**COMP 3711 Design and Analysis of Algorithms**

**Spring 2015 Final Exam Solution**

**Problem 1 (12 pts)**

**1.1(3 pts)** Hash table. (a) $O(1)$. (b) $O(1)$.
**1.2(3 pts)** A balanced BST such as the AVL Tree. (a) $O(\log n)$. (b) $O(\log n)$.
**1.3(3 pts)** A priority queue such as the min-heap. (a) $O(1)$. (b) $O(\log n)$. (c) $O(\log n)$. (Fibonacci heap is even better, but it's not covered.)
**1.4(3 + 3 pts)** The union-find data structure with a hash table (or a balanced BST) that maps the IP addresses to nodes in the union-find data structure. (a) $O(\log n)$. (b) $O(\log n)$.

**Problem 2 (15 pts)**

**Answer 1: Greedy** Choose the interval that covers 0 and has largest $y_i$. Then iteratively choose the interval that overlaps the already covered part with the largest $y_i$, until we reach 1. Naively picking such an interval takes $O(n)$ time, leading to $O(n^2)$ time in total. Below is a more efficient implementation of this greedy idea.

**Algorithm 1:** Interval-Covering **(7 pts)**

$S = \emptyset$, $covered = 0$, $I = \{1, ..n\}$;
sort the intervals by $x_i$ such that $x_1 \le x_2 \le \cdots \le x_n$;
$i \leftarrow 1$;
**while** $covered \le 1$ **do**
    starting from the $i$-th interval, scan to find the last $j$ such that $x_j \le covered$;
    from the $i$-th interval to the $j$-th interval, find $k$ such that $y_k$ is maximized;
    $S \leftarrow S \cup \{k\}, covered \leftarrow y_k$;
    $i \leftarrow j + 1$;

In the above algorithm, sorting takes $O(n \log n)$ time and the greedy loop takes $O(n)$ time because every interval is only examined once. So total running time is $O(n \log n)$. **(3 pts)**

Correctness proof. Let $G$ be the solution returned by this greedy algorithm, and $O$ be an optimal solution. Consider the first interval where $O$ is different from $G$. Suppose the interval chosen by $G$ is $[x_i, y_i]$ and the one chosen by $O$ is $[x_j, y_j]$. We must have both $x_i \le covered$ and $x_j \le covered$, and by the greedy choice, $y_i \ge y_j$. Now we modify $O$ by changing $[x_j, y_j]$ with $[x_i, y_i]$. This must still cover the interval $[0, 1]$. Repeatedly applying this transformation will convert $O$ into $G$. Thus G is also an optimal solution. **(5 pts)**

**Answer 2:** Consider each interval as a vertex of a graph. Build an edge $(u, v)$ if $x_u < x_v \le y_u < y_v$. This is a DAG, and the problem reduces to finding its shortest path, which takes $O(n^2)$ time. **(13 pts)**

**Problem 3 (13 pts)**

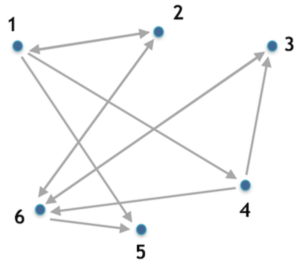
**Answer 1** For a given string $s$, denote by $S[i, j]$ the length of shortest symmetric supersequence of $s[i..j]$. We have the following recurrence relation (with base case $S[i, i] = 1$ and $S[i, i-1] = 0$): **(9 pts)**

$$S[i,j] = \begin{cases} S[i+1, j-1] + 2, & \text{if } i < j \ \& \ s[i] = s[j]; \\ min\{S[i, j-1], S[i+1, j]\} + 2, & \text{if } i < j \ \& \ s[i] \ne s[j]. \end{cases}$$

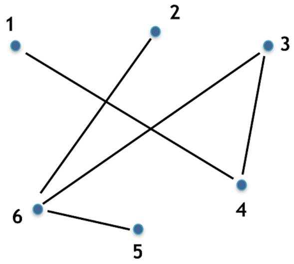
We can compute all the $S[i, j]$'s from small intervals to larger intervals. The running time is $O(n^2)$. **(4 pts)**

**Answer 2** Run the algorithm in the homework to find the length of longest symmetric subsequence $l(sub)$ and return $2n - l(sub)$ as the length of shortest symmetric supersequence. This also takes $O(n^2)$ time.

