《算法设计与分析》课程笔记

2019年6月4日 17:40

stable matching:

a. 思想：当有人没有匹配，按照他的偏好列表，找到当前没有人要的，或者女方爱他胜过当前finance的
b. 伪码：

c. 证明
    i. 复杂度：最多经过N^2次的while循环，这个算法就可以结束
        1) 证明：
            a) 观察发现：
                i) 所有女性一旦匹配了，就不可能不匹配
                ii) 男性挑选时，是按照偏好程度逐渐选择的，一旦被某个女生拒绝，就不可能再向她求婚（因为女性只会获得更好的，如果你之前被拒绝了，那么之后也必定被拒绝）
            b) 所以，每一次while循环时，男性都会向一个女性求婚（proposal），追求的女性的匹配情况都会至少向前移动一个位置。一共有n^2种可能
    ii. 正确性：
        1) 一定能够匹配成功
            a) 证明：假设M没有被匹配，那么必然还有W也没有被匹配，因为男女人数相等。这时，M propose to W,然后由于W还没有匹配，所以她服从匹配
        2) 没有不稳定的匹配
            a) 假设有不稳定的匹配，W - M，X - Y，W和Y都更喜欢对方而非自己当前的伴侣。
            则Y当时求婚时，会先向W求婚的，而非X。然后M是向W求婚的人中，W最喜欢的。所以，W更喜欢M而非Y。矛盾。
    iii. 实现（implementation）
        1) 有一个queue，保存着当前单身的男性，每一次从中整一个
        2) 用两个二维数组保存男女的优先列表
        3) 用一个大小为N的数组保存男性x求婚的次数（这样就可以快速地找到他要求婚的对象）
    iv. 特性：
        1) 这样的匹配中，每一个男性的匹配都是所有稳定匹配中，他能获得的最好的匹配
            a) 证明：
                引入概念：称在某一个稳定匹配中的对象为valid-partner
                i) 假设在G-S的S*匹配并不是man-optimal，假设S*中第一个被自己最好的valid-partner拒绝的人是Y，被A拒绝，拒绝Y时A和Z在一起。但是在另一个稳定匹配S中，A-Y，Z-B。
                ii) 假设在S* 中，A-Z。由于Y是第一个被valid-partner拒绝的人，所以Z当时还没有被valid-partner拒绝。但是Z当时没有选择B，所以Z更喜欢A而不是B。
                iii) 这样，Z更喜欢A而不是B，A更喜欢Z而不是Y，则S的匹配不稳定
        2) Woman-passive
            a) 假如不是，假如在S*中A和Y匹配，但是在S中A-Z，A更喜欢Y。
            b) 由于是man-optimal，所以Y最喜欢A，而A也更喜欢Y而非Z。所以S不稳定。

分析（analysis）：

1. 一些指数时间复杂度的算法由于极端条件很少见，所以还是广泛使用
2. asymptotic order of growth
    i. 下界：Ω
    ii. tight bound：Θ
    iii. 小心误用O notation：f(x)=O(x)=g(x),但是：f(x) != g(x)
    iv. 满足一般的传递性（transitivity）：f(x)=O(g(x)), g(x)=O(h(x)),则有：f(x)=O(h(x)).Ω和Θ类似。

