题目要求：[26-P1596-Lake Counting S-题目要求.docx](26-P1596-Lake%20Counting%20S-题目要求.docx)

## DFS深搜算法

其实DFS就是一口气往一个方向搜索，然后遇到障碍之后再改一个方向搜索

优点：好写，不易出错，浅显易懂

缺点：往一个方向查找耗时，当寻找最优解时没有剪枝会卡时间

1. #include<cstdio>
2. using namespace std;
3. char a[101][101];
4. int ans;
5. int n,m;
6. void dfs(int x,int y){
7. a[x][y]='.';
8. int dx,dy;
9. for(int i=-1;i<=1;i++){
10. for(int j=-1;j<=1;j++){
11. dx=x+i;
12. dy=y+j;
13. if(dx>=0&&dx<=n&&dy>=0&&dy<m&&a[dx][dy]=='W'){
14. dfs(dx,dy);
15. }
16. }
17. }
18. return;
19. }
20. int main(){
21. scanf("%d%d",&n,&m);
22. for(int i=0;i<=n;i++){
23. scanf("%s",a[i]);*//避免换行带来问题这里直接读入字符串*
24. }
25. for(int i=0;i<=n;i++){
26. for(int j=0;j<m;j++){
27. if(a[i][j]=='W'){*//如果是W的话就直接开始遍历*
28. dfs(i,j);
29. ans++;*//水潭加一处*
30. }
31. }
32. }
33. printf("%d",ans);
34. return 0;
35. }

## BFS深搜算法

BFS就是维护一个队列，以一个点往四周搜索，如果符合条件的话就把它放进队列里

优点：同级优先搜索，在求最优解的时候可以避免许多无用的搜索，提高效率、可以避免递归

缺点：不好写，易出问题，用stl写队列很慢

值得一提的是，加上快读等优化以后已经做到stl队列能运行时间的极致了，然而提交就算开O2优化还是会有两个测试点被T

这就需要我们手写队列了以及各种优化了，但是这个代码适合bfs的初学者理解

1. #include<cstdio>
2. #include<queue>
3. using namespace std;
4. queue<int>hori;*//行的队列*
5. queue<int>para;*//列的队列*
6. int n,m;
7. int ans;
8. char s[101][101];
9. inline int read(){*//读入优化，可以加快数字的输入*
10. char p=0;int r=0,o=0;
11. for(;p<'0'||p>'9';o|=p=='-',p=getchar());
12. for(;p>='0'&&p<='9';r=(r<<1)+(r<<3)+(p^48),p=getchar());
13. return o?(~r)+1:r;
14. }
15. inline void bfs(int x,int y){*//不用递归时可以加inline，提高1ms的运行速度*
16. s[x][y]='.';
17. int dx,dy;
18. for(int i=-1;i<=1;i++){
19. for(int j=-1;j<=1;j++){
20. dx=x+i;
21. dy=y+j;
22. if(dx>=0&&dx<n&&dy>=0&&dy<m&&s[dx][dy]=='W'){
23. hori.push(dx);
24. para.push(dy);
25. }
26. }
27. }
28. }
29. int main(){
30. n=read();m=read();*//看不懂的话可以把这一行改成cin或scanf*
31. for(int i=0;i<n;i++){
32. scanf("%s",s[i]);
33. }
34. for(int i=0;i<n;i++){
35. for(int j=0;j<m;j++){
36. if(s[i][j]=='W'){
37. hori.push(i);
38. para.push(j);
39. while(!hori.empty()){*//如果队列不为空*
40. bfs(hori.front(),para.front());*//广搜队列前面的元素*
41. hori.pop();para.pop();*//弹出元素*
42. }
43. ans++;
44. }
45. }
46. }
47. printf("%d",ans);
48. return 0;
49. }

## 我的代码：

附上文件链接：[26-P1596-Lake Counting S.cpp](26-P1596-Lake%20Counting%20S.cpp)

1. #include<stdio.h>
2. #include<string.h>
3. int n,m,ans=0,check[105][105];
4. char area[105][105];*//记录地图*
5. int dirx[]={0,1,1, 1, 0,-1,-1,-1};
6. int diry[]={1,1,0,-1,-1,-1, 0, 1};
7. *//深搜算法的八个方向*
8. void sign(void);
9. void dfs(int x,int y);
10. int main(void)
11. {
12. scanf("%d%d",&n,&m);
13. for(int i=0;i<n;i++)scanf("%s",&area[i]);
14. sign();*//将地图转化为易于处理的整形数组*
15. for(int i=0;i<n;i++)
16. {
17. for(int j=0;j<m;j++)
18. {
19. if(check[i][j]==1)*//如果是水洼则结果加1并进行搜索*
20. {
21. ans++;
22. dfs(i,j);
23. }
24. }
25. }
26. printf("%d",ans);
27. }
28. void sign()
29. {
30. for(int i=0;i<n;i++)
31. {
32. for(int j=0;j<m;j++)
33. {
34. if(area[i][j]=='W') check[i][j]=1;*//是水洼则标记为1*
35. else check[i][j]=0;
36. }
37. }
38. }
39. void dfs(int x,int y)
40. {
41. check[x][y]=0;*//深搜时先把当前水洼点赋值为0表示已走过*
42. for(int i=0;i<8;i++)
43. if(check[x+dirx[i]][y+diry[i]]==1)
44. *//循环遍历八个反向，如果是水洼则对该水洼继续向前搜索*
45. dfs(x+dirx[i],y+diry[i]);
46. *//递归DFS搜索*
47. }

