Методические Указания

Для лабораторных работ
по дисциплине
«Анализ и проектирование алгоритмов»

Технический Университет Молдовы Департамент Программная Инженерия и Автоматика

Кишинёв, 2023

СОДЕРЖАНИЕ

ЛАБОРАТОРНАЯ РАБОТА № 1

Тема: Сравнение времени работы алгоритмов

Цель: Временной анализ функций

Краткие теоретические сведения

Для решения задачи часто приходится выбирать метод как из числа алгоритмов различных по принципу своей работы, так и из числа возможных реализаций одного алгоритма. За конечное число шагов, при разных исходных данных, все они приведут к правильному решению задачи. Но из всего спектра вариантов, следует выбирать оптимальные методы.

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

Выделяют *временную* и *пространственную* сложность. Временная сложность определяется количеством элементарных операций (инструкций), совершаемых алгоритмом для решения им поставленной задачи. Пространственная сложность измеряется объемом используемой алгоритмом памяти. Далее будет рассматриваться только временная сложность.

При анализе поведения функции трудоемкости алгоритма часто используют принятые в математике асимптотические обозначения, позволяющие показать порядок скорости роста временной функции, пренебрегая при этом конкретными значениями.

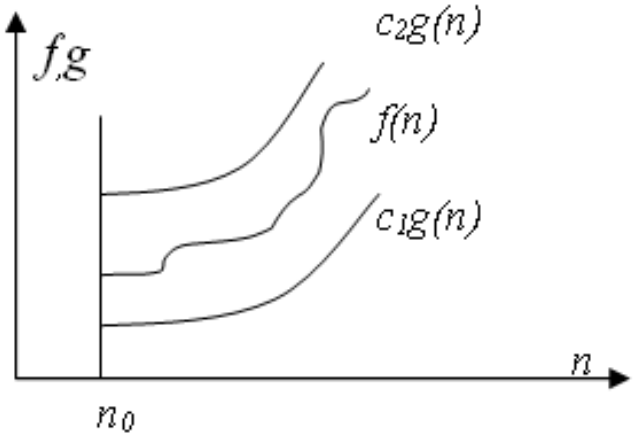
В асимптотическом анализе приняты следующие обозначения:

1 Асимптотически точная оценка θ (тетта)

Пусть $f(n)$ и $g(n)$ – положительные функции положительного аргумента, $n \geq 1$ (количество объектов на входе и количество операций – положительные числа), тогда:

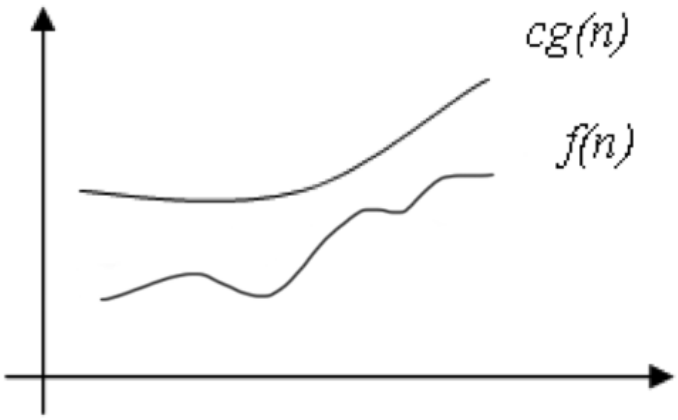
$f(n) = \theta(g(n))$, если существуют положительные $c_1 > 0, c_2 > 0$ и n_0 такие, что:
$c_1 g(n) \leq f(n) \leq c_2 g(n)$ при $\forall n \geq n_0$

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



2 Верхняя оценка O (омикрон)

В отличие от оценки θ, оценка O требует только, чтобы функция $f(n)$ не превышала $g(n)$ начиная с $\forall n \geq n_0$, с точностью до постоянного множителя:



$$\exists\ c > 0, n_0 > 0:$$

$$0 \leq f(n) \leq cg(n), \quad \forall n \geq n_0$$

Вообще, запись $O(g(n))$ обозначает класс функций, таких, что все они растут не быстрее, чем функция $g(n)$ с точностью до постоянного множителя, поэтому иногда говорят, что $g(n)$ мажорирует функцию $f(n)$.

Например, для всех функций:

$$f(n) = 1/n,$$

$$f(n) = 12,$$

$$f(n) = 3n + 17,$$

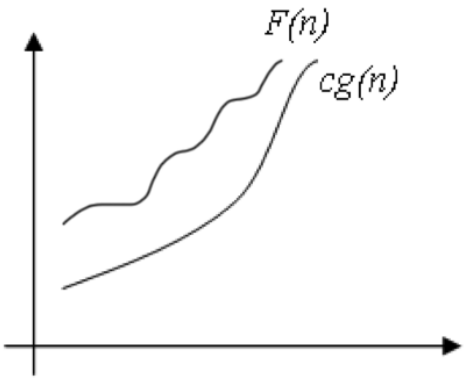
$$f(n) = n * ln(n),$$

$$f(n) = 6n^2 + 24n + 77$$

будет справедлива оценка $O(n^2)$, то есть все эти функции не будут расти быстрее чем n^2.

3 Нижняя оценка Ω (Омега)

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



$$\exists\ c > 0,\ n_0 > 0:$$

$$cg(n) \leq f(n) \leq \infty$$

при $\forall n \geq n_0$

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

Запись **f(n)** = O **(g(n))** означает, что с ростом **n** отношение **f(n)/g(n)** остаётся ограниченным.

Если к тому же

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

то пишем **f(n)** = ϭ **(g(n))** (читается «эф от эн есть ϭ малое от же от эн»).

Формально говоря, **f(n)** = ϭ **(g(n))**, если для всякого положительного ε > 0 найдётся такое n_0 что

0 ≤ f(n) ≤ εg(n) при ∀ n ≤ n_0.

Примеры:

$$1/3x^3 + 5x = O(x^3)$$

$$x^2 = o(x^3)$$

Задание:

I. Для алгоритма, решающий задачу размера n, определить время выполнения этой задачи. Найти его для функций, перечисленных в таблице и заполнить таблицу. Построить графики роста функции и расставить функции в порядке возрастания.

f(n) \ n	1	2	4	8	16	24
log n						
ln n						
$\sqrt{n}$						
n						
1/n						
n log n						
n^2						
n^3						
2^n						
n!						
n/ log n						
log log n						
log (n!)						
n^n						

II. Сравнение алгоритмов

Для всех клеток следующей таблицы ответьте «да» или «нет» на вопрос о том, можно ли записать A как $\dot{O}$, σ, Ω, ω или Θ от B ($k \geq 1$, $\varepsilon > 0$, c, $m > 1$ – некоторые константы). Приведите расчеты.

A	B	$\dot{O}$	σ	Ω	ω	Θ
$\mathbf{Log^k\, n}$	$\mathbf{n^{\varepsilon}}$					
$\mathbf{n^k}$	$\mathbf{c^n}$					
$\mathbf{\sqrt{n}}$	$\mathbf{n^{\sin n}}$					
$\mathbf{2^n}$	$\mathbf{2^{n/2}}$					
$\mathbf{n^{\log m}}$	$\mathbf{m^{\log n}}$					
$\mathbf{\log (n!)}$	$\mathbf{\log (n^n)}$					

Пример:

Дано:

$f(n) = n, g(n) = 2n^2.$

Определить асимптоту роста между функциями.

Решение:

Согласно определению $f(n) = \theta(g(n))$ записываем неравенство по формуле:

$c_1\, 2n^2 \leq n \leq 2n^2\, c_2$

Делим все неравенство на $2n^2$

$$c_1 \leq \frac{1}{2n} \leq c_2,$$

Выбираем какое-либо $n_0 > 0$.

