поиска кратчайших путей на графах на примере метода динамического программирования.
- Получение навыков практического применения алгоритмов на графах.

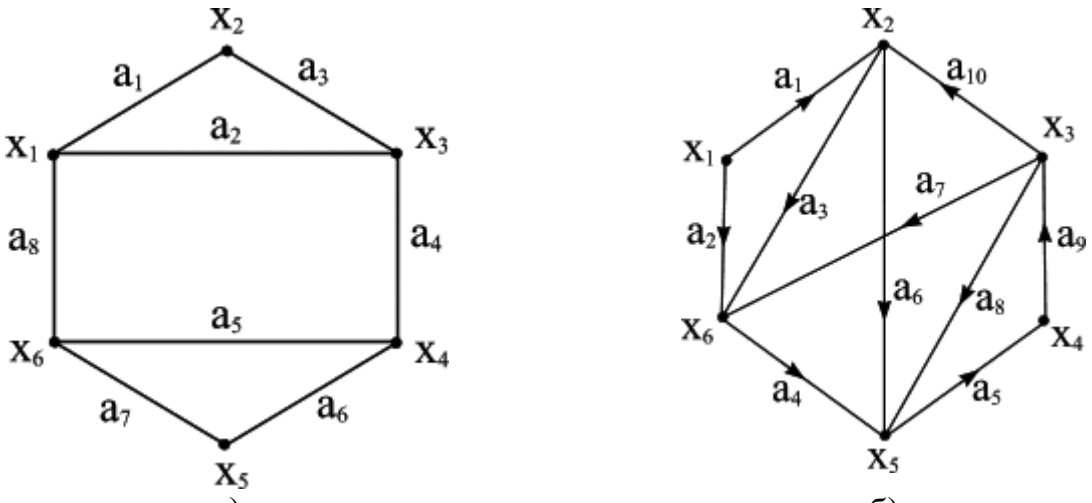
2 ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ

2.1 Основные определения

Путем (или *ориентированным маршрутом*) ориентированного графа называется последовательность дуг, в которой конечная вершина всякой дуги, отличной от последней, является начальной вершиной следующей. Так, на рисунке 1, а, последовательности дуг $\mu_1 = \{a_6, a_5, a_9, a_8, a_4\}$, $\mu_2 = \{a_1, a_6, a_5, a_9\}$, $\mu_3 = \{a_1, a_6, a_5, a_9, a_{10}, a_6, a_4\}$ являются путями.

Ориентированной цепью (*орцепью*) называется такой путь, в котором каждая дуга используется не больше одного раза. Так, например, приведенные выше пути μ_1 и μ_2 являются орцепями, а путь μ_3 не является таким, поскольку дуга a_6 в нем используется дважды.

Маршрут есть неориентированный "двойник" пути, и это понятие рассматривается в тех случаях, когда можно пренебречь направленностью дуг в графе.



а) б)

Рисунок 1 – Пример графа

Таким образом, маршрут есть последовательность ребер $a_1, a_2,...,a_q$, в которой каждое ребро a_i, за исключением, возможно, первого и последнего ребер, связано с ребрами a_{i-1} и a_{i+1} своими двумя концевыми вершинами.

Последовательности дуг на рисунке1, б, $\mu_4 = \{a_2, a_4, a_8, a_{10}\}$, $\mu_5 = \{a_2, a_7, a_8, a_4, a_3\}$ и $\mu_6 = \{a_{10}, a_7, a_4, a_8, a_7, a_2\}$ являются маршрутами.

Контуром (простой цепью) называется такой путь (маршрут), в котором каждая вершина используется не более одного раза. Например, путь μ_2 является контуром, а пути μ_1 и μ_3 - нет. Очевидно, что контур является также цепью, но обратное утверждение неверно. Например, путь μ_1 является цепью, но не контуром, путь μ_2 является цепью и контуром, а путь μ_3 не является ни цепью, ни контуром.

Аналогично определяется простая цепь в неориентированных графах. Так, например, маршрут μ_4 есть простая цепь, маршрут μ_5 – цепь, а маршрут μ_6 не является цепью.

Путь или маршрут можно изображать также последовательностью вершин. Например, путь μ_1 можно представить также: $\mu_1 = \{x_2, x_5, x_4, x_3, x_5, x_6\}$ и такое представление часто оказывается более полезным в тех случаях, когда осуществляется поиск контуров или простых цепей.

Иногда дугам графа G сопоставляются (приписываются) числа – дуге (x_i,x_j) ставится в соответствие некоторое число c_{ij}, называемое *весом*, или *длиной*, или *стоимостью* (*ценой*) дуги. Тогда граф G называется *взвешенным*. Иногда веса (числа v_i) приписываются вершинам x_i графа.

При рассмотрении пути μ, представленного последовательностью дуг ($a_1, a_2,...,a_q$), за его *вес* (или *длину*, или *стоимость*) принимается число $L(\mu)$, равное сумме весов всех дуг, входящих в μ, т. е.

$$L(\mu) = \sum_{(x_i,x_j)\in\mu} c_{i,j} . \qquad (1)$$

Таким образом, когда слова "длина", "стоимость", "цена" и "вес" применяются к дугам, то они эквивалентны по содержанию, и в каждом конкретном случае выбирается такое слово, которое ближе подходит по смыслу.

Длиной (или *мощностью*) пути и называется число дуг, входящих в него.

2.2 Задача о кратчайшем пути

Пусть дан граф $G=(X, \Gamma)$, дугам которого приписаны веса (стоимости), задаваемые матрицей $C = [c_{i,j}]$, $i = 1, m$, $j = 1, n$.

Задача о кратчайшем пути состоит в нахождении кратчайшего пути от заданной начальной вершины (*истока*) s до заданной конечной вершины (*стока*) t, при условии, что такой путь существует:

Найти $\mu(s,t)$ при$L(\mu)\rightarrow min$, $s,t \in X$, $t \in R(s)$, где $R(s)$ - множество, достижимое из вершины s.

В общем случае элементы c_{ij} матрицы весов C могут быть положительными, отрицательными или нулями. Единственное ограничение состоит в том, чтобы в G не было циклов с суммарным отрицательным весом. Отсюда следует, что дуги (ребра) графа G не должны иметь отрицательные веса.

Почти все методы, позволяющие решить задачу о кратчайшем (s – t)-пути, дают также (в процессе решения) и все кратчайшие пути от s к $x_i(\forall x_i \in X)$. Таким образом, они позволяют решить задачу с небольшими дополнительными вычислительными затратами.

Допускается, что матрица весов C не удовлетворяет условию треугольника, т. е. не обязательно $c_{i,j} \le c_{i,k} + c_{k,j}$ для всех i, j и k.

Если в графе G дуга (x_i, x_j) отсутствует, то ее вес полагается равным бесконечности.

Ряд задач, например, задачи нахождения в графах путей с максимальной надежностью и с максимальной пропускной способностью, связаны с задачей о кратчайшем пути, хотя в них характеристика пути (скажем, вес) является не суммой, а некоторой другой функцией характеристик (весов) дуг, образующих путь. Такие задачи можно переформулировать как задачи о кратчайшем пути и решать их соответствующим образом.

Существует множество методов решения данной задачи, отличающиеся областью применимости и трудоемкостью (Дейкстры, Флойда, динамического программирования). Среди них большое распространение получили частные алгоритмы, применяющиеся при решении частных задач, и имеющие меньшую трудоемкость. Эти частные случаи встречаются на практике довольно часто (например, когда $c_{i,j}$ являются расстояниями), так что рассмотрение этих специальных алгоритмов оправдано.

На практике задачу кратчайшего пути часто требуется решать для класса ориентированных ациклических графов. Такая задача успешно решается с помощью метода динамического программирования.

2.3.1 Алгоритм Дейкстры

Данный алгоритм предложил датский исследователь *Дейкстра* в 1959 году.

Пусть требуется найти кратчайшее расстояние из вершины s в вершину f. Изначально необходимо создать массив d[], в котором будет храниться для каждой вершины кратчайшее расстояние, за которое можно попасть в нее из исходной. При создании d[s] = 0, а все остальные элементы равны бесконечности (на практике выбирают очень большое число). Кроме того, потребуется массив *used*[], чтобы хранить бит для каждой вершины, определяющий помечена она или еще нет. Изначально все вершины являются непомеченными.

Сам алгоритм состоит из n (n – количество вершин) итераций, на очередной итерации выбирается такая непомеченная вершина, что кратчайшее расстояние от исходной до нее минимально. То есть вершина v такая, что $d[v]$ = min из всех вершин, которые еще не помечены (на первой итерации будет выбрана вершина s, что логично). А далее просматриваются все ребра, выходящие из вершины v, и производятся так называемые *релаксации*. Таким образом, если есть ребро ($v \to to$) с весом len, то будет предпринята попытка улучшить значение $d[to]$ = min($d[to]$, $d[v]$ + len). Действительно, выбирается минимальное значение между тем, что уже было определено и новым способом добраться до вершины to. На этом текущая итерация заканчивается. После окончания n итераций в массиве d будут лежать ответы – расстояния для каждой вершины. То есть в вершину f мы сможем добраться за $d[f]$ единиц расстояния.

Стоит заметить, что если до некоторых вершин добраться невозможно, то значение d для них не изменится, то есть останется равным бесконечности (тому числу, которым мы инициализировали массив).

Код программы приводится в приложении А.

2.3.2Метод динамического программирования

Прямая итерация. Пусть вершины пронумерованы так, что дуга (x_i, x_j) всегда ориентирована от вершины x_i к вершине x_j, имеющей больший номер. Для ациклического графа такая нумерация всегда возможна и производится очень легко. При этом начальная вершина s получает номер 1, а конечная t - номер n.

