目录

[BFS 1](#_Toc119173638)

[分析 1](#_Toc119173639)

[AC\_code 1](#_Toc119173640)

[DFS 9](#_Toc119173641)

这里只是为了展示一下搜索的几种写法

# BFS

[万圣节后的早晨](https://www.luogu.com.cn/problem/UVA1601)

### 分析

这题的关键在于奇妙的建图操作。

由**四个格子中至少有一个障碍物**和**格子联通和障碍物也各自联通**

因此，我们可以将空格单独提取出来，并且预处理出来所有的空格可以到的所有点。

这样就减少了状态空间，并且减少了每一次搜索的状态转移次数。

当然实现的方法很多，从最普通的BFS，到双向广搜，以及A\*，我会分别贴出来，并加上解析。

其中有一些细节。

写的时候是按照三个鬼写的，因此当小于3的时候，惊人的操作来了

if(n<=2) {deg[cnt]=1;G[cnt][0]=cnt;s[2]=ed[2]=cnt++;};  
if(n<=1) {deg[cnt]=1;G[cnt][0]=cnt;s[1]=ed[1]=cnt++;};

我们建立虚节点，使得不存在的鬼一直指向自己，这样就能永远合法了

### AC\_code

#### 普通BFS

**时间970ms**

#include<bits/stdc++.h>  
using namespace std;  
struct Node  
{  
 int a,b,c;  
};  
const int N = 200,INF = 0x3f3f3f3f;  
char g[20][20];//存图  
int G[N][5],dist[N][N][N],id[20][20];//G存的是每一个点能够到达的点的集合，dist是到达三个鬼当前状态的最短距离，id存的是每个图的编号  
int deg[N];//deg存的是每个点所能连接的点的数量  
int dx[5] = {0,0,1,0,-1},dy[5] = {0,1,0,-1,0};//方向数组  
int w,h,n,s[3],ed[3];//s存的是三个鬼的点开始的标号，ed存的是三个鬼的点结束的标号  
  
bool conflict(int a,int a2,int b,int b2)//判断是否发生冲突情况  
{  
 return (a2==b2)||(a==b2&&b==a2);//没发生要去同一个点或者交换位置的情况  
}  
  
int bfs()  
{  
 queue<Node> q;  
 q.push({s[0],s[1],s[2]});  
 memset(dist,-1,sizeof dist);  
 dist[s[0]][s[1]][s[2]] = 0;  
 while(q.size())  
 {  
 auto t = q.front();  
 q.pop();  
 int a = t.a,b = t.b,c = t.c;  
 if(a==ed[0]&&b==ed[1]&&c==ed[2]) break;  
 for(int i=0;i<deg[a];i++)  
 {  
 int a2 = G[a][i];  
 for(int j=0;j<deg[b];j++)  
 {  
 int b2 = G[b][j];  
 if(conflict(a,a2,b,b2)) continue;//判断是否a,b发生冲突  
 for(int k=0;k<deg[c];k++)  
 {  
 int c2 = G[c][k];  
 if(conflict(a,a2,c,c2)||conflict(b,b2,c,c2)) continue;//判断a,c与b,c是否冲突  
 if(dist[a2][b2][c2]==-1)  
 {  
 dist[a2][b2][c2] = dist[a][b][c] + 1;  
 q.push({a2,b2,c2});  
 }  
 }  
 }  
 }  
 }  
 return dist[ed[0]][ed[1]][ed[2]];  
}  
  
int main()  
{  
 while(scanf("%d%d%d", &w, &h, &n),n&&w&&h)  
 {  
 for(int i = 0; i < h; i++)  
 {  
 scanf(" ");  
 for(int j=0;j<w;j++)  
 g[i][j]=getchar();  
 }//读图  
 int cnt = 0;  
 int x[200],y[200];  
 for(int i=0;i<h;i++)  
 for(int j=0;j<w;j++)  
 if(g[i][j]!='#')  
 {  
 x[cnt] = i,y[cnt] = j,id[i][j] = cnt;  
 if(islower(g[i][j])) s[g[i][j]-'a'] = cnt;  
 else if(isupper(g[i][j])) ed[g[i][j]-'A'] = cnt;  
 cnt++;  
 }//将空格提取存起来  
   
