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 ci，价值是 wi。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:

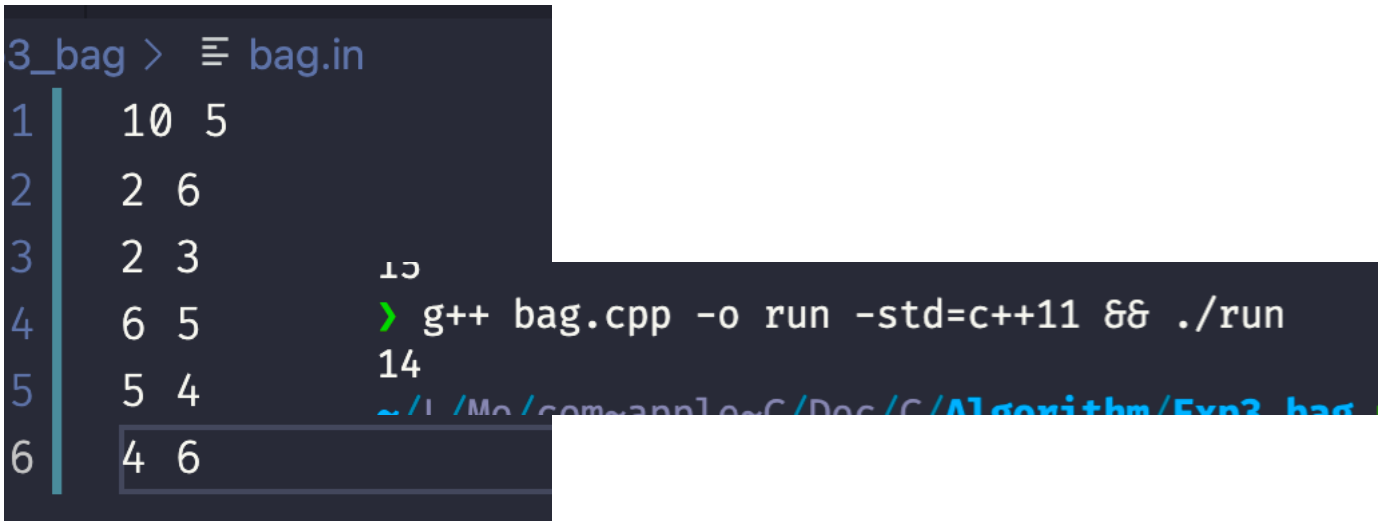
假设A[i][j]中每一项都代表一个value的总和.i代表选取的物品范围为[i:n], j代表当前的容量. A[i][j]的含义是[i:n]项物品中当前容量为j时的最大价值. 自底向下地计算出每一项A[i][j]的值, 而寻求的最大值就在于A[0][V]中.

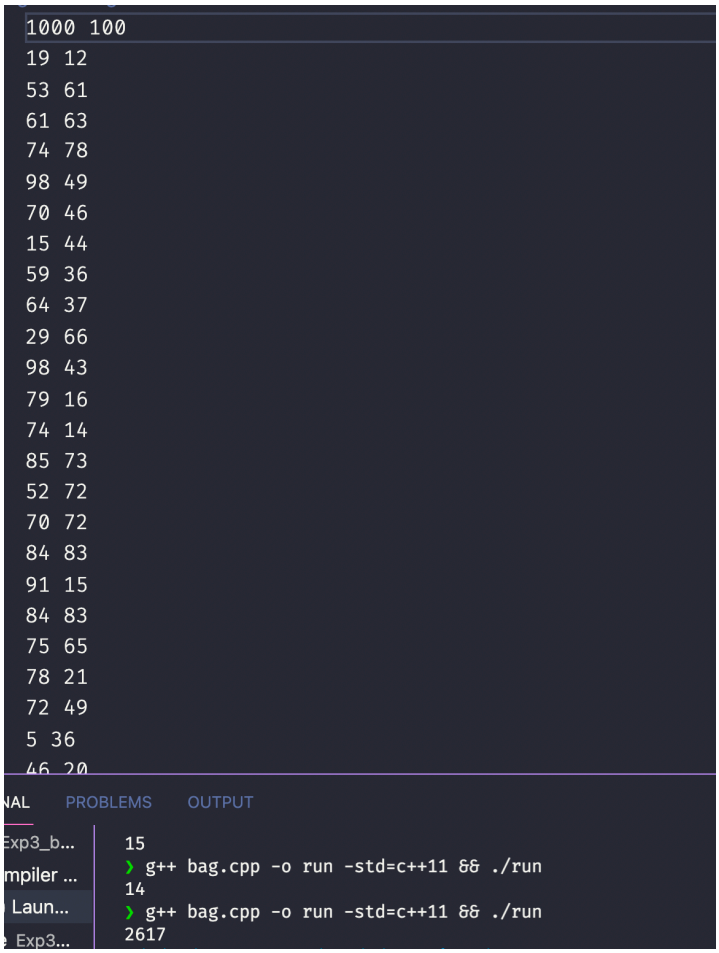
2.1.2 关键函数代码表示

```
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





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