



北京大学
PEKING UNIVERSITY

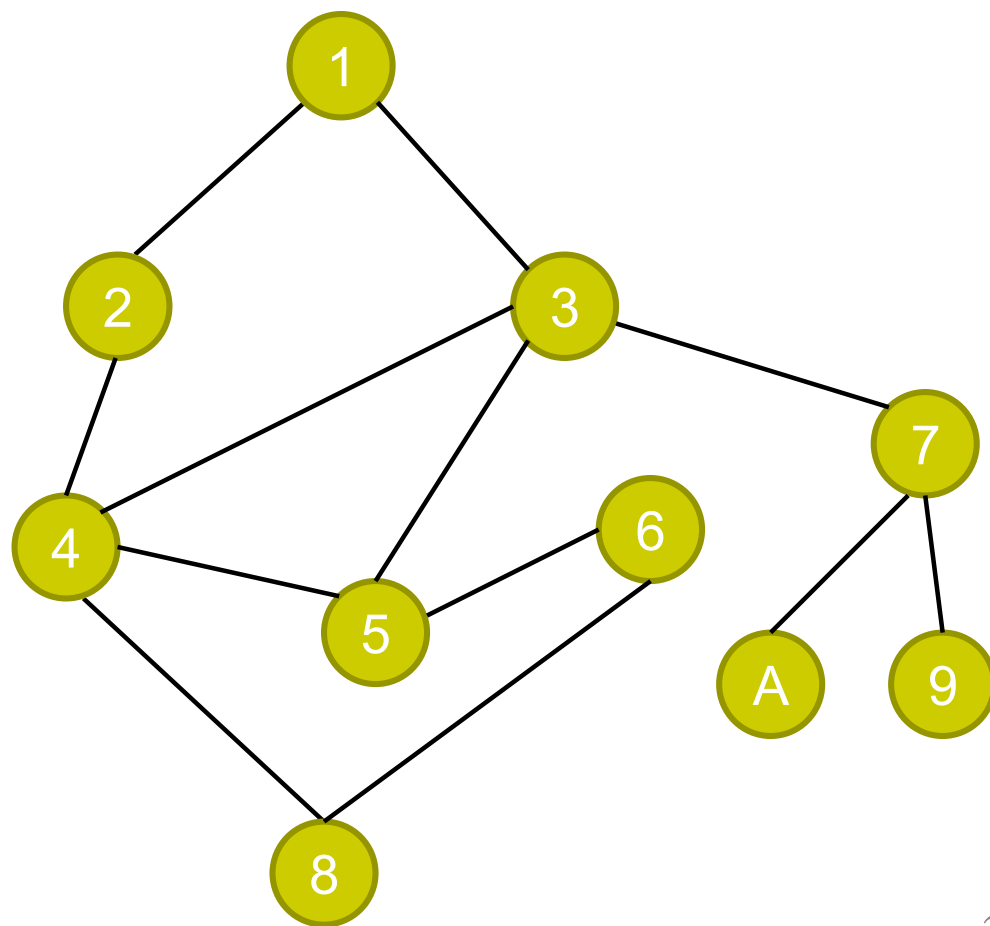
深度优先搜索

入门：城堡问题

在图上寻找路径

在图上如何寻找从1到8的路径？

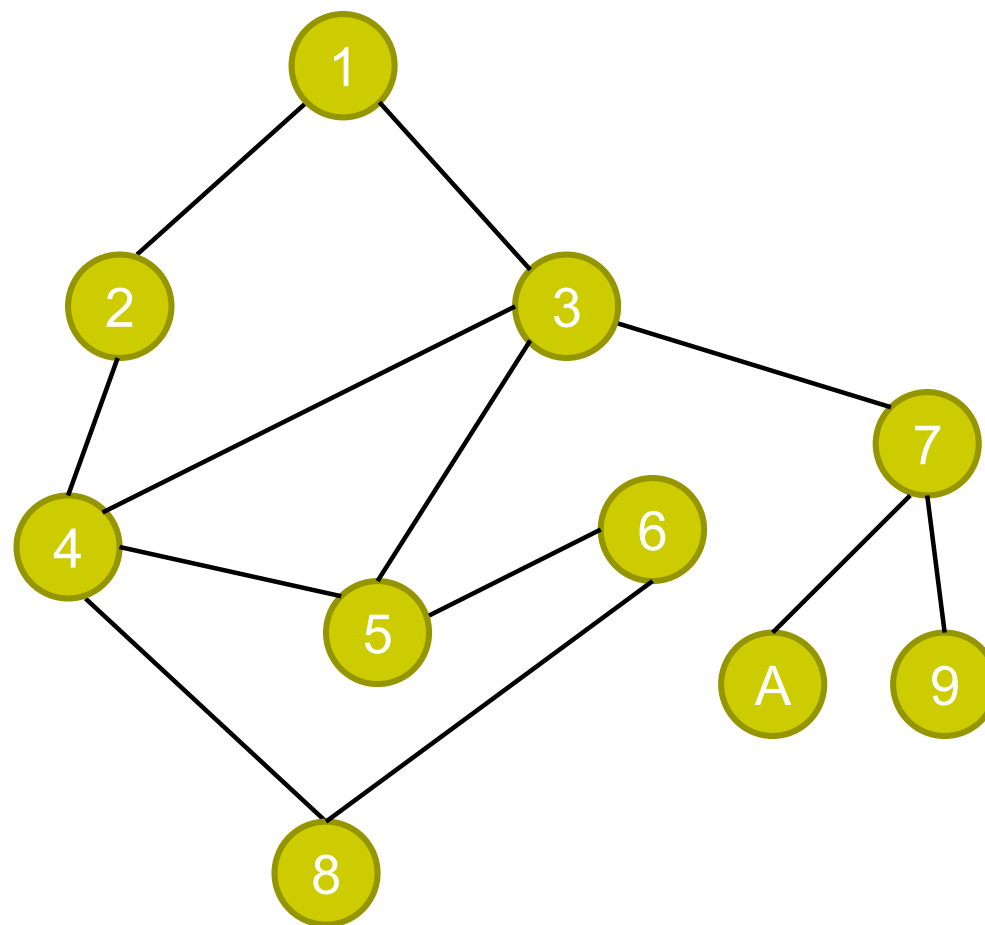
一种策略：只要能发现没走过的点，就走到它。有多个点可走就随便挑一个，如果无路可走就回退，再看有没有没走过的点可走



