



北京大学
PEKING UNIVERSITY

北京大学暑期课 《ACM/ICPC竞赛训练》

北京大学信息学院 郭炜

guo_wei@PKU.EDU.CN

<http://weibo.com/guoweiofpku>

课程网页: http://acm.pku.edu.cn/summerschool/pku_acm_train.htm

练习网址: <http://acmicpc.openjudge.cn/>



北京大学
PEKING UNIVERSITY

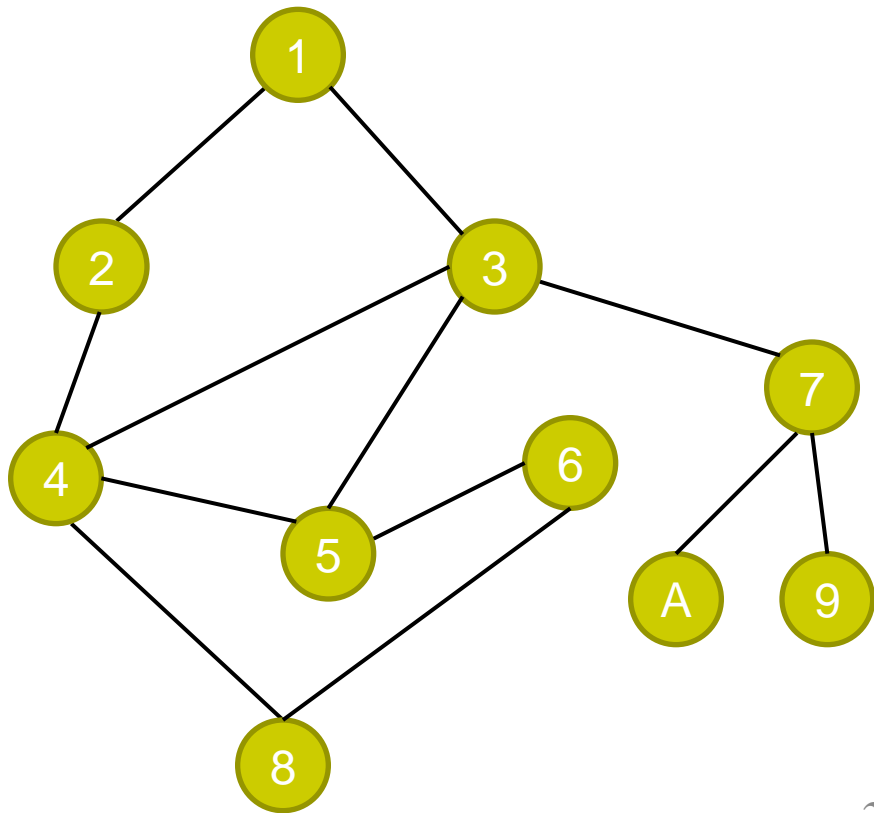
深度优先搜索

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？

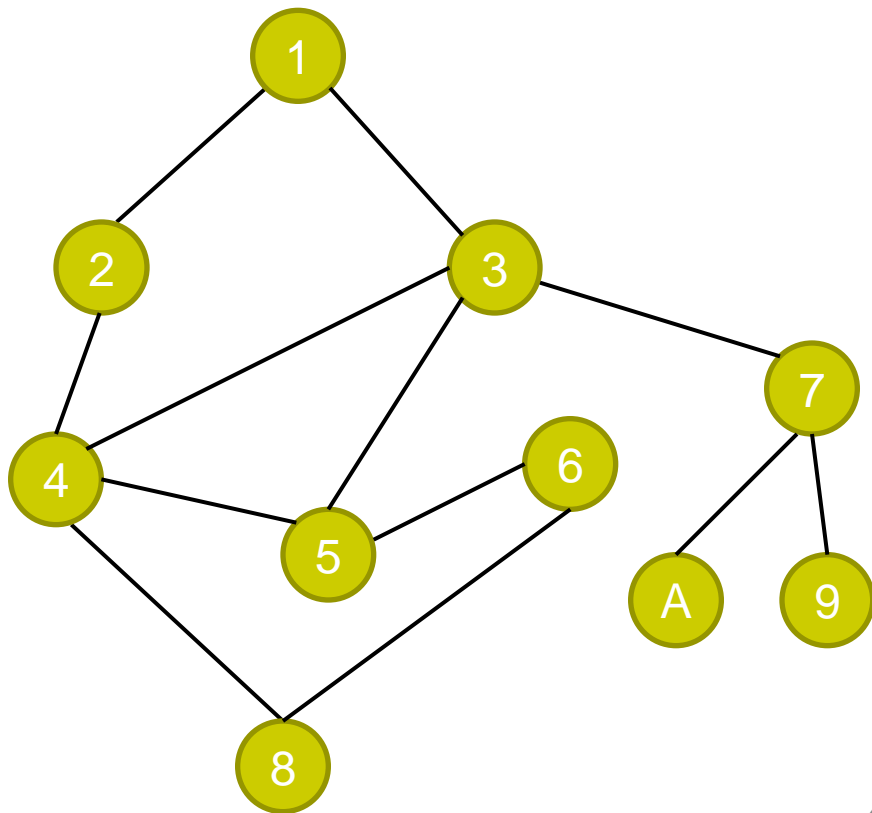
一种策略：只要能发现没走过的点，就走到它。有多个点可走就随便挑一个，如果无路可走就回退，再看有没有没走过的点可走



