**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Чорноморський національний університет   
імені Петра Могили**

**Факультет комп’ютерних наук**

**Кафедра інтелектуальних інформаційних систем**

**ЗВІТ**

**ПРО НАВЧАЛЬНУ ПРАКТИКУ**

122 – ПРН.ПЗ.00 – 101.1910109

Виконав: студент 1 курсу, групи 101

Спеціальності   
122 «Комп’ютерні науки»

Грабовський Є.О.

(підпис, ініціали та прізвище)

(дата)

Керівник: в.о. викладач каф. ІІС

М.О. Таранов

(підпис, ініціали та прізвище)

(дата)

**м. Миколаїв  – 2020 рік**

**ЗМІСТ**

[ВСТУП 2](#_Toc37078951)

[ЗМІСТОВА ЧАСТИНА 3](#_Toc37078952)

[1. Жадібний алгоритм 3](#_Toc37078953)

[2. Приклад 1 – «Задача на заявки» 4](#_Toc37078954)

[3. Приклад 2 – «Коди Хаффмена» 5](#_Toc37078955)

[ВИКОНАННЯ 11](#_Toc37078956)

[1. Задача 1 12](#_Toc37078957)

[Умова 12](#_Toc37078958)

[Реалізація 12](#_Toc37078959)

[2.Задача 2 13](#_Toc37078960)

[Умова 13](#_Toc37078961)

[Пояснення до задачі 2 13](#_Toc37078962)

[Реалізація 13](#_Toc37078963)

[3.Задача 3 14](#_Toc37078964)

[Умова 14](#_Toc37078965)

[Реалізація 14](#_Toc37078966)

[ВИСНОВКИ 16](#_Toc37078967)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 17](#_Toc37078968)

# **ВСТУП**

Навчальна практика, що проходила у Чорноморському державному університеті імені Петра Могили на кафедрі «Інтелектуальних інформаційних систем» з 23 березня по 5 квітня, націлена на практичне опрацювання навичок набутих протягом 2 триместрів навчання. Завданням на практичне опрацювання було: дізнатися про «жадібний алгоритм» використовуючи можливості Microsoft Visual Studio 2017 та знань мови програмування C++. При виконані завдання дотримуватися вимогам поставленого завдання.

Моєю задачею є «жадібні алгоритми», а саме: дізнатися, що це, в чому їх перевага, як їх використовувати, та привести приклади програм з використанням «жадібного алгоритму».

# **ЗМІСТОВА ЧАСТИНА**

Основним завданням проходження практики було написати код програми, де використовується «жадібний алгоритм». Для того, щоб виконати поставлене завдання було використано програму Microsoft Visual Studio 2017. Для того, щоб написати програму , яка містить цей алгоритм , потрібно було ознайомитись з ним.

## **Жадібний алгоритм**

Жадібний алгоритм — простий і прямолінійний евристичний алгоритм, який приймає найкраще рішення, виходячи з наявних на поточному етапі даних, не турбуючись про можливі наслідки, сподіваючись врешті-решт отримати оптимальне рішення. Легкий в реалізації і часто дуже ефективний за часом виконання. Багато задач не можуть бути розв'язані з його допомогою.

Зазвичай, жадібний алгоритм базується на п'яти принципах:

1. Набір можливих варіантів, з яких робиться вибір;
2. Функція вибору, за допомогою якої знаходиться найкращий варіант;
3. Функція придатності, яка визначає придатність отриманого набору;
4. Функція цілі, оцінює цінність рішення, не виражена явно;
5. Функція розв'язку, яка вказує на те, що знайдене кінцеве рішення.

Придатний набір варіантів — такий, що обіцяє не просто отримання рішення, а отримання оптимального рішення задачі.

На відміну від динамічного програмування, що розв'язує проблему знизу догори, жадібна стратегія робить це згори донизу, роблячи один жадібний вибір за іншим, зводячи велику задачу до малої.

Наприклад, використання жадібної стратегії для задачі комівояжера породжує наступний алгоритм: «На кожному етапі вибирати найближче з невідвіданих міст».