Пусть $\lambda(x_i)$ – пометка вершины x_i, равная длине кратчайшего пути от 1 до x_j, s – начальная вершина (источник), t – конечная вершина (сток).

Шаг 1. Положить $\lambda(s) = 0$, $\lambda(x_i) = \infty$ для всех вершин $x_i \in X \backslash s; i = 1$.

Шаг 2. $i=i+1$. Присвоим вершине x_j пометку $\lambda(x_j)$, равную длине кратчайшего пути от 1 до x_j, используя для этого соотношение

$$\lambda(x_j) = \min\{\lambda(x_i) + c_{i,j}\}. \qquad (2)$$

Шаг 3. Повторить п.2. до тех пор, пока последняя вершина n не получит пометку $\lambda(t)$.

Необходимо отметить, что если вершина x_j помечена, то пометки $\lambda(x_j)$ известны для всех вершин $x_i \in \Gamma^{-1}(x_j)$, так как в соответствии со способом нумерации это означает, что $x_i < x_j$ и, следовательно, вершины x_i уже помечены в процессе применения алгоритма.

Пометка $\lambda(t)$ равна длине самого короткого пути от s до t. Сами дуги, образующие путь, могут быть найдены способом последовательного возвращения. А именно дуга (x_i, x_j),согласно (2), принадлежит пути тогда и только тогда, когда

$$\lambda(x_j) = \lambda(x_i) + c_{i,j}. \quad (3)$$

Обратная итерация: начиная с вершины t, имеющей номер n, полагаем на каждом шаге x_j равной такой вершине (скажем, x_j^*), для которой выполняется соотношение (3), и так продолжаем до тех пор, пока не будет достигнута начальная вершина (т.е. пока не будет $x_j^* \equiv s$).

Совершенно очевидно, что пометка $\lambda(x_j)$ вершины x_j даёт длину кратчайшего пути μ от s до x_j.

2.4 Алгоритм топологической сортировки

В некоторых случаях исходный граф является ациклическим, но имеет неправильную нумерацию – содержит дуги (x_j, x_i),ориентированные от вершины x_j к вершине x_i, имеющей меньший номер ($j > i$). Для успешного нахождения кратчайшего пути с помощью метода динамического программирования к такому графу сначала применяется алгоритм топологической сортировки вершин.

Алгоритм топологической сортировки вершин простой и эффективный. Он позволяет не только правильно перенумеровать вершины графа, но и определить его ацикличность.

Шаг 1. Положить $i = n$, где n – число вершин графа G.

Шаг 2. В графе определяется вершина x_k, для которой выполняется условие $|\Gamma(x_k)| = \varnothing$ (т.е., вершина, из которой не выходит ни одна дуга). Вершина x_k получает порядковый номер i(перенумеруется) и исключается из дальнейшего рассмотрения вместе со всеми входящими в нее инцидентными дугами. $i=i-1$.

Шаг 3. Повторять п.2. до тех пор, пока не будет выполнено одно из условий:

1) $i = 1$ – достигнута начальная вершина. Вершины графа получили правильную нумерацию.

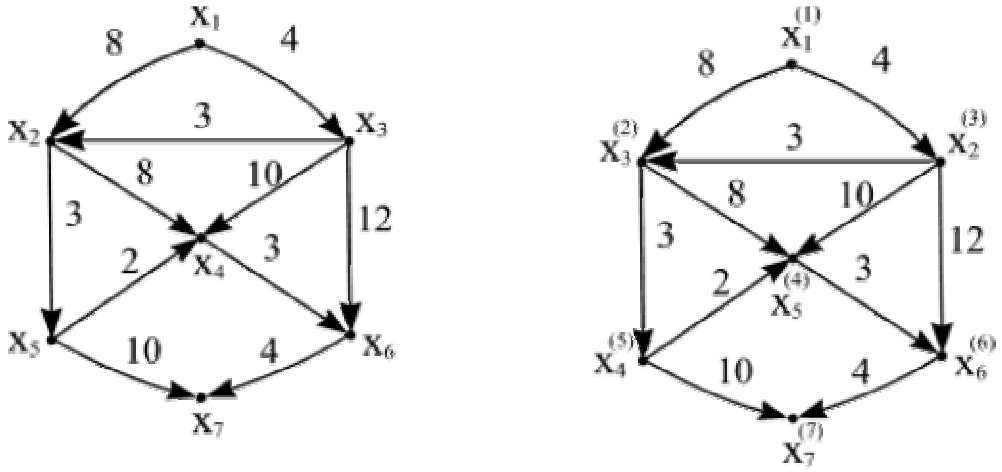
2) Невозможно определить вершину, для которой выполнялось бы условие $|\Gamma(x_k)| = \varnothing$. Следовательно, в графе имеется цикл.

В последнем случае алгоритм динамического программирования неприменим. Для поиска кратчайших путей на таком графе необходимо использовать более эффективные методы, например, алгоритм Дейкстры.

2.5 Демонстрационный пример

Для графа, изображенного ниже на рисунке 2, а), определим кратчайший путь между вершинами x_1 и x_7, используя метод динамического программирования.

Так как граф содержит дуги (x_3 ,x_2) и (x_5, x_4), имеющие неправильную нумерацию (от большего к меньшему), необходимо перенумеровать вершины графа, применяя алгоритм топологической сортировки вершин.



а) начальный вид графа б) после топологической сортировки

Рисунок 2 – Исходный и преобразованный графы

В нашем случае из графа будут последовательно исключаться вершины x_7, x_6, x_4, x_5, x_2, x_3, x_1. Соответственно, вершины графа получат новую нумерацию, и граф будет иметь вид, представленный на рисунке 2, б) (старая нумерация вершин сохранена в скобках).

На первом шаге оценка $\lambda(x_1) = 0$.Осуществляется переход к вершине x_2. Множество $\Gamma^{-1}(x_2)$ включает только одну вершину x_1.Следовательно, оценка для вершины x_2 определяемая по формуле (2), будет $\lambda(x_2) = \min\{0+4\} = 4$. Переходим к вершине x_3. Для вершины x_3 множество $\Gamma^{-1}(x_3) = \{x_1, x_2\}$. В этом случае, оценка будет выбираться как минимальная из двух возможных: $\lambda(x_3)$= =min{0+8, 4+3} = *7*.Для вершины x_4 оценка определяется снова однозначно: $\lambda(x_4)$= min{7+3} =10. Однако после перехода к вершине x_5 необходимо рассматривать сразу три входящих дуги (x_2, x_5), (x_3, x_5), (x_4, x_5).По формуле (2) определяется оценка для вершины x_5: $\lambda(x_5) = \min\{4+10, 7+8, 10+2\} = 12$.

Далее, аналогичным образом вершина x_6 получает оценку $\lambda(x_6)$= =min{*4+12*, *12+3*}=*15* и, наконец, вершина x_7получает оценку $\lambda(x_7)$=min{10+10, 15+4}=19.

Таким образом, конечная вершина x_7 пути $\mu(x_1, x_7)$достигнута, длина пути равна $L(\mu) = 19$.

С помощью выражения (3) последовательно определяются вершины, которые входят в кратчайший путь. Перемещаясь от конечной вершины x_7, выбирается последовательность вершин, для которой выражение (3) принимает значение «истинно»: x_6, x_5, x_4, x_3, x_2, x_1, т.е. кратчайший путь проходит последовательно через все вершины графа.

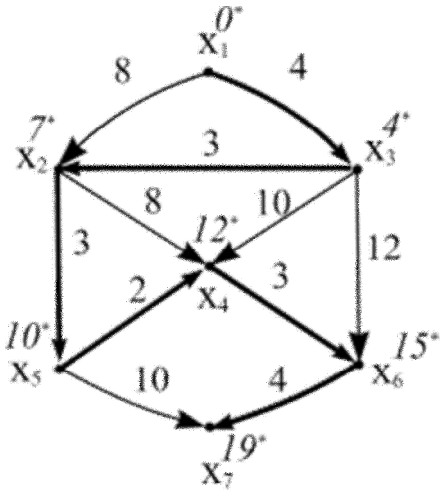


Рисунок 3 – Кратчайший путь на графе

Действительно, легко убедиться в истинности выражений:

- $\lambda(x_7) = \lambda(x_6) + c_{67}$: (19=15+4);
- $\lambda(x_6) = \lambda(x_5) + c_{56}$:(15=12+3);
- $\lambda(x_5) = \lambda(x_4) + c_{45}$: (12=10+2);
- $\lambda(x_4) = \lambda(x_3) + c_{34}$: (10=7+3);
- $\lambda(x_3) = \lambda(x_2) + c_{23}$: (7=4+3);
- $\lambda(x_2) = \lambda(x_1) + c_{12}$: (4=0+4).

После восстановления исходной нумерации вершин графа, окончательно определяется кратчайший путь: $\mu(x_1, x_7) = \{x_1, x_3, x_2, x_5, x_4, x_6, x_7\}$.

Решение задачи завершено, а его результат – в виде выделенного пути изображен на рисунке 3. Курсивом “со звездочкой” отмечены значения пометок вершин $\lambda(x_i)$.

Текст программы приводится в приложении А.

3. ПРОГРАММА ВЫПОЛНЕНИЯ ИССЛЕДОВАНИЙ

1.Получить задание у преподавателя в виде исходного ориентированного графа.

2.Составить структурную схему программы, определяющей кратчайший путь на графе от заданной начальной вершины *s* до заданной конечной вершины *t* с помощью метода динамического программирования.

3.Составить структурную схему программы, реализующей алгоритм топологической сортировки с произвольной нумерацией вершин графа.

4.Создать программу, реализующую метод динамического программирования и алгоритм топологической сортировки вершин. Исходный граф задается в виде матрицы смежности, вводимой построчно с помощью консоли. Указание: для определения вершин, входящих во множество $\Gamma^{-1}(x_i)$ используйте *j*-й столбец матрицы смежности.

