第 3 部分 数据结构与算法基础班

3.1 图的存储与遍历（赵宗昌）

知识点：

1.图的邻接矩阵存储方法。

2.图的遍历:深度优先（dfs）与广度优先（bfs）。

3.无向图的连通分量。

4.图的遍历的简单应用。

3.1.1 知识讲解

引例 1：公司数量

【问题描述】

在某个城市里住着 n 个人，现在给定关于 n 个人的 m 条信息（即某 2 个人认识），假设所有认识（直接或间接认识都算认识）的人一定属于同一个公司。

若是某两人不在给出的信息里，那么他们不认识，属于两个不同的公司。

已知人的编号从 1 至 n。

请计算该城市最多有多少公司。

【输入】

第一行：n（<=10000,人数），

第二行：m（<=100000，信息）

以下 m 行：每行两个数：i 和 j，中间一个空格隔开，表示 i 和 j 相互认识。

【输出】

公司的数量。

【输入输出样例】

city.in	city.out
11 9 1 2 4 5 3 4 1 3 5 6 7 10 5 10 6 10 8 9	3

【数据规模】

100%个数据：n<=10000， m<=100000。

分析：

把 n 个人看成 n 个独立的点，如果 i 和 j 相互认识，在点 i 和 j 之间连一条没有方向的边。样例如下：

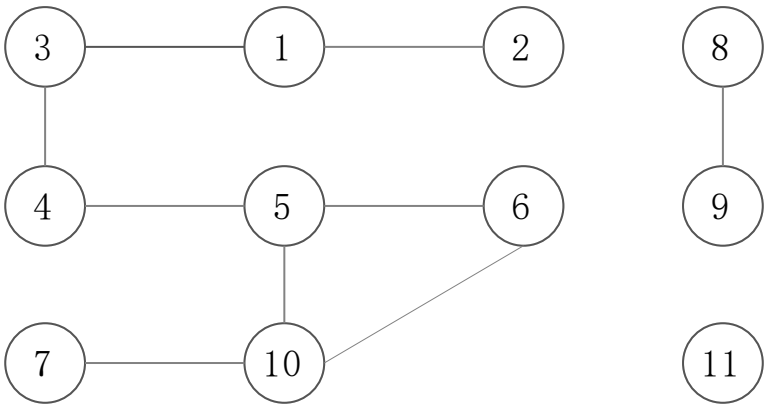


图 1

根据上图，很容易看出该城市有 3 个公司，对应图 1 中的 3 个独立子图。

问题变为：如何保存这个图？如何求该图由几独的子图构成？

引例 2：安排座位

【问题描述】

已知 n(<20)个人围着一个圆桌吃饭，其中每一个人都至少认识其他的 2 个客人。请设计程序求得 n 个人的一种坐法，使得每个人都认识他左右的客人。

【输入】

第一行:n（吃饭人的个数）。

以下 n 行：第 i 行的第一个数 k 表示第 i 个人认识的人数，后面 k 个数依次为 i 认识的人的编号。

【输出】

所有座法，要求第一个人为 1 号作为起点，按顺时针输出其它人的编号。

【输入输出样例】

seat.in	seat.out
6 2 3 6 3 4 5 6 3 1 4 6 3 2 3 5 3 2 4 6 4 1 2 3 5	1 3 4 2 5 6 1 3 4 5 2 6 1 6 2 5 4 3 1 6 5 2 4 3 4

分析：

类似引例 1，把每个人看作一个顶点，相互认识的两个人用一条无方向的边连接。样例建如下所示的图：

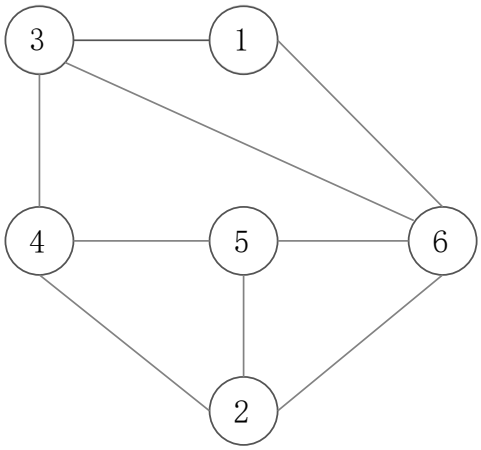


图 2

根据上述图 2，容易找出吃饭的其中一种座法，如：1 3 4 2 5 6。

以上两个引例问题的解决方法：

把每个人看成一个顶点，有关系的两个人连一条边，构成图的结构。

引例 1 的问题变为：求图由几个独立的子图构成。

引例 2 的问题变为：从其中的顶点 1 出发，沿着边经过所有的点走一遍（不能重复走点），最后回到 1。

解决上述问题需要掌握图的两个最基本的问题：图的存储方法与图的遍历。

1.无向图与有向图：

图（Graph）是由顶点集合和顶点间的关系集合（边集）组成的数据结构，通常用二元组 G(V,E)表示图，V 表示顶点集，顶点元素经常用 u，v 等符合表示，顶点的个数通常用 n 表示；E 表示边的集合，边的元素经常用 e 等符号表示，边的数量通常用 m 表示。

如图 2：顶点集 V={1,2,3,4,5,6}，边集 E={(1,3),(1,6),(2,4),(2,5),(2,6),(3,4),(3,6),(4,5),(5,6)}。

在边集 E 中，每个元素(u,v)是两个顶点组成的无序对（用圆括号括起来），表示顶点 u 和 v 相关联的一条无向边。（u,v）和（v,u）表示同一条边。如果图中所有的边都没有方向，这种图称为**无向图**。

下列图 3 的表示：V={1,2,3},E={<1,2>,<1,3>,<2,3>}。其中 E 的每个元素<u,v>是一对顶点的有序对（用尖括号括起来），表示从顶点 u 到顶点 v 的一条有向边，u 是起点，v 是终点，所以<u,v>和<v,u>不是同一条边。如果图中所有的边都有方向的，这种图称为**有向图**。

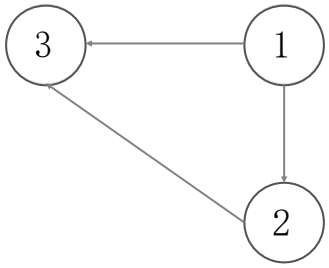


图 3

2.图的存储方法

一个图需要保存两个信息，一个是顶点信息，一个是顶点间的信息（边）。

顶点信息：一维数组即可。如果是 1 到 n，无需保存，直接用 1 到 n 表示就行了。

顶点间关系的存储常用的方法有两种：邻接矩阵与邻接表，这里只介绍简单的邻接矩阵。

顶点间的关系，用矩阵表示，称为**邻接矩阵**。

设 $G(V, E)$是一个具有 n 个顶点的图，则图的邻接矩阵是一个 $n \times n$ 的二维数组，定义为：

$$a[i][j]=\begin{cases}1 & 如果<i,j>\in E，或(i,j)\in E\\ 0 & 否则\end{cases}$$

图 2 的邻接矩阵：

$$a[6][6]=\begin{bmatrix}0&0&1&0&0&1\\0&0&0&1&1&1\\1&0&0&1&0&1\\0&1&1&0&1&0\\0&1&0&1&0&1\\1&1&1&0&1&0\end{bmatrix}$$

图 3 的邻接矩阵：

$$a[3][3]=\begin{bmatrix}0&1&1\\0&0&1\\0&0&0\end{bmatrix}$$

3.图的遍历

图的遍历是图论算法的基础，图的遍历（graph traversal），也称图的搜索（ search），就是从图中某个顶点出发，沿着一些边访遍图中所有的顶点，且使每个顶点仅被访问一次。

图的遍历可以采取两种方法进行：深度优先搜索（ DFS： depth first search）和广度优先搜索（ BFS：breadth first search）。

（1）DFS 遍历

深度优先搜索是一个递归过程，有回退过程。

DFS 算法思想：

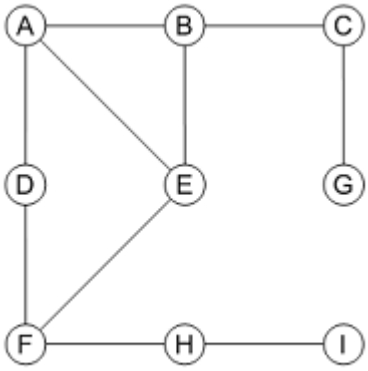
对于一个连通图，在访问图中某一起始顶点 u 后，由 u 出发，访问它的某一邻接顶点 v1；再从 v1 出发，访问与 v1 邻接但还没有访问过的顶点 v2；然后再从 v2 出发，进行类似的访问；…；如此进行下去，直至到达所有邻接顶点都被访问过为止；接着，回退一步，回退到前一次刚访问过的顶点 x，看是否还有 x 的其它没有被访问过的邻接顶点 y，如果有，则访问此顶点 y，之后再从此顶点 y 出发，进行与前述类似的访问；如果没有，就再回退一步进行类似的访问。重复上述过程，直到该连通图中所有顶点都被访问过为止。

如下列(a)无向图 G：

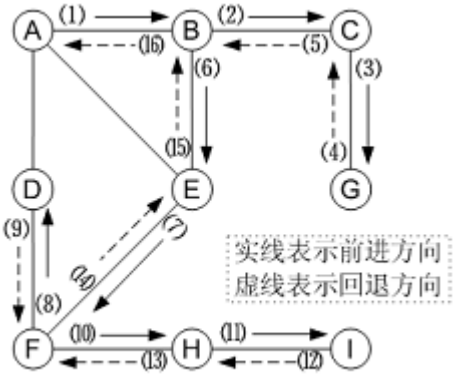
顶点数组：

1	2	3	4	5	6	7	8	9
A	B	C	D	E	F	G	H	I

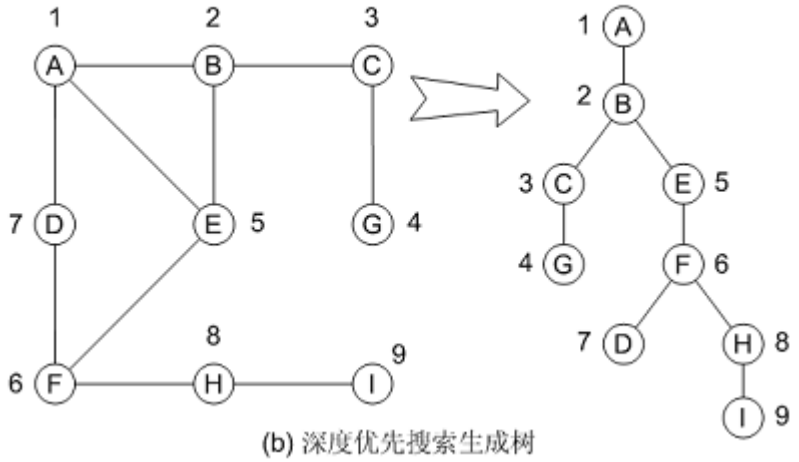
假设在多个未被访问的邻接顶点中进行选择时，按顶点序号从小到大的顺序进行选择。选择从编号最小的 A 开始：



(a) 无向图*G*



(a) 深度优先过程

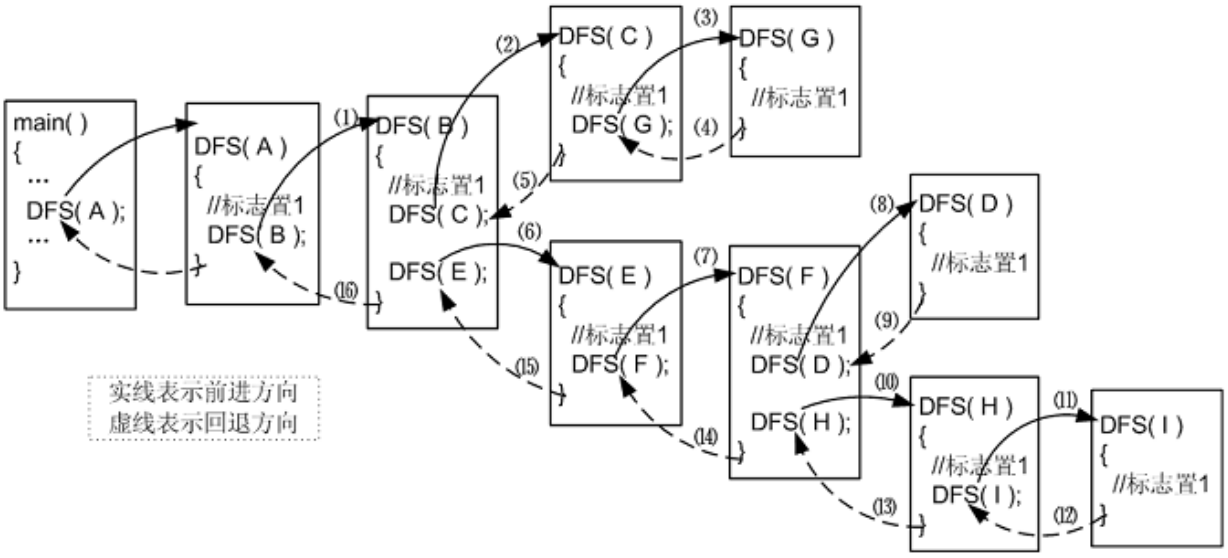


(b) 深度优先搜索生成树

每个顶点外侧的数字标明了进行深度优先搜索时各顶点访问的次序，称为顶点的深度优先数。图(b)给出了访问 n 个顶点时经过的 n-1 条边，这 n-1 条边将 n 个顶点连接成一棵树，称此图为原图的深度优先生成树，该树的根结点就是深度优先搜索的起始顶点。

上图 DFS 遍历的过程（递归实现）：

为避免重复访问，需要一个状态数组 visited[n]，用来存储各顶点的访问状态。如果 visited[i] = 1，则表示顶点 i 已经访问过；如果 visited[i] = 0，则表示顶点 i 还未访问过。初始时，各顶点的访问状态均为 0。



DFS 算法框架：

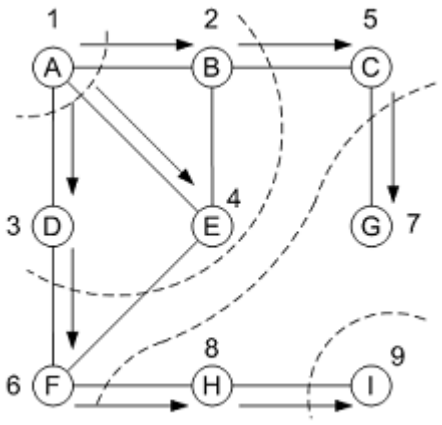
```
DFS( 顶点 i ) //从顶点 i 进行深度优先搜索{
    visited[ i ] = 1; //将顶点 i 的访问标志置为 1
    for( j=1; j<=n; j++ ) //对其他所有顶点 j{
        //j 是 i 的邻接顶点，且顶点 j 没有访问过
        if( a[i][j]==1 && !visited[j] ){
            //递归搜索前的准备工作需要在这里写代码
            DFS( j ) //从顶点 j 出发进行 DFS 搜索
            //以下是 DFS 的回退位置，在很多应用中需要在这里写代码
        }
    }
}
```

（1）BFS 遍历

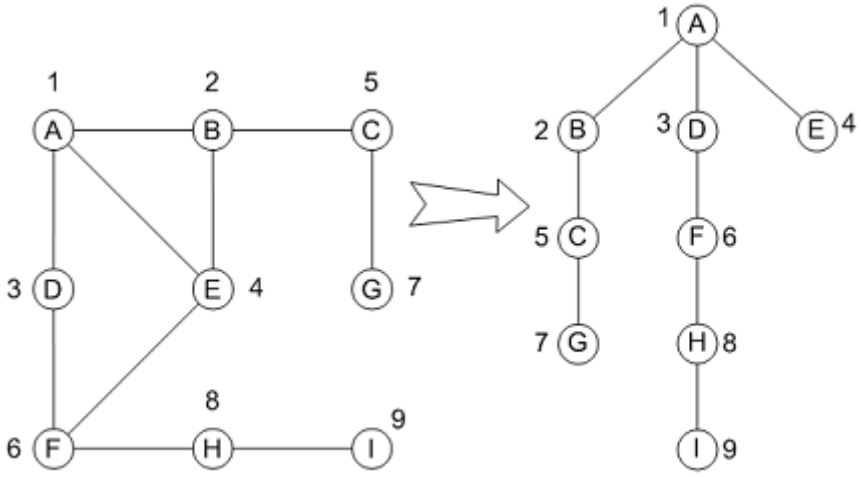
广度优先搜索（ BFS， Breadth First Search）是一个分层的搜索过程，没有回退过程，是非递归的。

BFS 算法思想:

对一个连通图，在访问图中某一起始顶点 u 后，由 u 出发，依次访问 u 的所有未访问过的邻接顶点 v1, v2, v3, …vt；然后再顺序访问 v1, v2, v3, …vt 的所有还未访问过的邻接顶点；再从这些访问过的顶点出发，再访问它们的所有还未访问过的邻接顶点， ……，如此直到图中所有顶点都被访问到为止。



(a) 广度优先过程

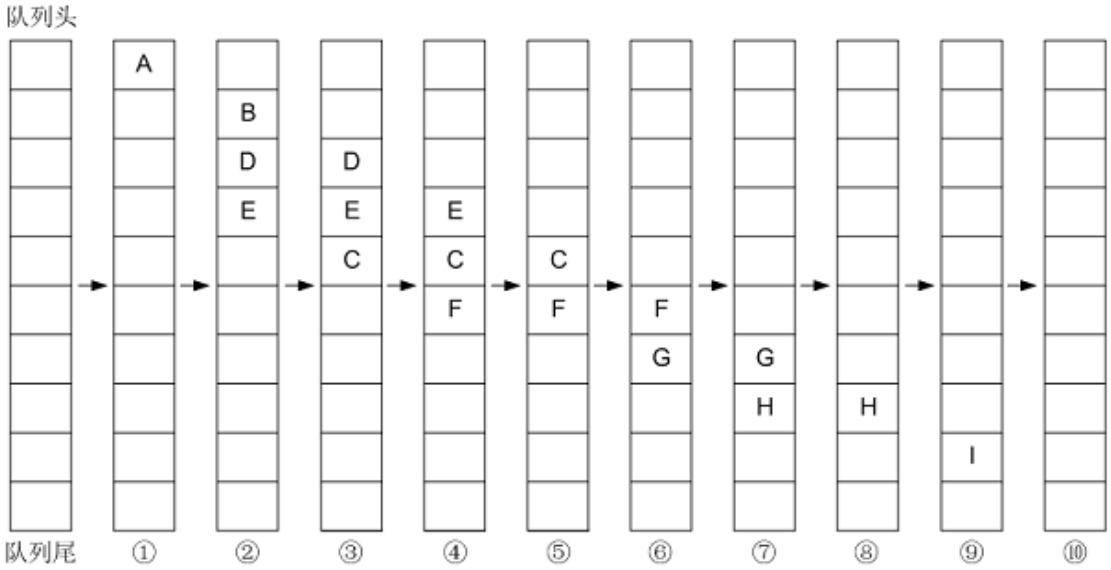


(b) 广度优先搜索生成树

BFS 算法的实现：

与深度优先搜索过程一样，为避免重复访问，也需要一个状态数组 visited[n]，用来存储各顶点的访问状态。如果 visited[i] = 1，则表示顶点 i 已经访问过；如果 visited[i] = 0，则表示顶点 i 还未访问过。初始时，各顶点的访问状态均为 0。

为了实现逐层访问， BFS 算法在实现时需要使用一个队列，来记忆正在访问的这一层和上一层的顶点，以便于向下一层访问（约定在队列中，取出元素的一端为队列头，插入元素的一端为队列尾,初始时，队列为空）：



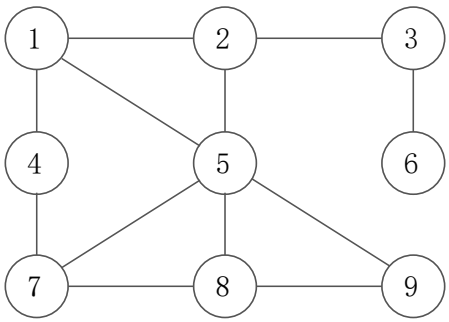
BFS 算法：

```
BFS( 顶点 i ) //从顶点 i 进行广度优先搜索{
    visited[ i ] = 1; //将顶点 i 的访问标志置为 1
    将顶点 i 入队列;
    while( 队列不为空 ){
        取出队列头的顶点，设为 k
        for( j=0; j<n; j++ ) //对其他所有顶点 j{
            //j 是 k 的邻接顶点，且顶点 j 没有访问过
```

```
            if( a[k][j]==1 && !visited[j] ){
                将顶点 j 的访问标志置为 1
                将顶点 j 入队列
            }
        } //end of for
    } //end of while
} //end of BFS
```

【上机实践】

对下图进行存储（邻接矩阵）和遍历（用 DFS 和 BFS 分别实现）。



输入：

第一行：顶点数 n。

第二行：边数 m。

以下 m 行，每行两个顶点编号 u，v。

输入输出样例：

9
12
1 2
1 4
1 5
2 3
2 5
3 6
4 7
5 7
5 8
5 9
7 8
8 9

参考代码：

DFS 算法：

```
#include<cstdio>
#include<iostream>
using namespace std;
int a[1001][1001];
int vis[1001];
int n,m;
```

```
void dfs(int u){
    cout<<u<<endl;
    vis[u]=1;
    for(int i=1;i<=n;i++)
        if(a[u][i]==1&&vis[i]==0) dfs(i);
}
int main(){
    cin>>n>>m;
    for(int i=0;i<m;i++){
        int x,y;
        cin>>x>>y;
        a[x][y]=a[y][x]=1;
    }
    dfs(1);
    return 0;
}
```

BFS 算法：

```
#include<cstdio>
#include<iostream>
using namespace std;
int q[1001];
int a[1001][1001];
int vis[1001];
int n,m;
void bfs(int u){
    int head=0,tail=1;
    q[0]=u;
    vis[u]=1;
    while(head<tail){
        int p=q[head++];
        cout<<p<<endl;
        for(int i=1;i<=n;i++)
            if(a[p][i]==1&&vis[i]==0){
                q[tail++]=i;
                vis[i]=1;
            }
    }
}
int main(){
    cin>>n>>m;
    for(int i=0;i<m;i++){
        int x,y;
        cin>>x>>y;
        a[x][y]=a[y][x]=1;
```

```
    }
    bfs(1);
    return 0;
}
```

说明：上述是对一个连通图（图中任意两点都有路线可达）的遍历方法，可以从任意一个顶点作为起点开始 dfs 或 bfs 都能完成整个连通图的遍历。

4.无向图的连通分量

如果一个图是非连通图，那么一次 dfs 或 bfs 只能遍历一个连通分量。非连通图中极大连通子图的数量称为无向图的连通分量。

求图的连通分量的方法很简单：每次找一个没有遍历的顶点 i 作为起点 dfs(i)即可，调用的次数就是连通分量。

求无向图的连通分量算法：

```
for(int i=1;i<=n;i++)
    if(!vis[i]){
        dfs(i);//或bfs(i)
        cnt++;//连通分量
    }
```

图的遍历是图的算法的基础，掌握了图的遍历后，在此基础上稍加变化，能解决很实际问题。

回到开始的两个引例

引例 1：公司数量

以人为图的顶点，相互认识的建立无向边，求无向图的连通分量。

参考代码：

```
#include<cstdio>
#include<iostream>
using namespace std;
int a[10001][10001]={0};
int vis[10001]={0};
int n,m,cnt=0;
void dfs(int k){
    vis[k]=1;
    for(int i=1;i<=n;i++)
        if(!vis[i]&&a[k][i])dfs(i);
}
int main(){
    scanf("%d%d",&n,&m);
    int x,y;
    for(int i=1;i<=m;i++){
        scanf("%d%d",&x,&y);
        a[x][y]=a[y][x]=1;
    }
    for(int i=1;i<=n;i++)
        if(!vis[i]){
            dfs(i);
            cnt++;
        }
```

```
    cout<<cnt<<endl;
    return 0;
}
```

引例 2：安排座位

根据认识关系建立无向图，从 1 开始遍历图，找一个包含所有顶点的环（最后回到 1）。

参考代码：

```
#include<cstdio>
#include<cstring>
int a[21][101],b[21],v[21],n,ans=0;
void dfs(int step){
    if(step==n&&a[1][b[n]]==1){
        ans++;
        for(int i=1;i<=n;i++) printf("%d ",b[i]);
        printf("\n");
    }
    for(int i=1;i<=n;i++)
        if(v[i]==0&&a[i][b[step]]==1){
            b[step+1]=i;
            v[i]=1;
            dfs(step+1);
            v[i]=0;
        }
}
int main(){
    memset(a,0,sizeof(a));
    memset(v,0,sizeof(v));
    scanf("%d",&n);
    for(int i=1;i<=n;i++){
        int k;
        scanf("%d",&k);
        for(int j=1;j<=k;j++){
            int x;
            scanf("%d",&x);
            a[i][x]=a[x][i]=1;
        }
    }
    b[1]=1;
    v[1]=1;
    dfs(1);
    printf("%d\n",ans);
    return 0;
}
```

3.1.2 典型例题

例 3.1.1 油田(zoj1709， poj1562)

题目描述：

GeoSurvComp 地质探测公司负责探测地下油田。每次 GeoSurvComp 公司都是在一块长方形的土地上来探测油田。在探测时，他们把这块土地用网格分成若干个小方块，然后逐个分析每块土地，用探测设备探测地下是否有油田。方块土地底下有油田则称为 pocket，如果两个 pocket 相邻，则认为是同一块油田，油田可能覆盖多个 pocket。

你的工作是计算长方形的土地上有多少个不同的油田。

输入描述：

输入文件中包含多个测试数据，每个测试数据描述了一个网格。每个网格数据的第一行为两个整数：m n，分别表示网格的行和列；如果 m = 0，则表示输入结束，否则 1≤m≤100，1 ≤n≤100。接下来有 m 行数据，每行数据有 n 个字符（不包括行结束符）。每个字符代表一个小方块，如果为“*”，则代表没有石油，如果为“@”，则代表有石油，是一个 pocket。

输出描述：

对输入文件中的每个网格，输出网格中不同的油田数目。如果两块不同的 pocket 在水平、垂直、或者对角线方向上相邻，则被认为属于同一块油田。每块油田所包含的 pocket 数目不会超过 100。

样例输入：

```
3 5
*@*@*
**@**
*@*@*
5 5
****@
*@@*@
*@**@
@@@*@
@@**@
0 0
```

样例输出：

```
1
2
```

分析：

从网格中某个“@”字符位置开始进行 DFS 搜索，可以搜索到跟该“@”字符位置同属一块油田的所有“@”字符位置。遍历整个图，求图的连通分量。注意是 8 连通。

```
//zoj1709 Oil Deposits
#include<cstdio>
#include<iostream>
#include<cstring>
using namespace std;
int dx[8]={-1,1,0,0,-1,1,-1,1};
int dy[8]={0,0,-1,1,-1,1,1,-1};
int map[110][110];
int n,m,Sx,Sy,Dx,Dy;
void dfs(int x,int y){
    map[x][y]=0;
    for(int i=0;i<8;i++){
```

```
            int xx=x+dx[i],yy=y+dy[i];
            if(xx>=1&&xx<=n&&yy>=1&&yy<=m&&map[xx][yy]==1)
                dfs(xx,yy);
        }
    }
    int main(){
        //freopen("maze.in","r",stdin);
        while((cin>>n>>m)&&(n>0)){
            char ch;
            memset(map,0,sizeof(map));
            for(int i=1;i<=n;i++)
                for(int j=1;j<=m;j++){
                    cin>>ch;
                    map[i][j]=(ch=='@'?1:0);
                }
            int cnt=0;
            for(int i=1;i<=n;i++)
                for(int j=1;j<=m;j++)
                    if(map[i][j]==1){
                        dfs(i,j);
                        cnt++;
                    }
            cout<<cnt<<endl;
        }
        return 0;
    }
```

扩展：如何求最大油田的面积？

例 3.1.2 红与黑(zoj2165　poj1979)

题目描述：

有一个长方形的房间，房间里的地面上布满了正方形的瓷砖，瓷砖要么是红色，要么是黑色。一男子站在其中一块黑色的瓷砖上。男子可以向他四周的瓷砖上移动，但不能移动到红色的瓷砖上，只能在黑色的瓷砖上移动。

编写程序，计算他在这个房间里可以到达的黑色瓷砖的数量。

输入描述：

输入文件中包含多个测试数据。每个测试数据的第 1 行为两个整数 W 和 H，分别表示长方形房间里 x 方向和 y 方向上瓷砖的数目。W 和 H 的值不超过 20。

接下来有 H 行，每行有 W 个字符，每个字符代表了瓷砖的颜色，这些字符的取值及含义为：

1）'.' －　黑色的瓷砖；

2）'#' －　红色的瓷砖；

3）'@' －　表示该位置为黑色瓷砖，且一名男子站在上面，注意每个测试数据中只有一个'@'符号。

输入文件中最后一行为两个 0，代表输入文件结束。

输出描述：

对输入文件中每个测试数据，输出占一行，为该男子从初始位置出发可以到达的黑色瓷砖的数目（包括他初始时所处的黑色瓷砖）。

样例输入：

```
6 9
....#.
.....#
......
......
......
......
......
#@...#
.#..#.
11 9
.#.........
.#.#######.
.#.#.....#.
.#.#.###.#.
.#.#..@#.#.
.#.#####.#.
.#.......#.
.#########.
...........
0 0
```

样例输出：

```
45
59
```

分析：

求男子所在位置能到达的黑色的连通块的大小。

从起点一遍 dfs 即可完成。

```
//zoj2165 Red and Black
#include<cstdio>
#include<iostream>
#include<cstring>
using namespace std;
int dx[4]={-1,1,0,0};
int dy[4]={0,0,-1,1};
int map[110][110];
int n,m,sx,sy,cnt;
void dfs(int x,int y){
    cnt++;
    map[x][y]=0;
    for(int i=0;i<4;i++){
        int xx=x+dx[i],yy=y+dy[i];
        if(xx>=1&&xx<=n&&yy>=1&&yy<=m&&map[xx][yy]==1)
            dfs(xx,yy);
    }
```

```
}
int main(){
    //freopen("maze.in","r",stdin);
    while((cin>>m>>n)&&(n+m>0)){
        char ch;
        memset(map,0,sizeof(map));
        for(int i=1;i<=n;i++)
            for(int j=1;j<=m;j++){
                cin>>ch;
                if(ch=='.') map[i][j]=1;
                if(ch=='@'){sx=i;sy=j;}
            }
        cnt=0;
        map[sx][sy]=0;
        dfs(sx,sy);
        cout<<cnt<<endl;
    }
    return 0;
}
```

例 3.1.3 骨头的诱惑(zoj2110)

题目描述：

一只小狗在一个古老的迷宫里找到一根骨头，当它叼起骨头时，迷宫开始颤抖，它感觉到地面开始下沉。它才明白骨头是一个陷阱，它拼命地试着逃出迷宫.