Для багатьох оптимізаційних задач є більш прості і швидкі алгоритми, ніж динамічне програмування.Такий алгоритм робить на кожному кроці локально оптимальний вибір, - в надії, що підсумкове рішення також виявиться оптимальним. Це не завжди так - але для багатьох завдань такі алгоритми справді дають оптимум. Наш перший приклад - проста, але не цілком тривіальна задача про вибір заявок. Далі ми обговорюємо, для яких завдань підходять жадібні алгоритми.

## **Приклад 1 – «Задача на заявки»**

Нехай дано n заявок на проведення занять в одній і тій же аудиторії. Два різних заняття не можуть проводитися в один і той ж час. У кожній заявці вказані початок і кінець заняття (si и fi для заявки і). Різні заявки можуть перетинатися, і тоді можна задовольнити тільки одну з них. Ми ототожнюємо кожну заявку з проміжком [si , fi), так що кінець одного заняття може збігатися з початком іншого, і це не вважається перетином. Формально кажучи, заявки з номерами i та j сумісні, якщо інтервали [si , fi) та [sj,fj) не перетинаються (інакше кажучи, якщо

fi ≤ sj або fj ≤ si). Завдання про вибір заявок полягає в тому, щоб набрати максимальну кількість спільних між собою заявок .

Жадібний алгоритм працює наступним чином. Ми припускаємо, що заявки впорядковані в порядку зростання часу закінчення:

Якщо це не так, то можна відсортувати їх за час O(n log n); заявки з однаковим часом кінця маємо в довільному порядку.

Тоді алгоритм виглядає так (f та s – масиви):

Greedy-Activity- Selector (s, f)

1 n ← length [s]

2 A ← {1}

3 j ← 1

4 for i ← 2 to n

do if si ≥ fj

then A ← A ∪{i}

j ← i

return A

Робота цього алгоритму показана зверху. Множина F складається з номерів обраних заявок, j – номер останньої з них; при цьому

оскільки заявки відсортовані по зростанню часу закінчення. Спочатку А містить заявку номер 1, і j = 1 (рядки 2-3). Далі (цикл в рядках 4-7) шукається заявка, що починається не раніше закінчення заявки номер j. Якщо така знайдена, вона включається в множину Ф і змінної j присвоюється її номер (рядки 6-7).

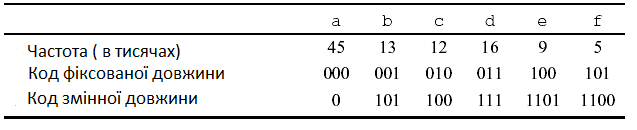
Жадібний алгоритм вимагає всього лише θ (n) кроків (не рахуючи попереднього сортування). Як і личить жадібному алгоритму, на кожному кроці він робить вибір так, щоб залишається вільним часом було максимально.

## **Приклад 2 – «Коди Хаффмена»**

Коди Хаффмена (Huffman codes) широко використовуються при стисненні інформації. Ці коди можна побудувати за допомогою жадібного алгоритму, який був запропонований Хаффменом.

**Коди Хаффмена** - широко поширений і дуже ефективний метод стиснення даних, який, залежно від характеристик цих даних, зазвичай дозволяє заощадити від 20% до 90% об'єму. Ми розглядаємо дані, що представляють собою послідовність символів. У жадібному алгоритмі Хаффмана використовується таблиця, що містить частоти появи тих чи інших символів. За допомогою цієї таблиці визначається оптимальне представлення кожного символу у вигляді бінарної рядка. Припустимо, що є файл даних, що складається з 100 000 символів, який потрібно стиснути. Символи в цьому файлі зустрічаються з частотою, представленої в табл. 9.1. Таким чином, всього файл містить шість різних символів, а, наприклад, символ a зустрічається в ньому 45000 разів.

**Таблиця 9**.1 Задача про кодування послідовності символів



Існує безліч способів кодування інформації, представленою у файлі даних. Якщо розглядати задачу з розробки бінарного коду символів (binary character code), в якому кожен символ представляється унікальним бінарним рядком. Якщо використовується код фіксованої довжини, або рівномірний код (xed-length code), то для представлення шести символів знадобиться 3 біта: a = 000, b = 001,. . . , f = 101. При використанні такого методу для кодування всього файлу знадобиться 300000 бітів. Чи можна домогтися кращих результатів?

