图论基础

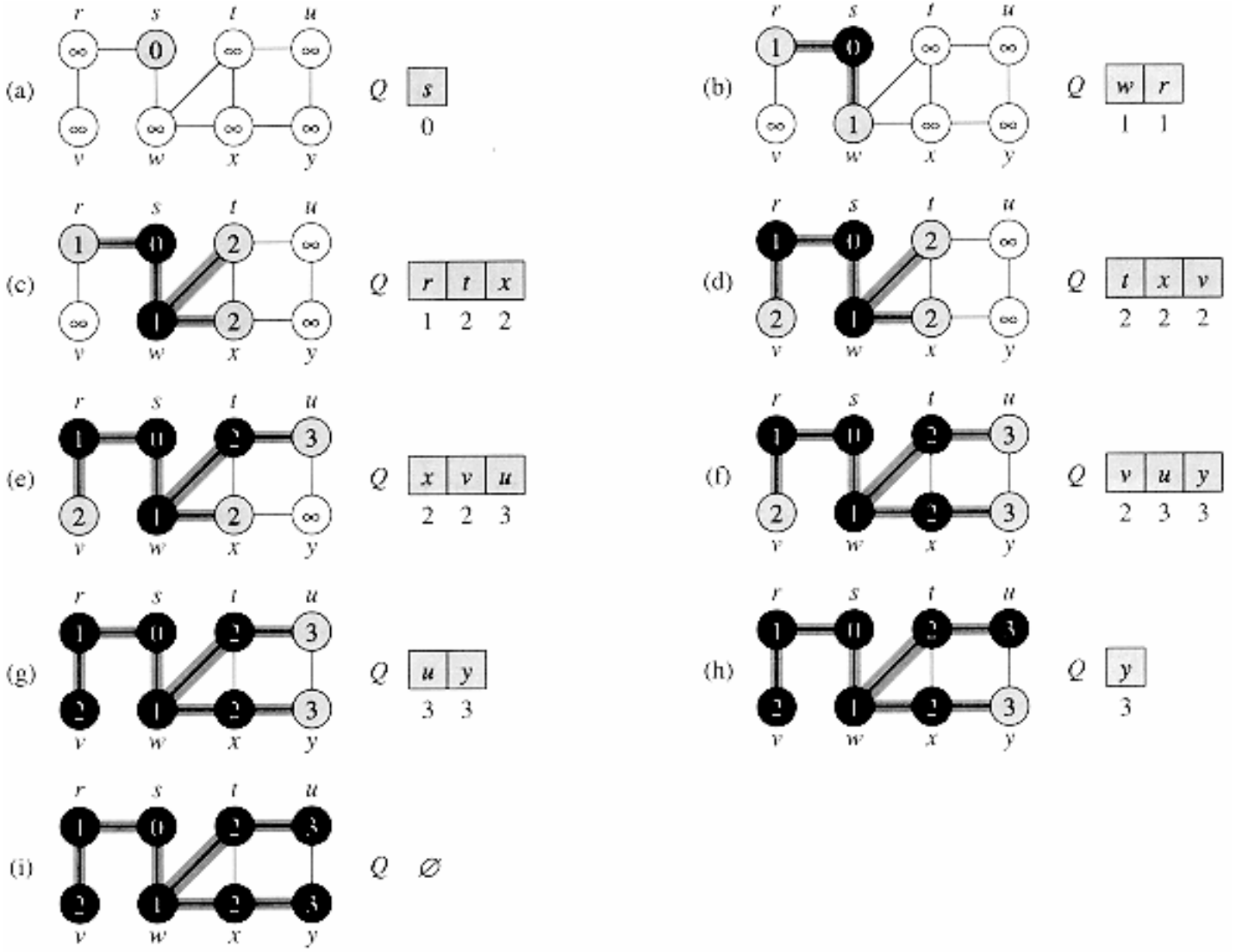
(版本20060709)

刘汝佳

目录

一、宽度优先遍历
二、深度优先遍历
三、最短路问题
四、最小生成树问题

一、宽度优先遍历


(a)
Q
s
0
(b)
Q
w r
1 1
(c)
Q
r t x
1 2 2
(d)
Q
t x v
2 2 2
(e)
Q
x v u
2 2 3
(f)
Q
v u y
2 3 3
(g)
Q
u y
3 3
(h)
Q
y
3
(i)
Q
∅

例1. 二分图判定

- 给出判断图是否为二分图的线性时间算法

例2. 树的直径

- 一棵树T的直径定义为结点两两间距离的最大值. 给出求树直径的线性时间算法

例3. 幼儿园小朋友分组

- 幼儿园里有*n*个小孩。不幸的是小孩常常打架。每个小孩有不超过3个仇敌。是否有可能将所有小孩分成两组使得每个小孩最多和他的一个仇敌同组？

分析

- 这道题目可以看作是给所有的点黑白二染色。先给1染黑色，放入队列。每次从队列中取出一个点，把与其相邻的还未染色的点染成相反的颜色（因为不同的连通分量之间互不相干，可以假设所有点构成一个连通图），这样一次广搜就能给所有的点一个初始色。如果发现某个点*x*有两个以上的仇敌与它染同样的颜色，就把*x*的颜色变一下。如此反复就能求得答案。
- **问题1:** 这样变是不是总能得到答案？
- **问题2:** 这样做的时间复杂度是多少？

重要结论

- 有一个重要的结论：一个点颜色至多变一下。也就是说不会出现某个点的颜色反复变的情况。假如这个结论成立，则上面提出的两个问题都能解决
  - 由于不会反复变，在有限步内（至多n步）就不能继续变了，这时的状态就是答案
  - 每个点至多变一次，所以时间复杂度是$O(n)$

证明

- 我们用广搜给每个点初始色，于是形成了一颗搜索树，深度为奇数的点染黑色、深度为偶数的点染白色。如果一个点*x*存在两个以上的仇敌和它染同样的颜色，我们称*x*是一个“坏点”。
- 任意一个非叶子结点显然不是一个坏点。首先是根结点*R*，由于我们采用的是广搜，所有与*R*有仇的点都被染成了白色（*R*是黑色），所以*R*必然不是坏点。对于任意一个非根、非叶子结点*p*，都存在一个父亲和至少一个孩子，所以*p*就至少有两个仇敌与它染不同颜色；而题目中说每个人至多3个仇敌，故而与*p*染同色的*p*的仇敌至多一个，因此*p*也不是坏点。
- 所以: 坏点必然是叶子结点！

坏点

- 设某个坏点*x*，与它染同色的两个仇敌是*a*，*b*（显然*a*,*b*都不是根结点）。我们把*x*反色，就相当于令*x*成为*a*的孩子。需证明***x*的颜色不可能再次改变**
- *x*改色的必要条件是*a*，*b*中至少有一个改色。由于*x*成为了*a*的孩子，*a*不是叶子结点，因此*a*不可能在*x*之前改色，所以必然是*b*改色。设*b*的父亲是*t*，那么*t*与*x*这两个*b*的仇敌都和*b*染不同的颜色，要想使得*b*改色，必须先把*t*，*x*两者至少一个改色。而*t*不是叶子结点，所以必须把*x*改色才有可能让*b*改色。

实现

- 归纳：x改色前b要改色；b改色前x要改色。这显然是一个不可能的任务。于是x的颜色不会被第二次改变（具体地说a，b的颜色也不会改变）。
- 具体实现的时候把所有的坏点放在一个栈里面，每次取一个点如果还是坏点就反色，同时考察所有与之相邻的点，看看是不是变成了坏点，如果是就放入栈中。总的时间复杂度是$O(n)$。

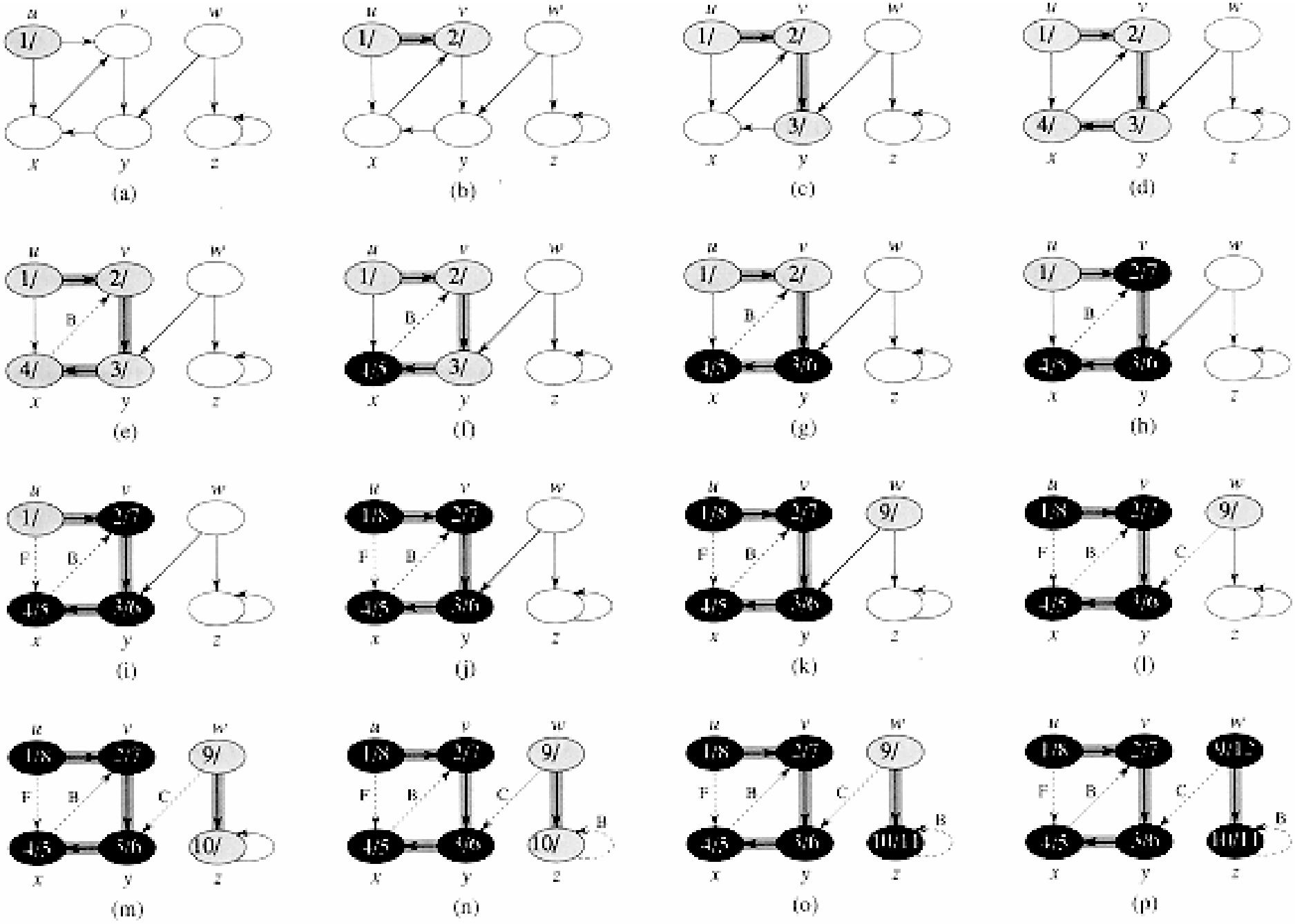
二、深度优先遍历

- 新发现的结点先扩展
- 得到的可能不是一棵树而是森林, 即深度优先森林(Depth-first forest)
- 特别之处: 引入时间戳(timestamp)
  - 发现时间d[v]: 变灰的时间
  - 结束时间f[v]: 变黑的时间
  - 1<=d[v] < f[v] <= 2|V|
- 初始化: time为0, 所有点为白色, dfs森林为空
- 对每个白色点u执行一次DFS-VISIT(u)

DFS-VISIT算法

- 初始化: time为0, 所有点为白色, dfs森林为空
- 对每个白色点u执行一次DFS-VISIT(u)
- 时间复杂度为O(n+m)

```
DFS-VISIT(u)
1  color[u] ← GRAY            ▷ White vertex u has just been discovered.
2  d[u] ← time ← time + 1
3  for each v ∈ Adj[u]        ▷ Explore edge (u, v).
4       do if color[v] = WHITE
5             then π[v] ← u
6                  DFS-VISIT(v)
7  color[u] ← BLACK           ▷ Blacken u; it is finished.
8  f[u] ← time ← time + 1
```


u
v
w
x
y
z
1/
2/
3/
4/
B
(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)
F
C
1/8
2/7
3/6
4/5
9/
10/
10/11
(i)
(j)
(k)
(l)
(m)
(n)
(o)
(p)

DFS树的性质

- 括号结构性质
- 对于任意结点对(u, v), 考虑区间[d[u], f[u]]和[d[v], f[v]], 以下三个性质恰有一个成立:
  - 完全分离
  - u的区间完全包含在v的区间内, 则在dfs树上u是v的后代
  - v的区间完全包含在u的区间内, 则在dfs树上v是u的后代
- 三个条件非常直观

DFS树的性质

- **定理1(嵌套区间定理):** 在DFS森林中v是u的后代当且仅当d[u]<d[v]<f[v]<f[u], 即区间包含关系. 由区间性质立即得到.
- **定理2(白色路径定理):** 在DFS森林中v是u的后代当且仅当在d[u]时刻(u刚刚被发现), v可以由u出发*只经过白色结点*到达. 证明: 由嵌套区间定理可以证明

边分类规则

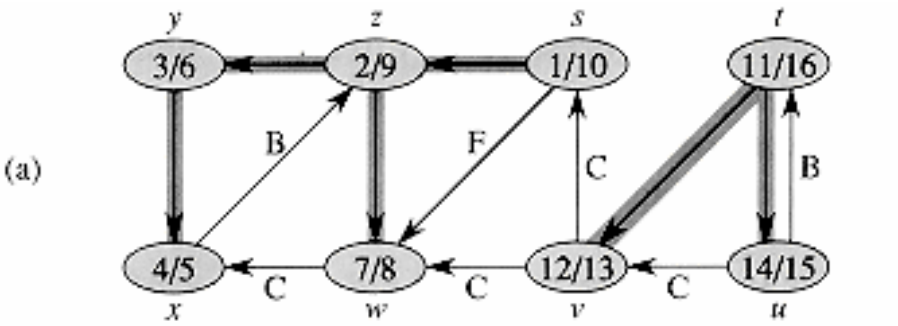
- 一条边(u, v)可以按如下规则分类
  - 树边(Tree Edges, T): v通过边(u, v)发现
  - 后向边(Back Edges, B): u是v的后代
  - 前向边(Forward Edges, F): v是u的后代
  - 交叉边(Cross Edges, C): 其他边，可以连接同一个DFS树中没有后代关系的两个结点, 也可以连接不同DFS树中的结点
- 判断后代关系可以借助定理1