5.Создать программу, которая использует приведенный в данной работе алгоритм Дейкстры для заданного графа.

6.Сравнить время выполнения двух алгоритмов.

4. СОДЕРЖАНИЕ ОТЧЁТА

1. Исходное задание и цель работы.
2.Структурная схема программы по п.2.
3. Структурная схема программы по п.3.
3. Распечатка текста программы по п.4.
4. Распечатка текста программы по п.5.
5. Контрольный пример и результаты машинного расчета.
6. Выводы по работе.

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Дайте определение пути, маршрута, цепи, контура.
2. В чём состоит отличие дуги от ребра?
3. Какой граф называется взвешенным?
4. Как определяется длина пути графа?
5. В чём состоит задача нахождения кратчайшего пути на графе.
6. В чём особенности реализации метода динамического программирования для нахождения кратчайшего пути на графе?
7. Каковы ограничения применения метода динамического программирования для нахождения кратчайшего пути на графе?
8. Что называется правильной нумерацией вершин графа?
9. Каковы особенности применение алгоритма топологической сортировки для перенумерации вершин графа?
10. Сравните алгоритм Дейкстры с методом динамического программирования.
11. Всегда ли задача на нахождение кратчайшего пути имеет решение?
12. Перечислите способы представления графом в программах и отметьте их достоинства и недостатки.

ЛАБОРАТОРНАЯ РАБОТА № 6.
“ИССЛЕДОВАНИЕ АЛГОРИТМА ПОСТРОЕНИЯ КРАТЧАЙШИХ ОСТОВЫХ ДЕРЕВЬЕВ ГРАФА”

1. ЦЕЛЬ РАБОТЫ

- Исследование алгоритма Прима-Краскала построения кратчайших остовых деревьев графа методом пооператорного анализа.
- Получение навыков выполнения пооператорного анализа фундаментальных алгоритмов

2. КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

2.1 Основные определения

Одним из наиболее важных понятий теории графов является *дерево*.

Неориентированным деревом называется связанный граф, не имеющий циклов.

Суграфом графа G является подграф G_p, содержащий все вершины исходного графа.

Если $G = (X, A)$ – неориентированный граф с n вершинами, то связный суграф G_p, не имеющий циклов, называется *остовым деревом* (*остовом*) графа G.

Для остового дерева справедливо соотношение:

$$G_p = (X_p, A_p) \subseteq G, \text{ где } X_p = X, A_p \subseteq A. \quad (1)$$

Можно показать, что остовое дерево обладает следующими свойствами:

- остовое дерево графа с n вершинами имеет n – 1 ребро, то есть $|X_p| = |A_p| - 1$;
- существует единственный путь, соединяющий любые две вершины остова графа, сиречь $\forall x_i x_j \in X_p\ (i \neq j) \rightarrow \exists! \mu(x_i x_j)$.

Например, если G – граф, показанный на рисунке 1,а), то графы на рисунках 1,б) и 1,в) являются остовами графа G. Из сформулированных выше определений вытекает, что остов графа G можно рассматривать как минимальный связанный остовый подграф графа G.

Понятие дерева как математического объекта было впервые предложено Кирхгофом в связи с определением фундаментальных циклов, применяемых при анализе электрических цепей. Приблизительно десятью годами позже Кэли вновь (независимо от Кирхгофа) ввел понятие дерева и получил большую часть первых результатов в области исследования свойств деревьев.

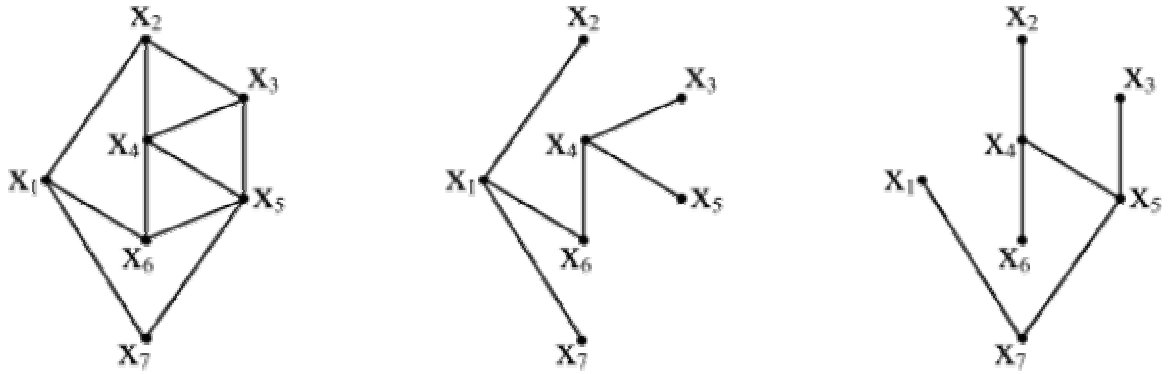


Рисунок 1 – Представление графа в виде остовых деревьев

Известна следующая теорема Кэли: На графе с n вершинами можно построить n^{n-2} остовых деревьев.

Ориентированное дерево представляет собой ориентированный граф без циклов, в котором полустепень захода каждой вершины, за исключением одной (вершины r), равна единице, а полустепень захода вершины r (называемой корнем этого дерева) равна нулю.

Из приведенного определения следует, что ориентированное дерево с n вершинами имеет n – 1 дуг и связно. Ниже, на рисунке 2, показан граф, который является ориентированным деревом с корнем в вершине x_1.

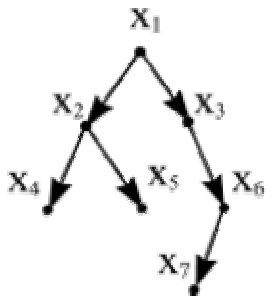


Рисунок 2 – Ориентированное дерево

Неориентированное дерево можно преобразовать в ориентированное: надо взять его произвольную вершину в качестве корня и ребрам приписать такую ориентацию, чтобы каждая вершина соединялась с корнем (только одной) простой цепью.

"Генеалогическое дерево", в котором вершины соответствуют лицам мужского пола, а дуги ориентированы от родителей к детям, представляет собой хорошо известный пример ориентированного дерева. Корень в этом дереве соответствует "основателю" рода (лицу, родившемуся раньше остальных).

2.2 Кратчайший остов графа

В лабораторной работе исследуется метод прямого построения *кратчайших остовых деревьев* во взвешенном графе (в котором веса приписаны дугам). Кратчайшее остовое дерево графа находит применение при прокладке дорог (газопроводов, линий электропередач и т. д.), когда необходимо связать n точек некоторой сетью так, чтобы общая длина "линий связи" была минимальной. Если точки лежат на евклидовой плоскости, то их можно считать вершинами полного графа G с весами дуг, равными соответствующим "прямолинейным" расстояниям между концевыми точками дуг. Если "разветвление" дорог допускается только в заданных n точках, кратчайшее остовое дерево графа G будет как раз требуемой сетью дорог, имеющей наименьший вес.

Рассмотрим взвешенный связный неориентированный граф $G = (X, A)$, а вес ребра (x_i, x_j) обозначим c_{ij}. Из большого числа остовов графа нужно найти один, у которого сумма весов ребер наименьшая. Такая задача возникает, например, в том случае, когда вершины являются клеммами электрической сети, которые должны быть соединены друг с другом с помощью проводов наименьшей общей длины (для уменьшения уровня наводок). Другой пример: вершины представляют города, которые нужно связать сетью трубопроводов; тогда наименьшая общая длина труб, которая должна быть использована для строительства (при условии, что вне городов "разветвления" трубопроводов не допускаются), определяется кратчайшим остовом соответствующего графа.

Задача построения кратчайшего остова графа является одной из немногих задач теории графов, которые можно считать полностью решенными.

2.3 Алгоритм Прима-Краскала

Этот алгоритм порождает остовое дерево посредством разрастания только одного поддерева, например X_p, содержащего больше одной вершины. Поддерево постепенно разрастается за счет присоединения ребер (x_i, x_j), где $x_i \in X_p$ и $x_j \notin X_p$, причем добавляемое ребро должно иметь наименьший вес c_{ij}. Процесс продолжается до тех пор, пока число ребер в A_p не станет равным n–1. Тогда поддерево $G_p = (X_p, A_p)$ будет требуемым остовым деревом. Впервые такая операция была предложена Примом и Краскалом (с разницей – в способе построения дерева), поэтому данный алгоритм получил название Прима-Краскала.

Алгоритм начинает работу с включения в поддерево начальной вершины. Поскольку остовое дерево включает все вершины графа G, то выбор начальной вершины не имеет принципиального значения. Будем каждой очередной вершине присваивать пометку $\beta(x_i) = 1$, если вершина x_i принадлежит поддереву X_p и $\beta(x_j) = 0$ – в противном случае.

Алгоритм имеет вид:

Шаг 1. Пусть $X_p=\{x_1\}$, где x_1 – начальная вершина, и $A_p= \varnothing$ (A_p является множеством ребер, входящих в остовое дерево). Вершине x_1 присвоить пометку $\beta(x_1) = 1$. Для каждой вершины $x_i \notin X_p$ присвоить $\beta(x_i)=0$.