За допомогою коду змінної довжини, або нерівномірного коду (variable length code), вдається отримати значно кращі результати, ніж за допомогою коду фіксованої довжини. Це досягається за рахунок того, що символам, які найчастіше зустрічаються в тексті зіставляються короткі кодові слова, а тим, що рідко зустрічаються - довгі. Такий код представлений в останньому рядку табл. 8.1. У ньому символ a представлений 1-бітовим рядком 0, а символ f - 4-бітовим рядком 1100. Для подання файла за допомогою цього коду потрібно (45 · 1 + 13 · 3 + 12 · 3 + 16 · 3 + 9 · 4 + +5 4) · 1000 = 224000 бітів.

Отже, завдяки нерівномірного кодування додатково економиться 25% об'єму пам'яті.

Далі ми розглядаємо лише коди, в яких з двох послідовностей бітів, що представляють різні символи, жодна не є префіксом іншого - префіксние коди. Можна показати (хоча тут ми не станемо цього робити), що оптимальне стиснення даних, якого можна досягти за допомогою кодів, завжди досяжно при використанні префіксного коду, тому розгляд одних лише префіксних кодів не приводить до втрати спільності.

Для будь-якого бінарного коду символів кодування тексту - дуже простий процес: треба просто з'єднати кодові слова, що представляють кожен символ у тексті. Наприклад, в кодуванні за допомогою префіксного коду змінної довжини, представленого в табл. 8.1, трьосимвольний файл abc має вигляд 0 · 101 · 100 = 0101100, де символом "·" позначена операція конкатенації.

Перевагу префіксним кодами додає ще те, що вони спрощують декодування Оскільки ніяке кодове слово не виступає в ролі префікса іншого, кодове слово, з якого починається закодований файл, визначається однозначно. Початкову кодове слово легко ідентифікувати, перетворити його у вихідний символ і продовжити декодування решти закодованого файлу. У розглянутому прикладі рядок 001011101 однозначно розкладається на підрядка 0 · 0 · 101 · 1101, що декодується як aabe.

Для спрощення процесу декодування потрібне зручне представлення префіксного коду, щоб кодове слово можна було легко ідентифікувати Одним з таких уявлень є бінарне дерево, листям якого є кодуємі символи. Бінарне кодове слово, що представляє символ, інтерпретується як шлях від кореня до цього символу. У такій інтерпретації 0 означає "перейти до лівого дочірньому вузлу", а 1 - "перейти до правого дочірньому вузлу". На рис. 9.1 показані такі дерева для двох кодів, узятих з нашого прикладу. Кожен лист на малюнку позначений відповідним йому символом і частотою появи, а внутрішній вузол - сумою частот листя його піддерев. У частині а малюнка наведено дерево, відповідне коду фіксованої довжини, де a = 000,. . . , f = 101. У частині б показано дерево, відповідне оптимальним префіксним кодам a = 0, b = 101,. . . , f = 1100. Зауважимо, що зображені на малюнку дерева не є бінарними деревами пошуку, оскільки листя в них не обов'язково розташовані в порядку сортування, прийнятим у бінарних деревах, а внутрішні вузли не містять ключів символів.

Оптимальний код файлу завжди може бути представлений повним бінарним деревом, в якому у кожного вузла (крім листя) є по два дочірніх вузла.

Код фіксованої довжини, представлений в розглянутому прикладі, не є оптимальним, оскільки відповідне йому дерево, зображене на рис. 9.1а, - неповне бінарне дерево: деякі слова коду починаються з 10.. . , Але жодне з них не починається з 11.. . . Оскільки в нашому обговоренні ми можемо обмежитися тільки повними бінарними деревами, можна стверджувати, що якщо C - алфавіт, з якого здобуваються кодуються символи, і всі частоти, з якими зустрічаються символи, додатні, то дерево, яке представляє оптимальний префіксний код, містить рівно | C | листя, по одному для кожного символу з множини C, і рівно | C | - 1 внутрішніх вузлів.