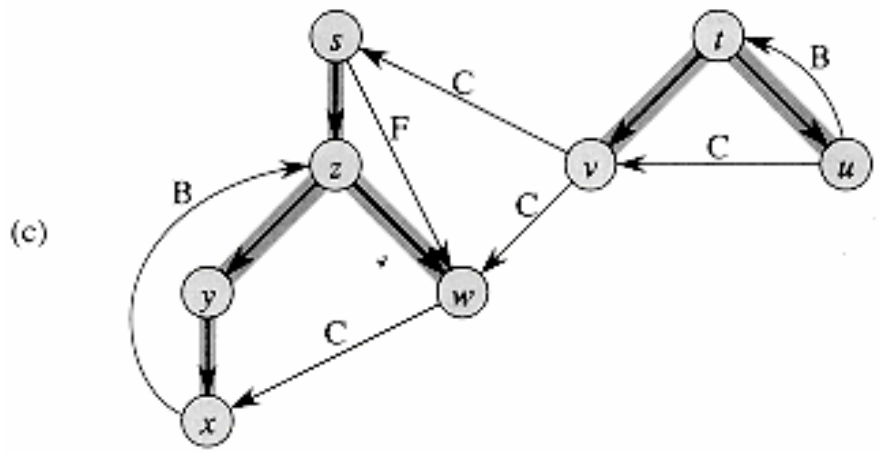
边分类算法

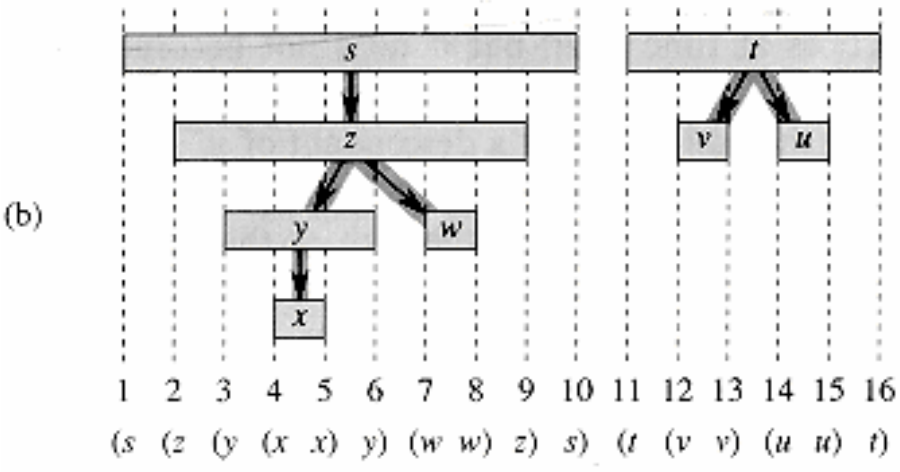
- 当(u, v)第一次被遍历, 考虑v的颜色
  - 白色, (u,v)为T边
  - 灰色, (u,v)为B边 (*只有它的祖先*是灰色)
  - 黑色: (u,v)为F边或C边. 此时需要进一步判断
    - d[u]<d[v]: F边 (v是u的后代, 因此为F边)
    - d[u]>d[v]: C边 (v早就被发现了, 为另一DFS树中)
- 时间复杂度: O(n+m)
- 定理: 无向图只有T边和B边 (易证)

DFS树的性质演示

- 图a: DFS森林
- 图b: 括号性质
- 图c: 重画以后的DFS森林, 边的分类更形象



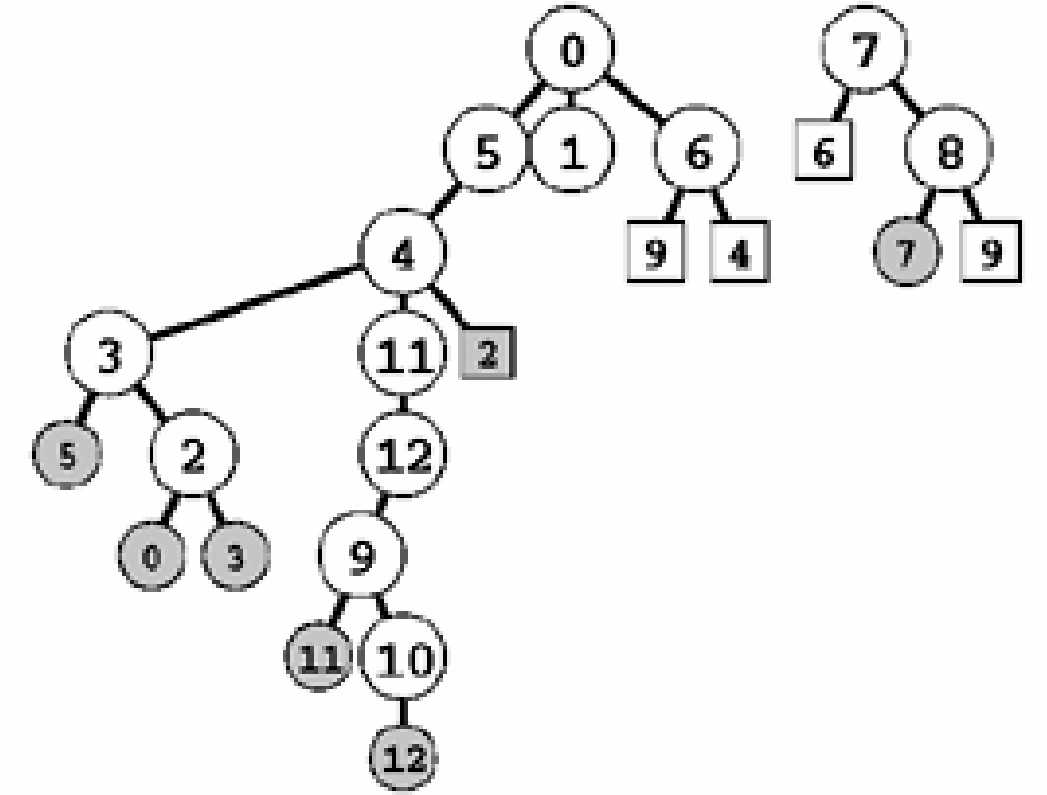


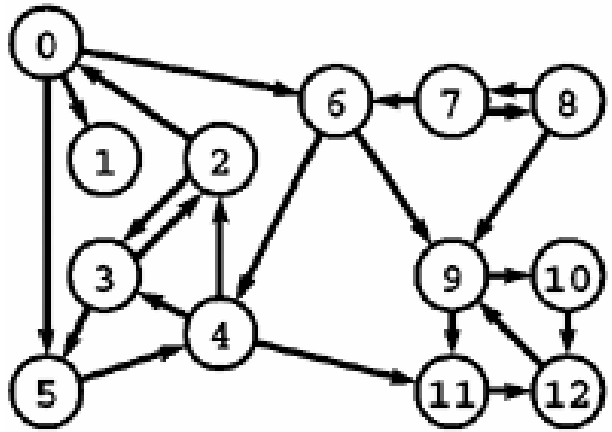


实现细节

- 颜色值以及时间戳可以省略, 用意义更加明确的pre数组和post代替d和f数组, pre[u]和post[u]代表*点**u**的先序/后序编号*, 则检查(u,v)可以写为
  - if (pre[v] == -1) dfs(v); //树边, 递归遍历
  - else if (post[v] == -1) show(“B”); //后向边
  - else if (pre[v] > pre[u]) show(“F”); // 前向边
  - else show(“C”); // 交叉边
- pre和post的初值均为-1, 方便了判断

使用pre和post的DFS





	0	1	2	3	4	5	6	7	8	9	10	11	12
pre	0	9	4	3	2	1	10	11	12	7	8	5	6
post	10	8	0	1	6	7	9	12	11	3	2	5	4

例3. 单连通判定

- 一个有向图是单连通的, 如果u可达v, 则u到v只有唯一一条简单路径. 设计一个判断一个图是否为单连通的算法

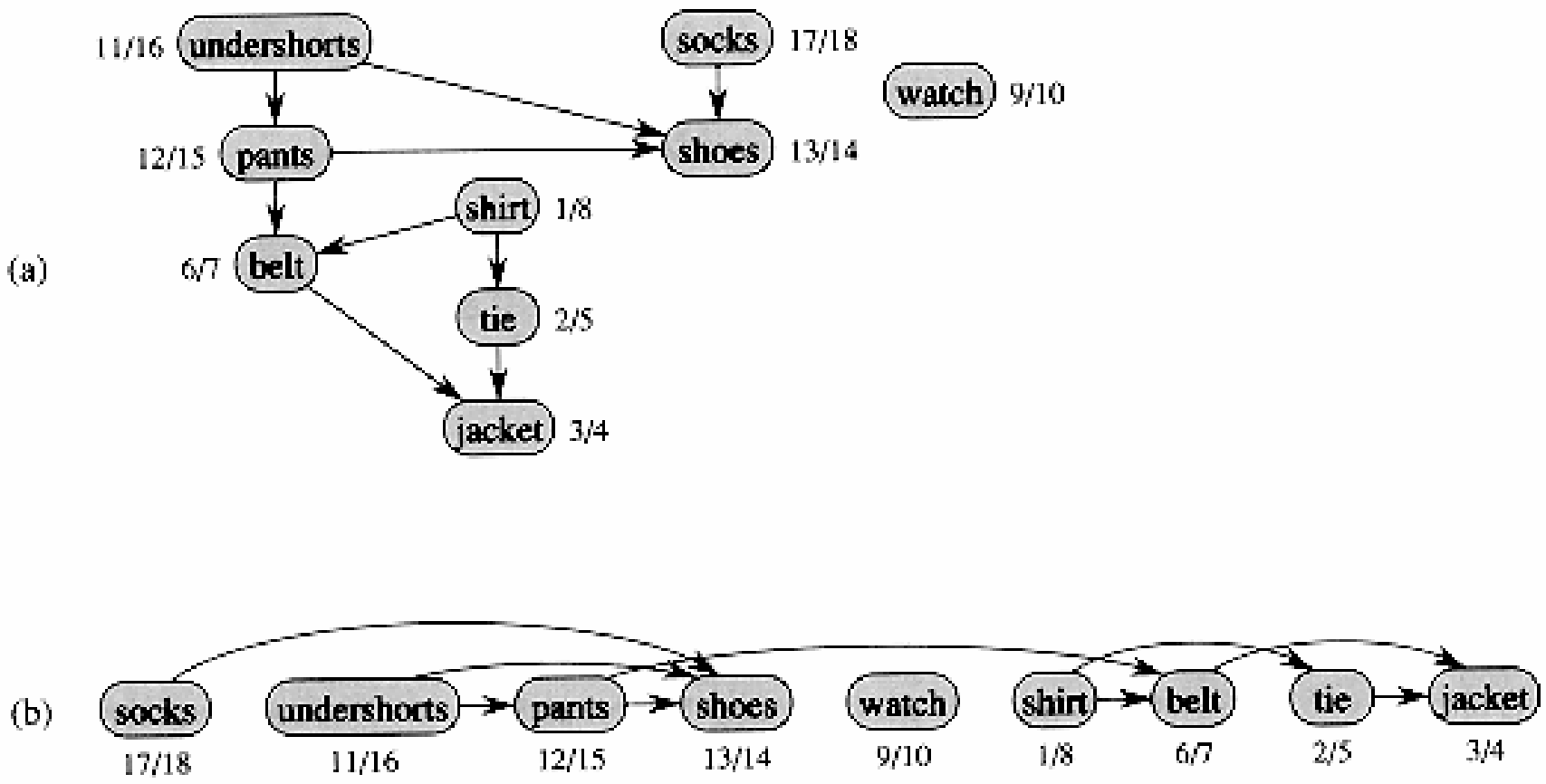
例4. 连通图标号

- 任给一个M条边的连通图，给它的每条边不重复遗漏地用1…M的整数编号，使得任意与顶点u关联的所有边的最大公约数为1。

分析

- 对图作一次DFS，按照访问的顺序依次给每条边编号就可以了。第一个顶点出发的边有一个编号为1；而其他所有顶点出发的边都有两个的编号相邻（想一想，为什么？），因此对于每个顶点*u*，*u*连出的所有边的编号的最大公约数为1。用BFS无法到达这个要求，因为它无法保证其他顶点出发有两个相邻编号

应用1、拓扑顺序


(a)
11/16
undershorts
socks
17/18
watch
9/10
12/15
pants
shoes
13/14
shirt
1/8
6/7
belt
tie
2/5
jacket
3/4
(b)
socks
17/18
undershorts
11/16
pants
12/15
shoes
13/14
watch
9/10
shirt
1/8
belt
6/7
tie
2/5
jacket
3/4

拓扑排序算法

- 对图G使用DFS算法, 每次把一个结点变黑的同时加到链表首部
- **定理1:** 有向图是DAG当且仅当没有B边
  - 如果有B边, 有环(易证)
  - 如果有环c, 考虑其中第一个被发现的结点v, 环中v的上一个结点为u, 则沿环的路径v→u是白色路径, 有白色路径定理, u是v的后代, 因此(u, v)是B边
- **定理2:** 该算法正确的得到了一个拓扑顺序

例4. 无向圈判定

- 设计一个算法判断无向图是否含圈. 时间复杂度应为O(V), 和E无关

例5. 字母排序

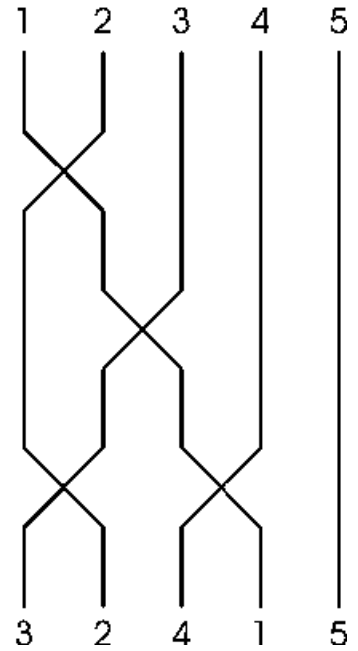
- 给出含若干字母的串的排序，求这些字母的顺序。
- 例: XWY ZX ZXY ZXW YWWX
- 输出: XZYW

分析

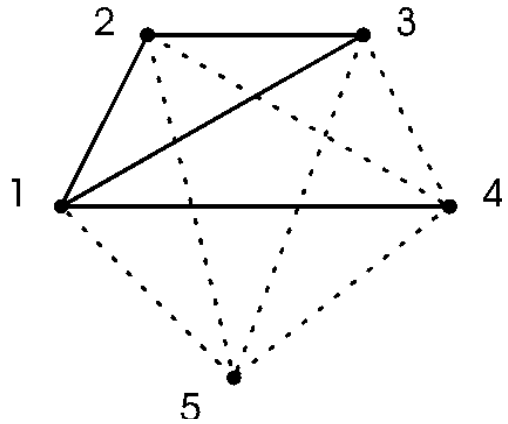
- 根据两个串的顺序, 可以得到拓扑顺序
- XWY<ZX<ZXY<ZXW<YWWX
- 根据第1,3,4个小于号,有 X<Z, Y<W, Z<Y

例6. 导线的编号

- 网线从机房连接到办公室
- 在机房，所有网线从左到右编号为1,2,3,…,*N*
- 给出每两条线是否交叉的信息，请计算办公室内从左到右各条线的编号



(a)



(b)

三、欧拉回路

- 如下图, 每条边经过一次且仅一次的称为欧拉回路(euler cycle, euler circuit).