Graph：

1. 相关定义：
   i. simple path：path中所有的点都只出现一次
2. BFS：分层实现。L0={s}　　L1=所有与S相邻的点　　L2=所有点中，既不属于L1又不属于L0，并且与L1中的某点共边的点　　Li+1=所有点中，既不属于之前的点，又与Li中的某点共边的点
   i. 时间复杂度O（e+v）
      1) 证明：每条边都要遍历一次，每个点都要恰好入队一次，所以复杂度是O（e+v）
   ii. 应用：画图时填充颜色。先确定出发点，然后使用BFS遍历边界线内的所有像素点
3. bipartite（二分图）
   i. 定义：无向图中每一条边都恰好有一个A颜色的端点和一个B颜色的端点
   ii. 性质：当且仅当图中没有奇数长度的环路时，这个图是二分图
      1) 证明：
         a) 引理1：如果一个图是二分图，则它不能有奇数长度的环路
            i) 证明：哪怕是一个奇数长度的环路本身也不能是二分的，更别说图了
         b) 引理2：我们用BFS对图进行分层，那么：
            i) 如果有一条边连接了同一层中的两个点，则这个图不是二分的
            ii) 否则它就是二分的
            iii) 证明：
               One. 对于i)，我们可以从BFS的出发点S出发，通过BFS找到它到被这条边连接的两个点的path，然后易知这三个点构成的loop的长度是奇数，然后通过引理1可知这个图不是二分的
               Two. 对于ii)，我们可以每一层染一个颜色，然后就二分了
4. 有向图
   i. 相关定义：
      1) mutually reachable 互相可达。两个点互相主键有path
      2) strongly connected 强连通。图中所有点互相可达
         a) 引理：如果有一个点s可以到达其他所有点，而其他所有点都可以到达S，则说明这个图是强连通图
         b) O（m+n）时间内强连通的判断：
            i) 先从S出发，BFS遍历所有点，说明S可以到达任意点
            ii) 再将这个图的所有边反转，然后仍然从S出发，如果能遍历整个图，则说明然后在原图中所有点都可以到达S
            iii) 证毕
   ii. DAG（directed acyclic graph）中，一定有一个点没有入度
      1) 证明：
         a) 反证法。假设每一个点都有入度，则任取图中某一点v。找到指向v的边的另一个节点，比如u，再找backward from u的另一个节点。不断这样。因为节点的数量有限，而这个操作可以无限进行，所以我们必定会多次遇到某个点，即图中有环
      2) DAG中必定有topological order。每次去掉一个没有入度的点以及与它相关的边，这样就可以得到一个拓扑序了。
      3) 证明获得拓扑序的时间复杂度是O（m+n）
         a) 一开始遍历所有节点获得入度为0的节点，装进S中。同时用一个数组count[]记录每一个节点的入度。
         b) 初始化：BFS一遍，O（m+n）
         c) 然后每一次都从S中取出一个节点并输出，将它的子节点的入度都减1.如果为0了，就装进S中。
         d) 这样，每条边都要遍历恰好一遍，每个节点也都是只装进S中一遍。O（m+n）。
5. Greedy algorithm
   i. 例子1：一段时间里最多能完成的任务数量
      1) 算法：每次都找最先结束的任务
      2) 证明：反证法。设新算法从i开始在任务选择上不同。发现如果不找最先结束的那个任务，并不能比老算法表现更好。
   ii. 例子2：输入各个任务的DDL和耗时，求为了使各个任务拖延时间之和最小应该怎么安排任务？
      1) 每次都挑选DDL最早的。
      2) 证明：反证法。
   iii. 例子3：cache替换
      1) 当request没有满足时替换cache不比随时随地替换差

6. MST（minimum spaning tree）
   i. 都使用了greedy algorithm
      1) Kruskal

            a) 使用uni-find set
        2) Prim
            a) 本质选择是当前割中的最小边
        3) Reverse-Delete algorithm：一直删除当前边权最小的边，直到图不连通
    ii. 与割（cut）有关的性质：
        1) 任意节点的子集的割中最小的边必定被MST包含
            a) 证明：如果不包含，则假设这个MST是T，我们将这条边加入T中构成环，然后将对应的那条边移除，发现T依旧连通，而且总边权值更小了。
    iii. 与环有关的性质：
        1) 图中的某个环中最大边不包含于MST
            a) 证明：如果包含，则假设这个MST是T，我们将这条最小边加入T中构成环，然后将对应的那条边移除，发现T依旧连通，而且总边权值更小了。
        2) 环的任意割的边数为偶数
    iv. 如果出现一样大的边权，则perturb它们，给它们加一点干扰，比如再比较下标值lexicographically

7. clustering
    i. Clustering of Maximum Spacing：将图的点分成k个团（cluster），使各个团之间距离（spacing）的最小值最大
    ii. 两个团之间的距离的定义为：来自这两个团中各一个点的距离的最小值
    iii. 算法：将MST去掉k-1个最大的边
        1) 使用Kruskal算法，算到还有k-1条边时就不算了

8. Huffman tree
    i. 相关定义：
        1) average bits per letter： abbr->ABL
        2) full：每一个中间节点都有两个子节点
    ii. 建树方法：只在叶节点上有label
    iii. 性质：最优前缀码树必须是full
        1) 证明：反证法。如果不是，通过调节子节点位置，可以让它更优
    iv. Shannon-Fano tree
        1) 让每个节点左右子树的总频率相等，递归式的计算
        2) top-down
        3) 非最优的prefix code
    v. 哈夫曼树核心构建方法：
        1) 让频率最低的两个节点成为sibling（同一个父母节点），然后将它们绑在一起成为一个节点，参与新一轮的top-2最低频率节点评选。Recursively。
        2) 也是贪心算法
        3) bottom-up
        4) 伪码：
            a) Huffman（S）{

```
    if（|S|==2）
        return S
    else
        取S中频率最低的两个节点x,y
        令fz=fx+fy
        S‘=S
        将x,y从S‘中去除
        把z加入S‘中
        T‘=Huffman（S‘）
        T=T‘中z节点下加上x,y这两个子节点
        return T
}
```

        5) 时间复杂度：
            a) T（n）=T（n-1）+O（n）
            b) 所以O(n^2)
            c) 时间复杂度优化：使用优先队列优化最低频率节点查找：
               T（n）=T（n-1）+O（log（n））

