Lesson1

> **DFS（执着）**
>
> 数据结构：栈 空间：**O(h)** 在空间上比BFS有优势，h表示高度
>
> 不具有最短性
>
> 两个特性
>
> - **回溯**
>
>   回溯完之后注意**恢复现场**
>
> - **剪枝**
>
> 顺序与搜索树
>
> - e1: 全排列问题
>
> ```java
> static int N = 10;
>
> static int n;
> static int[] path = new int[N];
> static boolean[] st = new boolean[N];
>
> static void dfs(int u) {
>     if (u == n) {   //退出条件
>         for (int i=0; i<n; i++) out.print(path[i]+" ");
>         out.println();
>         return;
>     }
>
>     for (int i=1; i<=n; i++)
>         if (!st[i]) {
>             path[u] = i;
>             st[i] = true;
>             dfs(u+1);
>             st[i] = false;  //恢复现场
>         }
> }
>
> public static void main(String[] args) throws Exception {
>     ins.nextToken(); n = (int)ins.nval;
>
>     dfs(0);
>
>     out.flush();
> }
> ```
>
> - e2: n皇后问题
>
>   第一种搜索顺序：
>
> ```java
> static int N = 10;
> ```

```java

static int n;
static char[][] g = new char[N][N];
static boolean[] col = new boolean[N], dg = new boolean[N*2], udg = new boolean[N*2];

static void dfs(int u) {
    if (u == n) {
        for (int i=0; i<n; i++) {
            for (int j=0; j<n; j++) out.print(g[i][j]);
            out.println();
        }
        out.println();
        return;
    }

    for (int i=0; i<n; i++)
        if (!col[i] && !dg[u+i] && !udg[u-i+n]) {
            g[u][i] = 'Q';
            col[i] = dg[u+i] = udg[u-i+n] = true;
            dfs(u+1);
            col[i] = dg[u+i] = udg[u-i+n] = false;
            g[u][i] = '.';
        }
}

public static void main(String[] args) throws Exception {
    ins.nextToken(); n = (int)ins.nval;

    for (int i=0; i<n; i++)
        for (int j=0; j<n; j++) g[i][j] = '.';

    dfs(0);

    out.flush();
}
```

第二种搜索顺序：

```java
static int N = 10;

static int n;
static char[][] g = new char[N][N];
static boolean[] row = new boolean[N], col = new boolean[N], dg = new boolean[N*2], udg = new boolean[N*2];

static void dfs(int x, int y, int s) {
    if (y == n) {
        y = 0; x++;
    }

    if (x == n) {
        if (s == n) {
            for (int i=0; i<n; i++) {
                for (int j=0; j<n; j++) out.print(g[i][j]);
                out.println();
            }
            out.println();
        }
        return;
    }

    //不放皇后
    dfs(x, y+1, s);

    //放皇后
    if (!row[x] && !col[y] && !dg[x+y] && !udg[y-x+n]) {
        g[x][y] = 'Q';
        row[x] = col[y] = dg[x+y] = udg[y-x+n] = true;
        dfs(x, y+1, s+1);
        row[x] = col[y] = dg[x+y] = udg[y-x+n] = false;
        g[x][y] = '.';
    }
}

public static void main(String[] args) throws Exception {
    ins.nextToken(); n = (int)ins.nval;

    for (int i=0; i<n; i++)
        for (int j=0; j<n; j++) g[i][j] = '.';

    dfs(0, 0, 0);

    out.flush();
}
```

BFS（稳重，层层遍历）

数据结构：队列 空间：O(2^h)，h：高度

当每条边权重相同时，能找到最短路（DFS不具备）

```
queue <- 初始
while queue不空 {
    t <- 对头
    扩展 t 所有邻点
}
```