Шаг 2. Из всех вершин $x_j \in \Gamma(X_p)$, для которых $\beta(x_i)=0$, найти вершину x_j^* такую, что

$$c(x_i, x_j^*) = \min_{x_j \in \Gamma(X_p)} \{c(x_i, x_j)\}, \text{ где } x_i \in X_p \text{ и } x_j \notin X_p. \quad (2)$$

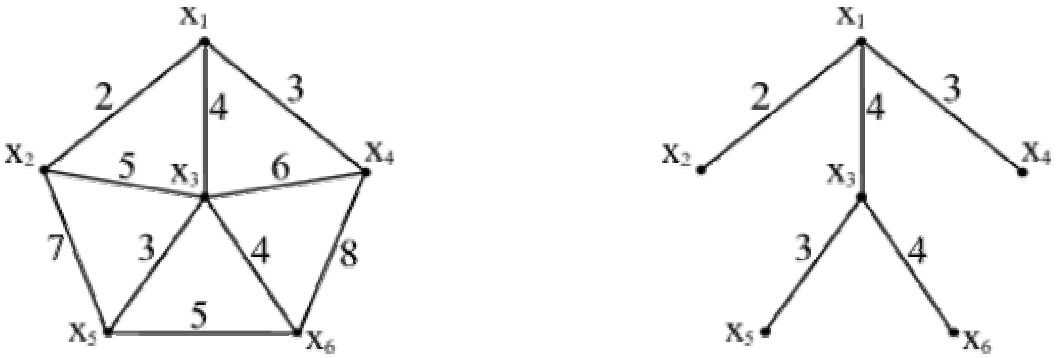
Шаг 3. Обновить данные: $X_p = X_p \cup \{x_j^*\}$, $A_p = A_p \cup (x_i, x_j)$.

Присвоить $\beta(x_j^*) = 1$.

Шаг 4.Если $|X_p| = n$, то остановиться. Ребра в A_p образуют кратчайший остов графа.

Если $|X_p|<n$, то перейти к шагу 2.

2.4 Контрольный пример



а) б)

Рисунок 3 – Исходный граф и минимальное остовое дерево

Для примера рассмотрим граф, изображенный на рисунке 3, а). Найдем для него кратчайшее остовое дерево, используя для этой цели рассмотренный выше алгоритм Прима-Краскала. Обозначим множество смежных вершин, не входящих в порожденное поддерево, как $\Gamma^*(X_p)$. Таким образом, для всех вершин, входящих в это множество, оценка $\beta(x_j)=0$, $\forall\ x_j \in \Gamma^*(X_p)$.Вектор B представляет собой множество оценок $\beta(x_i)$ для всех вершин графа G: $\forall\ x_i \in X$. Длина порожденного поддерева обозначается как L.

Итерация 1.

$X_p=\{x_1\}$; $A_p = \varnothing$, $\Gamma^*(X_p)=\{x_2, x_3, x_4\}$; $c(x_1, x_2^*)=2$; $B=\{1, 1, 0, 0, 0, 0\}$; L=2.

Итерация 2. $X_p = \{x_1, x_2\}$; $A_p = \{(x_1, x_2)\}$; $\Gamma^*(X_p) = \{x_3, x_4, x_5\}$; $c(x_1, x_4^*) = 3$; $B = \{1, 1, 0, 1, 0, 0\}$; L = 2 + 3 = 5.

Итерация 3. $X_p = \{x_1, x_2, x_4\}$; $A_p = \{(x_1, x_2); (x_1, x_4)\}$; $\Gamma^*(X_p) = \{x_3, x_5, x_6\}$; $c(x_1, x_3^*) = 4$; $B = \{1, 1, 1, 1, 0, 0\}$; $L = 5 + 4$.

Итерация 4. $X_p = \{x_1, x_2, x_3, x_4\}$; $A_p=\{(x_1, x_2); (x_1, x_4); (x_1,x_3)\}$; $\Gamma^*(X_p) = \{x_5, x_6\}$; $c(x_3, x_5^*) = 2$; $B=\{1, 1, 1, 1, 1, 0\}$; $L = 9 + 3 = 12$.

Итерация 5. $X_p = \{x_1, x_2, x_3, x_4, x_5\}$; $A_p = \{(x_1, x_2); (x_1, x_4); (x_1, x_3); (x_3, x_5)\}$; $\Gamma^*(X_p) = \{x_6\}$; $c(x_3, x_6^*) = 4$; $B = \{1, 1, 1, 1, 1, 1\}$; L = 12 + 4 = 16.

В результате получаем кратчайшее остовое дерево:

$X_p = \{x_1, x_2, x_3, x_4, x_5, x_6\}$; $A_p = \{(x_1, x_2); (x_1, x_4); (x_1, x_3); (x_3, x_5); (x_3, x_6)\}$.

Суммарная длина кратчайшего остового дерева L=16.

Результат решения задачи представлен на рисунке 3, б).

3. ПРОГРАММА ВЫПОЛНЕНИЯ ИССЛЕДОВАНИЙ

1. Получить задание у преподавателя в виде исходного неориентированного графа. Использовать данные матрицы из приложения Б, построив на её оснговании симметричную.

2. Составить схему алгоритма программы, определяющей кратчайшее остовое дерево графа с помощью алгоритма Прима-Краскала.

3. Использовать программу, реализующую алгоритм Прима-Краскала, приведённую в приложении А.

4. Дополнить программу операторами подсчёта количества элементарных операций.

5. Выполнить прогон программы для заданного графа. Сравнить результаты: фактическое количество операций и расчёты по функции трудоёмкости.

6. Сделать выводы, оформит отчёт.

4. СОДЕРЖАНИЕ ОТЧЁТА

1. Исходное задание и цель работы.
2. Структурная схема программы по п.2.
3. Распечатка текста программы по п.3.
4. Контрольный пример и результаты машинного расчета.
5. Выводы по работе.

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

1.Дайте определение дерева; ориентированного дерева.

2.Какое дерево называется остовым?

3. Свойства остовых деревьев. Теорема Кэли.

4. Что называется корнем дерева?

5. Как преобразовать неориентированное дерево в ориентированное?

6. Сколько ребер содержит остовое дерево графа?

7. Задача нахождения кратчайшего остова графа.

8. Приведите практические примеры нахождения кратчайшего остова графа.

9. Реализация алгоритма Прима-Краскала для нахождения кратчайшего остова графа.

ЛАБОРАТОРНАЯ РАБОТА № 7.
“АНАЛИЗ АЛГОРИТМОВ СОРТИРОВКИ”

1. ЦЕЛЬ РАБОТЫ

- Экспериментально сравнить между собой алгоритмы сортировки по критериям сложности и количества операций.
- Получить навыки экспериментальной оценки сложности и количества операций для алгоритмов сортировки.

2. КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Задача сортировки последовательности вельми богата существующими алгоритмами её решения. В данной работе предлагается рассмотреть и реализовать некоторые из алгоритмов сортировки и сравнить их производительность.

Приведём формальную постановку задачи сортировки.

Определение. Задача сортировки заключается в построении алгоритма со следующими входом и выходом:

ВХОД: последовательность $A = (a_1, a_2, a_3, \dots, a_n)$ сравнимых элементов некоторого множества.

ВЫХОД: перестановка $A = (a_1^{\nabla}, a_2^{\nabla}, a_3^{\nabla}, \dots, a_n^{\nabla})$, в которой $a_1^{\nabla} \blacklozenge a_2^{\nabla} \blacklozenge a_3^{\nabla} \blacklozenge \dots \blacklozenge a_n^{\nabla}$, где знак ♦ обозначает отношение “≤” и, в этом случае говорят о сортировке по возрастанию, либо “≥”, когда сортировка производится по убыванию.

Для проведения исследований будем использовать следующие алгоритмы.

2.1 Сортировка вставками

В данном алгоритме используется идея: предполагается, что массив уже отсортирован до некоторого элемента с номером i, и тогда просто необходимо вставить очередной элемент на необходимое место.

2.2 Сортировка прямым выбором

Идея сортировки выбором заключается в поиске максимального элемента в неупорядоченной части последовательности и его последующего исключения путём записи в конец массива.

2.3 Сортировка пузырьком

При сортировке пузырьком в последовательности сравниваются соседние элементы и, если они не упорядочены, меняются местами. Смена мест может быть организована по-разному, так же могут выполняться проверки на упорядоченность массива. Исходя из последних соображений существуют различные модификации алгоритма.

2.4 Сортировка слиянием

Данный алгоритм сортировки относится к классу «разделяй и властвуй». Массив будет делиться на две половины, а затем функция сортировки будет вызывать саму себя для сортировки каждой из половин. После этого нам необходимо совместить два уже отсортированных массива, что можно сделать с помощью двух указателей.

2.6 Сортировка Шелла

Данная сортировка является модификацией сортировки вставки, но с использованием шага, то есть рассматриваются не все элементы, а только те, которые находятся на расстоянии шага. Шаг постоянно уменьшается в два раза, что позволяет добиться сложности $O(\log(N))$.

2.7 Пирамидальная сортировка

Пирамидальная сортировка основывается на построение кучи (пула, пирамиды) – бинарного дерева, в котором элемент каждого узла больше, чем элементы его потомков. После построение такого дерева несложно найти отсортированный массив используя алгоритм слияния.

Коды программ сортировок приведены в приложении А.

3. ПРОГРАММА ВЫПОЛНЕНИЯ ИССЛЕДОВАНИЙ

1. Для каждого из приведённых алгоритмов найдите в литературе оценку для количества шагов и количества требуемой памяти.
2. Создайте структуру и модернизируйте программы реализации соответствующих алгоритмов сортировки согласно варианту задания. Направление сортировки указано в варианте.
3. Дополните алгоритмы операторами контроля числа операций, выполняемых в ходе сортировки, пользуясь средствами операционной системы,

определите объёмы программного кода, соответствующие исследуемым алгоритмам.

4. Выполните вычислительные эксперименты
5. Сравните производительность различных алгоритмов.

