**算法分析与设计实验报告**

**第 2 次实验**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 姓名 | 邹林壮 | 学号 | 202208040412 | | 班级 | 计算机科学与技术（拔尖班）2201 |
| 时间 | 2024.10.19 | 地点 | 院楼432 | | | |
| 实验名称 | 贪心算法求解背包问题 | | | | | |
| 实验目的 | 通过上机实验，要求掌握贪心算法的问题描述、算法设计思想、程序设计。 | | | | | |
| 实验原理 | 利用贪心策略对物品进行排序，然后按照单位价值从高到低依次取出，直到受到容量的限制，最终计算出程序运行所需要的时间。 | | | | | |
| 实验步骤 | 1. 首先确定输入输出，输入为n件物品，n件物品的质量w，价值v，背包的容量c,输出为装入物品的最大价值,约束是容量c，可以取出部分。 2. 采用贪心策略，首先要选取单位价值最大的物品，因此首先对物品的单位价值进行排序，从高往低依次选取，直到背包全部被填满。 | | | | | |
| 关键代码 | // 比较函数  bool cmp(const object &a, const object &b) {  return a.vpw > b.vpw; // 降序排序  }  // 贪心算法实现的背包问题  double bag\_greedy(int c, int n, object \*x) {  object x1[N]; // 临时存储物品  for (int i = 1; i <= n; i++) {  x1[i].w = x[i].w;  x1[i].v = x[i].v;  x1[i].vpw = x[i].vpw;  }  double max = 0; // 最大价值  sort(x1 + 1, x1 + n + 1, cmp); // 排序  for (int i = 1; i <= n && c > 0; i++) {  if (x1[i].w <= c) {  max += x1[i].v; // 完全装入  c -= x1[i].w; // 剩余容量  } else {  max += x1[i].vpw \* c; // 部分装入  c = 0; // 背包装满  }  }  return max; // 返回最大价值  } | | | | | |
| 测试结果 | 1）正确性：保证了结果的正确性  IMG_256  2）时间复杂度：O(nlogn)，主要由排序决定。  空间复杂度：O(N)，主要由物品结构体数组的大小决定。  3）构造了三个规模的数据集：100\*1000，1000\*5000，100000\*1000000  IMG_256 | | | | | |
| 实验心得 | 1. 针对贪心算法，首先要确保问题的最优化性质，不能只得到一个局部最优解，适用于被局部最优解构成全局最优解的问题，而且贪心算法在时间和空间上效率较高。 2. 在很多算法中，排序对问题的规模和可解性影响很大。 | | | | | |
| 实验得分 |  | 助教签名 | |  | | |

**附录：完整代码**

**问题代码：**

**#include<iostream>**

**#include<algorithm>**

**#include<iomanip>**

**using namespace std;**

**#define N 1000001**

**typedef long long ll;**

**ll n;**

**ll c;**

**struct object{**

**ll w;**

**ll v;**

**double vpw;**

**};**

**object x[N];**

**bool cmp(const object &a,const object &b){**

**return a.vpw>b.vpw;**

**}**

**double bag\_greedy(int c,int n,object \*x){**

**object x1[n+1];**

**for(int i=1;i<=n;i++){**

**x1[i].w=x[i].w;**

**x1[i].v=x[i].v;**

**x1[i].vpw=x[i].vpw;**

**}**

**double max;**

**sort(x1+1,x1+n+1,cmp);**

**for(int i=1;i<=n&&c>0;i++){**

**if(x1[i].w<=c){**

**max+=x1[i].v;**

**c-=x1[i].w;**

**//cout<<x1[i].w<<endl;**

**}else{**

**max+=x1[i].vpw\*c;**

**c=0;**

**}**

**}**

**return max;**

**}**

**int main(){**

**cin>>n>>c;**

**for(int i=1;i<=n;i++){**

**cin>>x[i].w>>x[i].v;**

**x[i].vpw=(double)x[i].v/x[i].w;**

**}**

**double max=bag\_greedy(c,n,x);**

**cout<<fixed<<setprecision(2)<<max;**

**}**

**计算时间代码：**

**#include <iostream>**

**#include <algorithm>**

**#include <iomanip>**

**#include <vector>**

**#include <cstdlib>**

**#include <ctime>**

**#include <chrono>**

**using namespace std;**

**typedef long long ll;**

**struct object {**

**ll w; // weight**

**ll v; // value**

**double vpw; // value per weight (v / w)**

**};**

**// Comparator function to sort objects by value/weight ratio in descending order**

**bool cmp(const object &a, const object &b) {**

**return a.vpw > b.vpw;**

**}**

**// Greedy algorithm for fractional knapsack problem**

**double bag\_greedy(ll c, ll n, vector<object> &x) {**

**vector<object> x1 = x; // Create a copy of the input array**

**double maxValue = 0.0; // Initialize maximum value to 0.0**

**// Sort the objects by value/weight ratio in descending order**

**sort(x1.begin(), x1.end(), cmp);**

**// Select items according to the greedy strategy**

**for (ll i = 0; i < n && c > 0; i++) {**

**if (x1[i].w <= c) {**

**maxValue += x1[i].v; // Take the full item**

**c -= x1[i].w; // Reduce capacity by the item's weight**

**} else {**

**maxValue += x1[i].vpw \* c; // Take fraction of the item**

**c = 0; // Knapsack is full**

**}**

**}**

**return maxValue;**

**}**

**// Function to generate random data**

**void generate\_random\_data(ll n, ll c, vector<object> &x) {**

**srand(time(0));**

**for (ll i = 0; i < n; i++) {**

**ll weight = rand() % 10000 + 1; // Random weight between 1 and 10000**

**ll value = rand() % 10000 + 1; // Random value between 1 and 10000**

**x[i] = {weight, value, (double)value / weight};**

**}**

**}**

**int main() {**

**// 三种不同规模的数据集**

**vector<pair<ll, ll>> sizes = {{100, 1000}, {1000, 5000}, {100000, 1000000}};**

**for (auto &size : sizes) {**

**ll n = size.first; // Number of items**

**ll c = size.second; // Capacity of the knapsack**

**vector<object> x(n); // Create a vector of size n to store objects**

**// Generate random data for the objects**

**generate\_random\_data(n, c, x);**

**// Measure execution time**

**auto start = chrono::high\_resolution\_clock::now();**

**double maxValue = bag\_greedy(c, n, x);**

**auto end = chrono::high\_resolution\_clock::now();**

**auto duration = chrono::duration\_cast<chrono::milliseconds>(end - start);**

**// Output the result**

**cout << "问题规模: 物品数量 = " << n << "，背包容量 = " << c << endl;**

**cout << "运行时间: " << duration.count() << " 毫秒。" << endl;**

**cout << "-------------------------------------------" << endl;**

**}**

**return 0;**

**}**