 for(int i=0;i<cnt;i++)//预处理出来所有点能到的点的集合  
 {  
 deg[i] = 0;  
 for(int j=0;j<5;j++)  
 {  
 int nx = x[i] + dx[j],ny = y[i] + dy[j];  
 if(nx<0||nx>=h||ny<0||ny>=w) continue;  
 if(g[nx][ny]=='#') continue;  
 G[i][deg[i]++] = id[nx][ny];  
 }  
 }  
  
 if(n<=2) {deg[cnt]=1;G[cnt][0]=cnt;s[2]=ed[2]=cnt++;};//建立虚拟点  
 if(n<=1) {deg[cnt]=1;G[cnt][0]=cnt;s[1]=ed[1]=cnt++;};  
 printf("%d\n",bfs());  
 }  
 return 0;  
}

#### 双向BFS

**时间580ms**

#include<bits/stdc++.h>  
using namespace std;  
const int N = 210;  
struct Node  
{  
 int a,b,c;  
};  
char g[N][N];  
int G[N][5],dist[2][N][N][N];//dist建的是双向的数组  
int id[20][20],x[N],y[N],deg[N];  
int dx[5] = {0,0,1,0,-1},dy[5] = {0,1,0,-1,0};  
int w,h,n,s[3],t[3];  
  
bool conflict(int a,int a2,int b,int b2)  
{  
 return (a2==b2)||(b2==a&&b==a2);  
}  
  
int extend(queue<Node> &q,int f1,int f2)//每次进行拓展的时候就将当前队列中的所有元素进行拓展  
{  
 int len = q.size();  
 while(len--)  
 {  
 auto t = q.front();  
 q.pop();  
 int a = t.a,b = t.b,c = t.c;  
 for(int i=0;i<deg[a];i++)  
 {  
 int a2 = G[a][i];  
 for(int j=0;j<deg[b];j++)  
 {  
 int b2 = G[b][j];  
 if(conflict(a,a2,b,b2)) continue;  
 for(int k=0;k<deg[c];k++)  
 {  
 int c2 = G[c][k];  
 if(conflict(a,a2,c,c2)||conflict(b,b2,c,c2)) continue;  
 if(dist[f2][a2][b2][c2]!=-1) return dist[f2][a2][b2][c2]+dist[f1][a][b][c]+1;//若找到就直接返回  
 if(dist[f1][a2][b2][c2]!=-1) continue;  
 dist[f1][a2][b2][c2] = dist[f1][a][b][c] + 1;  
 q.push({a2,b2,c2});  
 }  
 }  
 }  
 }  
 return -1;  
}  
  
int bfs()  
{  
 queue<Node> q[2];  
 memset(dist,-1,sizeof dist);  
 dist[0][s[0]][s[1]][s[2]] = 0;  
 dist[1][t[0]][t[1]][t[2]] = 0;  
 q[0].push({s[0],s[1],s[2]}),q[1].push({t[0],t[1],t[2]});  
 while(q[0].size()&&q[1].size())  
 {  
 int len1 = q[0].size(),len2 = q[1].size();  
 int t;  
 if(len1<len2) t = extend(q[0],0,1);//每次拓展元素少的队列  
 else t = extend(q[1],1,0);  
 if(t!=-1) return t;  
 }  
 return -1;  
}  
  
