Московский авиационный институт
(национальный исследовательский университет)

Институт информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №7 по курсу «Дискретный анализ»

Студент: Е. А. Кондратьев
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-19
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 данных: В первой строке заданы $1 \leq n \leq 1001 \leq m \leq 5000$. В последующих n строках через пробел заданы параметры предеметов: w_i и c_i.

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

1 Описание

Динамическое программирование - это метод при котором решение сложной задачи составляется из решений простых задач.

Пусть d(i,c) - максимальная стоимость любого количества вещей типов от 1 до i, суммарным весом до c включительно.

Тогда меняя i от 1 до N, рассчитаем на каждом шаге d(i,c), для c от 0 до W, по рекуррентной формуле:

d(i,c)=

{d(i - 1, c)max(d(i - 1, c), for c = 0, . . . , wi - 1;

{d(i, c - wi) + pi), for c = wi,. . . ,W;

После выполнения в d(N,W) будет лежать максимальная стоимость предметов, помещающихся в рюкзак.

Если не нужно восстанавливать ответ, то можно использовать одномерный массив d(c) вместо двумерного и использовать формулу:

d(c) = max(d(c), d(c - wi) + pi);

Сложность алгоритма O(NW).

2 Исходный код

Опишем матрицы DPl и DPc для $dp_i + 1$ и dp_i, матрицы cur и prev для хранения множества предметов.

Код: main.cpp

```cpp
#include <bits/stdc++.h>

using namespace std;
#define fastIO \
    ios_base::sync_with_stdio(0); \
    cin.tie(0); \
    cout.tie(0)
#define fori(k, n) for (int i(k); i < (n); ++i)
#define forj(h, m) for (int j(h); j < (m); ++j)
#define foru(l, p) for (int u(l); u < (p); ++u)
typedef unsigned long long ull;
typedef long double ld;
typedef long long ll;

typedef vector<long long> vll;
typedef vector<int> vi;
typedef pair<ll, ll> pl;

#define MAX_N 100
//ull MOD = 1000000007LL;
//vector<int> G[MAX_N];
//map<ull, ull> m;

void solve() {
    int n, m, ans(0);
    cin >> n >> m;

    // vector<long long>
    vi w(n);
    vi c(n);

    for (int i = 0; i < n; ++i) cin >> w[i] >> c[i];

    bitset<MAX_N> res;
    vector<vi> DPl(n + 1, vi(m + 1));
    vector<vi> DPc(n + 1, vi(m + 1));
    vector<vector<bitset<MAX_N>>> prev(n + 1, vector<bitset<MAX_N>>(m + 1));
    vector<vector<bitset<MAX_N>>> cur(n + 1, vector<bitset<MAX_N>>(m + 1));


    fori(1, n + 1) {
        forj(1, m + 1) {
            DPl[i][j] = DPl[i - 1][j];
            prev[i][j] = prev[i - 1][j];

            if(c[i - 1] > DPl[i][j] and j - w[i - 1] == 0) {
                DPl[i][j] = c[i - 1];
                prev[i][j] = 0;
                prev[i][j][i - 1] = true;
            }
            if(DPl[i][j] > ans) {
                ans = DPl[i][j];
                res = prev[i][j];
            }
        }
```

```
56      }
57      fori(2, n + 1) {
58          forj(1, n + 1) {
59              foru(1, m + 1) {
60                  DPc[j][u] = DPc[j - 1][u];
61                  cur[j][u] = cur[j - 1][u];
62                  if(u - w[j - 1] > 0 and DPl[j - 1][u - w[j - 1]] > 0) {
63                      if((c[j - 1] + DPl[j - 1][u - w[j - 1]] / (i - 1)) * i > DPc[j][u])
                            {
64                          DPc[j][u] = (c[j - 1] + DPl[j - 1][u - w[j - 1]] / (i - 1)) * i;
65                          cur[j][u] = prev[j - 1][u - w[j - 1]];
66                          cur[j][u][j - 1] = true;
67                      }
68                  }
69                  if(DPc[j][u] > ans) {
70                      ans = DPc[j][u];
71                      res = cur[j][u];
72                  }
73              }
74          }
75          swap(cur, prev);
76          swap(DPc, DPl);
77      }
78      cout << ans << '\n';
79      fori(0, n)
80          if(res[i])
81              cout << i + 1 << ' ';
82  }
83
84  signed main() {
85  #ifdef _ONPC_
86      freopen("input.txt", "r", stdin);
87      freopen("output.txt", "w", stdout);
88  #define TIMEIT
89      timeit
90  #endif // _ONPC_
91
92      fastIO;
93      int t(1);
94      //cin >> t; // UNCOMMENT IF WITH TESTS //
95      fori(0, t) solve();
96      cout << '\n';
97  #ifdef TIMEIT
98      endtimeit
99  #endif // TIMEIT
100     return 0;
101 }
```

3 Консоль

```
C:/Users/egork/Desktop/space/cmake-build-debug/space.exe
3 6
2 1
5 4
4 2
6
1 3
C:/Users/egork/Desktop/space/cmake-build-debug/space.exe
10 43
2 7
43 88
34 92
25 24
4 29
9 2
1 2
11 28
3 22
44 2
600
1 3 5 9
```

4 Тест производительности

Сравним реализованный алгоритм с приближённым алгоритмом.
Моя реализация:

```
C:/Users/egork/Desktop/space/cmake-build-debug/space.exe
Sort 0.74 ms

C:/Users/egork/Desktop/space/cmake-build-debug/space.exe
Sort 0.119 ms

C:/Users/egork/Desktop/space/cmake-build-debug/space.exe
Sort 0.204 ms

C:/Users/egork/Desktop/space/cmake-build-debug/space.exe
DP 0.388 ms

C:/Users/egork/Desktop/space/cmake-build-debug/space.exe
DP 1.584 ms

C:/Users/egork/Desktop/space/cmake-build-debug/space.exe
DP 125.560 ms
```

Приближённый алгоритм гораздо быстрее динамического программирования, потому что он сортирует предметы по уменьшению веса и возрастанию цены, а также количество предметов маленькое.

5 Выводы

В ходе выполнения лабораторной работы я изучил классические задачи динамического программирования и их методы решения, реализовал алгоритм для своего варианта задания.

Динамическое программирование позволяет разработать точные и относительно быстрые алгоритмы для решения сложных задач, в то время, как переборное решение слишком медленное, а жадный алгоритм не всегда даёт правильный результат.

Узнал что a / b * c и (a / b) * c - это ловушка

Список литературы

[1] *Поисковик - Google.*
URL: https://www.google.com/

[2] *Сайт с подробной документацией библиотек C++*
URL: https://en.cppreference.com/

[3] *Задача о рюкзаке*
URL: https://neerc.ifmo.ru/wiki/index.php?title=Задача_