

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

```
1 static int N = 10;
2
3 static int n;
4 static int[] path = new int[N];
5 static boolean[] st = new boolean[N];
6
7 static void dfs(int u) {
8     if (u == n) { //退出条件
9         for (int i=0; i<n; i++) out.print(path[i]+" ");
10        out.println();
11        return;
12    }
13
14    for (int i=1; i<=n; i++)
15        if (!st[i]) {
16            path[u] = i;
17            st[i] = true;
18            dfs(u+1);
19            st[i] = false; //恢复现场
20        }
21 }
22
23 public static void main(String[] args) throws Exception {
24     ins.nextToken(); n = (int)ins.nval;
25
26     dfs(0);
27
28     out.flush();
29 }
```

- e2: n皇后问题
第一种搜索顺序:

```
1 static int N = 10;
```

```

2
3 static int n;
4 static char[][] g = new char[N][N];
5 static boolean[] col = new boolean[N], dg = new boolean[N*2], udg = new
  boolean[N*2];
6
7 static void dfs(int u) {
8     if (u == n) {
9         for (int i=0; i<n; i++) {
10             for (int j=0; j<n; j++) out.print(g[i][j]);
11             out.println();
12         }
13         out.println();
14         return;
15     }
16
17     for (int i=0; i<n; i++)
18         if (!col[i] && !dg[u+i] && !udg[u-i+n]) {
19             g[u][i] = 'Q';
20             col[i] = dg[u+i] = udg[u-i+n] = true;
21             dfs(u+1);
22             col[i] = dg[u+i] = udg[u-i+n] = false;
23             g[u][i] = '.';
24         }
25 }
26
27 public static void main(String[] args) throws Exception {
28     ins.next token(); n = (int)ins.nval;
29
30     for (int i=0; i<n; i++)
31         for (int j=0; j<n; j++) g[i][j] = '.';
32
33     dfs(0);
34
35     out.flush();
36 }

```

第二种搜索顺序:

```

1 static int N = 10;
2
3 static int n;
4 static char[][] g = new char[N][N];
5 static boolean[] row = new boolean[N], col = new boolean[N], dg = new
  boolean[N*2], udg = new boolean[N*2];
6
7 static void dfs(int x, int y, int s) {
8     if (y == n) {
9         y = 0; x++;
10    }
11
12    if (x == n) {
13        if (s == n) {
14            for (int i=0; i<n; i++) {
15                for (int j=0; j<n; j++) out.print(g[i][j]);
16                out.println();

```

```

17         }
18         out.println();
19     }
20     return;
21 }
22
23 //不放皇后
24 dfs(x, y+1, s);
25
26 //放皇后
27 if (!row[x] && !col[y] && !dg[x+y] && !udg[y-x+n]) {
28     g[x][y] = 'Q';
29     row[x] = col[y] = dg[x+y] = udg[y-x+n] = true;
30     dfs(x, y+1, s+1);
31     row[x] = col[y] = dg[x+y] = udg[y-x+n] = false;
32     g[x][y] = '.';
33 }
34 }
35
36 public static void main(String[] args) throws Exception {
37     ins.nextToken(); n = (int)ins.nval;
38
39     for (int i=0; i<n; i++)
40         for (int j=0; j<n; j++) g[i][j] = '.';
41
42     dfs(0, 0, 0);
43
44     out.flush();
45 }

```

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)

```

1 queue <- 初始
2 while queue不空 {
3     t <- 对头
4     扩展 t 所有邻点
5 }

```

- e1: 走迷宫

```

1 static int N = 110;
2
3 static int n, m;
4 static int[][] g = new int[N][N], d = new int[N][N];
5 static int hh, tt = -1;
6 static PII[] q = new PII[N*N];
7 static PII[][] pre = new PII[N][N];
8

