搜索

搜索算法的原理就是**枚举**。利用计算机的高性能，给出人类制定好的规则，枚举出所有可行的情况，找到可行解或者最优解。

比较常见的搜索算法是 **深度优先搜索（又叫深度优先遍历）** 和 **广度优先搜索（又叫广度、宽度优先遍历）**。各种图论的算法基本都是依靠这两者进行展开的。

深度优先搜索一般用来求**可行解**，利用剪枝进行优化，在树形结构的图上用处较多；而广度优先搜索一般用来求**最优解**，配合哈希表进行状态空间的标记，从而避免重复状态的计算。

迷宫问题图

	起点 (1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)	(1,8)	
	(2,1)	(2，2)						(2,8)	
	(3,1)	(3,2)		(3,4)	(3,5)	(3,6)		(3,8)	
	(4,1)	(4,2)		(4,4)		(4,6)		(4,8)	
	(5,1)	(5,2)		(5,4)		(5,6)		(5,8)	
	(6,1)	(6,2)		(6,4)		(6,6)		(6,8)	
	(7,1)			(7,4)		(7,6)		(7,8)	
	(8,1)	(8,2)	(8,3)	(8,4)		(8,6)		终点	

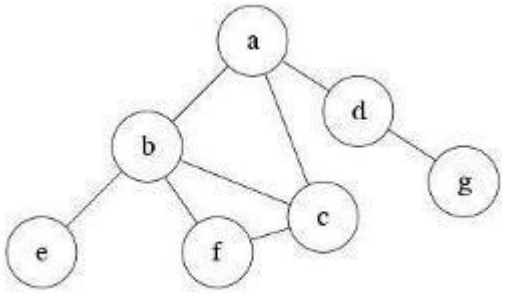


Figure 1: depth-first search 1

深度优先搜索：

a-->b-->e-->b-->f-->c-->f-->b-->c-->b-->a-->c-->a-->d-->g-->d-->a

广度优先搜索：

a-->b-->d-->e-->f-->c-->g

深度优先搜索（dfs）

深度优先搜索（Depth First Search），是图遍历算法的一种。用一句话概括就是：“**一直往下走，走不通回头，换条路再走，直到无路可走**”。

具体算法描述为：

选择一个起始点 u 作为当前结点，执行如下操作：

a. 访问当前结点，并且标记该结点已被访问，然后跳转到 b；

b. 如果存在一个和当前结点相邻并且尚未被访问的结点 v，则将 v 设为当前结点，继续执行 a；

c. 如果不存在这样的 v，则进行回溯，回溯的过程就是回退当前结点。

上述所说的**当前结点**需要用一个**栈**来维护，每次访问到的结点入栈，回溯的时候出栈。除了栈，另一种实现深度优先搜索的方式是**递归**，代码更加简单，相对好理解。

例题 1：迷宫问题

题目链接：https://vjudge.csgrandeur.cn/problem/OpenJ_Bailian-2790

```
#include <iostream>
#include <cstring>
#include <algorithm>
#define MAXN 110
using namespace std;

int t,n,vis[MAXN][MAXN];
int sx,sy,ex,ey;
char g[MAXN][MAXN];
int dir[4][2]={{-1,0},{0,1},{1,0},{0,-1}};//移动方向-->上右下左
/*
//也可以写成该种(只有上下左右)
int dx[4]={-1,0,1,0};
int dy[4]={0,1,0,-1};
//一个点周围一圈（左上、上、右上、右、右下、下、左下、左）
int dx[8] = {-1, -1, -1, 0, 1, 1, 1, 0};
int dy[8] = {-1, 0, 1, 1, 1, 0, -1, -1};
*/
int dfs(int x,int y){
    if(g[x][y]=='#')//当前点的位置不能通行
        return 0;
    if(x==ex&&y==ey)//该点到达终点位置坐标，返回1
        return 1;
    vis[x][y]=1;//每次标记当前点走过
```

```
    for(int i=0;i<=3;i++){
        int dx=x+dir[i][0],dy=y+dir[i][1];//偏移位置坐标
        if(dx>=0&&dx<=n-1&&dy>=0&&dy<=n-1&&vis[dx][dy]==0)//判断当前位置合法且是否被访问过
            if(dfs(dx,dy))//能借助偏移位置到达终点，递归返回1
                return 1;
    }
    return 0;//所有情况都走过仍然无法到达终点，返回0
}

int main(){
    cin >> t;
    while(t--){
        scanf("%d\n",&n);
        memset(vis,0,sizeof(vis));
        for(int i=0;i<=n-1;i++)
            scanf("%s",g[i]);
        cin >> sx >> sy >> ex >> ey;
        if(dfs(sx,sy)==1)
            printf("YES\n");
        else printf("NO\n");
    }
    return 0;
}
```

