**课程论文**

**题目：基于动态规划背包算法的甜品店应用**

|  |  |
| --- | --- |
| 课 程 名 称 | **算法分析与设计课程实验** |
| 考 查 学 期 | **2021-2022学年第2学期** |
| 考 查 方 式 | **课程设计** |
| 姓 名 | **Wby** |
| 学 号 |  |
| 专 业 | **软件工程** |
| 成 绩 |  |
| 指 导 教 师 | **祝玉杰** |

摘要

近年来，随着计算机行业的兴起，国内开始了针对这一行业设置并增生出大量的需求及应用，而算法作为计算机专业学习的基石，关于算法的研究与应用就显得十分重要甚至必要。而在算法中，有个十分经典的算法叫做动态规划，弄懂它对于我们学习算法之路有很大帮助。因此，本文针对动态规划算法这方面的知识，通过其中的经典问题之一，即动态规划背包问题（由于背包算法有多种情况，所以本文仅使用其中的0/1背包问题与完全背包问题进行应用），写出基于动态规划背包算法的甜品店应用，希望能够利用该应用，让我们对动态规划算法有一个更加清晰的认识，也希望能够通过该应用让更多人的算法学习之路更加顺畅。

**关键词：**算法;算法应用;动态规划算法;0/1背包问题;完全背包问题

目录

[摘要 2](#_Toc101390539)

[目录 3](#_Toc101390540)

[1 绪论 4](#_Toc101390541)

[2 关键技术和算法介绍 5](#_Toc101390542)

[3 负责模块的详细介绍 1](#_Toc101390543)1

[4 总结与展望 1](#_Toc101390543)9

[5 系统实现与代码 2](#_Toc101390544)0

[参考文献 33](#_Toc101390545)

[课程论文（设计）评分表 34](#_Toc101390546)

1 绪论

算法作为计算机专业学习的基石，关于算法的研究与应用是十分重要甚至必要的。在算法中，经典算法之一即动态规划，将作为本文主要研究及应用对象。针对动态规划算法这方面的知识，通过其中的经典问题之一，即动态规划背包问题（由于背包算法有多种情况，所以本文仅使用其中的0/1背包问题与完全背包问题进行应用），写出基于动态规划背包算法的甜品店应用，希望能够利用该应用，让我们对动态规划算法有一个更加清晰的认识，也希望能够通过该应用让更多人的算法学习之路更加顺畅。

在本文应用例子即基于动态规划背包算法的甜品店应用里，解决的是动态规划算法中的经典问题：背包问题（由于背包算法有多种情况，所以本文仅使用其中的0/1背包问题与完全背包问题进行应用）。本文将针对背包问题，利用基于动态规划背包算法的甜品店应用这一具体应用对其0/1背包问题与完全背包问题技术进行解释说明，从而帮助更多人理解并掌握其算法思想，从而能够解决问题。

2 关键技术和算法介绍

2.1 动态规划五部曲

动态规划，英文：Dynamic Programming，简称DP，每一个状态一定是由上一个状态推导出来的。

例如：有N件物品和一个最多能背重量为W 的背包。第i件物品的重量是weight[i]，得到的价值是value[i] 。每件物品只能用一次，求解将哪些物品装入背包里物品价值总和最大。

动态规划中dp[j]是由dp[j-weight[i]]推导出来的，然后取max(dp[j], dp[j - weight[i]] + value[i])。

2.1.1二维数组：

2.1.1.1 确定dp数组以及下标的含义

对于背包问题，有一种写法， 是使用二维数组，即dp[i][j] 表示从下标为[0-i]的物品里任意取，放进容量为j的背包，价值总和最大是多少。

2.1.1.2 确定递推公式

有两个方向推出来dp[i][j]：

2.1.1.2.1 不放物品i：由dp[i - 1][j]推出，即背包容量为j，里面不放物品i的最大价值，此时dp[i][j]就是dp[i - 1][j]。(其实就是当物品i的重量大于背包j的重量时，物品i无法放进背包中，所以被背包内的价值依然和前面相同。)

2.1.2.2.2 放物品i：由dp[i - 1][j - weight[i]]推出，dp[i - 1][j - weight[i]] 为背包容量为j - weight[i]的时候不放物品i的最大价值，那么dp[i - 1][j - weight[i]] + value[i] （物品i的价值），就是背包放物品i得到的最大价值。

所以递归公式： dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - weight[i]] + value[i]);

2.1.1.3 dp数组如何初始化

首先从dp[i][j]的定义出发，如果背包容量j为0的话，即dp[i][0]，无论是选取哪些物品，背包价值总和一定为0。

其他情况，状态转移方程 dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - weight[i]] + value[i]); 可以看出i 是由 i-1 推导出来，那么i为0的时候就一定要初始化。

dp[0][j]，即：i为0，存放编号0的物品的时候，各个容量的背包所能存放的最大价值。

那么很明显当 j < weight[0]的时候，dp[0][j] 应该是 0，因为背包容量比编号0的物品重量还小。

当j >= weight[0]时，dp[0][j] 应该是value[0]，因为背包容量放足够放编号0物品。

代码初始化如下：

|  |
| --- |
| // 初始化 dp  vector<vector<int>> dp(weight.size(), vector<int>(bagweight + 1, 0));  for (int j = weight[0]; j <= bagweight; j++) {  dp[0][j] = value[0];  } |