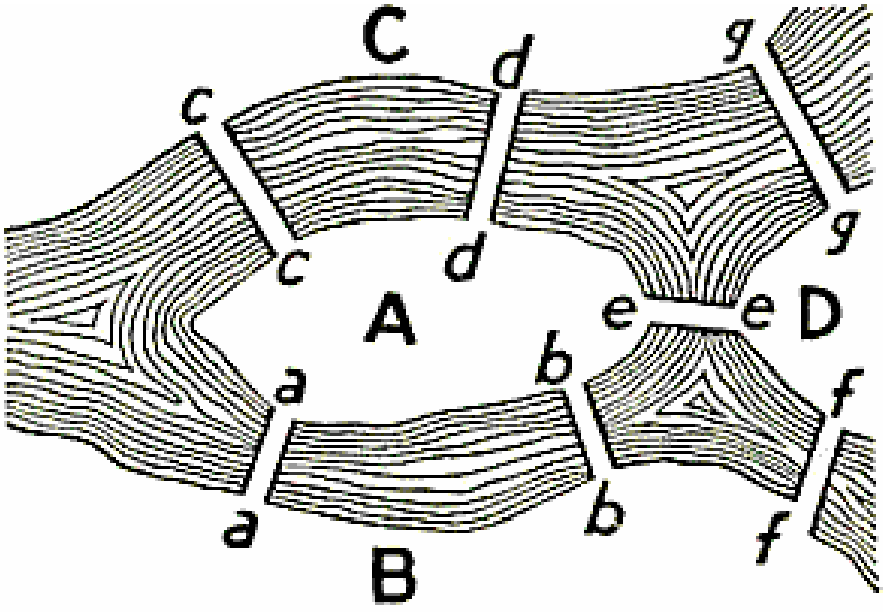
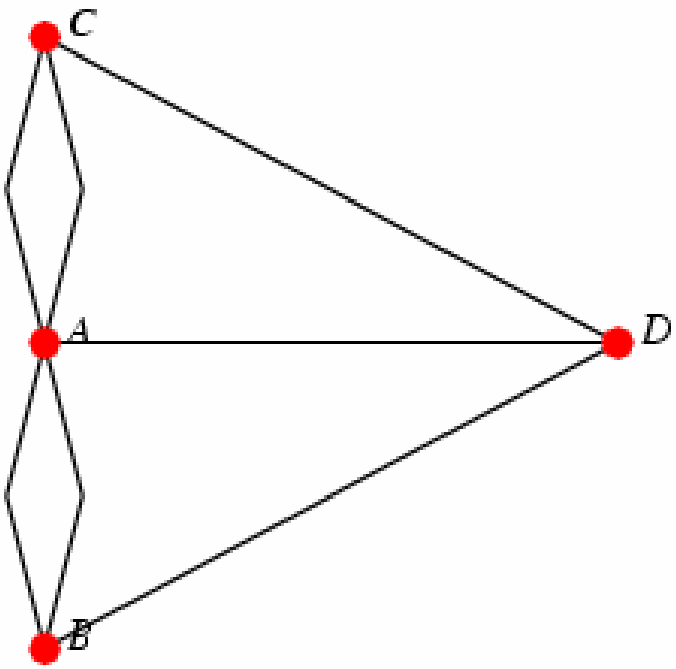
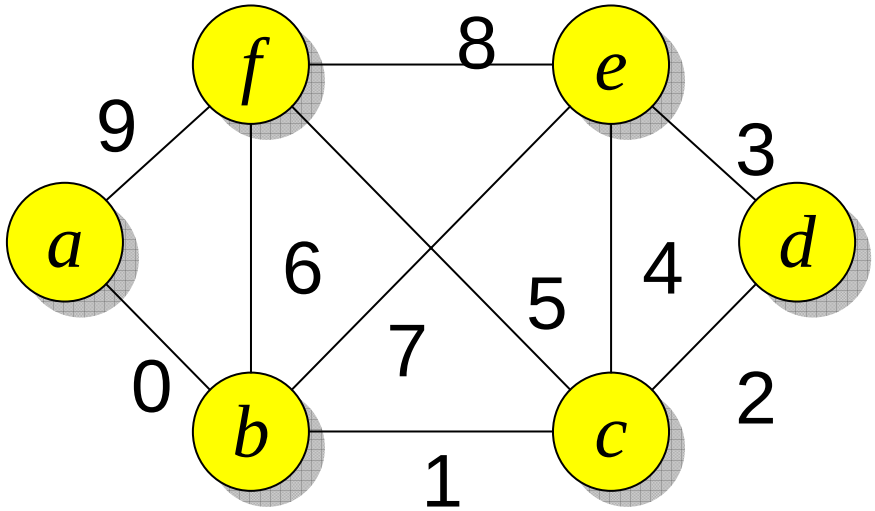


FIGURE 98. *Geographic Map: The Königsberg Bridges.*



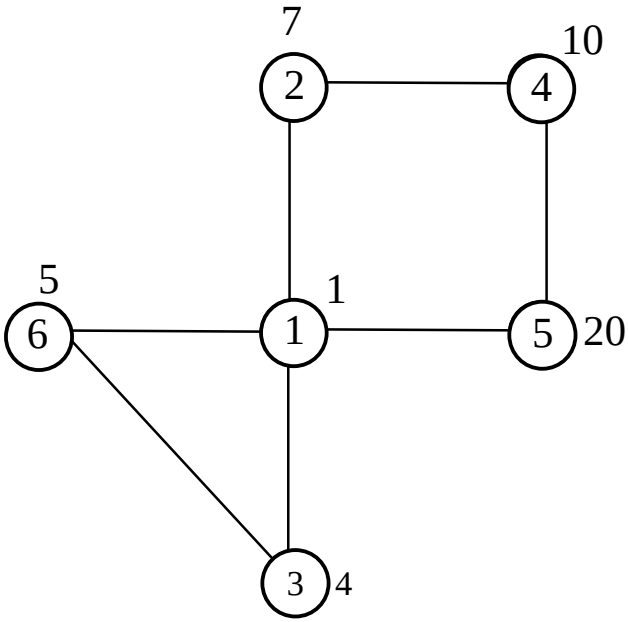
算法

- 算法：从一个点u出发DFS，每次标记边(u,v)和(v,u)，递归调用EulerRoute(v)，然后把(u,v)放到栈中
- 问题：递归边界是什么？在什么情况下u的所有关联边都被访问过了？考虑度数



例1. 邮递员

- 每个村子可能处于两条路的十子路口上，4条路的米子路口上，或者一条路的中央
- 每个村庄都有一个期望值W_i，如果该村子是邮递员经过的第K个不同的村子，如果$K \leqslant W_i$，则村子付给邮递员W_i-k元钱，否则邮递员付给村子$K-W_i$元。每经过一条不同的路，邮局会付给邮递员1元钱
- 邮局规定每条路（总共m条）都至少经过一次。邮递员应该怎样走，才能挣最多的钱呢



例2. 特殊的中国邮路

- 给出加权无向图, 最多两个奇度点
- 求经过每条边至少一次的最短回路

分析

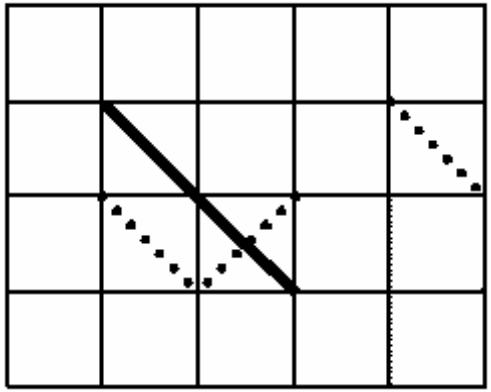
- 倍边法: 确定两个奇度点之间的某条路径, 路径上的每条边经过两次(相当于把这些边复制了一份)
- 倍边后存在欧拉回路, 因此关键是倍边的开销应尽量小
- 用dijkstra求出两奇点的最短路

例3. 字典序最小欧拉回路

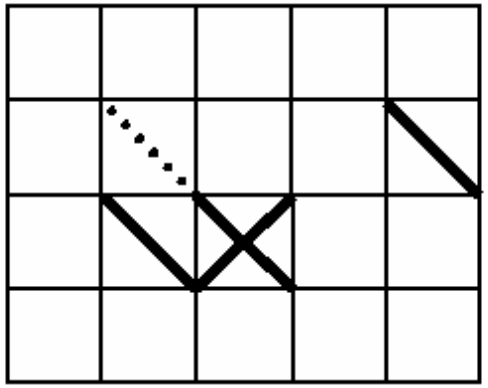
- 求字典序最小的欧拉回路

例4. 绣花

- 交替地在布的两面穿线. 布是一个$n \times m$的网格, 线只能在网格的顶点处才能从布的一面穿到另一面。每一段线都覆盖一个单位网格的两条对角线之一, 而在绣的过程中，一针中连续的两段线必须分处布的两面



- 给出布两面的图案（实线代表有线，虚线代表背面有线）,最少需要几针才能绣出来？一针是指针不离开布的一次绣花过程。例如图(b)的图案最少需要4针。



分析

- 抽象成图. 正面的线：正边; 背面的线：负边; 有边相连：连通块
- 每个连通块分别求. 对于某个顶点I, |正边数-负边数|=K>0时以该顶点为开始或结束的针数>=K, 可以恰好为K针. 所有K值加起来，除以2（每一针有两个端点）即可。

例5. 原始基因

- 一个抽象化原始基因的编码是一串连续的整数$K=(a_1, a_2, \ldots, a_n)$，一个原始基因的特征是一个有序的整数对$(l, r)$，它在基因编码中连续出现，即在原始基因中存在$i$，使$a_i=l$, $a_{i+1}=r$。在基因序列中不会出现形如(p,p)的特征
- 编程计算包含这些特征的最短原始基因

分析

- 把每个整数看作图中的顶点，每一个有序整数对看作一条有向边。问题转化为求在一个有向图中加最少的边，使图中存在欧拉道路（或回路）。可以知道：若一个图中已经有欧拉回路，则不需要加边；否则可以先加边使图中存在欧拉回路，然后再任意去掉一条边，求可以得到一条欧拉道路。以下只考虑求欧拉回路。

单连通分支

- 先考虑一个不存在欧拉回路的只有一个连通分支的有向图。刚才已经讲过，一个有向图存在欧拉回路的充要条件是图中只有一个连通分支且每一点的出度等于入度，因此需要加上边，使每个点的出度等于入度，这样图中就会有一条欧拉道路。具体的做法是求出每一点入度出度差的绝对值的和*m*，*m*/2就是形成欧拉回路所需要加的边数

多连通分支

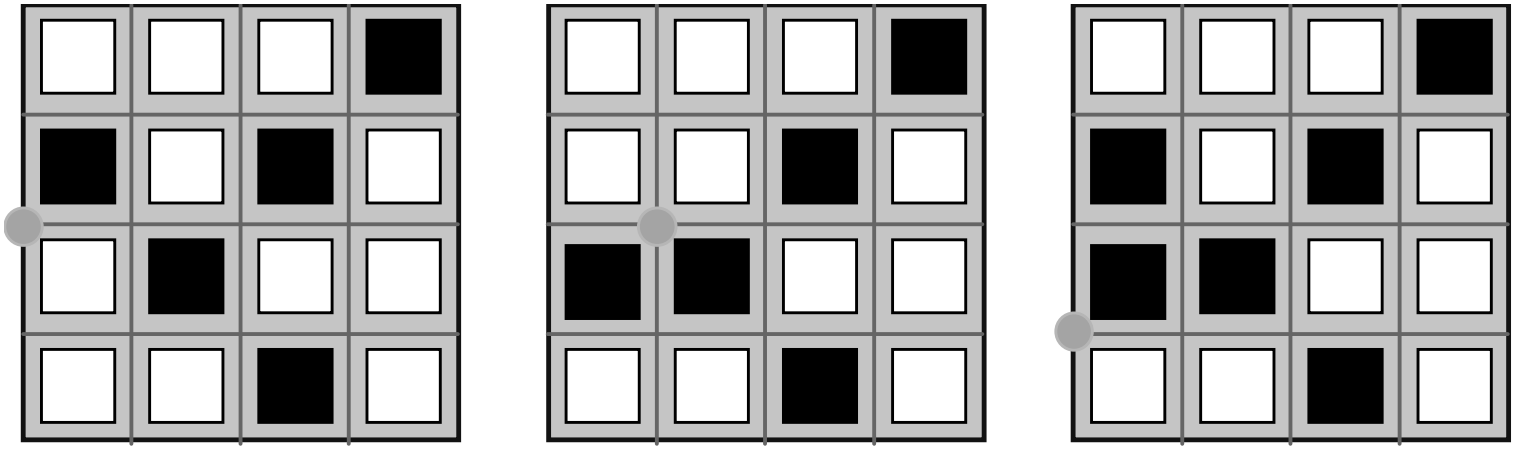
- 图中可能存在不止一个连通分支，这就需要将所有连通分支连起来。如果某一个连通分支中有一个点曾经加过一条边，则可以令这条边的另一个顶点在其他连通分支中，这样可以将这个连通分支于其他的连通分支连接起来。如此，只要这个连通分支中原先不存在欧拉回路，就可以通过加边的方法使它存在连通分支，同时将它和其他连通分支连接起来；反之，如果某个连通分支中没有一个点曾增加过边，则需要另加一条边以使这个连通分支与其他连通分支连接起来