迷宫是一个 N ×M 大小的长方形，迷宫有一个门。刚开始门是关着的，并且这个门会在第 T 秒钟开启，门只会开启很短的时间（少于一秒），因此小狗必须恰好在第 T 秒达到门的位置。每秒钟，它可以向上、下、左或右移动一步到相邻的方格中。但一旦它移动到相邻的方格，这个方格开始下沉，而且会在下一秒消失。所以，它不能在一个方格中停留超过一秒，也不能回到经过的方格。

小狗能成功逃离吗？请你帮助他。

输入描述：

输入文件包括多个测试数据。每个测试数据的第一行为三个整数：N　M　T，（1<N,　M<7 ；0<T<50 ），分别代表迷宫的长和宽，以及迷宫的门会在第 T 秒时刻开启。

接下来 N 行信息给出了迷宫的格局，每行有 M 个字符，这些字符可能为如下值之一：

X：　墙壁，小狗不能进入　　S:　小狗所处的位置

D：　迷宫的门　　.：　空的方格

输入数据以三个 0 表示输入数据结束。

输出描述：

对每个测试数据，如果小狗能成功逃离，则输出"YES"，否则输出"NO"。

样例输入：

```
3 4 5
S...
.X.X
...D
4 4 8
.X.X
```

```
..S.
....
DX.X
4 4 5
S.X.
..X.
..XD
....
0 0 0
```

样例输出：

YES
YES
NO

分析：

假设小狗所在的位置为（Sx,Sy），迷宫的门的位置为（Dx，Dy），门的开启时间为 t。

dfs 搜索函数带有 3 个参数：dfs(x,y,dep):已经到达位置(x,y),且已经花费的时间为 dep 秒，如果到达门的位置且时间符合要求，则搜索停止；否则从相邻位置继续进行搜搜。

成功逃离的条件是：x=Dx,y=Dy,dep=t。

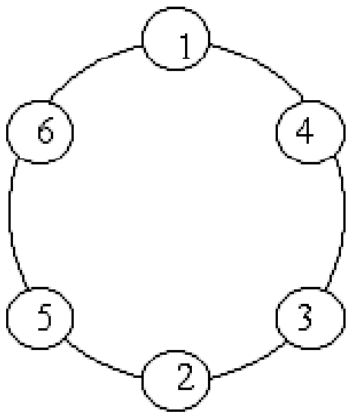
没有要求输出路线，可以不用记录路线，只记住当前位置即可（记下路径也可以）。

```
//zoj2110 Tempter of the bone
#include<cstdio>
#include<iostream>
#include<cstring>
using namespace std;
int dx[4]={-1,1,0,0};
int dy[4]={0,0,-1,1};
int map[110][110];
int n,m,t,ok,Sx,Sy,Dx,Dy;
void dfs(int x,int y,int dep){
    if(x==Dx&&y==Dy&&dep==t){
        ok=1;
        return;
    }
    for(int i=0;i<4;i++){
        int xx=x+dx[i],yy=y+dy[i];
        if(xx>=1&&xx<=n&&yy>=1&&yy<=m&&map[xx][yy]==0){
            map[xx][yy]=1;
            dfs(xx,yy,dep+1);
            if(ok) return;
            map[xx][yy]=0;
        }
    }
}
int main(){
```

```
    //freopen("maze.in","r",stdin);
    while((cin>>n>>m>>t)&&(n+m+t!=0)){
        char ch;
        memset(map,0,sizeof(map));
        for(int i=1;i<=n;i++)
            for(int j=1;j<=m;j++){
                cin>>ch;
                if(ch=='.')map[i][j]=0;
                if(ch=='X')map[i][j]=1;
                if(ch=='S'){Sx=i;Sy=j;};
                if(ch=='D'){Dx=i;Dy=j;};
            }
        ok=0;
        map[Sx][Sy]=1;
        dfs(Sx,Sy,0);
        if(ok)printf("YES\n");
        else printf("NO\n");
    }
    return 0;
}
```

例 3.1.4 素数环问题（uva524）

输入一个偶数 n（<=20）,输出所以的排列方式，要求相邻的两个数的和为素数，第 1 个数和最后的一个数和也为素数，即 n 个数组成一个素数环。下图是 n=6 的素数环。



输入：

n

输出：

以 1 开头的素数环，按字典序输出。

分析：

思路和引例 2 安排座位类似。这里无需建立图。

```
Sample Input
 8 6
Sample Output
Case 1:
```

```
1 4 3 2 5 6
1 6 5 2 3 4

Case 2:
1 2 3 8 5 6 7 4
1 2 5 8 3 4 7 6
1 4 7 6 5 8 3 2
1 6 7 4 3 8 5 2
```

注意输出要求：两个样例之间一个空行。

```
//uva524 素数环问题
#include<cstdio>
#include<iostream>
#include<cstring>
using namespace std;
int n,a[30],vis[30]={0};
bool is_prime(int k){
    for(int i=2;i*i<=k;i++)
        if(k%i==0) return 0;
    return 1;
}
void dfs(int cur){
    if(cur==n&&is_prime(1+a[cur])){
        for(int i=1;i<n;i++) printf("%d ",a[i]) ;
        printf("%d\n",a[n]);
    }
    for(int i=1;i<=n;i++){
        if(vis[i]==0&&is_prime(i+a[cur])){
            a[cur+1]=i;
            vis[i]=1;
            dfs(cur+1);
            vis[i]=0;
        }
    }
}
int main(){
    int T=0;
    while(scanf("%d",&n)==1&&n>0){
        if(T>0) printf("\n");
        printf("Case %d:\n",++T);
        memset(vis,0,sizeof(vis));
        a[1]=1;
        vis[1]=1;
        dfs(1);
```

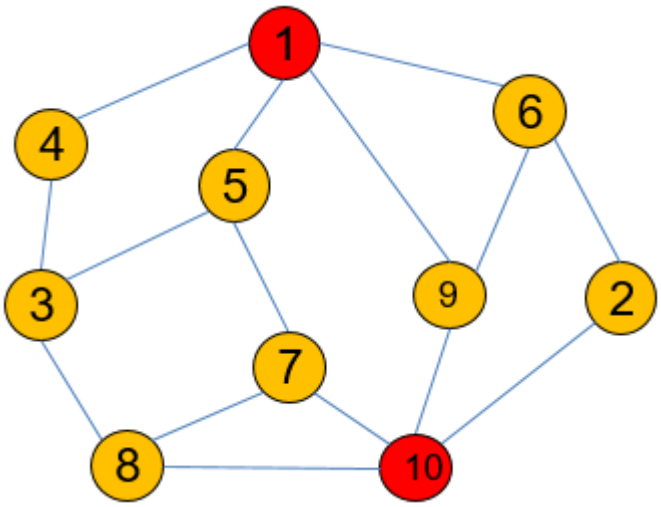
```
    }
    return 0;
}
```

例 3.1.5 最少换乘次数

给定 n(<=100)个城市及城市间的交通路线（双向），每列火车只能在固定的两个城市间运行，也就是说从城市 A 到城市 B，如果中间经过城市 C，则从 A 到 C 后，必须在 C 处换乘另一辆火车才能到达 B。

求从 1 号城市到 n 号城市的最少的换乘次数。

从 1 到 10，最少换乘 1 次。



输入：

```
10
15
1 4
1 5
1 9
1 6
4 3
5 3
5 7
9 10
6 9
6 2
3 8
7 8
7 10
2 10
8 10
```

输出：

```
1
```

分析：

求一条从 1 到 n 的路线，要求中间经过的点最少。

如果用 dfs，需要找出所有路线比较，找最短的。

如果采用 bfs 找到即可，无需比较。

```
#include<cstdio>
#include<iostream>
using namespace std;
struct node{
    int x;   //存顶点
    int dep; //存深度：x 点所在的层数
};//结构体
node q[101];
int vis[101]={0};
int a[101][101];
int n,m;
int bfs(){
    int head=0,tail=1;
    q[0].x=1;
    q[0].dep=0;
    vis[1]=1;
    while(head<tail){
        node p=q[head++];
        if(p.x==n) return p.dep-1;
        for(int i=1;i<=n;i++)
            if(a[p.x][i]==1&&!vis[i]){
                q[tail].x=i;
                q[tail].dep=p.dep+1;
                vis[i]=1;
                tail++;
            }
    }
}
int main(){
    cin>>n>>m;
    int x,y;
    for(int i=0;i<m;i++){
        cin>>x>>y;
        a[x][y]=a[y][x]=1;
    }
    cout<<bfs()<<endl;
    return 0;
}
```

3.2 深度优先搜索算法（周鑫）

【知识点】

理解深度优先搜索算法的原理，掌握深搜算法的框架，执行过程，各种形式的灵活应用。

【知识讲解】

从当前初始状态开始，按深度方向搜索，能往下走就走，不能走就返回到上一层，继续找没有被访问的结点，直至所有结点都被访问为止。

执行过程如下：
（1）定义合适的状态描述（有用的状态）；
（2）初始状态(起点)、目标状态；
（3）明确从某一状态出发向下一个状态扩展的规则和方法
Dfs 最基本的框架：

```
void dfs(i:当前状态 a[i]） //开始 dfs(0);
[
  if 当前状态 i 是目标 then 输出方案；
  按 k 个规则扩展新状态 xx；
        if 状态 xx 符合要求 then
            [
              记下状态 x[i+1]=xx；
              [标记新状态，避免重复走； ]
              dfs(i+1)；//新状态作为新起点继续搜索
              [去掉标记，供后面继续走； ]
             ]
]
```

例 3.2.1 犯罪团伙

警察抓到了 n 个罪犯，警察根据经验知道他们属于不同的犯罪团伙，却不能判断有多少个团伙，但通过警察的审讯，知道其中的一些罪犯之间相互认识，已知同一犯罪团伙的成员之间直接或间接认识。有可能一个犯罪团伙只有一个人。请你根据已知罪犯之间的关系，确定犯罪团伙的数量。已知罪犯的编号从 1 至 n。

输入：

第一行：n（<=1000,罪犯数量），

第二行：m（<100000，关系数量）

以下若干行：每行两个数：I 和 j，中间一个空格隔开，表示罪犯 i 和罪犯 j 相互认识。

输出：

一个整数，犯罪团伙的数量。

样例输入：

```
11
8
1 2
4 5
3 4
1 3
5 6
```

7 10
5 10
8 9
样例输出：
3
说明：共三个团伙。

分析：用二维数组 g[i][j]存储 i 是否认识 j，g[i][j]=1 表示 i 和 j 认识，g[i][j]=0 表示 i 和 j 不认识。

实现方法：

```
#include<cstdio>
#include<iostream>
using namespace std;
const int maxn=10010;
int n,m,x,y,ans,v[maxn];
int g[maxn][maxn];
void dfs(int k)
{
    v[k]=1;
    for(int i=1;i<=n;i++)
       if(!v[i]&&g[i][k]==1) dfs(i);
}
int main()
{
    cin>>n>>m;
    for(int i=1;i<=m;i++)
    {
        cin>>x>>y;
        g[x][y]=1;g[y][x]=1;
    }
    for(int i=1;i<=n;i++)
    {
        if(!v[i]){
            ans++;
            dfs(i);
        }
    }
    cout<<ans<<endl;
}
```

例 3.2.2 生成 1~n 的全排列（1<=n<=10）。

例： n=3 的全排列为：

1 2 3
1 3 2
2 1 3
2 3 1
3 1 2

3 2 1

6

实现方法一：

```
//生成 1~n 的全排列方法 2：回溯，设置辅助变量
#include<cstdio>
#include<iostream>
#include<cstring>
using namespace std;
int a[12],b[12];
int n,ans;
void dfs(int i)
{
    if(i==n)
    {
        ans++;
        for(int j=1;j<=n;j++) printf("%d ",a[j]);
        printf("\n");
        return;
    }
    for(int j=1;j<=n;j++)
      if(!b[j])
      {
        a[i+1]=j;
        b[j]=1;
        dfs(i+1);
        b[j]=0;
      }
}
int main()
{
    cin>>n;
    dfs(0);
    printf("%d\n",ans);
    return 0;
}
```

实现方法二：

```
//生成 1~n 的全排列方法 1：设置标志变量
#include<cstdio>
#include<iostream>
#include<cstring>
using namespace std;
int a[12];
int n,ans=0;
void dfs(int i)
{
```

```
    if(i==n){
        ans++;
        for(int j=1;j<=n;j++) printf("%d ",a[j]);
        printf("\n");
        return;
    }

  for(int j=1;j<=n;j++)
  {
    int ok=1;
    for(int k=1;k<=i;k++)
        if(a[k]==j) ok=0;
    if(ok)
    {
        a[i+1]=j;
        dfs(i+1);
    }
  }
}
int main()
{
    scanf("%d",&n);
    dfs(0);
    printf("%d\n",ans);
    return 0;
}
```

例 3.2.3　借书问题

学校放暑假时，信息学辅导教师有 n 本书要分给参加培训的 n 个学生。如：A，B，C，D，E 共 5 本书要分给参加培训的张、刘、王、李、孙 5 位学生，每人只能选 1 本。教师事先让每个人将自己喜爱的书填写在如下的表中，然后根据他们填写的表来分配书本，希望设计一个程序帮助教师求出可能的分配方案，使每个学生都满意。

输入：

第一行一个数 n（学生的个数，书的数量）

以下共 n 行，每行 n 个 0 或 1（由空格隔开），第 i 行数据表示第 i 个同学对所有书的喜爱情况。0 表示不喜欢该书，1 表示喜欢该书。

输出：

所有的分配方案，每种方案一行：依次输出每个学生分到的书号。

样例输入：

```
5
0 0 1 1 0
1 1 0 0 0
0 1 1 0 0
0 0 0 1 0
0 1 0 0 1
```

样例输出：

3 1 2 4 5

	A	B	C	D	E
张			Y	Y	
王	Y	Y			
刘		Y	Y		
孙				Y	
李		Y			Y

分析：设二维数组 like [i][j]表示第 i 个人是否喜欢第 j 本书，希望，值为 1，不喜欢，值为 0; 用数组 a [i]存储第 i 个人借书的编号；用数组 can [i]表示第 i 本书是否可借，值为 1 表示可以借，值为 0 表示不可以借，已被借走。

实现方法：

```
#include<iostream>
using namespace std;
const int maxn=120;
int like[maxn][maxn],a[maxn],can[maxn],n;
void dfs(int i)
{
  if(i==n+1)    {
    for(int i=1;i<=n-1;i++) cout<<a[i]<<" ";
    cout<<a[n]<<endl;
    return;
  }
  for(int j=1;j<=n;j++)
      if(like[i][j] && !can[j])
           {
             a[i]=j;
             can[j]=1;
             dfs(i+1);
             can[j]=0;
           }
}
int main()
{
    cin>>n;
    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
           cin>>like[i][j];
    dfs(1);
return 0;
}
```

例 3.2.4 铺瓷砖

有一长度为 N(1<=N<=10)的地板，给定三种不同瓷砖：一种长度为 1，一种长度为 2，另一种长度为 3，数目不限。要将这个长度为 N 的地板铺满，并且要求长度为 1 的瓷砖 不能相邻，一共有多少种不同的铺法？在所有的铺设方法中，一共用了长度为 1 的瓷砖多少块？

例如，长度为 4 的地面一共有如下 4 种铺法，并且，一共用了长度为 1 的瓷砖 4 块：

4=1+2+1

4=1+3

4=2+2

4=3+1

编程求解上述问题。

输入格式

只有一个数 N，代表地板的长度。

输出格式

所有的铺设方案和方案数量。

样例输入

4

样例输出

4=1+2+1

4=1+3

4=2+2

4=3+1

4

分析：在长度为 cur 的基础上再铺的时候，要注意长度为 1 的瓷砖不能相邻的限制条件。

实现方法：

```
#include<cstdio>
int n,ans,a[31];
void dfs(int step,int cur)
{
 if(cur==n)
 {
     ans++;
     printf("%d=%d",n,a[1]);
     for(int i=2;i<=step;i++) printf("+%d",a[i]);
     printf("\n");
 }
 for(int i=1;i<=3;i++) if(cur+i<=n) if(cur+i<=n)
 {
     if(!(step>0 && a[step]==1 && i==1))
     {
         a[step+1]=i;
         dfs(step+1,cur+i);
     }
 }
}
int main()
{
```

```
    scanf("%d",&n);
    dfs(0,0);
    printf("%d\n",ans);
    return 0;
}
```

例 3.2.5 **马走日（题目来源：http://noi.openjudge.cn/ch0205/8465/）**

马在中国象棋以日字形规则移动。

请编写一段程序，给定 n*m 大小的棋盘，以及马的初始位置(x，y)，要求不能重复经过棋盘上的同一个点，计算马可以有多少途径遍历棋盘上的所有点。

输入

第一行为整数 T(T < 10)，表示测试数据组数。

每一组测试数据包含一行，为四个整数，分别为棋盘的大小以及初始位置坐标 n,m,x,y。(0<=x<=n-1,0<=y<=m-1, m < 10, n < 10)

输出

每组测试数据包含一行，为一个整数，表示马能遍历棋盘的途径总数，0 为无法遍历一次。

样例输入

1

5 4 0 0

样例输出

32

分析：如下图所示，马走日有八个方向，若原来的坐标为（i,j），则八个方向的移动可表示为：

1：（i,j）→（i-2,j-1）

2：（i,j）→（i-2,j+1）

3：（i,j）→（i-1,j+2）

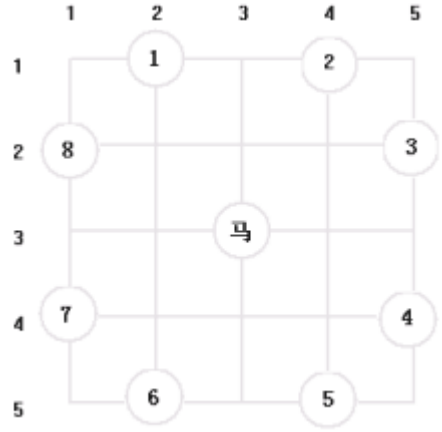
4：（i,j）→（i+1,j+2）

5：（i,j）→（i+2,j+1）

6：（i,j）→（i+2,j-1）

7：（i,j）→（i+1,j-2）

8：（i,j）→（i-1,j-2）



搜索策略：从初始位置出发，按照移动规则依次选定某个方向，所到达的位置做上标记，直到棋盘上所有的格子都遍历完，途径数加 1。

实现方法：

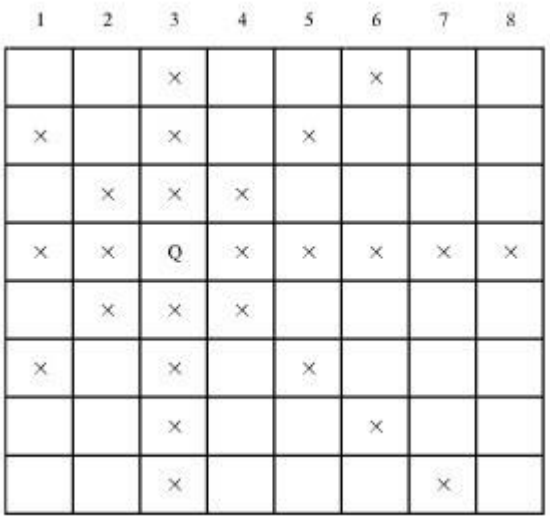
```
#include<cstdio>
#include<iostream>
```

```
#include<cstring>
using namespace std;
int n,m,x,y,ans,b[20][20];
int dx[]={-2,-2,-1,-1,1,1,2,2},dy[]={-1,1,-2,2,-2,2,-1,1};
void dfs(int x,int y,int step)
{
  if(step==n*m){ans++;return;}
  for(int i=0;i<8;i++)
  {
    int xx=x+dx[i];
    int yy=y+dy[i];
    if(xx<0||yy<0||xx>n-1||yy>m-1)continue;
    if(!b[xx][yy])
    {
      b[xx][yy]=1;
      dfs(xx,yy,step+1);
      b[xx][yy]=0;
    }
  }
}
int main()
{
    int t;
    cin>>t;
    while(t--)
    {
      ans=0;
      memset(b,0,sizeof(b));
      cin>>n>>m>>x>>y;
      b[x][y]=1;
      dfs(x,y,1);
      cout<<ans<<endl;
    }
return 0;
}
```

例 3.2.6 八皇后问题

要在国际象棋棋盘中放八个皇后，使任意两个皇后都不能互相吃。（提示：皇后能吃同一行、同一列、同一对角线的任意棋子。）输出所有的方案及方案总数。

分析：问题的关键在于如何判定某个皇后所在的行、列、斜线上是否有别的皇后；可以从矩阵的特点上找到规律，如果在同一行，则行号相同；如果在同一列，则列号相同；如果同在/斜线上，则行列值之和相同；如果同在\斜线上，则行列值之差相同。从下图可以验证：



考虑到每行有且仅有一个皇后，设一维数组 a[1..8]表示皇后的位置：第 i 行皇后放置第 j 列，用 a[i]=j 来表示，即下标是行数，内容是列数。例如：a[3]=5 表示第 3 行的皇后放在第 5 列上。

判断皇后是否安全，即检查同一列、同一对角线是否已有皇后，建立标志数组 b[1..8]，控制同一列只能有一个皇后，若两皇后在同一对角线上，则其行列坐标之和或行列坐标之差相等，故可建立标志数组 c[1..16]、d[-7..7]控制同一对角线上只能有一个皇后。因为主对角线的值有为负数的情况，所以我们在标记的时候应该加>=7 的数,所有值都加了>=7 所以标记的效果并没有改变。

实现方法：

```
#include<iostream>
#include<cstdio>
using namespace std;
int a[10],b[10],c[20],d[20],sum;
void print()
{
   for(int i=1;i<=8;i++)
    for(int j=1;j<=8;j++)
        if(a[i]==j)cout<<a[i];
   cout<<endl;
}
void dfs(int i)
{
   for(int j=1;j<=8;j++)
     if(!b[j] && !c[i+j] && !d[i-j+7])
     {
        a[i]=j;
        b[j]=1;
        c[i+j]=1;
        d[i-j+7]=1;
        if(i==8){
            sum++;
            print();
        }
```

```
            dfs(i+1);
            b[j]=0;
            c[i+j]=0;
            d[i-j+7]=0;
        }
    }
    int main()
    {
    dfs(1);
    cout<<sum<<endl;
    return 0;
    }
```

例 3.2.7 数的计算（题目来源：2001 年 NOIP 全国联赛普及组试题）

我们要求找出具有下列性质数的个数(包含输入的自然数 n):先输入一个自然数 n(n<=1000),然后对此自然数按照如下方法进行处理:

1. 不作任何处理;

2. 在它的左边加上一个自然数，但该自然数不能超过原数的一半;

3. 加上数后,继续按此规则进行处理,直到不能再加自然数为止.