4. ВАРИАНТЫ ЗАДАНИЙ

	Простейший алгоритм 1	Простейший алгоритм 2	Быстрый алгоритм 3	Структура и критерий сортировки
1	Метод «пузырька»	Сортировка вставками	Сортировка Шелла	Структура дата, содержит день, месяц и год, сортировка по возрастанию дат
2	Модифицированный метод «пузырька»	Сортировка посредством выбора	«Быстрая» сортировка	Структура дата, содержит день, месяц и год, сортировка по убыванию дат
3	Сортировка вставками	Метод «пузырька»	Сортировка слиянием	Структура Студент, содержит ФИО, курс, факультет, сортировка по возрастанию курса
4	Сортировка посредством выбора	Модифицированный метод «пузырька»	Быстрая сортировка	Структура Студент, содержит ФИО, курс, факультет, сортировка по убыванию курса
5	Метод «пузырька»	Сортировка вставками	Пирамидальная сортировка	Структура Время, содержит часы, минуты, секунды, сортировка по возрастанию времени
6	Модифицированный метод «пузырька»	Сортировка посредством выбора	Сортировка Шелла	Структура Время, содержит часы, минуты, секунды, сортировка по убыванию времени
7	Сортировка вставками	Метод «пузырька»	«Быстрая» сортировка	Структура Сотрудник, содержит ФИО, профессия, ЗП. Сортировка по возрастанию ЗП.
8	Сортировка посредством выбора	Модифицированный метод «пузырька»	Сортировка слиянием	Структура Сотрудник, содержит ФИО, профессия, ЗП. Сортировка по убыванию ЗП.
9	Метод «пузырька»	Сортировка вставками	Сортировка Шелла	Структура студент, содержит ФИО, группа. Сортировка по ФИО.

	Простейший алгоритм 1	Простейший алгоритм 2	Быстрый алгоритм 3	Структура и критерий сортировки
10	Модифицированный метод «пузырька»	Сортировка посредством выбора	Пирамидальная сортировка	Структура дробь, содержит числитель и знаменатель, сортировка по возрастанию дробей.
11	Сортировка вставками	Метод «пузырька»	Сортировка Шелла	Структура дробь, содержит числитель и знаменатель, сортировка по убыванию дробей.
12	Сортировка посредством выбора	Модифицированный метод «пузырька»	«Быстрая» сортировка	Структура Продукт, содержит название и стоимость, сортировка по возрастанию стоимости.
13	Метод «пузырька»	Сортировка вставками	Сортировка слиянием	Структура Продукт, содержит название и стоимость, сортировка по убыванию стоимости.
14	Модифицированный метод «пузырька»	Сортировка посредством выбора	Быстрая сортировка	Структура Продукт, содержит название и стоимость, сортировка по названию.
15	Сортировка вставками	Метод «пузырька»	Пирамидальная сортировка	Структура Товар, содержит название, стоимость и код, сортировка по возрастанию кода товара.
16	Сортировка посредством выбора	Модифицированный метод «пузырька»	Сортировка Шелла	Структура Товар, содержит название, стоимость и код, сортировка по убыванию кода товара.
17	Метод «пузырька»	Сортировка вставками	«Быстрая» сортировка	Структура Товар, содержит название, стоимость и код, сортировка по возрастанию стоимости.
18	Модифицированный метод «пузырька»	Сортировка посредством выбора	Сортировка слиянием	Структура Товар, содержит название, стоимость и код, сортировка по убыванию стоимости.
19	Сортировка вставками	Метод «пузырька»	Сортировка Шелла	Структура заказ, содержит день, месяц, год, а также часы и минуты заказа, сортировка по возрастанию времени заказа.

	Простейший алгоритм 1	Простейший алгоритм 2	Быстрый алгоритм 3	Структура и критерий сортировки
20	Сортировка посредством выбора	Модифицированный метод «пузырька»	Пирамидальная сортировка	Структура заказ, содержит день, месяц, год, а также часы и минуты заказа, сортировка по убыванию времени заказа.

5. СОДЕРЖАНИЕ ОТЧЁТА

1. Цель работы.
2. Вариант задания.
3. Текст программы, реализующей расчеты по соответствующему варианту.
4. Анализ результатов работы программы в виде таблицы результатов и графиков.
5. Развернутый вывод по работе.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Для элементов каких множеств можно корректно поставить задачу сортировки?
2. Что такое стабильная сортировка?
3. Что такое беспорядок?
4. Какова сложность оптимального алгоритма сортировки в худшем случае?
5. Всегда ли существует решение задачи сортировки?
6. Единственно ли решение задачи сортировки?
7. Как работает встроенная сортировка std::sort?
8. К какому классу относятся функции трудоёмкости алгоритмов сортировки?
9. Как проявляют себя алгоритмы сортировки с точки зрения асимптотического анализа?

ЛАБОРАТОРНАЯ РАБОТА № 8.
“ИССЛЕДОВАНИЕ АЛГОРИТМОВ ДЛЯ НАХОЖДЕНИЯ НАИМЕНЬШЕГО ОБЩЕГО ДЕЛИТЕЛЯ (НОД) ЦЕЛЫХ ЧИСЕЛ”

1. ЦЕЛЬ РАБОТЫ

- Исследовать алгоритмы нахождения НОД на основании пооперационного анализа.
- Сравнить результаты вычислительных экспериментов с функциями трудоёмкости.
- Закрепить практические навыки исследование трудоёмкости алгоритмов.

2. КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

2.1 Алгоритма Евклида

Пусть $a \in Z$, $b \in N$, и необходимо найти d = НОД (a, b). Производится деление a на b с остатком:

$$a = q_0 b + r_0, 0 \le r_0 \le b.$$

Если $r_0 > 0$, то b делится на r_0:

$$b = q_1 r_0 + r_1, 0 \le r_1 \le r_0.$$

Получается последовательность равенств с убывающими остатками r_j:

$$r_{j-2} = q_j \cdot r_{j-1} + r_j, j = 0, 1, 2, \ldots \quad (1)$$

где $r_{-2} = a$, $r_{-1} = b$. Если r_k — последний ненулевой остаток, то $r_{k-1} = q_{k+1} r_k$.

Тогда справедливо равенство d =НОД (a, b) = r_k. Вычислительный процесс по рекуррентной формуле (1) носит имя алгоритма Евклида.

2.2 Бинарный алгоритм

На входе алгоритма заданы числа a, $b \in N$, $a \ge b$.

На выходе: d = НОД (a, b).

1-й шаг. Присвоить r остаток от деления a на b. Затем $a := b$, $b := r$.

2-й шаг. Если $b = 0$, то выдать значение a и закончить работу. Иначе положить k := 0, и затем, пока a и b оба чётные, выполнять циклическую процедуру:

$k := k + 1$,
$a := a/2$,
$b := b/2/$

После этого 2^k идет в НОД (a, b), оставшаяся часть НОД будет нечетной.

3-й шаг. По крайней мере, одно из чисел a и b нечетное. Если a четное, то делать присвоение $a := a / 2$ до тех пор, пока a не станет нечетным.
То же выполнять для b.

4-й шаг. a и b оба нечетные. Присвоить $t := (a – b) /2$. Если $t = 0$, то выдать $2^k a$ и закончить работу.

5-й шаг. До тех пор, пока t чётное число, выполнять $t := t/2$.

6-й шаг. t нечетное, $t \neq 0$. Если $t > 0$, то $a := t$. Если $t < 0$, то $b := – t$. Идти на 4-й шаг.

Конец алгоритма

2.3 Примеры решения задач лабораторной работы

Даны два натуральных числа a и b, не равные нулю одновременно. Вычислить НОД (a, b) – наибольший общий делитель a и b.

```
int gcd(int a, int b) // функция, возвращающая НОД двух чисел
{
     intk = min(a, b);// НОД не может быть больше минимума из них
     for(; a % k || b % k; k--); // перебрать все числа, пока не
                                 // найдено нужное
     return k;
}
```

Рисунок 1 – Листинг программы НОД

Рассмотрим решение данной задачи с помощью алгоритма Эвклида:

Будем считать, что НОД (0, 0) = 0.

Тогда НОД(a, b)=НОД($a – b$, b)=НОД(a, $b – a$); НОД (a, 0)=НОД(0, a) = a для всех a, b > = 0.

Реализация алгоритма показана на рисунке 2.

Заметим, что данный алгоритм можно улучшить, заменив оператор $b – = a$ на b % = a. Действительно, вычитая постоянно a, в итоге получится остаток от деления на него. В этом случае сложность алгоритма будет $O(\log(\max(a, b)))$. Данную функцию рекомендуется реализовать самим.

Благодаря алгоритму Эвклида можно за такую же сложность найти наименьшее общее кратное – НОК (a, b).

Очевидно, что НОК (a, b) = a*b / НОД(a, b).

```
int gcd(int a, int b)
{
      if(b < a)
            swap(a, b); // b > a
      while(a) {// пока a != 0, в этом случае b будет являться
                  //ответом
            b -= a;    // уменьшить b на a
            if(a>b)
                  swap(a, b); // bвсегда должно быть больше a
      }
      returnb;
}
```

Рисунок 2 – Листинг программы алгоритма Эвклида

3. ПРОГРАММА ВЫПОЛНЕНИЯ ИССЛЕДОВАНИЙ

1.Ознакомиться с теоретическим разделом настоящих методических указаний и повторить соответствующий лекционный материал.

2.Разработать алгоритм нахождения НОД или НОК в соответствии с вариантом задания.

3. Построить методом пооператорного анализа функцию трудоёмкости полученного алгоритма.

4. Дополнить программный код средствами подсчёта элементарных операций.

5. Провести серию вычислительных экспериментов. Сравнить результаты выполняемых операций с прогнозом, полученным в ходе асимптотического анализа функции трудоёмкости.

4. Сделать выводы по работе, оформить отчет, подготовить ответы на контрольные вопросы.

