CMPS 101
Homework Assignment 7 **Solutions**

1. Let $N(n, h)$ denote the number of nodes at height $h \geq 0$ in an almost complete binary tree (ACBT) on $n \geq 0$ nodes. Prove that $N(n, h)$ satisfies the following double recurrence formula.

$$N(n,h) = \begin{cases} n - \lfloor n/2 \rfloor & \text{if } h = 0 \\ N(\lfloor n/2 \rfloor, h-1) & \text{if } h \geq 1 \end{cases}$$

(Hint: Let *T* be an ACBT with *n* nodes. First, show that *T* must have $n - \lfloor n/2 \rfloor$ leaves. Second, let $h \geq 1$ and consider the set of nodes in *T* of height *h*. Form a new ACBT T' by deleting all leaves from *T*. Argue that (1) T' has $\lfloor n/2 \rfloor$ nodes, and that (2) the nodes in T' of height $h-1$ are precisely the nodes in *T* of height *h*.)

Solution:
Let *T* be an ACBT on *n* nodes. If we index the nodes of *T* with integers in the range 1 to *n*, assigning 1 to the root, and descending level by level from left to right (just as in a heap), then we find that the parent of the node with index *i* has index $\lfloor i/2 \rfloor$. Therefore the parent of the rightmost leaf on the bottom level has index $\lfloor n/2 \rfloor$.

$$\underbrace{A_1 \;\; A_2 \;\; \cdots\cdots \;\; A_{\lfloor n/2 \rfloor}}_{\text{Internal Nodes}} \; \underbrace{A_{\lfloor n/2 \rfloor + 1} \;\; \cdots\cdots \;\; A_{n-1} \;\; A_n}_{\text{Leaves}}$$

This parent of the rightmost leaf must be the internal node of highest index. Thus *T* contains exactly $\lfloor n/2 \rfloor$ internal nodes, and hence $(\text{\# of Leaves in T}) = n - \lfloor n/2 \rfloor$. But these leaves are precisely the nodes at height 0 in *T*, and so $N(n, 0) = n - \lfloor n/2 \rfloor$.

Now let $h \geq 1$ and consider the set of nodes in *T* of height *h*. Form a new ACBT T' by deleting all leaves from T. Observe that T' has $n - (n - \lfloor n/2 \rfloor) = \lfloor n/2 \rfloor$ nodes. Observe also that upon passing to T', every internal node in *T* has had its height reduced by 1, since every descending path to a leaf was shortened by 1. Thus

$$\{\text{ Nodes in } T \text{ of height } h\} = \{\text{ Nodes in } T' \text{ of height } h\text{-}1\},$$

and therefore

$$N(n, h) = N(\lfloor n/2 \rfloor, h-1),$$

as was claimed. ///

2. Let $N(n,h)$ be defined as in the previous problem. Prove that $N(n,h) \le \left\lceil \frac{n}{2^{h+1}} \right\rceil$. (Hint: use the above recurrence formula together with (weak) induction on h starting at $h=0$.)

Solution:

I. Observe $n = \lfloor n/2 \rfloor + \lceil n/2 \rceil$ for any $n \ge 0$. Thus $N(n,0) = n - \lfloor n/2 \rfloor = \lceil n/2 \rceil$, so the base case is satisfied.

II. Let $h > 0$ and assume for any $m \ge 0$ that $N(m, h-1) \le \left\lceil \frac{m}{2^h} \right\rceil$. Then

$$\begin{aligned}
N(n,h) &= N\big(\lfloor n/2 \rfloor, h-1\big) && \text{by the recurrence formula for } N(n,h) \\
&\le \left\lceil \frac{\lfloor n/2 \rfloor}{2^h} \right\rceil && \text{by the induction hypothesis with } m = \lfloor n/2 \rfloor \\
&\le \left\lceil \frac{n/2}{2^h} \right\rceil && \text{since } \lfloor x \rfloor \le x \text{ and } y \le z \Rightarrow \lceil y \rceil \le \lceil z \rceil \\
&= \left\lceil \frac{n}{2^{h+1}} \right\rceil
\end{aligned}$$

The result therefore holds for all $n \ge 0$ and $h \ge 0$. ///

3. 6.5-3 page 165
Write pseudocode for the procedures HeapMinimum, HeapExtractMin, HeapDecreaseKey, and HeapInsert that implement a min-priority queue with a min-heap.

