搜索

- 通过不断地尝试寻找答案的一种算法,通过穷尽所有的可能来找到最优解
- 基础的搜索有朴素暴力，深搜，广搜，本次只提及这些
- 至于迭代加深，双向广搜，A*等可以之后再学

DFS

在搜索算法中，通常指利用**递归**实现暴力的枚举算法

在写DFS的时候最好把**退出的条件写在最前面**

- ### $Eg1$: 输出 $1-n$ 的全排列

https://ac.nowcoder.com/acm/contest/23156/1001

思路:每一层枚举选择一个未选择的数，然后再递归做相同的事，时间复杂度**O(n!)**

```
#include<bits/stdc++.h>
using namespace std;
constexpr int n = 10;

int vis[100],a[100];
void print(){
    for(int i = 1;i<=n;++i){
        printf("%d ",a[i]);
    }
    printf("\n");
}

void dfs(int i){
    for(int j = 1;j<=n;++j){//1-n的枚举
        if(vis[j]) continue;//如果遍历过了就continue
        vis[j] = 1;//设置标记，在往下dfs的时候正确处理
        a[i] = j;
        if(i==n){
            print();
            return;
        }else{
            dfs(i+1);
        }
        vis[j] = 0;//回溯操作，给explanation
    }
}
signed main(){
    dfs(1);
    return 0;
}
```

- $Eg2$：**求n个里面选k个的异或和的最大值**

https://atcoder.jp/contests/abc386/tasks/abc386_e

思路:

上一题的思路可不可以做?

在数据小的时候可以。

朴素暴力，每一个尝试选或不选，时间复杂度$\binom{n}{k}$，此外还可以考虑一些小的优化，如果剩下的个数等于我还需要选的数字个数，那么可以直接全部选上，维护一个后缀异或和即可

```
#include<bits/stdc++.h>
using namespace std;
using i64 = long long;
using i128 = __int128;

constexpr int maxn = 2e5+10;
i64 n,K,suf[maxn],a[maxn],ans;
void dfs(int i,int k,i64 x){
    if(k==0){
        ans =max(ans,x);
        return;
    }
    if(i+k-1==n){
        ans = max(ans,x^suf[i]);
        return;
    }
    dfs(i+1,k,x);
    dfs(i+1,k-1,x^a[i]);
}
signed main(){
    ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    cin>>n>>K;
    for(int i = 1;i<=n;++i) cin>>a[i];
    for(int i = n;i>=1;--i) suf[i] = suf[i+1]^a[i];
    dfs(1,K,0ll);
    cout<<ans<<"\n";
    return 0;
}
```