- e1：走迷宫

```
static int N = 110;

static int n, m;
static int[][] g = new int[N][N], d = new int[N][N];
static int hh, tt = -1;
static PII[] q = new PII[N*N];
static PII[][] pre = new PII[N][N];

static int bfs() {
    for (int i=0; i<n; i++) Arrays.fill(d[i], -1);

    int[] dx = {-1, 0, 1, 0}, dy = {0, 1, 0, -1};

    d[0][0] = 0;
    q[++tt] = new PII(0, 0);

    while (hh <= tt) {
        PII t = q[hh++];

        for (int i=0; i<4; i++) {
            int x = t.first+dx[i], y = t.second+dy[i];
            if (x >= 0 && x < n && y >= 0 && y < m && g[x][y] != 1 && d[x][y] == -1) {
                d[x][y] = d[t.first][t.second]+1;
                // pre[x][y] = t;
                q[++tt] = new PII(x, y);
            }
        }
    }

    // 打印路径
    // int x = n-1, y = m-1;
    // while (x > 0 || y > 0) {
    //     out.println(x+" "+y);
    //     PII t = pre[x][y];
    //     x = t.first; y = t.second;
    // }

    return d[n-1][m-1];
}

public static void main(String[] args) throws Exception {
    ins.nextToken(); n = (int)ins.nval;
    ins.nextToken(); m = (int)ins.nval;

    for (int i=0; i<n; i++)
        for (int j=0; j<m; j++) { ins.nextToken(); g[i][j] = (int)ins.nval; }

    out.println(bfs());

    out.flush();
}

static class PII {
    int first, second;

    PII (int f, int s) {
        first = f; second = s;
    }
}
```

- e2: 八数码问题

```java
static Queue<String> q = new LinkedList<String>();
static Map<String, Integer> d = new HashMap<String, Integer>();

static int bfs(String st) {
    d.put(st, 0); q.offer(st);

    int[] dx = {-1, 0, 1, 0}, dy = {0, 1, 0, -1};

    while (!q.isEmpty()) {
        String t = q.poll();

        if (t.equals("12345678x")) return d.get(t);

        int k = t.indexOf("x"), dist = d.get(t);
        int x = k/3, y = k % 3;

        for (int i=0; i<4; i++) {
            int a = x+dx[i], b = y+dy[i];

            if (a >= 0 && a < 3 && b >= 0 && b < 3) {
                char[] arr = t.toCharArray();
                char tp = arr[k]; arr[k] = arr[3*a+b]; arr[3*a+b] = tp;
                String str = new String(arr);

                if (d.get(str) == null) {    //保证队列一定会清空
                    q.offer(str);
                    d.put(str, dist+1);
                }
            }
        }

    }

    return -1;
}

public static void main(String[] args) throws Exception {
    String str = inb.readLine().replaceAll(" ", "");

    out.println(bfs(str));

    out.flush();
}
```

树与图的存储

树是一种特殊的图（无环连通图）

图的类型

- 有向图

  存储方式

  - 邻接矩阵，使用二维数组 g[a, b] （不能保存重边）

空间复杂度n^2，适用于稠密图

  - **邻接表**（每个结点开一个单链表，与拉链法哈希表一直）

    适用于**稀疏图**

```
const int N = 100010, M = N*2;  //N代表结点数，M代表边数

int h[N], e[M], ne[M], idx;

void add(int a, int b) {
    e[idx] = b, ne[idx] = h[a], h[a] = idx++;
}
```

- 无向图

  对于一条无向边，建两条有向边

树与图的深度优先遍历

- 遍历方式

```
const int N = 100010, M = N*2;

int h[N], e[M], ne[M], idx;
bool st[N];

void add(int a, int b) {
    e[idx] = b, ne[idx] = h[a], h[a] = idx++;
}

void dfs(int u) {
    st[u] = true;   //已经被遍历

    for (int i=h[u]; i!=-1; i=ne[i]) {
        int j = ne[i];
        if (!st[j]) dfs(j);
    }
}
```