Пусть $n_0 = 1$.

Находим c_2:

$c_2 = 1/2$,

при $\forall n \geq n_0\ c_1 \rightarrow 0$, поэтому выбираем оценку для $\dot{O}$

$n = \dot{O}(2n^2)$.

Проверяем для σ малое, применяя свойства предела

$$\lim_{(n->\infty)} \frac{n}{2n^2} = 0.$$

Получаем оценку для σ малое

$n = \sigma\,(2n^2)$.

Результаты записываем в таблицу.

A	B	O'	o'	Ω	ω	Θ
n	$2n^2$	+	+	–	–	–

III. Сравнение скорости роста

Расположите следующие функции в порядке увеличения скорости роста. Приведите расчеты. Постройте графики этих функции для наглядности.

1. $2^{\log m}$ $(\sqrt{2})^{\log m}$
2. $2^{2^{**}n}$ $2^{2^{**}n+1}$
3. n! (n+1)!
4. log (n!) (log n)!
5. $(3/2)^n$ e^n $n*2^n$
6. n 2n n^3 n^2
7. $n^{1/\log n}$ $n^{\log\log n}$ $(\log n)^{(\log n)}$
8. ln n ln ln n
9. 4 log n $\log^2 n$ n log n
10. `√ (log n) 2√ (2log n)

Контрольные вопросы:

1. Что означает – асимптота роста? Какие обозначения вы знаете?
2. В чем отличие между **O'** и **o', Ω** и **ω?**
3. Какие способы существуют для сравнения скорости роста двух алгоритмов?
4. Каковы недостатки и преимущества каждого способа?
5. Как по графикам можно определить какой из алгоритмов имеет более быстрый рост?
6. Верно ли высказывание, что один и тот же алгоритм может иметь различную эффективность про малых и больших размерности входа? П

Содержание отчета

Задание 1.

1. Заполнение таблицы.
2. Построение графиков роста функций.
3. Упорядочивание функций в порядке возрастания их роста.

Задание 2.

1. Заполнение таблицы
2. Приведение алгоритма сравнения функций.
3. Построение графиков роста функций.
4. Обоснование результатов.

Задание 3.

1. Алгоритм сравнения функций.
2. Обоснование полученных результатов.

Задание 4.

1. Ответы на контрольные вопросы.
2. Вывод по лабораторной работе.

ЛАБОРАТОРНАЯ РАБОТА № 2

Тема: Методы быстрой сортировки и их сравнительный анализ

Цель: Реализация и анализ алгоритмов сортировки

Краткие теоретические сведения

Перечень быстрых методов сортировки

1. Сортировка вставками (INSERTIONSORT).
2. Сортировка методом слияния (MERGESORT).
3. Быстрая сортировка (QUICKSORT).
4. Сортировка подсчетом. (COUNTINGSORT).
5. Восходящая поразрядная сортировка. (MSD RADIXSORT).
6. Нисходящая поразрядная сортировка. (LSD RADIXSORT).

Существует еще множество алгоритмов сортировки, которые основаны на вышеприведенных. (TimSort, ShellSort, BucketSort и другие).

Задания к лабораторной работе

1. Для трех любых алгоритмов сортировки записать алгоритм с помощью псевдокода.

Пример:

INSERTION-SORT(A)

```
1.  for j←2 to length[A]
2.       key ← A[j]
3.       ·> добавить A[j] к отсортированной части A[1..j]
4.       i ← j - 1
5.      while i > 0 & A[i] > key
6.           A[i+1] ←A[i]
7.           i ← i - 1
8.      A[i+1] ← key
```

Таким же образом выполняется для двух других алгоритмов.

2. Продемонстрировать работу алгоритмов (вывести массив на каждом шагу и трассировку этапов) для заданного массива:

№ варианта	Массив А
1	21, 31, 24, 10, 31, 41, 8, 2, 57, 15
2	15, 25, 10, 16, 5, 30, 57, 15, 8, 2
3	3, 42, 31, 84, 89, 5, 73, 40, 44, 32
4	1, 99, 33, 5, 3, 37, 44, 32, 57, 15
5	7, 5, 2, 96, 37, 54, 49, 8, 44, 32
6	56, 83, 25, 9, 55, 1, 10, 90, 8, 2
7	21, 35, 56, 41, 24, 88, 6, 11, 57, 15
8	67, 82, 36, 34, 55, 89, 4, 18, 44, 32
9	17, 3, 7, 80, 23, 11, 64, 32, 8, 2
10	49, 28, 74, 56, 81, 14, 76, 44, 57, 15
11	87, 22, 57, 13, 9, 11, 31, 89, 8, 2
12	16, 54, 33, 34, 17, 19, 90, 2, 44, 32
13	53, 50, 60, 78, 5, 77, 86, 12, 57, 15
14	3, 5, 16, 9, 31, 15, 32, 87, 8, 2, 44
15	15, 25, 10, 16, 3, 5, 30, 57, 1, 44, 32
16	1, 99, 33, 5, 3, 37, 7, 44, 32, 57, 15
17	56, 83, 55, 9, 55, 1, 10, 12, 90, 44, 32
18	67, 3, 82, 36, 34, 55, 83, 4, 18, 8, 2
19	49, 28, 74, 56, 81, 19, 14, 76, 4, 57, 15
20	16, 54, 33,5, 58, 17, 19, 90, 2, 44, 32
21	3, 5, 16, 9, 31, 15, 32, 35, 87, 8, 2
22	21, 30, 7, 24, 10, 31, 41, 8, 2, 57, 15
23	83, 3, 42, 31, 84, 89, 5, 73, 46, 8, 2
24	7, 5, 2, 17, 96, 37, 54, 49, 8, 44, 32
25	11, 35, 56, 53, 41, 24, 88, 6, 21, 57, 15
26	17, 38, 7, 80, 3, 11, 64, 32, 36, 8, 2
27	87, 22, 57, 63, 13, 9, 11, 31, 89, 44, 32
28	53, 50, 60, 78, 5, 77, 11, 86, 12, 57, 15
29	9, 6, 93, 84, 67, 33, 52 ,99, 81, 8, 2
30	3, 5, 16, 9, 31, 15, 32, 87, 8, 2, 44

Пример (вывод массива, и трассировка шагов):

Трассировка работы алгоритма INSERTION-SORT

INSERTION-SORT(A) A = (5, 2, 4, 6, 1, 3)

1 for j←2 to 6 (i, j)
2 do key←A[2] = 2
3 ›добавить A[2] к отсортированной части A[1] = 5
4 i←j-1 = 1
5 while 1 > 0 and A[1] = 5 > key = 2
6 do A[2] ← A[1] = 5
7 i ← i - 1 = 0
5 while i > 0
8 A[1] ← key = 2 A = (2, 5, 4, 6, 1, 3) (i, j)

1 j = 3, 6
2 do key ← A[3] = 4
3 ›добавить A[3] к отсортированной части A[1, 2] = (2, 5)
4 i ← i - 1 = 2
5 while 2 > 0 and A[2] = 5 > key = 4
6 do A[3]←A[2] = 5
7 i ← i - 1 = 1
5 while 1 > 0 and A[1] = 2 > key = 4
8 A[2] ← key = 4 A = (2, 4, 5, 6, 1, 3) (i, j)

1 j = 4, 6
2 do key ← A[4] = 6
3 ›добавить A[3] к отсортированной части A[1, 2, 3] = (2, 4, 5)
4 i ← i - 1 = 3
5 while 3 > 0 and A[3] =5 > key = 6
8 A[4] ← key = 6 A = (2, 4, 5, 6, 1, 3) (i, j)

и т. д.