所以：O（nlog（n））

6) 正确性：
   a) 据观察，出现频率最低的项的level应该最低。所以最优前缀码应该让出现频率最低的两个节点成为sibling。
   b) 引理：ABL（T）=fz+ABL（T‘）
      i) 证明：
         ABL（T）=Sum（节点depth*频率）
         =（x,y的频率之和）*（z的深度+1）+其他点的（节点depth*频率）之和
         =fz+（fz）*(z的深度)++其他点的（节点depth*频率）之和
         =fz+ABL（T‘）
   c) 哈夫曼树的ABL最小
      i) 证明
         使用数学归纳法。
         One. 当n=2时，显然哈夫曼树最优
         Two. 假设有z=x+y代替的哈夫曼树T‘最优
         Three. 如果哈夫曼树不是最优的，有另一种F树更好。
            然后由引理可知，ABL（T）=fz+ABL（T‘），
            ABL（F）=fz+ABL（F‘）
            所以T也是最优的
         Four. 证毕

9. divide-＆-conquer
   i. 基本思想：将大问题转化为几个小问题
      分别解决这几个小问题
      然后将它们拼接在一起
   ii. 案例
      1) merge-sort
         a) T（n）=2T（n/2）+O（n）
         b) 引入概念
            i) inversion：逆序对
      2) 最近点对 Closest Pair of Points
         a) 伪码：

```
closes-pair(p1,p2,p3,...,pn){
        sort all points by x and get the middle line L            O(n logn)
        c1=closes-pair(left part)                                 T(n/2)
        c2=closes-pair(right part)                                T(n/2)
        c=min(c1, c2)                                             O(1)
        delete all points whose distance from L is more than c    O(n)
        sort left points by y                                     O(n logn)
        for every points                                          O(n)
                for its next 11 points
                        if their distance is less than c
                                update c
        return c
}
```

         b) T(n)=2T(n/2)+O(n logn)
         c) 优化：不要每次都从头开始（from scratch）sort。可以每一次都返回当前按照x和y sort的list
      3) 乘法multiplication
         a) 复数乘法的优化 Guass
            i) (a+bi)(c+di)=x+yi
               x=ac-bd   y=(a+b)(c+d)-ac-bd
               将四个乘法优化到了三个

10. dynamic programming
   i. 描述greedy algorithm、divide-and-conquer、DP的定义
      1) greedy algorithm：get the solution by partly optimazing step by step.
      2) divide-and-conquer:break up the proble into several sub-problem, and solve the recursively one by one. Then combine them into the answer of original problem.
      3) DP: Break up the problem into several overlap sub-problems and get the final answer by combine smaller condition into larger condition.
   ii. 例子：
      1) weighted interval scheduling

a) pseudocode:

```
input: n, v1,v2,....,vn,s1,...,sn,f1,...,fn
p(j) = 下标小于j并且与j相容的任务片段
以结束时间排序任务
计算所有的p[i]
将所有的m[i]设置为null
opt(i){
    if(m[i]不是null)
        return m[i]
    else
        m[i]=max{(opt(p[i]) +v[j] ), opt(i-1)};

}
```

2) 0-1背包问题

a) 质量为W的包，v1,...,vn, w1,...,wn

i) 这里采用了top-down方法，不用递归

伪码：

```
for w=0 to W
    m[0,w]=0
for i=1 to n
    for w=0 to w=W
        if wi>w
            m[i][w]=m[i-1][w]
        else
            m[i][w]=max{ m[i-1][w] , wi+ m[i-1][w-wi]}
return  m[n][M]
```

iii. $$OPT(j)=\begin{cases} 0 & \text{if } j=0 \\ \min\limits_{1\le i\le j} \{ e(i,j)+c+OPT(i-1) \} & \text{otherwise} \end{cases}$$

这样比较方便地表示对1-j的情况的综合比较

11. Bellman-Ford

i. for i=1 to |V|-1

```
遍历每一个节点
    如果上一轮遍历中有更新节点
        如果该节点在上一次遍历中更新了
            找到它所有的子节点，更新比较到起点的距离
    如果上一轮没有更新
        函数终止
```

ii. 时间复杂度：O（m+n）

12. 网络流

i. 相关定义：

1) capacity
   a) 从partition A中流出的边的权值之和
2) cut
   a) 把点集分成两部分 s,t
3) minimum cut
   a) capacity最小的cut
4) Residual edge
5) Residual Graph
6) augmenting path：增广路径
7) bottleneck capacity

ii. 引理

1) 流出A的-流入A的=流的值
   a) 证明

v(f)=流出s的值

=A中所有点（流出和-流入和）

但是因为中间节点流守恒，所以

=流出A的-流入A的

2) weak duality
   a) 流的值v(f) =capacity（A，B）
3) 最大流等于最小割

iii. Ford-Fulkerson algorithm
   1) 每次都找到一条增广路径，路径上最小残余流即为bottleneck capacity，路径上每一步都减去这个值，然后再找一条增广路径，直到找不到为止

iv. 耗时
   1) 最多f。因为每一次至少加一
   2) 如果最大capacity是1，则算法时间复杂度为O（m+n）