МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА 25

КУРСОВАЯ РАБОТА (ПРОЕКТ)   
ЗАЩИЩЕНА С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| доц., канд. техн. наук |  |  |  | Е.М. Линский |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ (ПРОЕКТУ) |
| ЗАДАЧА О НАЗНАЧЕНИЯХ |
| по дисциплине: |
| Основы программирования |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ гр. № | 2354 |  |  |  | Фомин А.А. |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2024

Оглавление

[Постановка задачи 3](#_Toc185952764)

[Алгоритм 4](#_Toc185952765)

[1.1 Алгоритм полного перебора 4](#_Toc185952766)

[1.2 Пошаговый пример для алгоритма 5](#_Toc185952767)

[2.1 Жадный алгоритм 8](#_Toc185952768)

[2.2 Пошаговый пример для алгоритма 9](#_Toc185952769)

[3.1 Псевдокод 11](#_Toc185952770)

[4.1 Сравнение алгоритмов 12](#_Toc185952771)

[Инструкция пользователя 13](#_Toc185952772)

[Тестовые примеры 13](#_Toc185952773)

[Заключение 17](#_Toc185952774)

[Список литературы 17](#_Toc185952775)

[ПРИЛОЖНИЕ 17](#_Toc185952776)

# Постановка задачи

Задача о назначениях состоит в отыскании назначения, имеющего минимальную стоимость и при котором каждая работа выполняется некоторым человеком.

Дополнительные требования к задаче:

* Имеется n работников и n работ.
* Стоимость назначения i-ого работника на j-ую работу равна c\_{i,j}.
* Вход и выход программы: матрица работа-работники.
* Для решения данной задачи требуется написать два алгоритма: полного перебора и жадный.
* Провести исследование и сравнить время и точность жадного алгоритма и полного перебора.

Для задания использовалась следующая литература: А. Кофман, Введение в прикладную комбинаторику, Наука, 1975 г

https://publ.lib.ru/ARCHIVES/K/KOFMAN\_Arnold\_(injener,\_matematik)/%ca%ee%f4%ec%e0%ed%20%c0.\_%20%c2%e2%e5%e4%e5%ed%e8%e5%20%e2%20%ef%f0%e8%ea%eb%e0%e4%ed%f3%fe%20%ea%ee%ec%e1%e8%ed%e0%f2%ee%f0%e8%ea%f3.(1975).pdf

Пример, показывающий, что задача имеет смысл (таблица 1):

Таблица 1

|  |  |  |  |
| --- | --- | --- | --- |
|  | Работа 1 | Работа 2 | Работа 3 |
| Работник 1 | 100 | 200 | 300 |
| Работник 2 | 150 | 150 | 150 |
| Работник 3 | 200 | 150 | 250 |

Из таблицы, очевидно, следует, что лучшим назначением будет: работник 1 на работу 1, работник 2 на работу 3, работник 3 на работу 2. Так как общая стоимость будет минимальной и равна 400. При этом распределение, например, работник 1 на работу 1, работник 2 на работу 2, работник 3 на работу 3, будет стоить 500.

# Алгоритм

В решении данной задачи требуется построить два алгоритма: полного перебора и жадный.

Двумерный вектор **cost** хранит всю матрицу работников-работ.

## 1.1 Алгоритм полного перебора

Алгоритм полного перебора подразумевает рассмотрение абсолютно всех вариантов назначений и выбор наилучшего.

После запуска функции, рассмотренной в приложении 1, в вектор *perm* записываются числа от 1 до n (рисунок 1). Этот вектор нужен для рассмотрения всех расстановок. Переменная *minCost* принимает максимально возможное числовое значение. И так как требуется найти минимальное, оно будет обновляться с каждым новым минимальным найденным значением.

