搜索与图和树之间有着密不可分的关系。无论是什么搜索，本质上都可以使用图论问题来进行转化。我们将以树和图的遍历来作为搜索和图论的开端，尝试在树和图的结构上进行访问，记录和推导。这节课中，我将介绍建图的常见方法，以及树的遍历产生的几种常见序列。基于此，可以对dfs和bfs两种搜索在图论中搜索过程有一个较为清晰的认知。

图和树的遍历

图(树就是一种特殊的图)的基本操作基本上就是两个，一个添加新的边，一个是遍历从一个点出发的所有边，并且找到这些边的相关信息，如这条边的边权为什么，这条边的终点是哪一个点。因为通过这样的遍历，我们可以做到从某一个点出发，找到我们要前往的下一个点是哪个。这样一步一步的往下走，就可以依照某种顺序完成对于整个图的遍历。

因此对于以下的所有的建图方法，我们最重要要考虑到的特质就是

1.存储的空间

2.添加新的边的方法(如何利用这个方法去建图)

3.对于一个确定的点如何遍历所有的出边（我们可以通过这个子功能，实现遍历整个图的功能）

常见的建图方法

邻接矩阵

最最简单和直白的表现点与点的关系的建图方法。

具体就是建立一个二维数组$a[i][j]$，当对于任意一对i,j，$a[i][j]=1$表示有一条i指向j的边，否则表示没有这样的边。

具体就是建立一个二维数组$a[i][j]$，当对于任意一对i,j，$a[i][j]=x$表示有一条i指向j的边权为x的边，否则表示没有这样的边。

256MB的空间能开多少int，$2*10^7$

```
const int N=1010;//题目中最多有多少个点
int a[N][N];
int n,m;
int main()
{
    cin>>n>>m;//这里以及下面的n和m都是n代表点数，m代表需要读入的边数
    for(int i=1;i<=m;i++)
    {
        int x=read(),y=read();//有一条x->y的边，的同时，还有一条y->x的边
        a[x][y]=1;a[y][x]=1;
    }
    //如何对于一个确定的点如何遍历所有的出边
    //比如我们现在要遍历x的所有出边
    int x=read();
    for(int i=1;i<=n;i++)
    {
        if(a[x][i]==1)
        {
            cout<<i<<' ';
        }
    }
    cout<<endl;//我们就输出了x只通过一条边能够到达的所有点
```

```
}
/*

a[1][2]=10
a[2][3]=1
a[1][3]=1

4 4
1 2
2 4
3 4
1 3

5 4
1 2
1 3
3 4
3 5

*/
```

这个存储方法的劣势非常明显，对于稀疏图(即边较少，而点较多的图)，我们需要$(n*n)$的空间来建图，而每次想要遍历，则需要枚举所有的点的编号，一个一个查询每个点作为重点是否和这个点有连边

```
int x;
cin>>x;
for(int i=1;i<=n;i++)//O(n)
{
    if(a[x][i]==1)//对于点x，我想要遍历他的所有出边 x->i
    {
        dfs(i);//找到了x有一个通向i的边。
    }
}
```

而这个过程，也是$O(n)$的

邻接矩阵的优势体现在存储密集图的时候。这个的原因会在之后讲解其他建图方法时讲到。
但是非常不幸的是，你能见到的绝大多数的题目都是稀疏图，也就是常常看到的$1 \leq n, m \leq 10^5$。而这种情况下，我们为了建图去开一个$a[100001][100001]$是非常不现实的，简单的计算就可以得到，我们开了10^10个int的空间，而$256MB$的最多支持你开2×10^7个int。(如果不会计算可以记住)
这个空间是我们无法接受的。更何况，有很多的题目给的点数更是10^6级别的。

vector

这个方法并没有一个特别常用的名字。我觉得也许是因为它实在实在是太简洁了，以至于没有为了这个方法起一个名字的必要。唯一的特点就是，它利用了$C++$自带的一个stl类型$vector$。所以他可以叫做用vector建图。

vector又名变长数组。它的特点就是，当你不向里面存储数字的时候，它不会占用空间。

```
vector<int> fir;//声明了一个vector类型变量，里面的每一个元素都是int类型

//到现在位置，他不占用空间
```

```
int main()
{
    fir.push_back(1);//往末尾添加一个数字1
    fir.push_back(2);//往末尾添加一个数字2
    fir.push_back(1);//往末尾添加一个数字1
    fir.push_back(3);//往末尾添加一个数字3
    for(int i=0;i<fir.size();i++)//遍历了整个vector数组
    {
        cout<<fir[i]<<' ';
    }
    cout<<endl;
}
```

