HW14

Page370：23.2-6
Page381：24.1-3
Page383：24.2-2
Page383：25.2-6
Page383：25.3-4，25.3-5

Ch.23

P187 23.2-6

***23.2-6** 假定一个图中所有的边权重均匀分布在半开区间[0，1)内。Prim 算法和 Kruskal 算法哪一个可以运行得更快?

解答：参见课本上对Prim算法和Kruskal算法时间复杂度的分析：

```
MST-KRUSKAL(G,w)
A=∅
for each vertex v∈G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge(u,v)∈G.E, taken in nondecreasing order by weight
    if FIND-SET(v)≠FIND-SET(v)
        A=A∪{(u,v)}
        UNION(u,v)
return A
```

对于图 $G=(V, E)$，Kruskal 算法的运行时间依赖于不相交集合数据结构的实现方式。假定使用 21.3 节所讨论的不相交集合森林实现，并增加按秩合并和路径压缩的功能，因为这是目前已知的渐近时间最快的实现方式。在这种实现模式下，算法第 1 行对集合 A 的初始化时间为 $O(1)$，第 4 行对边进行排序的时间为 $O(E\lg E)$(稍后将会讨论算法第 2～3 行 **for** 循环中的 $|V|$ 个 MAKE-SET 操作的代价)。算法第 5～8 行的 **for** 循环执行 $O(E)$ 个 FIND-SET 和 UNION 操作。与 $|V|$ 个 MAKE-SET 操作一起，这些操作的总运行时间为 $O((V+E)\alpha(V))$，这里 α 是 21.4 节所定义的一个增长非常缓慢的函数。由于假定图 G 是连通的，因此有 $|E|\geqslant|V|-1$，所以不相交集合操作的时间代价为 $O(E\alpha(V))$。而且，由于 $\alpha(|V|)=O(\lg V)=O(\lg E)$，Kruskal 算法的总运行时间为 $O(E\lg E)$。如果再注意到 $|E|<|V|^2$，则有 $\lg|E|=O(\lg V)$，因此，我们可以将 Kruskal 算法的时间重新表示为 $O(E\lg V)$。

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

Prim 算法的运行时间取决于最小优先队列 Q 的实现方式。如果将 Q 实现为一个二叉最小优先队列(请参阅第 6 章的内容)，我们可以使用 BUILD-MIN-HEAP 来执行算法的第 1～5 行，时间成本为 $O(V)$。**while** 循环中的语句一共要执行 $|V|$ 次，由于每个 EXTRACT-MIN 操作需要的时间成本为 $O(\lg V)$，EXTRACT-MIN 操作的总时间为 $O(V\lg V)$。由于所有邻接链表的长度之和为 $2|E|$，算法第 8～11 行的 **for** 循环的总执行次数为 $O(E)$。在 **for** 循环里面，我们可以在常数时间内完成对一个结点是否属于队列 Q 的判断，方法就是对每个结点维护一个标志位来指明该结点是否属于 Q，并在将结点从 Q 中删除的时候对该标志位进行更新。算法第 11 行的赋值操作涉及一个隐含的 DECREASE-KEY 操作，该操作在二叉最小堆上执行的时间成本为 $O(\lg V)$。因此，Prim 算法的总时间代价为 $O(V\lg V+E\lg V)=O(E\lg V)$。从渐近意义上来说，它与 Kruskal 算法的运行时间相同。

如果使用斐波那契堆来实现最小优先队列 Q，Prim 算法的渐近运行时间可以得到进一步改善。第 19 章的内容告诉我们，如果斐波那契堆中有 $|V|$ 个元素，则 EXTRACT-MIN 操作的时间摊还代价为 $O(\lg V)$，而 DECREASE-KEY 操作(用于实现算法第 11 行的操作)的摊还时间代价为 $O(1)$。因此，如果使用斐波那契堆来实现最小优先队列 Q，则 Prim 算法的运行时间将改进到 $O(E+V\lg V)$。

由以上分析，Kruskal算法的运行时间依赖于排序：$O(E\lg E)$，和不相交数据结构的MAKE-SET和UNION：$O(E\alpha(V))$，最终总时间复杂度是$O(E\lg E)=O(E\lg V)$。如果限定边权值在$[0,1)$之内，则可以用更快的桶排序算法，桶排序的时间复杂度是$O(E)$，则可以做到更快的时间复杂度：$O(E)+O(E\alpha(V))=O(E\alpha(V))$。（$\alpha$函数的增长速度慢于lg函数。）

Prim算法的运行时间取决于最小优先队列的实现方式，算法运行过程中并不需要排序，而限定边权值在$[0,1)$之内这一信息对最小优先队列的实现并没有帮助。

所以本题的最终答案是：Krskual算法可以加速到$O(E\alpha(V))$，而Prim算法不能运行得更快。

Ch.24

P381 24.1-3

24.1-3 **给定 $G=(V, E)$是一带权重且没有权重为负值的环路的有向图，对于所有结点 $v\in V$，从源结点 s 到结点 v 之间的最短路径中，包含边的条数的最大值为 m。(这里，判断最短路径的根据是权重，不是边的条数。)请对算法 BELLMAN-FORD 进行简单修改，可以让其在 $m+1$ 遍松弛操作之后终止，即使 m 不是事先知道的一个数值。**