代码链接：https://paste.org.cn/VkgufAWbKa

例题 2：n-皇后问题

题目链接：https://www.acwing.com/problem/content/845/

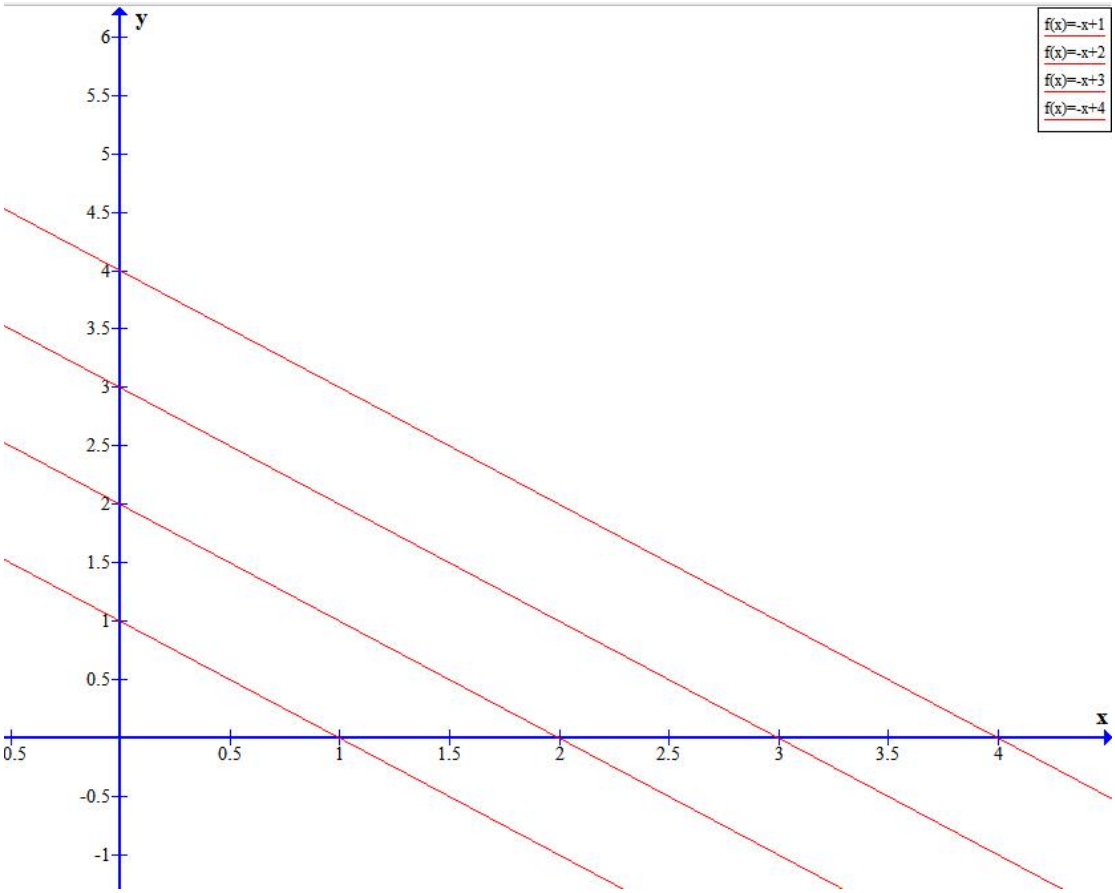
题目分析：

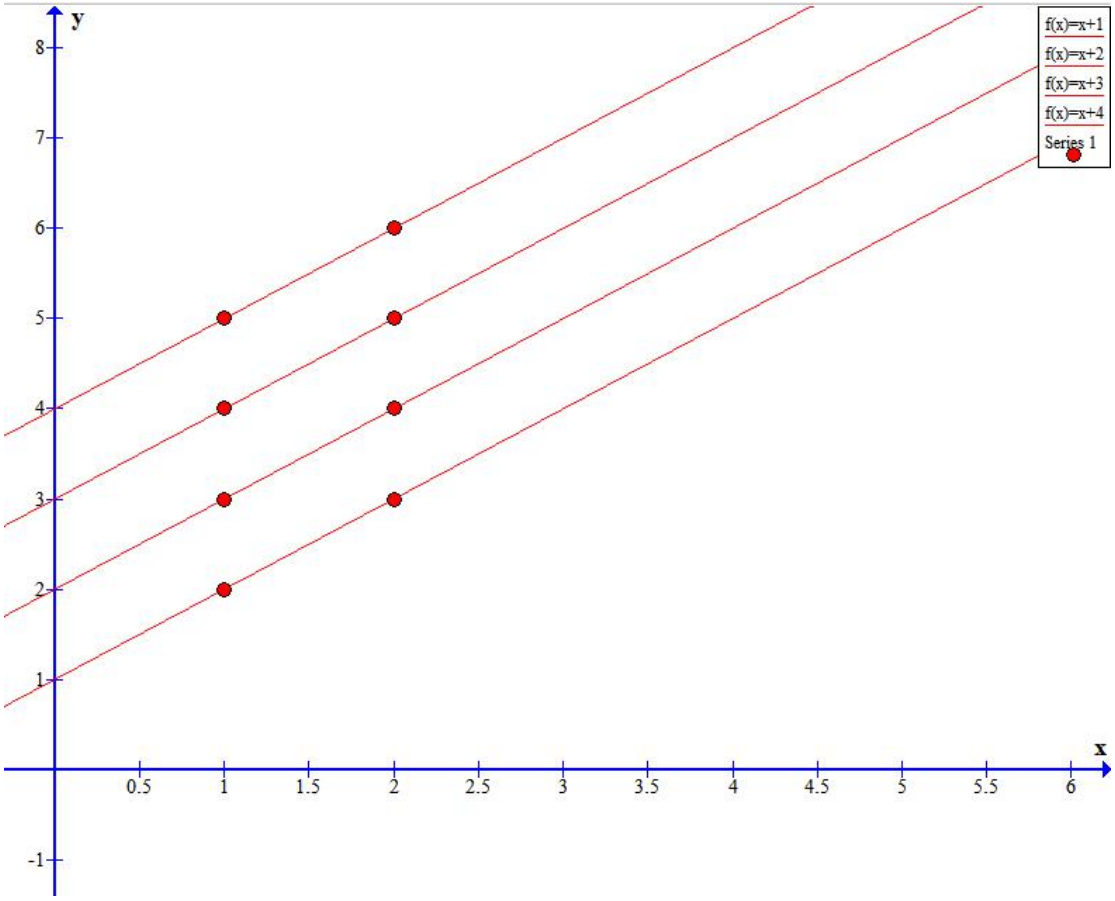
方法 1： dfs 按行枚举，时间复杂度为 $O(n!)$

列 i 可用 col[i]容易处理标记，此处解决怎么表示对角线和反对角线标记问题。
用 dg[]数组存储对角线标记，用 udg[]数组存储反对角线标记。

正对角线 $y=-x+b$，截距 $b=y+x$，故对角线用 dg[x+y]表示即可；

反对角线 $y=x+b$，截距为 $b=y-x$，由于把 b 当做数组下标使用，b 不能为负数，为保证结果为正，故反对角线用 udg[y-x+n]表示即可。


y
x
f(x)=-x+1
f(x)=-x+2
f(x)=-x+3
f(x)=-x+4


y
x
f(x)=x+1
f(x)=x+2
f(x)=x+3
f(x)=x+4
Series 1

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>
#define MAXN 10
using namespace std;

int n;
char g[MAXN][MAXN];
int col[MAXN],dg[2*MAXN],udg[2*MAXN];//注意对角线和反对角线是 2 倍的空间

void dfs(int x){
    if(x==n){//表示已搜完 n 行,输出结果
        for(int i=0;i<=n-1;i++)
            printf("%s\n",g[i]);
        printf("\n");
        return ;
    }
    for(int y=0;y<=n-1;y++)
        if(col[y]==0&&dg[y+x]==0&&udg[y-x+n]==0)//当前列和对角线和反对角线都未访问
        {
            g[x][y]='Q';
            col[y]=dg[y+x]=udg[y-x+n]=1;
            dfs(x+1);
            col[y]=dg[y+x]=udg[y-x+n]=0;//恢复现场
            g[x][y]='.';
        }
}