2.1.1.4 确定遍历顺序

有两个遍历的维度：物品与背包重量。本文先遍历物品，代码如下：

|  |
| --- |
| // weight数组的大小 就是物品个数  for(int i = 1; i < weight.size(); i++) { // 遍历物品  for(int j = 0; j <= bagweight; j++) { // 遍历背包容量  if (j < weight[i]) dp[i][j] = dp[i - 1][j];  else dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - weight[i]] + value[i]);  }  } |

dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - weight[i]] + value[i]); 递归公式中可以看出dp[i][j]是靠dp[i-1][j]和dp[i - 1][j - weight[i]]推导出来的。dp[i-1][j]和dp[i - 1][j - weight[i]] 都在dp[i][j]的左上角方向（包括正上方向）。

2.1.2 一维数组(使用滚动数组)：

2.1.2.1 确定dp数组的定义

在一维dp数组中，dp[j]表示：容量为j的背包，所背的物品价值可以最大为dp[j]。

2.1.2.2 一维dp数组的递推公式

dp[j]为 容量为j的背包所背的最大价值，可以通过dp[j - weight[i]]推导出来，dp[j - weight[i]]表示容量为j - weight[i]的背包所背的最大价值。

dp[j - weight[i]] + value[i] 表示 容量为 j - 物品i重量 的背包 加上 物品i的价值。（也就是容量为j的背包，放入物品i了之后的价值即：dp[j]）此时dp[j]有两个选择，一个是取自己dp[j] 相当于 二维dp数组中的dp[i-1][j]，即不放物品i，一个是取dp[j - weight[i]] + value[i]，即放物品i，指定是取最大的，毕竟是求最大价值。

所以递归公式为：dp[j] = max(dp[j], dp[j - weight[i]] + value[i])。可以看出相对于二维dp数组的写法，就是把dp[i][j]中i的维度去掉了。

2.1.2.3 一维dp数组如何初始化

假设物品价值都是大于0的，所以dp数组初始化的时候，都初始为0就可以了。

2.1.2.4 一维dp数组遍历顺序代码

|  |
| --- |
| for(int i = 0; i < weight.size(); i++) { // 遍历物品  for(int j = bagWeight; j >= weight[i]; j--) { // 遍历背包容量  dp[j] = max(dp[j], dp[j - weight[i]] + value[i]);  }  } |

一维dp遍历的时候，背包是从大到小，背包容量一定是要倒序遍历，一旦正序遍历了，那么物品0就会被重复加入多次！如果遍历背包容量放在上一层，那么每个dp[j]就只会放入一个物品，即：背包里只放入了一个物品。

倒序遍历的原因是，本质上还是一个对二维数组的遍历，并且右下角的值依赖上一层左上角的值，因此需要保证左边的值仍然是上一层的，从右向左覆盖。

2.2 0/1背包问题

2.2.1 问题与算法思想

实际问题中我们经常需要解决最优化问题, 即研究如何在限制条件下, 求出优化函数的最优值。

在0/1背包问题中, 需对容量为c的背包进行装载。从n个物品中选取装入背包的物品, 每件物品i的重量为, 价值为。对于可行的背包装载, 背包中物品的总重量不能超过背包的容量, 最佳装载是指所装入的物品价值最高, 即 取得最大值。约束条件为≤c和∈[0,1](a≤i≤n)。在这个表达式中，需求出的值。=1表示物品i放入背包中，=0表示物品i不放入背包[1,2]。

对重量为c=100的“背包”进行装载。从n=5个物品中选择需要装入“背包”的物品, 每件物品i的重量为，价值为。例如:w=[30, 18, 36, 40, 26], v=[40, 25, 24, 50, 26]。使得在满足取得最大值，其中约束条件为≤c和∈[0，1]，(1≤i≤5)。

因此，可以将上述总结为：有n件物品和一个最多能背重量为w 的背包。第i件物品的重量是weight[i]，得到的价值是value[i] 。每件物品只能用一次，求解将哪些物品装入背包里物品价值总和最大。

2.2.2 核心代码

本次应用采用二维数组写法，如下：

|  |
| --- |
| // weight数组的大小 就是物品个数  for(int i = 1; i < weight.size(); i++) { // 遍历物品  for(int j = 0; j <= bagweight; j++) { // 遍历背包容量  if (j < weight[i]) dp[i][j] = dp[i - 1][j];  else dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - weight[i]] + value[i]);  }  } |

若为一维数组写法，则如下：

|  |
| --- |
| for(int i = 0; i < weight.size(); i++) { // 遍历物品  for(int j = bagWeight; j >= weight[i]; j--) { // 遍历背包容量  dp[j] = max(dp[j], dp[j - weight[i]] + value[i]);  }  } |

注意：在一维数组中，0/1背包内嵌的循环是从大到小遍历，为了保证每个物品仅被添加一次。

2.3 完全背包

2.3.1 问题与算法思想

在完全背包问题中，有N件物品和一个最多能背重量为W的背包。第i件物品的重量是weight[i]，得到的价值是value[i] 。每件物品都有无限个（也就是可以放入背包多次），求解将哪些物品装入背包里物品价值总和最大。

完全背包和01背包问题唯一不同的地方就是，每种物品有无限件。

2.3.2 核心代码

本次应用采用一维数组写法，如下：

|  |
| --- |
| // 先遍历物品，再遍历背包  for(int i = 0; i < weight.size(); i++) { // 遍历物品  for(int j = weight[i]; j <= bagWeight ; j++) { // 遍历背包容量  dp[j] = max(dp[j], dp[j - weight[i]] + value[i]);  }  } |

注意：完全背包的物品是可以添加多次的，所以要从小到大去遍历。

2.3.3 举例

2.3.3.1 问题

袋子为：

|  |  |
| --- | --- |
|  | 容量 |
| 小号纸袋 | 7 |
| 中号纸袋 | 15 |
| 大号纸袋 | 30 |
| 超大号纸袋 | 50 |