在这个过程中，这个名为fir的`vector`类型的变量所占用的空间是随着我们添加数字的增加逐渐变大的。

而正因为这个特性，下面这种建图的方式的空间复杂度就可以变得十分优秀

```
vector< int > a[100001];//声明了100001个vector类型的变量
//a[i]这个vector的内容的含义是,i这个节点有哪些出边
//到此为止他不花多少空间
int main()
{
    int n,m;
    cin>>n>>m;
    for(int i=1;i<=m;i++)//添加m条边，只增加了m的空间
    {
        int x,y;
        cin>>x>>y;
        a[x].push_back(y);//添加边
        //这样就存储了一条(x,y)的边
    }
}
```

在这个名为a的vector数组中，下标为x的vector代表了从x出发，能够走到那些点。
而因为vector不存东西就不会变大的特性，我们的空间复杂度就被压缩到了$O(m)$的级别，也就是，有几条边，我们便用了多少空间。
而这个空间复杂度是完全可接受的。

```
for(int i=0;i<a[x].size();i++)//遍历并输出了从x出发的所有边的终点，也就是在这张图上，x能够走到哪些点。
{
    int u=a[x][i];
    cout<<u<<' ';
}
```

vector建图的空间复杂度是$O(m)$的级别，是事实上，任何会需要正常建图的题目，m的数量级都不会超过$1e6$。也就是基本上所有需要建图的情况都是可以直接使用vector建图的。

而vector遍历一个点的所有出边的复杂度就等于这个点有多少条出边。相比邻接矩阵的$O(n)$遍历出边，是非常非常优秀的。它遍历一整个图的均摊复杂度是$O(n)$级别的，而邻接矩阵则是确定的$O(n^2)$。

邻接表

有些人称它为链式前向星。因为它本质就是链表

与vector相比，邻接表只需要数组就可以完成。是更加朴素且优美的方法。我个人写题基本上图论都是使用邻接表完成的。当然，他的理解也是更有难度的。

```
struct edge
{
    int next,to;//下一个节点的位置，以及当前节点的内容
}e[100001];//e的本质是一个链表
int head[100001],tot;//head记录了每一个链的开头。
/*
邻接表其实是将每一个点的一系列出边，在e数组中串联成为了一个链表。
e数组next的含义就是这个链表的下一项是什么，而to中存储的是链表上这个节点的信息，也就是从这个节点出发的一条边的终点。
而head[x]则是记录了x这个节点，他的出边在e数组中所连接成的链表，的开头，是哪个位置。
*/
inline void add(int i,int j)//在head[i]表示的链的头部，插入了一个新的节点
{
    tot++;//e[tot]便是我们新添加的点 tot=10
    e[tot].next=head[i];
    head[i]=tot;
    e[tot].to=j;
}
int main()
{
    int n,m;
    cin>>n>>m;
    for(int i=1;i<=m;i++)
    {
        int x,y;
        cin>>x>>y;
        add(x,y);//代表建一条从x到y的边
    }

    int x;
    cin>>x;//x=1
    for(int i=head[x];i!=0;i=e[i].next)//遍历了从x出发能够到达的所有的点
    {
        int u=e[i].to;//找到了(x,u)zhe'tiao'bian
        cout<<u<<' ';
    }
}
```

邻接表的空间复杂度和时间复杂度和vector建图是完全对应的，都是相同的数量级，因此在绝大部分情况下是可以被完全替代的。

但是我依旧希望你们能够理解邻接表的运行方式并且能够独立的编写，因为在一些特殊的情况下，邻接表也有很大的优势

而基于上面的建图方法，我们就能够通过dfs的方式来遍历整个图。

```
vector<int> a[100001];//声明了100001个vector类型的变量
```

```cpp
bool vis[100001];
void dfs(int x)
{
    vis[x]=1;//标记，这个点走过了。
    cout<<x<<' ';
    for(int i=head[x];i!=0;i=e[i].next)//遍历了从x出发能够到达的所有的点
    {
        int u=e[i].to;//遍历这个点能够走到哪些点
        if(vis[u]==1)continue;//如果这个点被走过，那就不要走进去
        dfs(u);//走进去
    }
}
int main()
{
    int n,m;
    cin>>n>>m;
    for(int i=1;i<=m;i++)
    {
        int x,y;
        cin>>x>>y;
        a[x].push_back(y);//添加双向边
        a[y].push_back(x);
    }
    dfs(1);//从1开始遍历整张图
}
```