int main(){
    cin >> n;
    for(int i=0;i<=n-1;i++)
        for(int j=0;j<=n-1;j++)
            g[i][j]='.';
    dfs(0);
    return 0;
}
```

代码链接：https://paste.org.cn/7tYLRo4asW

方法 2：dfs 按每个元素枚举，时间复杂度为 $O(2^{n^2})$

每个位置都有两种情况，总共有 n^2 个位置。

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>
#define MAXN 10
using namespace std;

int n;
char g[MAXN][MAXN];
int row[MAXN],col[MAXN],dg[2*MAXN],udg[2*MAXN];//注意对角线和反对角线是 2 倍的空间

void dfs(int x,int y,int cnt){//cnt 表示放上皇后的个数
    if(y==n)
        y=0,x++;
    if(x==n){
        if(cnt==n){
            for(int i=0;i<=n-1;i++)
                printf("%s\n",g[i]);
            printf("\n");
        }
        return ;
    }

    //放皇后(当前位置)
    if(row[x]==0&&col[y]==0&&dg[x+y]==0&&udg[y-x+n]==0){
        g[x][y]='Q';
        row[x]=col[y]=dg[x+y]=udg[y-x+n]=1;
        dfs(x,y+1,cnt+1);
        row[x]=col[y]=dg[x+y]=udg[y-x+n]=0;//恢复现场
        g[x][y]='.';
    }

    //不放皇后（当前位置）
    dfs(x,y+1,cnt);
}

int main(){
    cin >> n;
    for(int i=0;i<=n-1;i++)
        for(int j=0;j<=n-1;j++)
            g[i][j]='.';
    dfs(0,0,0);
    return 0;
}
```

代码链接：https://paste.org.cn/cXk1kdeKAC

扩展:2n-皇后问题 此题可自行尝试解决
题目链接：https://www.dotcpp.com/oj/problem1460.html

广度优先搜索（bfs）

广度优先搜索即 Breadth First Search，也是图遍历算法的一种。用一句话概括就是：“**我会分身我怕谁？！**”。

BFS 的具体算法描述为：

选择一个起始点 u 放入一个先进先出的**队列**中，执行如下操作：

a. 如果队列不为空，弹出一个队列首元素，记为当前结点，执行 b；否则算法结束；

b. 将与当前结点相邻并且尚未被访问的结点的信息进行更新，并且全部放入队列中，继续执行 a；

维护广搜的数据结构是队列和哈希表，哈希表（一般用数组标记实现）主要是用来标记状态的。

例题：迷宫最短路问题

题目链接：https://www.dotcpp.com/oj/problem1672.html

```
#include <iostream>
#include <cstring>
#include <algorithm>
#include <queue>
#define MAXN 110
using namespace std;

int t;
int n,m,sx,sy,ex,ey,vis[MAXN][MAXN];
char g[MAXN][MAXN];
int dir[4][2]={{-1,0},{0,1},{1,0},{0,-1}};
struct node{
    int x;
    int y;
    int step;
};

int bfs(){
    queue<node> q;
    q.push({sx,sy,0});
    vis[sx][sy]=1;

    while(q.size()>0){
        struct node temp=q.front();
        q.pop();
```

```
        if(temp.x==ex&&temp.y==ey)
            return temp.step;
        for(int i=0;i<=3;i++){
            int dx=temp.x+dir[i][0],dy=temp.y+dir[i][1];
            if(dx>=0&&dx<=n-1&&dy>=0&&dy<=m-1&&(g[dx][dy]=='-'||g[dx][dy]=='E')&
&vis[dx][dy]==0){
                q.push({dx,dy,temp.step+1});
                vis[dx][dy]=1;
            }
        }
    }
    return -1;
}

int main(){
    cin >> t;
    while(t--){
        memset(vis,0,sizeof(vis));
        cin >> n >> m;
        for(int i=0;i<=n-1;i++){
            cin >> g[i];
            for(int j=0;j<=m-1;j++){
                if(g[i][j]=='S')
                    sx=i,sy=j;
                else if(g[i][j]=='E')
                    ex=i,ey=j;
            }
        }
        int res=bfs();
        printf("%d\n",res);
    }
    return 0;
}
```

代码链接：https://paste.org.cn/hZVepfxYjF