天津大学

程序设计综合实践课程报告

搜索算法实验

学生姓名 陈秋澄

学院名称 智能与计算学部

专　　业 大类

学　　号 3022244290

1. 正方形

1.1 题目分析

先求边长，如果不能整除或者边数小于 4 都直接输出 no，其他情况使用 dfs 深度优先搜索，当所有的边都被遍历过而且每条边都被使用时输出 yes，所以这个就可以作为 dfs 的结束条件。在 dfs 内部，先设定一个数组，记录这条边是否被选择过。用 t 记录边长，当遍历到一条边时，先用 t 减去这条边的边长，如果 t 小于 0，就代表不合适，返回上一个递归，如果等于零，就代表正好，可以直接进行下一次的递归，而当 t 还大于零的时候，就需要继续减去下一个边长，以此类推，直至搜索结束。

1.2 题目代码

```
#include <iostream>
#include <bits/stdc++.h>
using namespace std;
int a[20];
int n=0,bc=0;
int hs(int now,int cnt){
    if(now==n||cnt ==n){  //如果已经把所有的边都遍历或者每条边都被使
//用

        if(now==n&&cnt ==n){//如果两个都符合
            return 1;
        }
        return 0;
    }
    int vis[20];
    memset(vis,0,sizeof(vis));
    int t=bc;
    for(int i=now+1;i<=n;i++){
        if(vis[i]==0){      //如果这条边没被用过，就用这条边
            vis[i]=1;       //代表这条边已经被使用
            t-=a[i];        //现在的 t 代表还需要的长度
            if(t==0){       //如果还需要的长度等于 0，代表已经构成了一条边
```

```
                cnt++;      //构成的边数加 1
                return hs(i,cnt);   //递归
            }
            else if(t<0){   //如果这条边的长度大于边长
                vis[i]=0;   //代表这条边重新恢复没被使用的状态
                return hs(i-1,cnt);  //递归 i-1 条边
            }
        }
    }
    return 1;
}
int main(){
    int T;
    cin>>T;
    while(T--){
        int count=0;
        memset(a,0,sizeof(a));
        cin>>n;
        for(int i=1;i<=n;i++){//why not from 0
            cin>>a[i];         //输入每条边的长度
            count+=a[i];
        }
        if(count<4){           //如果边数小于 4，不能形成正方形
            cout<<"no"<<endl;
            //break;
        }
        if(count%4!=0){        //如果 sum 不是 4 的倍数，不能形成正方形
            cout<<"no"<<endl;
            continue;
        }
        bc=count/4;
        sort(a+1,a+n+1);    //把所有的边按从小到大的顺序排序
        if(hs(0,0))         )//从 a[0]，使用的边数是零开始递归，如果返回值
//是 1

            cout<<"yes"<<endl;
        else
            cout<<"no"<<endl;
    }
}
```

2. prime circle

2.1 题目分析

本题需要建立一个判断素数的函数，之后使用 “dfs”，遍历 2 及其后的元素，如果遍历的位置小于等于 n，就代表可以继续进行，从 1 到 n 找元素（for 循环内），如果是素数就让 a[i]等于现在的 i，往后遍历，如果已经到 n，就计算 a[n]+a[1]是不是素数，是就输出。不是的话继续遍历，但是先把 pick[i]赋值为零，代表没遍历过这个位置。如此，循环往复，直至得到答案。

2.2 题目代码

```
#include <iostream>
#include <bits/stdc++.h>
using namespace std;
int a[20],vis[20];
int n=0;
int zhishu(int k){
    for(int j=2;j<k;j++){
        if(k%j==0)
            return 0;
    }
    return 1;
}
void dfs(int p){
    if(p>n){   //如果当前元素的序号大于 n
        if(zhishu(a[n]+a[1])){    //
            for(int i=1;i<n;i++){//如果最后一个元素加第一个元素是素数
                cout<<a[i]<<" ";
            }
            cout<<a[n]<<endl;
        }
    }
    else{
        for(int i=1;i<=n;i++){   //当前元素的序号小于等于 n
            if(zhishu(i+a[p-1])&&vis[i]==0){
                                    //如果 i 没遍历过且 i+a[p-1]是素数
                a[p]=i;             //a[p]等于 i（这样可以保证加起来都是素数）
```

```
                vis[i]=1;          //记录已经遍历过这个位置的元素

                dfs(p+1);          //递归下一个位置的元素
                vis[i]=0;           //如果 p+1 大于 n 之后就会执行这条语句，
//从而可以输出所有的情况
            }
        }
    }

}

int main(){

    int b=1;
    while(cin>>n){
        a[1]=1;                    //第一个数是 1
        vis[1]=1;                 //第一个元素已经遍历过
        cout<<"Case"<<" "<<b<<":"<<endl;
        dfs(2);                   //直接遍历第二个元素
        cout<<endl;
        b++;
    }
}
```

