Московский авиационный институт
(национальный исследовательский университет)

**Факультет информационных технологий и прикладной математики**

**Кафедра вычислительной математики и программирования**

**Лабораторная работа №7 по курсу «Дискретный анализ»**


| | |
|---|---|
| Студент: | К. А. Калугин |
| Преподаватель: | А. А. Кухтичев |
| Группа: | М8О-307Б |
| Дата: | |
| Оценка: | |
| Подпись: | |


**Москва, 2021**

# Лабораторная работа №7

**Задача:** При помощи метода динамического программирования разработать алгоритм решения задачи, определяемой своим вариантом; оценить время выполнения алгоритма и объем затрачиваемой оперативной памяти. Перед выполнением задания необходимо обосновать применимость метода динамического программирования.

Разработать программу на языке C или C++, реализующую построенный алгоритм. Формат входных и выходных данных описан в варианте задания:

У вас есть рюкзак, вместимостью m, а так же n предметов, у каждого из которых есть вес wi и стоимость ci. Необходимо выбрать такое подмножество I из них, чтобы: $\sum_{i \in I} w_i \leqslant m$. $(\sum_{i \in I} c_i) * |I|$ является максимальной из всех возможных. |I| – мощность множества I.

# 1 Описание

Основной идеей ДП является разбиение сложной задачи на несколько простых. Поэтому мы разделяем задачу по сборке рюкзака весом в X кг на задачи оптимальной сборки рюкзаков весом 1-X кг. После этого, зная как оптимально собрать, скажем, рюкзак на 2 и 4 кг, мы можем оптимально собрать рюкзак на 6 кг, объединив их.

Задача о рюкзаке является известной NP-полной задачей, которая при некоторых ограничениях решается за полиномиальное время с помощью метода динамического программирования.

Стадартный вариант задачи описан и доказан в [2]. Для моего варианта задания $dp_{i,j,k}$ — максимальная стоимость j вещей из первых i, таких, что их суммарный вес не превышает k. То есть алгоритм будет перебирать количество предметов, которые будут в рюкзаке.

Пусть существует оптимальное решение в $dp_{i,j,kwj1}$, тогда $dp_{i+1,j+1,k} = max(dp_{i,j,kwj1+cj+1}$, $dp_{i+1,j,k})$. В рекуррентной формуле рассматривается два варианта: взять вещь j + 1 или нет. Такое решение имеет $n^2m$ состояния, в каждое можно перейти из двух других. Так временная сложность алгоритма O($n^2m$). Хранение всей таблицы состояний слишком дорого по памяти, но необходимо для восстановления ответа. Поэтому будем хранить только $dp_i$ и $dp_{i+1}$ и битовые множества предметов, которые оптимальны для решения подзадачи. Пространственная сложность такого подхода O($nm$).