3. Провести временной анализ алгоритмов сортировки (проанализировав псевдокод программ, то есть теоретически), и сравнить данные с практическими. Построить графическую зависимость времени от размерности массива для каждого из алгоритмов на массиве от n случайных элементов, не менее 3-х раз с размерностями больше, чем на предыдущем шагу.

Пример:

Теоретически (по оценке псевдокода):

Время работы сортировки вставкой $T(n) = \theta(n^2)$

Время работы сортировки слиянием $T(n) = \theta(n \log n)$

Время работы быстрой сортировки $T(n) = \theta(n \log n)$

Практически (результаты работы программы)

Время работы алгоритмов:

- Количество элементов: 5000

  INSERTION SORT: 0.0372 с.

  MERGE SORT: 0.0093453 с.

  QUICK SORT: 0.003425 с.

- Количество элементов: 16000

  INSERTION SORT: 0.14725 с.

  MERGE SORT: 0.06683 с.

  QUICK SORT: 0.04999 с.

- Количество элементов: 246000

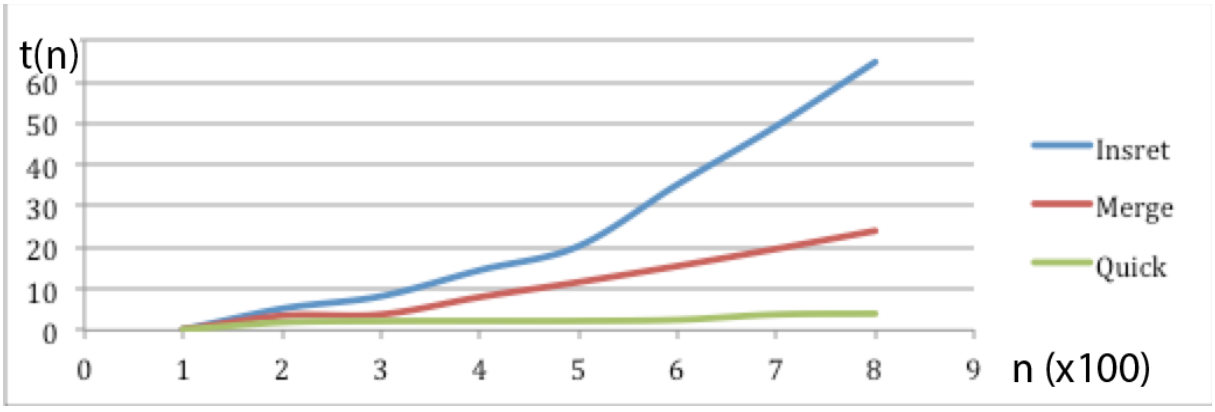
  INSERTION SORT: 0.24345 с.

  MERGE SORT: 0.033453 с.

  QUICK SORT: 0.035592 с.

Что совпадает с теоретическими данными.

Графики скорости работы алгоритмов:



4. Привести скриншоты результатов работы программы.

Пример:

[17 3 7 80 3 11 64 32] -> INSERTIONSORT -> [3 3 7 11 17 32 64 80] за 0.0002763с

[17 3 7 80 3 11 64 32] -> MERGESORT -> [3 3 7 11 17 32 64 80] за 0.0000492с

[17 3 7 80 3 11 64 32] -> QUICKSORT -> [3 3 7 11 17 32 64 80] за 0.0000187с

Примечание

При использовании старой библиотеки <time.h> может возникнуть проблема, что на экран выводится 0.0 – это связано с неточностью работу встроенных функций. Предлагается использовать <chrono> — это библиотека имеется только в стандарте C++11 и выше, но она позволяет вычислить очень точное время работы.

5. Привести листинг программы (исходный код программы).

Содержание отчета:

1. Псевдокод алгоритмов.
2. Трассировка работы алгоритмов для заданного массива.
3. Сравнительное время работы (теоретическое).
4. Графики скорости роста работы алгоритмов.
5. Результаты работы.
6. Вывод.
7. Листинг программы.

Контрольные вопросы:

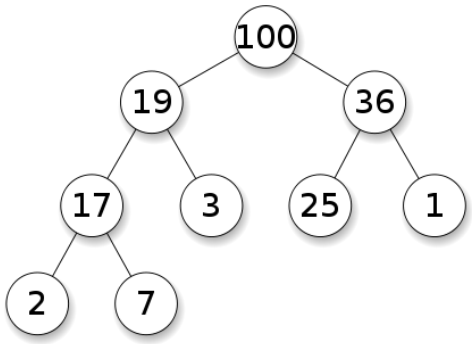
1. Какие виды сортировок вы знаете?
2. Почему разные алгоритмы работают с разной эффективностью?
3. Каким образом измеряется время работы алгоритма?
4. Для чего требуется улучшение эффективности алгоритмов?
5. Алгоритмы с какой скоростью роста являются применимы на практике, какие нет?

ЛАБОРАТОРНАЯ РАБОТА № 3

Тема: Организация очереди с приоритетом с помощью кучи

Цель: Изучение основного свойства кучи, ее построение и сортировка. Организация очереди с приоритетом с помощью кучи.

Краткие теоретические сведения



Куча (Пирамида) — это специализированная структура данных типа дерево, которая удовлетворяет *свойству кучи:* если B является узлом-потомком узла A, то ключ (A) ≥ ключ (B). Для данной выше кучи, 17 и 3 являются потомками 19, и следовательно, их ключи (значения) меньше 19-ти. Куча является самым эффективным методом реализации так называемой *очереди с приоритетом* – типа данных, в котором реализованы 2 основные операции – извлечение максимального элемента, и добавление нового элемента. Алгоритм сортировки кучей является одним из самых эффективных алгоритмов, как по времени, так и по памяти, и широко применяется в современных вычислительных системах.

Над кучами обычно проводятся следующие операции

- поддержание основного свойства кучи (HEAPIFY);
- построение кучи (BUILD-HEAP);
- сортировка массива (HEAP-SORT);
- взятие наибольшего элемента массива (HEAP-EXTRACT-MAX);
- добавление элемента (HEAP-INSERT).

Задания к лабораторной работе:

1. Привести, используя псевдокоды, алгоритмы основных операций над кучей:
   - поддержание основного свойства кучи (HEAPIFY);

– построение кучи (BUILD-HEAP);

– сортировка массива (HEAP-SORT);

– взятие наибольшего элемента массива (HEAP-EXTRACT-MAX);

– добавление элемента (HEAP-INSERT);

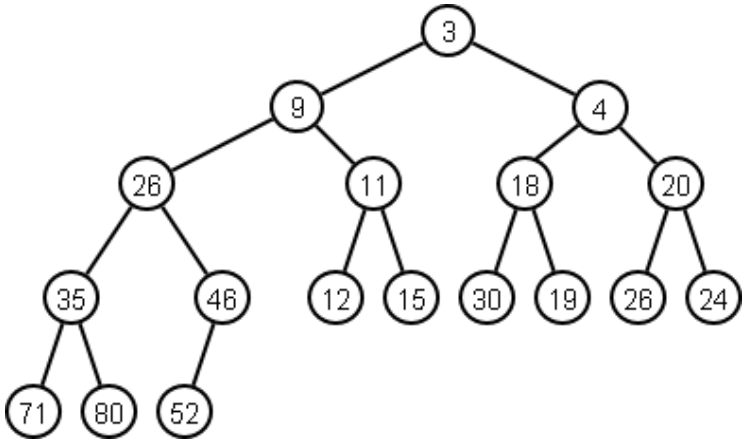
2. Реализовать алгоритм построения кучи.
3. Реализовать алгоритм сортировки с помощью кучи.
4. Организовать очередь с приоритетами и показать работу процедур:
   – HEAP-INSERT (A,key) для кучи A;
   – HEAP-EXTRACT-MAX для этой кучи.