BFS

主要的特点，用队列解决问题，每次用队首转移状态，把新状态加入队尾，直至队列为空

- ## $Eg1$：迷宫问题

链接：https://ac.nowcoder.com/acm/contest/23156/1014

给你一个n*m的迷宫，这个迷宫中有以下几个标识：

s代表起点

t代表终点

x代表障碍物

.代表空地

现在知了想知道能不能从起点走到终点不碰到障碍物（只能上下左右进行移动，并且不能移动到已经移动过的点）。

思路:从队首开始，往四周开始尝试，知道找到终点或者走过所有能走过的点

```
#include<bits/stdc++.h>
using namespace std;
using i64 = long long;
using i128 = __int128;

constexpr int maxn = 500+10;
int n,m;
char mp[maxn][maxn];
bool vis[maxn][maxn];
int dx[4]={0,0,-1,1},dy[4] = {1,-1,0,0};

bool bfs(array<int,2> &s,array<int,2> &t){
    queue<array<int,2>> q;
    q.push(s);
    vis[s[0]][s[1]]=1;
    while(!q.empty()){
        auto tmp = q.front();
        q.pop();
        if(tmp==t) return 1;
        for(int i = 0;i<4;++i){
            int tx = tmp[0]+dx[i],ty = tmp[1]+dy[i];
            if(vis[tx][ty]||tx<1||tx>n||ty<1||ty>m||mp[tx][ty]=='x') continue;
            q.push({tx,ty});
            vis[tx][ty]=1;
        }
    }
    return 0;
}
void solve(){
    cin>>n>>m;
    array<int,2> st,ed;
    for(int i = 1;i<=n;++i){
        for(int j = 1;j<=m;++j){
            vis[i][j]=0;
            cin>>mp[i][j];
            if(mp[i][j]=='s') st={i,j};
            if(mp[i][j]=='t') ed={i,j};
        }
    }
    cout<<(bfs(st,ed)?"YES":"NO")<<"\n";
}

signed main(){
    ios::sync_with_stdio(0);cin.tie(0);
    int t = 1;
    cin>>t;
    while(t--){
        solve();
    }
    return 0;
}
```

当然本题也可以用DFS解决，大家自己尝试解决一下

尝试了不会可以参考以下程序

https://ac.nowcoder.com/acm/contest/view-submission?submissionId=74424974

- ## $Eg3$：迷宫问题II

https://atcoder.jp/contests/abc387/tasks/abc387_d

for practice

问题陈述

给定一个有 H 行和 W 列的网格。让 (i,j) 表示从上往下第 i 行和从左往上第 j 列的单元格。

每个单元格都是以下其中之一：起始单元格、目标单元格、空单元格或障碍单元格。这些信息由长度为 W 的 H 字符串$S_1,S_2....,S_h$描述。具体来说，如果 S_i 的 j -th 字符是 "S"，则 (i,j) 单元是起始单元；如果是 "G"，则是目标单元；如果是"."，则是空单元；如果是 "#"，则是障碍单元。可以保证起始单元格和目标单元格各有一个。

您目前在起始单元格上。您的目标是通过重复移动到与您所在的单元格相邻的单元格。但是，您不能移动到障碍单元格上，也不能移动到网格外，而且每次移动都必须在垂直移动和水平移动之间交替进行。(第一次移动的方向可以任意选择）。

确定是否有可能到达目标单元格。如果有可能，请找出所需的最少移动次数。

更具体地说，检查是否存在满足以下所有条件的单元格序列 $(i_1,j_1),(i_2,j_2),\ldots,(i_k,j_k)$ 。如果存在这样的序列，请找出 $k-1$ 的最小值。

- 对于每个 $1\le l\le k$，都认为 $1\le i_l\le H$ 和 $1\le j_l\le W$ 以及 (i_l,j_l) 不是障碍单元格。
- (i_1,j_1) 是起始格。
- (i_k,j_k) 是目标单元。
- 对于每个 $1\le l\le k-1$,$|i_l-i_{l+1}|+|j_l-j_{l+1}|=1$.
- 每 $1\le l\le k-2$，如果 $i_l\ne i_{l+1}$, 则 $i_{l+1}=i_{l+2}$
- 每 $1\le l\le k-2$，如果 $j_l\ne j_{l+1}$，则 $j_{l+1}=j_{l+2}$.

限制因素

- $1\le H,W\le 1000$
- H and W are integers.
- Each S_i is a string of length W consisting of `S`, `G`, `.`, `#`.
- There is exactly one start cell and exactly one goal cell.

思路:考虑到达某一点是水平的还是竖直的

为什么可以在一次BFS里面跑?