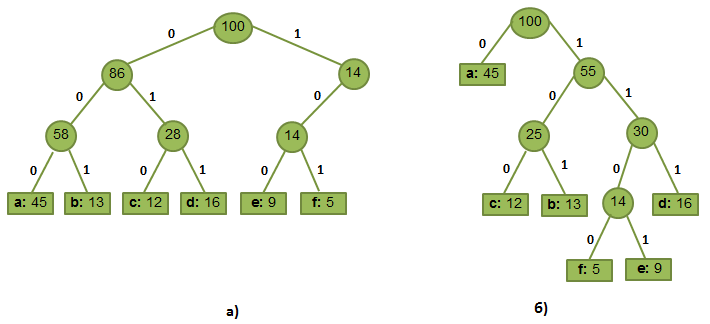


Рисунок 9.2   - Дерева, що відровідає схемам кодування, приведенним в табл. 9.1

Досить легко знайти кількість бітів, необхідне для кодування файлу, якщо ми знаємо дерево T, відповідне префіксний коду. Нехай f [c] позначає число входжень символу c з алфавіту C у файл, dT (c) - глибину відповідного листа в дереві (довжину послідовності бітів, що кодує c). Отримуємо, що для кодування файлу необхідно стільки бітів:

 . Назвемо це число вартістю дерева T.

Наведемо жадібний алгоритм Хаффмена, що будує оптимальний префіксний код - код Хаффмена. При цьому припускаємо, що для будь-якого символу c ∈ C задана його частота f [c]. В алгоритмі будується дерево T, відповідне оптимальній коду, причому побудова йде у висхідному напрямку. Процес побудови починається з множини, що складається з | C | листя, після чого послідовно виконується | C | - 1 операцій "злиття", в результаті яких утворюється кінцеве дерево. Для ідентифікації двох зустрічаються найменше об'єктів, що підлягають злиттю, використовується чергу з пріоритетами Q, ключами в якій є частоти f. В результаті злиття двох об'єктів утворюється новий об'єкт, частота появи якого є сумою частот об'єднаних об'єктів:

На кожному етапі показано вміст черги, елементи якої розсортовані в порядку зростання їх частот. На кожному кроці роботи алгоритму об'єднуються два об'єкти (дерева) з найнижчими частотами. Листя зображені у вигляді прямокутників, в кожному з яких вказана буква і відповідна їй частота. Внутрішні вузли представлені колами, що містять суму частот дочірніх вузлів. Ребро, що з'єднує внутрішній вузол з лівим дочірнім вузлом, має позначку 0, а ребро, що з'єднує його з правим дочірнім вузлом, - мітку 1. Слово для букви утворюється послідовністю міток на ребрах, що з'єднують корінь з листом, що представляє цю букву. Оскільки дану множину містить шість букв, розмір вихідної черги дорівнює 6 (частина а рисунка), а для побудови дерева потрібно п'ять злиттів. Проміжні етапи зображені у частинах б-д. Кінцеве дерево (рис. 9.2е) являє оптимальний префіксний код. Як вже говорилося, слово коду для букви - це послідовність міток на шляху від кореня до листа з цією буквою.