№ варианта	Массив A	Элемент добавления, key
1	21, 31, 24, 10, 31, 41, 8, 2, 57, 15	26
2	15, 25, 10, 16, 5, 30, 57, 15, 8, 2	20
3	3, 42, 31, 84, 89, 5, 73, 40, 44, 32	40
4	1, 99, 33, 5, 3, 37, 44, 32, 57, 15	30
5	7, 5, 2, 96, 37, 54, 49, 8, 44, 32	50
6	56, 83, 25, 9, 55, 1, 10, 90, 8, 2	60
7	21, 35, 56, 41, 24, 88, 6, 11, 57, 15	40
8	67, 82, 36, 34, 55, 89, 4, 18, 44, 32	50
9	17, 3, 7, 80, 23, 11, 64, 32, 8, 2	20
10	49, 28, 74, 56, 81, 14, 76, 44, 57, 15	60
11	87, 22, 57, 13, 9, 11, 31, 89, 8, 2	30
12	16, 54, 33, 34, 17, 19, 90, 2, 44, 32	20
13	53, 50, 60, 78, 5, 77, 86, 12, 57, 15	65
14	3, 5, 16, 9, 31, 15, 32, 87, 8, 2, 44	20
15	15, 25, 10, 16, 3, 5, 30, 57, 1, 44, 32	27
16	1, 99, 33, 5, 3, 37, 7, 44, 32, 57, 15	40
17	56, 83, 55, 9, 55, 1, 10, 12, 90, 44, 32	60
18	67, 3, 82, 36, 34, 55, 83, 4, 18, 8, 2	50
19	49, 28, 74, 56, 81, 19, 14, 76, 4, 57, 15	50
20	16, 54, 33,5, 58, 17, 19, 90, 2, 44, 32	40
21	3, 5, 16, 9, 31, 15, 32, 35, 87, 8, 2	34
22	21, 30, 7, 24, 10, 31, 41, 8, 2, 57, 15	25
23	83, 3, 42, 31, 84, 89, 5, 73, 46, 8, 2	50
24	7, 5, 2, 17, 96, 37, 54, 49, 8, 44, 32	40
25	11, 35, 56, 53, 41, 24, 88, 6, 21, 57, 15	55
26	17, 38, 7, 80, 3, 11, 64, 32, 36, 8, 2	60
27	87, 22, 57, 63, 13, 9, 11, 31, 89, 44, 32	60
28	53, 50, 60, 78, 5, 77, 11, 86, 12, 57, 15	70
29	9, 6, 93, 84, 67, 33, 52 ,99, 81, 8, 2	70
30	3, 5, 16, 9, 31, 15, 32, 87, 8, 2, 44	20

Содержание отчета

1. Показать выбранный массив в виде неотсортированной кучи.

Пример:



2. Привести псевдокоды функций.

Например:

HEAP-EXTRACT (A)

1 if heap-size [A] < 1

2 error “Очередь пуста”

3. max ← A [1]

4. A [1] ← A [heap-size]

5. heap-size ← heap-size - 1

6. HEAPIFY (A, 1)

7. return max

3. Продемонстрировать эти алгоритмы на примере.

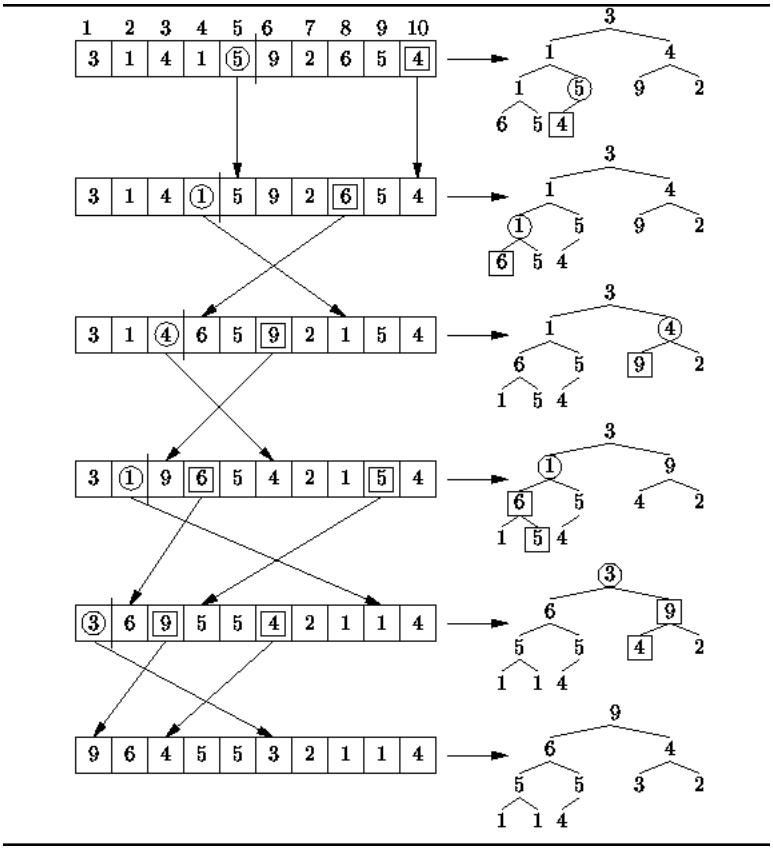
BUILD-HEAP (A)

1 heap-size[A] ← length[A]

2 for i ← ⌊length / 2⌋ down to 1

3 HEAPIFY(A, i)

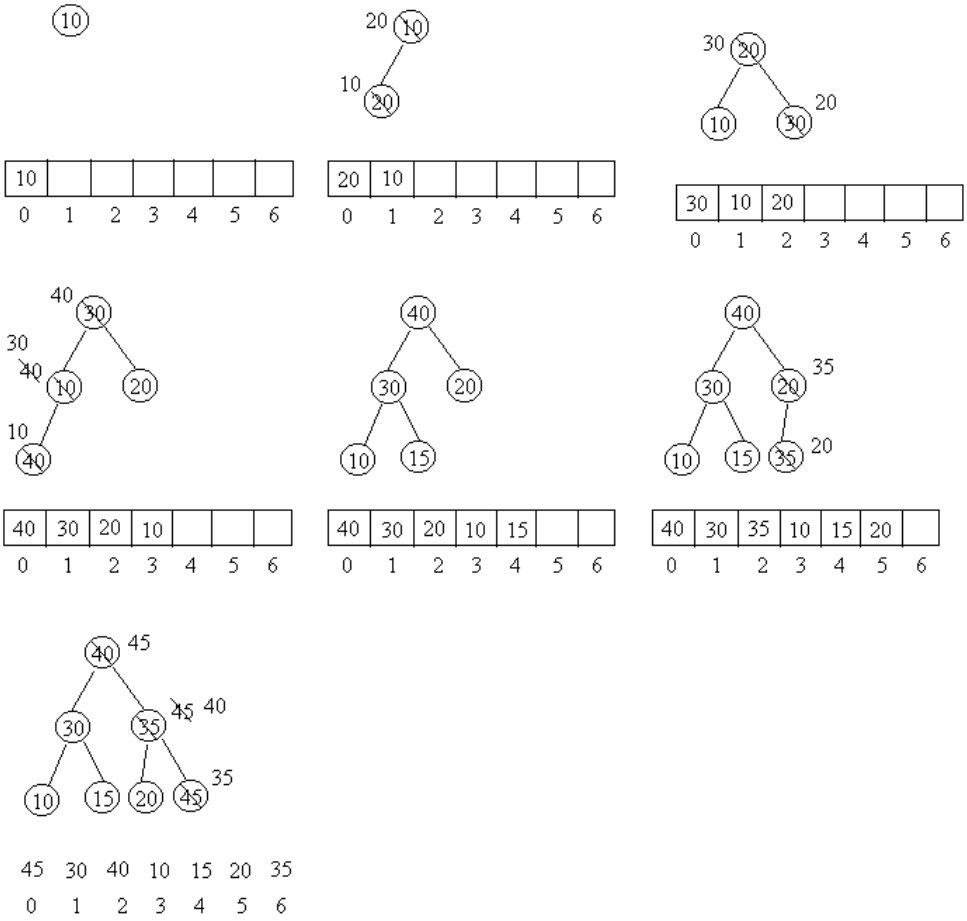
Ниже показана трассировка алгоритма построения кучи:



HEAP-INSERT(A, key)

1. heap-size [A] ← heap-size [A] + 1
2. i ← heap-size [A]
3. while i > 1 и A [PARENT(i)] < key
4. do A[i] ← A [PARENT(i)]
5. i ← PARENT(i)
6. A[i] ← key

Ниже показана трассировка алгоритма HEAP-INSERT(A, key):



Подобным образом показать трассировку работы для других алгоритмов. Можно продемонстрировать работу программы на массиве, без использования деревьев.

Из примера работы HEAP-INSERT видно, что есть возможность реализовать функцию BUILD-HEAP на основе HEAP-INSERT, так как после каждого шага HEAP-INSERT получается куча.

Дополнительное задание: реализовать оба вида алгоритма (BUILD-HEAP (A) и BUILD-HEAP′(A)) и сравнить их работу.

4. Провести сравнение времени работы алгоритма сортировки кучей с алгоритмами сортировки, рассмотренными в лабораторной работе № 2.

Сформулировать выводы.

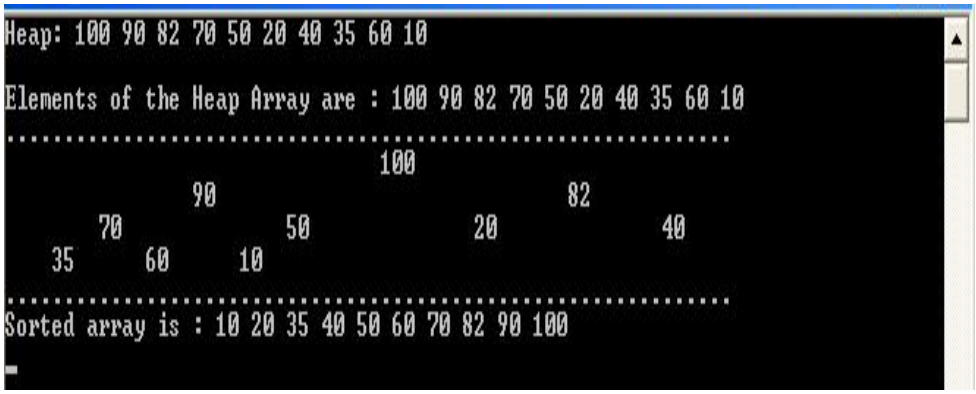
Пример:

Сравнение алгоритмов реализовано теоретически

Алгоритм	Худший случай	Среднее время	Худший случай
QUICKSORT	n log n	n log n	n^2
MERGE-SORT	n log n	n log n	n log n
HEAPSORT	n log n	n log n	n log n
INSERTION-SORT	n	n^2	n^2

Откуда видно, что HEAP-SRORT - самый эффективный (если учитывать затрачиваемую память).

5. Результаты работы программ.



6. Вывод

7. Листинг программы

Контрольные вопросы:

1. Каким образом работает сортировка кучей и другие процедуры для работы с кучей?
2. Какова эффективность (скорость работы) алгоритма сортировки?
3. Почему этот алгоритм является одним из наиболее эффективных алгоритмов?
4. Есть ли у него недостатки? Какие?
5. Какой способ построения кучи наиболее удобен и эффективен?

ЛАБОРАТОРНАЯ РАБОТА № 4

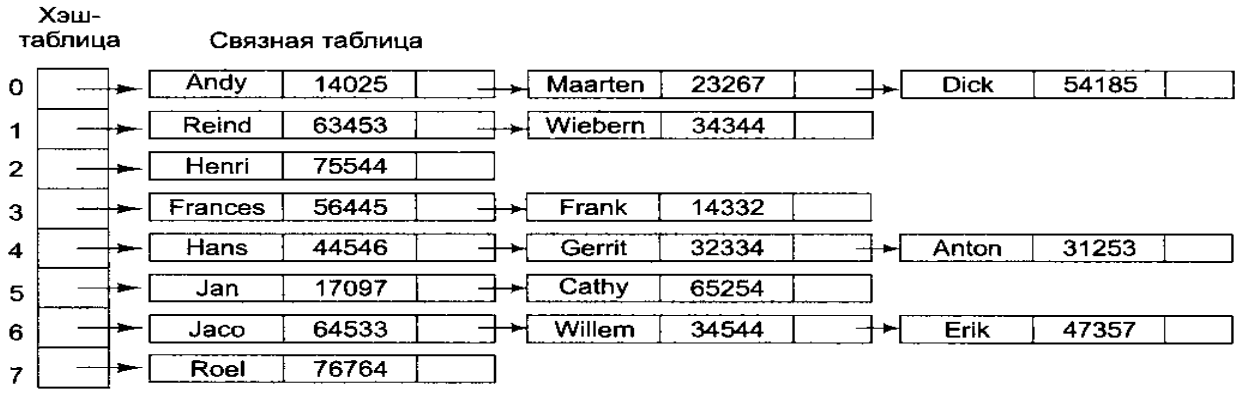
Тема: Хеш-таблицы

Цель: Изучить работу алгоритмов: прямой адресации, хеш-таблицы и открытой адресации. Реализовать перечисленные выше алгоритмы и провести их сравнительный анализ.

Краткие теоретические сведения

Хеш-таблица — это структура данных, реализующая интерфейс ассоциативного массива, а именно, она позволяет хранить пары (ключ, значение) и выполнять три операции: операцию добавления новой пары, операцию поиска и операцию удаления пары по ключу.

В общем виде хэш таблица выглядит следующим образом:



Для адресации (получения индекса элемента) используется хэш-функции.

В зависимости от кого, как они выглядят, различают 3 вида адресации:

1) **Прямая адресация** – каждому ключу соответствует равнозначная позиция в массиве. $h(k) = k$

2) Закрытая адресация (хэш-таблица)

Способы построения хеш-функции:

- деление с остатком:

  $h(k) = k \ mod \ m;$

- умножение:

  $h(k) = [m * ((k * A) \ mod \ 1)]$, где $A = 0{,}61803;$

Например: ключ $k = 59$, то используя способ умножения построения хеш-функции, ключ будет записан в ячейку с индексом, равным $h(59) = 6$.

$$h(59) = [20 * ((59 * 0{,}61803) mod\ 1)] = [20 * ((36{,}312) mod\ 1)] =$$
$$= [20 * 0{,}312] = 6.$$

3) Открытая адресация

– *Линейная*

$$h(k, i) = (h(k) + i)\ mod\ m.$$

В данном случае считается количество попыток записать число (i), и, если ячейка по полученному индексу уже занята, пробуем еще раз, считая увеличенное число попыток (i + 1).

– *Квадратичная*

$$h(k) = (h_1(k) + c_1 i + c_2 i^2)\ mod\ m\ ,\ h_1(k) = k\ mod\ m,$$

c_1, c_2 - любые константы на выбор (например, $c_1 = 1, c_2 = 3$).

