Федеральное государственное бюджетное образовательное учреждение высшего образования  
«Сибирский государственный университет   
телекоммуникаций и информатики»

**Расчетно-графическое задание**

по курсу программирования

**«Алгоритм решения задачи о рюкзаке»**

Выполнилa: Щёголева В.А.

Группа: ИА\_331

Вариант: 31

Проверил: Лебеденко Л.Ф.

Новосибирск 2024

Задание

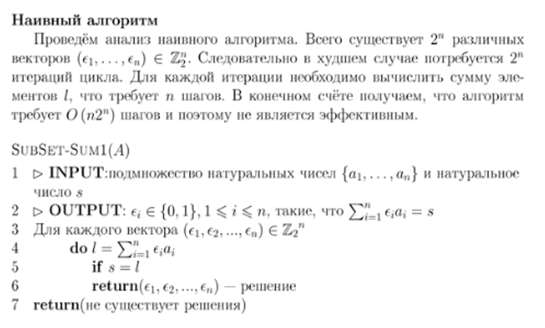
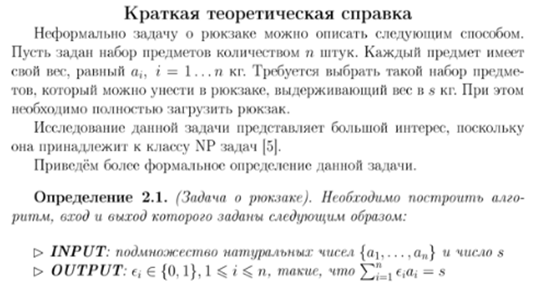
Разработать программу реализующую алгоритм рюкзака. Предусмотреть получение исходных данных через аргументы командной строки.

Критерии оценки

Оценка**«хорошо»**: реализован алгоритм наивного рюкзака. Определить производительность алгоритма.

Оценка**«отлично»**: реализован алгоритм наивного рюкзака и включен в динамическую библиотеку. Определить производительность алгоритма.

Указания к выполнению задания



Анализ задачи

Алгоритм решения задачи о рюкзаке

Способ **полного перебора**.

Обозначим максимальный вес ранца, как W, а количество различных вещей N. При этом в наборе могут быть абсолютно одинаковые предметы.

Каждый предмет имеет вес и стоимость (ценность):

|  |  |  |
| --- | --- | --- |
| Название | Вес | Стоимость |
| Книга | 1 | 600 |
| Бинокль | 2 | 5000 |
| Аптечка | 4 | 1500 |
| Ноутбук | 2 | 40000 |
| Котелок | 1 | 500 |

Чтобы решить задачу, необходимо составить все комбинации наборов предметов и выбрать тот набор, масса которого не более W, а общая стоимость (по отношению к другим подходящим наборам) максимальна.

Это означает, что нужно рассмотреть все *перестановки из N*. Общее количество перестановок находится по формуле: N!

Если предметов пять, то придётся рассмотреть 5! = 120 различных наборов предметов.

Вычислительная сложность алгоритма полного перебора для решения **задачи о рюкзаке** равна O(N!).

Минусом способа полного перебора всех предметов, является то, что если количество вещей велико, то алгоритм не даст решения за приемлемое время, поскольку факториал является чрезвычайно быстро растущей функцией.

Метод динамического программирования Сложность — O(NW)

Пусть A(k,s) есть максимальная стоимость предметов, которые можно уложить в рюкзак вместимости s, если можно использовать только первые k предметов, то есть {n1,n2,…,nk}, назовем этот набор допустимых предметов для A(k,s).

A(k,0)=0

A(0,s)=0

Найдем A(k,s). Возможны 2 варианта:

Если предмет k не попал в рюкзак. Тогда A(k,s) равно максимальной стоимости рюкзака с такой же вместимостью и набором допустимых предметов {n1,n2,…,nk−1}, то есть A(k,s)=A(k−1,s)

Если k попал в рюкзак. Тогда A(k,s) равно максимальной стоимости рюкзака, где вес s уменьшаем на вес k-ого предмета и набор допустимых предметов {n1,n2,…,nk−1}плюс стоимость k, то есть A(k−1,s−wk)+pk

A(k,s)={A(k−1,s),A(k−1,s−wk)+pk,bk=0bk=1

То есть: A(k,s)=max(A(k−1,s),A(k−1,s−wk)+pk)

Стоимость искомого набора равна A(N,W), так как нужно найти максимальную стоимость рюкзака, где все предметы допустимы и вместимость рюкзака W.

Восстановим набор предметов, входящих в рюкзак

Будем определять, входит ли ni предмет в искомый набор. Начинаем с элемента A(i,w), где i=N, w=W. Для этого сравниваем A(i,w) со следующими значениями:

Максимальная стоимость рюкзака с такой же вместимостью и набором допустимых предметов {n1,n2,…,ni−1}, то есть A(i−1,w)

Максимальная стоимость рюкзака с вместимостью на wi меньше и набором допустимых предметов {n1,n2,…,ni−1}плюс стоимость pi, то есть A(i−1,w−wi)+pi

Заметим, что при построении A мы выбирали максимум из этих значений и записывали в A(i,w). Тогда будем сравнивать A(i,w)c A(i−1,w), если равны, тогда ni не входит в искомый набор, иначе входит.

Метод динамического программирование всё равно не позволяет решать задачу за полиномиальное время, потому что его сложность зависит от максимального веса.