```

```

9  static int bfs() {
10     for (int i=0; i<n; i++) Arrays.fill(d[i], -1);
11
12     int[] dx = {-1, 0, 1, 0}, dy = {0, 1, 0, -1};
13
14     d[0][0] = 0;
15     q[++tt] = new PII(0, 0);
16
17     while (hh <= tt) {
18         PII t = q[hh++];
19
20         for (int i=0; i<4; i++) {
21             int x = t.first+dx[i], y = t.second+dy[i];
22             if (x >= 0 && x < n && y >= 0 && y < m && g[x][y] != 1 &&
d[x][y] == -1) {
23                 d[x][y] = d[t.first][t.second]+1;
24                 // pre[x][y] = t;
25                 q[++tt] = new PII(x, y);
26             }
27         }
28     }
29
30     // 打印路径
31     // int x = n-1, y = m-1;
32     // while (x > 0 || y > 0) {
33     //     out.println(x+" "+y);
34     //     PII t = pre[x][y];
35     //     x = t.first; y = t.second;
36     // }
37
38     return d[n-1][m-1];
39 }
40
41 public static void main(String[] args) throws Exception {
42     ins.nextToken(); n = (int)ins.nval;
43     ins.nextToken(); m = (int)ins.nval;
44
45     for (int i=0; i<n; i++)
46         for (int j=0; j<m; j++) { ins.nextToken(); g[i][j] =
(int)ins.nval; }
47
48     out.println(bfs());
49
50     out.flush();
51 }
52
53 static class PII {
54     int first, second;
55
56     PII (int f, int s) {
57         first = f; second = s;
58     }
59 }

```

- e2: 八数码问题

```

1  static Queue<String> q = new LinkedList<String>();
2  static Map<String, Integer> d = new HashMap<String, Integer>();
3
4  static int bfs(String st) {
5      d.put(st, 0); q.offer(st);
6
7      int[] dx = {-1, 0, 1, 0}, dy = {0, 1, 0, -1};
8
9      while (!q.isEmpty()) {
10         String t = q.poll();
11
12         if (t.equals("12345678x")) return d.get(t);
13
14         int k = t.indexOf("x"), dist = d.get(t);
15         int x = k/3, y = k % 3;
16
17         for (int i=0; i<4; i++) {
18             int a = x+dx[i], b = y+dy[i];
19
20             if (a >= 0 && a < 3 && b >= 0 && b < 3) {
21                 char[] arr = t.toCharArray();
22                 char tp = arr[k]; arr[k] = arr[3*a+b]; arr[3*a+b] = tp;
23                 String str = new String(arr);
24
25                 if (d.get(str) == null) { //保证队列一定会清空
26                     q.offer(str);
27                     d.put(str, dist+1);
28                 }
29             }
30         }
31
32     }
33
34     return -1;
35 }
36
37 public static void main(String[] args) throws Exception {
38     String str = inb.readLine().replaceAll(" ", "");
39
40     out.println(bfs(str));
41
42     out.flush();
43 }

```

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



```
1  const int N = 100010, M = N*2; //N代表结点数，M代表边数
2
3  int h[N], e[M], ne[M], idx;
4
5  void add(int a, int b) {
6      e[idx] = b, ne[idx] = h[a], h[a] = idx++;
7  }
```

- 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



```
1  const int N = 100010, M = N*2;
2
3  int h[N], e[M], ne[M], idx;
4  bool st[N];
5
6  void add(int a, int b) {
7      e[idx] = b, ne[idx] = h[a], h[a] = idx++;
8  }
9
10 void dfs(int u) {
11     st[u] = true; //已经被遍历
12
13     for (int i=h[u]; i!=-1; i=ne[i]) {
14         int j = ne[i];
15         if (!st[j]) dfs(j);
16     }
17 }
```