– *Двойное хеширование*

$$h(k, i) = \left(h_1(k) + i h_2(k)\right) mod\ m,$$

где $h_1(k) = k\ mod\ m,$

$$h_2(k) = (1 + k\ mod\ m).$$

Задания к лабораторной работе

1. Пусть размер хеш-таблицы равен m = 20, а хеш-функция имеет вид
   h(k) = [m ((kA) mod 1)], где A = 0,61803.
   В какие позиции попадут ключи (см. таблицу, столбец 3)?

2. Как будет выглядеть хеш-таблица с цепочками после того, как в неё последовательно поместили элементы с ключами (см. таблицу, столбец 2) (в указанном порядке)? Число позиций в таблице равно 9, хеш-функция имеет вид h(k) = k mod 9.

3. Выполните добавление ключей (в указанном порядке, см. таблицу. столбец 2) в хеш-таблицу с открытой адресацией размера m = 11. Для вычисления последовательности проб используется линейный метод с h'(k) = k mod m. Выполните то же задание, если используется квадратичный метод с той же h', c_1 = 1, c_2 = 3, а также для двойного хеширования с h_1 = h' и h_2(k) = 1 + (k mod (m - 1)).

Таблица

№ варианта	Массив	Ключи
1	10, 22, 31, 4, 15, 28, 17, 88, 59	61, 62, 63, 64 и 65
2	21, 31, 24, 10, 40, 41, 9, 2, 20	51, 52, 53, 54 и 55
3	15, 25, 10, 16, 5, 30, 57, 18, 19	41, 42, 43, 44 и 45
4	3, 42, 31, 84,9, 5, 73, 4, 13	31, 32, 33, 34 и 35
5	1, 99, 33, 5, 3, 37, 44, 32, 12	21, 22, 23, 24 и 25
6	7, 5, 2, 19, 37, 54, 49, 8, 10	46, 47, 48, 49 и 50
7	56, 18, 55, 9, 50, 1, 10, 88, 27	36, 37, 38, 39 и 40
8	11, 35, 56, 41, 24, 88, 6, 2, 26	44, 45, 46, 47 и 48
9	67, 76, 37, 34, 57, 13, 4, 1, 10	56, 57, 58, 59 и 60
10	17, 3, 9, 73, 6, 10, 64, 32, 21	66, 67, 68, 69 и 70
11	49, 32, 75, 53, 85, 19, 76, 44, 5	81, 82, 83, 84 и 85
12	80, 22, 57, 13, 9, 18, 31, 86, 42	43, 44, 45, 46 и 47
13	16, 5, 33, 34, 17, 29, 35, 6, 38	33, 34, 35, 36 и 37
14	59, 50, 4, 28, 5, 77, 86, 12, 22	53, 54, 55, 56 и 57
15	3, 5, 16, 9, 21, 14, 36, 44, 17	87, 86, 88, 89 и 90
16	3, 42, 31, 12, 33, 24, 13, 40, 10	63, 64, 65, 66 и 67
17	19, 37, 54, 49, 10, 1, 45, 18, 13	83, 84, 85, 86 и 87
18	17, 88, 59, 4, 15, 28, 10, 22, 31	65,63, 64, 62 и 61
19	9, 2, 20, 10, 40, 41, 21, 31, 24	55, 54, 53, 52 и 51
20	57, 18, 19, 16, 5, 30, 15, 25, 10	45, 44, 43, 42 и 41
21	73, 4, 13, 84,9, 5, 3, 42, 31	35, 34, 33, 32 и 31
22	44, 32, 12, 5, 3, 37, 1, 99, 33	25, 24, 23, 22 и 21
23	49, 8, 10, 19, 37, 54, 7, 5, 2	50, 49, 48, 47 и 46
24	10, 88, 27, 9, 50, 1, 56, 18, 55	40, 39, 38, 37 и 36
25	88, 6, 2, 26, 56, 41, 24, 56, 18	48, 47, 46, 45 и 44
26	6, 2, 26, 41, 24, 88, 11, 35, 56	60, 59, 58, 57 и 56
27	64, 32, 21, 73, 6, 10, 17, 3, 9	70, 69, 68, 67 и 66
28	76, 44, 5, 53, 85, 19, 80, 22, 57	85, 84, 83, 82 и 81
29	31, 86, 42, 13, 9, 18, 80, 22, 57	47, 46, 45, 44 и 43
30	35, 6, 38, 34, 17, 29, 16, 5, 33	37, 36, 35, 34 и 33

Содержание отчета

1. Привести все используемые алгоритмы в псевдокодах.

   Здесь необходимо указать использованные хэш-функции HASH(k), а также процедуры добавления элементов HASH-INSERT(k).

2. Продемонстрировать работу алгоритмов на примере (см. раздел «задание к лабораторной работе»).

Для этого приведите окончательные результаты в виде массива с ключами, расставленными соответствующе правильному ответу.

Например:

Хэш функция: $h(k) = [20*(k*0,61803 \bmod 1)]$

Массив ключей: 51, 52, 63, 54 и 55

Конечный массив:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
		52					54	46		51								63	

3. Привести сравнительный анализ хэш-функций для каждого типа адресации и между разными типами адресации по их эффективности и реализации.
4. Результаты работы программ.
5. Вывод.
6. Листинг программ.

Контрольные вопросы:

1. Почему использование хэширования повышает эффективность работы со структурами данных?
2. Каким образом надо выбирать хэш-функцию?
3. Почему используются методы открытой адресации?
4. Какими недостатками обладает данный метод хранения данных (хэш-таблицы)?
5. Какой из данных видов хэширования вы считаете наиболее эффективным и почему?

ЛАБОРАТОРНАЯ РАБОТА № 5

Тема: Динамическое программирование. Задача об умножении последовательности матриц

Цель: Найти такую (полную) расстановку скобок в произведении матриц, чтобы вычисление произведения требовало наименьшего числа умножений

Краткие теоретические сведения

Если имеется, например, 4 матрицы, которые будут перемножаться, то для этих четырех матриц $A_1A_2A_3A_4$ массив p задает размеры матриц:

<$p_0\ p_1p_2p_3p_4$> то есть матрицы будут иметь размеры:

$A_1 = (p_0 x p_1)\quad A_2 = (p_1 x p_2)\quad A_3 = (p_2 x p_3)\quad A_4 = (p_3 x p_4)$

Следовательно, необходимо найти оптимальный способ расстановки скобок при умножении, при котором количество операций умножения будет минимальным.

Если умножаем матрицы $A_1A_2A_3$, то при этом существует 2 варианта умножения:

$(A_1A_2)A_3$ и $A_1(A_2A_3)$

при размерах матриц A_1 (5x10), A_2 (10x15), A_3 (15x5).

Для первого варианта количество перемножений будет равно n=5x10x15 + 5x10x15 = 1500, а для второго n = 10x15x5 + 5x10x5 = 1000, откуда видно, что в данном случае второй вариант оптимальнее по количеству операции, и соответственно - скорости работы.