int main()  
{  
 while(scanf("%d%d%d",&w,&h,&n),n||w||h)  
 {  
 for(int i=0;i<h;i++)  
 {  
 scanf(" ");  
 for(int j=0;j<w;j++) g[i][j]=getchar();  
 }  
 int cnt = 0;  
 for(int i=0;i<h;i++)  
 for(int j=0;j<w;j++)  
 if(g[i][j]!='#')  
 {  
 char c = g[i][j];  
 x[cnt] = i,y[cnt] = j,id[i][j]=cnt;  
 if(islower(c)) s[c-'a'] = cnt;  
 else if(isupper(c)) t[c-'A'] = cnt;  
 cnt++;  
 }   
 for(int i=0;i<cnt;i++)  
 {  
 deg[i] = 0;  
 for(int j=0;j<5;j++)  
 {  
 int nx = x[i] + dx[j],ny = y[i] + dy[j];  
 if(nx<0||nx>=h||ny<0||ny>=w) continue;  
 if(g[nx][ny]=='#') continue;  
 G[i][deg[i]++] = id[nx][ny];  
 }  
 }  
 if(n<=2)  
 {  
 deg[cnt]=1;  
 G[cnt][0]=cnt;  
 s[2] = t[2] = cnt++;  
 }  
 if(n<=1)  
 {  
 deg[cnt]=1;  
 G[cnt][0]=cnt;  
 s[1] = t[1] = cnt++;  
 }  
 printf("%d\n",bfs());  
 }  
 return 0;  
}

#### A\*

**时间260ms**

简单说一下A\*的思路

首先预处理出来，所有鬼到其目标点的最短路。

将当前状态插入到堆中时，dist[i]设置为now\_dist[i]+max(f[0] [i],f[1] [i],f[2] [i])

#include<bits/stdc++.h>  
using namespace std;  
const int N = 210;  
struct Node  
{  
 int a,b,c,dist;  
 bool operator<(const Node& W)const{  
 return dist>W.dist;  
 }  
};  
char g[20][20];  
int G[N][5],dist[N][N][N],f[3][N];  
bool st[N];  
int id[20][20],x[200],y[200],deg[N];  
int dx[5] = {0,0,1,0,-1},dy[5] = {0,1,0,-1,0};  
int w,h,n,s[3],ed[3];  
  
bool conflict(int a,int a2,int b,int b2)  
{  
 return (a2==b2)||(b2==a&&b==a2);  
}  
int H(int a,int b,int c)  
{  
 return max(f[0][a],max(f[1][b],f[2][c]));  
}  
  
int bfs()  
{  
 priority\_queue<Node> q;  
 q.push({s[0],s[1],s[2],H(s[0],s[1],s[2])});  
 memset(dist,0x3f,sizeof dist);  
 dist[s[0]][s[1]][s[2]] = 0;  
 while(q.size())  
 {  
 auto t = q.top();  
 q.pop();  
 int a = t.a,b = t.b,c = t.c;  
 // cout<<a<<" "<<b<<" "<<c<<endl;  
 if(a==ed[0]&&b==ed[1]&&c==ed[2]) return dist[a][b][c];  
 for(int i=0;i<deg[a];i++)  
 {  
 int a2 = G[a][i];  
 for(int j=0;j<deg[b];j++)  
 {  
 int b2 = G[b][j];  
 if(conflict(a,a2,b,b2)) continue;  
 for(int k=0;k<deg[c];k++)  
 {  
 int c2 = G[c][k];  
 if(conflict(a,a2,c,c2)||conflict(b,b2,c,c2)) continue;  
 if(dist[a2][b2][c2]>dist[a][b][c]+1)  
 {  
 dist[a2][b2][c2] = dist[a][b][c] + 1;  
 q.push({a2,b2,c2,dist[a2][b2][c2]+H(a2,b2,c2)});  
 }  
 }  
 }  
 }  
 }  
 return -1;  
}  
  