4. ВАРИАНТЫ ЗАДАНИЙ

№	Условие
1	Написать алгоритм Евклида, использующий соотношения НОД(a, b) = НОД(a mod b, b) при $a >= b$; НОД (a, b) = НОД (a, b mod a) при $b >= a$
2	Написать алгоритм Евклида, использующий соотношения НОД (a, b) = НОД (a, b mod a) при $b >= a$
3	Даны натуральные числа a и b, не равные нулю одновременно. Найти d = НОД (a, b) и такие целые x и y, что $d = a * x + b * y$
4	Даны натуральные числа a и b, не равные нулю одновременно. Найти d = НОД (b, a) и такие целые x и y, что $d = a * x + b * y$
5	Даны натуральные числа a и b, не равные нулю одновременно. Найти НОД (b, a) и НОК (b, a)

№	Условие
6	Даны натуральные числа a и b, не равные 0 одновременно. Найти d = НОД (a, b) и такие целые x и y, что $d = a * x + b / y$.
7	Написать вариант алгоритма Евклида, использующий соотношения НОД(2*a, 2*b) = 2 * НОД (a, b)
8	Написать вариант алгоритма Евклида, использующий соотношения НОД(2*a, b) = НОД (a, b) при нечетном b
9	Написать бинарный алгоритм, использующий соотношения НОД (a, b) = НОД(a mod b, b) при $a >= b$ НОД (a, b) = НОД (a, b mod a) при $b >= a$
10	Даны натуральные числа a и b, не равные 0 одновременно. Найти d = НОД (a, b) и такие целые x и y, что $d = a * x + b * y$, используя бинарный алгоритм

5. СОДЕРЖАНИЕ ОТЧЁТА

1. Цель работы.
2. Вариант задания.
3. Текст программы, реализующей расчеты по соответствующему варианту.
4. Анализ результатов работы программы в виде таблицы результатов и графиков.
5. Развернутый вывод по работе.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Дайте формальное определение алгоритма поиска НОД.
2. Дайте формальное определение алгоритма поиска НОК.
3. Поясните основные этапы алгоритма Эвклида.
4. Сравните сложности алгоритма Эвклида и простого алгоритма поиска НОД двух чисел.
5. Как найти НОК (a, b) при помощи алгоритма Эвклида?
6. При каких значениях a и b алгоритм будет работать быстрее всего?
7. При каких значениях a и b алгоритм будет работать дольше всего?
8. Откуда возник логарифм в оценке $O(\log(\max(a, b)))$ алгоритма НОД?

ЗАКЛЮЧЕНИЕ

"*Если бы наука сама по себе не приносила никакой практической пользы, то и тогда нельзя было назвать её бесполезной, лишь бы только она изощряла ум и наводила в нём порядок*"

Ф. Бэкон

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

Книжные издания

1. Агафонов В. Н. Сложность алгоритмов и вычислений / В. Н. Агафонов. Новосибирск : НГУ, 2010. – 70 с.
2. Ахо А. В. Построение и анализ вычислительных алгоритмов / А. В. Ахо, Дж. Е. Хопкрофт, Дж.Д. Ульман. – М. : Мир, 1979. – 536 с.
3. Глушков В. М. Алгебра. Языки, Программирование / В. М. Глушков, Г. Е. Цейтлин, Е. Л. Ющенко. – К. : Наукова думка, – 318 с.
4. Грин Д. Математические методы анализа алгоритмов / Д. Грин, Д. Кнут. – М. : Мир, 1987. – 120 с.
5. Кнут Д. Искусство программирования. / Д. Кнут. т.1, т.2. – М. : Техносфера, 2001. – т.1 – 682 с., т.2 – 788 с.
6. Кормен Т. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест. М. : – МЦИМО, 2001. – 960 с.
7. Левитин А. В. Алгоритмы: введение, разработка и анализ / А. В. Левитин. – М. : Вильямс, 2006. – 576 с.
8. Макконнелл Дж. Основы современных алгоритмов / Дж. Макконнелл. – М. : Техносфера, 2004. – 368 с.
9. Мальцев А. И. Алгоритмы и рекурсивные функции. – М. : Наука, 1986. – 368 с.
10. Рейнгольд Э. Комбинаторные алгоритмы: Теория и практика / Э. Рейнгольд, Ю. Нивергельт, Н. Део. – М. : Мир, 1980. – 468 с.
11. Ульянов М. В. Математическая логика и теория алгоритмов, часть 2: Теория алгоритмов / М. В. Ульянов, М. В. Шептунов. – М. : МГАПИ, 2003. – 80 с.
12. Ху Т.Ч., Шинг М. Т. Комбинаторные алгоритмы / Т. Ч. Ху, М. Т. Шинг. — Нижний Новгород : Изд-во Нижегородского госуниверситета им. Н. И. Лобачевского, 2004. — 330 с.

Электронные ресурсы

1. Игошин, В. И. Теория алгоритмов : учебное пособие / В. И. Игошин. – Москва : ИНФРА-М, 2012. - 318 с. (Высшее образование). - URL: http://znanium.com/catalog/product/241722 (дата обращения: 28.06.2022). — ISBN 978-5-16-005205-2. - Текст: электронный. — Режим доступа: для авториз. пользователей.
2. Игошин, В. И. Сборник задач по математической логике и теории алгоритмов: учеб. пособие/ В. И. Игошин. — Москва : КУРС : ИНФРА-М, 2017. — 392 с. — (Бакалавриат). - URL: http://znanium.com/catalog/product/524332 (дата обращения: 28.06.2022). - ISBN 978-5-906818-08-9. - Текст: электронный. — Режим доступа: для авториз. пользователей.
3. Крупский, В. Н. Теория алгоритмов. Введение в сложность вычислений: учебное пособие для вузов / В. Н. Крупский. — 2-е изд., испр. и доп. — Москва : Юрайт, 2022. — 117 с. — (Высшее образование). — ISBN 978-5-534-04817-9. — Текст: электронный // Образовательная платформа Юрайт [сайт]. — URL: https://www.urait.ru/bcode/492937 (дата обращения: 28.06.2022). — Режим доступа: для авториз. пользователей.
4. Пруцков, А. В. Математическая логика и теория алгоритмов : учебник / А. В. Пруцков, Л. Л. Волкова. — Москва : КУРС : ИНФРА-М, 2017. — 152 с. - URL: http://znanium.com/catalog/product/773373 (дата обращения: 28.06.2022). — Режим доступа: для авториз. пользователей. — ISBN 978-5-906818-74-4. — Текст: электронный.

1. Судоплатов, С. В. Математическая логика и теория алгоритмов: учебник и практикум для вузов / С. В. Судоплатов, Е. В. Овчинникова. — 5-е изд., стер. — Москва : Юрайт, 2022. — 207 с. — (Высшее образование). ISBN 978-5-534-12274-9. — Текст: электронный // Образовательная платформа Юрайт [сайт]. — URL: https://www.urait.ru/bcode/447321 (дата обращения: 28.06.2022). — Режим доступа: для авториз. пользователей.
2.

ПРИЛОЖЕНИЕ А
(обязательное)
ТЕКСТЫ УЧЕБНЫХ ПРОГРАММ

1. Пример программы интервального анализа

Определения интервалов предпочтения алгоритмов для функций трудоемкости $F(n) = 1.75 * n^2$ и $G(n) = 18 * n * \ln(n)$

```
/* подключение основных используемых в программе библиотек*/
#include …
…

main()
{
double
        Fn,             //F(n)
        Gn,             //G(n)
        ATg_FG,
        ATg_GF,
        pi,
        Nbegin,     //Левая граница интервала
        Nend,               //Правая граница интервала
        step,       //Шаг изменения аргумента
        phi,        //Значение угла
        k,          //Коэффициент кратности
        Delta,      //Оценка «Дельта»
        Theta,      //Оценка «Тетта»
        O_large,    //Оценка «О-большое»
        ii;         //Значение аргумента функций трудоемкости

        FILE *stream;
              //Указатель на файл, в который осуществляется ввод-вывод
              расчетов

        stream = fopen("Example_TA.TXT", "w+"); // открытие файла для
записи
              /*Ввод значений границ интервалов, шага изменения
аргумента внутри интервала, коэффициента кратности*/

        cout<<"Input Nbegin  ";     cin>>Nbegin; //Левая граница, ввод
значения
        cout<<"Input Nend    ";     cin>>Nend;     //Правая граница, ввод
значения
        cout<<"Input step    ";     cin>>step;     //Шаг изменения аргумента
```

```
        cout<<"Input koefficient";  cin>>k;          //Коэффициент кратности

        phi = M_PI/k;               /*Определили угол изменения как π/k, M_Pi –
встроенная константа яз.С, число π = 3,14159…*/

     ii=Nbegin;                     //Аргумент функций равен левой границе
интервала
      while (ii<=Nend)
        {       Fn     = 1.75*ii*ii;  //Расчет значения функции F(n)
                Gn     = 18*ii*log(ii);      //Расчет значения функции G(n)
                ATg_FG     = atan(Fn/Gn);
                ATg_GF     = atan(Gn/Fn);
                pi      = ATg_FG - ATg_GF;
                Delta   = phi - pi;
                Theta= fabs(pi) - phi;
                O_large = pi + phi;
                fprintf(stream, "%f %f %f %f %f %f %f %f %f\n", ii, Fn, Gn,
ATg_FG, ATg_GF, pi, Delta, Theta, O_large);
                //Запись расчетов в файл
                ii=ii+step;           //Получение следующего значения аргумента
        }//end while
        fclose(stream);           //Закрыли файл
}// end main
```