考虑用水平的起点S_1和竖直的起点S_2，设S_1更新了某点X以水平的方向,之后S_2又以相同的方向更新了X,那么第二次的更新会重复第一次的更新轨迹，所以即使第二次更新时 vis 数组已经被置1了，也不会影响答案的正确性

```
#include<bits/stdc++.h>
using namespace std;
using i64 = long long;
using i128 = __int128;

constexpr int maxn = 1e3+10;
int n,m,vis[maxn][maxn][2];
char mp[maxn][maxn];
int dx[4]={0,0,-1,1},dy[4] = {1,-1,0,0};

inline bool check(int x,int y,int dir){
    if(vis[x][y][dir]||mp[x][y]=='#'||x<1||x>n||y<1||y>m) return 0;
    return 1;
}

inline int bfs(array<int,2> &st,array<int,2>&ed){
    int ans = 1e9;
    array<int,4> st0={st[0],st[1],0,0};
    array<int,4> st1={st[0],st[1],1,0};
    queue<array<int,4>> q;
    q.push(st0);
    q.push(st1);
    vis[st[0]][st[1]][st[2]]=1;
    while(!q.empty()){
        array<int,4> tmp = q.front();
        q.pop();
        if(tmp[0]==ed[0]&&tmp[1]==ed[1]){
            ans = min(ans,tmp[3]);
```

```
                continue;
            }
            for(int i = 0;i<4;++i){
                int tx = tmp[0]+dx[i],ty = tmp[1]+dy[i];
                if(tmp[2]==0&&i<2&&check(tx,ty,tmp[2]^1)){
                    q.push({tx,ty,tmp[2]^1,tmp[3]+1});
                    vis[tx][ty][tmp[2]^1]=1;
                }else if(tmp[2]&&i>1&&check(tx,ty,tmp[2]^1)){
                    q.push({tx,ty,tmp[2]^1,tmp[3]+1});
                    vis[tx][ty][tmp[2]^1]=1;
                }
            }
        }
        return ans;
}

signed main(){
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n>>m;
    array<int,2> st,ed;
    for(int i = 1;i<=n;++i)
    for(int j = 1;j<=m;++j){
        cin>>mp[i][j];
        if(mp[i][j]=='S') st = {i,j};
        if(mp[i][j]=='G') ed = {i,j};
    }
    int ans = 1e9;
    ans = min(ans,bfs(st,ed));
    if(ans>=1e9) cout<<"-1\n";
    else cout<<ans<<"\n";

    return 0;
}
```

- ## $Eg3$：寻找联通块的个数

- 联通块是指块内部是可以任取两点是可以相互到达的,块与块之间的点没有边连接

https://www.luogu.com.cn/problem/P1451

题目描述

一矩形阵列由数字 0 到 9 组成，数字 1 到 9 代表细胞，细胞的定义为沿细胞数字上下左右若还是细胞数字则为同一细胞，求给定矩形阵列的细胞个数。

输入格式

第一行两个整数代表矩阵大小 n 和 m。

接下来 n 行，每行一个长度为 m 的只含字符 `0` 到 `9` 的字符串，代表这个 $n \times m$ 的矩阵。

输出格式

一行一个整数代表细胞个数。

数据规模与约定

对于 100% 的数据，保证 $1 \le n, m \le 100$。

教一下开嵌套的东西 like vector<vector<array<int,2>>>,传到外面要取值引用

思路:

每次遇到一个没有遍历过的数值大于0的点，就进行一次BFS，直到遍历完所有的地图,时间复杂度 $O(nm)$

```cpp
#include<bits/stdc++.h>
using namespace std;
using i64 = long long;
using i128 = __int128;

int n,m;
int dx[4]={0,0,-1,1},dy[4]={1,-1,0,0};
void bfs(int i,int j,vector<vector<int>>& mp,vector<vector<bool>>&vis){
    queue<array<int,2>> q;
    q.push({i,j});
    vis[i][j]=1;
    while(!q.empty()){
        auto tmp = q.front();
        q.pop();
        for(int i = 0;i<4;++i){
            int tx = dx[i]+tmp[0],ty = dy[i]+tmp[1];
            if(tx>=1&&tx<=n&&ty>=1&&ty<=m&&!vis[tx][ty]&&mp[tx][ty]){
                q.push({tx,ty});
                vis[tx][ty]=1;
            }
        }
    }
}
signed main(){
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n>>m;
    vector<vector<int>> mp(n+1,vector<int>(m+1,0));
    vector<vector<bool>> vis(n+1,vector<bool>(m+1,0));
    for(int i = 1;i<=n;++i){
```

```
        for(int j = 1;j<=m;++j){
            char c;
            cin>>c;
            mp[i][j]= c-'0';
        }
    }

    int ans = 0;
    for(int i = 1;i<=n;++i){
        for(int j = 1;j<=m;++j){
            if(mp[i][j]&&!vis[i][j]){
                ++ans;
                bfs(i,j,mp,vis);
            }
        }
    }
    cout<<ans<<"\n";
    return 0;
}
```

