**《算法设计与分析》实验报告**

实验名称 ： 实验6 回溯法 1

实验日期 ： 　 2025/4/22

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. 熟悉回溯法的概念 2. 掌握递回溯法的使用 |
| **实验内容和要求** |
| **1、N皇后问题**  **问题描述：**  在n×n格的棋盘上放置彼此不受攻击的n个皇后。  按照国际象棋的规则，皇后可以攻击与之处在同一行或同一列或同一斜线上的棋子。n皇后问题等价于在n×n格的棋盘上放置n个皇后，任何两个皇后不放在同一行或同一列或同一斜线上。  编程要求：找出一个n×n格的棋盘上放置n个皇后并使其不能互相攻击的所有方案。    **2、图的着色问题**  **问题描述：**  给定无向连通图G=(V, E)和m种不同的颜色，用这些颜色为图G的各顶点着色，每个顶点着一种颜色。是否有一种着色法使G中相邻的两个顶点有不同的颜色?  这个问题是图的m可着色判定问题。若一个图最少需要m种颜色才能使图中每条边连接的两个顶点着不同颜色，则称这个数m为该图的色数。求一个图的色数m的问题称为图的m可着色优化问题。  编程计算：给定图G=(V, E)和m种不同的颜色，找出所有不同的着色法和着色总数。  **输入：**  第一行是顶点的个数n（2≤n≤10），颜色数m（1≤m≤n）。  接下来是顶点之间的相互关系：a b  表示a和b相邻。当a，b同时为0时表示输入结束。  **输出：**  输出所有的着色方案，表示某个顶点涂某种颜色号，每个数字的后面有一个空格。最后一行是着色方案总数。    \*3、马拦过河卒（选做）  **题目描述：**  棋盘上A点有一个过河卒，需要走到目标B点。卒行走的规则：可以向下、或者向右。同时在棋盘上C点有一个对方的马，该马所在的点和所有跳跃一步可达的点称为对方马的控制点。因此称之为“马拦过河卒”。  棋盘用坐标表示，A点(0, 0)、B点(n, m)(n, m为不超过15的整数)，同样马的位置坐标是需要给出的。现在要求你计算出卒从A点能够到达B点的路径的条数，假设马的位置是固定不动的，并不是卒走一步马走一步。。  **输入:**  一行四个数据，分别表示B点坐标和马的坐标。  **输出：**  一个数据，表示所有的路径条数。  **输入样例**：6 6 3 3  **输出样例**：6  **【算法分析】**  从起点开始往下走(只有两个方向可以走)，如果某个方向可以走再继续下一步，直到终点，此时计数。最后输出所有的路径数。这种方法可以找出所有可能走法，如果要输出这些走法的话这种方法最合适了，但是本题只要求输出总的路径的条数，当棋盘比较大时，本程序执行会超时，此时最好能找出相应的递推公式更合适。 |
| **运行结果（写清题号）** |
| |  | | --- | | Task1.cpp | | #include <bits/stdc++.h>  #define endl '\n'  typedef long long ll;  std::vector<ll> loc;  std::vector<bool> col;  ll n, ans;  bool check() {  for (ll i = 1; i <= n; i++) {  for (ll j = 1; j <= n; j++) {  if (i == j)  continue;  if (abs(i - j) == abs(loc[i] - loc[j]))  return false;  }  }  return true;  }  void dfs(ll pos) {  if (pos > n && check()) {  for (ll i = 1; i <= n; i++)  std::cout << loc[i] << " ";  std::cout << endl;  ans++;  return;  }  for (ll i = 1; i <= n; i++) {  bool k = true;  for (ll j = 1; j < pos; j++)  if (abs(pos - j) == abs(i - loc[j])){  k = false;  break;  }  if (!col[i] && k) {  col[i] = true;  loc[pos] = i;  dfs(pos + 1);  col[i] = false;  }  }  }  void solve() {  std::cin >> n;  loc.assign(n + 1, 0);  col.assign(n + 1, false);  ans = 0;  dfs(1);  std::cout << "Total:" << ans << endl;  }  int main() {  std::ios::sync\_with\_stdio(0);  std::cin.tie(0);  int T;  T = 1;  while (T--) {  solve();  }  return 0;  } |  |  | | --- | | Task2.cpp | | #include <bits/stdc++.h>  #define endl '\n'  typedef long long ll;  ll n,m,a,b,ans\_count;  std::vector<std::vector<bool> > mp;  std::vector<ll> ans;  void dfs(ll pos){  if(pos>n){  for(ll i=1;i<=n;i++)  std::cout<<ans[i]<<" ";  std::cout<<"\n";  ans\_count++;  return;  }  for(ll i=1;i<=m;i++){  bool ok=true;  for(ll j=1;j<pos;j++)  if(mp[pos][j]&&ans[j]==i){  ok=false;  break;  }  if(!ok)  continue;  ans[pos]=i;  dfs(pos+1);  }  }  void solve() {  std::cin>>n>>m;  ans.assign(n+1,-1);  mp.assign(n+1,std::vector<bool>(n+1,false));  ans\_count=0;  while(std::cin>>a>>b){  if(a==0&&b==0)  break;  mp[a][b]=true;  mp[b][a]=true;  }  dfs(1);  std::cout << "Total:" << ans\_count << endl;  }  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>  #define endl '\n'  typedef long long ll;  std::vector<std::vector<ll> > mp;  ll bx, by, cx, cy, ans;  int xto[2] = {1, 0};  int yto[2] = {0, 1};  int clocx[9] = {0, 1, 2, 2, 1, -1, -2, -2, -1};  int clocy[9] = {0, -2, -1, 1, 2, 2, 1, -1, -2};  ll max(ll a, ll b) {  return a > b ? a : b;  }  void dfs(ll x,ll y){  if(mp[y][x]||(y>max(cy + 2, by)||x>max(cx + 2, bx)))  return;  if(x==bx&&y==by)  ans++;  for(int i=0;i<2;i++)  dfs(x+xto[i],y+yto[i]);  }  void solve() {  std::cin >> bx >> by >> cx >> cy;  ans = 0;  mp.assign(max(cy + 2, by)+2, std::vector<ll>(max(cx + 2, bx)+2, 0));  for(int i=0;i<9;i++)  mp[cy+clocy[i]][cx+clocx[i]]=1;    dfs(0,0);  std::cout<<ans<<endl;  }  int main() {  std::ios::sync\_with\_stdio(0);  std::cin.tie(0);  int T;  T = 1;  while (T--) {  solve();  }  return 0;  } | |
| **实验的体会与建议** |
| 学习了回溯算法，极大的提高了搜索效率。 |