**《算法设计与分析》实验报告**

实验名称 ： 实验7 回溯法 2

实验日期 ： 　 2025/5/6

姓 名 ： 　　　高心阳

学 号 ： 　 084623237

班 级 ： 　 计算机232

成 绩 ：

**人工智能与信息技术学院**

**南京中医药大学**

|  |
| --- |
| **实验目的：** |
| 1. 熟悉回溯法的实际应用 2. 理清优先队列的用途 |
| **实验内容和要求** |
| **1、旅行推销员问题**  **问题描述：**  旅行推销员问题（英语：Travelling salesman problem, TSP）是这样一个问题：给定一系列城市和每对城市之间的距离，求解访问每一座城市一次(选做：访问完再回到起始城市) 的最短回路。  它是组合优化中的一个NP难问题，在运筹学和理论计算机科学中非常重要。  **注意：**选择下一个城市时涉及到优先队列。  **输入**：城市之间存在的路径及长度  **输出**：最短路径长度，及经过的路径先后情况  **输入示例**：（内容如下）    **输出示例**：  最短路径长度为23,经过的路径为：1->4->3->5->2  **2、装载问题**  **问题描述：**  给定n个集装箱要装上一艘载重量为c的轮船，其中集装箱i的重量为wi。集装箱装载问题要求确定在不超过轮船载重量的前提下，将尽可能多的集装箱装上轮船。  由于集装箱问题是从n个集装箱里选择一部分集装箱，假设解向量为X(x1, x2, …, xn)，其中xi∈{0, 1}， xi =1表示集装箱i装上轮船， xi =0表示集装箱i不装上轮船。  **输入：**  每组测试数据：第1行有2个数C和n。C是轮船的载重量（0＜C＜30000），n是集装箱的个数（n≤20）。第2行有n个数w1, w2, …, wn，分别表示n个集装箱的重量。  **输出：**  对每个测试例，输出两行：第1行是装载到轮船的最大载重量，第2行是集装箱的编号。  **输入样例：**  34.1 3  21.1 10.1 5.3  **输出样例：**  31.2  1 2  \*3、**算24点**（选做）  **题目描述：**  几十年前全世界就流行一种数字游戏，至今仍有人乐此不疲．在中国我们把这种游戏称为“算24点”。您作为游戏者将得到4个1~9之间的自然数作为操作数，而您的任务是对这4个操作数进行适当的算术运算，要求运算结果等于24。  您可以使用的运算只有：+，-，\*，/，您还可以使用（）来改变运算顺序。注意：所有的中间结果须是整数，所以一些除法运算是不允许的（例如，(2\*2)/4是合法的，2\*(2/4)是不合法的）。下面我们给出一个游戏的具体例子：  若给出的4个操作数是：1、2、3、7，则一种可能的解答是1+2+3\*7=24。。  **输入描述:**  只有一行，四个1到9之间的自然数。  **输出描述:**  如果有解的话，只要输出一个解，输出的是三行数据，分别表示运算的步骤。其中第一行是输入的两个数和一个运算符和运算后的结果，第二行是第一行的结果和一个输入的数据、运算符、运算后的结果；第三行是第二行的结果和输入的一个数、运算符和“=24”。如果两个操作数有大小的话则先输出大的。  如果没有解则输出“No answer!”。  **输入样例**：1 2 3 7  **输出样例**：  2+1=3  7\*3=21  21+3=24  **【算法分析】**  计算24点主要应用四种运算．开始状态有四个操作数，一个运算符对应两个操作数，所以一开始选择两个操作数分别对四个操作符进行循环检测，每一次运算后产生了新的数，两个数运算变成一个数，整体是少了一个操作数，所以四个数最终是三次运算。由于操作的层数比较少(只有三层)，所以可以用回溯的算法来进行检测，当找到一个解时便结束查找。如果所有的情况都找过后仍然没有，则输出无解的信息。 |
| **运行结果（写清题号）** |
| |  | | --- | | Task1.cpp | | #include <iostream>  #include <vector>  #include <queue>  #include <limits>  #define endl '\n'  typedef long long ll;  struct cmp  {  bool operator()(const std::pair<int, int>& a, const std::pair<int, int>& b) {  return a.second > b.second;  }  };  const int N = 25;  int map[N][N];  int noden, edgen;  ll Min\_cost = INT\_MAX;  std::vector<int> best\_path;  void dfs(int cur\_node, ll cur\_cost, std::vector<int>& path, std::vector<bool>& visited, int start\_node) {  if (path.size() == noden) {  // Back  if (map[cur\_node][start\_node] != -1) {  ll total\_cost = cur\_cost + map[cur\_node][start\_node];  if (total\_cost < Min\_cost) {  Min\_cost = total\_cost;  best\_path = path;  }  }  // Not Back  // if (cur\_cost < Min\_cost) {  // Min\_cost = cur\_cost;  // best\_path = path;  // }  return;  }  if (cur\_cost >= Min\_cost)  return;  std::priority\_queue<std::pair<int, int>, std::vector<std::pair<int, int>>, cmp> q;  for (int i = 1; i <= noden; i++) {  if (!visited[i] && map[cur\_node][i] != -1) {  q.push(std::make\_pair(i, map[cur\_node][i]));  }  }  while (!q.empty()) {  std::pair<int, int> next\_pair = q.top();  q.pop();  int next\_node = next\_pair.first;  int edge\_cost = next\_pair.second;  if (!visited[next\_node]) {  visited[next\_node] = true;  path.push\_back(next\_node);  dfs(next\_node, cur\_cost + edge\_cost, path, visited, start\_node);  path.pop\_back();  visited[next\_node] = false;  }  }  }  void TSP() {  std::vector<int> path;  std::vector<bool> visited(noden + 1, false);  int start\_node = 1;  path.push\_back(start\_node);  visited[start\_node] = true;  dfs(start\_node, 0, path, visited, start\_node);  std::cout << Min\_cost << endl;  if (best\_path.size() != 0){  std::cout << start\_node;  for (int i = 1; i < best\_path.size(); i++)  std::cout << "->" << best\_path[i];  // std::cout << "->" << start\_node;  std::cout << endl;  }  }  void solve() {  std::cin >> noden >> edgen;  int from, to, dist;  for (int i = 0; i <= noden; i++)  for (int j = 0; j <= noden; j++)  map[i][j] = -1;  for (int i = 0; i < edgen; i++) {  std::cin >> from >> to >> dist;  map[from][to] = dist;  map[to][from] = dist;  }  TSP();  }  int main() {  std::ios::sync\_with\_stdio(0);  std::cin.tie(0);  int T;  T = 1;  // cin >> T;  while (T--) {  solve();  }  return 0;  } |  |  | | --- | | Task2.cpp | | #include <bits/stdc++.h>  #define endl '\n'  typedef long long ll;  double C, max\_w = -1;  ll n;  std::vector<double> v;  std::vector<int> best\_ans;  void dfs(double cur\_w, ll cur\_i, ll cur\_len, std::vector<int> &ans) {  if (cur\_i==n) {  if (cur\_w > max\_w) {  max\_w = cur\_w;  best\_ans = ans;  }  return;  }  for (int i = cur\_i + 1; i < n; i++) {  dfs(cur\_w, cur\_len, i, ans);  }  ans.push\_back(cur\_i);  for (int i = cur\_i + 1; i < n; i++) {  dfs(cur\_w + v[cur\_i], cur\_len + 1, i, ans);  }  ans.pop\_back();  }  void solve() {  std::cin >> C >> n;  v.resize(n);  for (int i = 0; i < n; i++)  std::cin >> v[i];  // sort(v.begin(), v.end());  std::vector<int> ans;  dfs(0, 0, 0, ans);  std::cout << max\_w << endl;  for (int i = 0; i < best\_ans.size(); i++)  std::cout << best\_ans[i]+1 << " ";  }  int main() {  std::ios::sync\_with\_stdio(0);  std::cin.tie(0);  int T;  T = 1;  while (T--) {  solve();  }  return 0;  } |  |  | | --- | | Task3.cpp | | #include <bits/stdc++.h>  using namespace std;  #define endl '\n'  typedef long long ll;  struct step {  int num1;  int num2;  char op;  int ans;  };  int max(int a, int b) {  return a > b ? a : b;  }  int min(int a, int b) {  return a < b ? a : b;  }  void show(step s) {  cout << max(s.num1, s.num2) << " " << s.op << " " << min(s.num1, s.num2) << " = " << s.ans << endl;  }  int Num[4];  vector<step> ans\_path;  bool exist;  void dfs(vector<int> num, vector<step> path) {  int size = num.size();  if (size == 1 || exist) {  if (num[0] == 24) {  ans\_path = path;  exist = true;  }  return;  }  for (int i = 0; i < size; i++) {  for (int j = 0; j < size; j++) {  if (i == j)continue;  vector<int> nnum;  for (int k = 0; k < size; k++) {  if (k != i && k != j) {  nnum.push\_back(num[k]);  }  }  step s;  s.num1 = num[i];  s.num2 = num[j];  s.op = '+';  s.ans = num[i] + num[j];  nnum.push\_back(s.ans);  path.push\_back(s);  dfs(nnum, path);  path.pop\_back();  nnum.pop\_back();  s.op = '-';  s.ans = num[i] - num[j];  nnum.push\_back(s.ans);  path.push\_back(s);  dfs(nnum, path);  path.pop\_back();  nnum.pop\_back();  s.op = '\*';  s.ans = num[i] \* num[j];  nnum.push\_back(s.ans);  path.push\_back(s);  dfs(nnum, path);  path.pop\_back();  nnum.pop\_back();  // Error Input:1 1 1 1  if (num[j] != 0 && num[i] % num[j] == 0) { // AC  // if (num[i] % num[j] == 0 && num[j] != 0) { // WA????  s.op = '/';  s.ans = num[i] / num[j];  nnum.push\_back(s.ans);  path.push\_back(s);  dfs(nnum, path);  path.pop\_back();  nnum.pop\_back();  }  }  }  }  void solve() {  vector<int> num(4);  for (int i = 0; i < 4; i++) {  cin >> num[i];  }  exist = false;  ans\_path.clear();  vector<step> path;  dfs(num, path);  if (exist) {  for (int i = 0; i < ans\_path.size(); i++)  show(ans\_path[i]);  } else {  cout << "No answer!\n";  }  }  int main() {  std::ios::sync\_with\_stdio(false);  std::cin.tie(nullptr);  int T;  T = 1;  while (T--) {  solve();  }  return 0;  } | |
| **实验的体会与建议** |
| 进一步学习回溯算法的编写，深切体会到了回溯算法的优异性。 |