输入：

一个数 n

输出：

满足条件的数的个数

样例输入

6

样例输出

6

数据范围及提示

6 个数分别是：

6

16

26

126

36

136

分析：一个数 n　　
　　不做任何处理　　} dfs(n)
　　左侧扩展本身的一半 1-n/2　　
　　对每一个一半 i，同样处理　　------dfs(i)

实现方法：

```
#include<iostream>
#include<cstdio>
using namespace std;
int x;
int dfs(int n)
{
```

```
    int i,ans=0;
    if(n==1) return 0;
    else
    {
       ans+=n/2;
       for(i=1;i<=n/2;i++)
         ans+=dfs(i);
    }
    return ans;
}
int main()
{
    cin>>x;
    cout<<dfs(x)+1;
    return 0;
}
```

例 3.2.8 水池数目（题目来源：http://acm.nyist.net/JudgeOnline/problem.php?pid=27）

描述

南阳理工学院校园里有一些小河和一些湖泊，现在，我们把它们通一看成水池，假设有一张我们学校的某处的地图，这个地图上仅标识了此处是否是水池，现在，你的任务来了，请用计算机算出该地图中共有几个水池。

输入：

第一行输入一个整数 N，表示共有 N 组测试数据

每一组数据都是先输入该地图的行数 m(0<m<100)与列数 n(0<n<100)，然后，输入接下来的 m 行每行输入 n 个数，表示此处有水还是没水（1 表示此处是水池，0 表示此处是地面）

输出:

输出该地图中水池的个数。

要注意，每个水池的旁边（上下左右四个位置）如果还是水池的话的话，它们可以看做是同一个水池。

样例输入：

2
3 4
1 0 0 0
0 0 1 1
1 1 1 0
5 5
1 1 1 1 0
0 0 1 0 1
0 0 0 0 0
1 1 1 0 0
0 0 1 1 1

样例输出：

2
3

分析：水池用数组 g 存储；用数组 v 标记这个点是否被访问过；dir 数组存储四个方向。

实现方法：

```
#include<iostream>
#include<cstring>
using namespace std;
int m,n;
int g[105][105],vis[105][105];
int dir[4][2]={{1,0},{-1,0},{0,1},{0,-1}};
void dfs(int x,int y)//找出（x,y）跟谁是统一水域
{
    int dx,dy;
    vis[x][y]=1;
    for(int i=0;i<4;i++)
    {
        dx=x+dir[i][0],dy=y+dir[i][1];
        if(dx>=0&&dx<m&&dy>=0&&dy<n&&g[dx][dy]&&!vis[dx][dy])
        dfs(dx,dy);
    }
}
int main()
{
    int t;
    cin>>t;
    while(t--)
    {
        int num=0;
        cin>>m>>n;
        for(int i=0;i<m;i++)
          for(int j=0;j<n;j++)
             cin>>g[i][j];
        for(int i=0;i<m;i++)
          for(int j=0;j<n;j++)
            if(g[i][j]==1&&!vis[i][j])
            {
                num++;
                dfs(i,j);
            }
        cout<<num<<endl;
    }
}
```

例 3.2.9 工作安排

问题描述：

n 个人从事 n 项工作，每个人做不同的工作获得的效益不同，现在要求每人只能从事其中的一项工作，且不能重复，求最佳安排使效益最高。

如有 A，B，C，D，E 五人从事 J1，J2，J3，J4，J5 五项工作，每人只能从事一项，他们的效益如下：

	J1	J2	J3	J4	J5
A	13	11	10	4	7
B	13	10	10	8	5
C	5	9	7	7	4
D	15	12	10	11	5
F	10	11	8	8	4

当 A 从事 J5，B 从事 J3， C 从事 J4 ， D 从事 J1 ，E 从事 J2 时收益最大值：50

输入：

第一行：n。（n<=20）

以下 n 行是 n*n 的矩阵。

输出：

第一行，最大效益。

以下 n 行，分别是每个人的工作安排情况。

样例输入：

5

13 11 10 4 7

13 10 10 8 5

5 9 7 7 4

15 12 10 11 5

10 11 8 8 4

样例输出：

50

1:5

2:3

3:4

4:1

5:2

说明：所有的输入数据以及结果都不超过 1000000000。

分析：用数组 f 存储搜索工作选择的方案；数组 g 存储最优的工作选择方案；数组 p 用于表示某项工作有没有被选择。

实现方法：

```
#include<iostream>
#include<cstdio>
using namespace std;
const int maxn=20;
int n,data[maxn+2][maxn+2],g[maxn+2],f[maxn+2],p[maxn+2],max1=0;
void dfs(int step,int t)
{
    for(int i=1;i<=n;i++)
    if(!p[i])
    {
      f[step]=i;
      p[i]=1;
```

```
        t+=data[step][i];
        if(step<n) dfs(step+1,t);
        else if(t>max1)
        {
          max1=t;
          for(int j=1;j<=n;j++)
            g[j]=f[j];
        }
        t-=data[step][i];
        p[i]=0;
      }
  }
  int main()
  {
      cin>>n;
      for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
           cin>>data[i][j];
      dfs(1,0);
      cout<<max1<<endl;
      for(int i=1;i<=n;i++)
        cout<<i<<":"<<g[i]<<endl;
  return 0;
  }
```

例 3.2.10 城堡问题（题目来源：http://noi.openjudge.cn/ch0205/1817/）

问题描述：

1　2　3　4　5　6　7

############################

1 #　|　#　|　#　|　|　#

#####---#####---#---#####---#

2 #　#　|　#　#　#　#　#

#---#####---#####---#####---#

3 #　|　|　#　#　#　#　#

#---#########---#####---#---#

4 #　#　|　|　|　|　#　#

############################

(图 1)

 # = Wall

| = No wall

- = No wall

图 1 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×n(m≤50，n≤50)个方块，每个方块可以有 0~4 面墙。

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≤p≤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

分析：本题属于迷宫搜索类型的题，所以用递归的方法来做也比较形象，当给定房间的数字大于 8 则南可以走，-8 还大于等于四的可以往东走，-4 还大于等于二的往北走，-2 还大于等于一的则可以往西走。

实现方法：

```
#include <iostream>
#include<cstring>
#include<cstdio>
using namespace std;
int a[55][55],flag[55][55];
int recur(int m,int n,int x,int y)
{
    if(flag[x][y]==1) return 0;
    flag[x][y]=1;
    int p=a[x][y],sum=1;
    if(p==0)
    {
        if(y-1>0) sum+=recur(m,n,x,y-1);
        if(y+1<=n)sum+=recur(m,n,x,y+1);
        if(x+1<=m)sum+=recur(m,n,x+1,y);
        if(x-1>0) sum+=recur(m,n,x-1,y);
        return sum;
    }
    if(p>=8) p-=8;
    else  if(x+1<=m) sum+=recur(m,n,x+1,y);
    if(p>=4) p-=4;
    else  if(y+1<=n) sum+=recur(m,n,x,y+1);
    if(p>=2) p-=2;
    else if(x-1>0)  sum+=recur(m,n,x-1,y);
    if(p>=1) p-=1;
    else if(y-1>0)  sum+=recur(m,n,x,y-1);
    return sum;
}
```

```
int main()
{
    memset(flag,1,sizeof(flag[0])*55);
    int m=0,n=0;
    cin>>m>>n;
    int i=1;
    while(i<=m)
    {
        int j=1;
        while(j<=n)
        {
            cin>>a[i][j];
            j++;
        }
        i++;
    }
    int sum=0,max=0,x=0;
    for(i=1; i<=m; i++)
    {
        int j=1;
        while(j<=n)
        {
            x=recur(m,n,i,j);
            if(x>0) sum++;
            if(max<x) max=x;
            j++;
        }
    }
    cout<<sum<<endl<<max<<endl;
    return 0;
}
```

3.3 广度优先搜索算法（王乃广）

知识点

广度优先搜索算法（宽度优先搜索算法）

BFS，其英文全称是 Breadth First Search

利用队列数据结构（FIFO）

解决从初始状态到最终状态最少步骤问题

3.3.1 知识讲解

从初始状态出发，运用题设的规则找出经过一次扩展所能得到的状态（第一层结点），检查目标状态是否在这些状态中，若没有，再在这些状态上逐一进行一次扩展，得到第二层结点，并检查其中是否包含目标状态，若没有，再进行一次扩展，……，如此扩展、检查下去，直到找到目标状态或无法扩展出新状态为止。

也就是由近及远，依次找出由初始状态经过 1，2，3……所能到达的状态，主要用来解决“求最少步数”类的问题。

```
bfs 的基本框架
初始状态入队列;
while( 队列非空 ){
    队首元素 cur 出队;
    if( cur 是目标状态 )  return;
    for( int  i = 1; i <= 规则数; i++ ){
        由 cur 产生新状态 nxt;
        if( nxt 合法且没出现过 )  nxt 入队列;
    }
}
```

3.3.2 例题分析

例 3.3.1：跳马(Knight Moves), ZOJ1091, POJ2243

题目描述：

给定象棋棋盘上两个位置 a 和 b，编写程序，计算马从位置 a 跳到位置 b 所需步数的最小值。

输入描述：

输入文件包含多个测试数据。每个测试数据占一行，为棋盘中的两个位置，用空格隔开。棋盘位置为两个字符组成的串，第 1 个字符为字母 a～h，代表棋盘中的列；第 2 个字符为数字字符 1～8，代表棋盘中的行。

输出描述：

对输入文件中的每个测试数据，输出一行"To get from xx to yy takes n knight moves."，xx 和 yy 分别为输入数据中的两个位置， n 为求得的最少步数。

样例输入：	样例输出：
e2 e4	To get from e2 to e4 takes 2 knight moves.
a1 b2	To get from a1 to b2 takes 4 knight moves.
b2 c3	To get from b2 to c3 takes 2 knight moves.
a1 h8	To get from a1 to h8 takes 6 knight moves.
a1 h7	To get from a1 to h7 takes 5 knight moves.
h8 a1	To get from h8 to a1 takes 6 knight moves.
b1 c3	To get from b1 to c3 takes 1 knight moves.
f6 f6	To get from f6 to f6 takes 0 knight moves.

分析：

可以尝试用 DFS(深度优先搜索)来求解。dfs(x,y,step)表示从起始位置走到(x,y)需要 step 步，注意：第一次走到的步数不一定是最小的，需要找出所有的走法，再取最小。

起始位置是(sx,sy)，目标位置是(dx,dy)，主程序调用 dfs(sx,sy,0)。

```
void dfs(int x,int y,int step){
    if(step>=ans) return;
    if(x==dx&&y==dy){
        ans=min(ans,step); c[dx][dy]=inf; return;
    }
    for(int i=0;i<8;i++){
        int nx=x+d[i][0],ny=y+d[i][1];
        if(nx>=0&&nx<8&&ny>=0&&ny<8&&c[nx][ny]>step+1){
            c[nx][ny]=step+1; dfs(nx,ny,step+1);
        }
    }
    if(step>0) c[x][y]=inf;
}
```

如果方向按照如下的顺序来定义，可以通过样例数据以及 h8 d7，但 d7 h8 会严重超时。

```
const int
d[][2]={{-1,-2},{-2,-1},{-2,1},{-1,2},{1,2},{2,1},{2,-1},{1,-2}};
```

输入数据是 h8 d7：

```
To get from h8 to d7 takes 3 knight moves.
real    0m0.003s
```

输入数据是 d7 h8：

```
To get from d7 to h8 takes 3 knight moves.
real    2m15.271s
```

BFS(广度优先搜索)

状态中包含的信息：int x,y;表示位置(x,y)，int step;表示走到(x,y)的步数。

起始位置对应的状态是(sx,sy,0)，进入队列。

队首元素出队列，由该状态分别沿 8 个方向各走 1 步，新到达的位置进入队列，再取队首元素继续处理，直到走到目标位置或队列空。

```
void bfs(){
    memset(g,false,sizeof(g));
    g[cur.x][cur.y]=true;          //起始位置访问标志
    queue<node> q; q.push(cur);       //起始位置入队列
    while(!q.empty()){
        cur=q.front(); q.pop();
        if(cur.x==dx&&cur.y==dy){
            printf("To get from %s to %s takes %d knight
moves.\n",src,dst,cur.step);
            return;
        }
        for(int i=0;i<8;i++){
            nxt.x=cur.x+d[i][0]; nxt.y=cur.y+d[i][1];
```

```
    if(nxt.x>=0&&nxt.x<8&&nxt.y>=0&&nxt.y<8&&!g[nxt.x][nxt.y]){
                    nxt.step=cur.step+1; q.push(nxt);     //新位置入队列
                }
            }
        }
    }
```

例 3.3.2：仙岛求药,noi.openjudge.cn/ch0205/2727

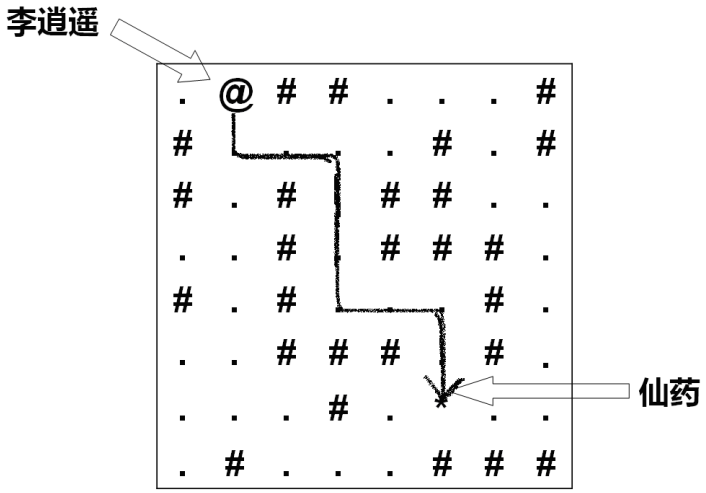
总时间限制：1000ms

内存限制：65536kB

描述

少年李逍遥的婶婶病了，王小虎介绍他去一趟仙灵岛，向仙女姐姐要仙丹救婶婶。叛逆但孝顺的李逍遥闯进了仙灵岛，克服了千险万难来到岛的中心，发现仙药摆在了迷阵的深处。迷阵由 M×N 个方格组成，有的方格内有可以瞬秒李逍遥的怪物，而有的方格内则是安全。现在李逍遥想尽快找到仙药，显然他应避开有怪物的方格，并经过最少的方格，而且那里会有神秘人物等待着他。现在要求你来帮助他实现这个目标。

下图显示了一个迷阵的样例及李逍遥找到仙药的路线.



输入

输入有多组测试数据．每组测试数据以两个非零整数 M 和 N 开始，两者均不大于 20。M 表示迷阵行数，N 表示迷阵列数。接下来有 M 行，每行包含 N 个字符,不同字符分别代表不同含义：

1）'@'：少年李逍遥所在的位置；

2）'.'：可以安全通行的方格；

3）'#'：有怪物的方格；

4）'*'：仙药所在位置。

当在一行中读入的是两个零时，表示输入结束。

输出

对于每组测试数据，分别输出一行，该行包含李逍遥找到仙药需要穿过的最少的方格数目(计数包括初始位置的方块)。如果他不可能找到仙药，则输出 -1。

样例输入	样例输出
8 8 .@##...#	10 8

#....#.# #.#.##.. ..#.###. #.#...#. ..###.#. ...#.*.. .#...### 6 5 .*.#. .#... ..##. ..... .#... ....@ 9 6 .#..#. .#.*.# .####. ..#... ..#... ..#... ..#... #.@.## .#..#. 0 0	-1

分析：

典型的迷宫问题求最小步数，状态信息包括坐标位置(x,y)和步数 step，

状态的判重：既可以用状态数组，也可以把走过的位置修改为'#'。

例 3.3.3：抓住那头牛，noi.openjudge.cn/ch0205/2971

总时间限制： 2000ms

内存限制： 65536kB

描述

农夫知道一头牛的位置，想要抓住它。农夫和牛都位于数轴上，农夫起始位于点 N(0<=N<=100000)，牛位于点 K(0<=K<=100000)。农夫有两种移动方式：

1、从 X 移动到 X-1 或 X+1，每次移动花费一分钟

2、从 X 移动到 2*X，每次移动花费一分钟

假设牛没有意识到农夫的行动，站在原地不动。农夫最少要花多少时间才能抓住牛？

输入：

两个整数，N 和 K。

输出：

一个整数，农夫抓到牛所要花费的最小分钟数。

样例输入

5 17

样例输出

4

分析：

给出了三种规则，求花费的最短时间，适合用 BFS 来求解。

X 可以移动到 X-1，X+1,2*X，只有 X 到 X-1 能使得位置的坐标减小，当 n>=k 时，不必 bfs，直接输出 n-k 即可。

例 3.3.4：简单的迷宫问题(Basic Wall Maze), POJ2935

题目描述：

在本题中，你需要求解一个简单的迷宫问题：

1）迷宫由 6 行 6 列的方格组成；

2）3 堵长度为 1～6 的墙壁，水平或竖直地放置在迷宫中，用于分隔方格；

3）一个起始位置和目标位置。

如下面的图 2.23 描述了一个迷宫。你需要找一条从起始位置到目标位置的最短路径。从任一个方格出发，只能移动到上、下、左、右相邻方格，并且没有被墙壁所阻挡。

输入描述：

输入文件中包含多个测试数据。每个测试数据包含 5 行：第 1 行为两个整数，表示起始位置的列号和行号；第 2 行也是两个整数，为目标位置的列号和行号，列号和行号均从 1 开始计起。；第 3～5 行均为 4 个整数，描述了 3 堵墙的位置；如果墙是水平放置的，则由左、右两个端点所在的位置指定，如果墙是竖直放置的，则由上、下两个端点所在的位置指定；端点的位置由两个整数表示，第 1 个整数表示端点距离迷宫左边界的距离，第 2 个整数表示端点距离迷宫上边界的距离。

假定这 3 堵墙互相不会交叉，但两堵墙可能会相邻于某个方格的顶点。从起始位置到目标位置一定存在路径。下面的样例输入数据描述了图 2.23 所示的迷宫。

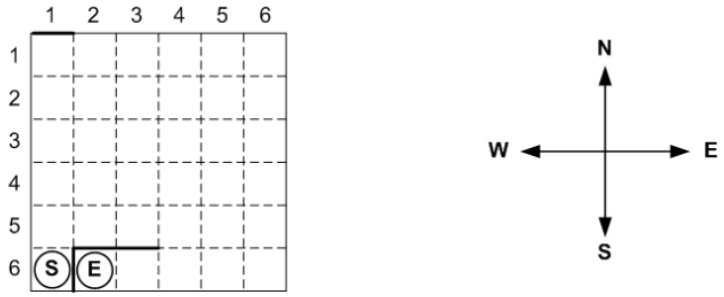


图 2.23 一个简单的迷宫问题

输入文件中最后一行为两个 0，代表输入结束。

输出描述：

对输入文件中的每个测试数据，输出从起始位置到目标位置的最短路径，最短路径由代表每一步移动的字符组成（ 'N'表示向上移动， 'E'表示向右移动， 'S'表示向下移动， 'W'表示向左移动）。

对某个测试数据，可能存在多条最短路径，对于这种情形，只需输入任意一条最短路径即可。

样例输入：	样例输出：
1 6 2 6 0 0 1 0 1 5 1 6 1 5 3 5 0 0	NEEESWW

分析：

迷宫求最短路径问题，用 BFS 来求解。

本题中的坐标系行列是颠倒的，参考样例及图示把握好起始位置、结束位置和墙的位置。

对于位置(x,y)，只需记录它的'E'方向和'S'方是否有墙即可；

(x,y)沿'E'方向到达(x+1,y)，如果(x,y)的'E'方向没有墙；

(x,y)沿'S'方向到达(x,y+1)，如果(x,y)的'S'方向没有墙；

(x,y)沿'W'方向到达(x-1,y)，如果(x,y)的'W'方向没有墙，(x,y)的'W'方向的墙对应(x-1,y)'E'方向的墙；

(x,y)沿'N'方向到达(x,y-1)，如果(x,y)的'N'方向没有墙，(x,y)的'N'方向的墙对应(x,y-1)'S'方向的墙。

本题要求输出的不是最短路径的长度，而是具体的路径。在状态中除了记录位置(x,y)之外，还需要记录(x,y)的前驱，即(x,y)是由那个位置走到的，以及沿哪个方向走到(x,y)。

```
struct node{
    int  x,y;      //位置(x,y)
    int  pre;      //(x,y)的前驱位置在队列中的位置
    char d;        //前驱位置到(x,y)的方向
};
```

BFS 结束后，由目标位置沿 pre 依次找前驱，用递归的思路输出路径的方向序列。

例 3.3.5：**机器人搬重物,www.luogu.org1126**

题目描述

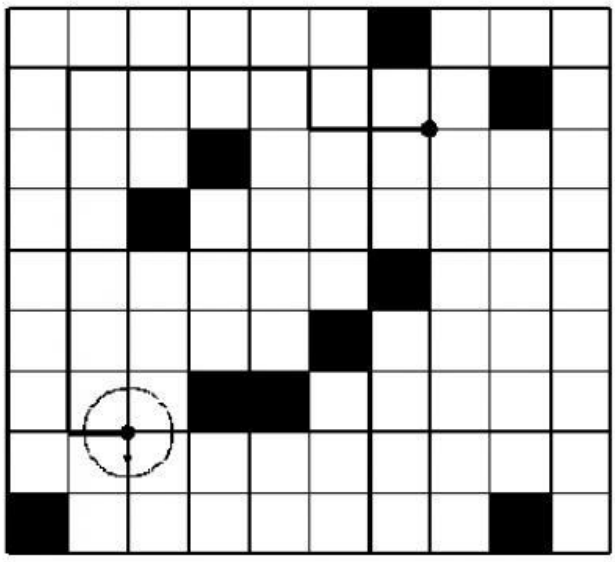
机器人移动学会（RMI）现在正尝试用机器人搬运物品。机器人的形状是一个直径 1.6 米的球。在试验阶段，机器人被用于在一个储藏室中搬运货物。储藏室是 一个 N*M 的网格，有些格子为不可移动的障碍。机器人的中心总是在格点上，当然，机器人必须在最短的时间内把物品搬运到指定的地方。机器人接受的指令有： 向前移动 1 步（Creep）；向前移动 2 步（Walk）；向前移动 3 步（Run）；向左转（Left）；向右转（Right）。每个指令所需要的时间为 1 秒。请你计算一下机器人完成任务所需的最少时间。

输入格式：

输入的第一行为两个正整数 N,M（N,M<=50），下面 N 行是储藏室的构造，0 表示无障碍，1 表示有障碍，数字之间用一个空格隔开。接着一行有四个整数和一个大写字母，分别为起始点和目标点左上角网格的行与列，起始时的面对方向（东 E，南 S，西 W，北 N），数与数，数与字母之间均用一个空格隔开。终点的面向方向是任意的。

输出格式：

一个整数，表示机器人完成任务所需的最少时间。如果无法到达，输出-1。



样例输入：	样例输出：
9 10 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 7 2 2 7 S	12

分析：

机器人能在格点(x,y)上的话，(x,y)周围的四个格子都没有障碍。

状态中需要记录位置(x,y)、步数 step 以及方向，从当前位置出发，可选的方案有：沿当前方向前进 1,2,3 步、左转、右转。

(x,y)走 1 步到达(x1,y1)的前提是(x,y)和(x1,y1)四周的格子都没有障碍；

(x,y)走 2 步到达(x2,y2)的前提是(x,y)、(x1,y1)、(x2,y2)四周的格子都没有障碍；

走 3 步也是类似的情形。

另外，如果(x1,y1)周围有障碍，2 步、3 步一定没法走。

状态判重时，位置和方向都相同的状态才算是同一个状态，位置相同但方向不同的是两个不同的状态。

例 3.3.6：倍数(Multiple), ZOJ1136, POJ1465

题目描述：

编写程序，实现：给定一个自然数 N， N 的范围为[0, 4999]，以及 M 个不同的十进制数字 X1, X2, …, XM（至少一个，即 M≥1），求 N 的最小的正整数倍数，满足：N 的每位数字均为 X1, X2, …,XM 中的一个。

输入描述：

输入文件包含多个测试数据，测试数据之间用空行隔开。每个测试数据的格式为：第 1 行为自然数 N；第 2 行为正整数 M；接下来有 M 行，每行为一个十进制数字，分别为 X1, X2, …, XM。

输出描述：

对输入文件中的每个测试数据，输出符合条件的 N 的倍数；如果不存在这样的倍数，则输出 0。

样例输入：	样例输出：
22 3 7 0 1 2 1 1	110 0

分析：

把给定的数字按升序排。例：1，2，3，可以组成的数(按从小到大的顺序)

1，2，3

11，12，13，21，22，23，31，32，33

111，112，113

121，122，123

131，132，133

211，212，213，……

显然具有队列的特点。

考虑用 BFS 来按顺序生成正整数，直到能被 N 整除。

状态的表示：

如果队列中存生成的数，那么对于无解的情况，没有明显的结束条件。

如样例 2：n=2，给定的数字只有一个 1，可以生成的正整数是 1,11,111,……，不存在能被 2 整除的正整数。

改进：

由于 0<=n<=4999，生成的正整数%n 不超过 5000。

生成的正整数中，如果存在 a%n==b%n 且 a<b，那么没有必要在 b 的基础上继续搜索。(?)

可以在队列中存余数，这样的总状态数不超过 5000.

```
struct node{
    int d;     //数字
    int pre;       //前驱
    int yu;        //余数
};
```

例 3.3.7：逃跑的拉尔夫，codevs.cn 1026

时间限制：1 s

空间限制：128000 KB

题目描述

年轻的拉尔夫开玩笑地从一个小镇上偷走了一辆车，但他没想到的是那辆车属于警察局，并且车上装有用于发射车子移动路线的装置。

那个装置太旧了，以至于只能发射关于那辆车的移动路线的方向信息。

编写程序，通过使用一张小镇的地图帮助警察局找到那辆车。程序必须能表示出该车最终所有可能的位置。

小镇的地图是矩形的，上面的符号用来标明哪儿可以行车哪儿不行。“.”表示小镇上那块地方是可以

行车的，而符号“X”表示此处不能行车。拉尔夫所开小车的初始位置用字符的“*”表示，且汽车能从初始位置通过。

汽车能向四个方向移动：向北(向上)，向南(向下)，向西(向左)，向东(向右)。

拉尔夫所开小车的行动路线是通过一组给定的方向来描述的。在每个给定的方向，拉尔夫驾驶小车通过小镇上一个或更多的可行车地点。

输入描述

输入文件的第一行包含两个用空格隔开的自然数 R 和 C，1≤R≤50，1≤C≤50，分别表示小镇地图中的行数和列数。

以下的 R 行中每行都包含一组 C 个符号(“.”或“X”或“*”)用来描述地图上相应的部位。

接下来的第 R+2 行包含一个自然数 N，1≤N≤1000，表示一组方向的长度。

接下来的 N 行幅行包含下述单词中的任一个：NORTH(北)、SOUTH(南)、WEST(西)和 EAST(东)，表示汽车移动的方向，任何两个连续的方向都不相同。

输出描述

输出文件应包含用 R 行表示的小镇的地图(象输入文件中一样)，字符“*”应该仅用来表示汽车最终可能出现的位置。

样例输入：	样例输出：
4 5 ..... .X... ...*X X.X.. 3 NORTH WEST SOUTH	..... *X*.. *.*.X X.X..