解答：假设从s到v的最短路径包含m条边，那么经过m轮RELAX操作就可以求出s的最短路径权值。所以m轮迭代后已经求出了以s为源到所有节点的最短路径权值。所以可以以图是否改变作为判断依据，第m+1轮迭代时图是不变的。参考伪代码如下：

```
BELLMAN-FORD-M(G,w,s)
INITALIZE-SINGLE-SOURCE(G,s)
change = true
while change == true
    change = false
    for each edge(u,v) in G.E
        RELAX-M(u,v,w)

RELAX-M(u,v,w)
if v.d > u.d + w(u,v)
    v.d = u.d + w(u,v)
    v.pi = u
    change = true
```

P383 24.2-2

24.2-2 假定将 DAG-SHORTEST-PATHS 的第 3 行改为：

3 **for** the first $|V|-1$ vertices，taken in topologically sorted order

证明：该算法的正确性保持不变。

DAG-SHORTEST-PATHS(G, w, s)

1 topologically sort the vertices of G

2 INITIALIZE-SINGLE-SOURCE(G, s)

3 **for** each vertex u，taken in topologically sorted order

4   **for** each vertex $v \in G.Adj[u]$

5   RELAX(u, v, w)

解答：因为拓扑排序后的节点序列中，最后一个节点一定没有出边。所以最后一个节点不执行第4，5行对算法的正确性没有影响。

Ch.25

P408 25.2-6

25.2-6 我们怎样才能使用 Floyd-Warshall 算法的输出来检测权重为负值的环路？

解答：

法1. 检查最终得到的矩阵的主对角线，即$d_{ii}^{(n)}$。存在某个**i**使得$d_{ii}^{(n)} < 0$，当且仅当存在权重为负值的环路。证明：

充分性：显然$d_{ii}^{(n)}$对应一条从i到自身的路径，即一个环，而它的权重为负值。

必要性：假设存在负环，证明的思路是说明在某一轮迭代之后，会存在$d_{ii}^{(k)} < 0$，进而导致$d_{ii}^{(n)} < 0$。考虑这些负环中含有边的数量最少的那一个负环。

①若它只含有一条边，那么存在$w_{ii} < 0$，根据d的递推关系，$d_{ii}^{(n)} \leq 2 * d_{ii}^{(n-1)} \leq \ldots \leq 2^n * w_{ii} < 0$。

②若它含有2条或以上的边，令k是其中序号最大的边，那么考虑$d_{ik}^{(k-1)}$和$d_{ki}^{(k-1)}$，因为取了边数最少的负环，不存在边数更少的负环了；而且$d_{ik}^{(k-1)}$和$d_{ki}^{(k-1)}$中不可包含节点k，所以这两条路径中不可能包含负环，所以到$d_{ik}^{(k-1)}$和$d_{ki}^{(k-1)}$这一步算法是无误的，可以正确地计算出从i到k和从k到i的，且节点序号不超过k-1的最短路径权值。而因为$i \to k \to i$是一个负环，所以$d_{ii}^{(k)}$=$d_{ik}^{(k-1)}$+$d_{ki}^{(k-1)} < 0$。那么仿照①，
$d_{ii}^{(n)} \leq 2 * d_{ii}^{(n-1)} \leq \ldots \leq 2^{n-k} * d_{ii}^{(k)} < 0$。

所以只要检查是否存在某个i使得$d_{ii}^{(n)} < 0$，就可以检测出是否有负环，不会出现误判和漏判的情况。

法2. 额外进行一轮Floyd-Warshall的迭代。这一轮额外迭代之后矩阵发生变化，当且仅当存在权重为负值的环路。证明：

充分性：显然若不存在负环，那么Floyd-Warshell算法是正确的，所有节点对的最短路径在额外迭代之前已经被求出，再进行一轮迭代也不会变化。所以发生变化一定有负环。

必要性：如果有负环，那么一些点对之间的最短路径是可以达到无穷小的，所以在经过一轮迭代以后一定会有变化。

所以只要检查是否在额外迭代后矩阵是否变化，就可以检测出是否有负环，不会出现误判和漏判的情况。

P412 25.3-4

25.3-4 Greenstreet 教授声称，他有一种比 Johnson 算法中所使用的更简单的办法来对边的权重进行重新赋值。设 $w^* = \min_{(u,v)\in E}\{w(u, v)\}$，只要对所有的边$(u, v) \in E$，定义$\hat{w}(u, v) = w(u, v) - w^*$ 即可。请问这种重新赋值有什么错误?

解答：不同的路径包含的边数不一样，全部边权值加一个常数会导致边数多的路径被加上了多次这个常数。具体的反例也很容易举出。例如：$w(a,b) = w(b,c) = -10$，$w(a,c) = -15$。全部减去-15后变为$w'(a,b) = w'(b,c) = 5, w'(a,c) = 0$。容易发现由a到c的最短路径由$a \to b \to c$变为了$a \to c$。

P412 25.3-5

25.3-5 假定在一个权重函数为 w 的有向图 G 上运行 Johnson 算法。证明：如果图 G 包含一条权重为 0 的环路 c，那么对于环路 c 上的每条边(u, v)，$\hat{w}(u, v) = 0$。

解答：因为$w'(u,v)=w(u,v)+h(u)-h(v)$，那么考虑所有环路c上的边，显然有$\sum w'=\sum w=0$。而调整后的权重又满足$w'\geq 0$，所以原结论得证。