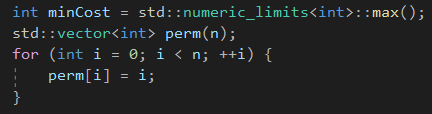


Рисунок 1

Затем, проходя по циклу, рассматривающим все размещения (внешний цикл), для каждого конкретного размещения создается переменная *currentCost* равная 0. Для каждого работника *i* от 0-ого до n-ого выбирается работа под номером, стоящим в размещении *perm*, имеющим индекс *i* (внутренний цикл). Сумма назначения записывается в *currentCost*. Если *currentCost* меньше чем *minCost*, то в *minCost* записывается значение *currentCost* и текущее размещение *perm* записывается в общее назначение **assignment**. (рисунок 2)

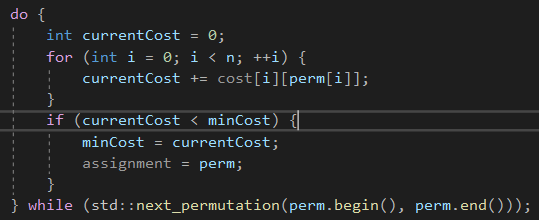


Рисунок 2

Асимптотическая сложность алгоритма полного перебора: *O(n!)*

### **1.2 Пошаговый пример для алгоритма**

Дана матрица 3 на 3 (таблица 2)

Таблица 2

|  |  |  |  |
| --- | --- | --- | --- |
|  | Работа 0 | Работа 1 | Работа 2 |
| Работник 0 | 10 | 19 | 5 |
| Работник 1 | 10 | 18 | 2 |
| Работник 2 | 13 | 16 | 3 |

Следовательно, n = 3, *perm* = {0, 1, 2}. Программа заходит во внешний цикл.

Самой первой расстановкой, которую рассмотрит полный перебор, будет (0, 1, 2).

Первая итерация внутреннего цикла:

*i* = 0

*currentCost* = *cost*[0][*perm*[0]] = *cost*[0][0] = 10.

Вторая итерация внутреннего цикла:

*i* = 1

*currentCost* = *currentCost* + *cost*[1][*perm*[1]] = 10 + *cost*[1][1] = 10 + 18 = 28.

Третья итерация внутреннего цикла:

*i* = 2

*currentCost* = *currentCost* + *cost*[2][*perm*[2]] = 28 + cost[2][2] = 28 + 3 = 31.

*currentCost* < *minCost* = true

*minCost* = *currentCost* = 31

Вторая итерация внешнего цикла (0, 2, 1):

Первая итерация внутреннего цикла:

*i* = 0

*currentCost* = *cost*[0][*perm*[0]] = *cost*[0][0] = 10.

Вторая итерация внутреннего цикла:

*i* = 1

*currentCost* = *currentCost* + *cost*[1][*perm*[1]] = 10 + *cost*[1][2] = 10 + 2 = 12.

Третья итерация внутреннего цикла:

*i* = 2

*currentCost* = *currentCost* + *cost*[2][*perm*[2]] = 12 + *cost*[2][1] = 12 + 16 = 28.

*currentCost* < *minCost* = true

*minCost* = *currentCost* = 28

Третья итерация внешнего цикла (1, 0, 2):

Первая итерация внутреннего цикла:

*i* = 0

*currentCost* = *cost*[0][*perm*[0]] = *cost*[0][1] = 19.

Вторая итерация внутреннего цикла:

*i* = 1

*currentCost* = *currentCost* + *cost*[1][*perm*[1]] = 19 + *cost*[1][0] = 19 + 10 = 29.

Третья итерация внутреннего цикла:

i = 2

*currentCost* = *currentCost* + *cost*[2][*perm*[2]] = 12 + *cost*[2][2] = 29 + 3 = 32.

*currentCost* < *minCost* = false

*minCost* = 28

Четвертая итерация внешнего цикла (1, 2, 0):

Первая итерация внутреннего цикла:

*i* = 0

*currentCost* = *cost*[0][*perm*[0]] = *cost*[0][1] = 19.

Вторая итерация внутреннего цикла:

*i* = 1

*currentCost* = *currentCost* + *cost*[1][*perm*[1]] = 19 + *cost*[1][2] = 19 + 2 = 21.

Третья итерация внутреннего цикла:

*i* = 2

*currentCost* = *currentCost* + *cost*[2][*perm*[2]] = 12 + *cost*[2][0] = 21 + 13 = 34.