例6. 超级翻转

- 有一个$N \times N$的网格，每一“格”上有一个可以翻转的方块，方块的两面分别涂成黑、白两种颜色。另外，有一个沿着网格线活动的东西——不妨称之为“动子”。初始时，每个方块随机地被翻成黑或白色，“动子”停在网格线的某个顶点上。例如图(a)就是一个4×4的网格的一种可能的开局情况
- 动子在网格线上运动时，从一个顶点A到相邻的另一个顶点B之后，以网格线AB为边的两个或一个网格上的方块就会翻转——白变黑，黑变白。例如图(a)的“动子”向右移动一步之后变成图(b)，向下移动一步之后变成图(c)

例6. 超级翻转

- 问题是：给定一个初始状态和一个目标状态，求“动子”的一种运动轨迹，可以把初始状态翻转成目标状态，最后“动子”停在哪里是无所谓的。

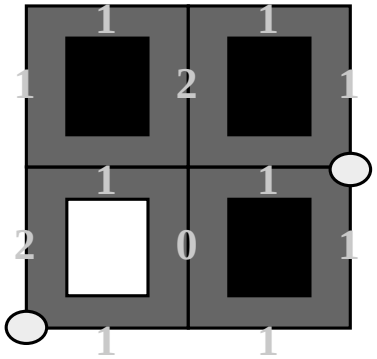


(a) (b) (c)

次数分析

- 考虑每条边经过的次数
- 一个格的四周有4条边，这4条边每经过一次，此格的状态就改变一次。那么如果这4条边经过了奇数次，则这个格子将会改变状态，否则状态不变。反过来说：若一个格子的初末状态不同，则它周围的边必经历了奇数次，否则为偶数次。
- 设动子起点和终点（动子最后停留的位置，通过枚举得到）为S和T，则对于某个顶点$V(V\neq S,T)$，V相邻的几条边必定总共经过了偶数次，因为动子每次到达这个点后必定会离开。当$S=T$时，S（T）相邻的几条边也必定总共经过了偶数次；当$S\neq T$时，S或T相邻的几条边必定总共经过了奇数次

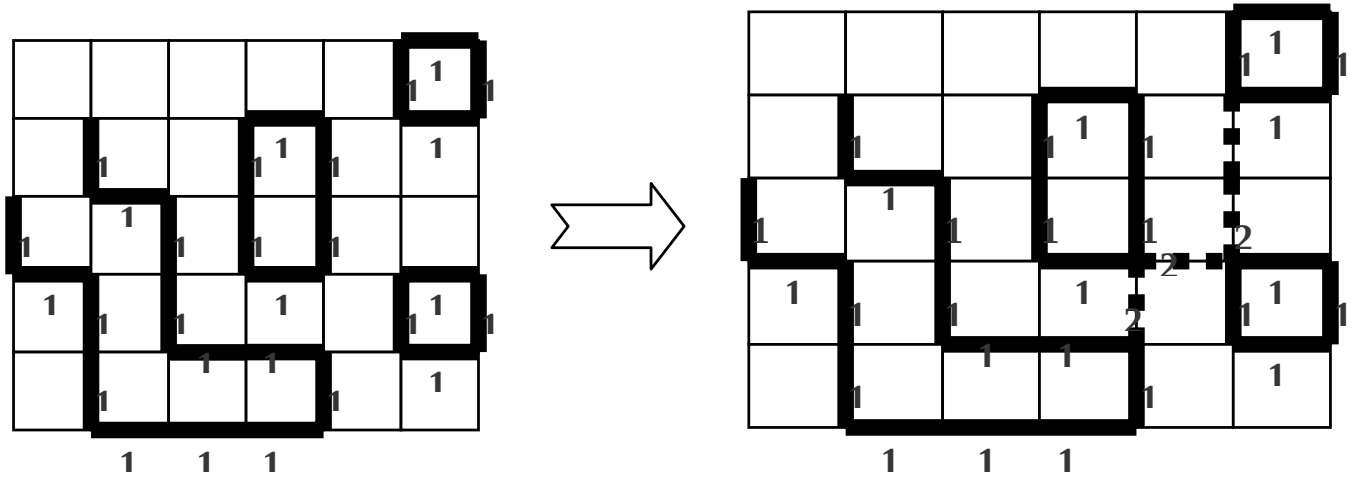
构造法



- 次数条件是必要的, 也是充分的
- 为了减少一些不必要的移动，我们可以将次数大于1的边不断的减少2，直至变为0或1，这样仍然能够满足上面两个条件。然后再把路径中的顺序考虑进来：若每条边的经过的次数已知了，怎么安排它们的顺序，得到一个可行的路线呢？如图
- 联想到欧拉路：如果所有要走的边都是连通的，那么存在一条*S*到*T*的欧拉路，这条路显然就是满足题意的。问题是，如果有多个连通块呢？其实这也很容易处理，可以通过添边，使得在仍然满足上述两个条件下，连通分量的数目减少

多连通块

- 方法一: 所有边的访问次数都增加2
- 方法二: 任选两个不同连通分块中的点 *a*,*b*，找一条*a*到*b*的路径，将路径上的边的次数都加2, 则所有次数大于0的边连通

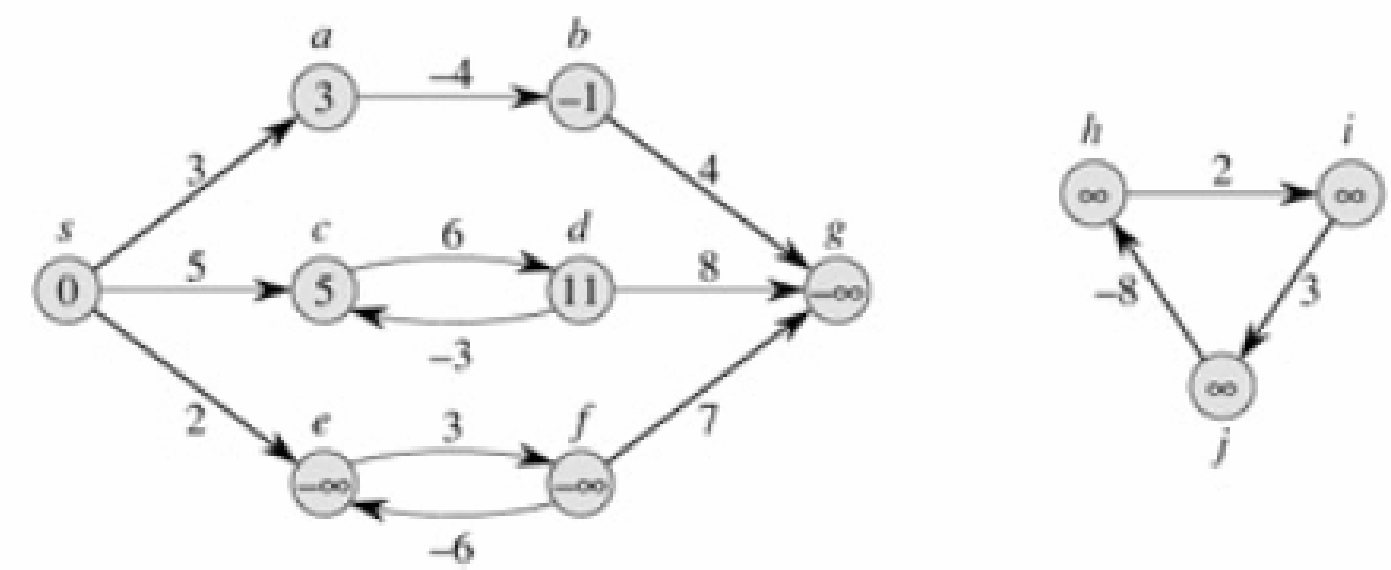


算法框架

- 步骤**1** 设X_i=0或1，表示某条边访问了的次数，若X_i=0则为偶数次，否则为奇数次。对于某个格子i，若它的初末状态不同，则它相邻的所有边的和mod 2 = 1，否则和mod 2=0。对于某个顶点i,如果($i \neq S$)and($i \neq T$)or($S=T$)那么所有它相邻的边的和mod 2=0,否则和mod 2 = 1；
- 步骤**2** 解出满足上面条件（边数有$2n(n+1)$，关系式$n \times n+(n+1) \times (n+1)=2n(n+1)+1$个，实际上就是一个$2n(n+1)$个未知数$2n(n+1)+1$个方程的模线性方程组）；
- 步骤**3** 合并连通块；
- 步骤**4** 求一条从S到T的欧拉路。

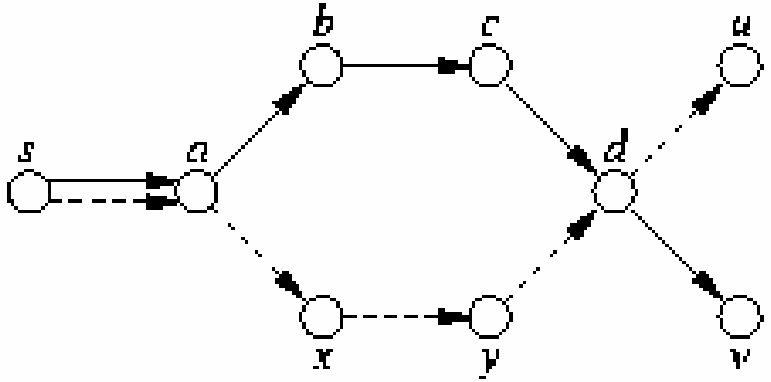
三、最短路问题

- 给加权图和一个起点s, 求s到所有点的最短路(SSSP)
- 最短路有环吗
  - 正环: 何必呢, 删除环则得到更短路
  - 负环: 无最短路, 因为可以沿负环兜圈子



最优性原理

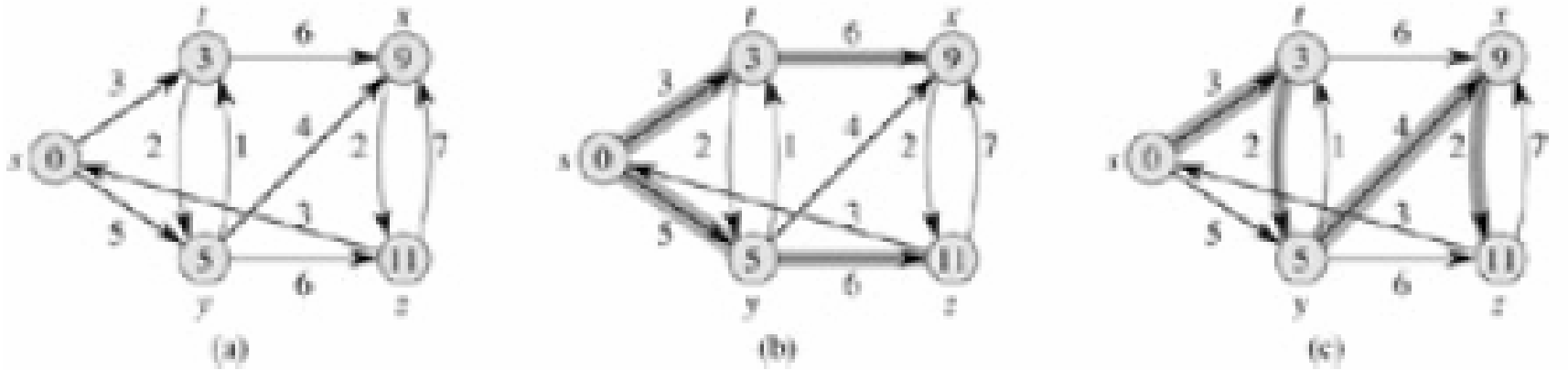
- 最优性原理: 若最短路u→v经过中间结点w, 则u→w和w→v的路径分别是u到w和w到v的最短路.
- 意义: 贪心、动态规划



If $s \to a \to b \to c \to d \to v$ and $s \to a \to x \to y \to d \to u$ are both shortest paths, then $s \to a \to b \to c \to d \to u$ is also a shortest path.