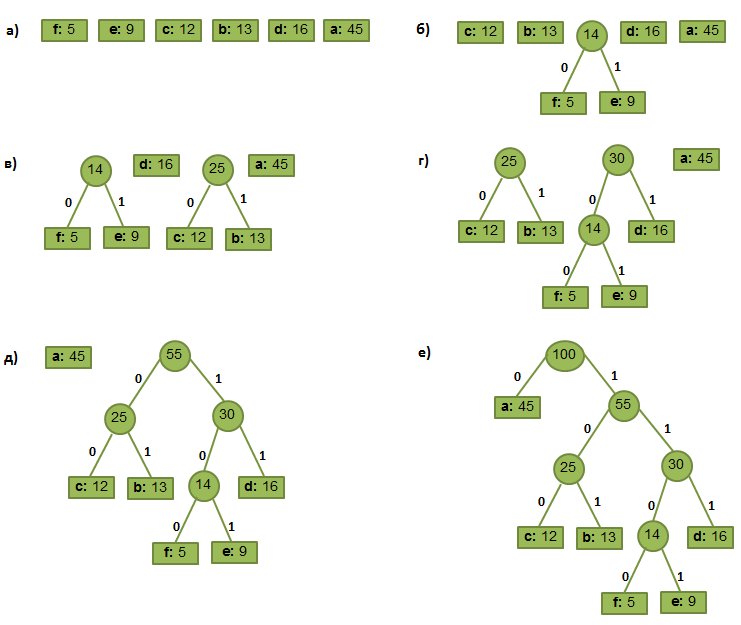


Рисунок 9.3   - Этапи роботи алгоритма Хаффмена для частот, заданих в табл.

# **Виконання**

Для виконання роботи, мною було обрано інтегроване середовище розробки програмного забезпечення Microsoft Visual Studio 2017 та мову програмування С++.

Для того, щоб показати «жадібний алгоритм» в дії я взяв декілька задач з сайту e.olymp.