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

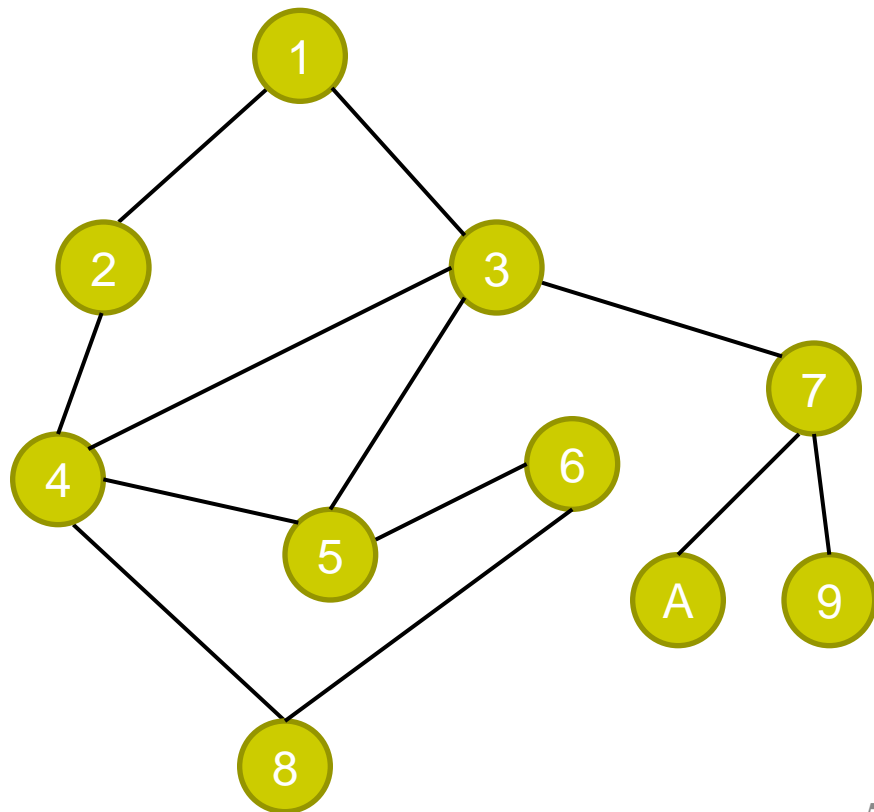


在图上寻找路径

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

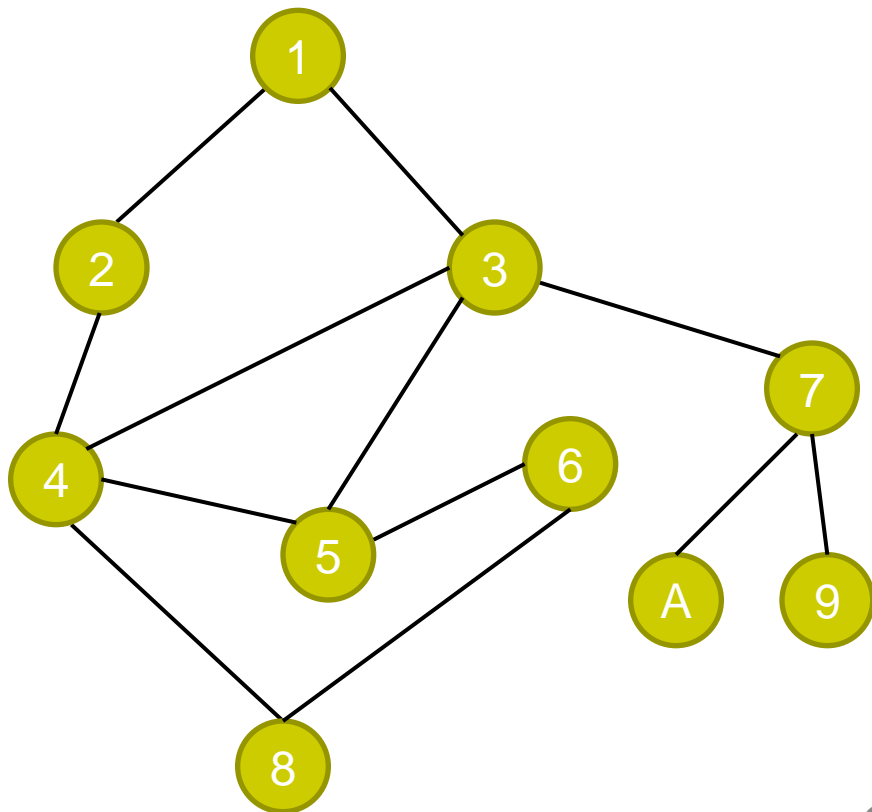