小结论

所有n个点，n-1条边的连通图都是树

例题

例1 猫猫和企鹅

题目描述

王国里有 n 个居住区，它们之间有 $n-1$ 条道路相连，并且保证从每个居住区出发都可以到达任何一个居住区，并且每条道路的长度都为 1。

除 1 号居住区外，每个居住区住着一个小企鹅，有一天一只猫猫从 1 号居住区出发，想要去拜访一些小企鹅。可是猫猫非常的懒，它只愿意去距离它在 d 以内的小企鹅们。

猫猫非常的懒，因此希望你告诉他，他可以拜访多少只小企鹅。

输入格式

第一行两个整数 n, d，意义如题所述。

第二行开始，共 $n-1$ 行，每行两个整数 u, v，表示居民区 u 和 v 之间存在道路。

输出格式

一行一个整数，表示猫猫可以拜访多少只小企鹅。

样例 #1

样例输入 #1

```
5 1
1 2
1 3
2 4
3 5
```

样例输出 #1

```
2
```

提示

对于 100% 的数据，满足 $1 \leq n, d \leq 10^5$，保证所有居民区从 1 开始标号。

基础的遍历。

例2 [USACO06DEC] Cow Picnic S

题目描述

The cows are having a picnic! Each of Farmer John's K (1 ≤ K ≤ 100) cows is grazing in one of N (1 ≤ N ≤ 1,000) pastures, conveniently numbered 1...N. The pastures are connected by M (1 ≤ M ≤ 10,000) one-way paths (no path connects a pasture to itself).

The cows want to gather in the same pasture for their picnic, but (because of the one-way paths) some cows may only be able to get to some pastures. Help the cows out by figuring out how many pastures are reachable by all cows, and hence are possible picnic locations.

$K(1 \leq K \leq 100)$ 只奶牛分散在 $N(1 \leq N \leq 1000)$ 个牧场．现在她们要集中起来进餐。牧场之间有 $M(1 \leq M \leq 10000)$ 条有向路连接，而且不存在起点和终点相同的有向路．她们进餐的地点必须是所有奶牛都可到达的地方。那么，有多少这样的牧场可供进食呢?

输入格式

Line 1: Three space-separated integers, respectively: K, N, and M

Lines 2..K+1: Line i+1 contains a single integer (1..N) which is the number of the pasture in which cow i is grazing.

Lines K+2..M+K+1: Each line contains two space-separated integers, respectively A and B (both 1..N and A != B), representing a one-way path from pasture A to pasture B.

输出格式

Line 1: The single integer that is the number of pastures that are reachable by all cows via the one-way paths.

样例 #1

样例输入 #1

```
2 4 4
2
3
1 2
1 4
2 3
3 4
```

样例输出 #1

```
2
```

提示

The cows can meet in pastures 3 or 4.

遍历并统计

例3 图的遍历

题目描述

给出 N 个点，M 条边的有向图，对于每个点 v，求 $A(v)$ 表示从点 v 出发，能到达的编号最大的点。

输入格式

第 1 行 2 个整数 N, M，表示点数和边数。

接下来 M 行，每行 2 个整数 U_i, V_i，表示边 (U_i, V_i)。点用 $1, 2, \ldots, N$ 编号。

输出格式

一行 N 个整数 $A(1), A(2), \ldots, A(N)$。

样例 #1

样例输入 #1

```
4 3
1 2
2 4
4 3
```

样例输出 #1

```
4 4 3 4
```

提示

- 对于 60% 的数据，$1 \leq N, M \leq 10^3$。
- 对于 100% 的数据，$1 \leq N, M \leq 10^5$。

先考虑60分的部分，再考虑100剩下100分的部分。

技巧：反图。正难则反的思路。