Перемножение двух матриц

```
MATRIX-MULTIPLY (A, B)
1 if columns[A] ≠ rows[B]
2       then error «умножить нельзя»
3       else for r ← 1 to rows [A]
4                 do for j ← 1 to columns [B]
5                           do C [i, j] ← 0
6                                 for k ← 1 to columns [A]
7                                       do C [i, j] ← C [i, j] + A [i, k] · B [k, j]
8 return C
```

Вычисление оптимальной стоимости

```
MATRIX-CHAIN-ORDER(p)
1   n ← length[p] – 1
2   for i ←1 to n
3           do m[i, i] ← 0
4   for l ←2 to n
5           do for i ←1  to n – l + 1
6                   do j ← i + l – 1
7                         m[i, j] ← ∞
8                for k ← i to j – 1
9                   do q← m[i, k] + m[k+1, j] + p_{i-1} p_k p_j
10                        if q < m[i, j]
11                          then m[i, j] ← q
12                                    s[i, j] ← k
13  return m, s
```

Построение оптимального решения

```
MATRIX- CHAIN- MULTIPLY (A, s, i, j)
1     if j > i
2        then X ← MATRIX-CHAIN-MULTIPLY (A, s,i, s[i, j])
3           У ← MATRIX-CHAIN-MULTIPLY (A, s, s[i, j] + 1, j)
4           return MATRIX-MULTIPLY (X, У)
5        else return A_i
```

Распечатка скобок в произведении последовательности матриц

```
PRINT_OPTIMAL_PARENS (S, i, j)

1   if i = j
2      then print “A” i
3      else print “(”
4      PRINT_OPTIMAL_PARENS (S, i, S[i, j])
5      PRINT_OPTIMAL_PARENS (S, S[i, j]+1, j)
6      PRINT “)”
```

Пример:

Найти оптимальную расстановку скобок в произведении последовательности матриц, размерности которых равны

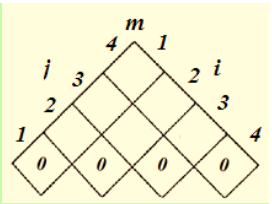
p = (4, 10, 15, 8, 5), A_1 = (4 x 10);

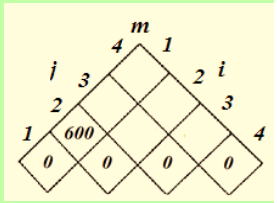
A_2 = (10 x 15); A_3 = (15 x 8); A_4 = (8 x 5)

Дано: p = < 4, 10, 15, 8, 5 >, n= 4

p_0 p_1 p_2 p_3 p_4

```
MATRIX-CHAIN-ORDER(p)
1   n ← length[p] - 1
2   for i ←1 to n
3    do  m[i, j] ← O
4   for l ←2 to n
5    do for i ←1  to n - l + 1
6          do j ← i + l – 1
7                m[i, j] ← ∞
8                for k ← i to j - 1
9                   do q-← m[i, k] + m[k+1, j] + p_{i-1} p_k p_j
10                    if q < m[i, j]
11                       then m[i, j] ← q
12                            s[i, j] ← k
13  return m, s
```





Трассировка:

1 n = 4

2 for I = $\overline{1, 4}$

3 m[1, 1] = ∅, m[2, 2] = ∅,
m[3, 3] = ∅, m[4, 4] = ∅,

4 for l = $\overline{2, 4}$

5 do for i=1 to 4 – 2 + 1 = 3

6 do j=1 + 2 – 1 = 2

7 m[1, 2] ← ∞

8 for k ← 1 to 2 – 1 = 1

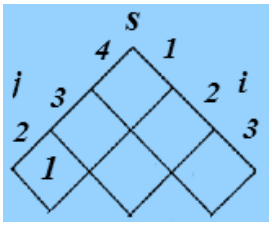
9 do q ← m[1, 1] + m[2, 2] + $p_0p_1p_2$ = 0 + 0 + 4·10·15 = 600

10 if 600 < ∞

11 then m[1, 2] ← 600

12 s[1, 2] ← 1

13 return m[1, 2] ← 600 s[1, 2] ← 1



5 do for i =2 to 4 – 2 + 1 = 3

6 do j=2 + 2 – 1 = 3

7 m[2, 3] ← ∞

8 for k ← 2 to 3 – 1 = 2

9 do q ← m[2, 2] + m[3, 3] + $p_1p_2p_3$ = 0 + 0 +10·15·8 =1200

10 if 1200 < ∞

11 then m[2, 3] ← 1200

12 s[2, 3] ← 2

13 return m[2, 3] ← 1200 s[2, 3] ← 2

5 do for i=3 to 4 – 2 + 1 = 3

6 do j=3 + 2 – 1 = 4

7 m[3, 4] ← ∞

8 for k ← 3 to 4 – 1 = 3

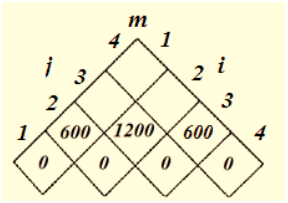
9 do q ← m[3, 3] + m[4, 4] + $p_2p_3p_4$ = 0 + 0 + 15·8·5 = 600

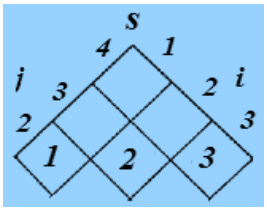
10 if 600 < ∞

11 then m[3, 4] ← 600

12 s[3, 4] ← 3

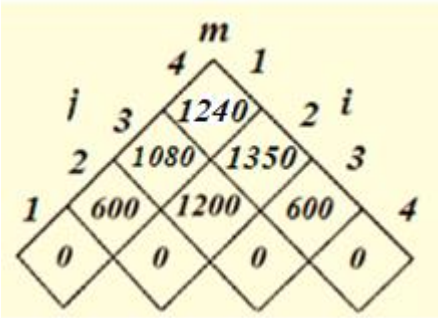
13 return m[3, 4] ← 600 s[3, 4] ← 3

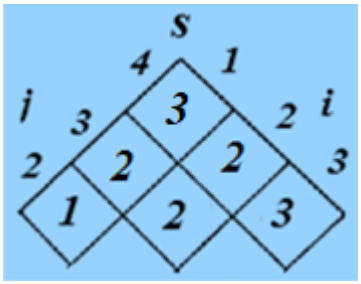




и т. д.

В результате получаем:





Расставим скобки в последовательности перемножения матриц.

```
PRINT_OPTIMAL_PARENS (S, i, j)
1. if i = j
2.     then print "A" i
3.     else print "("
4.     PRINT_OPTIMAL_PARENS (S, i, S[i, j])
5.     PRINT_OPTIMAL_PARENS (S, S[i, j]+1, j)
6.     PRINT ")"
```

PRINT_OPTIMAL_PARENS (S, 1, 4)

1. if 1 ≠ 4

3. else print "("

4. PRINT_OPTIMAL_PARENS (S, 1, S[1, 4]=3)

4 PRINT_OPTIMAL_PARENS (S, 1, 3)

1. if 1 ≠ 3

3. else print "("

4. PRINT_OPTIMAL_PARENS (S, 1, S[1, 2]=2)

4 PRINT_OPTIMAL_PARENS (S, 1, 2)

1. if 1 ≠ 2

3. else print "("

4. PRINT_OPTIMAL_PARENS (S, 1, S[1, 2]=1)

4 PRINT_OPTIMAL_PARENS (S, 1, 1)

1. if 1 = 1

2. then print "A" 1

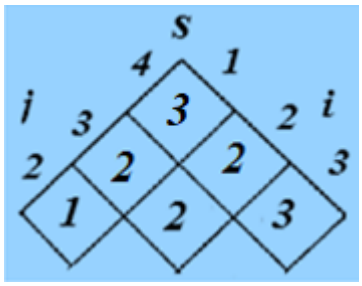
5 PRINT_OPTIMAL_PARENS (S, 2, 2)

1. if 2 = 2

2. then print "A" 2

6. PRINT ")"

((



((A1A2)

5 PRINT_OPTIMAL_PARENS (S, 3, 3)

1. if 3 = 3
2. then print “A” 3

(((A1A2) A3)

