CS180 Solution 4

1. Consider an undirected graph that weights of edges are *distinct*. We can define an order on the set of spanning trees of the graph as follows: each spanning tree has $n-1$ edges, by sorting the weights of all $n-1$ tree edges in the *increasing* order, we obtain a tuple $(w_1, \ldots, w_{n-1})$ where $w_1 < w_2 < \ldots < w_{n-1}$. We can easily define a lexicographic order on tuples by imagining that each tuple is a word with $n-1$ letter. More precisely, we say tuple $(w'_1, \ldots, w'_{n-1})$ is bigger than tuple $(w_1, \ldots, w_{n-1})$ if for the smallest index i at which $w'_i \neq w_i$, we have $w'_i > w_i$. We say tree $T_1 > T_2$ if T_1's tuple is lexicographically bigger than T_2's tuple. We define the *lexicographically minimum tree* to be the smallest spanning tree in lexicographical ordering. Prove that any minimum spanning tree (MST) algorithm returns the lexicographically minimum tree.[25pts]

   Solution:

   Proof. Proof by contradiction. Let A be the lexicographically minimum spanning tree $(w_1, \ldots, w_{n-1})$, B be the minimum spanning tree $(w'_1, \ldots, w'_{n-1})$. Let w_i be the smallest i such that $w_i \neq w'_i$. Since A is the lexicographically minimum spanning tree, we have $w_i < w'_i$. We can add the edge w_i to B. This creates a cycle in tree B. This cycle contains an edge from the set $\{w'_i, w'_{i+1}, \ldots, w'_{n-1}\}$. Otherwise all the edges of the cycle are from the set $\{w'_1, w'_2, \ldots, w'_{i-1}\} = \{w_1, w_2, \ldots, w_{i-1}\}$, and this means that there is a cycle consisting of edge from the set $\{w_1, w_2, \ldots, w_{i-1}, w_i\}$ in tree A which is a contradiction. Therefore adding w_i to B creates a cycle in B that contains an edge from $\{w'_i, w'_{i+1}, \ldots, w'_{n-1}\}$. The weight of all edges in $\{w'_i, w'_{i+1}, \ldots, w'_{n-1}\}$ is greater than w_i, and deleting one of them from the cycle gives us a new tree with weight less than the minimum spanning tree. This is a contradiction. □

2. We have a number of n files, and file i has a length ℓ_i and probability p_i that whether it is accessed, and we want to write these files on a tape in some order. We assume that every time we need to access a file, we have to start from the beginning of tape until we reach the start of the requested file, so the time to access a file is proportional to the total length of all the files that are saved before it. Design an algorithm that finds the optimal order of files on the tape. The optimal order is an order that minimizes the average access time. The average access time is $\sum_{i=1}^{n} d_i p_i$ where d_i is distance of the beginning of file i from the beginning of the tape (the total length of all the files that is saved before file i). [25pts]

   Solution:

   The algorithm is a simple greedy algorithm. Compute for each file p_i/ℓ_i and order the files in the decreasing order of p_i/ℓ_i. This algorithm takes $O(n \log n)$. We next show why this algorithm works. Assume the optimal ordering of the files are not in the decreasing order of values of p_i/ℓ_i. Then, there are two adjacent file j and k such $p_j/\ell_j > p_k/\ell_k$ and k appears before j on the tape. We claim that switching the order of j and k reduces the average access time. Note that switching j and k leaves the average access time of all other files unchanged. Let t be the average access time if k appears before j on the tape, and t' be the average access time if j appears k, then:

$$t - t' = (d_k p_k + (d_k + \ell_k) p_j) - (d_k p_j + (d_k + \ell_j) p_k) = \ell_k p_j - \ell_j p_k > 0$$

   Note that $\ell_k p_j - \ell_j p_k > 0$ holds because we assume $p_j/\ell_j > p_k/\ell_k$. Thus, $t > t'$ which is contradiction that t is minimum average access time.

3. There is a multi-day tennis tournament with $n = 2^k$ players. On a given day, each player plays one match, so $n/2$ matches are played per day. Over the course of the tournament, each player plays against every other player exactly once, so a total of $n(n-1)/2$ matches will be played. Write a recursive algorithm to schedule these matches in the minimum number of days. It should take as input a number $n = 2^k$ and return as output a table with $n-1$ rows, in which row i is a list of the $n/2$ matches to be played on day i.