分析：

本题与普通的迷宫类问题不同，给出了表示路线的方向序列，沿某个方向前进的距离可以有多个，所以小车能到达的目标位置会有多个，求所有可能的目标位置。

思路一：

从起始位置出发，沿第一个方向前进，可以找出所有可能到的位置，入队列；

再从这些位置出发，沿第二个方向前进，找出所有可能到的位置，入队列；

……；

从这些位置(沿倒数第二个方向到达的)出发，沿最后一个方向前进，找出所有可能到的位置，就是题目所求的位置。

队列由若干段组成，需要维护每个状态是走了几个方向走到的，以便再沿下一个方向前进。

思路二：

初始状态小车在起始位置，沿第一个方向前进，能到达的位置在一个二维数组上标记出来；

从这个二维数组上标记的位置出发，沿第二个方向前进，能到达的位置再在一个二维数组上标记出来；

再从第二个数组上标记的位置出发，沿第三个方向前进时，能到达的位置可以用第一个二维数组来标记，因为原有的内容没有用了。

也就是说，只需保存相邻两步的状态表。

例 3.3.8：单词序列,noi.openjudge.cn/ch0407/8468

总时间限制：1000ms

内存限制：1024kB

描述

给出两个单词（开始单词和结束单词）以及一个词典。找出从开始单词转换到结束单词，所需要的最短转换序列。转换的规则如下：

1、每次只能改变一个字母

2、转换过程中出现的单词(除开始单词和结束单词)必须存在于词典中

例如：

开始单词为：hit

结束单词为：cog

词典为：[hot,dot,dog,lot,log,mot]

那么一种可能的最短变换是： hit -> hot -> dot -> dog -> cog,

所以返回的结果是序列的长度 5；

注意：

1、如果不能找到这种变换，则输出 0；

2、词典中所有单词长度一样；

3、所有的单词都由小写字母构成；

4、开始单词和结束单词可以不在词典中。

输入：

共两行，第一行为开始单词和结束单词（两个单词不同），以空格分开。第二行为若干的单词（各不相同），以空格分隔开来，表示词典。单词长度不超过 5,单词个数不超过 30。

输出：

输出转换序列的长度。

样例输入：	样例输出：
hit cog hot dot dog lot log	5

分析：

求开始单词到结束单词的最短转换序列，从开始单词出发，依次尝试把单词中的每个字母分别进行替换，如果新生成的单词在字典中且没出现过，入队列，得到目标单词或队列空时结束。

3.4 动态规划初步（王乃广）

知识点

动态规划算法是解决"多阶段决策问题"的一种高效算法，常用于解决统计类问题（计算方案总数）和最优值问题（最大值或最小值）。

理解阶段、状态和状态转移方程，运用递推法和记忆化搜索求解动态规划类问题。

3.4.1 知识讲解

引例：求斐波那契数列第 n 项。

fib(1)=fib(2)=1，fib(i)=fib(i-2)+fib(i-1)　(i>2)。

递归形式：

```
#include <cstdio>
const int maxn=52;
int n,c[maxn];       //c[i]:fib(i)被调用的次数
int fib(int k){
    c[k]++;
    if(k<=2) return 1;
    return fib(k-2)+fib(k-1);
}
int main(){
    scanf("%d",&n);
    printf("%d\n",fib(n));
    for(int i=1;i<=n;i++)
        printf("%d:%d\n",i,c[i]);
    return 0;
}
```

n>=40，超时。

速度慢的原因是做了很多重复的递归调用，而这些重复实际上是没有必要的。

记忆化搜索：

```
#include <cstdio>
const int maxn=52;
int n,c[maxn];       //c[i]:fib(i)被调用的次数
int f[maxn];     //f[i]:数列第 i 项的值
int fib(int k){
    //如果 f[k]已求得，直接返回，不再递归求解
    if(f[k]) return f[k];
    c[k]++;
    if(k<=2) return 1;
    return f[k]=fib(k-2)+fib(k-1);
}
//主程序同上
```

增加一个数组 f，记录已求得的值。

如果 f(k)已求得，直接返回；否则，递归调用求的 f(k)。

对于每个 k，只递归调用 1 次。

递推形式：

```
#include <cstdio>
const int maxn=52;
int n,f[maxn];
int main(){
    scanf("%d",&n);
    f[1]=f[2]=1;
    for(int i=3;i<=n;i++)
        f[i]=f[i-2]+f[i-1];
    printf("%d\n",f[n]);
    return 0;
}
```

按由小到大的顺序依次求 f(1)..f(n)。

思考：int f[3];也能求出 f(n)。

引例虽然比较简单，但体现了动态规划中常见的几个概念：

1.阶段：

求 f(n)，需要先求得 f(n-1)和 f(n-2)，求 f(n-1)需要先求得 f(n-2)和 f(n-3)，……，可以把问题的求解分成 n 个阶段，按顺序计算 f(1),f(2),……f(n)。

2.状态：

问题发展到各个阶段时所处的各种客观情况，第 i 个阶段的状态就是 f(i)。

3:决策：

一个阶段的状态给定以后，从该状态演变到下一阶段某个状态的一种选择（行为）称为决策。

4.状态转移：

根据上一阶段的状态和决策来求解本阶段的状态。

3.4.2 例题分析

例 3.4.1：三角形最佳路径问题,noi.openjudge.cn/ch0206/7625

总时间限制：1000ms

内存限制：65536kB

描述

如下所示的由正整数数字构成的三角形：

```
7
3 8
8 1 0
2 7 4 4
4 5 2 6 5
```

从三角形的顶部到底部有很多条不同的路径。对于每条路径，把路径上面的数加起来可以得到一个和，和最大的路径称为最佳路径。你的任务就是求出最佳路径上的数字之和。

注意：路径上的每一步只能从一个数走到下一层上和它最近的下边（正下方）的数或者右边（右下方）的数。

输入：

第一行为三角形高度 100>=h>=1，同时也是最底层边的数字的数目。

从第二行开始，每行为三角形相应行的数字，中间用空格分隔。

输出：

最佳路径的长度数值。

样例输入：	样例输出：

5 7 3 8 8 1 0 2 7 4 4 4 5 2 6 5	30

分析：

每次有两种选择：正下方或右下方。如果用 DFS 求出所有可能的路线，就可以从中选出最优路线。但效率太低：1 个 n 层的数字三角形的完整路线有 2^(n-1)条，当 n 很大时会超时。

为了得到高效的算法，需要用抽象的方法思考问题：当前位置(i,j)的状态为 f(i,j)，表示从格子(i,j)出发时能得到的最大和。在这个状态定义下，原问题的解是 f(1,1)。

现在看看不同状态之间是如何转移的：

从格子(i,j)出发有两种决策。如果向正下方走，则走到(i+1,j)后需要求"从(i+1,j)出发后能得到的最大和"这一问题，即 f(i+1,j)；向右下方走到(i+1,j+1)之后需要求 f(i+1,j+1)。

状态转移方程：

f(i,j)=a(i,j)+max{f(i+1,j),f(i+1,j+1)}。

动态规划的题目按状态通常可以分为以下几类：线性动态规划、坐标类动态规划、区间动态规划、背包类动态规划、树型动态规划等。

一、线性模型：

线性动态规划的状态一般是一维的(f[i])，第 i 个元素的最优值只与前 i-1 个元素的最优值或第 i+1 个元素之后的最优值有关。

一个数的序列 b_1，当 $b_1 < b_2 < ... < b_S$ 的时候，我们称这个序列是上升的。对于给定的一个序列(a_1, a_2, ..., a_N)，我们可以得到一些上升的子序列(a_{i1}, a_{i2}, ..., a_{iK})，这里 1 <= i_1 < i_2 < ... < i_K <= N。比如，对于序列(1, 7, 3, 5, 9, 4, 8)，有它的一些上升子序列，如(1, 7), (3, 4, 8)等等。这些子序列中最长的长度是 4，比如子序列(1, 3, 5, 8).

例 3.4.2：拦截导弹,noi.openjudge.cn/ch0206/8780

总时间限制：1000ms

内存限制：65536kB

描述

某国为了防御敌国的导弹袭击，发展出一种导弹拦截系统。但是这种导弹拦截系统有一个缺陷：虽然它的第一发炮弹能够到达任意的高度，但是以后每一发炮弹都不能高于前一发的高度。某天，雷达捕捉到敌国的导弹来袭。由于该系统还在试用阶段，所以只有一套系统，因此有可能不能拦截所有的导弹。

输入导弹依次飞来的高度（雷达给出的高度数据是不大于 30000 的正整数），计算这套系统最多能拦截多少导弹。

输入：

第一行是一个整数 N（不超过 15），表示导弹数。

第二行包含 N 个整数，为导弹依次飞来的高度（雷达给出的高度数据是不大于 30000 的正整数）。

输出:一个整数，表示最多能拦截的导弹数。

样例输入：	样例输出：
8 389 207 155 300 299 170 158 65	6

分析：

问题的本质是给出 h1,h2,……,hn，找一个最长的子序列，满足 hi1<=hi2<=……<=hik。

定义 f(i)表示：从前向后，以 h[i]为最后一个元素的最长不上升子序列的长度。

要求解 f(i)，应该在所有满足条件（1<=j<i 且 h[j]>=h[i]）的 f(j)中取最优值，再加 1。

f(i)的取值只与前 i-1 个数有关，枚举前 i-1 个状态进行状态转移。

f(i)=max{f(j)}+1 (1<=j<i 且 h[j]>=h[i])

最终答案 ans=max{f(i)}　(1<=i<=n)。时间复杂度 O(n^2)。

例 3.4.3：开餐馆,noi.openjudge.cn/ch0206/6045

总时间限制：1000ms

内存限制：65536kB

描述

北大信息学院的同学小明毕业之后打算创业开餐馆.现在共有 n 个地点可供选择。小明打算从中选择合适的位置开设一些餐馆。这 n 个地点排列在同一条直线上。我们用一个整数序列 m_1, m_2, ... m_n 来表示他们的相对位置。由于地段关系,开餐馆的利润会有所不同。我们用 p_i 表示在 m_i 处开餐馆的利润。为了避免自己的餐馆的内部竞争,餐馆之间的距离必须大于 k。请你帮助小明选择一个总利润最大的方案。

输入：

标准的输入包含若干组测试数据。输入第一行是整数 T (1 <= T <= 1000)，表明有 T 组测试数据。紧接着有 T 组连续的测试。每组测试数据有 3 行,

第 1 行:地点总数 n (n < 100), 距离限制 k (k > 0 && k < 1000).

第 2 行:n 个地点的位置 m1 , m2, ... mn (1000000 > mi > 0 且为整数,升序排列)

第 3 行:n 个地点的餐馆利润 p1 , p2, ... pn (1000 > pi > 0 且为整数)

输出:对于每组测试数据可能的最大利润。

样例输入：	样例输出：
2 3 11 1 2 15 10 2 30 3 16 1 2 15 10 2 30	40 30

分析：

定义 f(i)表示：从 1..i 个地点中，在第 i 个地点开餐馆的最大利润。

要求解 f(i)，应该在所有满足条件（1<=j<i 且 m(i)-m(j)>k）的 f(j)中取最优值，再加上 pi。

f(i)的取值只与前 i-1 个数有关，枚举前 i-1 个状态进行状态转移。

f(i)=max{f(j)}+p(i) (1<=j<i 且 m(i)-m(j)>k。

最终答案 ans=max{f(i)}　　1<=i<=n。

例 3.4.4：计算字符串距离,noi.openjudge.cn/ch0206/2988

总时间限制：1000ms

内存限制：65536kB

描述

对于两个不同的字符串，我们有一套操作方法来把他们变得相同，具体方法为：

修改一个字符（如把"a"替换为"b"）

删除一个字符（如把"traveling"变为"travelng"）

比如对于"abcdefg"和"abcdef"两个字符串来说，我们认为可以通过增加/减少一个"g"的方式来

达到目的。无论增加还是减少"g"，我们都仅仅需要一次操作。我们把这个操作所需要的次数定义为两个字符串的距离。

给定任意两个字符串，写出一个算法来计算出他们的距离。

输入：

第一行有一个整数 n。表示测试数据的组数，

接下来共 n 行，每行两个字符串，用空格隔开。表示要计算距离的两个字符串字符串长度不超过 1000。

输出：针对每一组测试数据输出一个整数，值为两个字符串的距离。

样例输入：	样例输出：
3 abcdefg abcdef ab ab mnklj jlknm	1 0 4

分析：

定义 f(i1,i2)表示：s1 的前 i1 个字符与 s2 的前 i2 个字符之间的距离。

如果 s1[i1]==s2[i2]，

f(i1,i2)=s1 的前 i1-1 个字符与 s2 的前 i2-1 个字符之间的距离=f(i1-1,i2-1);

否则

删除 s1[i1]，状态转移到 f(i1-1,i2);

删除 s2[i2]，状态转移到 f(i1,i2-1);

修改 s1[i1]或 s2[i2]，使得 s1[i1]==s2[i2]，状态转移到 f(i1-1,i2-1);

f(i1,i2)取上述三种情况的最优值，再加 1。

初始值：f(i1,0)=i1; f(0,i2)=i2;

最终答案 ans=f(len1,len2)。

二、坐标类模型：

通常以二维空间为模版展开，状态转移方程也在坐标型中寻找，一般 f(i,j)表示坐标位置(i,j)的状态。

例 3.4.5：最低通行费，noi.openjudge.cn/ch0206/7614

总时间限制：1000ms

内存限制：65536kB

描述

一个商人穿过一个 N*N 的正方形的网格，去参加一个非常重要的商务活动。他要从网格的左上角进，右下角出。每穿越中间 1 个小方格，都要花费 1 个单位时间。商人必须在(2N-1)个单位时间穿越出去。而在经过中间的每个小方格时，都需要缴纳一定的费用。

这个商人期望在规定时间内用最少费用穿越出去。请问至少需要多少费用？

注意：不能对角穿越各个小方格（即，只能向上下左右四个方向移动且不能离开网格）。

输入：

第一行是一个整数，表示正方形的宽度 N (1 <= N < 100)；

后面 N 行，每行 N 个不大于 100 的整数，为网格上每个小方格的费用。

输出：至少需要的费用。

样例输入：	样例输出：
5 1 4 6 8 10 2 5 7 15 17 6 8 9 18 20	109

10 11 12 19 21 20 23 25 29 33	

提示：样例中，最小值为 109=1+2+5+7+9+12+19+21+33。

分析：

每一步只能向下或向右。（N*N，走 2N-1 步）

定义 f(i,j)表示：走到位置(i,j)的最低通行费。

考虑(i,j)的前一个位置，要么是(i-1,j)，要么是(i,j-1)，容易得出状态转移方程：

f(i,j)=min{f(i-1,j),f(i,j-1)}+g(i,j)

初始值：f(1,1,)=g(1,1)，最终答案 ans=f(n,n)。

注意边界情况：f(i,1)和 f(1,j)的计算。

三、区间模型

区间型动态规划是线性动态规划的拓展，它将区间长度作为阶段，长区间的最优值取决于相应短区间的最优值。

例 3.4.6：石子归并,codevs.cn/1048

时间限制：1 s

空间限制：128000 KB

题目描述

有 n 堆石子排成一列，每堆石子有一个重量 w[i]，每次合并可以合并相邻的两堆石子，一次合并的代价为两堆石子的重量和 w[i]+w[i+1]。问安排怎样的合并顺序，能够使得总合并代价达到最小。

输入描述

第一行一个整数 n（n<=100）

第二行 n 个整数 w1,w2...wn （wi <= 100)

输出描述：一个整数表示最小合并代价

样例输入：	样例输出：
4 4 1 1 4	18

分析：

贪心：每次选相邻石子和最小的两队进行合并。

虽然样例能得出正确答案，但存在反例：n=4, 7 4 4 7。

从合并的最后一步看，合并第 1 堆到第 n 堆，可以看作（合并第 1 对到第 k 堆）和（合并第 k+1 堆到第 n 堆）两部分，再进行 1 次合并，最优答案通过枚举 k 求得。这两个区间互不相干，每个区间都可以由更小的区间合并来求解。

区间起点为 i，终点为 j，定义 f(i,j)表示把区间[i,j]合并为一堆的最小代价。

f(i,j)=min{f(i,k)+f(k+1,j)}+sum(i,j) (i<=k<j)

sum(i,j)表示 a[i]+a[i+1]+……+a[j].

i,j 的枚举顺序不明显，可以考虑记忆化搜索的方法。

```
int dfs(int left,int right){
    //单独 1 堆石子，合并代价是 0
    if(left==right) return 0;
    if(f[left][right]) return f[left][right];
    f[left][right]=inf; //理论最大值
    for(int k=left;k<right;k++)
```

```
    f[left][right]=min(f[left][right],dfs(left,k)+dfs(k+1,right)+s
um[left][right]);
        return f[left][right];
    }
```

递推的思路：以区间长度为阶段，(i,j)表示以第 i 堆开始的 j 堆，也就是[i,i+j-1]，f(i,j)表示把[i,i+j-1]合并成 1 堆的最小代价，状态转移方程：

f(i,j)=min{f(i,k)+f(i+k,j-k)}+sum(i,i+j-1)　(1<=k<j)

初始值：f(i,1)=0

最终答案 ans=f(1,n)。

```
    for(int j=2;j<=n;++j){
            for(int i=1;i<=n+1-j;++i){
                for(int k=1;k<j;++k)
                    f[i][j]=f[i][k]+f[i+k][j-k]+sum(i,i+j-1);
            }
        }
```

拓展：codevs.cn/2102"n 堆石子排成 1 列"改为"n 堆石子围成 1 圈"，如何求解？

四、01 背包

例 3.4.7：采药,noi.openjudge.cn/ch0206/1775,noip2005 普及组

总时间限制：1000ms

内存限制：65536kB

描述

辰辰是个很有潜能、天资聪颖的孩子，他的梦想是称为世界上最伟大的医师。为此，他想拜附近最有威望的医师为师。医师为了判断他的资质，给他出了一个难题。医师把他带到个到处都是草药的山洞里对他说："孩子，这个山洞里有一些不同的草药，采每一株都需要一些时间，每一株也有它自身的价值。我会给你一段时间，在这段时间里，你可以采到一些草药。如果你是一个聪明的孩子，你应该可以让采到的草药的总价值最大。"

如果你是辰辰，你能完成这个任务吗？

输入：

输入的第一行有两个整数 T（1 <= T <= 1000）和 M（1 <= M <= 100），T 代表总共能够用来采药的时间，M 代表山洞里的草药的数目。接下来的 M 行每行包括两个在 1 到 100 之间（包括 1 和 100）的的整数，分别表示采摘某株草药的时间和这株草药的价值。

输出：

输出只包括一行，这一行只包含一个整数，表示在规定的时间内，可以采到的草药的最大总价值。

样例输入	样例输出
70 3 71 100 69 1 1 2	3

分析：

定义 f(i,j)表示在第 1 株到第 i 株草药中采摘，花费时间 j 能获得的最大价值。

考察第 i 株草药，有采和不采两种情况：

1.f(i-1,j)：不采第 i 株，在第 1 株到第 i-1 株草药中采，花费时间 j；

2.f(i-1,j-t(i))+v[i]:采第 i 株，剩余时间 j-t(i)，在第 1 株到第 i-1 株草药中采，前提是 j>=t(i)。

状态转移方程：

f(i,j)=max{f(i-1,j),f(i-1,j-t(i))+v[i] (j>=t(i))} (1<=i<=M,0<=j<=T)。

初始值：

if(j<t(1)) f(1,j)=0 else f(1,j)=v[1];

最终答案 ans=f(M,T)。

例 3.4.8：糖果，noi.openjudge.cn/ch0206/2989

总时间限制：1000ms

内存限制：65536kB

描述

由于在维护世界和平的事务中做出巨大贡献，Dzx 被赠予糖果公司 2010 年 5 月 23 日当天无限量糖果免费优惠券。在这一天，Dzx 可以从糖果公司的 N 件产品中任意选择若干件带回家享用。糖果公司的 N 件产品每件都包含数量不同的糖果。Dzx 希望他选择的产品包含的糖果总数是 K 的整数倍，这样他才能平均地将糖果分给帮助他维护世界和平的伙伴们。当然，在满足这一条件的基础上，糖果总数越多越好。Dzx 最多能带走多少糖果呢？

注意：Dzx 只能将糖果公司的产品整件带走。

输入：

第一行包含两个整数 N(1<=N<=100)和 K(1<=K<=100)

以下 N 行每行 1 个整数，表示糖果公司该件产品中包含的糖果数目，不超过 1000000。

输出：

符合要求的最多能达到的糖果总数，如果不能达到 K 的倍数这一要求，输出 0。

样例输入：	样例输出：
5 7 1 2 3 4 5	14

提示：Dzx 的选择是 2+3+4+5=14，这样糖果总数是 7 的倍数，并且是总数最多的选择。

分析：

定义 f(i,j)表示：从前 i 种产品中选取，能否得到 j 个糖果。

如果不选第 i 种，f(i,j)=f(i-1,j) 从前 i-1 种产品中选取，能否得到 j 个糖果；

如果选第 i 种，f(i,j)=f(i-1,j-a[i]) 从前 i-1 种产品中选取，能否得到 j-a[i]个糖果；(j>=a[i])

f(i,j)=f(i-1,j)|f(i-1,j-a[i])

初始值：f(i,0)=true;

最终答案：满足 f(n,j)==true 的最大的 j。

j 的取值范围：n*1000000=10^8

时间复杂度：O(n*j)=O(10^10)，超时。

效率低的原因：糖果的总数量规模大，数据 k(k<=100)在动规中没有体现出来。

改进：

从前 i 种产品中选取，能取得的糖果数量，根据%k 所得的余数分成 k 类：

f(i,0),f(i,1),……,f(i,k-1)。

对于每一类，只需保留最大值即可。

考虑 f(i,j)：

如果不选第 i 种产品，f(i,j)=f(i-1,j)；

如果选第 i 种产品，余数是 j，从前 i-1 种产品中选取时，余数为 x，需满足 (x+a[i])%k=j

x=(k+j-a[i]%k)%k;

如果 x==0，那么 j==a[i]%k，f(i,j)可以由 f(i-1,0)+a[i]求得；

x!=0 时，如果 f(i-1,x)非零，f(i,j)可以由 f(i-1,x)+a[i]求得。

复杂度：O(n*k)=O(10^4)。

3.5 图的最短路径（黄启红）

【知识点】

掌握图的传递闭包、最短路的概念，掌握最短路的求解方法，理解 **floyd 算法**、**dijkstra 算法**的适用范围，各种形式的灵活应用。

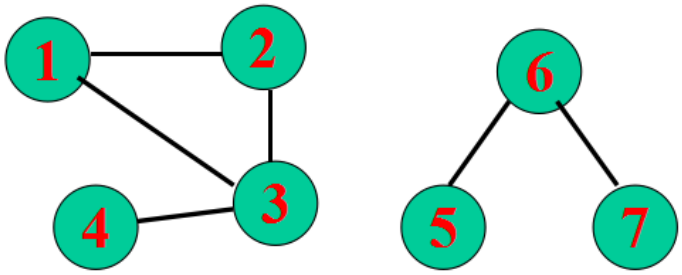
【知识讲解】

1.图的传递闭包

求解问题：对于图 G 的任意顶点 i 和 j，是否存在一条从 i 到 j 的路径（即 i 和 j 是否可达）？

（1） 对于图 G 的任意一个顶点 u，u 可达的顶点集合称为 u 的传递闭包。在无向图中，u 的传递闭包为和 u 处于同一连通分量的点集。

如下图：



邻接矩阵 G:　i 和 j 是否有边相连？

```
0 1 1 0 0 0 0
1 0 1 0 0 0 0
1 1 0 1 0 0 0
0 0 1 0 0 0 0
0 0 0 0 0 1 0
0 0 0 0 1 0 1
0 0 0 0 0 1 0
```

传递闭包 G*: i 和 j 是否有路相连？

```
1 1 1 1 0 0 0
1 1 1 1 0 0 0
1 1 1 1 0 0 0
1 1 1 1 0 0 0
0 0 0 0 1 1 1
0 0 0 0 1 1 1
0 0 0 0 1 1 1
```

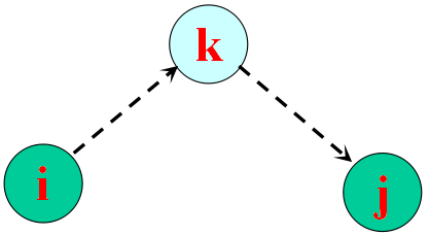
（2）求解传递闭包的算法

判断结点 i 到 j 是否有路径

① 结点 i 到 j 有边则有路径。

② i 到 j 之间没有边：

如果存在另外的一个结点 k，满足：i 到 k 有路径，k 到 j 有路径，则 i 到 j 有路径。否则 i 到 j 没有路径。



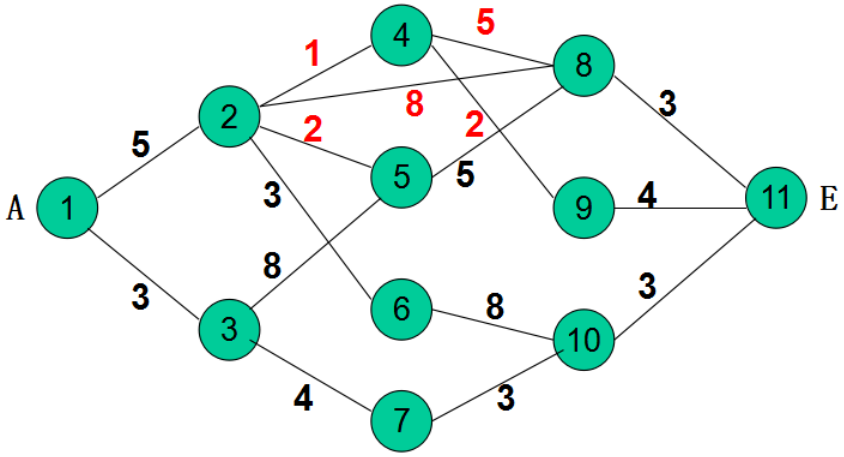
则：can[i][j]=can[i][j] || (can[i][k] && can[k][j])

初始化：can[i][j]=true:i 到 j 有边；　can[i][j]= false:无边。can[i][i]=true；
求解过程：

```
for(int k=1;k<=n;k++)   枚举中间点
    for(int i=1;i<=n;i++)    枚举起点
        for(int j=1;j<=n;j++)      枚举终点
            can[i][j]=can[i][j]||(can[i][k]&&can[k][j]);
```

2. 图的最短路径

【引例】已知各个城市之间的道路情况如下：



现在，我们想从城市 A 到达城市 E。
怎样走才能使得路径最短，最短路径的长度是多少？

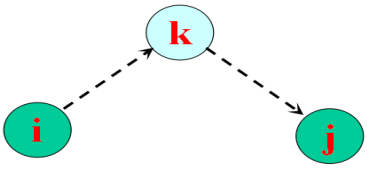
两类问题：
1、图中每对顶点（任意两点）之间的最短路径　（弗洛伊德算法：f loyed）。
2、图中一个顶点到其他顶点的最短路径　（迪杰斯特拉算法：dijkstra）。

一）、计算每一对顶点间的最短路径（floyd 算法）

（一）求最短距离

目标：把图中任意两点 i 与 j 之间的最短距离都求出来 d[i][j]。

原理：根据图的传递闭包思想：



if　(d[i][k]+d[k][j]<d[i][j])　　d[i][j]=d[i][k]+d[k][j]

算法思想：

设顶点集 S(初值为空)，用数组 D 的每个元素 D[i][j]保存从 V_i 只经过 S 中的顶点到达 V_j 的最短路径长度。

① 初始时令 S={ } ， D[i][j]的赋初值方式是：

② 将图中一个顶点 V_k 加入到 S 中，修改 D[i][j]的值，修改方法是：

D[i][j]=Min{D[i][j] , (D[i][k]+D[k][j]) }

原因： 从 V_i 只经过 S 中的顶点(V_k)到达 V_j 的路径长度可能比原来不经过 V_k 的路径更短。

③ 重复②，直到 G 的所有顶点都加入到 S 中为止。

$$D[i][j]=\begin{cases}0 & i=j \text{ 时}\\ W_{ij} & i\neq j \text{ 且} <v_i,v_j>\in E,\ w_{ij}\text{为弧上的权值}\\ \infty & i\neq j \text{ 且} <v_i,v_j>\text{不属于 } E\end{cases}$$

算法实现：

```
for (int k=1;k<=n;k++)
        for (int i=1;i<=n;i++)
              for (int j=1;j<=n;j++)
                   if (d[i][k]+d[k][j]<d[i][j])       d[i][j]=d[i][k]+d[k][j];
```

初始化条件：

D[i][i]=0; //自己到自己为 0；对角线为 0；

D[i][j]=边权，i 与 j 有直接相连的边

D[i][j]= +∞ ，i 与 j 无直接相连的边。

// 如果是 int 数组，采用 memset(d, 0x7f, sizeof(d))可全部初始化为一个很大的数

举例：

已知下图中给定的关系，顶点个数 n<=100,两点之间的距离 w<=1000,求给定两点之间的最短距离.

