算法设计与分析 (2017 年春季学期)

第三次作业参考答案

1 最小生成树问题 (20 分)

给定一个无向连通图 $G=(V,E)$，其中每条边的权值只可为 1 或 2。请设计一个时间复杂度为 $O(|V|+|E|)$ 的算法来求 G 的一棵最小生成树，并验证其时间复杂度。（注：你可以提出一个新算法或修改课堂上讲过的算法。）

解：

该问题有多种解法，这里仅给出一种解法的主要思想。该解法的思路是使用一个更简单的数据结构来代替 $Prim$ 算法中用到的优先队列 (*priority queue*) 从而使得队列的插入和查找操作的复杂度都是 $O(1)$，而不是 $O(\log n)$。具体做法是使用两个链表来代替优先队列。在 $Prim$ 算法的运行过程中，用链表 L_1 来存储所有长度为 1 的边，L_2 来存储所有长度为 2 的边，这样可以很容易的实现 *Extract-Min()* 和 *Decrease-Key()* 操作。这两个函数的具体实现留作小练习，这里不再给出。

2 最小生成树性质的证明 (20 分)

令 $G=(V,E)$ 为一个带权无向连通图，且其每条边的权值都不相同。请证明：G 的任何一棵最小生成树都会包含权值最小的那条边。[注：必须从头开始证明，即是说，证明过程中不可使用 MST 引理，即关于“安全边 (*Saft Edge*)”的引理，且证明不可建立在 $Kruskal$ 算法或 $Prim$ 算法成立的基础上。]

解：

假设权值最小的那条边 (记为 $e=(u,v)$) 不在该图的最小生成树 (记为 T) 中，如果我们把 e 加入到 T 中，就会导致新的子图 $T'=T\cup\{e\}$ 中包含一个环。令 e' 是该环中除了 e 之外的任意一条边，因为 e 是原图中权值最小的边，所以 $w(e')$ 一定大于 $w(e)$，我们令 $T''=T\cup\{e\}-\{e'\}$，即从子图 T' 中删去 e' 这条边，那么 T'' 仍然是原图的一棵生成树，并且有：

$$w(T'')=W(T)+w(e)-w(e')<w(T)$$

这和 T 是原图的最小生成树矛盾，因此命题得证。

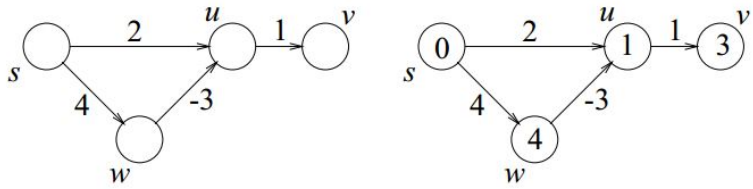
3 最短路问题 (20 分)

请给出一个边权可以为负的有向图实例，使得 $Dijkstra$ 算法在该图上无法得到正确的结果。并解释允许边权为负的情况下 $Dijkstra$ 算法不再正确的原因。(注：给出的实例应保证该有向图不存在权值和为负的环。)

解：

这个问题的关键就是在 $Dijkstra$ 算法的执行过程中，要在某个点的真实的最短路被求出之前将它加入已选集合。考虑如下图所示的实例，在计算到 s 的单源最短路的过程中，第一步我们会得到 $d[u]=2, d[w]=4$，因此我们先选择将 u 加入已选集合，从而得到 $d[v]=3$，下一步就会把 v 加入已选集合，最后再把 w 加入集合。这样我们最终得到 s 到 v 的最短路长度为 3，而事实上，路径 $<s,w,u,v>$ 的长度更短，为 2。

在允许边权为负的情况下 $Dijkstra$ 算法不再正确的原因如下：回忆在算法正确性证明过程中的第二种情况，令 y 是 s 到 u 的最短路上的任意一点且 $y\neq u$，在课上给出的证明过程中，我们断言因为 y 是 s 到 u 的最短路上的任意一点，所以一定有 $\delta(s,y)<\delta(s,u)$。这在所有边权为非



负的时候确实是正确的，但如果允许边权为负的话，该性质将不再成立。因此，此时 $Dijkstra$ 算法的正确性无法保证。

4 二分图判定问题 (20 分)

二分图是指一个无向图 $G = (V, E)$, 它的所有顶点可被分成两个子集，且同一个子集中任何两顶点间都没有边相连。（换言之，G 为二分图，当且仅当存在两个集合 V_1, V_2 满足 $V_1 \cup V_2 = V, V_1 \cap V_2 = \emptyset$，$E$ 中每条边都连接了 V_1 中某个点与 V_2 中某个点。）