在图上如何寻找从1到8的路径？

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

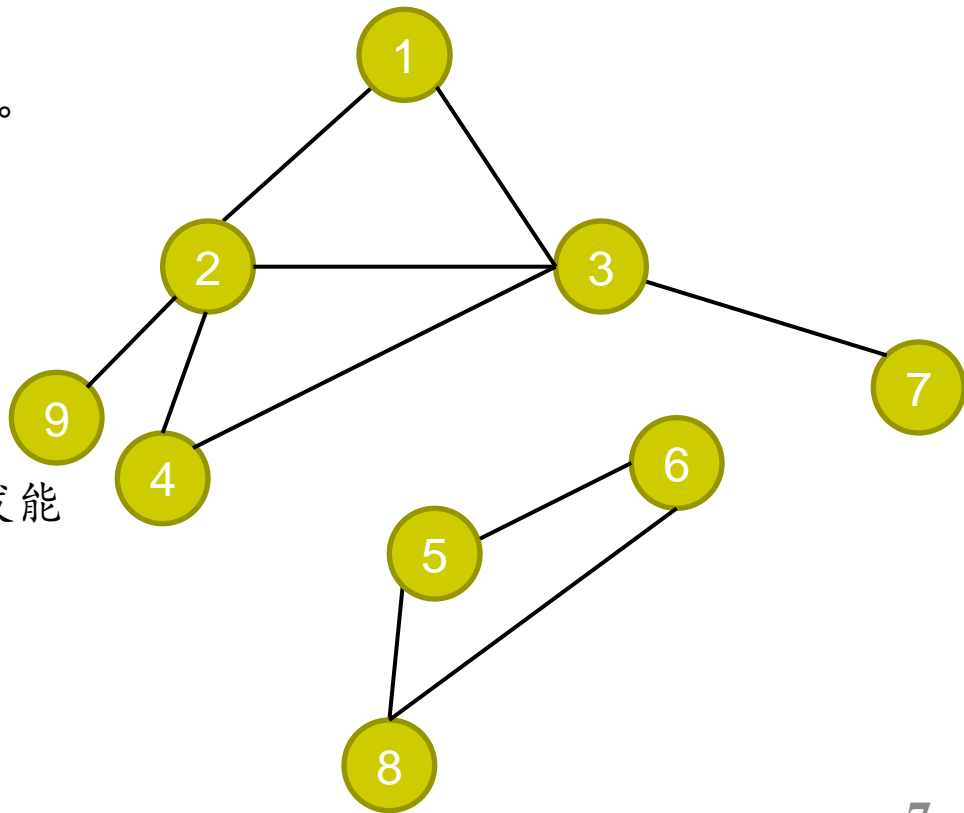
不连通的图，无法从节点1走到节点8。

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