输入：

5

1　5

1 2 23

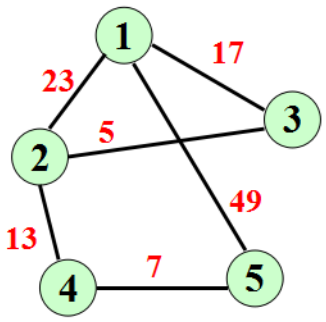
1 3 17

1 5 49

2 3 5

2 4 13

4 5 7



要求：输出最短距离 d[1][5]。

分析：D[i][j]:表示顶点 i 到顶点 j 之间的最短距离。

初始化如下：

0	23	17	49	∞
23	0	5	13	∞
17	5	0	∞	∞
∞	13	∞	0	7
49	∞	∞	7	0

floyed

0	22	17	35	42
22	0	5	13	20
17	5	0	18	25
35	13	18	0	7
42	20	25	7	0

参考代码：

```
#include<iostream>
#include<cstring>
using namespace std;
const int maxn=101;
const int maxw=1001;
int d[maxn][maxn];
int n,p,q;
void init(){
    cin>>n;cin>>p>>q;
    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
            if(i==j) d[i][j]=0;else d[i][j]=maxw;
    int i,j,k;
    while(cin>>i>>j>>k){
        d[i][j]=k;d[j][i]=k;
    }
}
void floyed(){
  for(int k=1;k<=n;k++)
   for(int i=1;i<=n;i++)
    for(int j=1;j<=n;j++)
      if (d[i][k]+d[k][j]<d[i][j])
          d[i][j]=d[i][k]+d[k][j];
};
void print(){
    cout<<d[p][q]<<endl;
}
int main(){
    init();
    floyed();
    print();
    return 0;
}
```

（二） floyed 输出最短路径路线

方法一：

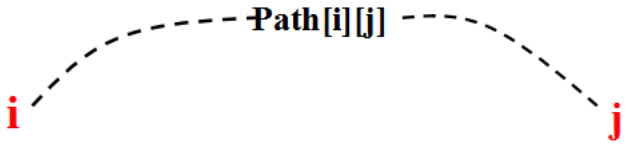
定义二维数组 Path[n][n](n 为图的顶点数) ， 元素 Path[i][j]保存从 Vi 到 Vj 的最短路径所经过的顶点。

①初始化为 Path[i][j]= - 1，表示从 Vi 到 Vj 不经过任何(S 中的中间)顶点。

②当某个顶点 V_k 加入到 S 中后使 A[i][j]变小时，令 Path[i][j]=k。

若 Path[i][j]==k：从 V_i 到 V_j 经过 V_k ，最短路径序列是(V_i，…, V_k，…, V_j) ，则路径子序列：(V_i，…, V_k)和(V_k，…, V_j)一定是从 V_i 到 V_k 和从 V_k 到 V_j 的最短路径。从而可以根据 Path[i][k]和 Path[k][j]的值再找

到该路径上所经过的其它顶点，…依此类推。



参考代码：

```
for (int k=1;k<=n;k++)
        for (int i=1;i<=n;k++)
             for (int j=1;j<=n;j++)
                if (d[i][k]+d[k][j]<d[i][j]) {
                       d[i][j]=d[i][k]+d[k][j];
                       path[i][j]=k;
                  };
```

初始化：

path[i][j]= -1; i 与 j 不经过任何中间点

输出 i 到 j 的最短路径：

输出 i;　　dfs(i,j);　　输出 j;

参考代码：

```
void dfs(int i, int j){ //  输出 i 到 j 之间的点，不包含 i 和 j 结点；
        if (path[i][j]!= -1)
             {
                dfs(i,path[i][j]);
                cout<<path[i][j]<<' ';
                dfs(path[i][j],j);
             };
 };
```

方法二：设 path[i][j] 记录 i 到 j 的最短路径中 j 的前驱顶点。

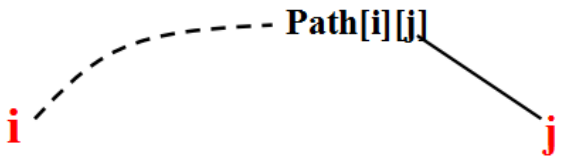
如样例：

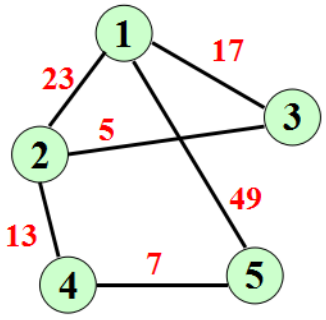
1-->4: 35: 1 3 2 4

path[1][4]=2

path[1][2]=3

path[1][3]=1






```
for (int k=1;k<=n;k++)
        for (int i=1;i<=n;k++)
             for (int j=1;j<=n;j++)
                if (d[i][k]+d[k][j]<d[i][j]) {
                       d[i][j]=d[i][k]+d[k][j];
                       path[i][j]=path[k][j];
                   };
```

- 初始化：

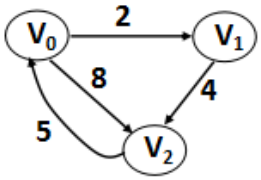
i 到 j 有边，则 path[i][j]=i;　path[j][i] =j;

- 输出 i 到 j 的路线

```
void dfs(int i,int j){
        if (i==j)   cout<<i<<' ';
        else if (path[i][j]>0)
           {
             dfs(i,path[i,j]);
             cout<<j<<' ';
           };
      };
```

拓展：

以右图为例，演示用 floyed 算法求任意一对顶点间最短路径过程。



$$\begin{pmatrix} 0 & 2 & 8 \\ \infty & 0 & 4 \\ 5 & \infty & 0 \end{pmatrix}$$

带权有向图及其邻接矩阵

用Floyd算法求任意一对顶点间最短路径过程演示

步骤	初态	k=0	K=1	K=2
A	$\begin{pmatrix} 0 & 2 & 8 \\ \infty & 0 & 4 \\ 5 & \infty & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 2 & 8 \\ \infty & 0 & 4 \\ 5 & 7 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 2 & 6 \\ \infty & 0 & 4 \\ 5 & 7 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 2 & 6 \\ 9 & 0 & 4 \\ 5 & 7 & 0 \end{pmatrix}$
Path	$\begin{pmatrix} -1 & -1 & -1 \\ -1 & -1 & -1 \\ -1 & -1 & -1 \end{pmatrix}$	$\begin{pmatrix} -1 & -1 & -1 \\ -1 & -1 & -1 \\ -1 & 0 & -1 \end{pmatrix}$	$\begin{pmatrix} -1 & -1 & 1 \\ -1 & -1 & -1 \\ -1 & 0 & -1 \end{pmatrix}$	$\begin{pmatrix} -1 & -1 & 1 \\ 2 & -1 & -1 \\ -1 & 0 & -1 \end{pmatrix}$
S	{ }	{ 0 }	{ 0, 1 }	{ 0, 1, 2 }

综合应用举例 1：产生数(noip2001)

【问题描述:】

给出一个正整数 n（n<10^50) 和 k 个变换规则（k<=15）。每个变换规则是指：

一位数可变换成另一个一位数：规则的右部不能为零。

例如：n=234。有规则（k＝2）：

2－> 5　　3－> 6

上面的整数 234 经过变换后可能产生出的整数为（包括原数）：

234　　534　　264　　564　　共 4 种不同的产生数。

【任务：】

给出一个整数 n 和 k 个变换规则。

求经过任意次的变换（0 次或多次），能产生出多少个不同整数。仅要求输出个数。

【输入：】

第一行：n。第二行：k。以下 k 行：每行两个一位数：x　y，中间一个空格，表示一个变换规则：x 可以变为 y。

【输出：】

一个整数（满足条件的个数）：

【输入样例 1】

234

2

2　5

3　6

【样例输出 1】

4

【输入样例 2】

234

7

2　5

2　8

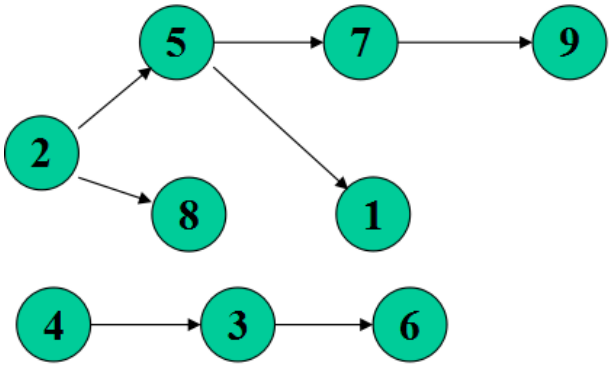
5　7

5　1

7　9

3　6

4　3



【样例输出 2】

36

分析：

①乘法原理直接进行计数。

用 f[i]表示数字 i 包括本身可以变成的数字总个数

这里的变成可以是直接变成也可以是间接变成：

比如　3->5,　5->7,那么　3->7

那么对于一个数 a（用数组存,长度为 n），根据乘法原理它能产生出不同整数数量：

f[a[1]]*f[a[2]]*f[a[3]]*…*f[a[n]]

2：6

3：2

4：3

6*2*3=36

②最多可能有 50 位数 每一位最多有 9 种变换方法，所以最大可能 9^50

需要模拟大数乘法 而每次乘数有一个 不超过 10 所以并不难。

参考代码：

```
#include<iostream>
#include<cstring>
```

```
#include<string>
bool f[10][10];
int k,ans[500]={1},l=1;
using namespace std;
void wk(int x)//高精度乘法
{
    for(int i=0;i<l;i++)
        ans[i]*=x;
    for(int i=0;i<l;i++)
        if(ans[i]>=10){ans[i+1]+=ans[i]/10;ans[i]%=10;}
    while(ans[l]>0){
        ans[l+1]=ans[l]/10;
        ans[l]=ans[l]%10;
        l++;
    }
}
 int main(){
    string a;
    cin>>a>>k;
    int x,y;
    for(int i=1;i<=k;i++) { cin>>x>>y;f[x][y]=1; }
    for(int i=0;i<=9;i++) f[i][i]=1;   //数字是可以转化为自己的
    for(int k=1;k<=9;k++)      //用 floyd 计算每个数可以转换的数字(包括本身)
        for(int i=0;i<=9;i++)
            for(int j=1;j<=9;j++)
                if(f[i][k]&&f[k][j])f[i][j]=1;
    int b[10];
    for(int i=0;i<=9;i++){     //统计每个数可以转换的数字数
        int tot=0;
        for(int j=0;j<=9;j++)
            if(f[i][j]) tot++;
        b[i]=tot;
    }
    for(int i=0;i<a.length();i++)     wk(b[a[i]-'0']);   //乘法原理
    for(int i=l-1;i>=0;i--)     cout<<ans[i];
    return 0;
}
```

二）、计算某一顶点到其它所有顶点的最短路径　（单源最短路径问题：dijkstra 算法）

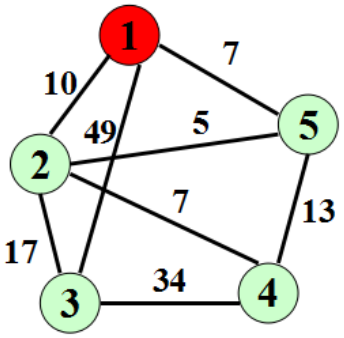
开始点（源点）：start

D[i]:顶点 start 到 i 的最短距离。

初始：

D[start]=0；

D[i]=a[start][i]



dijkstra 算法：
将顶点分为两个集合：已求得最短距离的点集合 1，待求点集合 2.
1、在集合 2 中找一个到 start 距离最近的顶点 k : min{d[k]}
2、把顶点 k 加到集合 1 中，同时修改集合 2 中的剩余顶点 j 的 d[j]是否经过 k 后变短。如果变短修改 d[j]
If (d[k]+a[k][j]<d[j])　d[j]=d[k]+a[k][j]
3、重复 1，直至集合 2 空为止。

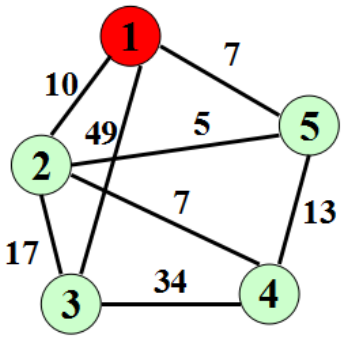
数据结构：
F[i]，值为 true，已求得最短距离，在集合 1 中，值为 false，在集合 2 中；
D[i]， start 到 i 的最短距离;
Path[i], i 的前驱结点
初始表：

顶点	**1**	**2**	**3**	**4**	**5**
F[i]	**T**	**F**	**F**	**F**	**F**
D[i]	**0**	**10**	**49**	**∞**	**7**
Path[i]	**1**	**1**	**1**		**1**

最终表：

顶点	1	2	3	4	5
F[i]	T	T	T	T	T
D[i]	0	10	27	17	7
Path[i]	1	1	2	2	1

数据输入：
5
1
1 2 10
1 5 7
2 3 17



2 4 7
2 5 5
3 4 34
4 5 13

参考代码：

```
#include<iostream>
#include<cstring>
using namespace std;
const int maxn=101;
int f[maxn],a[maxn][maxn],d[maxn],path[maxn];
int n,start;
const int inf=99999;
void init(){
    cin>>n>>start;
    int x,y,w;
    for(int i=1;i<=n;i++)
       for(int j=1;j<=n;j++){
          if(j==i) a[i][j]=0;
          else a[i][j]=inf;
       }

    while(cin>>x>>y>>w){
          a[x][y]=w;
          a[y][x]=w;
    }
}
void dijkstra(int s){
    for (int i=1;i<=n;i++)   { d[i]=a[s][i]; f[i]=false;path[i]=s;   }
    f[s]=true;    //   加入集合 1
    for (int i=2;i<=n;i++){
          int mind=inf;
          int k=0;//用来记录准备放入集合 1 的点
          for (int j=1;j<=n;j++)   //查找集合 2 中 d[]最小的点
                if   ( (!f[j]) &&(d[j]<mind) ){ mind=d[j]; k=j; };
          if   (mind==inf)   break; //更新结点求完了
          f[k]=true;    //   加入集合 1
          for (int j=1;j<=n;j++)   //修改集合 2 中的 d[j]
          if ((!f[j]) && (d[k]+a[k][j]<d[j])){ d[j]=d[k]+a[k][j];path[j]=k;}
    };
}
void dfs(int i){
    if(i!=start) dfs(path[i]);
    cout<<i<<' ';
```

```
}
void write(){
    cout<<start<<"到其余各点的最短距离是："<<endl;
    for(int i=1;i<=n;i++){
        if(i!=start){
            if(d[i]==inf) cout<<i<<"不可达！";
            else{
            cout<<i<<"的最短距离："<<d[i] <<",依次经过的点是：";
            dfs(path[i]);
            cout<<i<<endl;
            }
        }
    }
}

int main(){
    init();
    dijkstra(start);
    write();
}
```

【例 1】、最短路径问题

【问题描述】

平面上有 n 个点（n<=100），每个点的坐标均在-10000~10000 之间。其中的一些点之间有连线。若有连线，则表示可从一个点到达另一个点，即两点间有通路，通路的距离为两点间的直线距离。现在的任务是找出从一点到另一点之间的最短路径。

【输入格式】

输入文件为 short.in，共 n+m+3 行，其中:

第一行为整数 n。

第 2 行到第 n+1 行（共 n 行），每行两个整数 x 和 y，描述了一个点的坐标。

第 n+2 行为一个整数 m，表示图中连线的个数。

此后的 m 行，每行描述一条连线，由两个整数 i 和 j 组成，表示第 i 个点和第 j 个点之间有连线。

最后一行：两个整数 s 和 t，分别表示源点和目标点。

【输出格式】

输出文件为 short.out，仅一行，一个实数（保留两位小数），表示从 s 到 t 的最短路径长度。

【输入样例】

```
5
0 0
2 0
2 2
0 2
3 1
5
1 2
```

1 3
1 4
2 5
3 5
1 5
【输出样例】
3.41

【参考程序】

```
#include<cstdio>
#include<iostream>
#include<cmath>
#include<cstring>
using namespace std;
int a[101][3];
double f[101][101];
int n,i,j,k,x,y,m,s,e;
int main()
{
    freopen("short.in","r",stdin);
    freopen("short.out","w",stdout);
    cin >> n;
    for (i = 1; i <= n; i++)
        cin >> a[i][1] >> a[i][2];
    cin >> m;
    memset(f,0x7f,sizeof(f));                    //初始化 f 数组为最大值
    for (i = 1; i <= m; i++)                     //预处理出 x、y 间距离
    {
      cin >> x >> y;
      f[y][x] = f[x][y] = sqrt(pow(double(a[x][1]-a[y][1]),2)+pow(double(a[x][2]-a[y][2]),2));
                          //pow(x,y)表示 x^y，其中 x,y 必须为 double 类型，要用 cmath 库
    }
    cin >> s >> e;
    for (k = 1; k <= n; k++)                     //floyed  最短路算法
      for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++)
          if ((i != j) && (i != k) && (j != k) && (f[i][k]+f[k][j] < f[i][j]))
            f[i][j] = f[i][k] + f[k][j];
    printf("%.2lf\n",f[s][e]);
    return 0;
}
```

【例 2】最小花费
【问题描述】
在 n 个人中，某些人的银行账号之间可以互相转账。这些人之间转账的手续费各不相同。给定这些

人之间转账时需要从转账金额里扣除百分之几的手续费，请问 A 最少需要多少钱使得转账后 B 收到 100 元。

【输入格式】

第一行输入两个正整数 n,m，分别表示总人数和可以互相转账的人的对数。

以下 m 行每行输入三个正整数 x,y,z，表示标号为 x 的人和标号为 y 的人之间互相转账需要扣除 z% 的手续费 (z<100)。

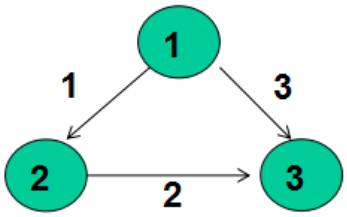
最后一行输入两个正整数 A,B。数据保证 A 与 B 之间可以直接或间接地转账。

【输出格式】

输出 A 使得 B 到账 100 元最少需要的总费用。精确到小数点后 8 位。

【输入样例】

3 3
1 2 1
2 3 2
1 3 3
1 3



【输出样例】

103.07153164

【数据规模】

1<=n<=2000

【参考程序】

```
#include<iostream>
using namespace std;
double a[2001][2001],dis[2001]={0},minn;
int n,m,i,j,k,x,y,f[2001]={0};
void init()
{
   cin>>n>>m;
   for(i=1;i<=m;i++)
   {scanf("%d%d",&j,&k);
    scanf("%lf",&a[j][k]);
    a[j][k]=(100-a[j][k])/100;
    a[k][j]=a[j][k];
   }
   cin>>x>>y;
}
void dijkstra(int x)
{
   for(i=1;i<=n;i++)dis[i]=a[x][i];
   dis[x]=1;f[x]=1;
   for(i=1;i<=n-1;i++)
   { minn=0;
      for(j=1;j<=n;j++)
         if(f[j]==0&&dis[j]>minn){k=j;minn=dis[j];}
      f[k]=1;
```

```
        if(k==y)break;
        for(j=1;j<=n;j++)
          if(f[j]==0&&dis[k]*a[k][j]>dis[j])dis[j]=dis[k]*a[k][j];
      }
}
int main()
{
   init();
   dijkstra(x);
   printf("%0.8lf",100/dis[y]);
   return 0;
}
```

3.6 最小生成树（胡芳）

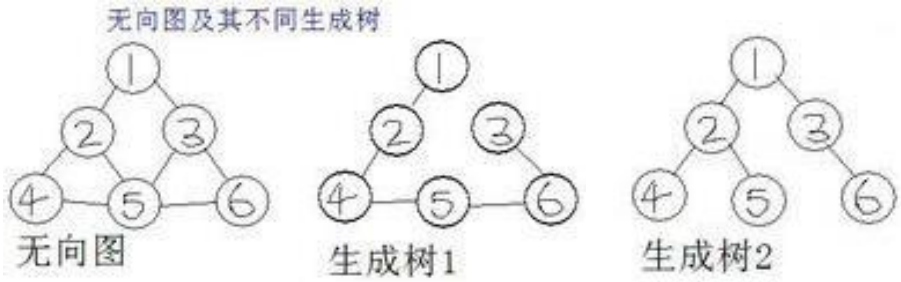
知识点

掌握图的最小生成树的概念，掌握 Prim 算法和 Kruskal 算法求图的最小生成树。

知识讲解

3.6.1 什么是图的最小生成树（MST）？

不知道大家还记不记得树的一个定理：N 个点用 N-1 条边连接成一个连通块，形成的图形只可能是树，没有别的可能。



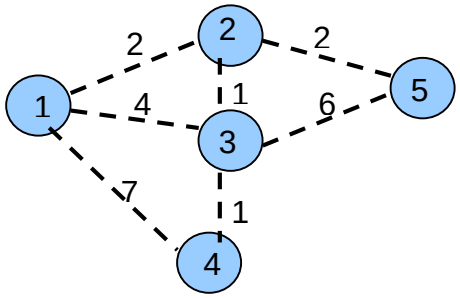
一个有 N 个点的图，边一定是大于等于 N-1 条的。图的最小生成树，就是在这些边中选择 N-1 条出来，连接所有的 N 个点。这 N-1 条边的边权之和是所有方案中最小的。

3.6.2 最小生成树用来解决什么问题 ？

就是用来解决如何用最小的“代价”用 N-1 条边连接 N 个点的问题。

【引例】

有一张城市地图，图中的顶点为城市，无向边代表两个城市间的连通关系，边上的权为在这两个城市之间修建高速公路的造价，研究后发现，这个地图有一个特点，即任一对城市都是连通的。现在的问题是，要修建若干高速公路把所有城市联系起来，问如何设计可使得工程的总造价最少？



3.6.3 用 Prim 算法求最小生成树

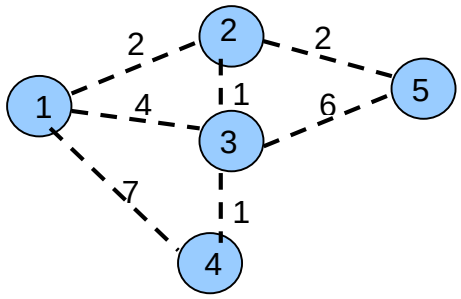
（1）算法分析&思想讲解：

Prim 算法采用“蓝白点”思想：白点代表已经进入最小生成树的点，蓝点代表未进入最小生成树的点。

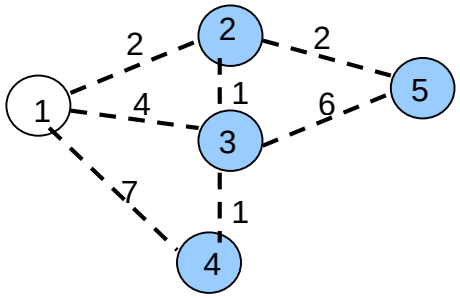
Prim 算法每次循环都将一个蓝点 u 变为白点，并且此蓝点 u 与白点相连的最小边权 min[u]还是当前所有蓝点中最小的。这样相当于向生成树中添加了 n-1 次最小的边，最后得到的一定是最小生成树。

我们通过对下图最小生成树的求解模拟来理解上面的思想。蓝点和虚线代表未进入最小生成树的点、边；白点和实线代表已进入最小生成树的点、边。

初始时所有点都是蓝点，min[1]=0,min[2、3、4、5]=∞。权值之和 MST=0。



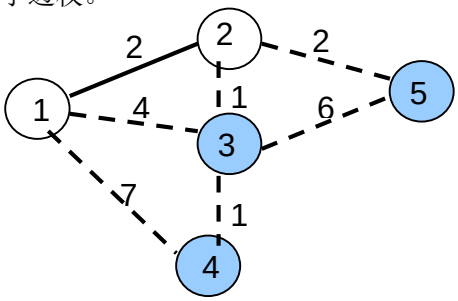
第一次循环自然是找到 min[1]=0 最小的蓝点 1。将 1 变为白点，接着枚举与 1 相连的所有蓝点 2、3、4，修改它们与白点相连的最小边权。



min[2]=w[1][2]=2;

min[3]=w[1][3]=4;

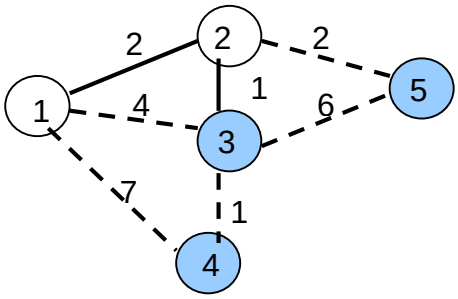
第二次循环是找到 min[2]最小的蓝点 2。将 2 变为白点，接着枚举与 2 相连的所有蓝点 3、5，修改它们与白点相连的最小边权。



min[3]=w[2][3]=1;

min[5]=w[2][5]=2;

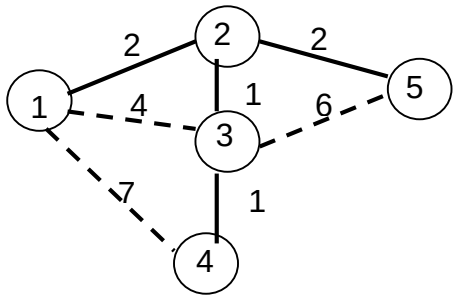
第三次循环是找到 min[3]最小的蓝点 3。将 3 变为白点，接着枚举与 3 相连的所有蓝点 4、5，修改它们与白点相连的最小边权。



min[4]=w[3][4]=1;

由于 min[5]=2 < w[3][5]=6;

最后两轮循环将点 4、5 以及边 w[2][5],w[3][4]添加进最小生成树。



最后权值之和 MST=6。

这 n 次循环，每次循环我们都能让一个新的点加入生成树，n 次循环就能把所有点囊括到其中；每次循环我们都能让一条新的边加入生成树，n-1 次循环就能生成一棵含有 n 个点的树；每次循环我们都取一条最小的边加入生成树，n-1 次循环结束后，我们得到的就是一棵最小的生成树。