物品为：

|  |  |  |
| --- | --- | --- |
|  | 重量 | 价值 |
| 葡式蛋挞 | 5 | 7 |
| 抹茶泡芙 | 20 | 15 |
| 提拉米苏 | 25 | 20 |
| 草莓甜甜圈 | 30 | 35 |

每件甜品有无数件。

定义：dp[i][j]表示从下标为[0-i]的甜品里任意取，放进容量为j的纸袋，价值总和最大是多少。

请计算出最终顾客选择的甜品总能量值。

2.2.3.2 核心代码

|  |
| --- |
| // 完全背包  // 先遍历物品，在遍历背包  int choose\_max\_2(vector<int> weight, vector<int> value, int bagWeight) {  vector<int> dp(bagWeight + 1, 0);  // 遍历物品  for(int i = 0; i < weight.size(); i++) {  // 遍历背包容量  for(int j = weight[i]; j <= bagWeight; j++) {  dp[j] = max(dp[j], dp[j - weight[i]] + value[i]);  }  }  return dp[bagWeight];  } |

本次假设选择了2个提拉米苏，1个草莓甜甜圈和超大号纸袋，则dp状态图如下：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | ... | 30 | ... | 40 | ... | 50 |
| 用提拉米苏遍历超大号纸袋 | 0 | 0 | ... | 20 | ... | 20 | ... | 20 |
| 用提拉米苏遍历超大号纸袋 | 0 | 0 | ... | 20 | ... | 40 | ... | 40 |
| 用草莓甜甜圈遍历超大号纸袋 | 0 | 0 | ... | 35 | ... | 40 | ... | 40 |

dp[j] 是根据 下标j之前所对应的dp[j]计算出来的。 只要保证下标j之前的dp[j]都是经过计算的就可以了。

**3 负责模块的详细介绍**

在本次基于动态规划背包算法的甜品店应用中，我们实现的功能函数总共有9个，如图1所示。

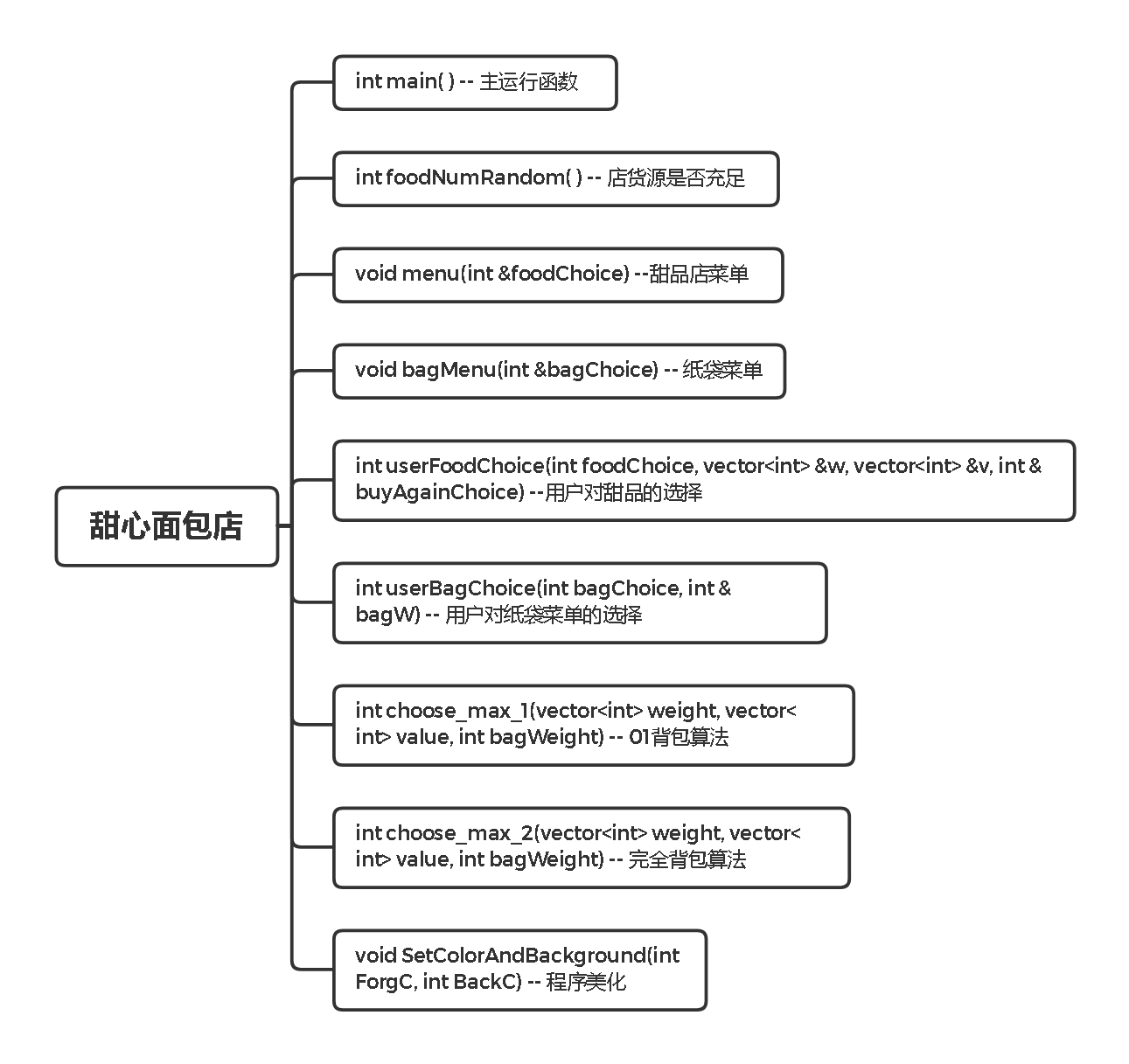


图1 基于动态规划背包算法的甜品店的实现函数

在本次基于动态规划背包算法的甜品店应用的总体实现中，我负责了4部分，分别为：1.总体框架与流程设计与实现；2.甜品存量情况随机设计与实现；3.袋子菜单设计与选择实现；4.购买甜品功能中完全背包算法实现。接下来我将一一进行介绍。