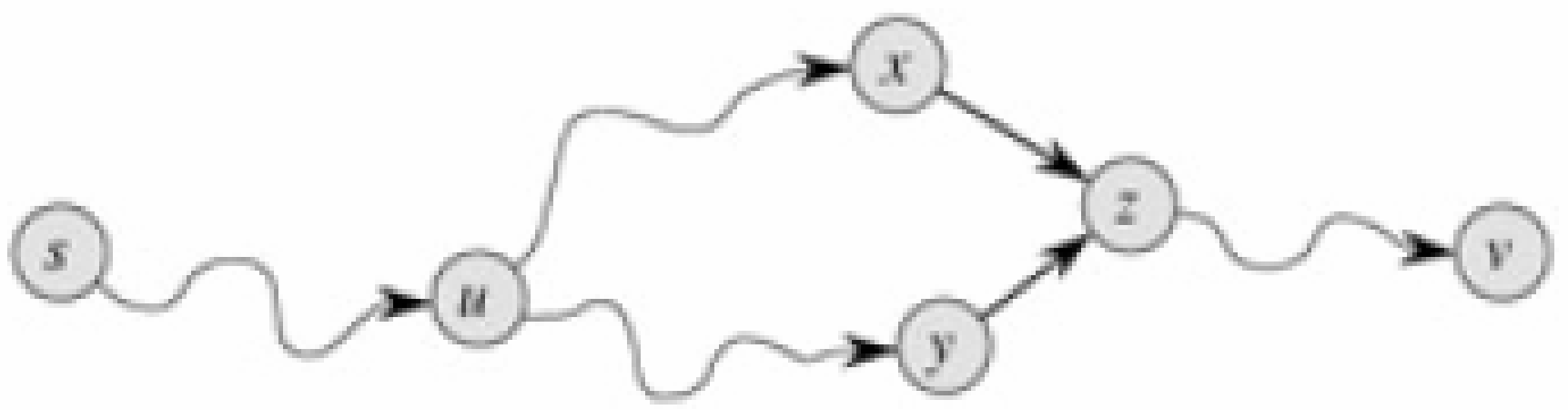
最短路的表示

- 最短路的表示
  - s到所有点的最短路不需要分别表示
  - 最短路树: 到每个点都沿着树上的唯一路径走
  - 实际代码: 记录每个点的父亲pred[u]即可
  - 输出最短路: 从终点沿着pred[u]递推回起点

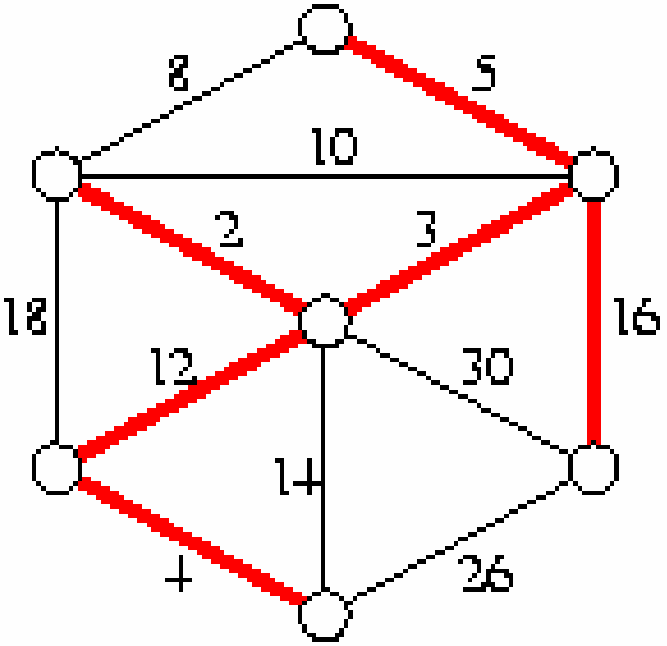


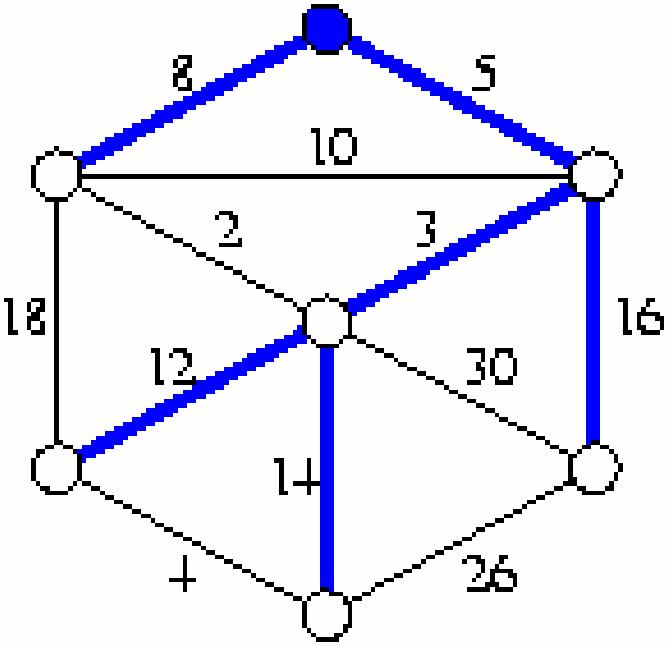
为什么单源最短路形成树?

- 考虑下图
- 如果u→z的路只取一条即可



最短路树和最小生成树



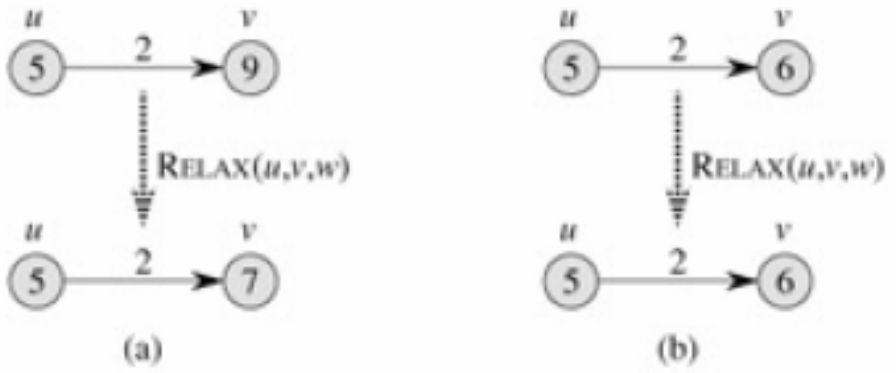


一般SSSP算法

- 临时最短路
  - 存在此路，即真实的最短路长度不大于此路长度
  - 但是有可能有更短的，所以*此路长度只是一个上界*
- 给定起点s，对于每个顶点v，定义
  - dist(v)为临时最短路树中s->v的长度
  - pred(v)为临时最短路树中s->v中v的前驱
  - 初始化: dist(s)=0, pred(s)=NULL,
  - 初始化: 所有其他dist(v)为无穷，pred(v)=NULL
- dist(v)称为点v的标号(label), 它是最短路的上界
- 基本想法: 让标号不断趋近, 最终达到最短路

一般SSSP算法

- 什么样的标号明显可以改进(趋近最短路)?
  - 一条边(u,v)被称为紧的(tense), 如果 dist(u)+w(u,v)<dist(v)
  - 可以松弛：dist(v)=dist(u)+w(u,v), pred(v)=u



- 结论
  - 存在紧的边，一定没有正确的求出最短路树
  - 不存在紧的边，一定正确的求出最短路树

一般SSSP算法的正确性

- (u,v)被称为紧的：dist(u)+w(u,v)<dist(v)
- 不存在紧边，一定求出最短路树
  - 即由pred表示出的路径上所有边权和等于dist(v)（归纳于松弛的次数）
  - 结束时对s到v的任意路s→v，dist(v)<=w(s→v)
    - 归纳于s→v所含边数，假设s→u-v（u=pred(v)）
    - dist(u)<=w(s→u)，两边加w(u,v)得：
    - dist(u)+w(u,v)<=w(s→v)。因为无紧边，所以
    - dist(v)<=dist(u)+w(u,v)<=w(s→v)

一般SSSP算法的结束条件

- 刚才已经证明
  - 结束时dist(v)和pred(v)相容
  - 若算法结束，则结果正确
- 算法何时能结束呢？
  - 含负圈（能到达的），则永不结束，因为在一次松弛以后，负圈上一定有紧边（反证）
  - 不含负圈，则一定结束，因为要么减少一个无穷dist值，要么让所有有限dist值之和至少减少一个“不太小的正值”。

一般SSSP算法

- 一般算法
  - 可以以任意顺序寻找紧边并松弛
  - 收敛时间没有保障
- 解决方案：把结点放到bag中，每次取一个出来检查
- 特殊bag：dijkstra(heap), bellman-ford(queue)

```
INITSSSP(s):
  dist(s) ← 0
  pred(s) ← NULL
  for all vertices v ≠ s
      dist(v) ← ∞
      pred(v) ← NULL
```

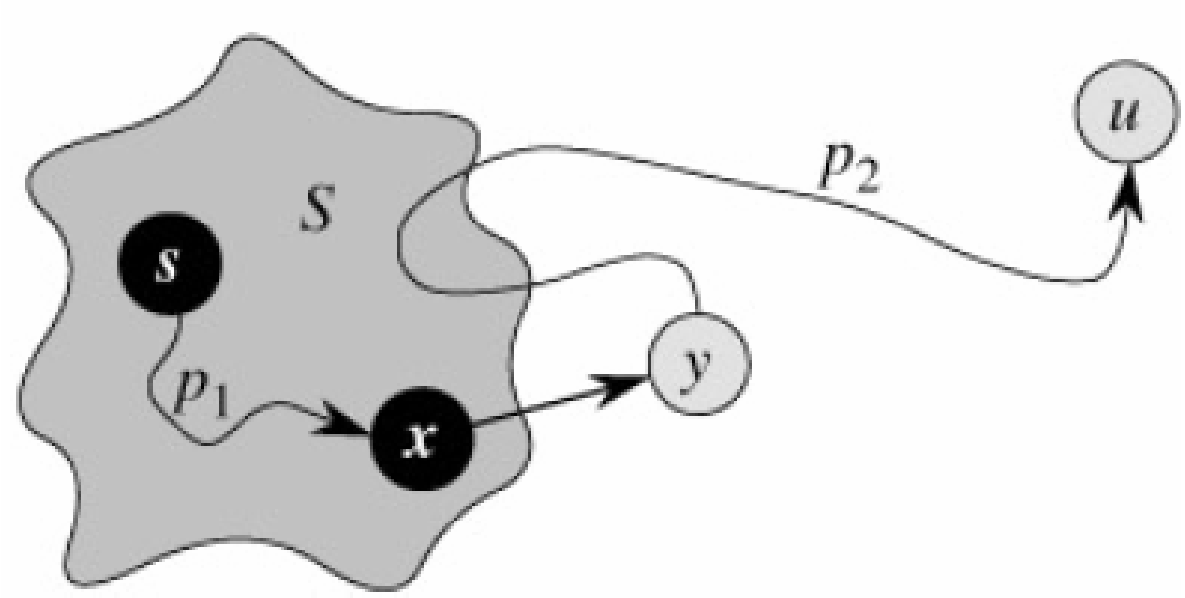
```
GENERICSSSP(s):
  INITSSSP(s)
  put s in the bag
  while the bag is not empty
      take u from the bag
      for all edges u → v
          if u → v is tense
              RELAX(u → v)
              put v in the bag
```

Dijkstra算法

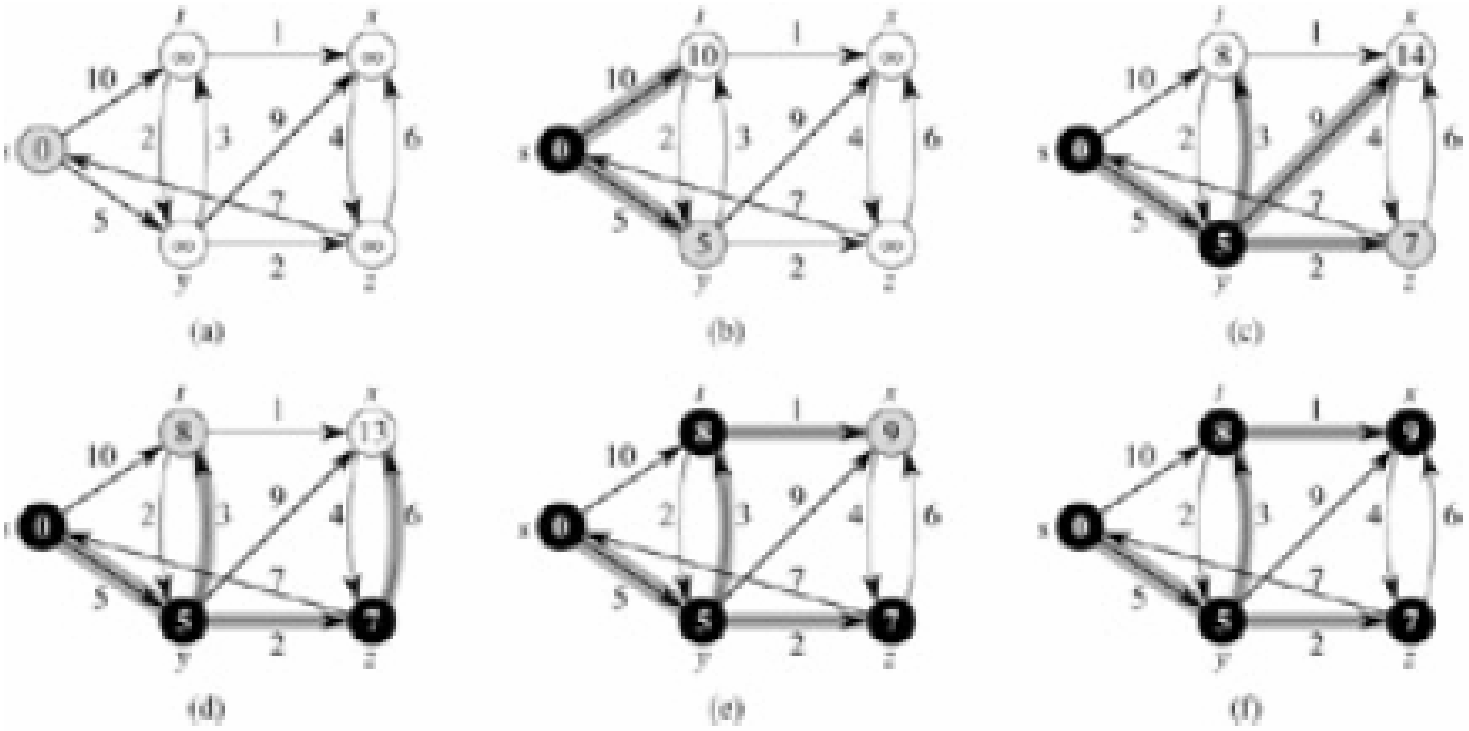
- E.W.Dijkstra. *A note on two problems in connection with graphs.* Num.Math.,1:269-271, 1959
- 原始是$O(n^2)$, 可以用各种形式的堆加速

Dijkstra算法

- 标号设定算法: 每次dist(v)最小的一个恰好等于它的最短值，予以固定
- 正确性证明 (注意为什么需要权*非负*)



```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S ← ∅
3  Q ← V[G]
4  while Q ≠ ∅
5       do u ← EXTRACT-MIN(Q)
6          S ← S □{u}
7          for each vertex v □ Adj[u]
8               do RELAX(u, v, w)
```



时间复杂度

- Dijkstra算法使用了一个优先队列
  - INSERT (line 3), 每个结点一次
  - EXTRACT-MIN, 每个结点一次
  - DECREASE-KEY (line 8, 在RELAX过程中), 一共|E|次
- 直接实现: $O(V^2)$
- 二项堆: O(ElogV)
- Fibonacci堆: O(E+VlogV)