2. Подсчёт операций переприсваивания

```
/* подключение основных используемых в программе библиотек*/
#include …
…

int vector[10];
// Создание массива из 10 псевдослучайных целых чисел  величиной от
0 до 100
// массив записывается в файл Example_TA2.TXT, на экран выводим
максимальное // целое

void create_array(int Nmax) {
 int i;
 FILE *stream;
 //Nmax = 10; соответствует размерности массива
 stream = fopen("Example_TA2.TXT", "w+");
 randomize();
 cout<<"Maximal integer  ";     cout<<RAND_MAX;
 printf("\n%d%s\n", Nmax, " random numbers from 0 to 100");
```

```
 for(i=0; i<Nmax; i++){
        vector[i]= rand() % 100;
        printf("%d\n",  vector[i]);
        fprintf(stream,"%d\n", vector[i]);
 }
 fclose(stream);
 }


void main(){

     int        i,
                N,
                min, // значение минимума
                cnt; //счетчик операций переприсваивания

     double  result;

     cout<<"Input amount of numbers"; cin>>N;
     result = harmonic(N);
                // harmonic(N) – функция подсчета n-го гармонического числа
        cout<<result;
     create_array(N);
                //генерация массива псевдослучайных чисел
        min = vector[0];
     cnt = 1;
     for (i=1;i<N;i++){
     if(vector[i]<min) {min = vector[i];        cnt++;}
     }
        printf("%s%d%s%d\n",  "Minimal ", min, " Num oper ", cnt);
     }
```

3. Решение задачи о сумме

```
/* подключение основных используемых в программе библиотек*/
 #include …
 …

 #define SIZE ;

 int vector[4];
 int counter[4];
 int N=4, V=5;
 int flag;
 FILE *stream;
```

```
void TaskSum(void){
  int i,j,sum, cnt, MaxN;
  MaxN = int(pow(2,N)-1);

  flag = 0; i=0;
  while(i<N){ counter[i]=0;  i++; }//end while(i<N)
  cnt = 1;
  do{
    sum = 0;
    i=0;
    while(i<N){
      sum = sum+counter[i]*vector[i];
      i++; }//end while (i<N)
    if (sum== V) {
      flag=1;
      return;
    }//end if
    j=N-1;
    while((counter[j]==1)&&(j!=0)){
      counter[j]=0;
      j = j-1;
    }//end while counter[j]==1
    counter[j]=1;
    cnt = cnt + 1;
  }//end do
  while(cnt<=MaxN);
}//end TaskSum

void main(){
      int        i,min,cnt, power, sum,j, temp, c;
      double  result;
      for(i=0; i<N; i++){
         if (fscanf(stdin, "%d", &temp))
           printf("The integer read was: %i\n",temp);
         vector[i]=temp;
      }
      fclose(stream);
      flag=0;
      TaskSum();
      if (flag==1){
       cout<<"OK\n";
       for(int k = 0; k<N; k++) printf("%d%s",counter[k]," ");
       printf("%s\n"," ");
       for(k = 0; k<N; k++) printf("%d%s",vector[k]," ");
```

```
      }//end if
     else cout<<"NO elements giving the sum\n";
}//end main
```

4. Алгоритм Дейкстры

```
void solve() {
        // n - количество вершин
        // g[n][n] - матрица смежности, g[i][j] = 0, если текущего ребра нет
        // d[n] - массив ответов
        // s - стартовая вершина
        // used - массив для пометок вершин
        int i,j, // i-я итерация, j - для поиска минимальной
                v, // минимальная вершина
                to, // ребро из вершины v в to
                len; // длины len
        d[s] = 0;
        for(i = 0; i < n; i++) {
                v = -1;
                for(j = 0; j < n; j++)  // поиск вершины с минимальным d[j]
                        if(!used[j] && (v == -1 || d[j] < d[v]))
                                v = j;
                used[v] = true; // пометка вершины
                for(to = 0; to < n; to++) {
                        if(g[v][to]) {
                                len = g[v][to];
                                if(d[v] + len < d[to]) {
                                        d[to] = d[v] + len;
                                }
                        }
                }
        }
}
```

5. Алгоритм метода динамического программирования

```
const int INF = 1000000000; // бесконечность
bool used[100] = {0};   // массив для пометок
int top[100] = {0};     // топологический список
int g[100][100] = {0};  // матрица смежности
int n;                          // количество вершин
int l;                          // l-я вершина для добавления
int s;                          // стартовая вершина
```

```
int f;                                  // конечная вершина
intd[100] = {0};                // массивответов

int dfs(intv) {
        if(used[v])
                return 0;
        used[v] = true;
        for(int to = 0; to < n; to++)
                if(g[v][to])
                        dfs(to);
        top[l++] = v; // добавление вершины v в отсортированный список
}

int topSort() {
        l = 0; // номер добавляемой вершины в отсортированный список
        for(int i = 0; i < n; i++)
                dfs(i); // запустить пробежку из всех вершин
        reverse(top, top+l); // развернутьмассив
        return 0;
}

int solve() {
        int i, j;
        for(i = 0; i < n; i++)
                d[i] = INF;
        d[s] = 0;
        for(i = 1; i < n; i++)
                for(j = 0; j < i; j++)
                        if(g[top[j]][top[i]])
                                d[top[i]] = min(d[top[i]], d[top[j]] + g[top[j]][top[i]]);
        return 0;
}
```

6. Алгоритмы сортировки

6.1. Сортировка вставками

```
void insertSort(int *a, int n)
{
        int i, j;
        for(i = 2; i <= n; i++) {   // установить i-й элемент на свое место
                a[0] = a[i];            // запомнить i-й элемент в нулевой ячейке
                for(j = i-1; a[j] > a[0]; j--)// пока j-ый элемент больше текущего
                        a[j+1] = a[j];          // сдвигать его влево
```

```
            a[j+1] = a[0];      // на свободное место вернуть исходный элемент
        }
}
```

6.2. Сортировка прямым выбором

```
void selectSort(int *a, int n)
{
        int i, j, imax;
        for(i = n; i > 1; i--) { // заполнение i-йячейки
                imax = 1;
                for(j = 1; j <= i; j++) // поиск максимального элемента
                        if(a[j] > a[imax])
                                imax = j;
                swap(a[i], a[imax]);//вставка максимального элемента на свое место
        }
}
```

6.3. Сортировка пузырьком

```
void bubbleSort(int *a, int n)
{
        int i, j;
        for(i = n; i > 1; i--)
                for(j = 1; j < i; j++)          // выталкивается самый тяжелый
                        if(a[j] > a[j+1])       // если он больше своего соседа справа
                                swap(a[j], a[j+1]);
}
```

6.4. Сортировка слиянием

```
void merge(int *a, int l, int r) {
        int m = (l+r) / 2;
        int i, j, k;

        // b - промежуточный массив, объявленный как глобальная переменная

        i = l;  // начало первой половины
        j = m+1; // начало второй половины
        k = l;
```

```
        while(i <= m || j <= r) { // пока в массив не добавятся все элементы
                if(i > m) { // если в первой половине чисел больше нет
                        b[k++] = a[j++]; // добавляется число из второй половины
                        continue;
                }
                if(j > r) { // если во второй половине чисел больше нет
                        b[k++] = a[i++]; // добавляется число из первой
                        continue;
                }
                if(a[i] < a[j]) // если в первой половине минимальное число
меньше //минимального во второй
                        b[k++] = a[i++]; // добавляем число из первой
                else  // иначе
                        b[k++] = a[j++]; // из второй
        }
        for(i = l; i <= r; i++) // возвращение данных из промежуточного
массива //в исходный
                a[i] = b[i];
}
void mergeSort(int *a, int l, int r) {
        if(l == r) // если один элемент, то он отсортирован
                return;
        int m = (l + r) / 2; // найти середину
        mergeSort(a, l, m);       // отсортивать первую половину
        mergeSort(a, m+1, r);   // отсортировать вторую половину
        merge(a, l, r);      // совместить две половины, не теряя свойства
        //возрастания
}
```

6.5. Быстрая сортировка

```
void quickSort(int *a, int n) {
        int i = 0, j = n;               // поставить указатели на исходные места
        int temp, p;

        p = a[n/2];           // центральный элемент

        // процедураразделения
        do {
                while (a[i] < p)
                        i++;
                while (a[j] > p)
                        j--;
```

```
            if (i <= j) {
                swap(a[i], a[j]);
                i++;
                j--;
            }
        }
        while(i <= j);
        // рекурсивные вызовы, если есть, что сортировать
        if(j > 0)
            quickSort(a, j);
        if(n > i)
            quickSort(a+i, n-i);
}
```

6.6. Сортировка Шелла

```
void shellSort(int *a, int n) {
        int i, j, temp, step = n / 2;//инициализацияшага
        while (step > 0) { //покашагненулевой
            for (i = 0; i < (n - step); i++) {
                j = i;
//будем идти начиная с i-го элемента
                while (j >= 0 && a[j] > a[j + step]) {
                    //пока не достигнуто начало массива
                    //и пока рассматриваемый элемент больше
                    //чем элемент находящийся на расстоянии шага
                    //элементы меняются местами
                    swap(a[j], a[j+step]);
                    j--;
                }
            }
       step = step / 2;//уменьшение шага
        }
```

6.7. Пирамидальная сортировка

```
void heapSort(int *a, int n) {
        int i, sh = 0; //смещение
bool b = false;
for(;;) {
            b = false;
            for (i = 0; i < n; i++ ) {
                if( i * 2 + 2 + sh < n ) {
                    if( ( a[i + sh] > a[i * 2 + 1 + sh] ) || ( a[i + sh] >  a[i * 2 +
2 + sh] ) ) {
```

```
                        if ( a[i * 2 + 1 + sh] < a[i * 2 + 2 + sh] ) {
                                swap( a[i + sh], a[i * 2 + 1 + sh] );
                                b = true;
                        }
                        else
                        if ( a[i * 2 + 2 + sh] < a[ i * 2 + 1 + sh]) {
                                swap( a[ i + sh], a[i * 2 + 2 + sh]);
                                        b = true;
                                }
                        }
                        //дополнительная проверка для последних двух
элементов
//с помощью этой проверки можно отсортировать пирамиду
//состоящую всего лишь из трех элементов
                        if( a[i*2 + 2 + sh] < a[i*2 + 1 + sh] ) {
                                swap( a[i*2+1+sh], a[i * 2 +2+ sh] );
                                b = true;
                        }
                }
                else
                        if( i * 2 + 1 + sh < n ) {
                                if( a[i + sh] > a[ i * 2 + 1 + sh] ) {
                                        swap( a[i + sh], a[i * 2 + 1 + sh] );
                                        b = true;
                                }
                        }
        }
        if (!b)
                sh++; //смещение увеличивается, когда на текущем
этапе
        //сортировать больше нечего
        if ( sh + 2 == n ) break;
    } //конец сортировки
}
```