6. PRINT “)”

```
PRINT_OPTIMAL_PARENS (S, i, j)
1. if i = j
2.     then print “A” i
3.     else print “(”
4.     PRINT_OPTIMAL_PARENS (S, i, S[i, j])
5.     PRINT_OPTIMAL_PARENS (S, S[i, j]+1, j)
6.     PRINT “)”
```

5 PRINT_OPTIMAL_PARENS (S, 4, 4)

1. if 4 = 4
2. then print “A” 4

(((A1A2) A3)A4)

6. PRINT “)”

Вычисление оптимальной стоимости

m[1, 1] = ∅, m[2, 2] = ∅, m[3, 3] = ∅, m[4, 4] = ∅;

$m[1, 2] = m[1, 1] + m[2, 2] + p_0p_1p_2 = 0 + 0 + 4 \cdot 10 \cdot 15 = 600$; k =1;

$m[2, 3] = m[2, 2] + m[3, 3] + p_1p_2p_3 = 0 + 0 + 10 \cdot 15 \cdot 8 = 1200$; k =2;

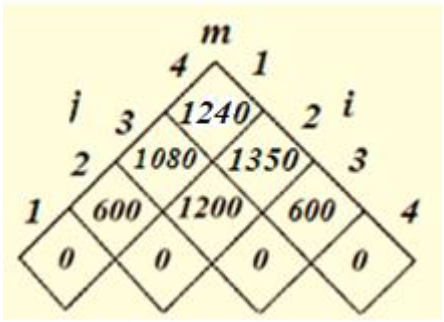
$m[3, 4] = m[3, 3] + m[4, 4] + p_2p_3p_4 = 0 + 0 + 15 \cdot 8 \cdot 5 = 600$; k = 3;

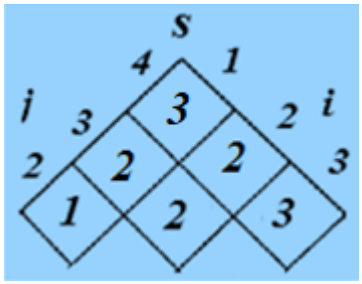
$$m[1, 3] = \min_{1 \le k < 3} \begin{cases} m[1,1] + m[2,3] + p_0p_1p_3 = 0 + 1200 + 4 \cdot 10 \cdot 8 = 1520, & k = 1; \\ m[1,2] + m[3,3] + p_0p_2p_3 = 600 + 0 + 4 \cdot 15 \cdot 8 = 1080, & k = 2; \end{cases}$$

$$m[2, 4] = \min_{1 \le k < 3} \begin{cases} m[2,2] + m[3,4] + p_1p_2p_4 = 0 + 600 + 10 \cdot 15 \cdot 5 = 1350, & k = 2; \\ m[2,3] + m[4,4] + p_1p_3p_4 = 1200 + 600 + 10 \cdot 8 \cdot 5 = 2200, & k = 3; \end{cases}$$

$$m[1, 4] = \min_{1 \le k < 4} \begin{cases} m[1,1] + m[2,4] + p_0p_1p_4 = 0 + 1350 + 4 \cdot 10 \cdot 5 = 1550, & k = 1; \\ m[1,2] + m[3,4] + p_0p_2p_4 = 600 + 600 + 4 \cdot 15 \cdot 5 = 1500, & k = 2; \\ m[1,3] + m[4,4] + p_0p_3p_4 = 1080 + 0 + 4 \cdot 8 \cdot 5 = 1240, & k = 3; \end{cases}$$

Результаты работы алгоритмов





(((A1A2) A3) A4)

Задания к лабораторной работе:

1. Реализовать алгоритм вычисления оптимальной стоимости MATRIX-CHAIN-ORDER(p).
2. Реализовать алгоритм построения оптимального решения MATRIX-CHAIN-MULTIPLY(A, s, i, j).
3. Реализовать алгоритм PRINT_OPTIMAL_PARENS (S, i, j), печатающий оптимальную расстановку скобок.

Найдите оптимальную расстановку в задаче о перемножении матриц, и выведите ответ на экран, если

№ варианта	Массив P
1	<5, 10, 3, 12, 6>
2	<12, 5, 50, 6, 3>
3	<10, 3, 5, 12, 4>
4	<12, 30, 5, 6, 10>
5	<3, 10, 5, 50, 5>
6	<50, 5, 10, 6, 4>
7	<4, 8, 3, 20, 5>
8	<20, 4, 8, 5, 10>
9	<10, 5, 20, 15, 10>
10	<10, 3, 5, 12, 4>
11	<2, 10, 5, 15, 10>
12	<10, 15, 5, 12, 4>
13	<10, 4, 5, 15, 8>
14	<6, 8, 4, 10, 15>
15	<10, 8, 5, 12, 5>
16	<4, 10, 6, 8, 5>
17	<4, 8, 6, 10, 20>
18	<10, 4, 10, 12, 5>
19	<6, 4, 15, 8, 5>
20	<4, 6, 8, 10, 5>
21	<4, 10, 8, 5, 10>
22	<5, 8, 6, 10, 5>
23	<6, 10, 15, 10, 5>
24	<4, 10, 15, 10, 4>
25	<15, 4, 8, 10, 15>
26	<4, 15, 20, 10, 5>
27	<10, 6, 15, 10, 5>
28	<10, 5, 20, 15, 10>
29	<4, 8, 3, 20, 5>
30	<20, 4, 8, 5, 10>

Содержание отчета

1. Привести, используя псевдокоды, перечисленные ниже алгоритмы:
   - вычисление оптимальной стоимости;
   - построение оптимального решения;
   - вывод решения на экран (расстановка скобок).
2. Продемонстрировать эти алгоритмы на примере (см. задание к лабораторной работе).
   В данном случае требуется привести полученный массив стоимостей (если возможно – на каждом шагу) и массив для вычисления оптимального решения, а также само решение в виде списка матриц, с расставленным скобками.

Пример:

Для $A_1 = (5\text{x}10)$, $A_2 = (10\text{x}15)$, $A_3 = (15\text{x}5)$

m

1000	1000	0
1500	0	
0		

– массив стоимостей

s

2	2	0
1	0	
0		

– массив для расстановки скобок

Вывести решение.

$(A_1(A_2A_3))$ - решение.

3. Результаты работы программ.
4. Вывод.
5. Листинги программ.

Контрольные вопросы:

1. Какое значение имеет данный подход к построению алгоритмов в вопросе оптимизации работы программы?
2. Каким образом заполняется массив стоимостей каждого перемножения?
3. Почему требуется запоминать минимальную стоимость каждого перемножения?
4. Знаете ли вы другие подходы для решения данной проблемы и каковы они?
5. Всегда ли есть возможность расставить скобки?
6. К какому типу оптимизации алгоритмов относится данная задача?

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. T. CORMEN, Ch. LEISERSON, R. RIVEST. Introducere în algoritmi. Computer Libris Agora, Cluj-Napoca, 2000.
2. OLTEAN Mihai. Proiectarea şi implementarea algoritmilor. Computer Libris Agora, Cluj-Napoca, 1999.
3. КОРМЕН Т., ЛЕЙЗЕРСОН Ч., РИВЕСТ Р. Алгоритмы: построение и анализ. – М.: МЦНМО, 2001. – 960с.
4. Donald E. KNUTH. Seminumerical Algorithms, volume 2 of The Art of Computer Programming. Addison-Wesley, 1981.
5. А. АХО, Д. УЛЬМАН, Д. ХОПКРОФТ. Структуры данных и алгоритмы. Издательский дом «Вильямс», 2000 – 348 с.
6. Хэзфилд Р. Кирби Л. и др. Искусство программирования на С. Фундаментальные алгоритмы, структуры данных и примеры приложений. Энциклопедия программиста. К.: Издательство “ДиаСофт”, 2001. – 736 с.