- e1: 树的重心



```
1  static int N = 100010, M = 2*N; //注意无向图
2
3  static int n;
4  static int idx;
5  static int[] h = new int[N], e = new int[M], ne = new int[M];
6  static boolean[] st = new boolean[N];
7  static int ans = N;
8
9  static void add(int a, int b) {
10     e[idx] = b; ne[idx] = h[a]; h[a] = idx++;
11 }
```

```

11 }
12
13 static int dfs(int u) { //返回以u为根的子树的结点个数，包括u
14     st[u] = true;
15
16     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
17     for (int i=h[u]; i!=-1; i=ne[i]) {
18         int j = e[i];
19         if (!st[j]) {
20             int s = dfs(j);
21             sum += s;
22             res = Math.max(res, s);
23         }
24     }
25
26     res = Math.max(res, n-sum);
27     ans = Math.min(res, ans);
28
29     return sum;
30 }
31
32 public static void main(String[] args) throws Exception {
33     ins.nextToken(); n = (int)ins.nval;
34
35     Arrays.fill(h, -1);
36
37     for (int i=0; i<n-1; i++) {
38         ins.nextToken(); int a = (int)ins.nval;
39         ins.nextToken(); int b = (int)ins.nval;
40         add(a, b); add(b, a);
41     }
42
43     dfs(1);
44
45     out.println(ans);
46
47     out.flush();
48 }

```

树与图的宽度优先遍历

- e1: 图中点的层次

```

1 static int N = 100010, M = N;
2
3 static int n, m;
4 static int idx;
5 static int[] h = new int[N], e = new int[M], ne = new int[M];
6 static int[] d = new int[N];
7 static int hh, tt = -1;
8 static int[] q = new int[N];
9
10 static void add(int a, int b) {

```

```

11     e[idx] = b; ne[idx] = h[a]; h[a] = idx++;
12 }
13
14 static int bfs() {
15     Arrays.fill(d, -1);
16
17     d[1] = 0;
18     q[++tt] = 1;
19
20     while (hh <= tt) {
21         int t = q[hh++];
22
23         for (int i=h[t]; i!=-1; i=ne[i]) {
24             int j = e[i];
25             if (d[j] == -1) {
26                 d[j] = d[t]+1;
27                 q[++tt] = j;
28             }
29         }
30     }
31
32     return d[n];
33 }
34
35
36 public static void main(String[] args) throws Exception {
37     ins.nextToken(); n = (int)ins.nval;
38     ins.nextToken(); m = (int)ins.nval;
39
40     Arrays.fill(h, -1);
41
42     for (int i=0; i<m; i++) {
43         ins.nextToken(); int a = (int)ins.nval;
44         ins.nextToken(); int b = (int)ins.nval;
45         add(a, b);
46     }
47
48     out.println(bfs());
49
50     out.flush();
51 }

```

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


```

1 queue <- 所有入度为0的点
2 while queue不空 {
3     t <- 队头
4     枚举t的所有出边 t -> j
5     删掉 t -> j, d[j]--;
6     if (d[j] == 0) queue <- j;
7 }

```

- e1: 有向图的拓扑序列

```

1 static int N = 100010, M = N;
2
3 static int n, m;
4 static int idx;
5 static int[] h = new int[N], e = new int[M], ne = new int[M];
6 static int hh, tt = -1;
7 static int[] q = new int[N];
8 static int[] d = new int[N];    //d[i]表示结点i的入度
9
10 static void add(int a, int b) {
11     e[idx] = b; ne[idx] = h[a]; h[a] = idx++;
12 }
13
14 static boolean topoSort() {
15     for (int i=1; i<=n; i++)
16         if (d[i] == 0) q[++tt] = i; //初始将所有入读为0的点加入队列
17
18     while (hh <= tt) {
19         int t = q[hh++];
20
21         for (int i=h[t]; i!=-1; i=ne[i]) {
22             int j = e[i];
23             d[j]--;
24             if (d[j] == 0) q[++tt] = j;
25         }
26     }
27
28     return tt == n-1;
29 }
30
31 public static void main(String[] args) throws Exception {
32     ins.nextToken(); n = (int)ins.nval;
33     ins.nextToken(); m = (int)ins.nval;
34
35     Arrays.fill(h, -1);
36
37     for (int i=0; i<m; i++) {
38         ins.nextToken(); int a = (int)ins.nval;
39         ins.nextToken(); int b = (int)ins.nval;
40         add(a, b); d[b]++;
41     }
42
43     if (topoSort())
44         for (int i=0; i<n; i++) out.print(q[i]+" ");

```

```
45     else out.print("-1");  
46  
47     out.flush();  
48 }
```