Solution:

<u>HeapMinimum(A)</u> pre: heapSize(A) ≥ 1
1. return A[1]

<u>HeapExtractMin(A)</u> pre: heapSize(A) ≥ 1
1. min = A[1]
2. exchange A[1] with A[heapSize[A]]
3. heapSize[A] = heapSize[A] - 1
4. Heapify(A, 1)
5. return min

<u>HeapDecreaseKey(A, i, k)</u> pre: $0 \le i \le \text{heapSize[A]}$

```
1. if k < A[i]
2.      A[i] = k
3.      while i ≥ 2 and A[parent(i)]>A[i]
4.           exchange A[i] with A[parent(i)]
5.           i = parent(i)
```

<u>HeapInsert(A, k)</u> pre: heapSize[A] < length[A]
1. heapSize[A] = heapSize[A] + 1
2. A[heapSize[A]] = ∞
3. HeapDecreaseKey(A, heapSize[A], k)

4. Let $G = (V, E)$ be a weighted directed graph and let $x \in V$. Suppose that after Initialize(G, s) is executed, some sequence of calls to Relax( ) causes $d[x]$ to be set to a finite value. Prove that G contains an s-x path of weight $d[x]$. (Hint: use induction on the number of calls to Relax( ).

   Proof: Let n denote the length of the relaxation sequence. If $n = 0$, then the only d-value which is finite after Initialization is that of the source s. Indeed, G does contain an s-s path of weight $d[s] = 0$, namely the trivial path. The base case is therefore verified.

   Let $n > 0$, and assume for any vertex x, that if $d[x]$ achieves a finite value during a sequence of fewer than n relaxations, then there exists an s-x path in G of weight $d[x]$. Now let $y \in V$ and consider a sequence of n relaxations in which $d[y]$ becomes finite. An edge of the form (x, y) must have been relaxed during this sequence, for some vertex x. On that relaxation step, $d[y]$ was set to $d[x] + w(x, y)$. Since we suppose that this number is finite, $d[x]$ must have been finite before Relax(x, y) was executed. Thus $d[x]$ became finite during a sequence of fewer than n relaxations, and by our induction hypothesis, there must exist an s-x path in G of weight $d[x]$. That path, followed by the edge (x, y), constitutes an s-y path in G of weight $d[x] + w(x, y) = d[y]$. ///

5. 24.1-3 p. 654
   Given a weighted directed graph $G = (V, E)$ with no negative-weight cycles, let m be the maximum over all vertices $x \in V$ of the minimum number of edges in a shortest path from the source $s \in V$ to x. (Here, the shortest path is by weight, not by the number of edges.) Suggest a simple change to the Bellman-Ford algorithm that allows it to terminate in $m+1$ passes, even if m is not known in advance.

   Solution:
   We reproduce Bellman-Ford here for reference:

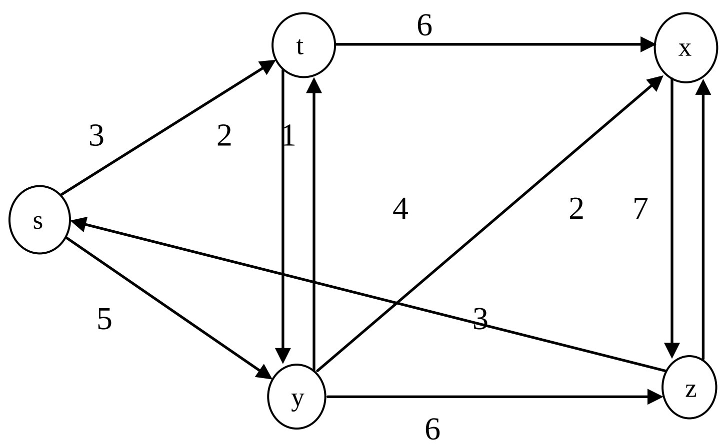
```
BellmanFord(G, s)
1. Initialize(G, s)
2. for i = 1 to |V| - 1
3.     for each edge (u,v) ∈ E
4.         Relax(u, v)
5. for each edge (u,v) ∈ E
6.     if d[v] > d[u] + w(u,v)
7.         return false
8. return true
```

   Note that we cannot simply alter the **for** statement on line 2 to say "for $i = 1 \text{ to } m+1$", since the value of m is not known ahead of time. Instead we modify Bellman-Ford so that loop 2-4 terminates as soon as one complete pass over the edge set results in no d-values being changed. Obviously no d-values will be changed by performing any further passes, so if we accept the correctness of Bellman-Ford (Lemma 24.2 and Theorem 24.4), the d and π values must be correct at that point. It remains only to show that this rule causes loop 2-4 to terminate after $m+1$ passes. To prove this it is sufficient to show that the *d*-values are correct after exactly *m* passes. This follows from the path-relaxation property (Lemma 24.15) which says:

> If $p=(v_0,v_1,\ldots,v_k)$ is a shortest path from $s=v_0$ to v_k, and the edges of *p* are relaxed in the order (v_0,v_1), (v_1,v_2), …., (v_{k-1},v_k), then $d[v_k]=\delta(s,v_k)$. This property holds regardless of any other relaxation steps that occur, even if they are intermixed with relaxations of the edges of *p*.