在图上寻找路径

在图上如何寻找从1到8的路径？

运气最好： 1->2->4->8

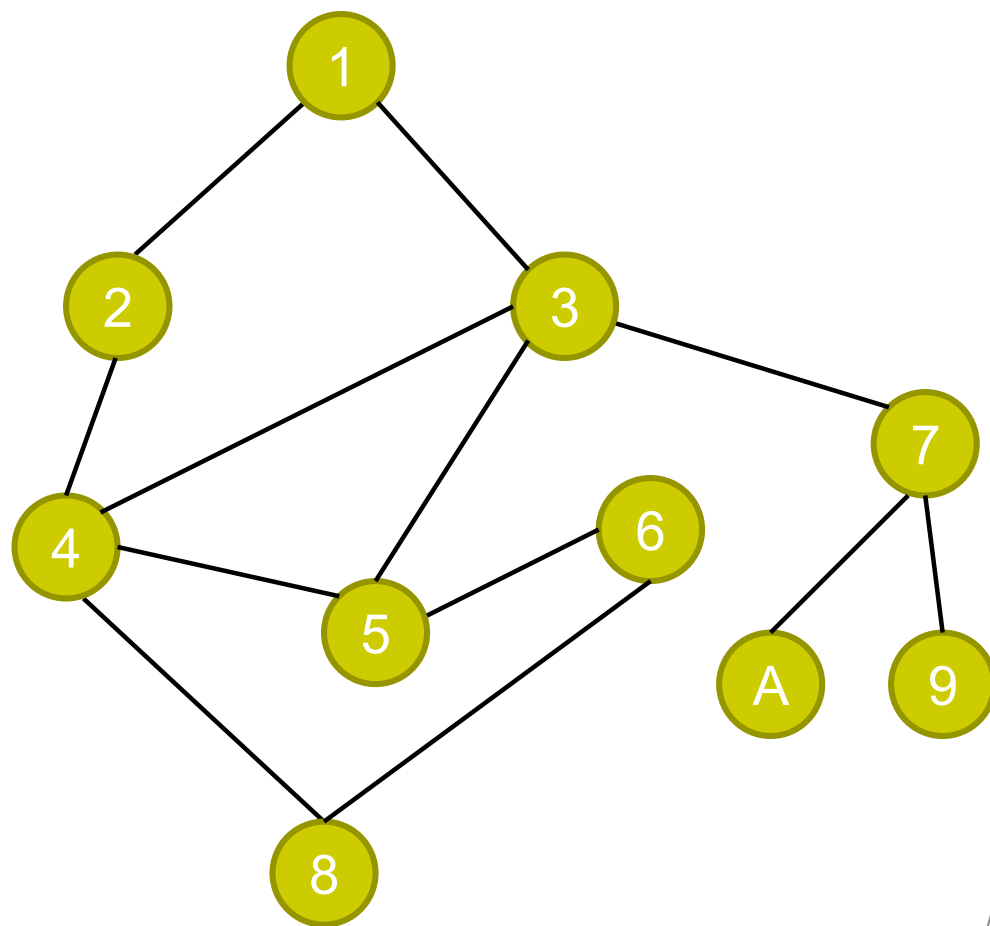


在图上寻找路径

在图上如何寻找从1到8的路径？

运气最好： 1->2->4->8

运气稍差： 1->2->4->5->6->8



在图上寻找路径

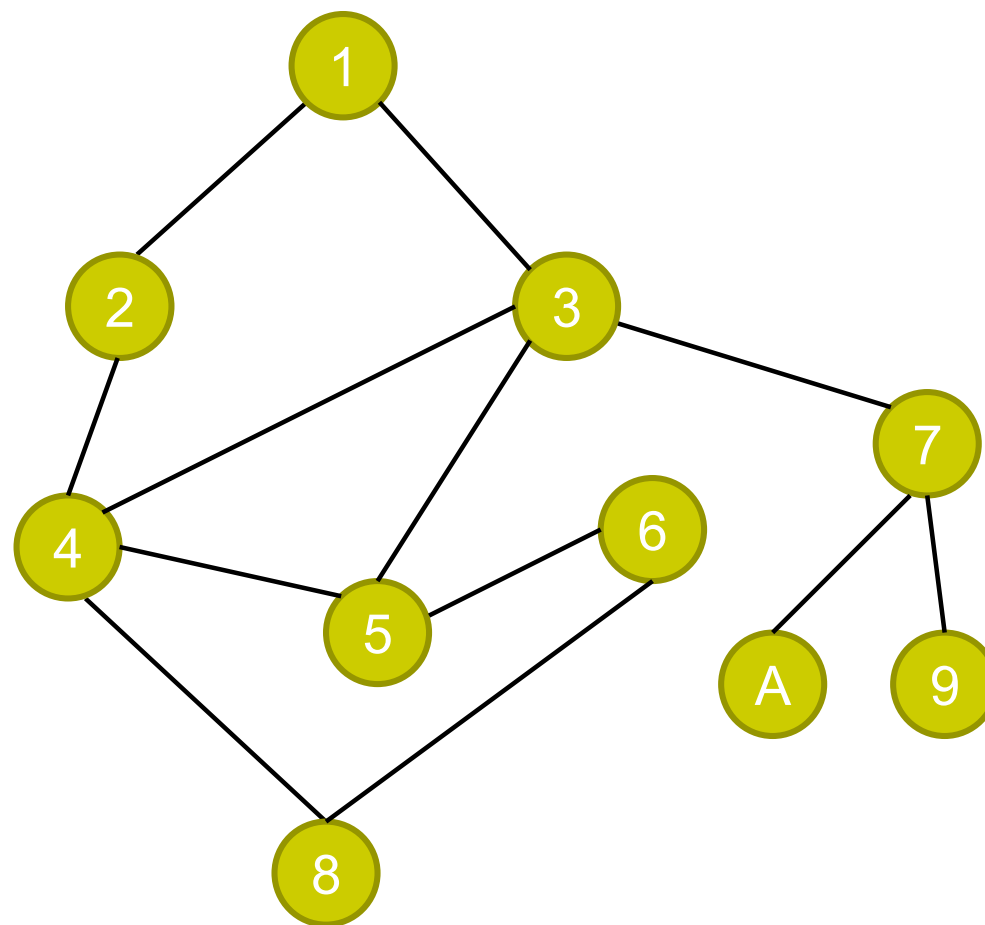
在图上如何寻找从1到8的路径?

运气最好: 1->2->4->8

运气稍差: 1->2->4->5->6->8

运气坏:

1->3->7->9=>7->A=>7=>3->5->6->8
(双线箭头表示回退)