3.1 总体框架与流程设计与实现

3.1.1 负责模块的作用及意义

在总体框架与流程设计与实现这部分中，我负责设计并绘制基于动态规划背包算法的甜品店应用的业务流程图，如图2所示；同时我负责实现基于动态规划背包算法的甜品店应用的主函数代码总体逻辑思路与代码实现框架。

在这部分中，用到的函数为图1中的主运行函数即int main()，为总体应用的实现提供总体逻辑思路流程，从而达到基于动态规划背包算法的甜品店总体顺利实现。

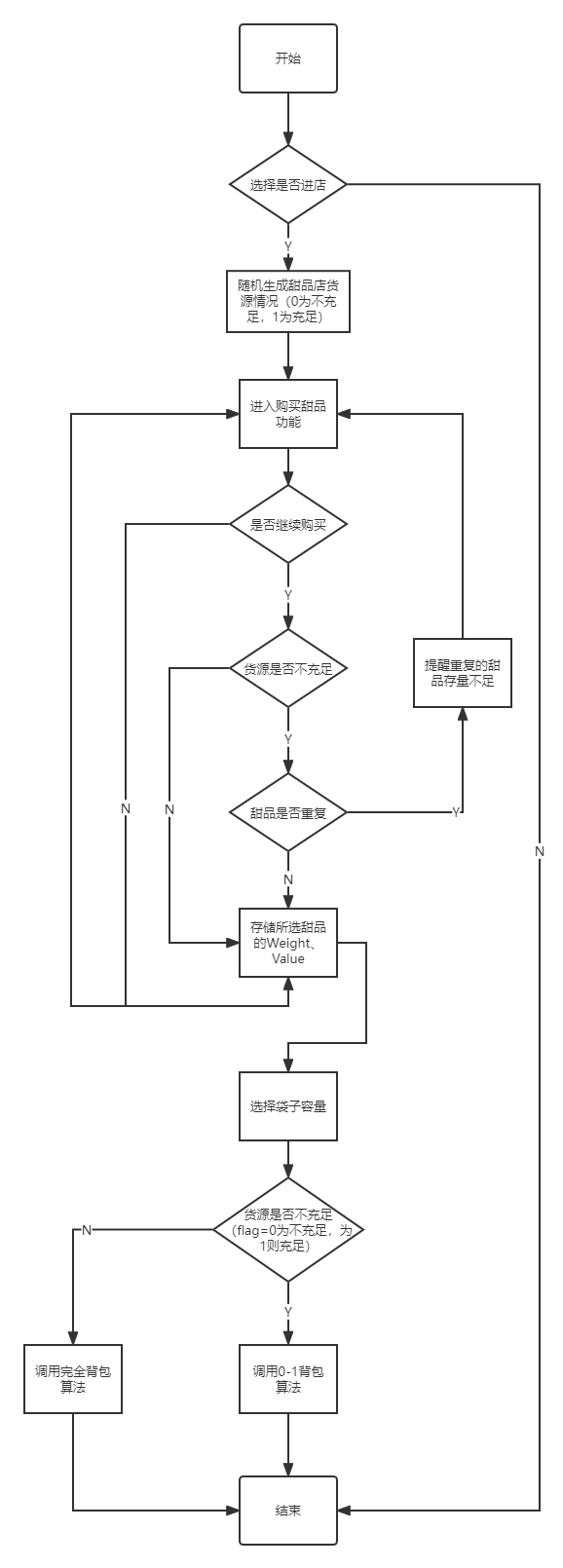


图2 基于动态规划背包算法的甜品店应用的业务流程图

3.1.2 具体代码

|  |
| --- |
| // 主函数  int main() {  int userChoice, foodChoice = 0, bagChoice, buyAgainChoice = 1;  int count = 0, result, flag = 0; //flag表示货源是否充足，0：不充足，1：充足  vector<int> foodRepeat = {}; // 当所有物品仅剩一件时，存储用户选过的foodChoice，防止重选  vector<int> w = {}; // 依次存储用户所选甜品的weight  vector<int> v = {}; // 依次存储用户所选甜品的value  vector<string> foodList = {"葡式蛋挞", "抹茶泡芙", "提拉米苏", "草莓甜甜圈"}; //记录甜品种类  int bagW = 0; // 用户所选纸袋的容量  // 老板显示面包店菜单  cout << "【老板】：欢迎光临Honey Bakery！请问您需要买什么吗？ " << endl;  cout << "【我】：1.是的，我刚来这个城市，饿了一天了！ \n 2.不用了，我只是路过。。 " << endl;  SetColorAndBackground(0, 7);  cout << "你选择： " << endl;  cin >> userChoice;  if(userChoice != 1 && userChoice != 2) {  cout << "[系统提示]请输入数字1或者数字2！ " << endl;  cin >> userChoice;  }  cout << "\n" << endl;  switch (userChoice) {  case 1:  // 甜品的选择  // 随机甜品存货情况: 0:每个甜品仅存1件; 1:甜品存量充足  if (foodNumRandom() == 0) {//货源不充足  SetColorAndBackground(7, 0);  cout << "【老板】:实在抱歉！今天生意太火了，目前店里所有甜品都仅剩最后一件了。" << endl;  cout << "【我】:没关系！" << endl;  cout << "【老板】:好的,请看一下我们的菜单。" << endl;  while (buyAgainChoice != 2) {//甜品的w、v进行存储  count++;  // cout << "第" << count << "次选择" << endl;  menu(foodChoice);  // 如果第一次输入，那么直接进入甜品的选择  if (count < 2) {  cout << "将用户选择【" << foodChoice << "】开始存储" << endl;  userFoodChoice(foodChoice, w, v, buyAgainChoice);  foodRepeat.push\_back(foodChoice);  } else {  // 先将foodRepeat[j]挨个和foodChoice作比较，如果都不相等，再进行对应foodChoice的w和v存储  int repeat\_flag = 1; // 默认情况下，是无重复值的  for (int i = 0; i < foodRepeat.size(); i++) {  // 通过遍历，判断用户是否输入重复的foodChoice  // cout << "foodRepeat[" << i << "] =" << foodRepeat[i] << endl;  if (foodChoice == foodRepeat[i]) {  SetColorAndBackground(4, 0);  cout << "[" << foodList[foodRepeat[i] - 1] << "]目前库存量：0，已无法再选择。" << endl;  repeat\_flag = 2; // 表示有重复值  break;  }  }  // 只有无重复值的情况下，才能进行对应foodChoice的w和v存储  if (repeat\_flag == 1) {  cout << "将用户选择【" << foodChoice << "】开始存储" << endl;  userFoodChoice(foodChoice, w, v, buyAgainChoice);  foodRepeat.push\_back(foodChoice);  }  }  }  } else {//货源充足  flag = 1;  SetColorAndBackground(7, 0);  cout << "【老板】:好的,请看下我们的菜单。" << endl;  cout << "【我】:好的，谢谢。" << endl;  while (buyAgainChoice != 2) {  menu(foodChoice);  userFoodChoice(foodChoice, w, v, buyAgainChoice);  }  }  // 袋子的选择  SetColorAndBackground(4,0);  cout << "[甜品选购完毕]" << endl;  SetColorAndBackground(7, 0);  cout << "【老板】：好的，请选择一下袋子。" << endl;  bagMenu(bagChoice);  userBagChoice(bagChoice, bagW);  // 结账  // 完成购物  SetColorAndBackground(7, 0);  if (flag == 0) {  result = choose\_max\_1(w, v, bagW);//算法调用  } else {  result = choose\_max\_2(w, v, bagW);  }  cout << "【老板】：谢谢您的惠顾。" << endl;  cout << "【老板】：先生，您选择的容量为【" << bagW << "】的袋子能够装下物品的最大能量值是： " << result << endl;  break;  case 2:  cout << "【老板】：没关系，有空常来看看啊。 " << endl;  break;  }  return 0;  } |

3.2甜品存量情况随机设计与实现

3.3.1 负责模块的作用及意义

在甜品存量情况随机设计与实现这部分中，我负责设计并实现基于动态规划背包算法的甜品店应用的甜品存量情况功能，从而达到随机生成甜品存量情况的实现。

在这部分中，用到的函数为图1中的甜品存量情况函数int foodNumRandom()，实现了总体应用的甜品存量情况。

3.3.2 具体代码

|  |
| --- |
| //甜品存量是否充足随机 0:每个甜品仅存1件; 1:甜品存量充足  int foodNumRandom(){  srand((unsigned)time(NULL));  int a = rand() % 2; // 要取得[0,1]的随机整数  return a;  }  //主函数中的应用为if (foodNumRandom() == 0) {//货源不充足}else{//货源充足} |

3.3袋子菜单设计与选择实现

3.3.1 负责模块的作用及意义

在袋子菜单设计与选择实现这部分中，我负责设计并实现基于动态规划背包算法的甜品店应用的袋子菜单功能，从而达到顾客，即代码运行时的【我】，能够选择出想要的袋子能够承载甜品的重量。

在这部分中，用到的函数为图1中的纸袋菜单函数void bagMenu(int &bagChoice)和用户对纸袋菜单的选择函数int userBagChoice(int bagChoice, int &bagW)，实现了总体应用的纸袋菜单设计与选择。

3.3.2 具体代码

|  |
| --- |
| //纸袋菜单  void bagMenu(int &bagChoice) {  SetColorAndBackground(6,0);  cout << "================= HONEY BAKERY ================" << endl;  cout << " 1.小号纸袋 容量:7" << endl;  cout << " 2.中号纸袋 容量:15" << endl;  cout << " 3.大号纸袋 容量:30" << endl;  cout << " 4.超大号纸袋 容量:50" << endl;  cout << "===============================================" << endl;  SetColorAndBackground(0,7); // 黑前景白背景  cout << "请选择你想要的袋子：" << endl;  cin >> bagChoice;  // 判断用户是否输出内容超出范围  if(bagChoice < 1 || bagChoice > 4){  cout << "[系统提示]请输入数字1-4之内的任意整数！" << endl;  cin >> bagChoice;  }  cout << "\n" << endl;  } |
| // 对袋子的选择int userBagChoice(int bagChoice, int &bagW){  switch(bagChoice) {  case 1:  bagW = 7;  break;  case 2:  bagW = 15;  break;  case 3:  bagW = 30;  break;  case 4:  bagW = 50;  break;  }  return 0;  } |