*currentCost* < *minCost* = false

*minCost* = 28

Пятая итерация внешнего цикла (2, 0, 1):

Первая итерация внутреннего цикла:

*i* = 0

*currentCost* = *cost*[0][*perm*[0]] = *cost*[0][2] = 5.

Вторая итерация внутреннего цикла:

*i* = 1

*currentCost* = *currentCost* + *cost*[1][*perm*[1]] = 5 + *cost*[1][0] = 5 + 10 = 15.

Третья итерация внутреннего цикла:

*i* = 2

*currentCost* = *currentCost* + *cost*[2][*perm*[2]] = 15 + *cost*[2][1] = 15 + 16 = 31.

*currentCost* < *minCost* = false

*minCost* = 28

Шестая итерация внешнего цикла (2, 1, 0):

Первая итерация внутреннего цикла:

*i* = 0

*currentCost* = *cost*[0][*perm*[0]] = *cost*[0][2] = 5.

Вторая итерация внутреннего цикла:

*i* = 1

*currentCost* = *currentCost* + *cost*[1][*perm*[1]] = 5 + *cost*[1][1] = 5 + 18 = 23.

Третья итерация внутреннего цикла:

*i* = 2

*currentCost* = *currentCost* + *cost*[2][*perm*[2]] = 23 + *cost*[2][0] = 23 + 13 = 36.

*currentCost* < *minCost* = false

*minCost* = 28

Итого *minCost* = 28, и **assignment** = {0, 2, 1}.

## 2.1 Жадный алгоритм

Жадный алгоритм нужен для того, чтобы сэкономить время работы программы, при этом, не сильно потеряв точность. Реализацией жадного алгоритма является выбор наилучшей работы для работников по очереди.

После запуска функции, показанной в приложении 2, создается булевой вектор **assigned** размера n, для хранения номеров уже назначенных работ (рисунок 3). И создается переменная *totalCost*, хранящая сумму назначений.



Рисунок 3

Запускается внешний цикл для работника *i* от 0-ого до n-ого, создаются переменные *minCost* и *jobIndex*. (Рисунок 4). Затем во внутреннем цикле, проходящем по работам *j* от 0-ой до n-ой, и если на эту работу еще не назначен работник, и назначение на *j*-ую работу минимальное для *i*-ого работника, то она назначается ему. К *totalCost* прибавляется это назначение.

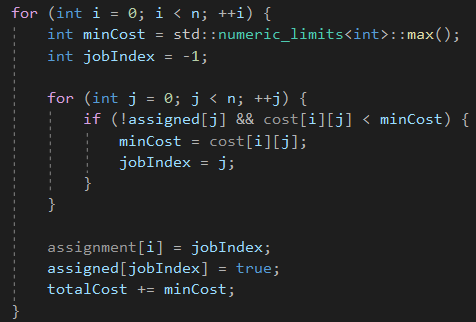


Рисунок 4

Асимптотическая сложность алгоритма полного перебора: O().

### **2.2 Пошаговый пример для алгоритма**

Дана матрица 3 на 3(таблица 2)

n = 3, **assigned** = {false, false, false}.

Программа заходит во внешний цикл:

Первая итерация внешнего цикла:

*i* = 0

Первая итерация внутреннего цикла:

*j* = 0

**assigned**[0] = false && *cost*[0][0] < *minCost*

*minCost* = 10

*jobIndex* = 0

Вторая итерация внутреннего цикла:

*j* = 1

**assigned**[1] = false && *cost*[0][1] > *minCost*

*minCost* = 10

*jobIndex* = 0

Третья итерация внутреннего цикла:

*j* = 2

**assigned**[2] = false && *cost*[0][2] < *minCost*

*minCost* = 5

*jobIndex* = 2

**assignment**[0] = 2

**assigned**[2] = true

*totalCost* = 0 + 5 = 5

Вторая итерация внешнего цикла:

*i* = 1

Первая итерация внутреннего цикла:

*j* = 0

**assigned**[0] = false && *cost*[1][0] < *minCost*

*minCost* = 10

*jobIndex* = 0

Вторая итерация внутреннего цикла:

*j* = 1

**assigned**[1] = false && *cost*[1][1] > *minCost*

*minCost* = 10

*jobIndex* = 0

Третья итерация внутреннего цикла:

*j* = 2

**assigned**[2] = true && *cost*[1][2] < *minCost*

*minCost* = 10

*jobIndex* = 0

**assignment**[1] = 0

**assigned**[0] = true

*totalCost* = 5 + 10 = 15

Третья итерация внешнего цикла:

*i* = 2

Первая итерация внутреннего цикла:

*j* = 0

**assigned**[0] =true && *cost*[2][0] < *minCost*

*minCost* = ∞

*jobIndex* = -1

Вторая итерация внутреннего цикла:

*j* = 1

**assigned**[1] = false && *cost*[2][1] < minCost

*minCost* = 16

*jobIndex* = 1

Третья итерация внутреннего цикла:

*j* = 2

**assigned**[2] = true && *cost*[2][2] < *minCost*

*minCost* = 16

*jobIndex* = 1

**assignment**[2] = 1

**assigned**[1] = true

*totalCost* = 15 + 16 = 31

Итого *totalCost* = 31, **assignment** = {2, 0, 1}.

## 3.1 Псевдокод

Задача о назначениях(n)

Вход: матрица n на n, с натуральными числами – стоимостью назначения работников на работы.

Выход: минимальное назначение полным перебором и жадным алгоритмом.

Полный перебор:

minCost = ∞

perm(n)

для каждой расстановки(perm(n)):

currentCost = 0

для каждого назначения работника(n):

currentCost += cost[i][perm[i]]

if (currentCost < minCost):

minCost = currentCost

assignment = perm

Жадный алгоритм:

assigned(n, false)

totalCost = 0

для каждого работника(n):

minCost = ∞

jobIndex = -1;

для каждой работы(n):

if (!assigned[j] && cost[i][j] < minCost):

minCost = cost[i][j]

jobIndex = j

assignment[i] = jobIndex

assigned[jobIndex] = true

totalCost += minCost

## 4.1 Сравнение алгоритмов

Таблица 3

|  |  |  |
| --- | --- | --- |
| Число n – кол-во работников | Полный перебор | Жадный алгоритм |
| 1 | 500 нс | 600 нс |
| 2 | 600 нс | 700 нс |
| 3 | 800 нс | 800 нс |
| 4 | 1100 нс | 1100 нс |
| 5 | 10500 нс | 1200 нс |
| 6 | 18500 нс | 1300 нс |
| 7 | 155200 нс | 1400 нс |
| 8 | 666600 нс | 1300 нс |
| 9 | 6379700 нс | 1400 нс |
| 10 | 76570300 нс | 1400 нс |
| 11 | 761096900 нс | 1600 нс |
| 12 | 9096906400 нс | 1700 нс |
| 13 | 119710262400 нс | 1800 нс |
| 14 | 1854436035900 нс | 2000 нс |

В таблице 3 представлена зависимость времени выполнения алгоритмов от числа n. График зависимости представлен на рисунке 5.

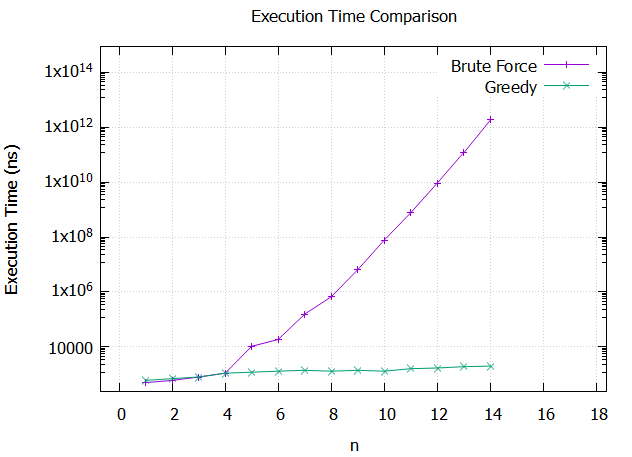


Рисунок 5

Для установления точности жадного алгоритма был проведен эксперимент. Для 10 случайных матриц размером n на n посчитана потеря жадного алгоритма от лучшего назначения, посчитано их среднее значение. График зависимости точности жадного алгоритма от n представлен на рисунке 6.