在图上寻找路径

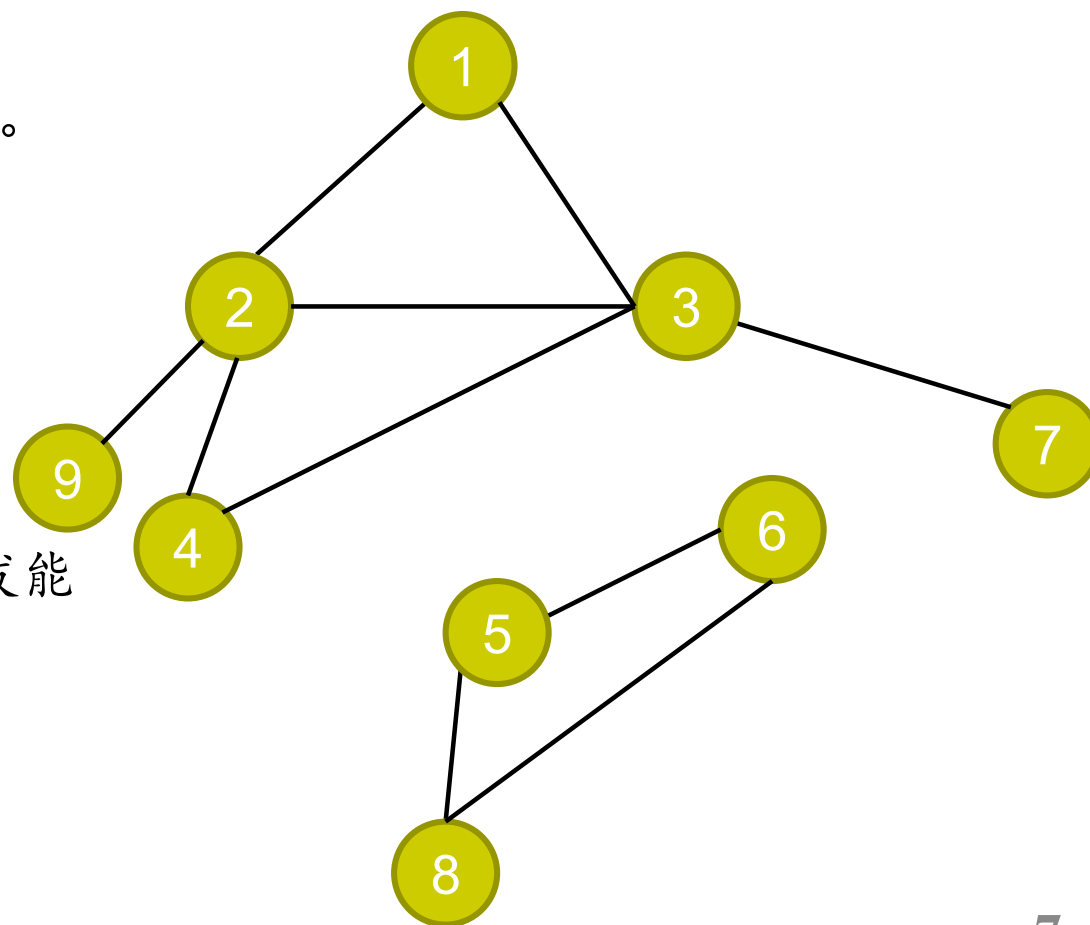
不连通的图，无法从节点1走到节点8。

完整的尝试过程可能如下：

1->2->4->3->7=>3=>4=>2->9=>2=>1

结论：不存在从1到8的路径

得出这个结论之前，一定会把从1出发能走到的点全部都走过。



在图上寻找路径

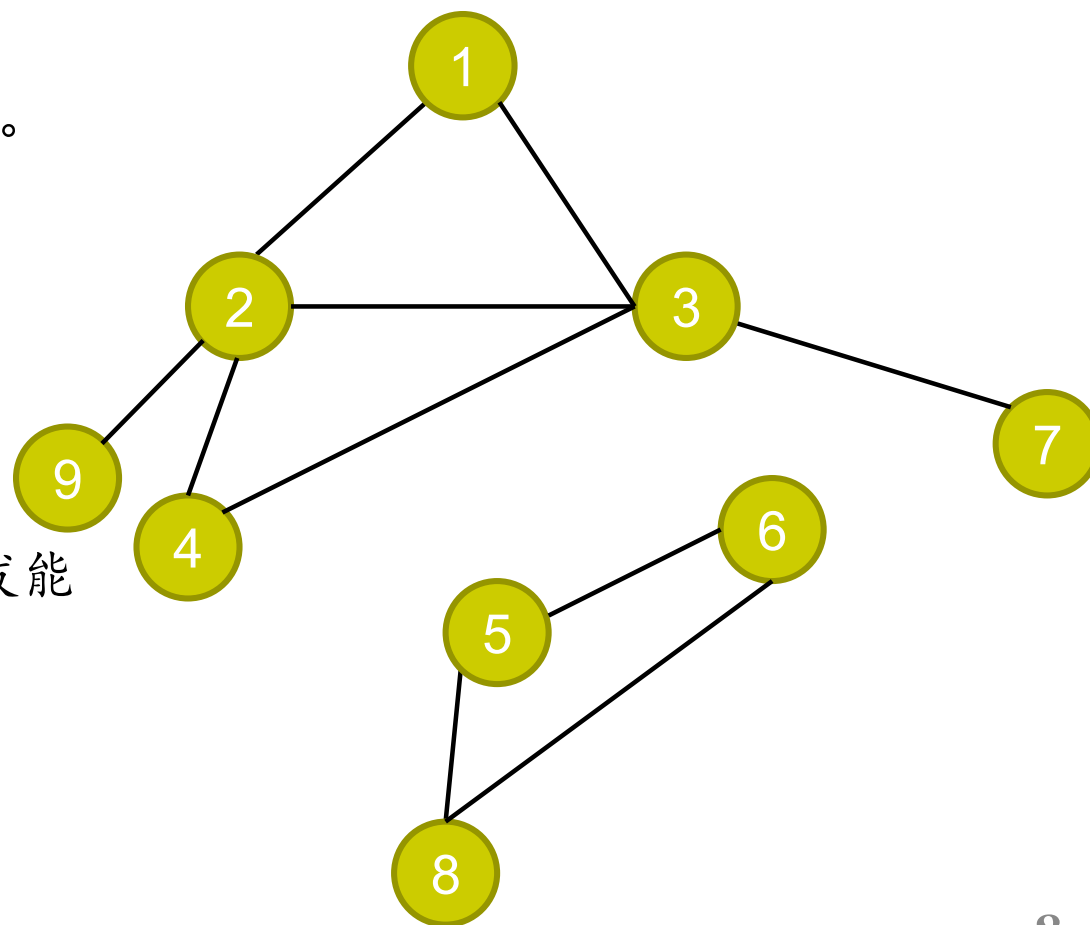
不连通的图，无法从节点1走到节点8。

完整的尝试过程可能如下：

1->2->4->3->7=>3=>4=>2->9=>2=>1

结论：不存在从1到8的路径

得出这个结论之前，一定会把从1出发能走到的点全部都走过。



深度优先搜索 (Depth-First-Search)

从起点出发，走过的点要做标记，发现有没走过的点，就随意挑一个往前走，走不了就回退，此种路径搜索策略就称为“深度优先搜索”，简称“深搜”。

其实称为“远度优先搜索”更容易理解些。因为这种策略能往前走一步就往前走一步，总是试图走得更远。所谓远近(或深度)，就是以距离起点的步数来衡量的。

在图上寻找路径

➤判断从V出发是否能走到终点:

```
bool Dfs(V) {  
    if( V 为终点)  
        return true;  
    if( V 为旧点)  
        return false;  
    将V标记为旧点;  
    对和V相邻的每个节点U {  
        if( Dfs(U) == true)  
            return true;  
    }  
    return false;  
}
```

在图上寻找路径

```
int main()  
{  
    将所有点都标记为新点 ;  
    起点 = 1  
    终点 = 8  
    cout << Dfs(起点) ;  
}
```

在图上寻找路径

➤判断从V出发是否能走到终点,如果能,要记录路径:

```
Node path[MAX_LEN]; //MAX_LEN取节点总数即可
int depth;
bool Dfs(V) {
    if( V为终点) {
        path[depth] = V;
        return true;
    }
    if( V 为旧点)
        return false;
    将V标记为旧点;
    path[depth]=V;
    ++depth;
```

在图上寻找路径

```
对和v相邻的每个节点U    {  
    if( Dfs(U) == true)  
        return true;  
}
```

```
--depth;
```

```
return false;
```

```
}
```

```
int main()
```

```
{
```

```
    将所有点都标记为新点;
```

```
    depth = 0;
```

```
    if( Dfs(起点) ) {
```

```
        for(int i = 0; i <= depth; ++ i)
```

```
            cout << path[i] << endl;
```

