***西安财经大学****信息学院*

**姓名** 王南蔺

**学号** 1831050318

**班级** 网络1802

**年级** 18级

**指导教师** 李翠

《算法设计与分析》 实验报告

**实验名称** 动态规划算法设计与实现 **实验室** 家里 **实验日期** 2020.6.7

|  |
| --- |
| 动态规划算法设计与实现   1. 实验目的与要求   1. 理解动态规划算法思想。  2. 实践动态规划算法设计过程与方法，代码实现框架与程序编写。  3. 按实验报告要求应用动态规划算法求解实际应用问题。   1. 实验内容   1. 0/1背包问题的动态规划算法  (1)问题描述  将N件物品选几件放入容量为W(重量)的背包中，每件物品有自己的重量w和价值v，要求在不超过背包容量情况下使背包内放入的物品总价值最高。  将问题描述得通俗易懂一些，有一个贼在偷窃一家商店时，发现有n件物品，第i件物品价值元，重磅，此处与都是整数。他希望带走的东西越值钱越好，但他的背包中至多只能装下W磅的东西，W为一整数。应该带走哪几样东西？这个问题之所以称为0-1背包，是因为每件物品或被带走或被留下，小偷不能只带走某个物品的一部分或带走同一物品两次  (2)算法设计思想及递推方程建立  算法设计思想：  0-1背包问题可以使用动态规划算法来解决，动态规划类似于分治算法，都是将一个大问题分解为一个个小问题，不同之处在于，动态规划记忆了重复的子问题，避免了运算过程中的重复计算。所以说动态规划适用于有重叠子问题和最优子结构性质（动态规划每一步求的都是最优解）的问题。  递推方程建立：  1.首先，物品的总重量是不超过背包容量W的，即  2.我们可以提取出一个子问题：将前i件物品放入容量为j的背包中，如果只考虑第i件物品放或者不放，那么问题就可以转化为只涉及前i-1的问题。  F=f[i][j]：表示前i件物品恰放入一个容量为v的背包可以获得的最大价值。  如果放了，问题变为“前i-1件物品放入剩下j-容量的背包中”，此时获得的最大价值F是  如果不放，问题转换为“前i-1件物品放入容量为j的背包中”，此时获得的最大价值F是  3.上述情况的优者为当前问题的最优解,所以最大价值为  其中，0<=i<=N,0<=j<=W  (3)算法代码实现  #include<iostream>  #include<time.h>  #include<map>  using namespace std;  int main()  {  // 假设5个物品的重量和价值分别为（5，12），（4，3），（7，10），（2，3），（6，6），背包容量15  int f[6][16] = {0};  int application[6][16];  int weight[5] = {5,4,7,2,6 };  int value[5] = {12,3,10,3,6};  int n = 10000;  while (n--)  {  for (int i = 1;i < 6;i++)  {  for (int j = 1;j < 16;j++)  {  if (j - weight[i - 1] >= 0)  {  if (f[i - 1][j] > f[i - 1][(j - weight[i - 1])] + value[i - 1])  {  f[i][j] = f[i - 1][j];//不放  application[i][j] = 0;  }  else  {  f[i][j] = f[i - 1][(j - weight[i - 1])] + value[i - 1];  application[i][j ] = 1;  }  //f[i][j] = f[i - 1][j] > f[i - 1][(j - weight[i - 1])] + value[i - 1] ? f[i - 1][j] : f[i - 1][(j - weight[i - 1])] + value[i - 1];  }  else  {  f[i][j] = f[i - 1][j];//不放  application[i][j] = 1;  }    }  }  }  cout << "动态规划表：" << endl;  for (int i = 0;i < 6;i++)  {  for (int j = 0;j < 16;j++)  {  cout << f[i][j]<<" ";  }  cout << endl;  }  cout << "放入背包的物品有：" << endl;  int j = 15;    for (int i = 5;i > 0;i--)  {  if (f[i][j] != f[i - 1][j])  {  cout<<"物品" << i << "加入背包" << endl;  j = j - weight[i-1];  }  }  return 0;  }  (4)数据输入和结果输出  假设5个物品的重量和价值分别为（5，12），（4，3），（7，10），（2，3），（6，6），背包容量15    (5)算法复杂性分析  在0-1背包问题上，动态规划算法的时间复杂度是O(V\*N)  2.硬币找零问题的动态规划算法  (1)问题描述  现在假设有几种硬币，如1、3、5，并且数量无限，请找出能够组成某个数目的找零所使用最少的硬币数。  (2)算法设计思想及递推方程建立  **本问题适用动态规划算法，**用待找零的数值k描述**子结构，**记作sum[k]，其值为所需的最小硬币数。  对于不同的硬币面值coin[0...n]，有  对应于给定数目的找零total，需要求解sum[total]的值。  (3)算法代码实现  #include<stdio.h>  #include<stdlib.h>  int coin[3] = { 1, 3, 5 }; /\* 硬币种类 \*/  int coin\_kind\_num = 3; /\* 硬币种类数量 \*/  int sum = 7; /\* 找零钱数 \*/  int min\_coin\_num; /\* 硬币最少数 \*/  int coin\_kind[1000]; /\* 存放最少数的方案 \*/  bool is\_find = false; /\* 是否找到硬币 \*/  bool output()  {  for (int i = 0; i < 1000; i++)  {  if (coin\_kind[i] == -1) break;  printf(" %d ", coin\_kind[i]);  }  printf("\n");  return coin\_kind;  }  void initCoinKind()  {  for (int i = 0; i < 100; i++) coin\_kind[i] = -1;  }  bool findCoin(int num, int sum) {  if (num == 0)  {  if (sum == 0) {  is\_find = true;  printf("找到最少硬币数 %d: ", min\_coin\_num);  output();  return true;  }  else if (sum > 0) {  return false;  }  else {  return false;  }  }  else  {  for (int i = 0; i < coin\_kind\_num; i++)  {  coin\_kind[num - 1] = coin[i];  findCoin(num - 1, sum - coin[i]);  }  }  }  bool startFind() {  int num = 1;  while (!is\_find) {  min\_coin\_num = num;  findCoin(num, sum);  num++;  }  return is\_find;  }  int main() {  initCoinKind();  startFind();  }  (4)数据输入和结果输出  硬币种类为1、3、5，找零钱数为7，输出结果如下    (5)算法复杂性分析  硬币找零问题，使用动态规划算法时间复杂度为O(C\*N)   1. 实验总结   总结来说，动态规划算法的基本思想与分治法类似，也是将待求解的问题分解为若干个子问题（阶段），按顺序求解子阶段，前一子问题的解，为后一子问题的求解提供了有用的信息。在求解任一子问题时，列出各种可能的局部解，通过决策保留那些有可能达到最优的局部解，丢弃其他局部解。依次解决各子问题，最后一个子问题就是初始问题的解。  通过本次实验，彻底弄清楚了动态规划的核心思想，以及明白了动态规划在问题中的使用方法。 |