void dijkstra(int u)  
{  
 priority\_queue<pair<int,int>,vector<pair<int,int>>,greater<pair<int,int>>> q;  
 memset(f[u],0x3f,sizeof f[u]);  
 memset(st,0,sizeof st);  
 f[u][ed[u]]=0;  
 q.push({0,ed[u]});  
 while(q.size())  
 {  
 auto t = q.top();  
 q.pop();  
 int ver = t.second;  
 if(st[ver]) continue;  
 st[ver] = 1;  
 for(int i=0;i<deg[ver];i++)  
 {  
 int ver2 = G[ver][i];  
 if(f[u][ver2]>f[u][ver]+1)  
 {  
 f[u][ver2] = f[u][ver] + 1;  
 q.push({f[u][ver2],ver2});  
 }  
 }  
 }  
}  
  
int main()  
{  
 while(scanf("%d%d%d",&w,&h,&n),n||w||h)  
 {  
 for(int i=0;i<h;i++)  
 {  
 scanf(" ");  
 for(int j=0;j<w;j++) g[i][j]=getchar();  
 }  
 int cnt = 0;  
 for(int i=0;i<h;i++)  
 for(int j=0;j<w;j++)  
 if(g[i][j]!='#')  
 {  
 char c = g[i][j];  
 x[cnt] = i,y[cnt] = j,id[i][j]=cnt;  
 if(islower(c)) s[c-'a'] = cnt;  
 else if(isupper(c)) ed[c-'A'] = cnt;  
 cnt++;  
 }   
 for(int i=0;i<cnt;i++)  
 {  
 deg[i] = 0;  
 for(int j=0;j<5;j++)  
 {  
 int nx = x[i] + dx[j],ny = y[i] + dy[j];  
 if(nx<0||nx>=h||ny<0||ny>=w) continue;  
 if(g[nx][ny]=='#') continue;  
 G[i][deg[i]++] = id[nx][ny];  
 }  
 }  
 if(n<=2)  
 {  
 deg[cnt]=1;  
 G[cnt][0]=cnt;  
 s[2] = ed[2] = cnt++;  
 }  
 if(n<=1)  
 {  
 deg[cnt]=1;  
 G[cnt][0]=cnt;  
 s[1] = ed[1] = cnt++;  
 }  
 for(int i=0;i<3;i++)  
 dijkstra(i);  
 printf("%d\n",bfs());  
 }  
 return 0;  
}

双端队列广搜就不说了，适用于01的路径情况

记一道状态记录比较好玩的代码

倒水问题

#include<bits/stdc++.h>  
using namespace std;  
const int N = 210;  
struct Node  
{  
 int v[3],dist;  
 bool operator<(const Node& W)const  
 {  
 return dist > W.dist;  
 }  
};  
bool st[N][N];  
int ans[N],cap[3];  
  
void update\_ans(Node &u)  
{  
 for(int i=0;i<3;i++)  
 {  
 int d = u.v[i];  
 if(ans[d]<0||u.dist<ans[d]) ans[d] = u.dist;  
 }  
}  
  
void bfs(int a,int b,int c,int d)  
{  
 cap[0] = a,cap[1] = b,cap[2] = c;  
 memset(st,0,sizeof st);  
 memset(ans,-1,sizeof ans);  
 priority\_queue<Node> q;  
 Node s;  
 s.v[0] = 0,s.v[1] = 0,s.v[2] = c;  
 s.dist = 0;  
 q.push(s);  
 st[0][0] = 1;  
 while(q.size())  
 {  
 auto t = q.top();  
 q.pop();  
 update\_ans(t);  
 if(ans[d]>=0) break;  
 for(int i=0;i<3;i++)  
 for(int j=0;j<3;j++)  
 if(i!=j)  
 {  
 if(t.v[i]==0||t.v[j]==cap[j]) continue;  
 int res = min(cap[j],t.v[i]+t.v[j]) - t.v[j];  
 Node u;  
 memcpy(&u,&t,sizeof t);  
 u.dist = t.dist + res;  
 u.v[i] -= res;  
 u.v[j] += res;  
 if(!st[u.v[0]][u.v[1]])  
 {  
 st[u.v[0]][u.v[1]] = 1;  
 q.push(u);  
 }  
 }  
 }  
 while(d>=0)  
 {  
 if(ans[d]>=0)  
 {  
 printf("%d %d\n",ans[d],d);  
 return ;  
 }  
 d--;  
 }  
}  
  
int main()  
{  
 int T;scanf("%d",&T);  
 while(T--)  
 {  
 int a,b,c,d;scanf("%d%d%d%d",&a,&b,&c,&d);  
 bfs(a,b,c,d);  
 }  
 return 0;  
}