**3.4购买甜品功能中完全背包算法实现**

3.4.1 负责模块的作用及意义

在购买甜品功能中完全背包算法实现这部分中，我负责设计并实现基于动态规划背包算法的甜品店应用购买甜品功能中的完全背包算法功能，从而实现计算出顾客，即代码运行中的【我】，选择的甜品在选择的纸袋中的最大能量值。

在这部分中，用到的函数为图1中的完全背包算法函数int choose\_max\_2(vector<int> weight, vector<int> value, int bagWeight)，实现了总体应用购买甜品功能中的完全背包算法，从而实现计算出顾客，即代码运行中的【我】，选择的甜品在选择的纸袋中的最大能量值。

3.4.2 具体代码

|  |
| --- |
| // 完全背包  // 先遍历物品，在遍历背包  int choose\_max\_2(vector<int> weight, vector<int> value, int bagWeight) {  vector<int> dp(bagWeight + 1, 0);  // 遍历物品  for(int i = 0; i < weight.size(); i++) {  // 遍历背包容量  for(int j = weight[i]; j <= bagWeight; j++) {  dp[j] = max(dp[j], dp[j - weight[i]] + value[i]);  }  }  return dp[bagWeight];  } |

4 总结与展望

在本次基于动态规划背包算法的甜品店应用中，我们实现的功能函数总共有9个，如图1所示。总体实现中，我负责了4部分，分别为：1.总体框架与流程设计与实现；2.甜品存量情况随机设计与实现；3.袋子菜单设计与选择实现；4.购买甜品功能中完全背包算法实现。

(一)总体框架与流程设计与实现中，用到的函数为图1中的主运行函数即int main()，为总体应用的实现提供总体逻辑思路流程，从而达到基于动态规划背包算法的甜品店总体顺利实现；

(二)甜品存量情况随机设计与实现中，用到的函数为图1中的甜品存量情况函数int foodNumRandom()，实现了总体应用的甜品存量情况；

(三)袋子菜单设计与选择实现中，用到的函数为图1中的纸袋菜单函数void bagMenu(int &bagChoice)和用户对纸袋菜单的选择函数，实现了总体应用的纸袋菜单设计与选择；

(四)购买甜品功能中完全背包算法实现中，用到的函数为图1中的完全背包算法函数int choose\_max\_2(vector<int> weight, vector<int> value, int bagWeight)，实现了总体应用购买甜品功能中的完全背包算法，从而实现计算出顾客，即代码运行中的【我】，选择的甜品在选择的纸袋中的最大能量值。

在动态规划背包问题中，我们了解并掌握了：

(一)动态规划五部曲中了解并掌握状态转移公式；

(二)通过0/1背包算法了解并掌握滚动数组；

(三)0/1背包算法的理论与实践；

(四)完全背包算法的理论与实践。

综上，本文选取算法中的经典算法之一，即动态规划，作为主要研究及应用对象，针对动态规划算法这方面的知识，通过其中的经典问题之一，即动态规划背包问题（由于背包算法有多种情况，所以本文仅使用其中的0/1背包问题与完全背包问题进行应用），在关键技术和算法部分介绍了其算法思想与核心代码，并最终写出基于动态规划背包算法的甜品店应用，希望看完本文后读者能够希望能够对动态规划算法有一个更加清晰的认识，从而在算法学习之路上更加顺畅。