1. 请证明：二分图中不存在长度为奇数的环。
2. 请设计一个基于广度优先搜索 (BFS) 的算法来判断无向图 G 是否为二分图，并分析算法的正确性及时间复杂度。

解：

1. **证明**：令 $G = (V, E)$ 为一个二分图，V_1, V_2 为题目所述的两个集合。考虑图 G 中的任意一个环 (记为 $C =< v_1, v_2, v_3, \ldots, v_n, v_1 >$)，不失一般性，我们假设 $v_1 \in V_1$，根据二分图的性质，可以得出结论：对于该环中任意一点 v_i，如果 i 是奇数，则 $v_i \in V_1$，如果 i 是偶数，则 $v_i \in V_2$。又因为存在一条边 (v_n, v_1)，故 n 一定是偶数。因此，G 中任意一个环的长度一定为偶数，不存在长度为奇数的环。

2. 为了简化问题，可以假定 G 是一个连通图 (G 不连通时的算法很容易从该算法扩展得到，留作小练习)。在图 G 上使用 BFS 进行遍历，可以得到一棵 BFS 树。对于图中的一条边 (u, v)，如果在 BFS 树上 u 既不是 v 的祖先，也不是 v 的子孙，那么我们把这条边 (u, v) 称为**交叉边** (*Cross Edge*)。我们的算法基于如下引理：

   引理：图 G 是二分图，当且仅当使用 BFS 对 G 进行遍历的过程中，任意一条交叉边的两个顶点 u, v 在 BFS 树中的深度都不相同。

   证明：假定 BFS 过程中不存在两顶点深度相同的交叉边。那么令 V_1 为所有的深度为奇数的点构成的集合，V_2 为所有深度为偶数的点构成的集合。回忆课上讲的 BFS 树的性质，图 G 中的交叉边两顶点的深度要么相同，要么相差 1。因此，如果不存在两顶点位于树上同一深度的交叉边，那么所有交叉边的两顶点一定一个属于 V_1，一个属于 V_2。从而可得，图 G 是一个二分图。

   反过来，假设图 G 中存在两顶点位于 BFS 树同一深度的交叉边，记该交叉边为 (u, v)。令 x 为该 BFS 树中点 u, v 的最近公共祖先(x 可以为根节点)，显然，我们可以从 x 走到 u，再从 u 走到 v，再从 v 走到 x，这条路径构成一个环并且长度为奇数。根据第一问中得到的结论，图 G 不是二分图。

   算法：根据上述引理可以得出一个很显然的算法来判断某图 G 是否是二分图。我们仅需要在图上执行 BFS 算法，并检查是否有一条交叉边的两顶点位于 BFS 树上的同一深度。该图是二分图当且仅当不存在这样的交叉边。该算法的时间复杂度和 BFS 相同，为 $O(|V| + |E|)$。

5 瓶颈值问题 (20 分)

令 $G = (V, E)$ 为一个带权无向图 (每条边 (u, v) 的权值为 $w(u, v)$)，且所有权值非负。对一条路径 $(u_0, u_1), (u_1, u_2), \ldots, (u_{n-2}, u_{n-1}), (u_{n-1}, u_n)$，该路径的**瓶颈值** (*Bottleneck Value*) 定义为 $min_{1 \le i \le n} w(u_{i-1}, u_i)$。

直观上来说，可以把图中的边看作水管，边的权值看作水管中每秒水的流量。在一秒内能流过一条路径的最大流量就是该路径上所有边权的最小值，也就是该路径的瓶颈值。

现给定两点 s,t，请设计一个算法找出一条从 s 到 t 的路径，它包含了 s 到 t 之间所有路径中最大的瓶颈值。(如果有多条这样的路径，仅需找出任意一条。) 请分析其正确性及时间复杂度。

提示：这个问题有多种解法。其中一种需要用到最大堆。最大堆和最小堆很像，唯一的区别是它每次都弹出最大的元素而不是最小的元素。它们的实现方法相似且运行时间相同。如果你的算法需要用到最大堆，你可以直接使用它而不必从头开始证明其正确性。

解：

在算法的最开始，我们可以先使用 BFS 来找出所有和 s 位于同一连通块的点，如果 t 和 s 没有位于同一连通块的话，那么可以直接返回无解；否则，我们可以仅考虑 s,t 所在的连通块。因此，在下面的分析中，我们假设图是连通的。

我们会给出一个和 $Dijkstra$ 算法类似的方法来解决该问题，不同的是，在该算法中我们使用最大堆。如果你之前从来没有了解过最大堆的话，可以使用最小堆，但把对 K 的比较替换为对 $-K$ 的比较。