# DFS

对于所有的DFS都有的优化

1. 优化搜索顺序
2. 排除等效冗余
3. 可行性剪枝
4. 最优性剪枝

主要就是记一下迭代加深和IDA\*

快速幂计算

#include<bits/stdc++.h>  
using namespace std;  
const int N = 1010;  
int res[N];  
int n;  
  
bool dfs(int maxd,int d){  
 if(res[d] == n) return true;  
 if(d == maxd) return false;  
 int maxv = res[0];  
 for(int i = 1;i <= d;i++) maxv = max(maxv,res[i]);  
 if((maxv << (maxd - d)) < n) return false;  
   
 for(int i = d;i >= 0;i--){  
 res[d + 1] = res[d] + res[i];  
 if(dfs(maxd,d+1)) return true;  
 res[d + 1] = res[d] - res[i];  
 if(dfs(maxd,d+1)) return true;  
 }  
 return false;  
}  
  
int main()  
{  
 while(cin>>n,n){  
 if(n==1)   
 {  
 cout<<0<<endl;  
 continue;  
 }  
 int dep = 1;  
 res[0] = 1;  
 while(!dfs(dep,0)) dep++;  
 cout<<dep<<endl;  
 }  
 return 0;  
}

旋转问题

#include<bits/stdc++.h>  
using namespace std;  
  
int g[24];  
int op[8][7] =   
{  
 {0,2,6,11,15,20,22},  
 {1,3,8,12,17,21,23},  
 {10,9,8,7,6,5,4},  
 {19,18,17,16,15,14,13},  
 {23,21,17,12,8,3,1},  
 {22,20,15,11,6,2,0},  
 {13,14,15,16,17,18,19},  
 {4,5,6,7,8,9,10}  
};  
int cell[8] = {6,7,8,11,12,17,16,15};  
int con[8] = {5,4,7,6,1,0,3,2};  
string ans;  
int step = 0;  
  
int f()  
{  
 int cnt[4] = {0,0,0,0};  
 for(int i=0;i<8;i++)  
 cnt[g[cell[i]]]++;  
 return 8 - max(cnt[1],max(cnt[2],cnt[3]));  
}  
  
void work(int u)  
{  
 int t = g[op[u][0]];  
 for(int i=1;i<7;i++)  
 g[op[u][i-1]] = g[op[u][i]];  
 g[op[u][6]] = t;  
}  
  
bool dfs(int maxd,int u,string res,int pre)  
{  
 int t = f();  
 if(u+t>maxd) return 0;  
 if(!t)  
 {  
 if(u==0) ans = "No moves needed";  
 else if(step==0) ans = res;  
 step = g[cell[0]];  
 return 1;  
 }  
 for(int i=0;i<8;i++)  
 if(pre==-1||i!=con[pre])  
 {  
 work(i);  
 char c = i + 'A';  
 res.push\_back(c);  
 if(dfs(maxd,u+1,res,i)) return 1;  
 res.pop\_back();  
 work(con[i]);  
 }  
 return 0;  
}  
  
int main()  
{  
 while(cin>>g[0],g[0])   
 {  
 for(int i=1;i<24;i++) cin>>g[i];  
 int dep = 0;  
 step = 0;  
 string res;  
 while(!dfs(dep,0,res,-1)) dep++;  
 cout<<ans<<endl<<step<<endl;  
 }  
 return 0;  
}

不论是IDA \*还是A \*，如果求点到点之间最短变化过程的话，我们可以反向从终点求一下最短路去优化。

而对于像八数码这种，将整个棋局作为状态的，则可以看看到最终状态最少还要几步，做一下优化