Функция, реализующая алгоритм наивного рюкзака

int knapsack(int weights[], int values[], int n, int capacity) {

if (n == 0 || capacity == 0) {

return 0;

}

if (weights[n-1] > capacity) {

return knapsack(weights, values, n-1, capacity);

} else {

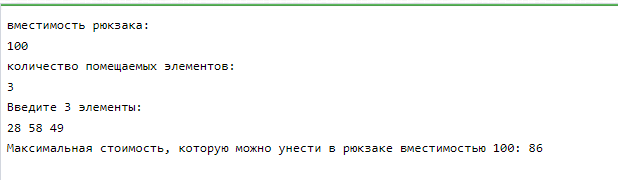
return max(values[n-1] + knapsack(weights, values, n-1, capacity-weights[n-1]),

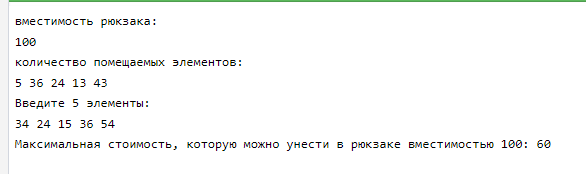
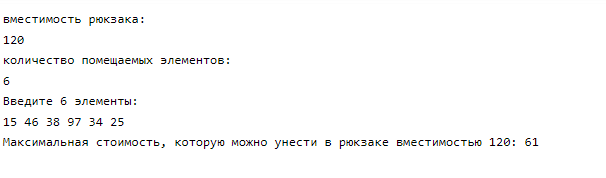
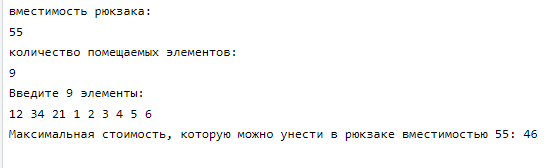
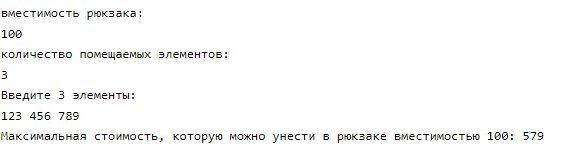
knapsack(weights, values, n-1, capacity));

}

}

Тестовые данные



1. 
2. 
3. 
4. 

Листинг программы.

Для того, чтобы создать многофайловый проект, я разбила код на 3 файла и обьединила их с помощью CMakeLists, так же в нем подключила динамическую библиотеку.

//func.c – файл, содержащий функцию наивного рюкзака

#include "func.h"

int max(int a, int b) {

    return (a > b) ? a : b;

}

int knapsack(int weights[], int values[], int n, int capacity) {

if (n == 0 || capacity == 0) {

return 0;

}

if (weights[n-1] > capacity) {

return knapsack(weights, values, n-1, capacity);

} else {

return max(values[n-1] + knapsack(weights, values, n-1, capacity-weights[n-1]),

knapsack(weights, values, n-1, capacity));

}

}

//func.h – заголовочный файл

#include <stdio.h>

#include <stdlib.h>

#ifndef \_\_FUNC\_H\_\_

#define \_\_FUNC\_H\_\_

int max(int a, int b);

int knapsack(int capacity, int values[], int weights[], int n);

#endif

//main.c – файл, реализующий проект

#include "func.h"

int main() {

int capacity;

printf("вместимость рюкзака:");

scanf("%d", &capacity);

int a;

printf("количество помещаемых элементов: ");

scanf("%d", &a);

int \*values = (int\*)malloc(a \* sizeof(int));

if (values == NULL) {

printf("Ошибка выделения памяти\n");

return 1;

}

printf("Введите %d элементы:\n", a);

for (int i = 0; i < a; i++) {

scanf("%d", &values[i]);

}

int weights[] = {10, 20, 30};

int n = sizeof(values) / sizeof(values[0]);

int max\_value = knapsack( weights, values, n, capacity);

printf("Максимальная стоимость, которую можно унести в рюкзаке вместимостью %d: %d\n", capacity, max\_value);

return 0;

}

//CMakeLists.txt

cmake\_minimum\_required(VERSION 3.10) # Проверка версии CMake.

project(cmake) # Название проекта

set(SOURCE\_EXE main.c) # Установка переменной со списком

set(SOURCE\_LIB func.c) # Тоже самое, но для библиотеки

add\_library(MY\_LIB SHARED ${SOURCE\_LIB})# Создание динамической библиотеки с именем MY\_LIB

add\_executable(main ${SOURCE\_EXE}) # Создает исполняемый файл с

именем main

target\_link\_libraries(main MY\_LIB) # Линковка программы с библиотекой

Заключение

В работе был реализован алгоритм наивного рюкзака. Данный алгоритм является одним из наиболее простых и понятных алгоритмов решения задачи о рюкзаке. Он перебирает все возможные подмножества предметов, сравнивая их стоимость и вес с ограничениями рюкзака, и выбирает оптимальное решение.

**Результаты:**

* Программа успешно реализует алгоритм наивного рюкзака, демонстрируя его работу на тестовых данных.
* Алгоритм наивного рюкзака показал свою эффективность для небольших задач, где количество предметов ограничено.
* Для больших задач, содержащих множество предметов, алгоритм наивного рюкзака становится неэффективным, так как его временная сложность экспоненциально возрастает с увеличением количества предметов. Алгоритм наивного рюкзака не подходит для решения задач о рюкзаке с большим количеством предметов. Для решения задач о рюкзаке с большим количеством предметов необходимо использовать более эффективные алгоритмы, такие как жадный алгоритм или динамическое программирование.