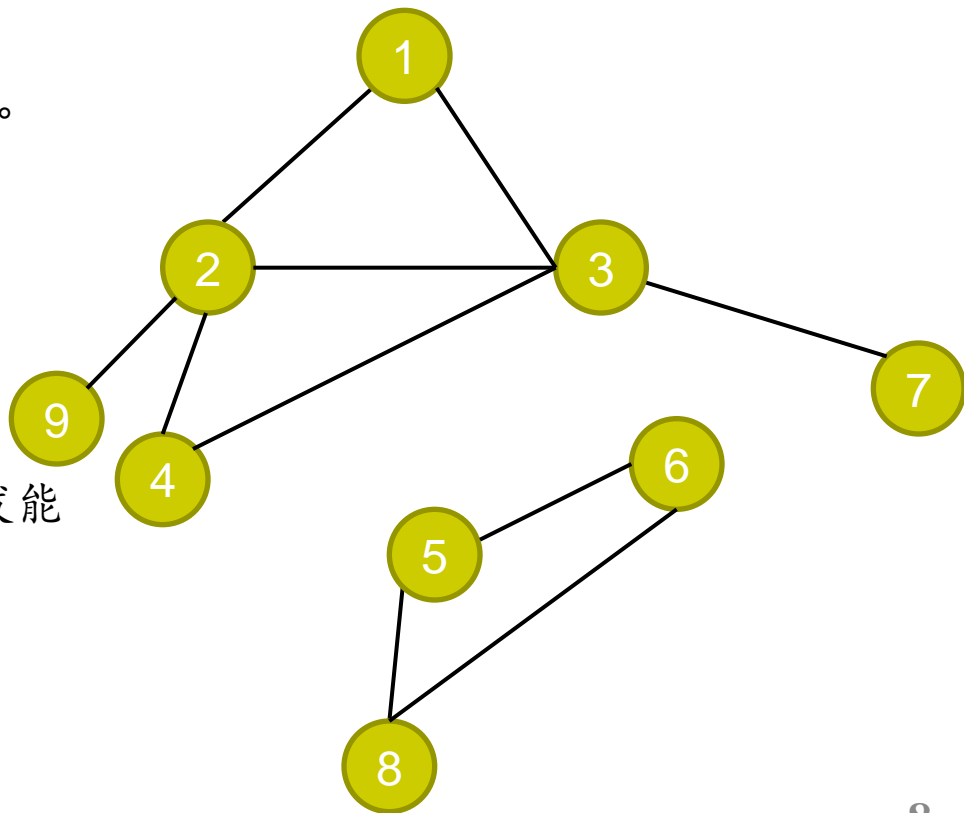
不连通的图，无法从节点1走到节点8。

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;
```

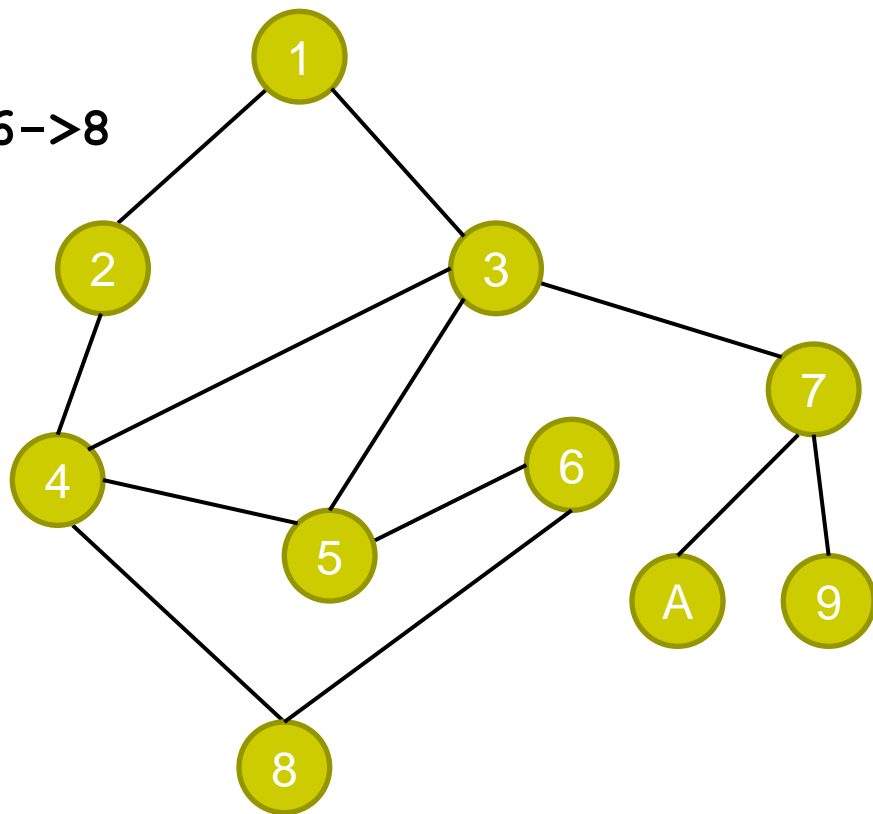
```
    }
```