# 2 Исходный код

Попробуем уложить рюкзаки весом 1-X кг оптимальным образом - так, чтобы не оставалось свободного места и чтобы цена была максимальна. После этого проверим - можно ли было добится лучшей укладки, если класть по 2, 3, 4... предмета в один рюкзак за раз. В конце проверки для очередного количества предметов результаты сохраняются и используются для следующих раскладок.

```
#include <bitset>
#include <iostream>
#include <vector>

using namespace std;
const int MAX_N = 100;//

int main () {
    int n, m;// ,
    long long bestSum = 0;//
    bitset <MAX_N> bestSol;//
    cin >> n >> m;
    vector <vector <long long>> prevSum (n + 1, vector <long long> (m + 1));//     ()
    vector <vector <bitset <MAX_N>>> prevSol (n + 1, vector <bitset <MAX_N> > (m + 1));
        //    ()
    vector <vector <long long>> currSum (n + 1, vector <long long> (m + 1));//
    vector <vector <bitset <MAX_N>>> currSol (n + 1, vector <bitset <MAX_N> > (m + 1));
        //
    vector <int> weight (n);//
    vector <long long> price (n);//

    for (int i = 0; i < n; i ++) {//
        cin >> weight [i] >> price [i];
    }

    for (int j = 1; j < n + 1; j ++) {//for
        for (int k = 1; k < m + 1; k ++) {//for
            prevSum [j][k] = prevSum [j - 1][k];//
            prevSol [j][k] = prevSol [j - 1][k];//
            if ((price [j - 1] > prevSum [j][k]) && (k - weight [j - 1] == 0)) {//

                prevSum [j][k] = price [j - 1];//
                prevSol [j][k] = 0; //
                prevSol [j][k][j - 1] = 1; //
            }
            if (prevSum [j][k] > bestSum) {//
                bestSum = prevSum [j][k];
                bestSol = prevSol [j][k];
            }
        }
    }
```

```

    for (long long i = 2; i < n + 1; i ++) {//for
        for (int j = 1; j < n + 1; j ++) {//for
            for (int k = 1; k < m + 1; k ++) {//for
                currSum [j][k] = currSum [j - 1][k];//
                currSol [j][k] = currSol [j - 1][k];//
                if ((k - weight [j - 1] > 0) && (prevSum [j - 1][k - weight [j - 1]] > 0)) {//          (   )
                    if (i * (price [j - 1] + prevSum [j - 1][k - weight [j - 1]] / (i - 1)) > currSum [j][k]) {//
                        currSum [j][k] = i * (price [j - 1] + prevSum [j - 1][k - weight [j - 1]] / (i - 1));//
                        currSol [j][k] = prevSol [j - 1][k - weight [j - 1]];//
                        currSol [j][k][j - 1] = 1; //
                    }
                }
                if (currSum [j][k] > bestSum) {//
                    bestSum = currSum [j][k];
                    bestSol = currSol [j][k];
                }
            }
        }
        swap (currSum, prevSum);//
        swap (currSol, prevSol);//
    }

    cout << bestSum << '\n';//
    for (int i = 0; i < n; i ++) {//
        if (bestSol [i]) {
            cout << i + 1 << ' ';
        }
    }
    cout << '\n';

    return 0;
}
```

# 3 Консоль

```
PS C:\VSC\DA>.\a.exe
3 6
2 1
5 4
4 2
6
1 3
```

# 4 Тест производительности

Сравним реализованный алгоритм с приближённым алгоритмом, который не всегда даёт верный ответ. Тесты состоят из 10, 50 и 100 вещей.

```
PS C:\VSC\DA>$ g++ Lab7.2 -g -O2 -pedantic -std=c++17 -Wall -Wextra -Werror
main.cpp -o solution
PS C:\VSC\DA>$ g++ bench -g -O2 -pedantic -std=c++17 -Wall -Wextra -Werror
benchmark.cpp -o benchmark
PS C:\VSC\DA>$ ./benchmark <tests/1.in
Sort 0.74 ms
PS C:\VSC\DA>$ ./solution <tests/1.in
DP 0.388 ms
PS C:\VSC\DA>$ ./benchmark <tests/2.in
Sort 0.119 ms
PS C:\VSC\DA>$ ./solution <tests/2.in
DP 1.584 ms
PS C:\VSC\DA>$ ./benchmark <tests/3.in
Sort 0.204 ms
PS C:\VSC\DA>$ ./solution <tests/3.in
DP 125.560 ms
```

Видно, что приближённый алгоритм гораздо быстрее динамического программирования, потому что он сортирует предметы по уменьшению веса и возрастанию цены, а количество предметов мало. Однако, как уже было сказано - он не всегда дает правильные ответы.

# 5 Выводы

Выполнив седьмую лабораторную работу по курсу «Дискретный анализ», я научился следующему:

Во-первых, бесконечные попытки дебага ни к чему не приведут при наличии фундаментальной ошибки в алгоритме. Таким образом, иногда лучше проанализировать весь алгоритм и, возможно, переделать его целиком, чем вечно устранять возникающие ошибки. Во-вторых, я научился правильно оценивать объемы затрачиваемой программой памяти. Так, например, оказалось, что трехмерный массив размера m*n*n, занимает не так уж и много места, если m <= 5000, а n <= 100. Причем массив является bitset‘ом, то есть на хранение одной ячейки тратится лишь один бит памяти.

# Список литературы

[1] *Задача о рюкзаке (Knapsack problem) простыми словами.*
URL: https://habr.com/ru/post/561120/ (дата обращения: 22.11.2021).

[2] *Задача о рюкзаке.*
URL: https://neerc.ifmo.ru/wiki/index.php?title=Задача_о_рюкзаке (дата обращения: 22.11.2021).