Each of the edges (v_0,v_1), (v_1,v_2), …., (v_{k-1},v_k) will be relaxed exactly once on each pass over the edge set, so *k* iterations of loop 2-4 suffice to correctly set the *d*-value of v_k. But by our definition of *m*, every vertex *v* (which is reachable from *s*) lies at the end of a shortest *s*-*v* path containing at most *m* edges, hence *m* iterations suffice to correctly set the *d*-values of all vertices in *G*. ///

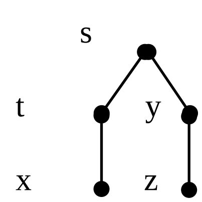
6. 24.3-1 p. 662
Run Dijkstra's algorithm on the directed graph of Figure 24.2 p. 648 (pictured below), first using vertex *s* as the source and then using vertex *z* as the source. Show the *d* and π values and the vertices in set *S* after each iteration of the **while** loop.



Solution:
With *s* as source:

	d/π-values after *i*-th iteration of while					
Vertex	0	1	2	3	4	5
s	0/ nil	0/ nil	0/ nil	0/ nil	0/ nil	0/ nil
t	∞/ nil	3/ s	3/ s	3/ s	3/ s	3/ s
x	∞/ nil	∞/ nil	9/ t	9/ t	9/ t	9/ t
y	∞/ nil	5/ s	5/ s	5/ s	5/ s	5/ s
z	∞/ nil	∞/ nil	∞/ nil	11/ y	11/ y	11/ y
Set *S*	$\varnothing$	{s}	{s, t}	{s, t, y}	{s, t, y, x}	{s, t, y, x, z}

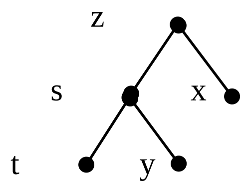
Predecessor subgraph:



With z as source:

	d/π-values after i-th iteration of while					
Vertex	0	1	2	3	4	5
s	∞/ nil	3/ z	3/ z	3/ z	3/ z	3/ z
t	∞/ nil	∞/ nil	6/ s	6/ s	6/ s	6/ s
x	∞/ nil	7/ z	7/ z	7/ z	7/ z	7/ z
y	∞/ nil	∞/ nil	8/ s	8/ s	8/ s	8/ s
z	0/ nil	0/ nil	0/ nil	0/ nil	0/ nil	0/ nil
Set S	$\varnothing$	{z}	{z, s}	{z, s, t}	{z, s, t, x}	{z, s, t, x, y}

Predecessor subgraph:



7. 24.3-6 p. 663

We are given a directed graph $G=(V,E)$ on which each edge $(u,v)\in E$ has an associated value $r(u,v)$, which is a real number in the range $0\le r(u,v)\le 1$ that represents the reliability of a communication channel from vertex u to vertex v. We interpret $r(u,v)$ as the probability that the channel from u to v will not fail, and we assume that these probabilities are independent. Give an efficient algorithm to find the most reliable path between two given vertices.

Solution:

Let p be a directed x-y path consisting of vertices: $x=v_0, v_1, v_2, \ldots, v_k=y$. Since the probabilities associated with each edge are independent, the probability that no edge along p fails is given by $r(p)=\prod_{i=1}^{k} r(v_{i-1},v_i)=r(v_0,v_1)\cdot r(v_1,v_2)\cdots r(v_{k-1},v_k)$. The most reliable x-y path which we seek, is the one that maximizes this quantity $r(p)$. Dijkstra's algorithm can be used to find this path by carefully defining an appropriate weight function on edges. Given $(u,v)\in E$, define $w(u,v)=-\log(r(u,v))$, where the log function can have any base greater than 1. Since $0\le r(u,v)\le 1$ we have $-\infty\le\log(r(u,v))\le 0$, and hence $0\le w(u,v)\le\infty$. Edge weights are therefore non-negative (and some may be infinite.) Running Dijkstra's algorithm on the source x will determine an x-y path which minimizes the quantity

$$
\begin{aligned}
w(p) &= \sum_{i=1}^{k} w(v_{i-1},v_i) \\
&= \sum_{i=1}^{k} -\log(r(v_{i-1},v_i)) \\
&= -\sum_{i=1}^{k} \log(r(v_{i-1},v_i)) \\
&= -\log\left(\prod_{i=1}^{k} r(v_{i-1},v_i)\right) \\
&= -\log(r(p)).
\end{aligned}
$$

But then p must maximize the quantity $\log(r(p))$, and since log is an increasing function, the path p also maximizes $r(p)$ as required. The following algorithm determines the most reliable directed x-y path in G.

Max-Reliable(G, x, y, r)
1. for each $(u,v) \in E(G)$
2. $w(u,v) = -\log(r(u,v))$
3. Dijkstra(G, w, x)
4. PrintPath(G, x, y)

///