```
}
```

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)

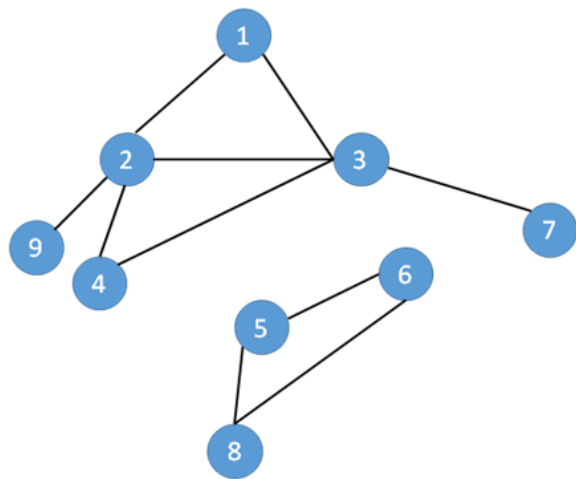
cout << **bestPath[i]** << endl;

}

}

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)。

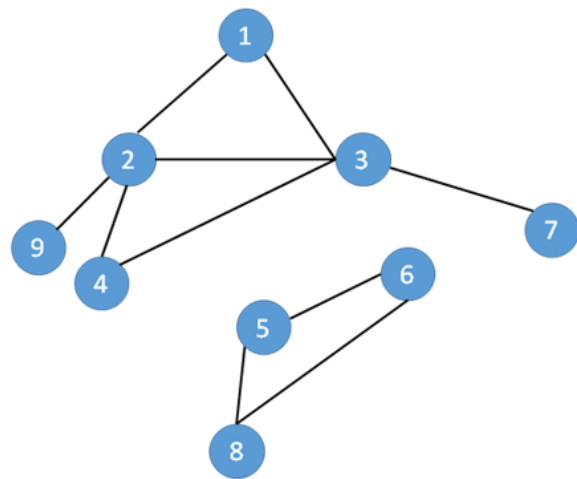
遍历复杂度： $O(n^2)$ n为节点数目

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。

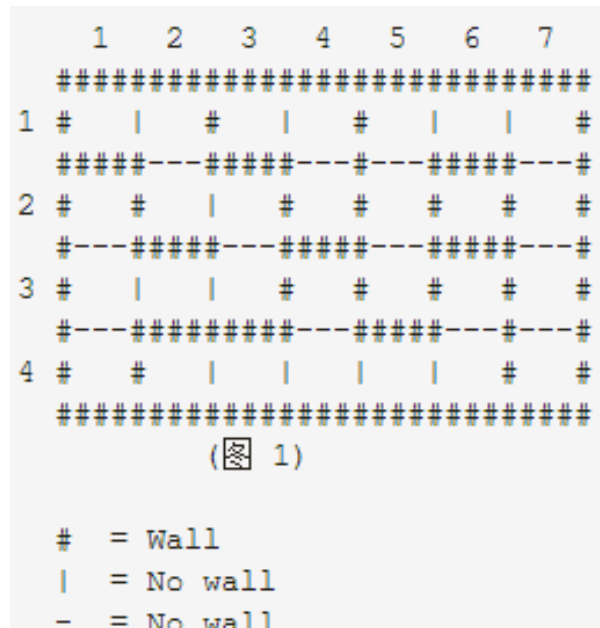
1	2	3		
2	1	4	9	3
3	1	4	7	2
4	2	3		
5	6	8		
6	5	8		
7	3			
8	5	6		
9	2			

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。



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

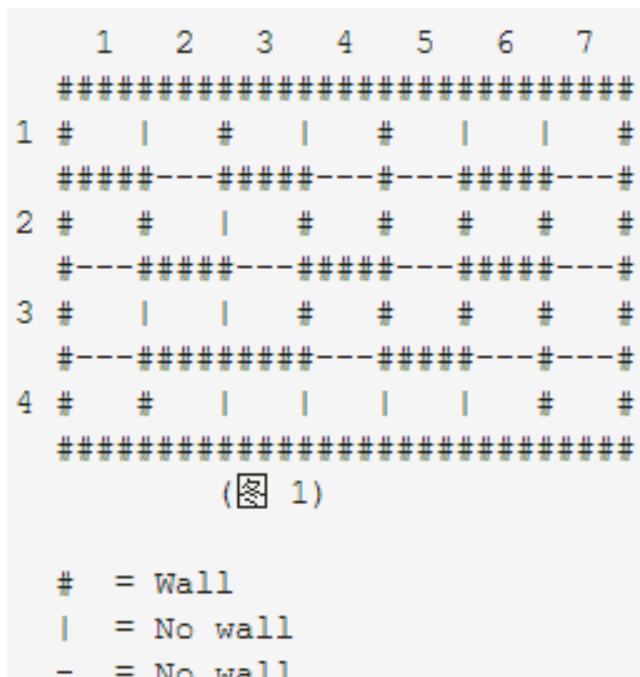
13 11 10 8 10 12 13

- 样例输出

5

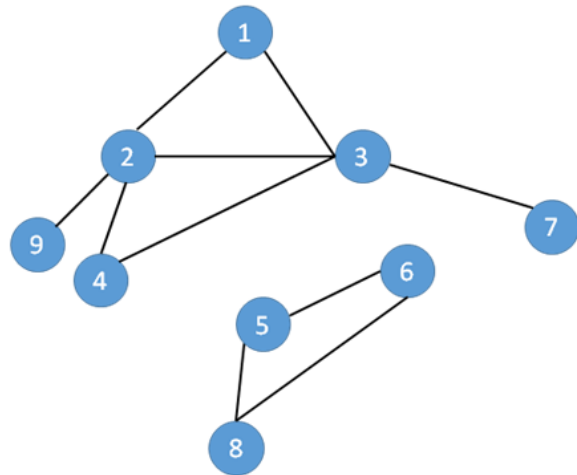
9

数据保证城堡
四周都是墙



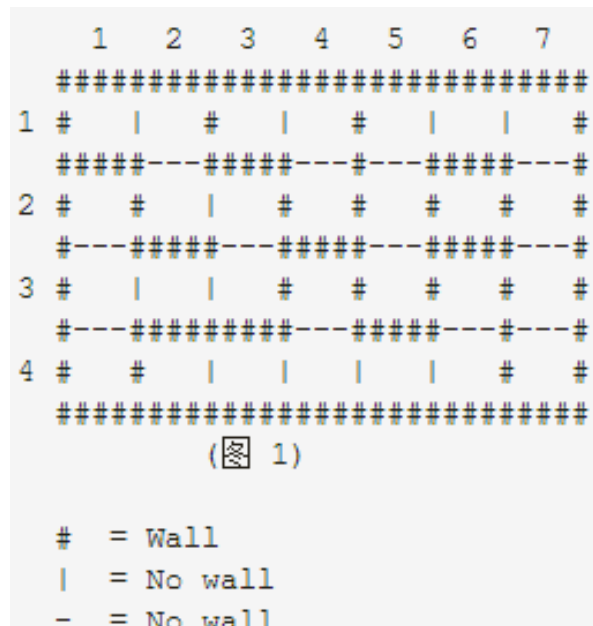
解题思路

- 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;
}
```