3. 棋盘问题

3.1 题目分析

题意即在 n*n 的图中，要求摆放 k 个棋子，而棋子只能摆放在‘#’的棋盘区域，求一共可以有几种摆放的方法。通过递归，循环即可求出结果。

3.2 题目代码

```
#include <bits/stdc++.h>
using namespace std;
int n,k;
int num;
int vis[20];
char qipan[20][20];
int dfs(int x,int y){
    if(y==k){          //棋子个数达到要求，方案数+1，返回
        num++;
        return 0;
    }
    for(int i=x;i<n;i++){
        for(int j=0;j<n;j++){
            if(!vis[j]&&qipan[i][j]=='#'){
                vis[j]=1;       //标记为 1
                dfs(i+1,y+1);
                vis[j]=0;       //清除标记
            }
        }
    }
    return 0;
}
int main(){
    while(cin>>n>>k){
        if(n==-1&&k==-1)
            break;
        memset(vis,0,sizeof(vis));
        memset(qipan,0,sizeof(qipan));
        for(int i=0;i<n;i++)
            cin>>qipan[i];      //输入棋盘
```

```
        num=0;
        dfs(0,0);
        cout<<num<<endl;
    }
}
```

4. 迷宫问题

4.1 题目分析

简而言之，通过函数、循环，遍历找到每一条路取最短路径即可。

4.2 题目代码

```
#include<bits/stdc++.h>
using namespace std;
#define INF 0x3fffffff
int mp[5][5];
int ans;
struct node{
    int x,y;
}lu[30],pa[30];                    //创建结构体，便于表示
bool vis[8][8];
const int dir[4][2]={-1,0,1,0,0,1,0-1};
void dfs(int x,int y,int deep){
    if(x==4&&y==4&&deep<ans){
        ans=deep;
        for(int i=0;i<deep;i++){
            pa[i].x=lu[i].x;
            pa[i].y=lu[i].y;
        }
    }
    for(int i=0;i<4;i++){
        int tx=x+dir[i][0];
        int ty=y+dir[i][1];
        if(tx<0||tx>4||ty<0||ty>4) continue;
        if(mp[tx][ty]==0&&!vis[tx][ty]){
            vis[tx][ty]=true;
            lu[deep].x=tx;
            lu[deep].y=ty;
            dfs(tx,ty,deep+1);
            vis[tx][ty]=false;       //记为 false
        }
    }
}
int main(){
    for(int i=0;i<5;i++){
```

```
        for(int j=0;j<5;j++){
            cin>>mp[i][j];
        }
    }
    memset(lu,0,sizeof(lu));
    memset(pa,0,sizeof(pa));
    memset(vis,0,sizeof(vis));      //进行数组初始化
    ans=INF;
    lu[0].x=lu[0].y=0;
    vis[0][0]=true;
    dfs(0,0,1);
    for(int i=0;i<ans;i++){
        cout<<'('<<pa[i].x<<','<<" "<<pa[i].y<<')'<<endl;
    }                               //输出结果
}
```

5. Find a way

5.1 题目分析

本题使用 2 次 bfs，从而得到两个人到 KFC 的最短距离，由于 kfc 不止 1 个，所以使用数组 step 记录到 i 行 j 列的 kfc 需要走的步数，最后计算出来之后，使用两个 for 循环在 ans 和两个 step 的和中取最小值，这样就可以得到最少的步数，最后输出 ans 即可。