下面我们会用黑色节点代表已被处理过的节点，白色节点代表未被处理过的节点。我们仍然通过计算 $d[u]$ 来得到从 s 到 u 的最大瓶颈值。$pred[u]$ 代表该最大瓶颈值所在的路径上，u 节点的前驱结点。

Algorithm 1 $Bottleneck(G, w, s)$

```
for u ∈ V do
    d[u] ← 0;
    color[u] ← WHITE;
end for
d[s] ← ∞;
pred[s] ← NULL;
Q ← (queuewithallvertices);
while Non–Empty(Q) do
    u ← Extract–Max(Q);
    for v ∈ Adj[u] with color[v] = WHITE do
        if min(d[u], w(u, v)) > d[v] then
            d[v] ← min(d[u], w(u, v));
            Increase–Key(Q, v, d[v]);
            pred[v] ← u;
        end if
    end for
    color[u] ← BLACK;
end while
```

该算法的时间复杂度和 $Dijkstra$ 算法相同，均为 $O(|E|\log|V|)$。

需要注意的是，在任意时刻，边集 $\{(pred(u),u) : u \text{ 为黑色节点 }\}$ 均构成一棵树。因为在 u 节点变黑之前，边 $(pred[u],u)$ 永远是连接了一个白色节点和一个黑色节点，因此在将边 $(pred[u],u)$ 加入边集后该边集中也不会有环。

因此，在算法结束后，边集 $\{(pred(u),u) : u \in V\}$ 构成一棵树。对任意一点 t，在树上都有唯一的一条从 s 到 t 的路径。

令 $D[u]$ 表示从 s 到 u 的真实的最大瓶颈值 (注意 $D[s]=\infty$，因为从自己到自己不需要经过任何路径)。我们可以断言，在上述树中的 s 到 t 的路径即为包含 s 到 t 的最大瓶颈值的路径。

定理：在点 u 被处理完后，则有 $d[u]=D[u]$。

证明：通过反证法来证明。假设该定理不成立，令 u 为第一个使得该定理不成立的节点，也就是说 $d[u]\neq D[u]$。因为 $d[u]$ 总是对应于从 s 到 u 的某条路径的瓶颈值，而 $D[u]$ 又是最大的瓶颈值，因此有 $D[u]>d[u]$。注意，在我们的假设下，在处理 u 时，所有的已被处理过的点 v 都有 $d[v]=D[v]$，并且 $d[u]$ 是当前优先队列中权值最大的。

令 $s=v_0,v_1,\ldots,v_{m-1},v_m$ 为 s 到 u 的最大瓶颈值所在的路径。即 $D[u]=min_{1\le i\le m}w(v_{i-1},v_i)$。

现在，对所有的 i，路径 $s=v_0,v_1,\ldots,v_i$ 的瓶颈值为 $min_{1\le j\le i}w(v_{j-1},v_j)$，因此：

$$D[u]=min_{1\le j\le m}w(v_{j-1},v_j)\le min_{1\le j\le i}w(v_{j-1},v_j)\le D[v_i]$$

令

$$i = min\{k : v_k \ not \ extracted \ from \ queue \ before \ u\}$$

为该路径上第一个未被处理的节点，可以发现一定有 $i < m-1$，因为，若 $i = m-1$，那么在 v_{m-1} 被处理时会有：

$$d[u] \geq min(D[v_{m-1}], w(v_{m-1}, u)) \geq D[u]$$

与我们的假设 $d[u] < D[u]$ 矛盾。

这意味着，在从队列中提取 u 时，v_i 仍然在队列中。然而，因为 v_{i-1} 已经被提取过了，因此，在处理 u 时，

$$\begin{aligned} d[v_i] &\geq min(D[v_{i-1}], w(v_{i-1}, v_i)) \\ &\geq min(D[v_{i-1}], D[u]) \\ &\geq D[u] \\ &> d[u] \end{aligned}$$

但是，因为我们此时提取的是 u，因此有 $d[u] \geq d[v_i]$，和上述结论矛盾。因此，定理得证。

注：还有另外一种做法来解决该问题。可以发现在该图的**最大生成树**上的从 s 到 t 的路径就是拥有最大瓶颈值的路径。最大生成树和最小生成树类似，是权值和最大的生成树。求一个图的最大生成树的算法也可以套用和最小生成树类似的算法，可以在 $O(|E| \log |V|)$ 的时间内解决。关于使用最大生成树解决该问题的详细资料可以参考论文：Navneet Malpani and Jianer Chen, "*Note on Practical Constructions of Maximum Bandwidth Paths*," Information Processing Letters, **83** (2002) 175-180. http://www.sciencedirect.com/science/article/pii/S0020019001003234