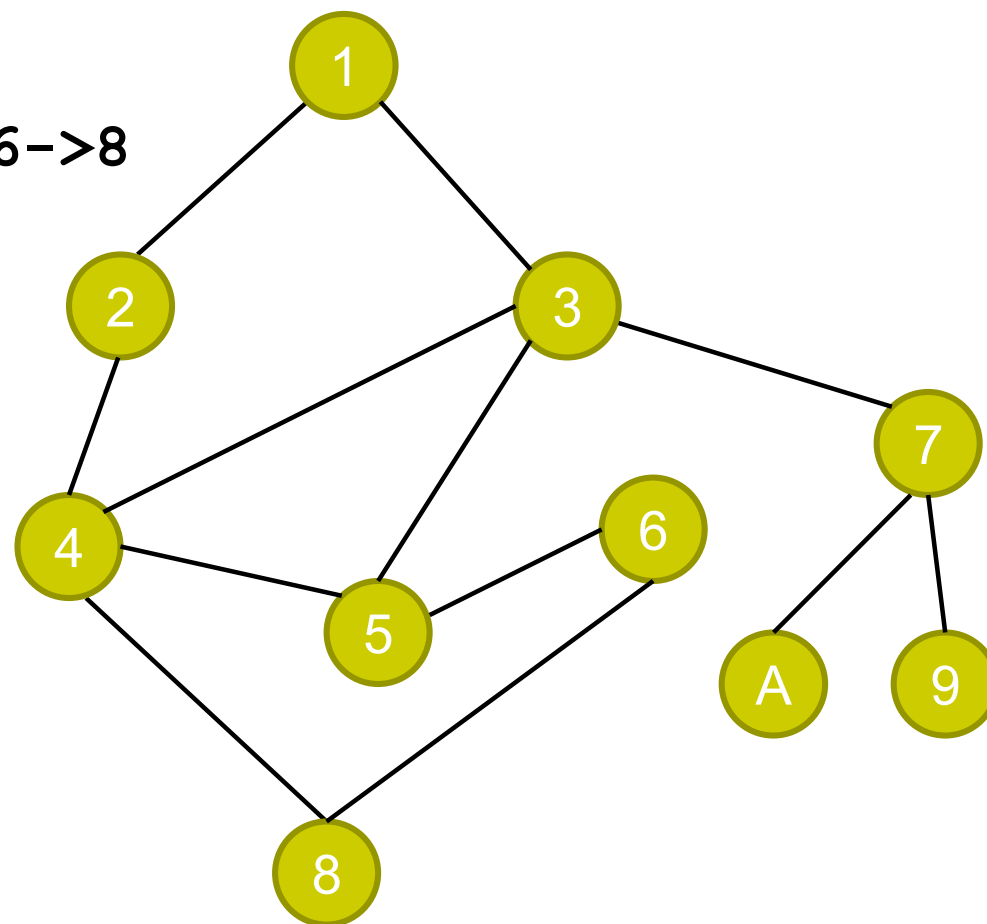
```
    }
```

```
}
```

在图上寻找路径

1->3->7->9=>7->A=>7=>3->5->6->8

path: 1,3,5,6,8



在图上寻找最优(步数最少)路径

```
Node bestPath[MAX_LEN];
int minSteps = INFINITE; //最优路径步数
Node path[MAX_LEN]; //MAX_LEN取节点总数即可
int depth;
void Dfs(V) {
    if( v为终点) {
        path[depth] = V;
        if( depth < minSteps ) {
            minSteps = depth;
            拷贝path到bestPath;
        }
        return;
    }
    if( v 为旧点) return;
    if( depth >= minSteps ) return ; //最优性剪枝
    将v标记为旧点;
    path[depth]=V;
    ++depth;
```

在图上寻找最优(步数最少)路径

```
对和v相邻的每个节点u    {
```

```
    Dfs (U) ;
```

```
}
```

```
--depth;
```

```
将v恢复为新点
```

```
}
```

```
int main()
```

```
{
```

```
    将所有点都标记为新点;
```

```
    depth = 0;
```

```
    Dfs (起点) ;
```

```
    if( minSteps != INFINITE) {
```

```
        for(int i = 0; i <= minSteps; ++ i)
```

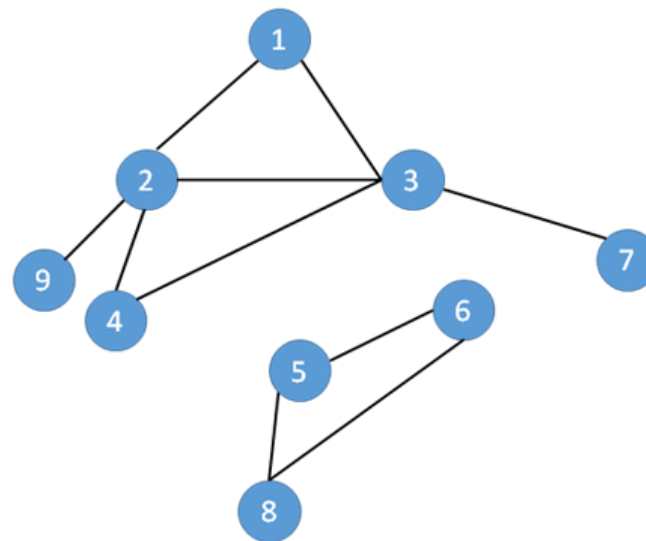
```
            cout << bestPath[i] << endl;
```

```
    }
```

```
}
```

遍历图上所有节点

```
Dfs(V) {  
    if( V是旧点)  
        return;  
    将v标记为旧点;  
    对和v相邻的每个点 U {  
        Dfs(U);  
    }  
}  
  
int main() {  
    将所有点都标记为新点;  
    while(在图中能找到新点k)  
        Dfs(k);  
}
```



