

算法第三次编程作业

0-1背包问题

陈童

2019211283

2019211304班

1. 题目描述

有 N 件物品和一个容量为 V 的背包。第 i 件物品的大小是 c_i ，价值是 w_i 。求解将哪些物品装入背包可使这些物品的费用总和不超过背包容量，且价值总和最大。

输入文件名为 bag.in。输入共两行。

第一行有 2 个整数 V 和 N ，用一个空格隔开， V 代表背包容量， N 代表物品数目。接下来的 N 行每行包括两个整数，分别表示某件物品的大小和其价值。

输出文件名为 bag.out。输出共一行。

第一行包含 1 个整数表示最大价值。

2. 算法实现

2.1 基础版背包问题算法实现

2.1.1 算法思路:

假设 $A[i][j]$ 中每一项都代表一个 value 的总和。 i 代表选取的物品范围为 $[i:n]$ ， j 代表当前的容量。 $A[i][j]$ 的含义是 $[i:n]$ 项物品中当前容量为 j 时的最大价值。自底向上地计算出每一项 $A[i][j]$ 的值，而寻求的最大值就在于 $A[0][V]$ 中。

2.1.2 关键函数代码表示

```
for (int itm = items.size() - 2; itm >= 0; itm--) {
    for (int cap = j; cap >= 0; cap--) {
        if (items[itm].weight > cap) {
            a[itm][cap] = a[itm + 1][cap];
        } else {
            a[itm][cap] = max(a[itm + 1][cap],
                              a[itm + 1][cap - items[itm].weight] + items[itm].val);
        }
    }
    // cout << a[itm][cap] << " ";
}
```

```

}
// cout << endl;
}
return 0;

```

2.1.3 测试结果

```

3_bag > ≡ bag.in
1 10 5
2 2 6
3 2 3
4 6 5
5 5 4
6 4 6

```

15

> g++ bag.cpp -o run -std=c++11 && ./run

14

~/I /Mo /com/apple/Doc/C/Algorithm/Exp3_bag

```

1000 100
19 12
53 61
61 63
74 78
98 49
70 46
15 44
59 36
64 37
29 66
98 43
79 16
74 14
85 73
52 72
70 72
84 83
91 15
84 83
75 65
78 21
72 49
5 36
46 20

```

15

> g++ bag.cpp -o run -std=c++11 && ./run

14

> g++ bag.cpp -o run -std=c++11 && ./run

2617

2.2 改进算法

2.2.1 算法思路

算法2.1中对所有 $A[i][j]$ 求值是过于浪费的，很多 (i, j) 处于不可能到达的状态。于是通过记录激增点的方式代替记录value值的和的方式可以更高效率的实现。对于不同的记录方式，我们选择把所有激增点按照占用空间排序，然后剔除所有不合格的点。到最后剩下的就是合格的点。

2.2.2 关键算法代码

```
void clear_point(list<node> &set) {
    list<node>::iterator it = set.begin();
    list<node>::iterator it_ = it;
    it_++;
    while (it_ != set.end()) {
        if (it->weight <= it_>weight && it->val > it_>val) {
            it_ = set.erase(it_);
        } else
        {
            it++;
            it_++;
        }
    }
}

void cal_weight(int j, vector<node> &items) {
    node temp_node(0, 0);
    a[items.size()].push_back(temp_node);
    list<node> temp_q, temp;
    auto it = items.rbegin();
```

```
temp_q.push_back(temp_node + *it);
```

```
for (int i = items.size() - 1; i >= 0; i--) {
```

```
    temp = a[i + 1];
```

```
    temp_q.merge(temp);
```

```
clear_point(temp_q);
```

```
a[i] = temp_q;
```

```
temp_q.clear();
```

```
it++;
```

```
for (const auto &it_ : a[i]) {
```

```
    temp_node = (it_ + *it);
```

```
    if (temp_node.weight <= j)
```

```
        temp_q.push_back(temp_node);
```

```
};
```

```
}
```

```
}
```

2.3 测试样例

The image displays two side-by-side screenshots of a C++ IDE, likely Visual Studio Code, showing test cases and their outputs for a bag problem. The left screenshot shows a list of test cases with their corresponding inputs and outputs. The right screenshot shows the same test cases with their corresponding outputs, including the execution time and memory usage.

Test Case	Input	Output
1000 100	19 12 53 61 61 63 74 78 98 49 70 46 15 44 59 36 64 37 29 66 98 43 79 16 74 14 85 73 52 72 70 72 84 83 91 15 84 83 75 65 78 21 72 49 5 36 46 20	58 3 99 79 96 97 64 24 28 10 29 55 43 95 35 41 2 67 30 28 86 14 56 11 27 86 9 50 85 63 13 69 65 39 46 62 51 10 31 49 99 22 56 88 97 60 11 7

The IDE interface shows the following details:

- Left Screenshot:** The output window shows the execution of the program for each test case. The output for each test case is a single line containing two numbers, representing the maximum value and the number of items in the bag.
- Right Screenshot:** The output window shows the execution of the program for each test case. The output for each test case is a single line containing two numbers, representing the maximum value and the number of items in the bag. Additionally, the execution time and memory usage are displayed for each test case.