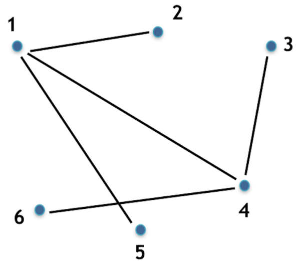
Correctness proof. There is a one-to-one mapping between symmetric subsequences and symmetric supersequences. For a given string $s$, let $sub$ be any symmetric subsequence. We can find a symmetric supersequence corresponding to $sub$ as follows. Keep all the characters in $sub$. Then for every remaining character, we add one to pair it off. We add a total of $n - l(sub)$ extra characters so the total length is $2n - l(sub)$. Likewise, for any symmetric supersequence $sup$, we can find its corresponding symmetric subsequence, by deleting all extra characters and their matched counterparts.



(a) Directed graph **(5 pts)**



(b) DFS-tree **(5 pts)**



(c) BFS-tree **(5 pts)**

## Problem 4 (15 pts)

## Problem 5 (15 pts)

**(a)(3 pts)** A counter example: a undirected triangle with weighted edges, $w(a,b) = 1$, $w(a,c) = 2$, $w(b,c) = 3$. The MST is $\{(a,b),(a,c)\}$ and edge $(a,c)$ is not the minimum-weight edge of cut $(\{a\},\{b,c\})$.
**(b)(12 pts)** Correctness proof. For any edge $e$ in $T$. Removing $e$ breaks $T$ into two parts $S$ and $V-S$. We claim that $e$ is the minimum-weight edge crossing the cut $(S, V-S)$. Indeed, if there is another $e'$ crossing the cut, we can replace $e$ with $e'$ to improve the MST, which contradicts with the fact that $T$ is an MST.

## Problem 6 (15 pts)

We have the following recurrence **(6 pts)**,

$$d(s,v) = max_{u,(u,v)\in E}\{d(s,u) + w(u,v)\} + w(v)$$

The algorithm is very similar to the one in the lecture notes. The running time is $O(V+E)$.

**Algorithm 2:** Maximum-weighted-Path-in-DAG **(9 pts)**

Topologically sort the vertices in $V$;
**for** *each vertex* $v \in V$ **do**
  $v.d \leftarrow -\infty$, $v.p \leftarrow$ *nil*;
$s.d \leftarrow w(s)$;
**for** *each vertex* $u$ *in topological order* **do**
  **for** *each vertex* $v \in Adj(u)$ **do**
    **if** $v.d < u.d + w(u,v) + w(v)$ **then**
      $v.d \leftarrow u.d + w(u,v) + w(v)$, $v.p \leftarrow u$;
**while** $t \neq s$ **do**
  print $t$;
  $t \leftarrow t.p$;
print $s$;

## Problem 7 (15 pts)

**(a)(2 pts)** The shortest path from $s$ to $t$ is $(s,b,t)$. $\Pr(s \to b \to t) = \frac{1}{2}$.
**(b)(4 pts)** There are 3 possible paths: $\Pr(s \to a \to c \to d \to t) = \frac{1}{4}$, $\Pr(s \to a \to b \to t) = \frac{1}{4}$, $\Pr(s \to b \to t) = \frac{1}{2}$. So $E[\text{length of the path chosen randomly}] = \sum_p \Pr(p) \cdot \mathit{length}(p) = \frac{11}{4}$.
**(c)(2 pts)** On $G_2$ the probability of routing along the shortest path is $\frac{1}{4}$.
**(7 pts)** Let $X(e)$ be the number of times edge $e$ appears on the randomly chosen path. By linearity of expectation, the expected total length is $\sum_e X(e)$. For $(s,a)$ and $(s,b)$, each of them appears once with probability $1/2$ and 0 times with probability $1/2$, so $E[X(s,a)] = E[X(s,b)] = 1/2$. Because the random path exits from $a$ or $b$ with equal probability, we also have $E[X(a,c)] = E[X(c,d)] = E[X(d,t)] = E[X(b,t)] = 1/2$. It only remains to compute $E[X(a,b)]$. This is the same as the waiting time problem with success (i.e., exit from the loop) probability $p = 1/2$, except that we do not count the success coin. So $E[(a,b)] = 1/p - 1 = 1$. Summing up all these expectations gives that the expected total length of the randomly chosen path is 4.