图的表示方法 -- 邻接表

用一个二维数组G存放图， $G[i][j]$ 表示节点i和节点j之间边的情况(如有无边，边方向，权值大小等)。

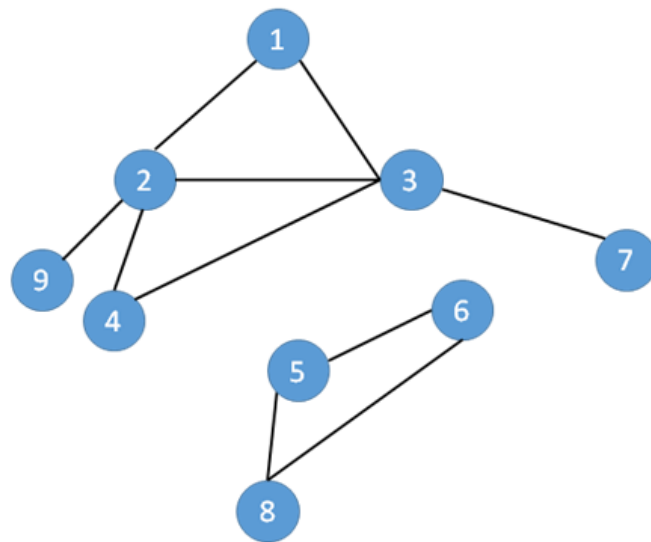
遍历复杂度： $O(n^2)$ n为节点数目

图的表示方法 —— 邻接表

每个节点V对应一个一维数组(vector)，里面存放从V连出去的边，边的信息包括另一顶点，还可能包含边权值等。

1	2	3		
2	1	4	9	3
3	1	4	7	2
4	2	3		
5	6	8		
6	5	8		
7	3			
8	5	6		
9	2			

遍历复杂度: $O(n+e)$
n为节点数目, e为边数目



例题：百练2815 城堡问题

- 右图是一个城堡的地形图。请你编写一个程序，计算城堡一共有多少房间，最大的房间有多大。城堡被分割成 $m \times n$ ($m \leq 50$, $n \leq 50$) 个方块，每个方块可以有0~4面墙。

```
      1   2   3   4   5   6   7
#####
1 #   |   #   |   #   |   |   #
#####---#####---#---#####---#
2 #   #   |   #   #   #   #   #
#---#####---#####---#####---#
3 #   |   |   #   #   #   #   #
#---#####---#####---#---#
4 #   #   |   |   |   |   #   #
#####
      (图 1)
```

= Wall
| = No wall
- = No wall

输入输出

- 输入

- 程序从标准输入设备读入数据。
- 第一行是两个整数，分别是南北向、东西向的方块数。