5 系统实现与代码

基于动态规划背包算法的甜品店应用的系统实现代码如下：

|  |
| --- |
| #include <iostream>  #include <vector>  #include <stdlib.h>  #include <time.h>  #include <bits/stdc++.h>  #include <windows.h>  using namespace std;  // 字体带颜色  void SetColorAndBackground(int ForgC, int BackC) {  WORD wColor = ((BackC & 0x0F) << 4) + (ForgC & 0x0F);  SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), wColor);  }  // 01背包  // 业务逻辑  int choose\_max\_1(vector<int> weight, vector<int> value, int bagWeight){  // 创建二维数组  vector<vector<int>> dp(weight.size(), vector<int>(bagWeight + 1, 0));  // 初始化二维数组  for(int j = weight[0]; j <= bagWeight; j++) {  dp[0][j] = value[0];  }  // 从物品开始遍历  for(int i = 1; i < weight.size(); i++) {  for(int j = 0; j <= bagWeight; j++) {  if(j < weight[i]) {  dp[i][j] = dp[i - 1][j];  }else {  dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - weight[i]] + value[i]);  }  }  }  return dp[weight.size() - 1][bagWeight];  }  // 完全背包  // 先遍历物品，在遍历背包  int choose\_max\_2(vector<int> weight, vector<int> value, int bagWeight) {  vector<int> dp(bagWeight + 1, 0);  // 遍历物品  for(int i = 0; i < weight.size(); i++) {  // 遍历背包容量  for(int j = weight[i]; j <= bagWeight; j++) {  dp[j] = max(dp[j], dp[j - weight[i]] + value[i]);  }  }  return dp[bagWeight];  }  // 面包店菜单  void menu(int &foodChoice) {  SetColorAndBackground(6,0);  cout << "================= HONEY BAKERY ================" << endl;  cout << " 1.葡式蛋挞 重量:5 能量值: 7" << endl;  cout << " 2.抹茶泡芙 重量:20 能量值: 15" << endl;  cout << " 3.提拉米苏 重量:25 能量值: 20" << endl;  cout << " 4.草莓甜甜圈 重量:30 能量值: 35" << endl;  cout << "===============================================" << endl;  SetColorAndBackground(0,7);  cout << "请选择你想要的甜品：" << endl;  cin >> foodChoice;  // 判断用户是否输出内容超出范围  if(foodChoice < 1 || foodChoice > 4){  cout << "[系统提示]请输入数字1-4之内的任意整数！" << endl;  cin >> foodChoice;  }  cout << "\n" << endl;  }  //纸袋菜单  void bagMenu(int &bagChoice) {  SetColorAndBackground(6,0);  cout << "================= HONEY BAKERY ================" << endl;  cout << " 1.小号纸袋 容量:7" << endl;  cout << " 2.中号纸袋 容量:15" << endl;  cout << " 3.大号纸袋 容量:30" << endl;  cout << " 4.超大号纸袋 容量:50" << endl;  cout << "===============================================" << endl;  SetColorAndBackground(0,7); // 黑前景白背景  cout << "请选择你想要的袋子：" << endl;  cin >> bagChoice;  // 判断用户是否输出内容超出范围  if(bagChoice < 1 || bagChoice > 4){  cout << "[系统提示]请输入数字1-4之内的任意整数！" << endl;  cin >> bagChoice;  }  cout << "\n" << endl;  }  // 用户对甜品的选择  int userFoodChoice(int foodChoice, vector<int> &w, vector<int> &v, int &buyAgainChoice) {  vector<string> foodList= {"葡式蛋挞", "抹茶泡芙", "提拉米苏", "草莓甜甜圈"}; //记录甜品种类  SetColorAndBackground(7,0);  if(foodChoice == 1){ // 选择葡式蛋挞  w.push\_back(5);  v.push\_back(7);  }else if(foodChoice == 2) { // 抹茶泡芙  w.push\_back(20);  v.push\_back(15);  }else if(foodChoice == 3) { // 提拉米苏  w.push\_back(25);  v.push\_back(20);  }else if(foodChoice == 4) { // 草莓甜甜圈  w.push\_back(30);  v.push\_back(35);  }  cout << "【老板】：好的，你选择了"<< foodList[foodChoice - 1] << "，还需要其它的吗？" << endl;  cout << "【我】：1.嗯，我再看看。 \n 2.不用了，谢谢。 " << endl;  SetColorAndBackground(0,7);  cout << "你选择：" << endl;  cin >> buyAgainChoice;  // 判断用户是否输入错误值  if(buyAgainChoice != 1 && buyAgainChoice != 2) { //解决方案链接https://blog.csdn.net/weixin\_50952196/article/details/108903861  cout << "【系统提示】：请输入数字1或者数字2！" << endl;  cin >> buyAgainChoice;  }  cout << "\n" << endl;  return buyAgainChoice;  }  // 对袋子的选择  int userBagChoice(int bagChoice, int &bagW){  switch(bagChoice) {  case 1:  bagW = 7;  break;  case 2:  bagW = 15;  break;  case 3:  bagW = 30;  break;  case 4:  bagW = 50;  break;  }  return 0;  }  //甜品存量是否充足随机  int foodNumRandom(){  srand((unsigned)time(NULL));  int a = rand() % 2; // 要取得[0,1]的随机整数  return a;  }  // 主函数  int main() {  int userChoice, foodChoice = 0, bagChoice, buyAgainChoice = 1;  int count = 0, result, flag = 0; //flag表示货源是否充足，0：不充足，1：充足  vector<int> foodRepeat = {}; // 当所有物品仅剩一件时，存储用户选过的foodChoice，防止重选  vector<int> w = {}; // 依次存储用户所选甜品的weight  vector<int> v = {}; // 依次存储用户所选甜品的value  vector<string> foodList = {"葡式蛋挞", "抹茶泡芙", "提拉米苏", "草莓甜甜圈"}; //记录甜品种类  int bagW = 0; // 用户所选纸袋的容量  // 老板显示面包店菜单  cout << "【老板】：欢迎光临Honey Bakery！请问您需要买什么吗？ " << endl;  cout << "【我】：1.是的，我刚来这个城市，饿了一天了！ \n 2.不用了，我只是路过。。 " << endl;  SetColorAndBackground(0, 7);  cout << "你选择： " << endl;  cin >> userChoice;  if(userChoice != 1 && userChoice != 2) {  cout << "[系统提示]请输入数字1或者数字2！ " << endl;  cin >> userChoice;  }  cout << "\n" << endl;  switch (userChoice) {  case 1:  // 甜品的选择  // 随机甜品存货情况: 0:每个甜品仅存1件; 1:甜品存量充足  if (foodNumRandom() == 0) {//货源不充足  SetColorAndBackground(7, 0);  cout << "【老板】:实在抱歉！今天生意太火了，目前店里所有甜品都仅剩最后一件了。" << endl;  cout << "【我】:没关系！" << endl;  cout << "【老板】:好的,请看一下我们的菜单。" << endl;  while (buyAgainChoice != 2) {//甜品的w、v进行存储  count++;  // cout << "第" << count << "次选择" << endl;  menu(foodChoice);  // 如果第一次输入，那么直接进入甜品的选择  if (count < 2) {  cout << "将用户选择【" << foodChoice << "】开始存储" << endl;  userFoodChoice(foodChoice, w, v, buyAgainChoice);  foodRepeat.push\_back(foodChoice);  } else {  // 先将foodRepeat[j]挨个和foodChoice作比较，如果都不相等，再进行对应foodChoice的w和v存储  int repeat\_flag = 1; // 默认情况下，是无重复值的  for (int i = 0; i < foodRepeat.size(); i++) {  // 通过遍历，判断用户是否输入重复的foodChoice  // cout << "foodRepeat[" << i << "] =" << foodRepeat[i] << endl;  if (foodChoice == foodRepeat[i]) {  SetColorAndBackground(4, 0);  cout << "[" << foodList[foodRepeat[i] - 1] << "]目前库存量：0，已无法再选择。" << endl;  repeat\_flag = 2; // 表示有重复值  break;  }  }  // 只有无重复值的情况下，才能进行对应foodChoice的w和v存储  if (repeat\_flag == 1) {  cout << "将用户选择【" << foodChoice << "】开始存储" << endl;  userFoodChoice(foodChoice, w, v, buyAgainChoice);  foodRepeat.push\_back(foodChoice);  }  }  }  } else {//货源充足  flag = 1;  SetColorAndBackground(7, 0);  cout << "【老板】:好的,请看下我们的菜单。" << endl;  cout << "【我】:好的，谢谢。" << endl;  while (buyAgainChoice != 2) {  menu(foodChoice);  userFoodChoice(foodChoice, w, v, buyAgainChoice);  }  }  // 袋子的选择  SetColorAndBackground(4,0);  cout << "[甜品选购完毕]" << endl;  SetColorAndBackground(7, 0);  cout << "【老板】：好的，请选择一下袋子。" << endl;  bagMenu(bagChoice);  userBagChoice(bagChoice, bagW);  // 结账  // 完成购物  SetColorAndBackground(7, 0);  if (flag == 0) {  result = choose\_max\_1(w, v, bagW);//算法调用  } else {  result = choose\_max\_2(w, v, bagW);  }  cout << "【老板】：谢谢您的惠顾。" << endl;  cout << "【老板】：先生，您选择的容量为【" << bagW << "】的袋子能够装下物品的最大能量值是： " << result << endl;  break;  case 2:  cout << "【老板】：没关系，有空常来看看啊。 " << endl;  break;  }  return 0;  } |