   Solution:

   Recursive algorithm: If $n = 2$, schedule the one match: $A[1,1] = (1,2)$. Else, schedule all matches among the first $\frac{n}{2}$ players and place those in block $A[1 \cdots (\frac{n}{2}-1), 1 \cdots \frac{n}{4}]$, and schedule all matches among the second $\frac{n}{2}$ players and place those in block $A[1 \cdots (\frac{n}{2}-1), (\frac{n}{4}+1) \cdots \frac{n}{2}]$.

   The remaining matches to be played each have one player from the first half and one player from the second half. Match these up one-to-one and use these matches on day $\frac{n}{2}$, then, for each next day, rotate the matching.

   Programmatically: Call these player halves $x_0, \ldots, x_{(\frac{n}{2}-1)}$ and $y_0, \ldots, y_{(\frac{n}{2}-1)}$. For $i = 0 \cdots (\frac{n}{2}-1)$ and $j = 0 \cdots (\frac{n}{2}-1)$, set $A[\frac{n}{2}+i, j+1] = (x_j, y_{(j+i) \bmod n/2})$. The only thing left to check is that no player appears twice in some row, and every player plays every other player. We can see this using the induction hypothesis and the the fact that for the last $\frac{n}{2}$ rows: player x_j always plays in column j and player y_j always plays in column $(j-i) \mod \frac{n}{2}$. Both algorithm have $O(n \log n)$ time complexity.

4. Given a undirected graph with *distinct* weights of edges $G = (V, E)$. We define a order between two paths A and B. We sort the weights of edges in the *decreasing* order and $A = \{w_1, w_2, \ldots, w_i\}, B = \{w'_1, w'_2, \ldots, w'_j\}$. We say A is lexicographically shorter than B, if the first different weight edge in A has smaller weight than the edge in B, more precisely if for the smallest index k in A at which $w'_k \neq w_k$, we have $w_k < w'_k$. If there is not such k that $w'_k \neq w_k$, the path with smaller number of edges is lexicographically shorter. We define the lexicographically shortest path as the path that is lexicographically shorter than any other path.

   (a) Design an algorithm for *all-pairs* lexicographically shortest path in the graph. The output is a matrix that in position (i, j) has value k where $e = \{i, k\}$ is the first edge on the shortest path from node i to node j.

   Solution:

   Use any MST algorithm to create MST. Given two nodes u and v, the lexicographically shortest path is the the path P that connects u and v in the MST.

   Theorem 1. *The lexicographically shortest path P from node u to v is the path defined by the MST tree*

   Proof. Proof by contradiction. Suppose there is another path P' such that the first different weight edge e' has the smaller weight than e in P ($w_{e'} < w_e$). We remove the edge e_b in the MST and disconnect the MST into two components, one component A contains the node u and the other B contains the node v. Since the path P' connect u with v, we can find a edge e_b in P' that connects A and B. Moreover, the weight of this edge $w_b \leq w_{e'} < w_e$. So, we can remove the edge e in the MST and add the edge e_b to it to get a new spanning tree such that it has the smaller weight than the original MST. Therefore, we get a contradiction and P is the path lexicographically shortest path. □

To construct the matrix, for position (i, j) we set it to the next node in the path from node i to node j in the MST.

(b) The matrix obviously is of size n^2. Can you renumber the nodes such that with the new numbering $1, \ldots, n$ with node i we associate a $1 \times (d-2)$ matrix that contains $d-2$ integers in 1 to n, where d is the degree of node i. Using this array given a j, node i can consult the array to find the desired first edge $e = \{i, k\}$ on the shortest path to j.

Solution:

We use the DFS to get a postorder numbering of the MST tree and record the the number interval for each nodes' children.

```
POSTNUMBER(u)
  mark u
  for each edge u − v
  if v is unmarked
      S[u − v] ← counter
      POSTNUMBER(v)
      E[u − v] ← counter
  number[v] ← clock
  clock← clock +1
```

Now, for each node u we know the post-order numbering interval for each child except its parent. Specifically, the postorder numbering of the subtree with child w as the root starts from $S[v-w]$ and ends at $E[v-w]$. Using the interval information we know want to find out the path from node u to node w and node w has the postorder number of x. We can check whether i is in the interval of one of u's child. If we have $S[u-v] \leq x < E[u-v]$, then the next node of the $u-w$ path is v. If x is not in any interval of u's child, the next node of $u-w$ path is u's parent.

Let u has child $v_1, v_2, \ldots v_i$. To further reduce the size of the interval, we notice that $E[u-v_k] = S[u-v_{k+1}]$ for $k \in \{1, i-1\}$ and $E[u-v_i] = \text{number}[u]$. Hence, we can drop the array E. The size of S is $d-1$.