- 在接下来的输入行里，每个方块用一个数字($0 \leq p \leq 50$)描述。用一个数字表示方块周围的墙，1表示西墙，2表示北墙，4表示东墙，8表示南墙。**每个方块用代表其周围墙的数字之和表示。**城堡的内墙被计算两次，方块(1,1)的南墙同时也是方块(2,1)的北墙。
- 输入的数据保证城堡至少有两个房间。

- 输出

- 城堡的房间数、城堡中最大房间所包括的方块数。
- 结果显示在标准输出设备上。

- 样例输入

4

7

11 6 11 6 3 10 6

7 9 6 13 5 15 5

1 10 12 7 13 7 5

13 11 10 8 10 12 13

- 样例输出

5

9

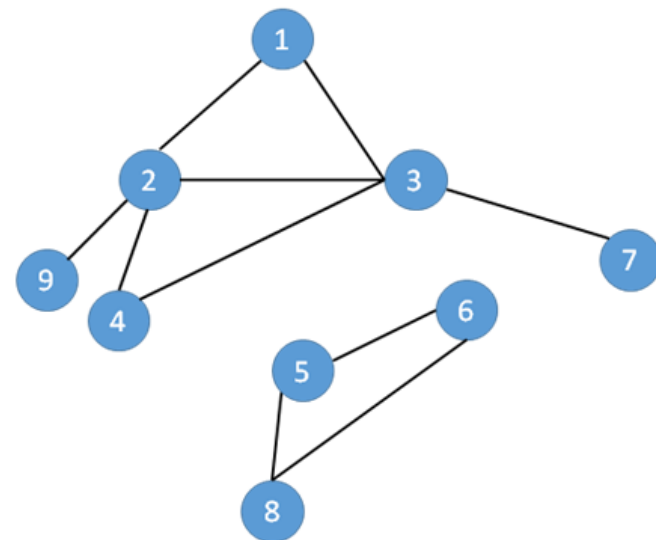
数据保证城堡
四周都是墙

```
      1   2   3   4   5   6   7
#####
1 #   |   #   |   #   |   #
#####---#####---#---#####---#
2 #   #   |   #   #   #   #   #
#---#####---#####---#####---#
3 #   |   |   #   #   #   #   #
#---#####---#####---#---#
4 #   #   |   |   |   |   #   #
#####
      (图 1)

#   = Wall
|   = No wall
-   = No wall
```

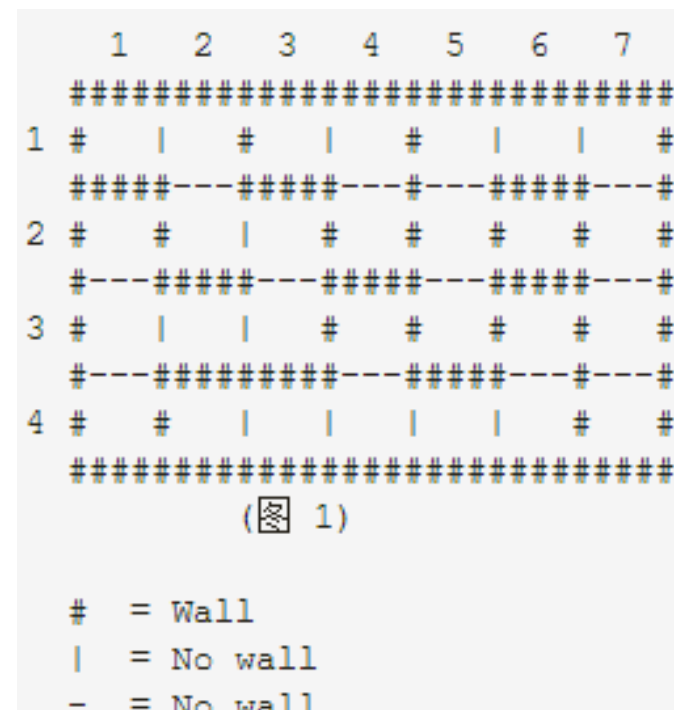
解题思路

- 把方块看作是节点，相邻两个方块之间如果没有墙，则在方块之间连一条边，这样城堡就能转换成一个图。
- 求房间个数，实际上就是在求图中有多少个极大连通子图。
- 一个连通子图，往里头加任何一个图里的其他点，就会变得不连通，那么这个连通子图就是极大连通子图。（如：(8, 5, 6)）