图论

- 常见的图论算法会有 拓扑排序，最短路，最小生成树，由于时间关系可能讲不了太多，剩下的靠大家自学
- 最短路里面有 **Dij**，SPFA , Floyd , ...
- 最小生成树有 Prim ， **Kruskal** ， ....

边的存储

- 有向边:有方向的边，eg $\mathrm{x} \to \mathrm{y}$是一条只能由 x 到 y 的边，同理无向图就是没有方向的边
- 边权:边的权值，同理，点权就是点的权值
- 自环(loop):从自己到自己的一条边
- 重边:指有不止一条 $\mathrm{x} \to \mathrm{y}$ 的边
- 邻接矩阵 :用二维矩阵实现，在搜索中我们已经尝试过了，这里就不再提及
- 链式前向星
- $vector$模拟邻接表

链式前向星

大部分情况都用$vector$为什么还要学?

$vector$存边在某些情况可能会导致 MLE （$vector$ 每次扩容默认申请 2 倍空间），并且网络流的一些实现也是使用链式前向星的

实现原理(画图):

$head[x]$ 存储的是访问与 x 节点相邻的边的入口

$edge$ 数组存储的是每条边的信息,边的另一端点，边的权重 ,下一条边在$edge$ 数组的索引位置

每次插入是**从头插的**，无向图中要正反存两次边

```
struct ty{
    int  to,w,nex;//另一个端点，权重，下一个索引位置
}edge[maxn];
int tot,head[maxn];
inline void add(int x,int y,int z){//加边
    edge[++tot]={y,z,head[x]};//先做++tot，从一号节点开始存
    head[x]=tot;
}

signed main(){
    memset(head,-1,sizeof head);
    ......
}

for(int i= head[x];~i;i = edge[i].nex){//访问
    ....
}
```

$vector$ 存边

```
constexpr int maxn = 2e5+10;
vector<array<int,2>> g[maxn];//要存的东西比较多时，可以开结构体
//vector<vector<int>> g(n,vector<int>());

//存边
g[x].push({y,w});//y是另一个端点，w是权重

//访问,假装跑DFS
for(auto y:g[x]){//直接使用智能指针或者确定类型
    //假设这里的y是array<int,2>
    if(y[0]==x) continue;
    dfs(y[0],x);
}
```

- ## Eg : 求树的直径

$\text{Def of the diameter of a tree}: $ 树上任意两点间的简单路径的长度的最大值

换句话说就是:一棵树上能找出来的最长的链

一棵树可以有多个直径，只要链的长度等于最大值即可

这里直接给出求树的直径的方法，证明可以参考链接: https://oi-wiki.org/graph/tree-diameter/

具体做法:

跑两遍 DFS , 从任意节点 x 出发，到达距离 x 最远的根节点 y ,再从 y 开始 DFS , 找到离 y 最远的 z ，那么 y 到 z 的简单路径长度就是树的直径.**注意**，该做法只适用于边权全为正的情况

