# 《数据结构与算法》实验报告

年级、专业、班级		2021 级计算机学院计算机科学与技术 卓越 2 班			姓名	文红兵	
实验题目		动态规划算法实践					
实验时间	2022.11.24		实验地点	竹园四栋			
实验成绩			实验性质	□验证性	E性 √设计性 □综合性		
教师评价:							
□算法/实验过程正确; □源程序/实验内容提交 □程序结构/实验步骤合理;							
□实验结果正确;    □语法、语义正确;    □报告规范;							
其他:							
评价教师签名:							
实验目的 1. 掌握动态规划算法的基本原理 2. 训练使用动态规划算法设计技术,通过编程解决不同难度问题的实践能力							
二、实验项目内容							
<u>实验课题 1:</u> 题目内容:							
7-1 真实的背包故事							
解题思路:  dp[i][j]:代表前i件物品背包容量为j的最大价值  状态转换方程: dp[i][j]= min{dp[i-1][j],dp[i-1][j-w[i]]+v[i]}  源代码:							
<b>最小</b> 数量,通过标记数组同瓣搜索							

```
#include<bits/stdc++.h>
2
    using namespace std;
3
 4
    int main(){
5
        // 输入
 6
         int N,W;
 7
         cin>>N>>W;
 8
         int v[N],w[N];
        for(int i=0;i<N;++i) cin>>v[i]>>w[i];
10
         // 滚动数组优化 dp[i][j]:代表前i件物品背包容量为j的最大价值
11
12
         int dp[W+1];
         int s[N][W+1]; // 回溯搜索个数
13
14
15
         // 初始化
16
         memset(s,0,sizeof(s));
17
         dp[0] = 0;
18
         for(int i=1;i<=W;++i) dp[i] = INT MIN;</pre>
         //填表
 20
         for(int i=0;i<N;++i){</pre>
 21
 22
              for(int j=W;j>=w[i];--j){
                 int temp = dp[j-w[i]]+v[i];
 23
 24
                 if(dp[j]<=temp){</pre>
 25
                      dp[j] = temp;
                     s[i][j] = 1; // 记录当前
 26
 27
 28
             }
 29
           // 回溯搜索路径
   31
           vector<int> path;
           int i=N-1;
           int j=W;
           while(i>0 && j>0){
              if(s[i][j] == 1){
                path.push_back(i);
                 j = j-w[i];
   39
   40
   41
   42
   43
           cout<<dp[W]<<" "<<path.size()<<endl;</pre>
   11
           system("pause");
   45
           return 0;
   46
```

#### 时间与空间复杂度分析:

时间复杂度是 O(n^2)

空间复杂度是 O(n^2)

## 实验课题 2:

#### 题目内容:

7-2 动态规划

#### 解题思路:

string dp[maxN][4]: dp[i][j]表示下标 i 前的字符串分割成 j 段的反转后的最小字符串

状态转移方程: dp[i][j] = min(dp[i][j],dp[k][j-1]+temp); // k

 $\in$  [0,i)

#### 源代码:

```
1 #include<bits/stdc++.h>
  using namespace std;
4 #define maxN 50
6 string dp[maxN][4]; //dp[i][j]表示下标i前的字符串分割成j段的反转后的最小字符串
8 int main(){
      string s:
10
      cin>>s:
      for(int i=0;i<s.size();++i) { // 初始化
11
         dp[i][1] = s.substr(0,i+1);
12
         reverse(dp[i][1].begin(),dp[i][1].end());
13
14
15
      for(int j=2;j<=3;++j){
          for(int i=j-1;i<s.size();++i){</pre>
16
                                          // 设置成为最大
             dp[i][j] = STRINGMAX;
17
             for(int k=0;k<i;++k){
                                          // 找出dp[i][j]的最小字符串
18
                string temp = s.substr(k+1,i-k);
19
20
                reverse(temp.begin(),temp.end());
21
                dp[i][j] = min(dp[i][j],dp[k][j-1]+temp);
22
             cout<<endl:
23
24
25
       cout<<dp[s.size()-1][3]<<endl; // 输出结果
26
27
      system("pause");
28
      return 0;
29 }
```

### 时间与空间复杂度分析:

时间复杂度为 O(n^3)

空间复杂度为 O(n)

#### (3) 思考题

0-1 背包问题, 当商品的重量和价值满足以下哪些条件, 就可用 比动态规划效率更好的算法解决?描述解题思路并分析时空复杂度。如果 没有更好的算法,说明理由(举反例)。

(1) 每件商品的价值相同

贪心算法:对重量排序,每次选择重量最小的放进背包,直到背包 放不下,时间复杂度: 0(nlogn) 空间复杂度: 0(1)

(2) 每件商品的重量相同

类似 1: 贪心算法: 对价值排序,每次选择价值最大的放进背包,直 到背包放不下,时间复杂度: 0(nlogn) 空间复杂度: 0(1)

(3) 每件商品的性价比相同 (性价比 = 价值/重量) 没有更好的算法解决 如背包容量:5

商品: (1,1), (1,1), (1,1), (1,1), (5,5)

(4) 重量越大, 性价比越低 没有更好的算法解决

如背包容量:5

商品: (1,10), (2,18), (3,24), (4,28), (5,30)

(5) 重量越大,价值越小

贪心算法:对重量排序,每次选择重量最小的,直到装不下

时间复杂度: 0(nlogn) 空间复杂度: 0(1)