效果如下：

0/1背包

|  |
| --- |
| 45aa71f7bd82886ae2dfa15313cbd916d556acaa67e1248efb4d25d2c87520b4650e4ac2fe22f3ad658e7e8d75307 |

完全背包

|  |
| --- |
| d818966674eb36fa633098a04908b263880e67696ee89a3bc7c2eb161910e75741fb5a81e4dd00cb98f3ebaf205a25a07e9ddc940e2941847a07e710cbca |

参考文献

[1]黄与林.0/1背包问题的贪心算法[J].鄂州大学学报, 2006, 13 (6) .

[2]王红珍, 李竹林, 延飞波.基于0/1背包问题的两种算法[J]信息技术, 2011, (02) .

课程论文（设计）评分表

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **一级指标** | **二级指标** | **分值** | **等级评定参考标准** | | | | | **得分** |
| 优秀  (100—90分) | 良好  (89—80分) | 中等  （79一70分） | 及格  (69—60分) | 不及格  （60以下) |  |
| **选题30** | **方向** | **10** | 全面符合培养目标，体现学科专业的基本要求。 | 符合培养目标，整体体现学科专业的基本要求。 | 较符合培养目标和体现专业的基本要求。 | 基本符合培养目标和体现专业的基本要求。 | 与培养目标不符或不能体现学科专业的基本要求。 |  |
| **价值** | **10** | 有很好的科学意义和实用价值。 | 有较高的科学意义和实用价值。 | 有一定科学意义和实用价值。 | 尚有科学意义和实用价值。 | 没有科学意义和实用价值。 |  |
| **难度**  **分量** | **10** | 研究领域涉及适当的新、深、广，难度适当偏大，待分析材料丰富，份量厚重，能达到综合训练的目的。 | 有较大的研究难度，待分析材料丰富，工作量适中。 | 在一般性领域中进行研究，有一定难度，工作量达到训练要求。 | 在一般性领域中进行研究，难度较小，工作量基本达到训练要求。 | 进行重复性研究，工作量未达到训练要求。 |  |
| **设计与答辩70** | **研究成果基础理论与专业知识** | **10** | 对研究的问题能较深刻分析或有独到见解，成果突出，反映出作者很好地掌握了有关基础理论与专业技术。 | 对研究的问题能正确分析或有新见解，成果比较突出，反映出作者较好地掌握了有关基础理论与专业知识。 | 对研究的问题能提出自己的见解，成果有一定意义，反映出作者基本掌握了有关基础理论与专业知识。 | 研究能力较弱，对某些问题提不出个人见解，未取得什么成果，反映出作者基础理论和专业知识掌握得不扎实。 | 缺乏研究能力，未取得任何成果，反映出作者基础理论和专业知识很不扎实。 |  |
| **技术水平** | **40** | 设计合理，理论分析与计算正确，实验数据准确可靠。 | 设计比较合理，理论分析与计算正确，实验数据比较准确。 | 设计比较合理，理论分析与计算基本准确，实验数据基本准确。 | 设计基本合理，理论分析与计算无大错，实验数据无原则差错。 | 设计不合理，理论分析与计算有原则错误，实验数据不可靠。 |  |
| **回**  **答**  **/**  **独立完成** | **20** | 能准确流利地回答各种问题回答问题沉着冷静，言简意深，重点突出，准确无误。 | 能较准确流利地回答与论文有关的问题。 | 能较恰当地回答与论文有关的问题。 | 答辩无大错，经提示后能作补充纠正。 | 主要问题答不出或错误较多，经提示后仍不能正确回答有关问题。 |  |