OJ地址: https://www.spoj.com/problems/PT07Z/

无向边从哪个节点开始 DFS 都没有关系，不过树的形态会随之变化

用有向边构建的树，在大部分情况都可以写成无向边的形式

```
#include<bits/stdc++.h>
using namespace std;
using i64 = long long;
using i128 = __int128;

constexpr int maxn = 2e5+10;
int n,dep[maxn];
vector<int> g[maxn];

void dfs(int x,int fa){
    dep[x] = dep[fa]+1;
    for(int y:g[x]){//auto 也可
        if(y==fa) continue;
        dfs(y,x);
    }
}
```

```cpp
signed main(){
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n;
    for(int i = 1;i<n;++i){
        int x,y;
        cin>>x>>y;
        g[x].push_back(y);
        g[y].push_back(x);
    }
    dfs(1,0);//从1开始
    int mx = *max_element(dep+1,dep+n+1);
    //返回开始位置的迭代器，到结束位置的的迭代器的前一个位置这个区间的最大值,支持自定义比较,
o(n)
    int p;
    for(int i = 1;i<=n;++i){
        if(mx==dep[i]){
            p = i;
            break;
        }
    }

    dfs(p,0);
    cout<<*max_element(dep+1,dep+n+1)-1;
    return 0;
}
```

最短路

字面意思，就是取求点 u 到点 v 的最短的路径长度D_{min}， $s.t.$ 任何其它 u 到达 v 的路径长度 $D_i \geq D_{min}$

- $Dijkstra$ 单源点最短路,**不适用有负权边的图**
- $Floyd$ 全源最短路,可求任意两点间的最短路
- $Bellman\text{–}Ford$ 及其优化 $SPFA$ 单源点最短路，可以处理有负权边的图

Dijstra

过程:(*)

将结点分成两个集合：已确定最短路长度的点集（记为 S 集合）的和未确定最短路长度的点集（记为 T 集合）。一开始所有的点都属于 T 集合。

- 初始化，将起点的最短路置0，$dis[s] = 0$ ，其它置为 $+\infty$
- 然后重复以下操作知道 T 为空:

  1.从集合 T 中，选择一个最短路长度最小的节点，移入 S 集合

  2.对刚刚加入 S 的节点 x ，将它的出边进行松弛(能否用到 x 的最短路+ $x \to y$ 的边权更新到 y 的最短路)

朴素实现: n 个节点，每次在 T 中暴力的找最短路最小的节点，时间复杂度是 $O(n^2)$

堆:一种可以快速获得最大值或者最小值的数据结构,插入/删除 时间复杂度为$O(log)$

堆优化：每成功一条边 (u, v) 就插入到堆中，每次取出堆顶更新最短路(如果不是最短路的话)，继续尝试松弛

```
using i64 = long long;
constexpr int maxn=2e5+10;
int n,m;//n个点,m条边
struct ty{
    i64 x,dis;
    bool operator<(const ty &u)const{
        return dis>u.dis;
    }
};
vector<array<i64,2>> g[maxn];
priority_queue<ty> q;
i64 dis[maxn];
bool vis[maxn];//判断最短路是否已经更新
void dij(int s){
    for(int i = 1;i<=n;++i){//初始化，有多少用多少
        dis[i]=1e9;//设置成无穷，一般int范围内1e9足够了
        vis[i]=0;
    }
    dis[s]=0;
    q.push({s,0});
    while(!q.empty()){
        auto tmp = q.top();
        q.pop();
        int x = tmp.x;
        if(vis[x]) continue;
        vis[x]=1;
        for(auto y:g[x]){
            int to =y[0];i64 w = y[1];
            if(!vis[to]&&dis[to]>dis[x]+w){
                dis[to] = dis[x]+w;
                q.push({to,dis[to]});
            }
        }
```

```
    }
}
```

关于算法的正确性做个简单的说明:

设有一条到 t 的最短路，比上述做法$(*)$ 求出的最短路 s 更短，那么会在某一步取出最短路时先更新属于 t 的路径,那么 s 就不会是 $(*)$ 求出的最短路. 与已知矛盾,所以上述做法求出来的最短路正确性得证

拓扑排序

●DAG : 有向无环图,图里面的边都是有向边，并且不存在回路

拓扑排序的作用就是给一张 DAG 的节点进行编号和排序