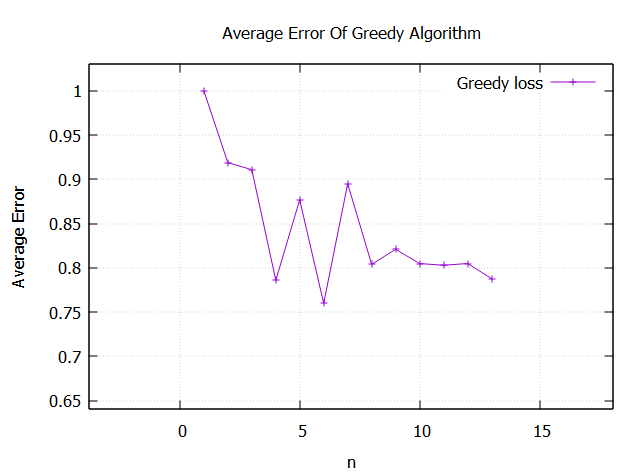


Рисунок 6

# Инструкция пользователя

Перед тем как запускать программы пользователю необходимо создать текстовый файл, например input.txt. Чтобы не пришлось указывать расположение файла, его рекомендуется поместить в директорию программы, чтобы в командной строке писать просто input.txt, вместо, например, D:\program\course\course\input.txt.

В первой строке файла написано натуральное число n, затем идет матрица n на n натуральных чисел. Каждой строке соответствует работник, каждой работе соответствует работа.

Программа выполняется автоматически, и результат выводится в файл output.txt в той же директории.

# **Тестовые примеры**

Пример 1:

input:

3

10 19 5

10 18 2

13 16 3

output:

Brute force:

Minimal cost: 28

Workers arrangement: Worker 1 job 1

Worker 2 job 3

Worker 3 job 2

Greedy algorithm:

Minimal cost: 31

Assignments: Worker 1 job 3

Worker 2 job 1

Worker 3 job 2

Пример 2:

input:

6

10 16 19 9 19 15

9 15 17 16 13 12

11 10 10 10 9 8

12 14 13 11 10 15

9 9 9 17 17 19

20 9 17 20 8 15

output:

Brute force:

Minimal cost: 54

Workers arrangement: Worker 1 job 4

Worker 2 job 1

Worker 3 job 6

Worker 4 job 5

Worker 5 job 3

Worker 6 job 2

Greedy algorithm:

Minimal cost: 62

Assignments: Worker 1 job 4

Worker 2 job 1

Worker 3 job 6

Worker 4 job 5

Worker 5 job 2

Worker 6 job 3

Пример 3:

input:

12

10 16 19 9 19 15 9 15 17 16 13 12

12 14 13 11 10 15 9 9 9 17 17 19

10 16 19 9 19 15 9 9 9 17 17 19

12 14 13 11 10 15 15 17 16 13 12 14

20 9 17 20 8 15 20 9 17 20 8 15

11 10 10 10 9 8 11 10 10 10 9 8

10 16 19 9 19 15 9 15 17 16 13 12

12 14 13 11 10 15 9 9 9 17 17 19

10 16 19 9 19 15 9 9 9 17 17 19

12 14 13 11 10 15 15 17 16 13 12 13

20 9 17 20 8 15 20 9 17 20 8 15

11 10 10 10 9 8 11 10 10 10 9 8

output:

Brute force:

Minimal cost: 115

Workers arrangement: Worker 1 job 1

Worker 2 job 3

Worker 3 job 4

Worker 4 job 5

Worker 5 job 2

Worker 6 job 6

Worker 7 job 7

Worker 8 job 8

Worker 9 job 9

Worker 10 job 10

Worker 11 job 11

Worker 12 job 12

Greedy algorithm:

Minimal cost: 126

Assignments: Worker 1 job 4

Worker 2 job 7

Worker 3 job 8

Worker 4 job 5

Worker 5 job 11

Worker 6 job 6

Worker 7 job 1

Worker 8 job 9

Worker 9 job 2

Worker 10 job 3

Worker 11 job 12

Worker 12 job 10

# Заключение

В данной работе были рассмотрены два алгоритма для решения задачи о назначениях: алгоритм полного перебора и жадный алгоритм. Также было выявлено, что алгоритм полного перебора, несмотря на свою точность, работает очень долго, уже для матрицы 20 на 20 алгоритм будет работать 30 лет. Жадный же алгоритм работает быстро, но при этом теряет свою точность.

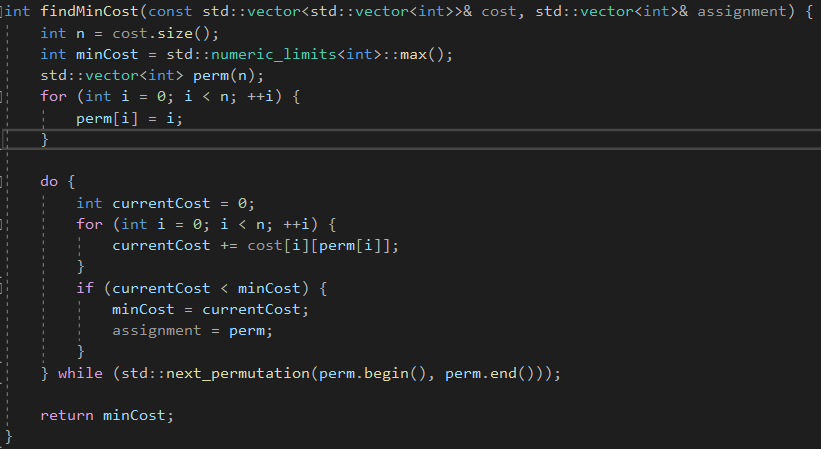
Для более точного вычисления времени работы алгоритмов, используется библиотека chrono, благодаря которой время работы алгоритма можно вычислять в наносекундах. График зависимости времени выполнения алгоритмов от числа n сделан в логарифмическом масштабе для более удобного представления.

# Список литературы

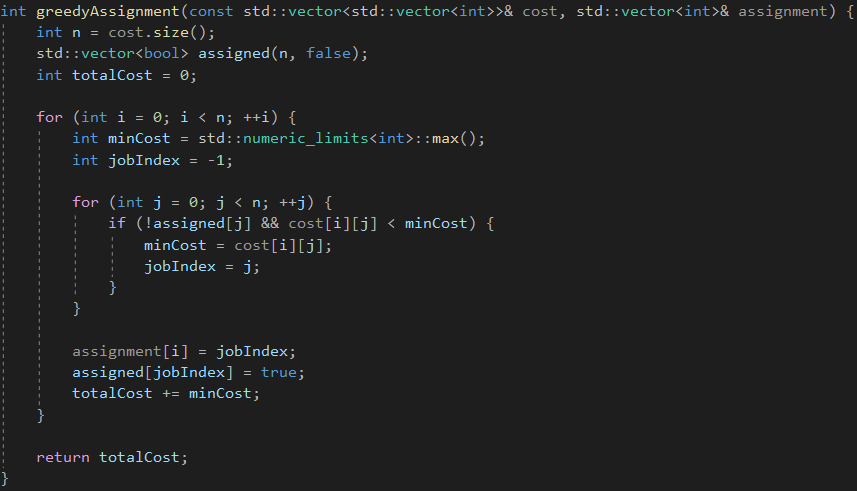
Кофман, А. Введение в прикладную комбинаторику / А. Кофман; Наука. – М., 1975. – 477 с.

Рейнгольд, Э. Комбинаторные алгоритмы / Э. Рейнгольд, Ю. Нивергельт, Н. Део; Мир. – М., 1980. – 476 с.

# ПРИЛОЖНИЕ



**Приложение 1. Алгоритм полного перебора**

****

**Приложение 2. Жадный алгоритм**