深度优先搜索

寻路问题

ROADS (POJ1724)

N个城市，编号1到N。城市间有R条单向道路。

每条道路连接两个城市，有长度和过路费两个属性。

Bob只有K块钱，他想从城市1走到城市N。问最短共需要走多长的路。如果到不了N，输出-1

$2 \leq N \leq 100$

$0 \leq K \leq 10000$

$1 \leq R \leq 10000$

每条路的长度 L ， $1 \leq L \leq 100$

每条路的过路费 T ， $0 \leq T \leq 100$

输入：

K

N

R

$s_1 e_1 L_1 T_1$

$s_1 e_2 L_2 T_2$

...

$s_R e_R L_R T_R$

$s e$ 是路起点和终点

解题思路

从城市 1 开始深度优先遍历整个图，找到所有能过到达 N 的走法，选一个最优的。

解题思路

从城市 1 开始深度优先遍历整个图，找到所有能过到达 N 的走法，选一个最优的。

最优性剪枝：

1) 如果当前已经找到的最优路径长度为 L ，那么在继续搜索的过程中，总长度已经大于等于 L 的走法，就可以直接放弃，不用走到底了

解题思路

从城市 1 开始深度优先遍历整个图，找到所有能到达 N 的走法，选一个最优的。

最优性剪枝：

- 1) 如果当前已经找到的最优路径长度为 L ，那么在继续搜索的过程中，总长度已经大于等于 L 的走法，就可以直接放弃，不用走到底了

保存中间计算结果用于最优性剪枝：

- 2) 用 $midL[k][m]$ 表示：走到城市 k 时总过路费为 m 的条件下，最优路径的长度。若在后续的搜索中，再次走到 k 时，如果总路费恰好为 m ，且此时的路径长度已经超过 $midL[k][m]$ ，则不必再走下去了。

解题思路

另一种通用的最优性剪枝思想 ---保存中间计算结果用于最优性剪枝:

- 2) 如果到达某个状态A时,发现前面曾经也到达过A,且前面那次到达A所花代价更少,则剪枝。这要求保存到达状态A的到目前为止的最少代价。

用 $midL[k][m]$ 表示:走到城市k时总过路费为m的条件下,最优路径的长度。若在后续搜索中,再次走到k时,如果总路费恰好为m,且此时的路径长度已经超过 $midL[k][m]$,则不必再走下去了。

```
#include <iostream>
#include <vector>
#include <cstring>
using namespace std;
int K,N,R;
struct Road {
    int d,L,t;
};
vector<vector<Road> > cityMap(110); //邻接表。cityMap[i]是从点i有路
连到的城市集合
int minLen = 1 << 30; //当前找到的最优路径的长度
int totalLen; //正在走的路径的长度
int totalCost ; //正在走的路径的花销
int visited[110]; //城市是否已经走过的标记
int minL[110][10100]; //minL[i][j]表示从1到i点的，花销为j的最短路的
长度
```

```
void Dfs(int s) //从 s开始向N行走
{
    if( s == N ) {
        minLen = min(minLen, totalLen);
        return ;
    }
    for( int i = 0 ; i < cityMap[s].size(); ++i ) {
        int d = cityMap[s][i].d; //s 有路连到d
        if(! visited[d] ) {
            int cost = totalCost + cityMap[s][i].t;
            if( cost > K)
                continue;
            if( totalLen + cityMap[s][i].L >= minLen ||
                totalLen + cityMap[s][i].L >= minL[d][cost])
                continue;
        }
    }
}
```

```
        totalLen += cityMap[s][i].L;
        totalCost +=  cityMap[s][i].t;
        minL[d][cost] = totalLen;
        visited[d] = 1;
        Dfs(d);
        visited[d] = 0;
        totalCost -=  cityMap[s][i].t;
        totalLen -= cityMap[s][i].L;
    }
}
}
```



```
int main()
{
    cin >>K >> N >> R;
    for( int i = 0;i < R; ++ i) {
        int s;
        Road r;
        cin >> s >> r.d >> r.L >> r.t;
        if( s != r.d )
            cityMap[s].push_back(r);
    }
    for( int i = 0;i < 110; ++i )
        for( int j = 0; j < 10100; ++ j )
            minL[i][j] = 1 << 30;
    memset(visited,0,sizeof(visited));
    totalLen = 0;
    totalCost = 0;
    visited[1] = 1;
```

```
minLen = 1 << 30;  
Dfs(1);  
if( minLen < (1 << 30))  
    cout << minLen << endl;  
else  
    cout << "-1" << endl;  
}
```



北京大学
PEKING UNIVERSITY

深度优先搜索

生日蛋糕

生日蛋糕 (POJ1190)

要制作一个体积为 $N\pi$ 的 M 层生日蛋糕，每层都是一个圆柱体。

设从下往上数第 i ($1 \leq i \leq M$)层蛋糕是半径为 R_i ，高度为 H_i 的圆柱。当 $i < M$ 时，要求 $R_i > R_{i+1}$ 且 $H_i > H_{i+1}$ 。

由于要在蛋糕上抹奶油，为尽可能节约经费，我们希望蛋糕外表面（最下一层的下底面除外）的面积 Q 最小。

$$\text{令 } Q = S\pi$$

请编程对给出的 N 和 M ，找出蛋糕的制作方案（适当的 R_i 和 H_i 的值），使 S 最小。
（除 Q 外，以上所有数据皆为正整数）

解题思路

- 深度优先搜索，枚举什么？

解题思路

- 深度优先搜索，枚举什么？
枚举每一层可能的高度和半径。
- 如何确定搜索范围？

解题思路

- 深度优先搜索，枚举什么？
枚举每一层可能的高度和半径。
- 如何确定搜索范围？
底层蛋糕的最大可能半径和最大可能高度
- 搜索顺序，哪些地方体现搜索顺序？

解题思路

- 深度优先搜索，枚举什么？
枚举每一层可能的高度和半径。
- 如何确定搜索范围？
底层蛋糕的最大可能半径和最大可能高度
- 搜索顺序，哪些地方体现搜索顺序？
从底层往上搭蛋糕，而不是从顶层往下搭
在同一层进行尝试的时候，半径和高度都是从大到小试
- 如何剪枝？

剪枝

- 剪枝1：搭建过程中发现已建好的面积已经超过目前求得的最优表面积，或者预见搭建完后面积一定会超过目前最优表面积，则停止搭建
(最优性剪枝)

剪枝

- 剪枝1：搭建过程中发现已建好的面积已经超过目前求得的最优表面积，或者预见到搭完后面积一定会超过目前最优表面积，则停止搭建
(最优性剪枝)
- 剪枝2：搭建过程中预见到再往上搭，高度已经无法安排，或者半径已经无法安排，则停止搭建(可行性剪枝)