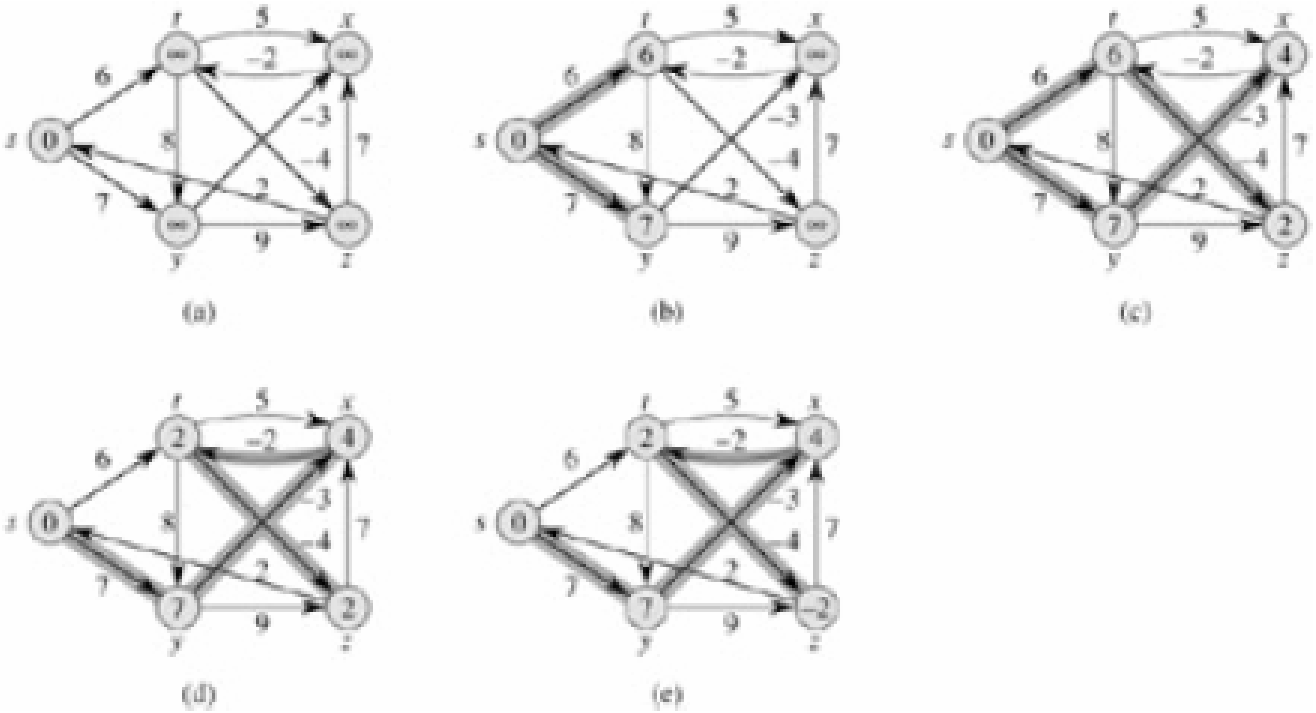
例1. 路的最小公倍数

- 给出一个带权无向图G
  - 边权为1…1 000的整数
  - 对于v_0到v_1的任意一条简单路p, 定义$s(p)$为p上所有边权的最大公约数
  - 考虑v_0到v_1的所有路$p1,p2,\ldots$, 求所有$s(p1),s(p2),\ldots$的最小公倍数

bellman-ford算法

- Ford 1956, Bellman 1958, Moore 1959.
- 如有最短路，则每个顶点最多经过一次
  - 这条路不超过*n*-1条边
  - ***长度为k的路由长度为k-1的路增加一条边得到***
  - 由最优性原理, 只考虑长度为1…k-1的最短路
  - 算法梗概: 每次迭代依次松弛每条边
- 时间复杂度
  - O(Dm)，v为迭代次数(v<=n-1)
  - 完全图边权在[0, 1]中均匀分布, 很大概率$D=O(\log^2 n)$
  - 若某次迭代没进行成功松弛, 可立即停止
  - 可用dijkstra得到初始dist

```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i ← 1 to |V[G]| - 1
3        do for each edge (u, v) □ E[G]
4              do RELAX(u, v, w)
5  for each edge (u, v) □ E[G]
6        do if d[v] > d[u] + w(u, v)
7              then return FALSE
8  return TRUE
```



Yen的修改算法

- 把G中的边(v_i, v_j)分为两类:
  - f边: i < j
  - b边: i > j
- 每次迭代先从v_1遍历到$v_{|V|}$, 松弛f边, 再从$v_{|V|}$遍历回v1, 松弛b边
- 则最多只需要|V|/2次(取上整)迭代, 但不降低时间复杂度

例2.套汇

- 套汇: 不停的换外汇, 换回来后赚钱(假设无手续费)
- 如三次兑换: 1*0.7*9.5*0.16=1.064>1
- 给出任意两种货币兑换的比率, 寻找一个方案(必须换回来), 使得比率大于1.01
- 如果有多种方案, 求长度最短的

分析

- 本题可以抽象为: 给出正权完全有向图, 找权乘积大于1.01的最短回路
- 类似Bellman-ford算法: 第k次迭代求出源点到所有点不超过k条边的最长路, 设为$d_i[j]$, 则若$d_i[j] * weight[j, i] > 1.01$, 立即输出.

差分约束系统

- 线性规划(linear programming, LP): 给m*n矩阵A、m维向量b和n维向量c, 求出x为向量使得Ax<=b, 且sum{$c_i x_i$}最小
- 可行性问题(feasibility problem): 只需要任意找出一组满足Ax<=b的解向量x
- 差分约束系统(system of difference constraints): A的每行恰好一个1和一个-1, 其他元素都是0. 相当于关于n个变量的m个差分约束, 每个约束都形如x_j-x_i<=b_k, 其中1<=i,j<=n, 1<=k<=m.

差分约束系统举例

- 左边的可行性问题等价于右边的差分约束系统

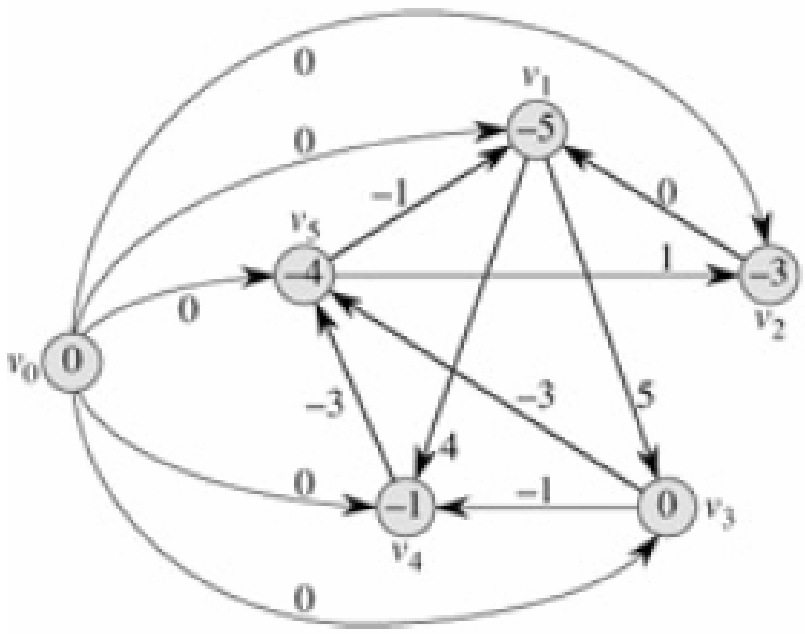
$$\begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} \leq \begin{pmatrix} 0 \\ -1 \\ 1 \\ 5 \\ 4 \\ -1 \\ -3 \\ -3 \end{pmatrix}$$

$$\begin{aligned}
&(24.3)\ x_1 - x_2 \leq 0, \\
&(24.4)\ x_1 - x_5 \leq -1, \\
&(24.5)\ x_2 - x_5 \leq 1, \\
&(24.6)\ x_3 - x_1 \leq 5, \\
&(24.7)\ x_4 - x_1 \leq 4, \\
&(24.8)\ x_4 - x_3 \leq -1, \\
&(24.9)\ x_5 - x_3 \leq -3, \\
&(24.10)\ x_5 - x_4 \leq -3
\end{aligned}$$

基本思路

- 定理: 给定差分约束系统的一组解, 给每个变量加上一个常数d, 将得到另外一组解
- 约束图: 结点是变量, 一个约束对应一条弧, 若有弧(u, v), 则得到x_u后, 有$x_v <= x_u + w(u,v)$

$$
\begin{aligned}
&(24.3)\ x_1 - x_2 \le 0,\\
&(24.4)\ x_1 - x_5 \le -1,\\
&(24.5)\ x_2 - x_5 \le 1,\\
&(24.6)\ x_3 - x_1 \le 5,\\
&(24.7)\ x_4 - x_1 \le 4,\\
&(24.8)\ x_4 - x_3 \le -1,\\
&(24.9)\ x_5 - x_3 \le -3,\\
&(24.10)\ x_5 - x_4 \le -3
\end{aligned}
$$



算法

- 定理: 如果约束图没有负圈, 则可取x_u为起点v_0到u的最短路长; 若约束图有负圈, 差分约束系统无解.
- 正确性证明
  - 无负圈: 由松弛条件可证明每个约束得到满足
  - 有负圈: 把负圈上的约束条件叠加将得到一个矛盾不等式
- 算法步骤
  - 构图, 得到n+1个结点m+n条边
  - 运行bellman-ford, 时间 $O(n^2+mn)$

练习

- 如何修改bellman-ford算法, 使得将它应用在差分约束系统的求解中, 使得当m比n小时, 时间复杂度可以由$O(n^2+mn)$降为$O(mn)$
- 如果差分约束中存在等式约束, 即$x_i-x_j=b_k$, 如何求解? $x_i<=b_k$ 或者$-x_i<=b_k$呢?
- 如果限定$x_i<=0$, 证明bellman-ford算法得到的可行解让x_i总和最大化
- 证明bellman-ford算法得到的可行解让$max\{x_i\}-min\{x_i\}$最小
- 修改算法使得当b取实数时可以得到整数解. 如果给定变量子集需要取整数呢?

例3. 出纳员的雇佣

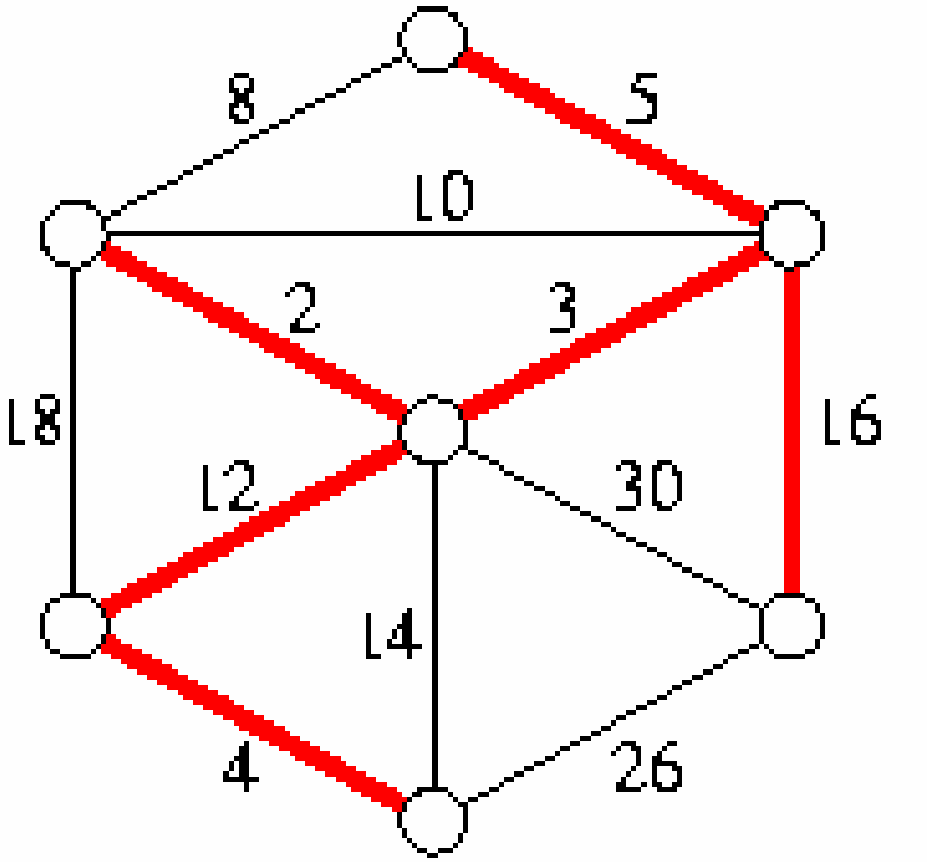
- 24小时营业的超市需要一批出纳员来满足它的需求, 每天的不同时段需要不同数目的出纳员
- 给出每小时需要出纳员的最少数R_0, …, R_{23}
  - R(0)表示从午夜到午夜1:00需要出纳员的最少数目，
  - R(1)表示上午1:00到2:00之间需要的…
  - 每一天，这些数据都是相同的
- 有N人申请这项工作, 如果第i个申请者被录用，他将从t_i刻开始连续工作8小时
- 计算为满足上述限制需要雇佣的最少出纳员数目
- i时刻可以有比对应的R_i更多的出纳员在工作

分析

- 前i小时的雇佣总数：s[i] (规定s[-1] = 0)
- 第i小时需要的出纳员：r[i]
- 第i小时申请的人数：t[i]
- 取(i,j)满足i = (j+8) mod 24, 则有不等式
  - 0 <= s[i] – s[i-1] <= t[i]
  - s[23] – s[-1] = sum
  - i > j时 s[i] – s[j] >= r[i]
  - I < j时 s[i] – s[j] >= r[i] – sum
- 当sum固定时, 此不等式组是差分约束系统
- 可以枚举sum, 也可以二分

四、最小生成树问题

- 连接每个点的连通图（一定是树）
- 权和尽量小

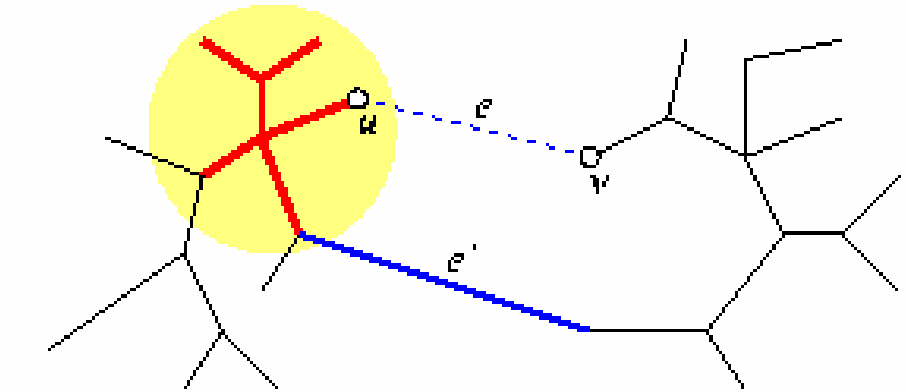


一般最小生成树算法

- 前提: 无相等边
- 维护生成森林F
- *e*为无用边, 若*e*的两个端点在F的同一个分量中, 但*e*不在F中
- 对于F的每一个分量
  - 从它出去(即恰好有一个端点在此分量内)的最小边为安全边
  - 不同的分量可以有相同的安全边
- 结论: MST含有所有安全边, 不含无用边.