5.2 题目代码

```
#include<bits/stdc++.h>
using namespace std;
const int maxn = 200;
const int INF = 0x3f3f3f3f;
char s[maxn][maxn];
int disx[] = {1,0,0,-1};
int disy[] = {0,1,-1,0};//代表右上下左四个走法
int n, m, vis[maxn][maxn], step[maxn][maxn][2];
void bfs(int x, int y, int flag)
{
    memset(vis, 0, sizeof(vis));//初始化数组
    queue<pair<int,int> >q;
    map<pair<int,int>,int>mp;
    q.push({x,y});//（x，y）进入队列
    vis[x][y] = 1;//记录 xy 已经走过
    mp[{x,y}] = 0;//
    while(!q.empty()) {//当队列不空的时候
    pair<int,int>t = q.front();//t 就等于队首元素
    q.pop();//弹出队首元素
    if(s[t.first][t.second] == '@')//如果队首元素是 KFC
    step[t.first][t.second][flag] = mp[{t.first,t.second}];//某
//个人到这的步数就是 mp[{t.first,t.second}]
    for(int i = 0; i < 4; i++) {//对于四个方向进行遍历
    int fx = t.first + disx[i];
    int fy = t.second + disy[i];//更新位置
    if(fx >= 0 && fx < n && fy >= 0 && fy < m && s[fx][fy] !=
'#' && !vis[fx][fy])
    {//如果没走出范围而且这个位置能走而且这个位置没走过
```

```
        vis[fx][fy] = 1;//走这个位置
        mp[{fx,fy}] = mp[{t.first,t.second}] + 1;//这个位置的步数
//++
        q.push({fx,fy});//现在的这个位置入队
 }
 }
 }
}
int main()
{
    while(cin >>n>>m)
    {
        int yx, yy, mx, my;//第一个人的坐标和第二个人的坐标
        for(int i = 0; i < n; i++)
        cin >>s[i];//输入路况
        for(int i = 0; i < n; i++) {
            for(int j = 0; j < m; j++) {
                if(s[i][j] == 'Y') {//找到 Y 的位置并且记录
                    yx = i;
                    yy = j;
                }
                else if(s[i][j]=='M') {//找到 M 的位置并且记录
                    mx = i;
                    my = j;
                }
            }
        }
        for (int i = 0; i < n; i++) {//把两个人到 ij 的时长初始化为极大值
            for (int j = 0; j < m; j++) {
                step[i][j][0] = INF;
                step[i][j][1] = INF;
            }
        }
 bfs(yx, yy, 0);//两次 bfs 分别计算到 KFC 的步数
 bfs(mx, my, 1);
 int ans = INF;
 for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
        if (s[i][j] == '@') {//如果是 KFC
        ans = min(ans, step[i][j][0] + step[i][j][1]);
         }//答案就等于 ans 和两个步数的和的最小值
    }
 }
 cout<<ans*11<<endl;//输出答案
```

```
 }
 return 0;
}
```

6. 马的遍历

6.1 题目分析

我们建立一张 m*n 的棋盘格，棋盘每格的值代表步数。初始值为-1，起始点作为源节点，步数为 0，从源结点作为起点开始 bfs。

6.2 题目代码

```
#include <bits/stdc++.h>
using namespace std;
queue<pair<int,int>>q;
int dx[8]={-1,-2,-2,-1,1,2,2,1};
int dy[8]={2,1,-1,-2,2,1,-1,-2};          //马走“日”，按顺序访问
int d[1001][1001];
bool vis[1001][1001];
int main(){
    int n,m,x,y;
    memset(d,-1,sizeof(d));
    memset(vis,false,sizeof(vis));
    cin>>n>>m>>x>>y;
    d[x][y]=0;
    vis[x][y]=true;                        //起点被访问
    q.push(make_pair(x,y));                //初始点入队
    while(!q.empty()){
        int x1=q.front().first;
        int y1=q.front().second;
        q.pop();                           //队头出队
        for(int i=0;i<8;i++){
            int s=x1+dx[i];
            int t=y1+dy[i];
            if(s<1||s>n||t<1||t>m||vis[s][t]){ //越界条件
                continue;
            }
            vis[s][t]=true;
            q.push(make_pair(s,t));            //可达点入队
            d[s][t]=d[x1][y1]+1;               //统计距离
        }
    }
    for(int j=1;j<=n;j++){
```

```
        for(int k=1;k<=m;k++){
            cout<<d[j][k]<<" ";                //按要求输出
        }
        cout<<endl;
    }
}
```