7. Определение общего делителя

```
Int gcd(inta, intb) {       // функция, возвращающая НОД двух чисел
    intk = min(a, b);   // НОД не может быть больше минимума из них
    for(; a % k || b % k; k--); // перебрать все числа, пока не
найдено//нужное
    return k;
}

int gcd(int a, int b) {
```

```
        if(b < a)
                swap(a, b); // b > a
        while(a) {    // пока a != 0, в этом случае b будет являться ответом
                b -= a;   // уменьшитьbнаа
                if(a>b)
                        swap(a, b); // bвсегда должно быть больше a
        }
        returnb;
}
```

8. Алгоритм Прима-Краскала

```
const int INF = 1000000000; // бесконечность
bool used[100] = {0};   // массив для пометок
int g[100][100] = {0};  // матрицасмежности
int n;                  // количествовершин


int solve() {
        int i, j, v, to;
        vector <int> min_e (n, INF), sel_e (n, -1);
        min_e[0] = 0;

        for(i = 0; i < n; i++) {
                v = -1;
                for (j = 0; j < n; j++)
                        if(!used[j] && (v == -1 || min_e[j] < min_e[v]))
                                v = j;
                if (min_e[v] == INF) {
                        cout <<"No MST!";
                        exit(0);
                }
                used[v] = true;
                if(sel_e[v] != -1)
                        cout << v <<" "<< sel_e[v] << endl;
                for(to = 0; to < n; to++)
                        if (g[v][to] < min_e[to]) {
                                min_e[to] = g[v][to];
                                sel_e[to] = v;
                        }
        }
        return 0;
}
```

Приложение Б
(обязательное)
ВАРИАНТЫ ТОПОЛОГИИ ГРАФОВ

Пояснения к вариантам.

1. Обозначения: «откуда», «куда» – узлы выхода и входа ребер ориентированного графа; цифры в таблице – веса ребер.

2. Для неориентированного графа элементы таблицы следует отобразить относительно главной диагонали

3. Пример ориентированного графа, соответствующего по варианту 1, дан ниже: X_i – узлы (вершины) графа, цифры на соединительных линиях – веса соответствующих рёбер.

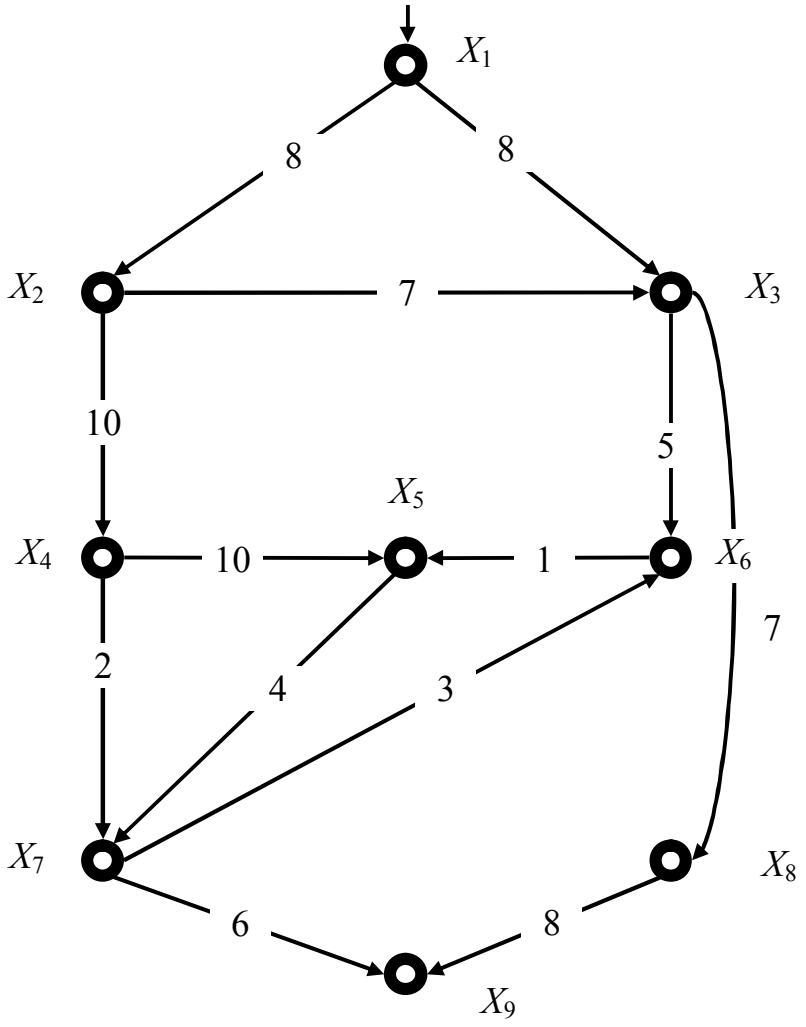


Рисунок Б.1 – Ориентированный граф к варианту 1

Вариант 1

Откуда \ Куда	1	2	3	4	5	6	7	8	9
1		8	8						
2			7	10					
3						5		7	
4					10		2		
5							4		
6					1				
7						3			6
8									5

Вариант 2

Откуда \ Куда	1	2	3	4	5	6	7	8	9
1		5	6						
2			3	8					
3					9	7		8	
4			3		2		6		
5							5		
6					7		10		
7									4
8									3

Вариант 3

Откуда \ Куда	1	2	3	4	5	6	7	8	9
1		2	1						
2			7	2					
3					2	3		6	
4			3				8		
5							4		
6					3				
7						2			10
8									1

Вариант 4

Откуда \ Куда	1	2	3	4	5	6	7	8	9
1		1	10						
2			3	3			7		
3					2	7		5	
4			2		4				
5							3		
6					2				
7						5			7
8									7

Вариант 5

Откуда \ Куда	1	2	3	4	5	6	7	8	9
1		4	10						
2			8						
3						8		3	
4					2		9		
5			6						
6				4	1			3	
7						1			5
8									7

Вариант 6

Откуда \ Куда	1	2	3	4	5	6	7	8	9
1		5	3						
2			4	2					
3						7	6	4	
4					10	5			
5							2	9	
6					7				
7						9			7
8									3

Вариант 7

Откуда \ Куда	1	2	3	4	5	6	7	8	9
1		9	4						
2			7	10					
3						4		6	
4			6		3				
5							1		
6					9			7	
7				1		8			6
8									4

Вариант 8

Откуда \ Куда	1	2	3	4	5	6	7	8	9
1		7	5						
2				3	9				
3		5			7	8		7	
4					7		4		
5							8		
6					3				
7						4			6
8									8

Вариант 9

Откуда \ Куда	1	2	3	4	5	6	7	8	9
1		9	1						
2			8	1					
3						4		3	
4			8		3		7		
5							6		
6					9			5	
7						6		1	4
8									1

Вариант 10

Откуда \ Куда	1	2	3	4	5	6	7	8	9
1		9	8						
2						5		1	
3		8							
4			9		1			4	
5							2		
6					8		7		
7				2					4
8									7

Вариант 11

Откуда \ Куда	1	2	3	4	5	6	7	8	9
1		8	6						
2			4						
3						5		8	
4					6		3		
5							10		
6				4	2		3		
7									9
8									5

Вариант 12

Откуда \ Куда	1	2	3	4	5	6	7	8	9
1		10	10						
2				1					
3				4		2		8	
4					3	4			
5							2		
6					9				
7				8					6
8									9

Вариант 13

Откуда \ Куда	1	2	3	4	5	6	7	8	9
1		7	3						
2			7		1		2		
3				1				4	
4					2		5		
5							10		
6					10				
7						5			6
8									4

Вариант 14

Откуда \ Куда	1	2	3	4	5	6	7	8	9
1		6	1						
2			8						
3						6			
4					3		1	1	
5							5		
6				2	4				
7						5			4
8									3

Вариант 15

Откуда \ Куда	1	2	3	4	5	6	7	8	9
1		5	6						
2			4	8					
3				2		7			
4					3		5		
5							2	5	
6					8			9	
7						5			1
8									5

Вариант 16

Откуда \ Куда	1	2	3	4	5	6	7	8	9
1		2	7						
2				8	3				
3								5	
4					4	1	3		
5							10	6	
6					6				
7						6			9
8									4

Вариант 17

Откуда \ Куда	1	2	3	4	5	6	7	8	9
1		3							
2				7	2				
3				1	2				
4					2		10	2	
5							8		
6					5				
7			3			2			10
8									6

Вариант 18

Откуда \ Куда	1	2	3	4	5	6	7	8	9
1		8	7						
2			3						
3						4		2	
4			8				3		
5							4		
6				8	2				
7						1			4
8									8

Вариант 19

Куда

Откуда	1	2	3	4	5	6	7	8	9
1		9	7						
2			5	4	4				
3						5		8	
4						1			
5				4		9	7		
6								3	
7				5		4			3
8									6

Вариант 20

Куда

Откуда	1	2	3	4	5	6	7	8	9
1		4	8						
2			2	1	4				
3						8	5	4	
4					2	10			
5							3	8	
6					8				
7						5			2
8									6

Заказ №		от “		“		2023		. Тираж		экз.
				Изд-во СевГУ						