一般最小生成树算法

- 结论: MST含有所有安全边, 不含无用边.
- 证明
  - 假设有一个分量(黄色), 它的安全边e不在T内. 则u到v有唯一路径, 它经过e', 它恰好有一个端点在黄色分量中. 由于e是安全边, w(e)<w'(e), 用e替换e'得到更小的T', 矛盾
  - 加入无用边将形成环



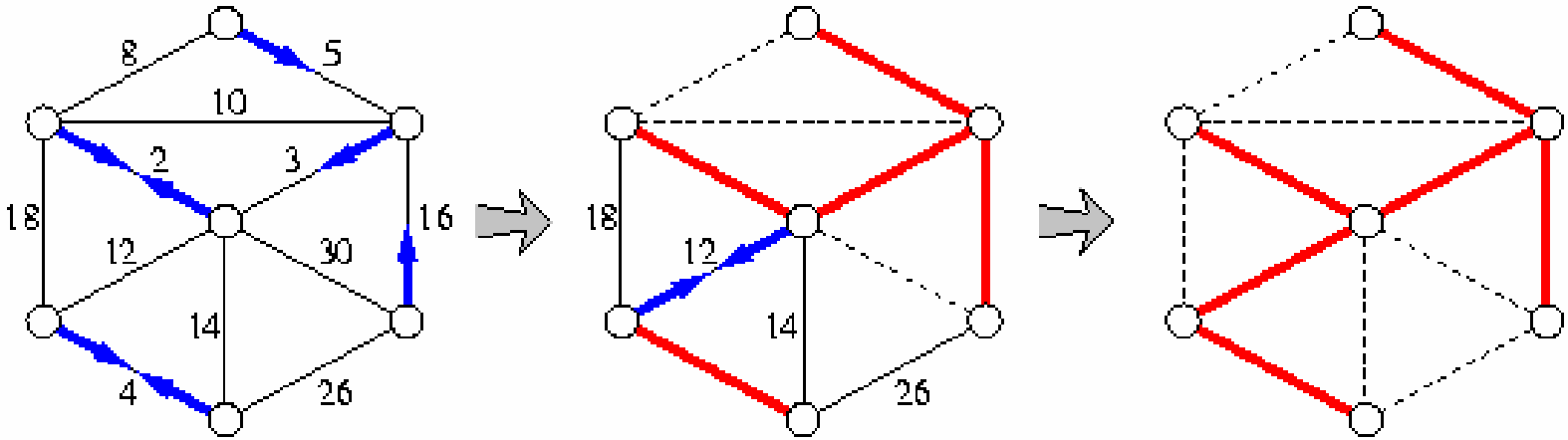
练习

- 证明最小边属于某棵MST中
- 对于某环上的最大边e, 证明原图中删除e后的MST和原来的相同
- 在图中减小一个边的权, 求新的MST
  - 情况一: e在原MST中
  - 情况二: e不在原MST中

Boruvka算法

- 由Boruvka于1926年提出(早于图论产生!)

BORŮVKA: Add all the safe edges and recurse.



Borůvka's algorithm run on the example graph. Thick edges are in F.
Arrows point along each component's safe edge. Dashed edges are useless.

Boruvka算法

- 每个分量设置'leader', 用DFS在m时间内求出
- 检查每条边一次以修正各分量的安全边权
- 第i次迭代每个分量大小至少为2^i
- 最多logV次迭代, 总O(ElogV)

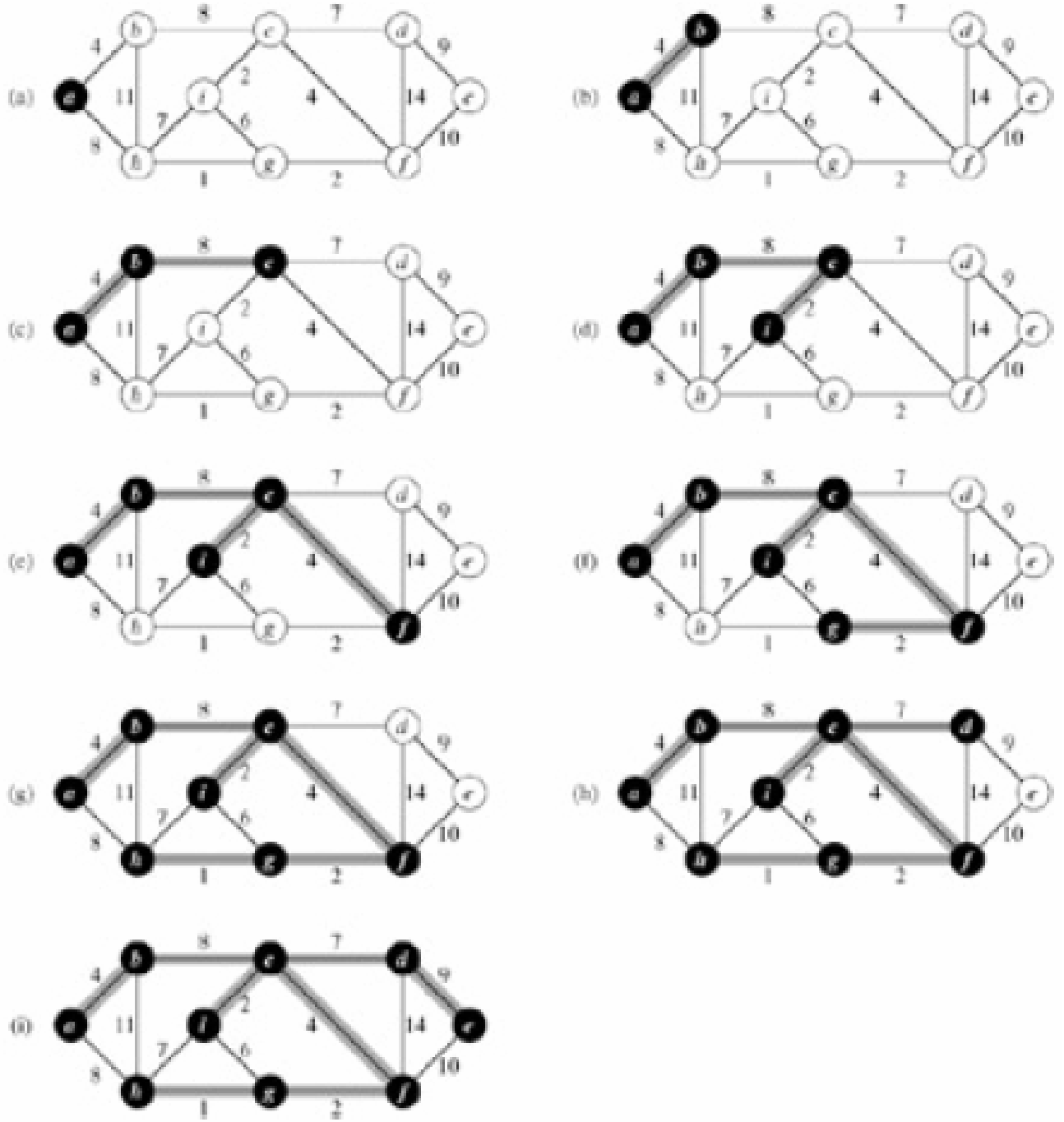
```
BORŮVKA(V, E):
  F = (V, ∅)
  while F has more than one component
      choose leaders using DFS
      FINDSAFEEDGES(V, E)
      for each leader v̄
          add safe(v̄) to F
```

```
FINDSAFEEDGES(V, E):
  for each leader v̄
      safe(v̄) ← ∞
  for each edge (u, v) ∈ E
      ū ← leader(u)
      v̄ ← leader(v)
      if ū ≠ v̄
          if w(u, v) < w(safe(ū))
              safe(ū) ← (u, v)
          if w(u, v) < w(safe(v̄))
              safe(v̄) ← (u, v)
```

Prim算法

- Prim提出, 但事实上Jarnik于1930于年更早提出
- 只关心一棵树T, 每次加入T的安全边
- 用堆保存到每个顶点(而非边)的安全边
- Insert/ExtractMin调用V次, DecreaseKey调用E次
- 二叉堆: O((E+V)logV), Fibonacci堆: O(E+VlogV)

```
MST-PRIM(G, w, r)
 1  for each u □ V [G]
 2       do key[u] ← ∞
 3          π[u] ← NIL
 4  key[r] ← 0
 5   Q ← V [G]
 6   while Q ≠ Ø
 7       do u ← EXTRACT-MIN(Q)
 8          for each v □ Adj[u]
 9              do if v □ Q and w(u, v) < key[v]
10                    then π[v] ← u
11                         key[v] ← w(u, v)
```


(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)
(i)
a
b
c
d
e
f
g
h
i
4
8
8
7
9
11
7
2
4
6
14
10
1
2