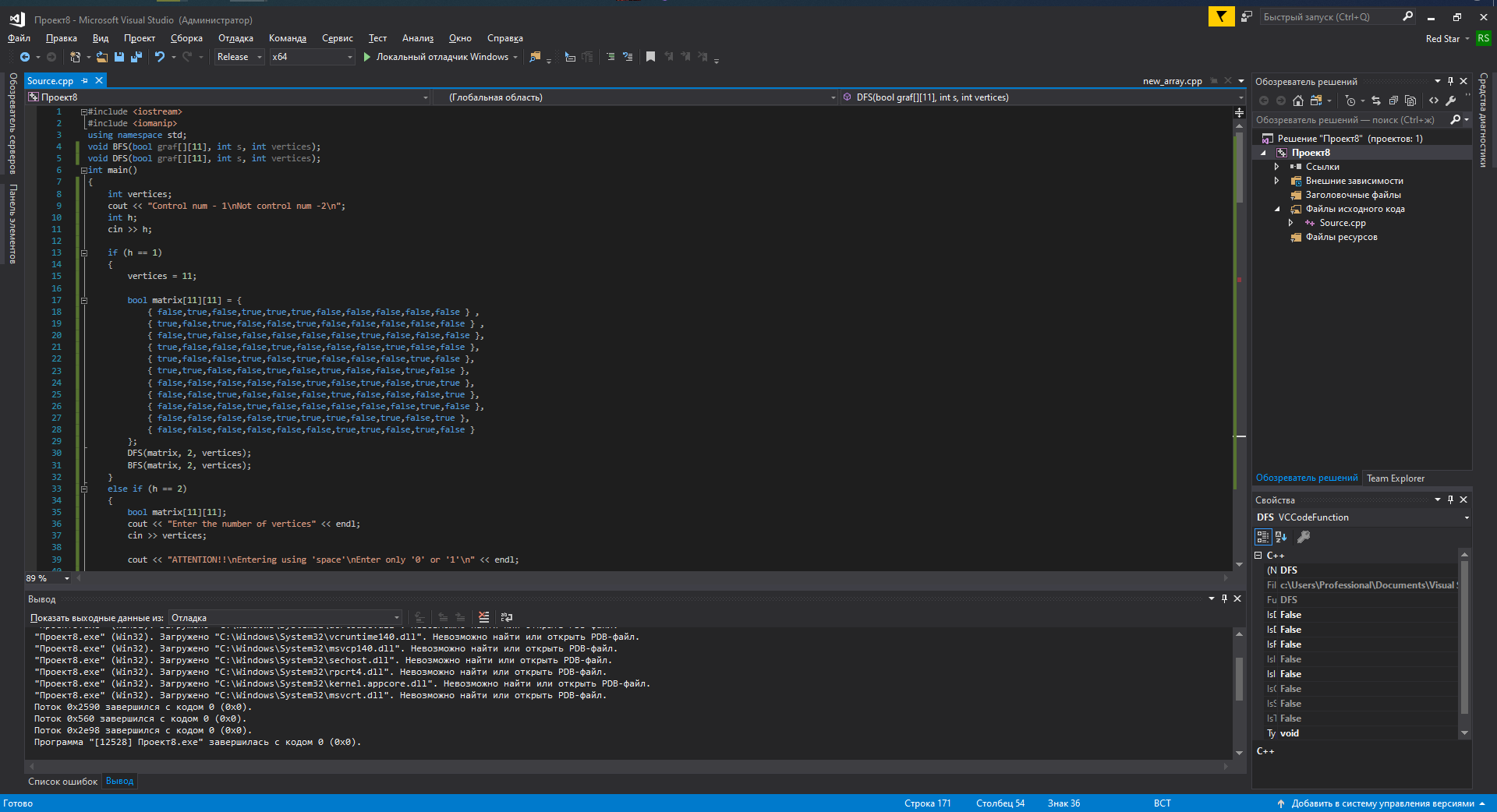


Рис. 1.3. Середовище розробки

## **Задача 1**

### **Умова**

В банкоматі є в достатній кількості купюри номіналом 10,20,50,100,200 и 500 грн.Знайти мінімальну кількість купюр,які необхідно використати,щоб видати сумму в n гривень,або вивести -1,якщо таку сумму видати не можна.

### **Реалізація**

#include <iostream>

using namespace std;

int main()

{

int bills[] = { 10, 20, 50, 100, 200, 500 };

int cash, x=0;

cin >> cash;

for (int i = 5; i >= 0; i--)

{

if (cash >= bills[i])

{

x += cash / bills[i];

cout << "The amount of " << bills[i] << " denomination - " << cash / bills[i] << endl;

cash %= bills[i];

}

}

if (cash > 0) x = -1;

cout << x;

system("pause");

return 0;

}

## **Задача 2**

### **Умова**

Професор Самодєлкін вирішив змайструвати об'ємну модель кубиків з сірників використовуючи сірники для ребер кубиків. Довжина ребра кожного кубика дорівнює одному сірнику.

Для побудови моделі трьох кубів в нього пішло 28 сірників.

Яку найменшу кількість сірників потрібно Самодєлкіну для побудови моделі з N кубиків.

Всі числа задачі не перевищують 2 \* 10^9

### **Пояснення до задачі 2**

Ця задача є прикладом такої,де не використовується «жадібний алгоритм».

### **Реалізація**

#include <iostream>

#include <math.h>

using namespace std;

int main()

{

int cubes, k = 1, i = 0, edges;

cin >> cubes;

for (; k<cubes; i++) {

k = pow(i + 1, 3);

}

if (!i)i++;

if (pow((i-1),3) < cubes && cubes <= (i - 1)\*(i - 1)\*i)

{

edges = 3 \* (i - 1)\*pow(i, 2);

cubes -= pow(i - 1, 3);

}

else if ((i - 1)\*(i - 1)\*i < cubes && cubes <= (i - 1)\*i\*i)

{

edges = 3 \* pow(i - 1, 3) + 9 \* pow(i - 1, 2) + 7 \* (i - 1) + 1;

cubes -= pow(i - 1, 2)\*i;

}

else if ((i - 1)\*i\*i < cubes && cubes <= pow(i,3))

{

edges = 3 \* (pow(i, 3) + pow(i, 2)) - i - 1;

cubes -= pow(i, 2)\*(i - 1);

}

else {

cout << "Overflow\n";

system("pause");

return 0;

}

for (int j = 1; cubes; j += 2)

{

for (int p = 0; p<j && cubes; p++, cubes--)

{

if (j == 1) {

edges += 8;

}

else if (p == 0 || p == j / 2) {

edges += 5;

}

else {

edges += 3;

}

}

}

cout << edges << endl;

system("pause");

return 0;

}

## **Задача 3**

### **Умова**

Для походу на Азерот Оргріму Думхаммеру знадобився ще один загін. На призив з'явились n орків. Здібності у ближньому бою та метанні спису кожного з них Оргрім відразу ж оцінив. Тепер він повинен визначити, кого з них призначити солдатом-піхотинцем (grunt), а кого метателем-мисливцем за головами (headhunter). При цьому, для того, щоб загін був боєздатним, необхідно, щоб в загоні було не менше g грунтів і не менше h хедхантерів. Після призначення кожного орка в якийсь рід військ, може бути визначена сила цього загону, як сума здібностей всіх орків в призначеній для них спеціалізації.