剪枝

- 剪枝1：搭建过程中发现已建好的面积已经**不小于**目前求得的最优表面积，或者预见到搭完后面积一定会超过目前最优表面积，则停止搭建
(**最优性剪枝**)
- 剪枝2：搭建过程中预见到再往上搭，高度已经无法安排，或者半径已经无法安排，则停止搭建(**可行性剪枝**)
- 剪枝3：搭建过程中发现还没搭的那些层的体积，一定会超过还缺的体积，则停止搭建(**可行性剪枝**)

剪枝

- 剪枝1：搭建过程中发现已建好的面积已经**不小于**目前求得的最优表面积，或者预见到搭完后面积一定会**不小于**目前最优表面积，则停止搭建（**最优性剪枝**）
- 剪枝2：搭建过程中预见到再往上搭，高度已经无法安排，或者半径已经无法安排，则停止搭建（**可行性剪枝**）
- 剪枝3：搭建过程中发现还没搭的那些层的体积，一定会超过还缺的体积，则停止搭建（**可行性剪枝**）
- 剪枝4：搭建过程中发现还没搭的那些层的体积，最大也到不了还缺的体积，则停止搭建（**可行性剪枝**）

```

#include <iostream>
#include <vector>
#include <cstring>
#include <cmath>
using namespace std;

int N,M;

int minArea = 1 << 30; //最优表面积
int area = 0; //正在搭建中的蛋糕的表面积
int minV[30]; // minV[n]表示n层蛋糕最少的体积
int minA[30]; // minA[n]表示n层蛋糕的最少侧表面积
int main()
{
    cin >> N >> M ;//M层蛋糕，体积N
    minV[0] = 0;
    minA[0] = 0;
    for( int i = 1; i<= M; ++ i) {
        minV[i] = minV[i-1] + i * i * i; //第i层半径至少i,高度至少i
        minA[i] = minA[i-1] + 2 * i * i;
    }
    if( minV[M] > N )
        cout << 0 << endl;
}

```

```

else {
    int maxH = (N - minV[M-1]) / (M*M) + 1; //底层最大高度
    //最底层体积不超过 (N-minV[M-1]), 且半径至少M
    int maxR = sqrt(double(N-minV[M-1]) / M) + 1; //底层高度至少M
    area = 0;
    minArea = 1 << 30;
    Dfs( N, M, maxR, maxH );
    if( minArea == 1 << 30 )
        cout << 0 << endl;
    else
        cout << minArea << endl;
}
}

```

```
void Dfs(int v, int n,int r,int h)
//要用n层去凑体积v,最底层半径不能超过r,高度不能超过h
//求出最小表面积放入 minArea
{
    if( n == 0 ) {
        if( v ) return;
        else {
            minArea = min(minArea,area);
            return;
        }
    }
    if( v <= 0)
        return ;
    if( minV[n] > v ) //剪枝3
        return ;
    if( area + minA[n] >= minArea) //剪枝1
        return ;
    if( h < n || r < n ) //剪枝2
        return ;
```

```

if( MaxVforNRH(n,r,h) < v )    //剪枝4
//这个剪枝最强！没有的话，5秒都超时，有的话，10ms过！
    return;
//for( int rr = n; rr <= r; ++ rr ) { 这种写法比从大到小慢5倍
for( int rr = r; rr >=n; -- rr ) {
    if( n == M ) //底面积
        area = rr * rr;
    for( int hh = h; hh >= n ; --hh ) {
        area += 2 * rr * hh;
        Dfs(v-rr*rr*hh,n-1,rr-1,hh-1);
        area -= 2 * rr * hh;
    }
}
}

```



```
int MaxVforNRH(int n,int r,int h)
{ //求在n层蛋糕，底层最大半径r，最高高度h的情况下，能凑出来的最大体积
    int v = 0;
    for( int i = 0; i < n ; ++ i )
        v += (r - i ) *(r-i) * (h-i);
    return v;
}
```

还有什么可以改进

还有什么可以改进

- 1) 用数组存放 $\text{MaxVforNRH}(n, r, h)$ 的计算结果，避免重复计算

还有什么可以改进

1) 用数组存放 `MaxVforNRH(n,r,h)` 的计算结果，避免重复计算

2)

```
for( int rr = r; rr >=n; -- rr ) {  
    if( n == M ) //底面积  
        area = rr * rr;  
    for( int hh = h; hh >= n ; --hh ) {  
        area += 2 * rr * hh;  
        Dfs(v-rr*rr*hh,n-1,rr-1,hh-1);  
        //加上对本次Dfs失败原因的判断。如果是因为剩余体积不够大而失败，那么就用不着试下一个  
        //高度，直接break; 或者由小到大枚举 h.....  
        area -= 2 * rr * hh;  
    }  
}
```