算法课第 1 次作业

1 题目 1 转换器

问题转换 把原题对应到稳定婚姻问题：n 条输出为 n 个女生，n 条输入为 n 个男生。对于每个男生 i，如果他和女生 j 在“输入 i”的从源头开始的第 k1 个链接点交汇，表示女生 j 是男生 i 排名第 k1 的心仪对象；反之，对每个女生 j，如果她和男生 i 在“输出 j”的从末尾反向开始的第 k2 个连接点交汇，表示男生 i 是女生 j 排名第 k2 的心仪对象。原问题转换为给定了 n 对男生和女生每个人对异性的喜好排序，求配对方式，使得配对稳定。

证明存在性 对于转换后的问题求解：每个循环内，所有未配对的男生向未配对的女生中自己最喜欢的求婚，所有接到求婚的女生在向自己求婚的男生中选择自己最喜欢的结成一队。由稳定婚姻可以必然存在一个可行解，下面证明该解同样满足原来的限制条件。 反证，假设不存在，不放设输入 i 和输出 j 被配对（配对点在 i 的 ip1 位置，j 的 jp1 位置），但是存在：（1）输入 i 上 ip2 位置（ip2<ip1），使得 ip2 在另一输出 k 上，且在 k 上 ip2 点之前 k 已经接通（2）输出 j 上 jp2 位置下游存在一个输入电流 i0 的交点 ip0 由对称性，只考虑第（2）种情况。由 i0 经过 j，说明 i0 最后配对的对象在 i0 心中排名低于 j，说明 i0 一定在之前的某轮向 j 表白且被拒了。j 没有选 i0，说明 j 选了比 i0 更好的（在 j 心中），但是实际 j 选的 i 没有 i0 排名高，矛盾！所以按照上述方法求得的配对满足上下游限制要求，也就是可行解。

求解算法 见“问题转换”

2 题目 2 求最大 n

$(a)n^2$ 由 $n^2 \leqslant 10^{10} \times 60 \times 60$
得到 $n \leqslant 6 \times 10^6$, 所以 n 最大为 6×10^6

$(b)n^3$ 由 $n^3 \leqslant 10^{10} \times 60 \times 60$
得到 $n \leqslant 33015$, 所以 n 最大为 33015

$(c)n^3$ 由 $100 \times n^2 \leqslant 10^{10} \times 60 \times 60$
得到 $n \leqslant 6 \times 10^6$, 所以 n 最大为 6×10^5

$(d)n \log n$ 令 $t = \log n$ $n \log n > n$ 得到 $2^t \times t \leqslant 10^{10} \times 60 \times 60$，两边取 log 得到 $t + \log t \leqslant 2 \log 6 + 12 \log 10$ 在区间 $[0, 2 \log 6 + 12 \log 10]$ 二分查找，当区间长度小于 0.000001 的时候停止得到 $n \approx 2^{41.31191}$

$(e)2^n$ 令 $2^n \leqslant 10^{10} \times 60 \times 60$, 取 log 得到 $n \leqslant 2\log 6 + 12\log 10$，n 最大为 45

$(f)2^{2^n}$ 令 $2^{2^n} \leqslant 10^{10} \times 60 \times 60$, 取 log 后再取 log 得到 $n \leqslant \log 2 \log 6 + 12 \log 10$，n 最大为 5

3 题目 3 比较大小

如无特殊说明，本题证明中的 $a < b$ 均表示

$$\lim_{n\to\infty} \frac{a}{b} = \infty$$

由

$$\log n < \sqrt{n} < n < n^2 < 2^n$$

得到

$$\sqrt{\log n} < \log n < \log n + 3 \times \log\log n < \log n + \frac{1}{3}\log n$$
$$< (\log n) \times (\log n) < (\sqrt{n}) \times (\sqrt{n}) = n < n^2 < 2^n$$

将上式每一个做 2 的指数，得到

$$g_1 = 2^{\sqrt{\log n}} < 2^{\log n} = n < 2^{\log n + 3 \times \log\log n} = g_4$$
$$< 2^{\log n + \frac{1}{3}\log n} = g_3 < 2^{(\log n)\times(\log n)} = g_5$$
$$< 2^{(\sqrt{n})\times(\sqrt{n})} = 2^n < 2^{n^2} = g_7 < 2^{2^n} = g_6$$

4 题目 4 判断函数复杂度真假

(a) false

e.g.

$$f(n) = 1, g(n) = 2$$

then

$$\log_2 f(n) = 0, \log_2 g(n) = 1$$

thus

$$O(\log_2 f(n)) = O(0) \neq O(\log_2 g(n)) = O(1)$$

(b) false
e.g.

$$f(n) = n + \log n, g(n) = n$$

then

$$O(f(n)) = O(n) = O(g(n)), \log_2 f(n) = 0, \log_2 g(n) = 1$$

thus

$$O(2^{f(n)}) = O(n \times 2^n) \neq O(2^{g(n)}) = O(2^n)$$

(c) true
given $f(n) \leqslant cg(n)$ for all $n \geqslant n_0$
we have
$(f(n))^2 \leqslant c^2(g(n))^2$ for all $n \geqslant n_0$

5 题目 5 找到最短路径数目

(算法思路) 采用 BFS，由近及远找到离起始点 v 的点。从开始点 v 作为结果集 $R_0()$ 表示从 v 一步走到的点数目（存储点集及其路径数），这里路径数为 1。之后第 i 轮开始时从 $R_{i-1}()$ 集合取出所有点，找到与他们直接链接的所有点，将其中未在之前判断过的点集中的点加入 $R_i()$, 其在 $R_i()$ 中的路径数为所有通向它的 $R_{i-1}()$ 集合中的点的路径数之和。等到搜索到目标点 w 停止，每个点至多扫描一次，每条边至多扫描一次，复杂度为 $O(m+n)$

(代码)

```
package test;

import java.util.HashSet;
import java.util.Iterator;

class DirNodes
{
        int NodeId;
        int [] directNodes;
}
public class BFS {
        int len;
        // represent edges
        DirNodes [] ConnectedNode = new DirNodes[len];
        // whether in set R_i()  yet
        int [] PathNr = new int[len];
        HashSet<Integer> Rset;
        int result;
        public void GetPathNr(int v, int w)
        {
                if(v == w)
                        result = 1;
                else{
                        //**********************
                        //**init ConnectedNode**
                        //**********************

                        for(int i = 0; i < len; i ++)
                                PathNr[i] = -1;
                        PathNr[v] = 1;
                        Rset = new HashSet<Integer>();
                        Rset.add(v);
                        RecursivelySearch(w, 0);
                }

        }
        private void RecursivelySearch(int w, int level) {
                // TODO Auto-generated method stub
                HashSet<Integer> NextRset =
                                new HashSet<Integer>();
                boolean findW = false;
                Iterator<Integer> it = Rset.iterator();
                while(it.hasNext()){
                        int nodeid = it.next();
                        for(int nextNodeId : ConnectedNode[nodeid].directNodes)
                        {
                                if(nextNodeId == w)
                                        findW = true;
                                if(PathNr[nextNodeId] < 0){
                                        NextRset.add(nextNodeId);
                                        PathNr[nextNodeId] = 0;
                                }
                                if(NextRset.contains(nextNodeId))
                                        PathNr[nextNodeId] += PathNr[nodeid];
                        }
                }
                if(findW)
                        result = PathNr[w];
                else if(NextRset.size() < 1){
                                result = 0;
                }else{
                        Rset = NextRset;
                        RecursivelySearch(w, level + 1);
                }
        }
}
```

2 3 4