解题思路

- 对每一个房间，深度优先搜索，从而给这个房间能够到达的所有位置染色。最后统计一共用了几种颜色，以及每种颜色的数量。
 - 比如
- | | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 2 | 3 | 3 | 3 |
| 1 | 1 | 1 | 2 | 3 | 4 | 3 |
| 1 | 1 | 1 | 5 | 3 | 5 | 3 |
| 1 | 5 | 5 | 5 | 5 | 5 | 3 |
- 从而一共有5个房间，最大的房间（1）占据9个格子



```
#include <iostream>
#include <stack>
#include <cstring>
using namespace std;
int R,C;    //行列数
int rooms[60][60];
int color[60][60]; //方块是否染色过的标记
int maxRoomArea = 0, roomNum = 0;
int roomArea;
void Dfs(int i,int k) {
    if( color[i][k] )
        return;
    ++ roomArea;
    color [i][k] = roomNum;
    if( (rooms[i][k] & 1) == 0 ) Dfs(i,k-1);    //向西走
    if( (rooms[i][k] & 2) == 0 ) Dfs(i-1,k);    //向北
    if( (rooms[i][k] & 4) == 0 ) Dfs(i,k+1);    //向东
    if( (rooms[i][k] & 8) == 0 ) Dfs(i+1,k);    //向南
}
```



```

int main() {
    cin >> R >> C;
    for( int i = 1;i <= R;++i)
        for ( int k = 1;k <= C; ++k)
            cin >> rooms[i][k];
    memset(color,0,sizeof(color));
    for( int i = 1;i <= R; ++i)
        for( int k = 1; k <= C; ++ k) {
            if( !color[i][k] ) {
                ++ roomNum ;    roomArea = 0;
                Dfs(i,k);
                maxRoomArea =
                    max(roomArea,maxRoomArea) ;
            }
        }
    cout << roomNum << endl;
    cout << maxRoomArea << endl;
}

```

复杂度: $O(R*C)$

例题：百练4982 踩方格

有一个方格矩阵，矩阵边界在无穷远处。我们做如下假设：

- a. 每走一步时，只能从当前方格移动一格，走到某个相邻的方格上；
- b. 走过的格子立即塌陷无法再走第二次；
- c. 只能向北、东、西三个方向走；

请问：如果允许在方格矩阵上走 n 步 ($n \leq 20$)，共有多少种不同的方案。2种走法只要有一步不一样，即被认为是不同的方案。

例题：百练4982 踩方格

思路：

递归

从 (i, j) 出发，走 n 步的方案数，等于以下三项之和：

从 $(i+1, j)$ 出发，走 $n-1$ 步的方案数。前提： $(i+1, j)$ 还没走过

从 $(i, j+1)$ 出发，走 $n-1$ 步的方案数。前提： $(i, j+1)$ 还没走过

从 $(i, j-1)$ 出发，走 $n-1$ 步的方案数。前提： $(i, j-1)$ 还没走过

```
#include <iostream>
#include <cstring>
using namespace std;
int visited[30][50];
int ways ( int i,int j,int n)
{
    if( n == 0)
        return 1;
    visited[i][j] = 1;
    int num = 0;
    if( ! visited[i][j-1] )
        num+= ways(i,j-1,n-1);
    if( ! visited[i][j+1] )
        num+= ways(i,j+1,n-1);
    if( ! visited[i+1][j] )
        num+= ways(i+1,j,n-1);
    visited[i][j] = 0;
    return num;
}
```

```

#include <iostream>
#include <cstring>
using namespace std;
int visited[30][50];
int ways ( int i,int j,int n)
{
    if( n == 0)
        return 1;
    visited[i][j] = 1;
    int num = 0;
    if( ! visited[i][j-1] )
        num+= ways(i,j-1,n-1);
    if( ! visited[i][j+1] )
        num+= ways(i,j+1,n-1);
    if( ! visited[i+1][j] )
        num+= ways(i+1,j,n-1);
    visited[i][j] = 0;
    return num;
}

```

	i,j,n	
i-1,j-1,n	i-1,j,n+1	i-1,j+1,n

```
int main()
{
    int n;
    cin >> n;
    memset(visited,0,sizeof(visited));

    cout << ways(0,25,n) << endl;
    return 0;
}
```