- e1: 树的重心

```
static int N = 100010, M = 2*N; //注意无向图

static int n;
static int idx;
static int[] h = new int[N], e = new int[M], ne = new int[M];
static boolean[] st = new boolean[N];
static int ans = N;

static void add(int a, int b) {
    e[idx] = b; ne[idx] = h[a]; h[a] = idx++;
}

static int dfs(int u) { //返回以u为根的子树的结点个数，包括u
    st[u] = true;

    int res = 0, sum = 1;   //res存储若删去当前点，剩下的连通块点数最大值
    for (int i=h[u]; i!=-1; i=ne[i]) {
        int j = e[i];
        if (!st[j]) {
            int s = dfs(j);
            sum += s;
            res = Math.max(res, s);
        }
    }

    res = Math.max(res, n-sum);
    ans = Math.min(res, ans);

    return sum;
}

public static void main(String[] args) throws Exception {
    ins.nextToken(); n = (int)ins.nval;

    Arrays.fill(h, -1);

    for (int i=0; i<n-1; i++) {
        ins.nextToken(); int a = (int)ins.nval;
        ins.nextToken(); int b = (int)ins.nval;
        add(a, b); add(b, a);
    }

    dfs(1);

    out.println(ans);

    out.flush();
}
```

树与图的宽度优先遍历

- e1: 图中点的层次

```
static int N = 100010, M = N;

static int n, m;
static int idx;
static int[] h = new int[N], e = new int[M], ne = new int[M];
static int[] d = new int[N];
static int hh, tt = -1;
static int[] q = new int[N];

static void add(int a, int b) {
```

```java
    e[idx] = b; ne[idx] = h[a]; h[a] = idx++;
}

static int bfs() {
    Arrays.fill(d, -1);

    d[1] = 0;
    q[++tt] = 1;

    while (hh <= tt) {
        int t = q[hh++];

        for (int i=h[t]; i!=-1; i=ne[i]) {
            int j = e[i];
            if (d[j] == -1) {
                d[j] = d[t]+1;
                q[++tt] = j;
            }
        }
    }

    return d[n];
}


public static void main(String[] args) throws Exception {
    ins.nextToken(); n = (int)ins.nval;
    ins.nextToken(); m = (int)ins.nval;

    Arrays.fill(h, -1);

    for (int i=0; i<m; i++) {
        ins.nextToken(); int a = (int)ins.nval;
        ins.nextToken(); int b = (int)ins.nval;
        add(a, b);
    }

    out.println(bfs());

    out.flush();
}
```

> **拓扑排序（有向图宽搜的应用）**
>
> 有环图不存在拓扑排序。可以证明，有向无环图一定存在拓扑序列
>
> 度数
>
> - 入度：有多少边指入，**入度为0**的点可以作为**拓扑序列的起点**
> - 出度：有多少边指出
>
> 步骤

```
queue <- 所有入度为0的点
while queue不空 {
    t <- 队头
    枚举t的所有出边 t -> j
        删掉 t -> j, d[j]--;
        if (d[j] == 0) queue <- j;
}
```

- e1: 有向图的拓扑序列

```java
static int N = 100010, M = N;

static int n, m;
static int idx;
static int[] h = new int[N], e = new int[M], ne = new int[M];
static int hh, tt = -1;
static int[] q = new int[N];
static int[] d = new int[N];    //d[i]表示结点i的入度

static void add(int a, int b) {
    e[idx] = b; ne[idx] = h[a]; h[a] = idx++;
}

static boolean topoSort() {
    for (int i=1; i<=n; i++)
        if (d[i] == 0) q[++tt] = i; //初始将所有入读为0的点加入队列

    while (hh <= tt) {
        int t = q[hh++];

        for (int i=h[t]; i!=-1; i=ne[i]) {
            int j = e[i];
            d[j]--;
            if (d[j] == 0) q[++tt] = j;
        }
    }

    return tt == n-1;
}

public static void main(String[] args) throws Exception {
    ins.nextToken(); n = (int)ins.nval;
    ins.nextToken(); m = (int)ins.nval;

    Arrays.fill(h, -1);

    for (int i=0; i<m; i++) {
        ins.nextToken(); int a = (int)ins.nval;
        ins.nextToken(); int b = (int)ins.nval;
        add(a, b); d[b]++;
    }

    if (topoSort())
        for (int i=0; i<n; i++) out.print(q[i]+" ");
    else out.print("-1");

    out.flush();
}
```