## 连通块问题

用到dfs的一般都是求全部解和连通块问题。搜索算法的可变性非常大（比如下面这个程序就是搜索，只是长得不太像而已）

这题乍一看是道求联通块的模板题，那么联通块各位大佬都有各种神奇的办法维护但是

我们不讲联通块，我们讲强连通分量！！！（不连边的tarjan呦）

观察题面，嗯不错这是道网格题的图，那么我们考虑用二维tarjan求强连通 分量，

我就不讲tarjan了，这是一个套模板的题，我们在【模板】缩点 的代码上加一维，具体其他和模板一样，套个代码观察观察：

1. #include<cstdio>
2. #include<iostream>
3. using namespace std;
4. int n,m,top=0,col=0,x=0,ans=0;*//ans 没蛋用；*
5. *//直接输出col（强连通分量个数）即可*
6. const int N=105;
7. struct node{
8. int x,y;
9. }stack[N\*N];
10. int low[N][N],dfn[N][N];*//tarjan标配*
11. char map[N][N];
12. int co[N][N];*//co记改点属于哪个强连通分量*
13. int dx[]={-1,-1,-1,0,0,1,1,1},dy[]={-1,0,1,-1,1,-1,0,1};
14. void tarjan(int ux,int uy){
15. if(map[ux][uy]=='.')return;*//防止对.进行tarjan*
16. dfn[ux][uy]=low[ux][uy]=++x;
17. top++;
18. stack[top].x=ux;
19. stack[top].y=uy;
20. for(int i=0;i<8;++i){
21. int vx=ux+dx[i];int vy=uy+dy[i];
22. if(vx<1 || vx>n || vy<1 || vy>m)continue;*//防止超边界*
23. if(map[vx][vy]=='.')continue;
24. if(!dfn[vx][vy]){
25. tarjan(vx,vy);
26. low[ux][uy]=min(low[ux][uy],low[vx][vy]);
27. }
28. else if(!co[vx][vy])
29. low[ux][uy]=min(low[ux][uy],dfn[vx][vy]);
30. }
31. if(dfn[ux][uy]==low[ux][uy]){
32. co[ux][uy]=++col;
33. map[ux][uy]='.';
34. while(stack[top].x!=ux || stack[top].y!=uy){
35. co[stack[top].x][stack[top].y]=col;
36. map[stack[top].x][stack[top].y]='.';
37. top--;
38. }
39. top--;
40. }
41. }
42. int main(){
43. std::ios::sync\_with\_stdio(false);*//关闭同步*
44. cin>>n>>m;
45. for(int i=1;i<=n;++i)for(int j=1;j<=m;++j){
46. cin>>map[i][j];
47. }
49. for(int i=1;i<=n;++i)for(int j=1;j<=m;++j)
50. if(!dfn[i][j])tarjan(i,j);
51. ans=col;
52. cout<<ans<<endl;
53. }

## 暴推+并查集

我们发现，所有的水坑都是连在一起的，八个方向相连。那么，这非常有利于我们直接暴推写并查集，我们从左往右自上而下地推，因为与八个方向相连，所以我们只需要判定四个方向是否为一个水坑，要求这四个方向所在直线互不重合

1. #include<iostream>
2. #include<cstdio>
3. using namespace std;
4. const int maxn=105;
5. int n,m,f[10005],ans;*//f是并查集数组（并为一维），ans是水坑个数*
6. char p[maxn][maxn];
7. int fa(int x){*//简单的寻找祖先，又名找爸爸*
8. if(f[x]==x)return x;
9. return fa(f[x]);
10. }
11. void hebing(int x,int y){*//合并祖先*
12. int fx=fa(x),fy=fa(y);
13. if(fx==x){
14. f[x]=fy;
15. return;
16. }
17. f[fy]=fx;
18. }
19. int main(){
20. scanf("%d%d",&n,&m);
21. for(int i=1;i<=n;++i){
22. getchar();*//读掉换行符*
23. for(int j=1;j<=m;++j){
24. scanf("%c",&p[i][j]);
25. f[(i-1)\*m+j]=(i-1)\*m+j;
26. }
27. }
28. for(int i=1;i<=n;++i)
29. for(int j=1;j<=m;++j){
30. if(p[i][j]=='W'){*//如果本身有积水*
31. int pp=(i-1)\*m+j;
32. if(p[i][j+1]=='W')hebing(pp,pp+1);*//正右方向*
33. if(p[i+1][j-1]=='W')hebing(pp,pp+m-1);*//左下方向*
34. if(p[i+1][j]=='W')hebing(pp,pp+m);*//正下方*
35. if(p[i+1][j+1]=='W')hebing(pp,pp+m+1);*//右下方向*
36. }
37. }
38. for(int i=1;i<=n;++i)
39. for(int j=1;j<=m;++j)
40. if(p[i][j]=='W')*//如果本身有积水且祖先为自己本身，则视为一个新的水坑*
41. ans+=(f[(i-1)\*m+j]==(i-1)\*m+j);
42. printf("%d",ans);
43. return 0;
44. }

## 代码优化

1.深搜算法的函数中，在判断是否进行下一步递归搜索时要加上判断是否出界的条件