这就是 Prim 采取贪心法生成一棵最小生成树的原理。算法时间复杂度：O (N2)。

（2）算法描述：

以 1 为起点生成最小生成树，min[v]表示蓝点 v 与白点相连的最小边权。

MST 表示最小生成树的权值之和。

a）初始化：min[v]= ∞(v≠1); min[1]=0;MST=0;

b）for (i = 1; i<= n; i++)

① 寻找 min[u]最小的蓝点 u。

② 将 u 标记为白点

③ MST+=min[u]

④ for 与白点 u 相连的所有蓝点 v

```
if (w[u][v]<min[v])
    min[v]=w[u][v];
```

c）算法结束： MST 即为最小生成树的权值之和

典型例题

例 3.6.1 最优布线问题 http://www.codevs.cn/problem/1231/

【问题描述】

学校有 n 台计算机，为了方便数据传输，现要将它们用数据线连接起来。两台计算机被连接是指它们间有数据线连接。由于计算机所处的位置不同，因此不同的两台计算机的连接费用往往是不同的。

当然，如果将任意两台计算机都用数据线连接，费用将是相当庞大的。为了节省费用，我们采用数据的间接传输手段，即一台计算机可以间接的通过若干台计算机（作为中转）来实现与另一台计算机的连接。

现在由你负责连接这些计算机，任务是使任意两台计算机都连通（不管是直接的或间接的）。

【输入格式】

输入文件 wire.in，第一行为整数 n（2<=n<=100），表示计算机的数目。此后的 n 行，每行 n 个整数。第 x+1 行 y 列的整数表示直接连接第 x 台计算机和第 y 台计算机的费用。

【输出格式】

输出文件 wire.out，一个整数，表示最小的连接费用。

【输入样例】

3

```
0 1 2
1 0 1
2 1 0
```

【输出样例】

```
2    （注：表示连接 1 和 2，2 和 3，费用为 2）
```

分析：这是一道最小生成树问题，可以用 Prim 算法求解。

【参考程序】

```
#include<iostream>
#include<cstdio>
#include<cstring>
using namespace std;
int g[101][101];                    //邻接矩阵
int minn[101];                      //minn[i]存放蓝点 i 与白点相连的最小边权
bool u[101];                        //u[i]=True，表示顶点 i 还未加入到生成树中
                                    //u[i]=False，表示顶点 i 已加入到生成树中
int n,i,j;
int main()
{
    cin >> n;
    for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++)
            cin >> g[i][j];
    memset(minn,0x7f,sizeof(minn));     //初始化为 maxint
    minn[1] = 0;
    memset(u,1,sizeof(u));              //初始化为 True，表示所有顶点为蓝点
for (i = 1; i <= n; i++)
    {
        int k = 0;
        for (j = 1; j <= n; j++)        //找一个与白点相连的权值最小的蓝点 k
            if (u[j] && (minn[j] < minn[k]))
                k = j;
        u[k] = false;                   //蓝点 k 加入生成树，标记为白点
        for (j = 1; j <= n; j++)        //修改与 k 相连的所有蓝点
            if (u[j] && (g[k][j] < minn[j]))
                minn[j] = g[k][j];
    }
    int total = 0;
    for (i = 1; i <= n; i++)            //累加权值
        total += minn[i];
    cout << total << endl;
    return 0;
}
```

说明：本算法在移动通信、智能交通、移动物流、生产调度等物联网相关领域都有十分现实的意义，

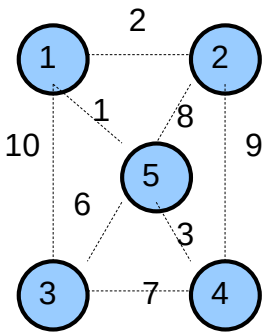
采用好的算法，就能节省成本提高效率。

3.6.4 用 Kruskal 算法求最小生成树

（1）算法分析&思想讲解：

Kruskal（克鲁斯卡尔）算法是一种巧妙利用并查集来求最小生成树的算法。Kruskal 算法将一个连通块当做一个集合。Kruskal 首先将所有的边按从小到大顺序排序（一般使用快排），并认为每一个点都是孤立的，分属于 n 个独立的集合。然后按顺序枚举每一条边。如果这条边连接着两个不同的集合，那么就把这条边加入最小生成树，这两个不同的集合就合并成了一个集合；如果这条边连接的两个点属于同一集合，就跳过。直到选取了 n-1 条边为止。

Kruskal（克鲁斯卡尔）算法开始时，认为每一个点都是孤立的，分属于 n 个独立的集合。

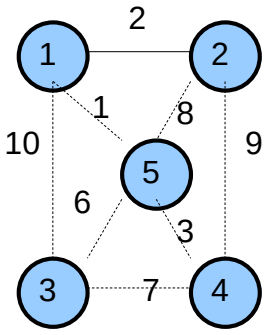


5 个集合{ {1}，{2}，{3}，{4}，{5} }

生成树中没有边

Kruskal 每次都选择一条最小的边，而且这条边的两个顶点分属于两个不同的集合。将选取的这条边加入最小生成树，并且合并集合。

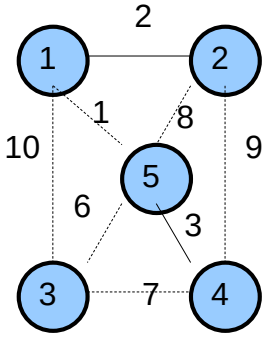
第一次选择的是<1,2>这条边，将这条边加入到生成树中，并且将它的两个顶点 1、2 合并成一个集合。



4 个集合{ {1，2}，{3}，{4}，{5} }

生成树中有一条边{ <1, 2> }

第二次选择的是<4,5>这条边，将这条边加入到生成树中，并且将它的两个顶点 4、5 合并成一个集合。

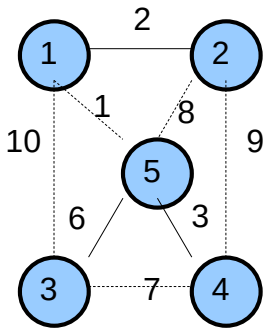


3 个集合{ {1，2}，{3}，{4，5} }

生成树中有 2 条边{ <1, 2> ，<4, 5>}

第三次选择的是<3,5>这条边，将这条边加入到生成树中，并且将它的两个顶点 3、5 所在的两个集合

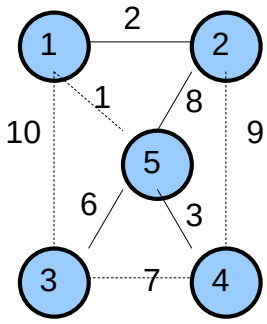
合并成一个集合。



2 个集合{ {1，2}，{3，4，5} }

生成树中有 3 条边{ <1, 2> ，<4, 5>，<3，5>}

第四次选择的是<2,5>这条边，将这条边加入到生成树中，并且将它的两个顶点 2、5 所在的两个集合合并成一个集合。



1 个集合{ {1，2，3，4，5} }

生成树中有 4 条边{ <1, 2> ，<4, 5>，<3，5>，<2, 5>}

算法结束，最小生成树权值为 19。

通过上面的模拟能够看到，Kruskal 算法每次都选择一条最小的，且能合并两个不同集合的边，一张 n 个点的图总共选取 n-1 次边。因为每次我们选的都是最小的边，所以最后的生成树一定是最小生成树。每次我们选的边都能够合并两个集合，最后 n 个点一定会合并成一个集合。通过这样的贪心策略，Kruskal 算法就能得到一棵有 n-1 条边，连接着 n 个点的最小生成树。

Kruskal 算法的时间复杂度为 O(E*logE)，E 为边数。

（2）算法描述:

a) 初始化并查集。father[x]=x。
b) tot=0
c) 将所有边用快排从小到大排序。
d) 计数器 k=0;
e) for (i=1; i<=M; i++)　　//循环所有已从小到大排序的边
if 这是一条 u,v 不属于同一集合的边(u,v)(因为已经排序，所以必为最小)
begin
① 合并 u,v 所在的集合，相当于把边(u,v)加入最小生成树。
② tot=tot+W(u,v)
③ k++
④ 如果 k=n-1,说明最小生成树已经生成，则 break;
end;
f) 结束，tot 即为最小生成树的总权值之和。

典型例题

例 3.6.2 最短网络 Agri-Net　　http://www.luogu.org/problem/show?pid=1546

【问题描述】

农民约翰被选为他们镇的镇长！他其中一个竞选承诺就是在镇上建立起互联网，并连接到所有的农场。当然，他需要你的帮助。约翰已经给他的农场安排了一条高速的网络线路，他想把这条线路共享给其他农场。为了用最小的消费，他想铺设最短的光纤去连接所有的农场。你将得到一份各农场之间连接费用的列表，你必须找出能连接所有农场并所用光纤最短的方案。每两个农场间的距离不会超过 100000。

【输入格式】

第一行：	农场的个数，N（3<=N<=100）。
第二行..结尾	后来的行包含了一个 N*N 的矩阵,表示每个农场之间的距离。理论上，他们是 N 行，每行由 N 个用空格分隔的数组成，实际上，他们限制在 80 个字符，因此，某些行会紧接着另一些行。当然，对角线将会是 0，因为不会有线路从第 i 个农场到它本身。

【输出格式】

只有一个输出，其中包含连接到每个农场的光纤的最小长度。

【输入样例】agrinet.in

```
4
0  4  9  21
4  0  8  17
9  8  0  16
21 17 16  0
```

【输出样例】agrinet.out

```
28
```

分析：这是一道最小生成树问题，可以用 Kruskal 算法求解。

【参考程序】

```
#include<cstdio>
#include<iostream>
#include<algorithm>                    //sort()需要用到<algorithm>库
using namespace std;
struct point
       {
           int x;
           int y;
           int v;
       };                              //定义结构类型，表示边
point a[9901];                         //存边
int fat[101];
int n,i,j,x,m,tot,k;
int father(int x)
{
```

```
    if (fat[x] != x) fat[x] = father(fat[x]);
    return fat[x];
}
void unionn(int x,int y)
{
    int fa = father(x);
    int fb = father(y);
    if (fa != fb) fat[fa] = fb;
}
int cmp(const point &a,const point &b)          //sort()自定义的比较函数
{
     if (a.v < b.v) return 1;
        else return 0;
}
int main()
{
    cin >> n;
    for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++)
        {
            cin >> x;
            if (x != 0)
            {
                m++;
                a[m].x = i; a[m].y = j; a[m].v = x;
              }
        }
    for (i = 1; i <= n; i++) fat[i] = i;
  sort(a+1,a+m+1,cmp);                                  //C++标准库中自带的快排
        //cmp 为自定义的比较函数。表示 a 数组的 1-m 按规则 cmp 排序

for (i = 1; i <= m; i++)
    {
        if (father(a[i].x) != father(a[i].y))
        {
            unionn(a[i].x,a[i].y);
            tot += a[i].v;
            k++;
        }
        if (k == n-1) break;
    }
    cout << tot;
    return 0;
}
```

3.6.5 课后训练

训练 1.局域网　http://www.rqnoj.cn/problem/370

【问题描述】

某个局域网内有 n(n<=100)台计算机，由于搭建局域网时工作人员的疏忽，现在局域网内的连接形成了回路，我们知道如果局域网形成回路那么数据将不停的在回路内传输，造成网络卡的现象。因为连接计算机的网线本身不同,所以有一些连线不是很畅通,我们用 f(i,j)表示 i,j 之间连接的畅通程度(f(i,j)<=1000)，f(i,j)值越小表示 i,j 之间连接越通畅，f(i,j)为 0 表示 i,j 之间无网线连接。现在我们需要解决回路问题，我们将除去一些连线，使得网络中没有回路，并且被除去网线的Σf(i,j)最大，请求出这个最大值。

【输入格式】

第一行两个正整数 n k

接下来的 k 行每行三个正整数 i j m 表示 i,j 两台计算机之间有网线联通，通畅程度为 m。

【输出格式】

一个正整数，Σf(i,j)的最大值

【输入输出样例】

【输入样例】

```
5  5
1  2  8
1  3  1
1  5  3
2  4  5
3  4  2
```

【输出样例】

```
8
```

训练 2.繁忙的都市　http://www.luogu.org/problem/show?pid=2330

【问题描述】

城市 C 是一个非常繁忙的大都市，城市中的道路十分的拥挤，于是市长决定对其中的道路进行改造。城市 C 的道路是这样分布的：城市中有 n 个交叉路口，有些交叉路口之间有道路相连，两个交叉路口之间最多有一条道路相连接。这些道路是双向的，且把所有的交叉路口直接或间接的连接起来了。每条道路都有一个分值，分值越小表示这个道路越繁忙，越需要进行改造。但是市政府的资金有限，市长希望进行改造的道路越少越好，于是他提出下面的要求：

改造的那些道路能够把所有的交叉路口直接或间接的连通起来。

在满足要求 1 的情况下，改造的道路尽量少。

在满足要求 1、2 的情况下，改造的那些道路中分值最大值尽量小。

【编程任务】

作为市规划局的你，应当作出最佳的决策，选择那些道路应当被修建。

【输入格式】city.in

第一行有两个整数 n,m 表示城市有 n 个交叉路口，m 条道路。接下来 m 行是对每条道路的描述，u, v, c 表示交叉路口 u 和 v 之间有道路相连，分值为 c。(1≤n≤300，1≤c≤10000)

【输出格式】city.out

两个整数 s, max，表示你选出了几条道路，分值最大的那条道路的分值是多少。

【输入样例】

```
4 5
```

1 2 3
1 4 5
2 4 7
2 3 6
3 4 8
【输出样例】
3 6

3.7 拓扑排序（胡芳）

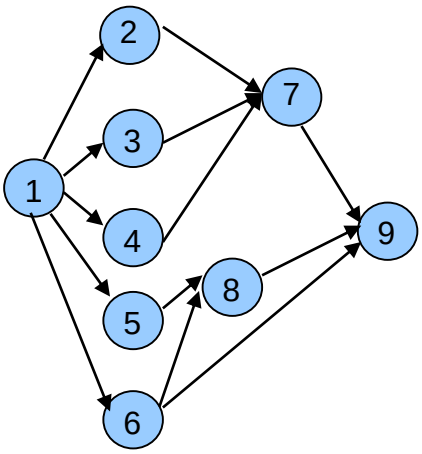
知识点

理解拓扑排序的概念，掌握求拓扑排序的方法。

知识讲解

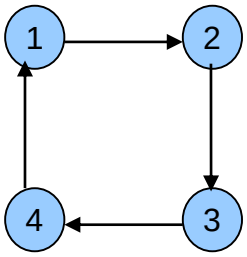
3.7.1 AOV 网

在日常生活中，一项大的工程可以看作是由若干个子工程（这些子工程称为“活动” ）组成的集合，这些子工程（活动）之间必定存在一些先后关系，即某些子工程（活动）必须在其它一些子工程（活动）完成之后才能开始，我们可以用有向图来形象地表示这些子工程（活动）之间的先后关系，子工程（活动）为顶点，子工程（活动）之间的先后关系为有向边，这种有向图称为“顶点活动网络” ，又称“AOV 网” 。



在 AOV 网中，有向边代表子工程（活动）的先后关系，我们把一条有向边起点的活动成为终点活动的前驱活动，同理终点的活动称为起点活动的后继活动。而只有当一个活动全部的前驱全部都完成之后，这个活动才能进行。例如在上图中，只有当工程 1 完成之后，工程 2、3、4、5、6 才能开始进行。只有当 2、3、4 全部完成之后，7 才能开始进行。

一个 AOV 网必定是一个有向无环图，即不应该带有回路。否则，会出现先后关系的自相矛盾。



上图就是一个出现环产生自相矛盾的情况。4 是 1 的前驱，想完成 1，必须先完成 4。3 是 4 的前驱，而 2 是 3 的前驱，1 又是 2 的前驱。最后造成想完成 1，必须先完成 1 本身，这显然出现了矛盾。

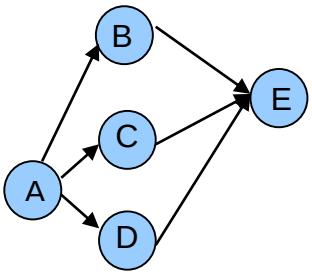
3.7.2 拓扑排序算法

拓扑排序算法，只适用于 AOV 网（有向无环图）。

把 AOV 网中的所有活动排成一个序列， 使得每个活动的所有前驱活动都排在该活动的前面，这个过程称为“拓扑排序”，所得到的活动序列称为“拓扑序列”。

一个 AOV 网的拓扑序列是不唯一的，例如下面的这张图，它的拓扑序列可以是：ABCDE，也可以是 ACBDE，

或是 ADBCE。在下图所示的 AOV 网中，工程 B 和工程 C 显然可以同时进行，先后无所谓；但工程 E 却要等工程 B、C、D 都完成以后才能进行。



构造拓扑序列可以帮助我们合理安排一个工程的进度，由 AOV 网构造拓扑序列具有很高的实际应用价值。

（1）算法分析&思想讲解：

1）选择一个入度为 0 的顶点并输出

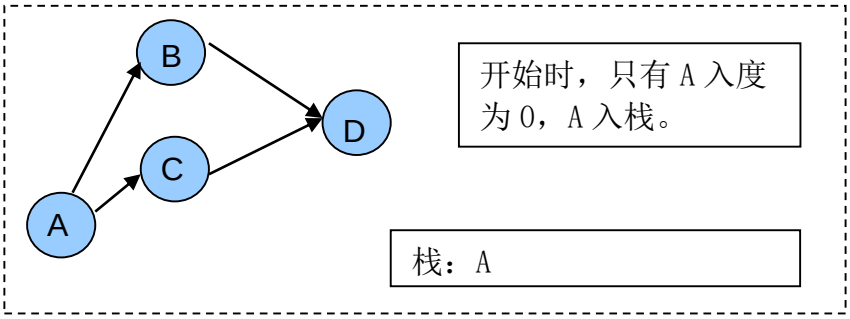
2）然后从 AOV 网中删除此顶点及以此顶点为起点的所有关联边；

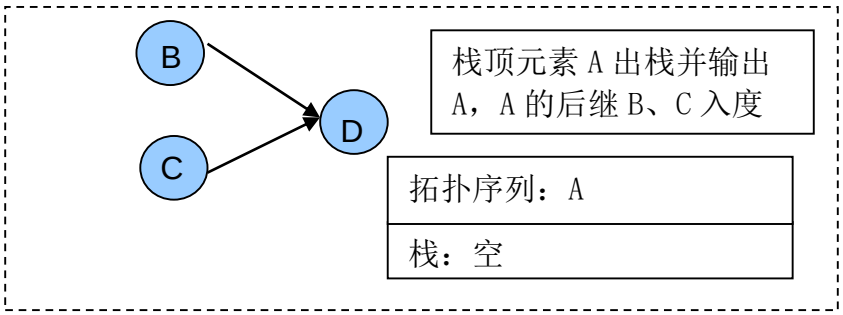
3）重复上述两步，直到不存在入度为 0 的顶点为止。

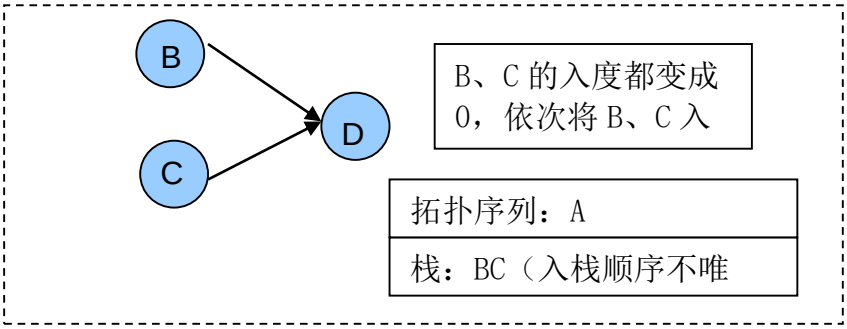
4）若输出的顶点数小于 AOV 网中的顶点数，则输出“有回路信息”，否则输出的顶点序列就是一种拓扑序列

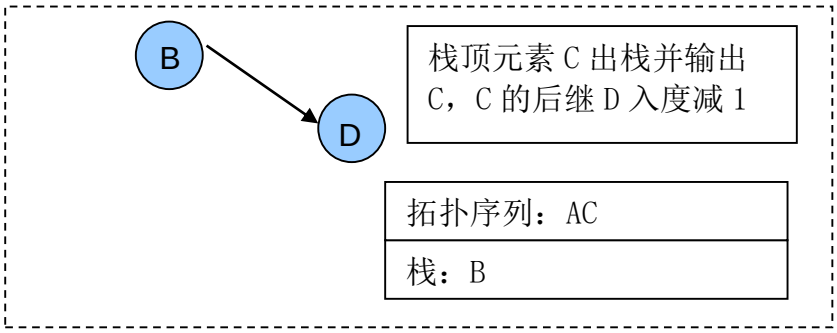
从第四步可以看出，拓扑排序可以用来判断一个有向图是否有环。只有有向无环图才存在拓扑序列。

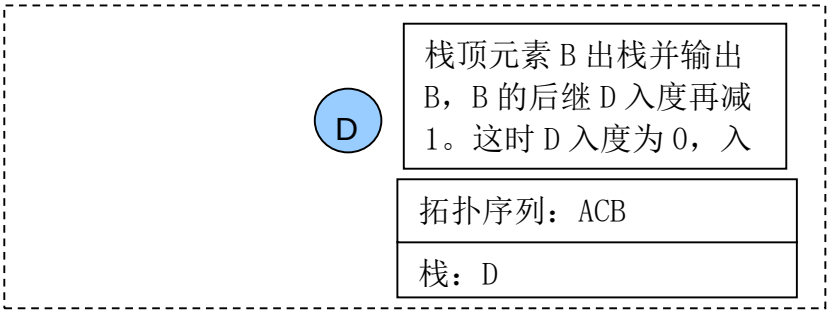
我们结合下图详细讲解：











简单&高效&实用的算法。上述实现方法复杂度 O(V+E)。

（2）算法实现：

a)数据结构：indgr[i]: 顶点 i 的入度； stack[ ]: 栈

b)初始化:top=0 (栈顶指针置零)

c)将初始状态所有入度为 0 的顶点压栈

d)I=0 (计数器)

e)while 栈非空(top>0)

i. 栈顶的顶点 v 出栈；top-1; 输出 v；i++;

ii. for v 的每一个后继顶点 u

1. indgr[u]--; u 的入度减 1

2. if （u 的入度变为 0） 顶点 u 入栈

f)算法结束

这个程序采用栈来找出入度为 0 的点，栈里的顶点，都是入度为 0 的点。

典型例题

例 3.7.1 家谱树

【问题描述】

有个人的家族很大，辈分关系很混乱，请你帮整理一下这种关系。

给出每个人的孩子的信息。

输出一个序列，使得每个人的后辈都比那个人后列出。

【输入格式】

第 1 行一个整数 N（1<=N<=100），表示家族的人数。

接下来 N 行，第 I 行描述第 I 个人的儿子。

每行最后是 0 表示描述完毕。

【输出格式】

输出一个序列，使得每个人的后辈都比那个人后列出。

如果有多解输出任意一解。

【输入样例】

```
5
0
4 5 1 0
1 0
5 3 0
3 0
```

【输出样例】

```
2 4 5 3 1
```

【参考程序】

```
#include<cstdio>
#include<iostream>
using namespace std;
int a[101][101],c[101],r[101],ans[101];
int i,j,tot,temp,num,n,m;
int main()
{
    cin >> n;
    for (i = 1; i <= n; i++)
    {
        do
        {
            cin >> j;
            if (j !=0 )
            {
                c[i]++;                    //c[i]用来存点 i 的出度
                a[i][c[i]] = j;
                r[j]++;                    //a[i,-1]用来存点 i 的入度。
            }
        }
        while (j != 0);
    }
    for (i = 1; i <= n; i++)
        if (r[i] == 0)
            ans[++tot] = i;            //把图中所有入度为 0 的点入栈，栈用一维数组 ans[]表示
    do
    {
        temp = ans[tot];
        cout << temp << " ";
        tot--;num++;                   //栈顶元素出栈并输出
        for (i = 1; i <= c[temp]; i++)
        {
```

```
                r[a[temp][i]]--;
                if (r[a[temp][i]] == 0) //如果入度减 1 后变成 0，则将这个后继点入栈
                    ans[++tot] = a[temp][i];
            }
        }
        while (num != n);                    //如果输出的点的数目 num 等于 n，说明算法结束
        return 0;
    }
```

例 3.7.2 奖金

【问题描述】

由于无敌的凡凡在 2005 年世界英俊帅气男总决选中胜出，Yali Company 总经理 Mr.Z 心情好，决定给每位员工发奖金。公司决定以每个人本年在公司的贡献为标准来计算他们得到奖金的多少。于是 Mr.Z 下令召开 m 方会谈。每位参加会谈的代表提出了自己的意见：“我认为员工 a 的奖金应该比 b 高！”Mr.Z 决定要找出一种奖金方案，满足各位代表的意见，且同时使得总奖金数最少。每位员工奖金最少为 100 元。

【输入格式】

第一行两个整数 n,m，表示员工总数和代表数；以下 m 行，每行 2 个整数 a,b，表示某个代表认为第 a 号员工奖金应该比第 b 号员工高。

【输出格式】

若无法找到合理方案，则输出“Poor Xed”；否则输出一个数表示最少总奖金。

【输入样例】

2 1
1 2

【输出样例】

201

【数据规模】

80％的数据满足 n<=1000，m<=2000；100％的数据满足 n<=10000，m<=20000。

【算法分析】

首先构图，若存在条件“a 的钱比 b 多”则从 b 引一条有向指向 a；然后拓扑排序，若无法完成排序则表示问题无解（存在圈）；若可以得到完整的拓扑序列，则按序列顺序进行递推：

设 f[i]表示第 i 个人能拿的最少奖金数；

首先所有 f[i]=100（题目中给定的最小值）；

然后按照拓扑顺序考察每个点 i，若存在有向边(j,i)，则表示 f[i]必须比 f[j]大，因此我们令 f[i] = Max { f[i] , f[j]+1 }即可；

递推完成之后所有 f[i]的值也就确定了，而答案就等于 f[1]+…+f[n]。

【参考程序】

```
#include<iostream>
using namespace std;
int a[10001][301]={0},into[10001],ans[10001],m,n,money;
void init()                                  //读入数据，并构建图，统计入度
{ int i,x,y;
   cin>>n>>m;
   for(i=1;i<=m;i++)
    {
      cin>>x>>y;
```

```
        a[y][0]++;                          //记录由 y 引出边的数目
        a[y][a[y][0]]=x;                    //记录由 a[y][0]引出边的顶点
        into[x]++;                          //统计入度
      }
}
bool topsort()                              //拓扑排序
{ int t,tot,k,i,j;
   tot=0;k=0;
   while(tot<n)                             //tot 顶点个数
    {
     t=0;                                   //用来判断有无环
     for(i=1;i<=n;i++)
        if(into[i]==0)
        {
          tot++;t++;money+=100;
          ans[t]=i;
          into[i]=0xffffffff;
         }
    if(t==0)return false;                   //存在环
    money+=k*t;k++;
    for(i=1;i<=t;i++)                       //去掉相连的边
       for(j=1;j<=a[ans[i]][0];j++)into[a[ans[i]][j]]--;
   }
   return true;
}
int main()
{
   init();money=0;
   if(topsort())cout<<money<<endl;
       else cout<<"Poor Xed"<<endl;
   return 0;
}
```