Напишіть програму, яка визначить максимально можливу силу заново сформованого загону.

### **Реалізація**

#include <iostream>

using namespace std;

int main()

{

int g, h, n;

cin >> n >> h >> g;

if (g + h > n)

{

cout << "The army cannot be created\n-1";

system("pause");

return 0;

}

int\*\* army = new int\*[n];

\*army = new int[n];

for (int i = 0; i < n; i++)

{

army[i] = new int[2];

cin >> army[i][0] >> army[i][1];

}

int numH = 0, numG = 0, sum = 0;

for (int i = 0; h != 0; i++)

{

h--;

for (int j = 0; j < n; j++)

{

if ((army[numH][0] - army[numH][1] < army[j][0] - army[j][1])|| army[numH][0] == -1)

{

numH = j;

}

}

sum += army[numH][0];

army[numH][0] = -1;

army[numH][1] = -1;

}

for (int i = 0; g != 0; i++)

{

g--;

for (int j = 0; j < n; j++)

{

if (( army[numG][1] - army[numG][0] < army[j][1]- army[j][0]) || army[numG][0] == -1)

{

numG = j;

}

}

sum += army[numG][1];

army[numG][0] = -1;

army[numG][1] = -1;

}

numG = 0;

for (int i = 0; i < n; i++)

{

if (army[i][0] != -1)

{

if (army[i][0] > army[i][1])

sum += army[i][0];

else

sum += army[i][1];

army[i][0] = -1;

army[i][1] = -1;

}

}

cout << sum << endl;

system("pause");

return 0;

}

### **ВИСНОВКИ**

Під час проходження практики я вдосконалив свої вміння в мові програмування С++, вивчив новий для себе спосіб розробки програм. Також я вдосконалив свої знання з дискретної математики, по причині, що «жадібний алгоритм» широко використовується в ній. Вдосконалив навички в використанні інтегрованого середовища розробки програмного забезпечення - Microsoft Visual Studio 2019 .

При виконанні завдання труднощів та питань виникало дуже мало, відповіді на ці питання можна знайти в мережі інтернет.

Набуті навички при виконанні завдання будуть корисні при подальшому вивченні інформаційних технологій та сприятимуть практичному вдосконаленню набутих вмінь.

У результатах дослідження літератури за питаннями оптимізації задач, ми відхилили припущення про використання «ясних алгоритмів», які нам дають наступні можливості:

* Скоротити час прийняття рішень
* Перевіряти всі можливі варіанти рішень на даному моменті часу

## **СПИ****СОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

1. Тім Рафгарден. Досконалий алгоритм. Жадібні алгоритми і динамічне програмування: Підручник. - X.: Компанія С-МІТ, 2020. - 256с.
2. Кормен Т і ін. Алгоритми: побудова й аналіз. - М .: МЦНМО, 2000.
3. Алфьорова З.В. Теорія алгоритмів. - М .: Статистика, 1973
4. Жадібний алгоритм [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/company/piter/blog/489014/.
5. Жадібний алгоритм [Електронний ресурс] – Режим доступу до ресурсу: <https://uk.wikipedia.org/wiki/%D0%96%D0%B0%D0%B4%D1%96%D0%B1%D0%BD%D0%B8%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC>.
6. Жадібний алгоритм [Електронний ресурс] – Режим доступу до ресурсу: https://dl.sumdu.edu.ua/textbooks/95351/522264/index.html#p5.
7. Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. Глава 16. Жадібні алгоритми Жадібний алгоритм [Електронний ресурс] – Режим доступу до ресурсу: https://dl.sumdu.edu.ua/textbooks/95351/522264/index.html#p5.