7. 求细胞数量

7.1 题目分析

经过分析，我们用一个 m*n 数组去记录区域有没有遍历过使用 bfs 去遍历所有连通细胞，以便求出细胞数量。

7.2 题目代码

```
#include <bits/stdc++.h>
using namespace std;
int dx[4]={0,1,0,-1};
int dy[4]={1,0,-1,0};
queue<pair<int,int> >q;
char juxing[100][100];              //字符矩阵
int xibao[100][100];                //标记矩阵
int m,n,cnt=0;
void bfs(int i,int j,int cnt)
{
    q.push({i,j});
    while(!q.empty())
    {
        int x1=q.front().first;
        int y1=q.front().second;
        q.pop();
        for(int i=0;i<4;i++)        //向四个方向尝试
        {
            int s=x1+dx[i],t=y1+dy[i];
            if(s<0||s>=n||t<0||t>=m)               //越界
            continue;
            if(xibao[s][t]>0||juxing[s][t]=='0')  //没访问过
            continue;
            xibao[s][t]=cnt;
            q.push({s,t});                        //入队
        }
    }
}
int main()
{
    cin >>n>>m;
```

```
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<m;j++)
        {
            cin >>juxing[i][j];
        }
    }                               //遇到没有标记的细胞,就以它为起点 bfs
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<m;j++)
        {
            if(xibao[i][j]==0&&juxing[i][j]!='0')
            {
                cnt++;
                bfs(i,j,cnt);
            }
        }
    }
    cout <<cnt<<endl;
    return 0;
}
```

8. 01 迷宫

8.1 题目分析

本题可用栈和队列相关做法解答，也可用数组地图实现。

这道题，我选择使用后者来解答。

通过标记当前位置在连通图中的坐标，保存当前连通图能移动到多少格，来逐次求解。

8.2 题目代码

```
#include<bits/stdc++.h>
using namespace std;
char _map[1001][1001];              //_map 数组保存地图
int flag[1001][1001],a[1000001];    //a 数组要开大一点，刚开始开 a[1001]错
//了 3 个点
//flag 数组保存各个点所在的连通图，以及是否已经处理过，a 数组保存各个连通图的
//大小
struct mg
{
    int x,y;
}q[1000001];
int main()
{
    int sx,sy,i,j,n,m,l,nx,ny,k,f,r,sum,d;
    int dx[4]={0,0,-1,1};
    int dy[4]={1,-1,0,0};           //四个方向
    scanf("%d %d",&n,&m);           //n 是正方形地图边长，m 是数据组数
    memset(a,0,sizeof(a));
    memset(flag,0,sizeof(flag));
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            cin>>_map[i][j];        //读入地图
    d=0;                            //d 用来保存当前是在第几个连通图中
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            if(flag[i][j]==0)   //如果当前位置不在已知连通图中（还未处理过）
            {
                d++;            //记录当前所在连通图数
```

```
                f=1;
                r=1;
                q[f].x=i;
                q[f].y=j;
                flag[i][j]=d;
                sum=1;          //初始化
                while(f<=r)
                {
                    for(k=0;k<4;k++)
                    {
                        nx=q[f].x+dx[k];
                        ny=q[f].y+dy[k];

if(flag[nx][ny]==0&&nx>=1&&nx<=n&&ny>=1&&ny<=n&&((_map[nx][ny]=='1'&&_
map[q[f].x][q[f].y]=='0')||(_map[nx][ny]=='0'&&_map[q[f].x][q[f].y]=='
1')))
                                    //如果新位置能走且在地图上
                        {
                            r++;
                            sum++;//计数器累加
                            flag[nx][ny]=d;//标记新位置在第 d 个连通图中
                            q[r].x=nx;
                            q[r].y=ny;     //更新位置
                        }
                    }
                    f++;
                }
                a[d]=sum;                  //保存当前连通图能移动到多少格
            }
    for(i=1;i<=m;i++)
    {
        cin>>sx>>sy;                      //读入询问
        cout<<a[flag[sx][sy]]<<endl;    //直接查找答案并输出
    }
    return 0;
}
```