3.7.3 课后训练

训练 1.烦人的幻灯片 http://acm.sdibt.edu.cn/JudgeOnline/problem.php?id=1244

【问题描述】

李教授将于今天下午作一次非常重要的演讲。不信的事他不是一个非常爱整洁的人，他把自己演讲要用的幻灯片随便堆在了一起。因此，演讲之前他不得不去整理这些幻灯片。作为一个讲求效率的学者，他希望尽可能简单地完成它。教授这次演讲一共要用 n 张幻灯片（n<=26），这 n 张幻灯片按照演讲要使用的顺序已经用数字 1~n 编了号。因为幻灯片是透明的，所以我们不能一下子看清每一个数字所对应的幻灯片。

现在我们用大写字母 A,B,C……再次把幻灯片依次编号。你的任务是编写一个程序，把幻灯片的数字

编号和字母编号对应起来，显然这种对应应该是唯一的；若出现多种对应的情况或是某些数字编号和字母编号对应不起来，我们称对应是无法实现的。

【输入格式】

文件的第一行只有一个整数 n，表示有 n 张幻灯片，接下来的 n 行每行包括 4 个整数 xmin,xmax,ymin,ymax（整数之间用空格分开）为幻灯片的坐标，这 n 张幻灯片按其在文件中出现的顺序从前到后依次编号为 A,B,C……，再接下来的 n 行依次为 n 个数字编号的坐标 x,y，显然在幻灯片之外是不会有数字的。

【输出格式】

若是对应可以实现，输出文件应该包括 n 行，每一行为一个字母和一个数字，中间以一个空格隔开，并且每行以字母的升序排列，注意输出的字母要大写并且定格；反之，若是对应无法实现，在文件的第一行顶格输出 None 即可。首行末无多余的空格。

【输入样例】

4
6 22 10 20
4 18 6 16
8 20 2 18
10 24 4 8
9 15
19 17
11 7
21 11

【输出样例】

A 4
B 1
C 2
D 3

训练 2 病毒

【问题描述】

有一天，小 y 突然发现自己的计算机感染了一种病毒！还好，小 y 发现这种病毒很弱，只是会把文档中的所有字母替换成其它字母，但并不改变顺序，也不会增加和删除字母。

现在怎么恢复原来的文档呢！小 y 很聪明，他在其他没有感染病毒的机器上，生成了一个由若干单词构成的字典，字典中的单词是按照字母顺序排列的，他把这个文件拷贝到自己的机器里，故意让它感染上病毒，他想利用这个字典文件原来的有序性，找到病毒替换字母的规律，再用来恢复其它文档。

现在你的任务是：告诉你被病毒感染了的字典，要你恢复一个字母串。

【输入格式】virus.in

第一行为整数 K（≤50000），表示字典中的单词个数。

以下 K 行，是被病毒感染了的字典，每行一个单词。

最后一行是需要你恢复的一串字母。

所有字母均为小写。

【输出格式】virus.out

输出仅一行，为恢复后的一串字母。当然也有可能出现字典不完整、甚至字典是错的情况，这时请输出一个 0。

【输入样例】

6
cebdbac

cac
ecd
dca
aba
bac
cedab

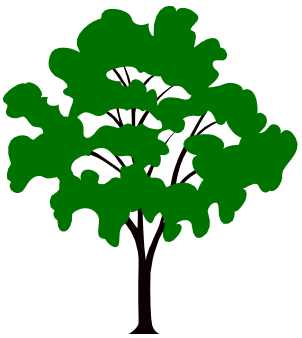
【输出样例】

abcde

3.8. 树的应用（李云军）

知识点

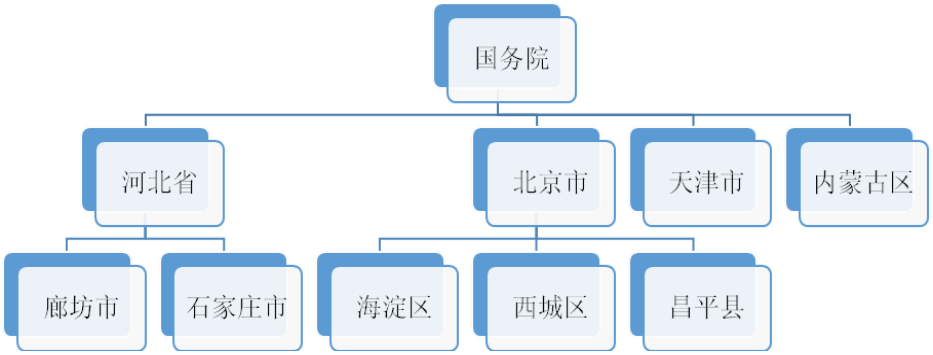
树与森林、二叉树的定义、性质、操作和相关算法的实现以及应用。



3.8.1 知识讲解

现实生活中有些数据存在一对多的层次关系，如国家及各种组织的管理结构、人类社会的族谱、硬盘的存储结构等，为了表示出这种关系需要用到我们今天学习的树形数据结构，它和图一样属于的非线性数据结构。

国家行政管理机构的一部分：



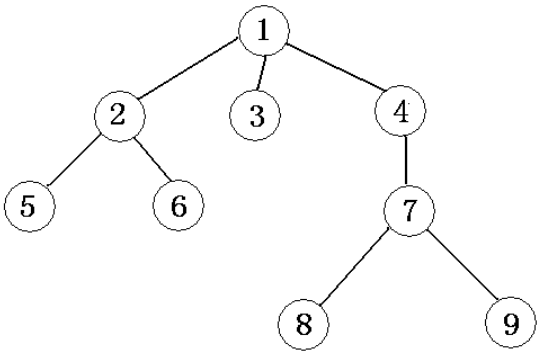
1、树的定义

树是由 n (n >=0)个结点组成的有限集合。

如果 n = 0，称为空树；

如果 n > 0，则:

有一个特定的结点称之为根(root)的结点，它只可以有后继，但没有前驱；　除根以外的其它结点划分为 m (m > =0)个互不相交的有限集合 T0, T1, …, Tm-1，每个集合本身又是一棵树，并且称之为根的子树(subTree)。每棵子树的根结点有且仅有一个直接前驱，但可以有 0 个或多个后继。



2、树相关的术语

结点的度（Degree）：结点的子树个数；

树的度：树的所有结点中最大的度数；

叶结点（Leaf）：度为 0 的结点；

父结点（Parent）：有子树的结点是其子树的根节点的父结点；

子结点/孩子结点（Child）：若 A 结点是 B 结点的父结点，则称 B 结点是 A 结点的子结点；

兄弟结点（Sibling）：具有同一个父结点的各结点彼此是兄弟结点；

路径和路径长度：从结点 n1 到 nk 的路径为一个结点序列 n1，n2，...，nk。ni 是 ni+1 的父结点。路径所包含边的个数为路径的长度；

祖先结点（Ancestor）：沿树根到某一结点路径上的所有结点都是这个结点的祖先结点；

子孙结点（Descendant）：某一结点的子树中的所有结点是这个结点的子孙；

结点的层次（Level）：规定根结点在 1 层，其他任一结点的层数是其父结点的层数加 1；

树的深度（Depth）：树中所有结点中的最大层次是这棵树的深度；

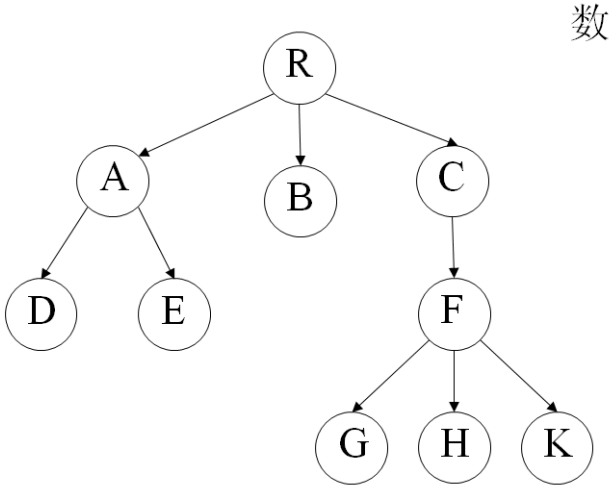
有序树和无序树：若结点的子树有次序排列，且先后次序不能互换，这样的树称为有序树，反之为无序树。

3、树的存储结构

(1)双亲表示法（顺序存储）

```
#define    MAX_TREE_SIZE    100
typedefstructPTNode {
TElemType      data;
int      parent;              //双亲位置域
}PTNode;
typedefstruct{
PTNodenodes[MAX_TREE_SIZE];
int      n;                   //结点数
}PTree;
```

双亲表示法举例



数组下标:

0	R	-1
1	A	0
2	B	0
3	C	0
4	D	1
5	E	1
6	F	3
7	G	6
8	H	6
9	K	6

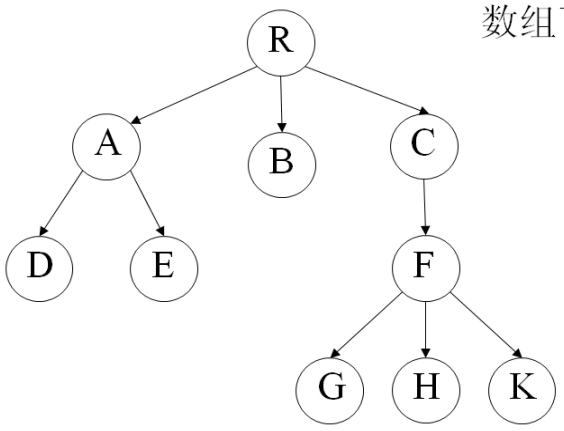
* 便于涉及双亲的操作；
* 求结点的孩子时需要遍历整棵树。

(2)孩子表示法（顺序存储）

```
#define    MAX_TREE_SIZE    100
typedefstructPTNode {
TElemType      data;
int      child[m];              //存储每个孩子的位置

}PTNode;
typedefstruct{
PTNodenodes[MAX_TREE_SIZE];
int      n;                     //结点数
}PTree;
```

孩子表示法举例



数组下标:

0	R	1	2	3
1	A	4	5	0
2	B	0	0	0
3	C	6	0	0
4	D	0	0	0
5	E	0	0	0
6	F	7	8	9
7	G	0	0	0
8	H	0	0	0
9	K	0	0	0

* 便于涉及孩子的操作；求双亲不方便；
* 采用同构的结点，空间浪费。

(3)孩子兄弟表示法

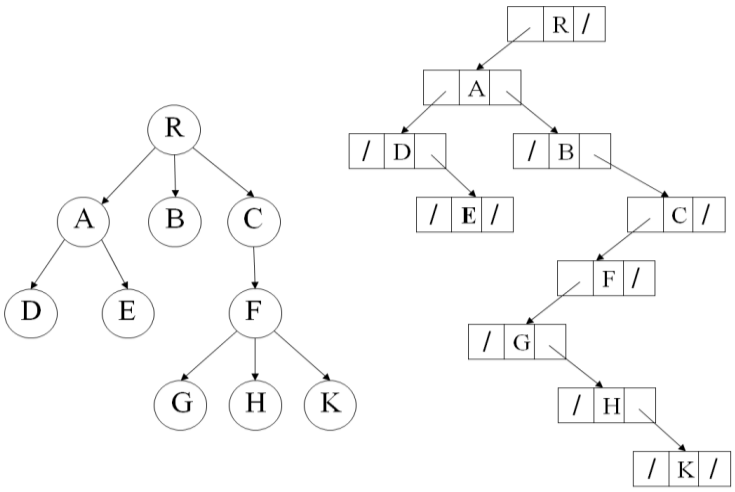
树的二叉链表(孩子兄弟)存储表示------

```
typedefstructCSNode {
ELemType      data;
structCSNode      *firstchild,*nextsibling;
 }CSNode,    *CSTree;
```

顺序存储

```
typedefstruct {
ELemType      data;
intfirstchild,nextsibling;
 }CSNode;
CSNnode t[MAX_TREE_SIZE];
```

孩子兄弟表示法示例：



0	R	1	0
1	A	4	2
2	B	0	3
3	C	6	0
4	D	0	5
5	E	0	0
6	F	7	0
7	G	0	8
8	H	0	9
9	K	0	0

4、二叉树的定义与相关概念

二叉树(Binary Tree)是另一种重要的树型结构。是度为 2 的有序树，它的特点是每个结点至多有两棵子树。和树结构的定义类似，二叉树的定义也可以用递归形式给出。

二叉树的递归定义

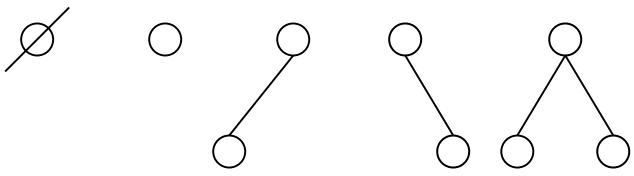
二叉树(BinaryTree)是 n(n≥0)个结点的有限集。它或者是空集(n=0)，或者同时满足以下两个条件：

(1) 有且仅有一个根结点；

(2) 其余的结点分成两棵互不相交的左子树和右子树。

二叉树与树有区别：树至少应有一个结点，而二叉树可以为空；树的子树没有顺序，但如果二叉树的根结点只有一棵子树，必须明确区分它是左子树还是右子树，因为两者将构成不同形态的二叉树。因此，二叉树不是树的特例。它们是两种不同的数据结构。

二叉树有 5 种基本形态：



a　　b　　c　　d　　e

(a) 空二叉树　(b) 只有根结点的二叉树 (c) 右子树为空的二叉树

(d) 左子树为空的二叉树(e) 左右子树均不为空的二叉树

两种特殊形态的二叉树：满二叉树和完全二叉树。

(1) 满二叉树(FullBinaryTree)

深度为 k，且有 2k-1 个结点的二叉树。

特点：1）每一层上结点数都达到最大

2）度为 1 的结点 n1=0，树叶都在最下一层上。

(2) 完全二叉树(Complete BinaryTree)

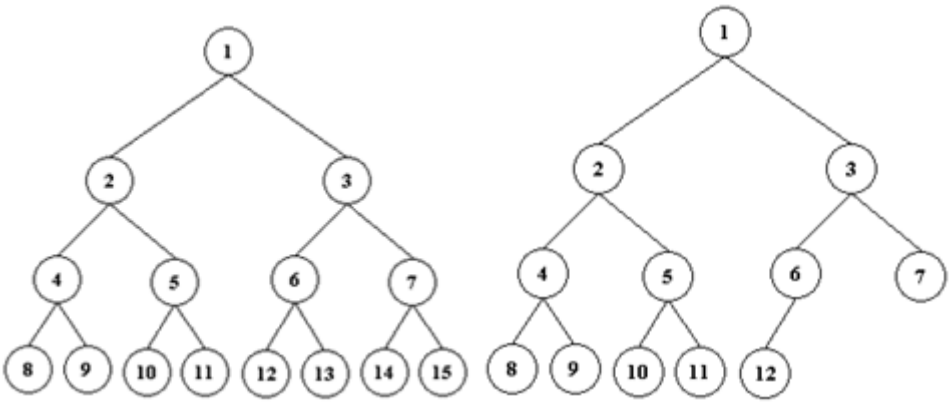
如果一棵深度为 K 的二叉树，1 至 k-1 层的结点都是满的，即满足 2i-1，只有最下面的一层的结点数小于 2i-1，并且最下面一层的结点都集中在该层最左边的若干位置，时，称为完全二叉树。

完全二叉树的特点：

1）每个结点 i 的左子树的深度 Lhi-其结点 i 的右子树的深度 Rhi 等于 0 或 1，即叶结点只可能出现在层次最大或次最大的两层上。

2）完全二叉树结点数 n 满足 2k-1-1＜n≤2k-1

3）满二叉树一定是完全二叉树，反之不成立。



满二叉树完全二叉树

5、二叉树的性质

性质 1　在二叉树的第 i 层上至多有 2i-1 个结点(i≥1)。

性质 2　深度为 k 的二叉树至多有 2k-1 个结点(k≥1)。

性质 3　二叉树中,终端结点数 n0 与度为 2 的结点数 n2 有如下关系:　　n0=n2+1

性质 4　结点数为 n 的完全二叉树，其深度为⌊log2n⌋+ l

性质 5　在按层序编号的 n 个结点的完全二叉树中，任意一结点 i(1≤i≤n)有：

⑴i=1 时，结点 i 是树的根；否则，结点 i 的双亲为结点⌊ i/2 ⌋(i>1)。

⑵2i＞n 时，结点 i 无左孩子，为叶结点；否则，结点 i 的左孩子为结点 2i。

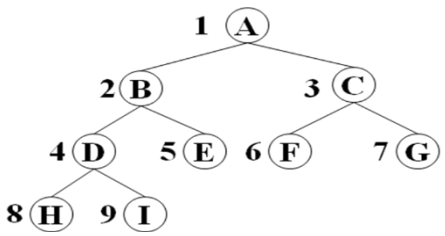
⑶2i+1＞n 时，结点 i 无右孩子；否则，结点 i 的右孩子为结点 2i+1。

6、二叉树的存储结构

同线性表一样，二叉树的存储结构也有顺序和链表两种结构。

（1）顺序存储结构

对于满二叉树和完全二叉树，可以对每个结点进行连续编号，所以通常采用顺序存储的方式。具体做法是：定义一个一维数组，把每个结点的编号作为数组的下标变量，将结点的值存放在数组中。二叉树的各结点的编号从根结点开始，根结点的标号为 1，然后，逐层由左向右进行连续编号，并将该结点的值存于对应编号为下标的一维数组中。

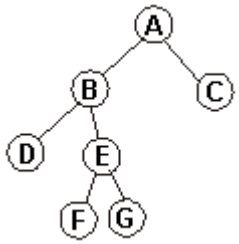


1	2	3	4	5	6	7	8	9
A	B	C	D	E	F	G	H	I

bt [3]的双亲为⌊3/2⌋=1，即在 bt[1]中；其左孩子在 bt[2i]=bt[6]中；其右孩子在 bt[2i+1]=bt[7]中。

一般二叉树也按完全二叉树形式存储，无结点处用 0 表示。

这种存储结构适合于完全二叉树，既不浪费存储空间，又能很快确定结点的存放位置、结点的双亲和左右孩子的存放位置，但对一般非完全二叉树，可能造成存储空间的大量浪费。



A	B	C	D	E	0	0	0	0	F	G	0	0	0	0

例如：深度为 k，且只有 k 个结点的右单枝树(•每个非叶结点只有右孩子)，需 2k-1 个单元，即有 2k-1-k 个单元被浪费。

（2）链式存储结构（二叉链表）

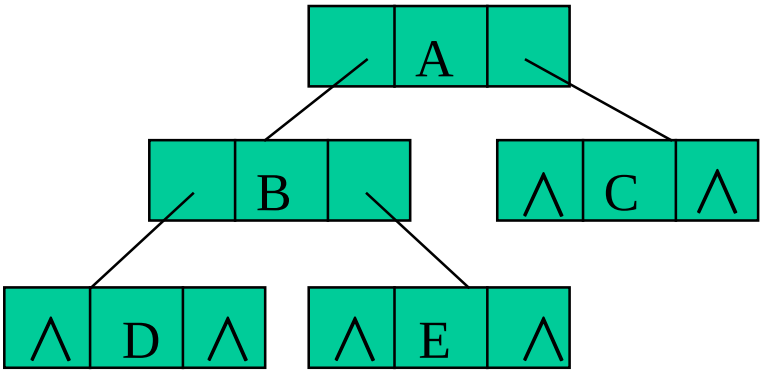
由于二叉树每个结点至多只有 2 个孩子，分别为左孩子和右孩子。因此可以把每个结点分成三个域：一个域存放结点本身的信息，另外两个是指针域，分别存放左、右孩子的地址。每个结点的结构表示为：

其中左链域 lchild 为指向左孩子的指针，右链域 rchild 为指向右孩子的指针，数据域 data 表示结点的值。若某结点没有左孩子或右孩子，其相应的链域为空指针。

对应的结构类型定义如下：

```
typedefstruct node
{ ElemType data;
struct node *lchild;
struct node *rchild;
} BTree,*tree;
```

其中，tree 是指向根结点的指针。



二叉链表

用数组实现的链式存储

datalchildrchild

typedefstruct

{　ElemType data;

intlchild;

intrchild;

} node;

node

mytree[MAX_TREE_SIZE]；

int bt

其中，bt 存储根节点位置。

也可以定义多个数组存储：

int f[maxn],l[maxn],r[maxn];

0		
1　A	2	3
2　B	4	5
3　C	0	0
4　D	0	0
5　E	0	0

说明：

●一个二叉链表由根指针 root 唯一确定。若二叉树为空，则 root=NULL；若结点的某个孩子不存在，则相应的指针为空。

●具有 n 个结点的二叉链表中，共有 2n 个指针域。其中只有 n-1 个用来指示结点的左、右孩子，其余的 n+1 个指针域为空。

●若经常要在二叉树中寻找某结点的双亲时，可在每个结点上再加一个指向其双亲的指针 parent，形成一个带双亲指针的二叉链表。就是三叉链表。

5、遍历二叉树

在二叉树的一些应用中，常常要求在树中查找具有某种特征的结点，或者对树中全部结点逐一进行某种处理。这就引入了遍历二叉树的问题，即如何按某条搜索路径访问树中的每一个结点，使得每一个结点仅切仅被访问一次。

遍历的次序：假如以 L、D、R 分别表示遍历左子树、遍历根结点和遍历右子树，遍历整个二叉树则有 DLR、LDR、LRD、DRL、RDL、RLD 六种遍历方案。若规定先左后右，则只有前三种情况，分别规定为：LDR 中序遍历；LRD 后序遍历；DLR 先序遍历

遍历方案的实现

1）中序遍历二叉树

算法思想:

若二叉树非空，则：1）中序遍历左子树；2）访问根结点；3）中序遍历右子树。

算法描述:

```
void Inorder (BiTreebt) //bt 为根结点指针
{if( bt)                    //根非空
{ Inorder (bt->lchild) ;
        visit (bt->data);
Inorder (bt->rchild) ;
}
}

voidinorder(int i)
{                   //数组存储中序遍历
    if(i)
    {
        inorder(l[i]);
```

```
            printf("%d ",v[i]);
            inorder(r[i]);
        }
}
```

2）后序遍历二叉树

算法思想:

若二叉树非空，则：1）后序遍历左子树；2）后序遍历右子树；3）访问根结点

算法描述:

```
void Postorder (BiTreebt) //bt 为根结点指针
{if( bt)//根非空
{ Postorder (bt->lchild) ;
Postorder (bt->rchild) ;
          visit (bt ->data);
}
}
voidsucorder(int i)
{                    //数组存储后序遍历
    if(i)
    {
          sucorder(l[i]);
          sucorder(r[i]);
          printf("%d ",v[i]);
     }
}
```

3）先序遍历二叉树

算法思想:

若二叉树非空，则：1）访问根结点；2）先序遍历左子树；3）先序遍历右子树

算法描述:

```
void Preorder (BiTreebt) //bt 为根结点指针
{if( bt)//根非空
{   visit (bt->data);
        Preorder (bt->lchild) ;
        Preorder (bt->rchild) ;
    }
}
void preorder(int i)
{    //数组存储先序遍历
if(i)
{    printf("%d ",v[i]);
          preorder(l[i]);
          preorder(r[i]);
    }
}
```

从二叉树的遍历定义可知，三种遍历算法的不同之处仅在于访问根结点和遍历左右子树的先后关系。如果在算法中隐去和递归无关的语句 printf()，则三种遍历算法是完全相同的。遍历二叉树的算法中的基本操作是访问结点，显然，不论按那种方式进行遍历，对含 n 个结点的二叉树，其时间复杂度均为 O(n)。

7、树、森林与二叉树的转换

（ 1 ）树与二叉树的对应关系

树与二叉树均可用二叉链表作为存储结构，因此给定一棵树，用二叉链表存储，可唯一对应一棵二叉树，反之亦然。

（2）树转换成二叉树

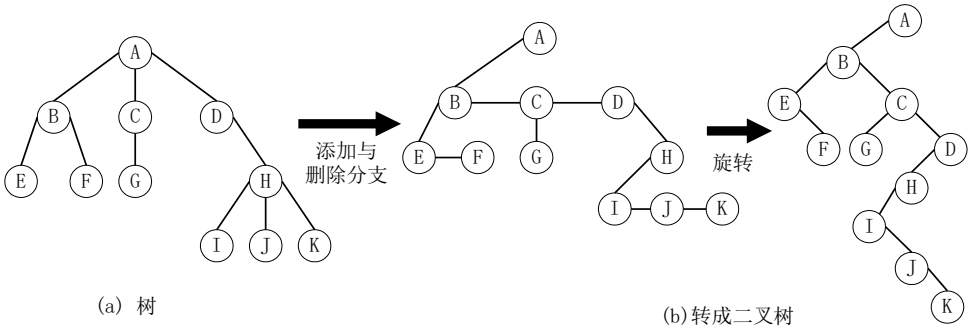
将一棵树转化为等价的二叉树方法如下：

1) 在树中各兄弟（堂兄弟除外）之间加一根连线。

2) 对于任一结点，只保留它与最左孩子的连线外，删去它与其余孩子之间的连线。

3) 以树根为轴心，将整棵树按顺时钟方向旋转约 45°。

特点：根无右子树



树转二叉树示意图

（3）森林转换成二叉树

森林是若干棵互不相交的树构成的集合。

森林也可转换成二叉树，但树转换成二叉树后根结点无右分支，而森林转换后的二叉树，其根结点有右分支。

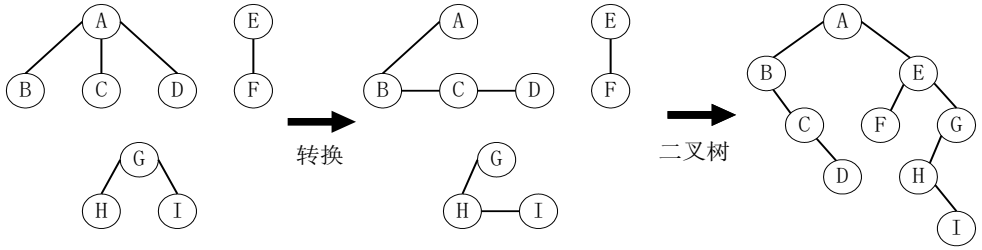
将森林转化为二叉树方法如下：

1) 将森林中的每一棵树转换成等价的二叉树。

2) 保留第一棵二叉树，自第二棵二叉树始，依次将后一棵二叉树的根结点作为前一棵二叉树根结点的右孩子，当所有的二叉树依此相连后，所得到的二叉树就是由森林转化成的二叉树。