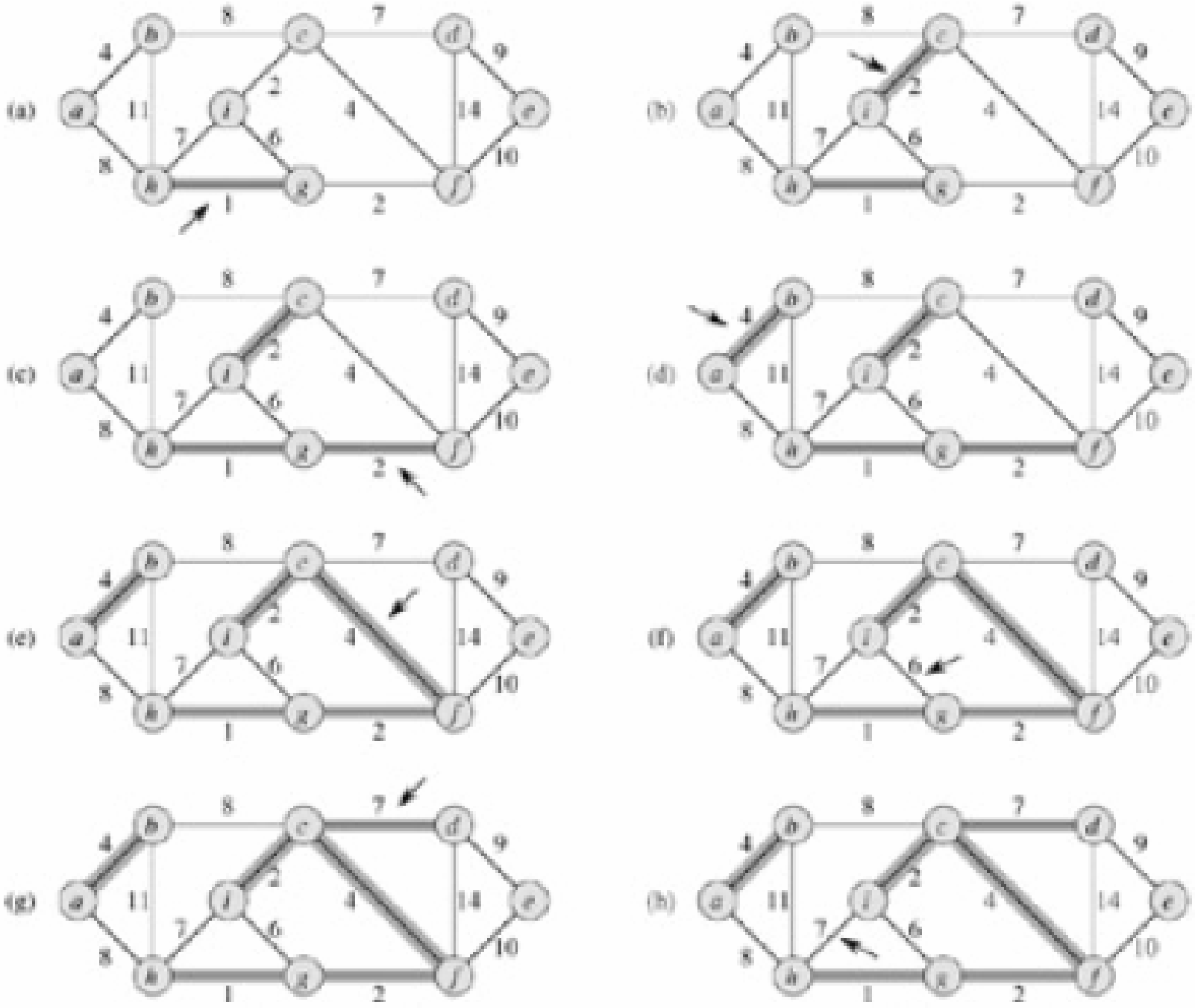
练习

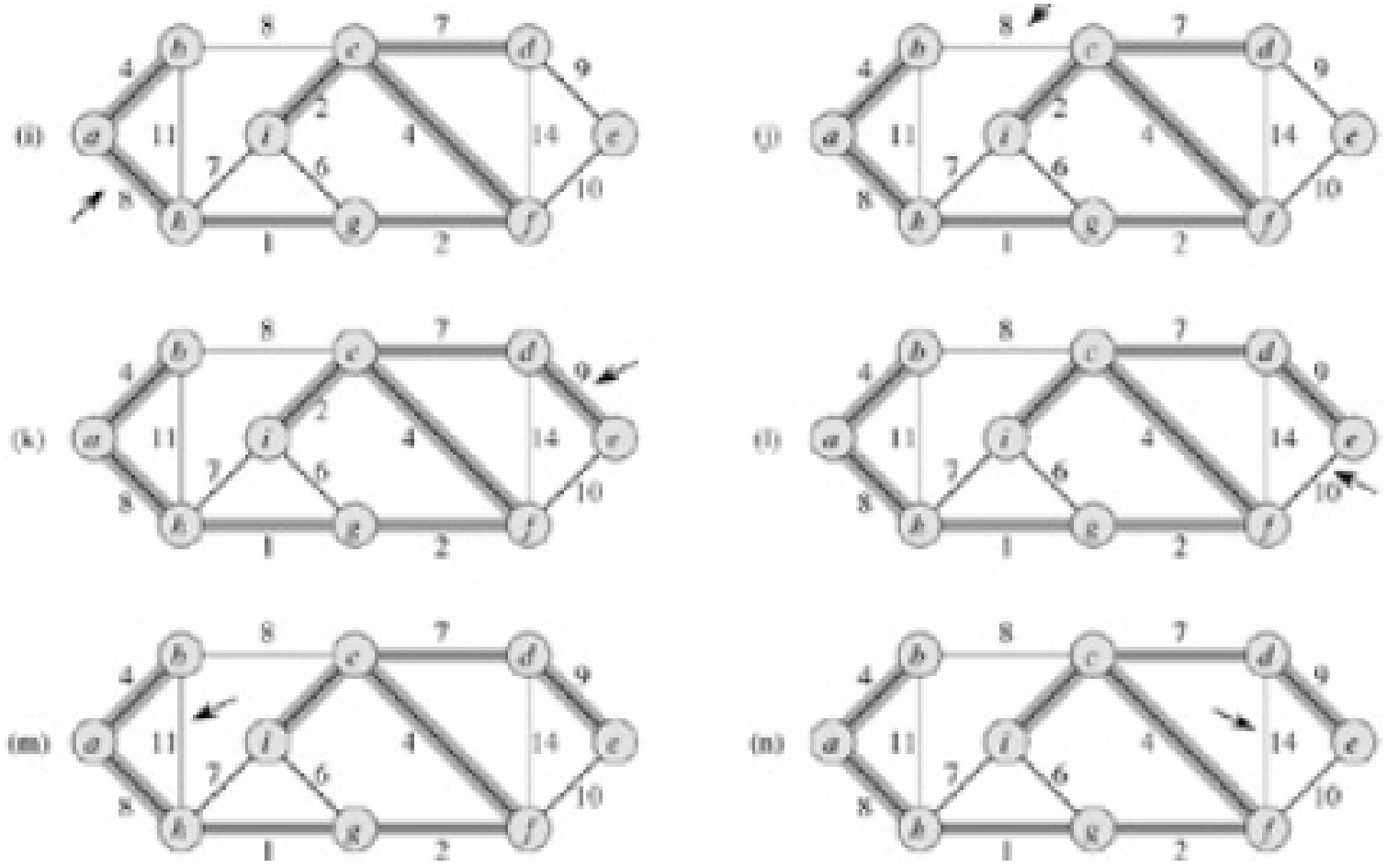
- 给定邻接矩阵, 设计一个$O(V^2)$算法
- 对于稀疏图中, 用k次Boruvka迭代可以加速MST计算. 如何选取k, 使得k次迭代以后调用Prim算法的时间复杂度变为O(EloglogV)

Kruskal算法

- Kruskal, 1956年提出
- 把边从小到大排序，每次填加一个安全边
- 如何知道边是否安全? 并查集, 每次约O(1)
- 总O(ElogE + E α (V))

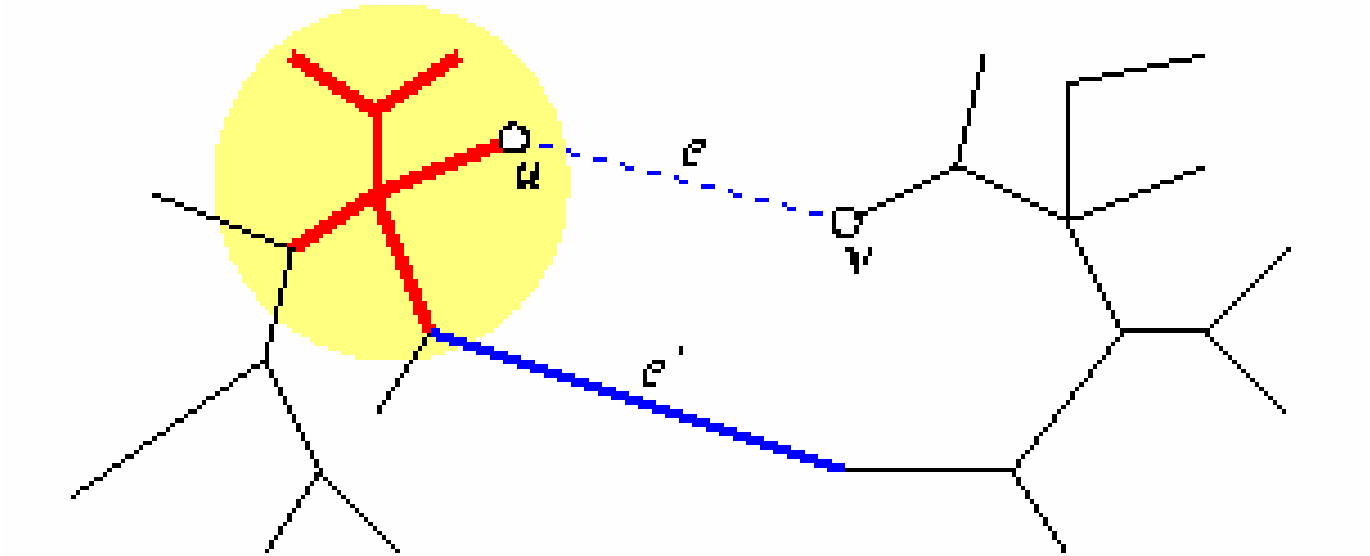
```
MST-KRUSKAL(G, w)
1  A ← ∅
2  for each vertex v □ V[G]
3       do MAKE-SET(v)
4  sort the edges of E into nondecreasing order by weight w
5  for each edge (u, v) □ E, taken in nondecreasing order by weight
6       do if FIND-SET(u) ≠ FIND-SET(v)
7             then A ← A □ {(u, v)}
8                  UNION(u, v)
9  return A
```


(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)
a
b
c
d
e
f
g
h
i
4
8
8
7
9
11
7
6
2
4
14
10
1
2


(i)
(j)
(k)
(l)
(m)
(n)
a
b
c
d
e
f
g
h
i
4
8
11
8
7
2
4
14
9
7
6
10
1
2

MST唯一性判定

- 考虑一般最小生成树算法
- 每一个分量出发安全边唯一, 不特殊处理, 否则
  - 若同时添加会形成环, 一定不唯一
  - 若同时添加不会形成环, 类似正确性证明, 即:
    - 假设某边e不在T中, 对应的e'一定比e大而不可能相等



MST唯一性判定

- 时间复杂度
  - Boruvka: 不变(只在和安全边比较时修改)
  - Prim: 不变(只在判断顶点标号时修改)
  - Kruskal: 不变(相等的边时特殊处理)
- 另一种思路: 最小=第二小

瓶颈生成树问题

- 边的最大值最小的生成树成为G的瓶颈生成树(bottleneck spanning tree), 把其中最大的边称为它的权. 显然, MST是一个瓶颈生成树, 但是瓶颈生成树可以算得更快
- 判定问题
  - 问题: 给一个实数b, 如何在线性时间内判定是否存在一个瓶颈生成树, 它的权不超过b?
  - 方法: 忽略超过b的边, 判断图是否连通

次小生成树

- T为图G的一个生成树，对于非树边a和树边b，插入边a并删除边b的操作记为(+a,-b)
- 如果T+a-b仍然是一个生成树，称(+a,-b)是T的一个*可行交换*
- 设T为图G的一个生成树，由T进行一次可行交换得到的新的生成树集合称为T的*邻集*

问题描述

- 给一个无向加权连通图G, 假设各边权都不相同, 则MST唯一
- 如果把所有生成树按权值从小到大排序, 第二小的称为次小生成树, 它不一定唯一
- **定理:** 次小生成树在最小生成树的邻集中
- 关键问题: 快速计算*邻集中的最小树*
- 枚举加边(u, v), 则形成唯一的环, 它由u→v在树中的*唯一路径*, 加上(u, v)边构成. 由于在邻集中最小的一棵为次小生成树, 只需考虑*删除环中最大边*

算法

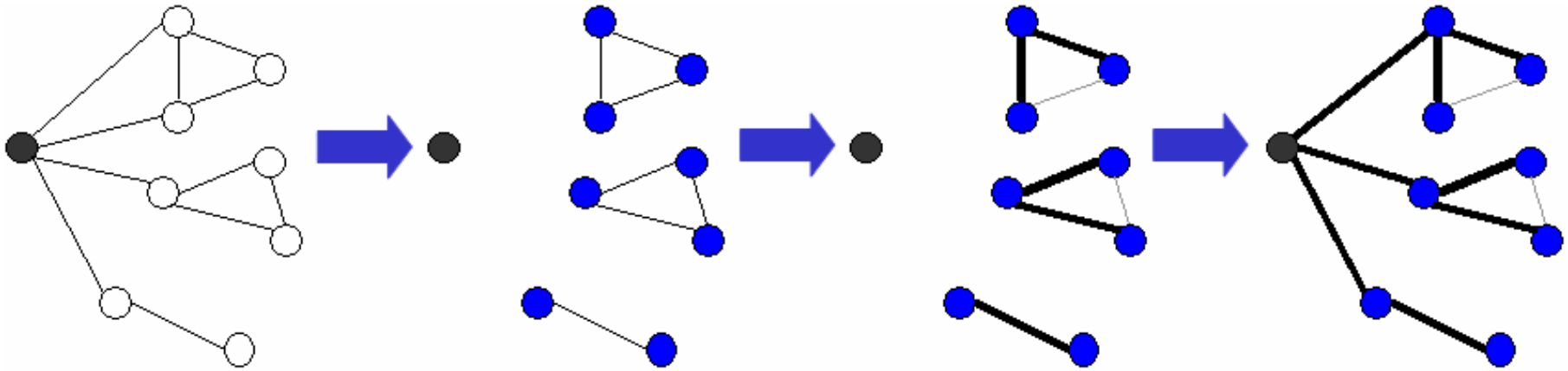
- 由于在邻集中最小的一棵为次小生成树, 只需考虑删除环中最大边, 记u→v的唯一路径中最大边为max(u, v), 则应考虑可行交换(+(u,v), -max(u,v))
- 动态规划
  - 固定u, 设d[v]为max(u,v)的权值
  - 边界: $d[v_0]$ 为负无穷
  - 转移方程为d[v] = max{d[f[v]], w(f[v],v)}, f[v]为v的父亲
- 每次预处理时间复杂度为$O(V^2)$, 枚举加边为O(E)
- 总时间复杂度: $O(V^2)$

最小度限制生成树

- 给定某结点v_0的度为k, 求在此情况下的最小生成树
- 定理: k+1度限制生成树在k度限制生成树的邻集中
- 基本算法:
  - 先计算让v_0度最小(设为p)的最小生成树
  - 若k<p, 无解, 否则
  - 不断增加v_0的度, 即: 依次求出p+1, p+2, … k度限制生成树

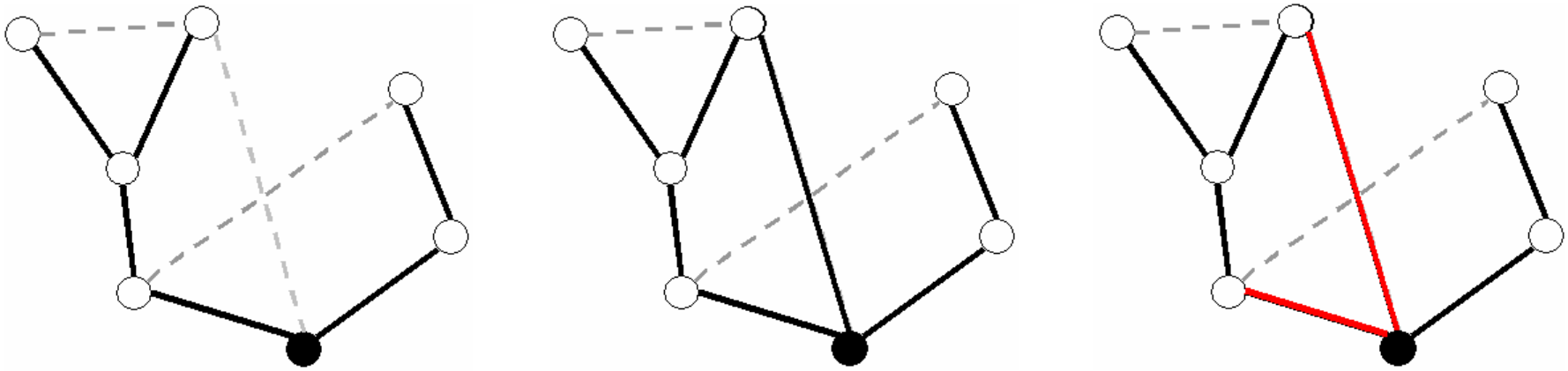
步骤一: 求最小度数

- V_0最小可能度数p和对应的最小生成树
  - 删除v_0, 每个连通分量分别求MST
  - 每个连通分量选一条连往v_0最小的边, 加入树中



步骤二: 增加度数

- 为了使v_0的度增加，枚举的可行交换中必须有一条边与v_0关联
  - 枚举与v_0关联的边, 加入后形成环
  - 需要删除环上最大的边, 才可能得到最小树
  - 与v_0关联的边是不能删除的, 否则度数不会增加



找环上可删除的最大边

- 核心操作: 枚举和v_0关联的边, 每次删除环上不和v_0关联的最大边
- 枚举加边(v_0, v)后, 环只有一个, 它是原树中v_0到v的唯一路径, 加上(v_0, v)形成的. 因此需要删除的边是***路径v_0→v上不和v_0关联的最长边***
- 动态规划
  - 设d[v]为路径v_0→v上不和v_0关联的最长边
  - 边界: d[v_0]和d[v']为负无穷(v'为v_0的邻居)
  - 转移方程为d[v] = max{d[f[v]], w(f[v], v)}, f[v]为v的父亲

算法总结

求出让v_0度最小的最小生成树

while v_0的度数<k do

预处理, 用动态规划计算d数组

枚举和v_0关联且不在树上的边$e = (v_0, v)$

加入e

删除环上不与v_0关联的最大边

如果这是第一个解或者此树更优, 更新最优解

end

- 显然, 每步的时间复杂度为O(V), 因此总时间复杂度为O(MST + kV)