3) 以树根为轴心，将整棵树按顺时钟方向旋转约 45°。

转换过程如图



8、树的遍历

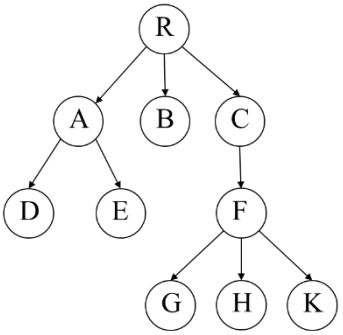
树的两种遍历方法：先根遍历、后根遍历：

先根遍历：

（1）访问树的根结点；

（2）依次先根遍历每棵子树。

右图遍历结果 R A D E B C F G H K



后根遍历：

（1）依次后根遍历每棵子树。

（2）访问树的根结点；

右图遍历结果 D E A B G H K F C R

9、二叉树的应用

树型结构具有广泛的应用领域，常见的有：二叉排序树、哈夫曼树、堆、线段树、后缀树等。

（1）二叉排序树

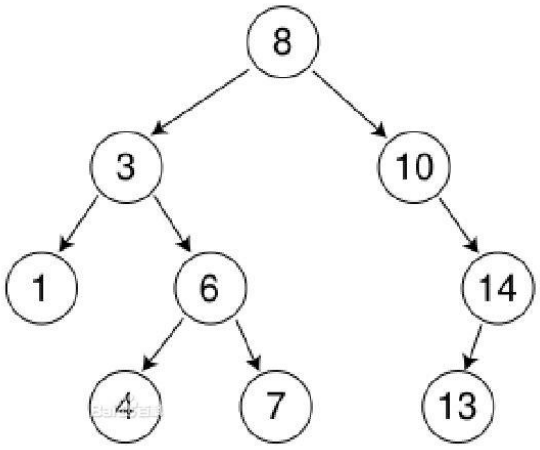
二叉排序树（Binary Sort Tree）或者是一棵空树；或者是具有下列性质的二叉树：

1）若根结点的左子树非空，则左子树中所有结点的值均小于根结点值；

2）若根结点的右子树非空，则右子树中所有结点的值均大于根结点值；

3）它的左右子树也分别为二叉排序树。

二叉排序树又称二叉查找树，亦称二叉搜索树，是一种动态树表。它支持高效的查找和插入、删除操作。



（2）路径长度和最优二叉树（哈夫曼树）

哈夫曼（Huffman）树又称最优二叉树或最优搜索树，是一种带权路径长度最短的二叉树。

在许多应用中，常常赋给树中结点一个有某种意义的实数，称此实数为该结点的权。设树中有 m 个叶结点，每个叶结点带一个权值 w i 且根到叶结点 i 的路径长度为 Li (i =1，2，. . m），则树的带权路径长度为树中所有叶结点的权值与路径长度的乘积的总和。

$$WPL=\sum_{K=1}^{M} W_k L_k$$

即：W P L ＝∑ W k L k

K ＝ 1

例如，给定 4 个叶结点，设权值分别为 1，3，5，7，据此可以构造出形状不同的 4 棵二叉树，如图 6-

19 所示。它们的带权路径长度分别为：

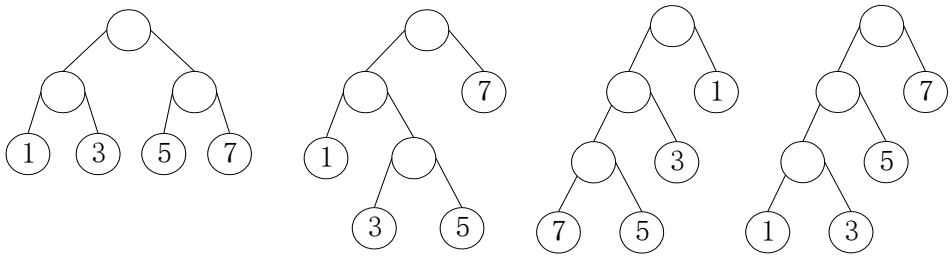
(a) WPL=1×2+3×2+5×2+7×2=32

(b) WPL=1×2+3×3+5×3+7×l=33

(c) WPL=7×3+5×3+3×2+1×1=43

(d) WPL=1×3+3×3+5×2+7×1=29

WPL 最小的二叉树是最优二叉树(Huffman 树)，图 4-19(d)所示。



图示由 4 个结点构成的不同的带权二叉树

哈夫曼树（或最优二叉树）的几个特性

①当叶子上的权值均相同时，完全二叉树一定是最优二叉树。否则完全二叉树不一定是最优二叉树。

②在最优二叉树中，权值越大的叶子离根越近。

③最优二叉树的形态不唯一，但 WPL 最小。

用哈夫曼算法构造最优二叉树：

哈夫曼算法的基本思想是：

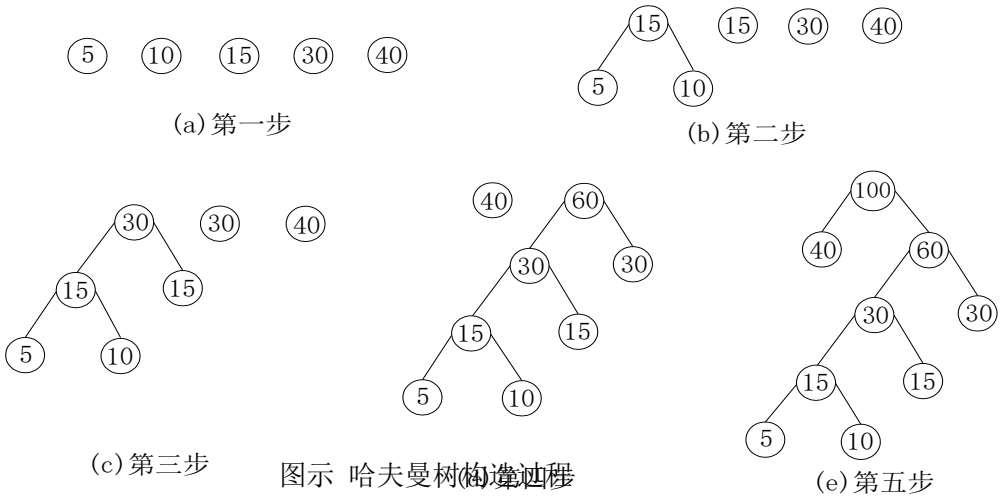
1)以权值分别为 W1,W2．．．Wn 的 n 个结点，构成 n 棵二叉树 T1,T2,．．．Tn 并组成森林 F={T1,T2,．．．Tn},其中每棵二叉树 Ti 仅有一个权值为 Wi 的根结点；

2) 在 F 中选取两棵根结点权值最小的树作为左右子树构造一棵新二叉树，并且置新二叉树根结点权值为左右子树上根结点的权值之和（根结点的权值=左右孩子权值之和，叶结点的权值= Wi）

3) 从 F 中删除这两棵二叉树，同时将新二叉树加入到 F 中;

4) 重复(2)．(３)直到 F 中只含一棵二叉树为止，这棵二叉树就是 Huffman 树。

例如，给定权值集合{5，15，40，30，10}构造哈夫曼树的过程如图 6-21 所示，其中最优的带权路径长度为：WPL＝(5＋10)×4＋15×3＋30×2＋40＝205。由图 4-21 可以看出，哈夫曼树的结点的度数为 0 或 2，没有度为 1 的结点。



图示 哈夫曼树的构造过程

哈夫曼树的存储结构及哈夫曼算法的实现

1) 哈夫曼树的存储结构

用大小为 2n-1 的一维数组来存储哈夫曼树中的结点，其存储结构为：

```
#define n 100                    //叶结点数目
#define m 2*n-1                  //树中结点总数
typedefstruct
{ float weight;                  //权值，设权值均大于零
intlchild,rchild,parent;   //左右孩子及双亲指针
} HTNode;
typedefHTNodeHuffmanTree[m]; //哈夫曼树是一维数组
```

因为 C 语言数组的下界为 0，用-1 表示空指针。树中结点的 lchild、rchild 和 parent 不等于-1 时，分别表示该结点的左、右孩子和双亲结点在数组中的下标。

设置 parent 域有两个作用：一是使查找某结点的双亲变得简单；二是可通过判定 parent 的值是否为-1 来区分根与非根结点。

2）哈夫曼算法的实现

```
voidCreateHuffmanTree(HuffmanTree T)
{ int i,p1,p2;                          //构造哈夫曼树，T[m-1]为其根结点
InitHuffmanTree(T); //T 初始化：将 T[0…m-1]中 2n-1 个结点里的三个指针均置为空(即置为-1)，权值置为 0。
InputWeight(T);        //输入叶子权值至 T[0．．n-1]的 weight 域
for(i=n;i<m;i++)
{   SelectMin(T,i-1,&p1,&p2);//共进行 n-1 次合并，新结点依次存于 T[i]中
                          //在 T[0…i-1]中选择两个权最小的根结点，其序号分别为 p1 和 p2
  T[p1].parent=T[p2].parent=i;
  T[i].1child=p1;                  //最小权的根结点是新结点的左孩子
  T[i].rchild=p2;                  //次小权的根结点是新结点的右孩子
  T[i].weight=T[p1].weight+T[p2].weight;
 }
}
```

3）哈夫曼编码

哈夫曼树的应用很广，哈夫曼编码就是哈夫曼树在电讯通信中的应用之一。

在电报通信中，电文是以二进制的 0，1 序列传送的。在发送端需要将电文中的字符转换成 0，1 序列（编码）发送，在接收端又需要把接收到的 0，1 序列还原成相应的字符序列（译码）。

最简单的二进制编码方式是等长编码。假定需传送的电文是 CDABB，在电文中仅使用 A，B，C，D 4 种字符，则只需用两个字符串便可分辨。可依次对其编码为：00，01，10，11。上述需发送的的电文是“1011000101”。译码员可按两位一组进行译码，恢复原来的电文。

例如：需将文字“ABACCDA”转换成电文。文之中有四种字符，用 2 位二进制便可分辨。

编码方案1：

A	B	C	D
00	01	10	11

则上述文字的电文为：00010010101100　　共 14 位。译码时，只需每 2 位一译即可。特点：等长等频率编码，译码容易，但电文不一定最短。

编码方案2：

A	B C	D
0	00	1	01

采用不等长编码，让出现次数多的字符用短码。则上述文字的电文为：000011010　　共 9 位。但无法译码，它既可译为 BBCCACA，也可译为 AAAACCDA 等。

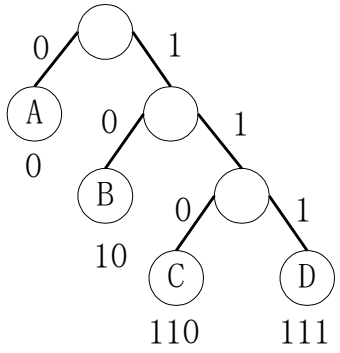
编码方案3：

A	B	C	D
0	110	10	111

采用不等长编码，让出现次数多的字符用短码，且任一编码不能是另一编码的前缀。则上述文字的电文为：0110010101110　　共 13 位。

设有 n 种字符，每种字符出现的次数为 Wi ,其编码长度为 Li (i=1,2,．．．n)，则整个电文总长度为Σ Wi Li ，要得到最短的电文，即使得Σ Wi Li 最小。也就是以字符出现的次数为权值，构造一棵 Huffman 树，并规定左分支编码位 0，右分支编码为 1，则字符的编码就是从根到该字符所在的叶结点的路径上的分支编号序列。用构造 Huffman 树编出来的码，称为 Huffman 编码。

为了获得传送电文的最短长度，可将字符出现的次数（频率）作为权值赋予该结点，构造一棵 WPL 最小的哈夫曼树，由此得到的二进制前缀编码就是最优前缀编码，也称哈夫曼编码。可以验证，用这样的编码传送电文可使总长最短。

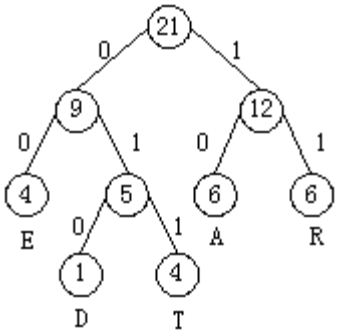


上图为哈夫曼编码图

例如，设一文本的字符序列是：DATA TRERTER ARE AREA ART，此文本的字符集为{A，D，T，R，E}，各字符出现的次数为{6，1，4，6，4}。以此为权值，构造一棵最优二叉树（哈夫曼树），如下图 6-23 所示。

约定从各非终端结点发出的左分支表示 0，右分支表示 1。于是，由根结点到叶结点的路径上所有 0 和 1 组成的序列，就是该叶结点所表字符的哈夫曼编码。如下所示：

字符	A	D	T	R	E
编码	10	010	011	11	00



由此可见，根据权值构造哈夫曼树得出的哈夫曼编码，使字符出现次数（频率）与码长呈反比关系，如此得到的电文总码长最短；同时，又避免了每一个字符编码是另一个字符编码的前缀，保证了译码的唯一性。

（3）堆

堆的定义：

堆是满足下面条件的完全二叉树：

a．父结点的键值总是大于或等于（小于或等于）任何一个子节点的键值。

b．每个结点的左子树和右子树都是一个二叉堆（都是最大堆或最小堆）。

当父结点的键值总是大于或等于任何一个子节点的键值时为最大堆。当父结点的键值总是小于或等于任何一个子节点的键值时为最小堆

堆的基本操作

1 ）Heapify(A,n,t)

该操作主要用于维持堆的基本性质。假定 t 的左右子树都已经是堆，然后调整以 t 为根的子树，使之成为堆。

```
void Heapify(int A[], int n, int t)
 {
    int left = 2*t;
    int right = 2*t+1;
    int max = t;
    if(left <= n)         max = A[left] > A[max] ? left : max;
    if(right <= n)          max = A[right] > A[max] ? right : max;
    if(max != A[t])
    {
       swap(A, max, t);
       Heapify(A, n, max);
    }
 }
```

2）BuildHeap(A,n)

该操作主要是将数组 A 转化成一个大顶堆。思想是，先找到堆的最后一个非叶子节点（即为第 n/2 个节点），然后从该节点开始，从后往前逐个调整每个子树，使之称为堆，最终整
个数组便是一个堆。

```
void BuildHeap(int A[], int n)
 {
    int i;
    for(i = n/2; i<=n; i++)
```

```
    Heapify(A, n, i);
  }
```

3）GetMaximum(A,n)

该操作主要是获取堆中最大的元素，同时保持堆的基本性质。堆的最大元素即为第一个元素，将其保存下来，同时将最后一个元素放到 A[1]位置，之后从上往下调整 A，使之成为一个堆。

```
void GetMaximum(int A[], int n)
 {
    int max = A[1];
    A[1] = A[n];
    n--;
    Heapify(A, n, 1);
    return max;
 }
```

4） Insert(A, n, t)

向堆中添加一个元素 t，同时保持堆的性质。算法思想是，将 t 放到 A 的最后，然后从该元素开始，自下向上调整，直至 A 成为一个大顶堆。

```
void Insert(int A[], int n, int t)
 {
    n++;
    A[n] = t;
    int p = n;
    while(p >1 && A[floor(p/2)] < t)
    {
       A[p] = A[floor(p/2)];
       p = floor(p/2);
     }
     A[p] = t;
  }
```

3.8.3 典型例题

例 3.8.1 选择题

1、一棵完全二叉树的结点总数为 18，其叶结点数为（）。

A.7 个　　B.8 个　　C.9 个　　D.10 个

2、一棵树 T 有 2 个度数为 2 的结点、有 1 个度数为 3 的结点、有 3 个度数为 4 的结点，那么树 T 有（）个树叶。

A.14　　B.6　　C.18　　D.7

3、一棵非空二叉树的先序遍历序列和后序遍历序列正好相反，则该二叉树一定满足。

A.所有结点均无左孩子　　B.所有的结点均无右孩子

C.只有一个叶子结点　　D.是任意一棵二叉树

4、某二叉树中序序列为 abcdefg，后序序列为 bdcafge，则前序序列是。

A.egfacbdB.eacbdgfC.eagcfbd　　D.以上答案都不对

5、一棵非空二叉树的先序遍历序列和后序遍历序列正好相反，则该二叉树一定满足。

A.所有结点均无左孩子　　B.所有的结点均无右孩子

C.只有一个叶子结点　　D.是任意一棵二叉树

例 3.8.2 二叉树的创建与遍历

建立二叉树，然后实现：输出先序遍历、中序遍历、后序遍历的结果。

输入：

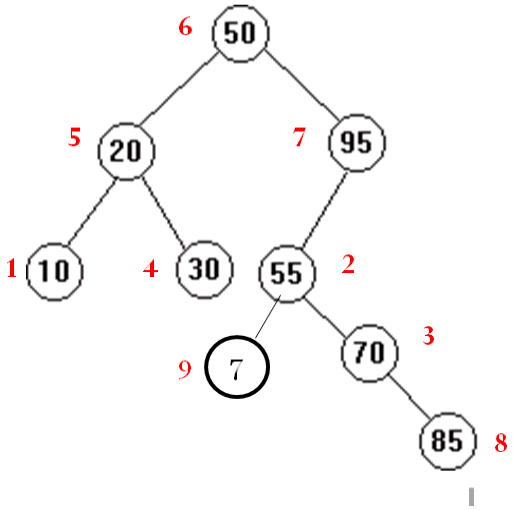
第一行：结点个数 n。N<=100;

以下 n 行，每行 4 个数依次是：结点编号，值，左孩子编号，右孩子编号。

输出：根、先中后序遍历结果。

样例输入：

```
9
6 50 5 7
5 20 1 4
1 10 0 0
4 30 0 0
7 95 2 0
2 55 9 3
3 70 0 8
8 85 0 0
9 7 0 0
```



分析：

我们可以用数组来实现排序二叉树的存储。

实现方法：

```
#include <cstdio>
#include <iostream>
using namespace std;
#define maxn 110
int f[maxn],l[maxn],r[maxn];
int v[maxn];
int n,t;
void init(){          //建树
    int p,k,lch,rch; //结点编号，权值，左右孩子指针
    scanf("%d",&n);
    for(int i=1;i<=n;i++){
        scanf("%d%d%d%d",&p,&k,&lch,&rch);
        v[p]=k;
        l[p]=lch; r[p]=rch;
        if(lch) f[lch]=p;
        if(rch) f[rch]=p;
    }
}
int root(){               //找根结点
    for(int i=1;i<=n;i++){
        if(f[i]==0) return i;
    }
}
```

```
void preorder(int i){   //先序遍历
    if(i){
        printf("%d ",v[i]);
        preorder(l[i]);
        preorder(r[i]);
    }
}
void inorder(int i){    //中序遍历
    if(i){
        inorder(l[i]);
        printf("%d ",v[i]);
        inorder(r[i]);
    }
}
void sucorder(int i){   //后序遍历
    if(i){
        sucorder(l[i]);
        sucorder(r[i]);
        printf("%d ",v[i]);
    }
}
void leaf(int i){       //输出叶结点
    if(i){
        if(l[i]+r[i]==0) printf("%d ",i);
        leaf(l[i]);
        leaf(r[i]);
    }
}
int h(int i){        //计算树的高度
    if(i==0) return 0;
    return max(h(l[i]),h(r[i]))+1;
}
int main(){
    freopen("tree.in","r",stdin);
    init();
    t=root(); printf("%d\n",t);
    preorder(t); printf("\n");
    inorder(t); printf("\n");
    sucorder(t); printf("\n");
    leaf(t); printf("\n");
    printf("%d\n",h(t));
    return 0;
}
```

例 3.8.3 已知先序和中序求后序

输入一棵二叉树的先序遍历和中序遍历序列，输出它的后序遍历序列。

样例输入：

DBACEGF　ABCDEFG

BCAD　CBAD

样例输出：

ACBFGED

CDAB

分析：

先序遍历的第一个字符是根，因此只需在中序遍历中找到它，就知道左右子树的先序和后序遍历了。可以编写一个递归程序实现。

实现方法：

```
#include<stdio.h>
#include<string.h>
constint MAXN = 30;

void build(int n, char* s1, char* s2, char* s) {
if(n <= 0) return;
int p = strchr(s2, s1[0]) - s2;   //找到根结点在中序遍历中位置
  build(p, s1+1, s2, s);              //递归构造左子树的后序遍历
  build(n-p-1, s1+p+1, s2+p+1, s+p);//递归构造右子树的后序遍历
  s[n-1] = s1[0];                     //把根结点添加到最后
}

int main() {
char s1[MAXN], s2[MAXN], ans[MAXN];
while(scanf("%s%s", s1, s2) == 2) {
int n = strlen(s1);
build(n, s1, s2, ans);
ans[n] = '\0';           //别忘了加上字符串结束标志
printf("%s\n", ans);
  }
return 0;
}
```

例 3.8.4 二叉排序树

根据输入的 n 个数建立二叉排序树，并从小到大输出。

输入：共两行第一行 n，第二行用空格隔开的 n 个数。

输出：共一行。排好序后的 n 个数

样例输入：

```
8
10  5  20  15  25  30  8  7
```

样例输出：

```
     5 7 8 10 15 20 25 30
```

分析：对二叉排序树中序遍历，可以得到从小到大的输出。

实现方法：

```
#include <cstdio>
#define maxn 110
```

```
int l[maxn],r[maxn],v[maxn];
intn,x,p;
void ldr(int i){        //中序遍历
    if(i==0) return;
    ldr(l[i]);
    printf("%d ",v[i]);
    ldr(r[i]);
}
void tree(inti,int x){     //把 x 加到以 i 为根结点的子树中
    if(x<v[i]){
        if(l[i]==0){
            p++; v[p]=x; l[i]=p;
        }else tree(l[i],x);
    }else{
        if(r[i]==0){
            p++; v[p]=x; r[i]=p;
        }else tree(r[i],x);
    }
}
inttmin(int i){
    if(l[i]==0) return v[i];
    returntmin(l[i]);
}
inttmax(int i){
    if(r[i]==0) return v[i];
    returntmax(r[i]);
}
int main(){
    scanf("%d",&n);
    scanf("%d",&x);
    p=1; v[1]=x;  //以第一个元素为根
    for(int i=2;i<=n;i++){
        scanf("%d",&x);
        tree(1,x);
    }
    ldr(1); printf("\n");
    printf("min=%d\n",tmin(1));
    printf("max=%d\n",tmax(1));
}
```

例 3.8.5 烽火传递

烽火台又称烽燧，是重要的军事防御设施，一般建在险要处或交通要道上。一旦有敌情发生，白天燃烧柴草，通过浓烟表达信息；夜晚燃烧干柴，以火光传递军情。在某两座城市之间有 n 个烽火台，每个烽火台发出信号都有一定的代价。为了使情报准确地传递，在连续 m 个烽火台中至少要有一个发出信号。现输入 n、m 和每个烽火台发出信号的代价，请计算总共最少花费多少代价，才能使敌军来袭之时，情报能

在这两座城市之间准确传递。

例如，有 5 个烽火台，它们发出信号的代价依次为 1、2、5、6、2，且 m 为 3，则总共最少花费的代价为 4，即由第 2 个和第 5 个烽火台发出信号。

分析：

设 F[I]表示将信息从 1 传到 I 所需花费的最小代价。由于要点燃 I，就必须点燃 [I-M+1，I-1]中的某一个，因而得到状态转移方程

F[I]：=MIN（F[K]+A[I]）；其中 K 属于区间[I-M+1，I-1]（不妨记为区间[A，B]）；

化简就是 F[I]：=A[I]+MIN（F[K]）；K 属于[A，B]；MIN（F[K]）的求解可以考虑使用堆进行优化。

```
#include <iostream>
using namespace std;
const int SIZE = 100;
int n, m, r, value[SIZE], heap[SIZE],
    pos[SIZE], home[SIZE], opt[SIZE];
    //heap[i]表示用顺序数组存储的堆 heap 中第 i 个元素的值
    //pos[i]表示 opt[i]在堆 heap 中的位置，即 heap[pos[i]]=opt[i]
    //home[i]表示 heap[i]在序列 opt 中的位置，即 opt[home[i]]=heap[i]
void swap(int i, int j)
//交换堆中的第 i 个和第 j 个元素
{
    int tmp;
    pos[home[i]] = j;
    pos[home[j]] = i;
    tmp = heap[i];
    heap[i] = heap[j];
    heap[j] = tmp;
    tmp = home[i];
    home[i] = home[j];
    home[j] = tmp;
}

void add(int k)
//在堆中插入 opt[k]
{
    int i;
    r++;
    heap[r] =    ①    ;
    pos[k] = r;
       ②    ;
    i = r;
    while ((i > 1) && (heap[i] < heap[i / 2])) {
       swap(i, i / 2);
       i /= 2;
    }
}
```

```
void remove(int k)
//在堆中删除 opt[k]
{
    int i, j;
    i = pos[k];
    swap(i, r);
    r--;
    if (i == r + 1)
        return;
    while ((i > 1) && (heap[i] < heap[i / 2])) {
        swap(i, i / 2);
        i /= 2;
    }
    while (i + i <= r) {
        if ((i + i + 1 <= r) && (heap[i + i + 1] < heap[i + i]))
            j = i + i + 1;
        else
                ③    ;
        if (heap[i] > heap[j]) {
                ④    ;
            i = j;
        }
        else
            break;
    }
}
int main()
{
    int i;

    cin>>n>>m;
    for (i = 1; i <= n; i++)
        cin>>value[i];
    r = 0;
    for (i = 1; i <= m; i++) {
        opt[i] = value[i];
        add(i);
    }
    for (i = m + 1; i <= n; i++) {
        opt[i] =     ⑤    ;
        remove(    ⑥    );
        add(i);
    }
    cout<<heap[1]<<endl;
    return 0;
```

```
}
```

答案：

① opt[k] ② home[r] = k ③ j = i + i（或 j = 2 * i 或 j = i * 2）

④ swap(i, j)（或 swap(j, i)）⑤ value[i] + heap[1]（或 heap[1] + value[i]）

⑥ i - m

3.8.3 课后训练

1、上机实现例 3.82 例 3.83 例 3.84

2、合并果子(fruit)

在一个果园里，多多已经将所有的果子打了下来，而且按果子的不同种类分成了不同的堆。多多决定把所有的果子合成一堆。每一次合并，多多可以把两堆果子合并到一起，消耗的体力等于两堆果子的重量之和。可以看出，所有的果子经过 n-1 次合并之后，就只剩下一堆了。多多在合并果子时总共消耗的体力等于每次合并所耗体力之和。 因为还要花大力气把这些果子搬回家，所以多多在合并果子时要尽可能地节省体力。假定每个果子重量都为 1，并且已知果子的种类数和每种果子的数目，你的任务是设计出合并的次序方案，使多多耗费的体力最少，并输出这个最小的体力耗费值。 例如有 3 种果子，数目依次为 1，2，9。可以先将 1、2 堆合并，新堆数目为 3，耗费体力为 3。接着，将新堆与原先的第三堆合并，又得到新的堆，数目为 12，耗费体力为 12。所以多多总共耗费体力=3+12=15。可以证明 15 为最小的体力耗费值。

输入格式：输入包括两行，第一行是一个整数 n(1<＝n<=10000)，表示果子的种类数。第二行包含 n 个整数，用空格分隔，第 i 个整数 ai(1<＝ai<=20000)是第 i 种果子的数目。

输出格式：输出包括一行，这一行只包含一个整数，也就是最小的体力耗费值。输入数据保证这个值小于 2^31。

样例输入：

```
3
1 2 9
```

样例输出：

```
15
```

数据规模：

对于 30％的数据，保证有 n<=1000：

对于 50％的数据，保证有 n<=5000；

对于全部的数据，保证有 n<=10000。