# Mock Mid Semester Exam

1. You are the king of the country USB. There are $n$ cities and $m$ bidirectional train routes connecting the cities in USB. Each route has a unique cost associated with it. Any two cities in the country have at least one path between them consisting of the train routes. Being a kind king, you want to reduce the cost of traveling from one city to another. Specifically, you want to keep only those routes that have a cost less than $x$. But you still want to keep at least one path between any pair of cities. Propose an efficient algorithm to determine the largest $x$ and analyze its running time. $18 + 7$

**Solution:** Construct a graph $G$ with $n$ vertices and $m$ edges denoting the cities and the train routes respectively. Each edge will have an unique weight associated with it. Let's define graph $G_x$ that results from removing every edge in $G$ having weight $x$ or larger. We want to find out the largest $x$ such that $G_x$ is not connected. Construct an array $W$ containing the $m$ distinct edge weights in $G$, and sort it in $O(m \log(m))$ time, e.g., using merge sort.
We will use binary search to find $x$. Specifically, consider an edge weight $x'$ in $W$ (initially the median edge weight), and run either BFS or DFS from an arbitrary vertex $v$ in $G$ in $O(n + m)$ time.
If exactly $n$ vertices are reachable from $v$, then $G_{x'}$ is connected and $x > x'$; recurse on strictly larger values for $x'$. Otherwise, $G_{x'}$ is not connected, so $x \leq x'$; recurse on non-strictly smaller values for $x'$.
By dividing the search range by a constant fraction at each step (i.e., by always choosing the median index weight of the unsearched space), binary search will terminate after $O(\log(m))$ steps, identifying the largest value of $x$ such that $G_x$ is not connected.

This algorithm takes $O(m \log(m))$ time to sort the edge weights. If we consider the median each time, binary search will take $O(\log(m))$ steps. In each step, it will compute whether all the vertices are reachable or not in $O(n + m)$ time.
The total complexity: $O(m \log(m)) + O((n + m) \log(m)) = O(m \log(m))$ time.

2. You are given an array $A$ containing $n$ integers. Consider an increasing subsequence of array indices $B = (b_0, b_1, \ldots, b_{m-1})$ where $0 \leq b_0 < b_1 < \cdots < b_{m-1} < n$. Your task is to find out the maximum value of the following function if the indices are picked optimally: $5 \times 5$

$$\sum_{i=0}^{m-1} (-1)^i A[b_i] = A[b_0] - A[b_1] + A[b_2] - A[b_3] + \ldots$$

Propose a dynamic programming solution to the problem. You need to define a set of subproblems, relate the subproblems recursively, provide base cases, construct a solution from the subproblems, and analyze the running time.

**Solution:** **Subproblems**

- $x(i, j)$ : maximum sum of any alternating subsequence from integers $i$ to $n$, assuming the sum starts with $j = +1$.

**Relate**

- Either the first integer is in the alternating sum or not (Guess!)
- $x(i, j) = max(j \cdot A[i] + x(i + 1, -j), x(i + 1, j))$

- Subproblems $x(i, j)$ only depend on strictly larger $i$, so acyclic.

**Base**

- No integers, no sum! $x(n, j) = 0$

**Solution**

- For $i \in [0, n-1]$, find $max(x(i, 1))$.

**Time**

- # of subproblems: $O(n)$
- Work per subproblem: $O(1)$
- Total: $O(n)$

3. *Seed for Need* is a racing video game set in Fortune City where the player needs to carry seeds to the farmers by driving their cars. There are $N$ towns in Fortune City. The towns are connected by $M$ roads. Each town has a positive integer difficulty level. When you go from town $u$ to town $v$, you will face obstacles if the difficulty level of town $u$ is strictly less than the difficulty level of town $v$. You are given the map of Fortune City containing the difficulty level of each town and the length of each road.

   Consider that you are in town $X$ and you need to go to town $Y$ carrying the seeds as fast as you can. Your car travels along the roads at a constant speed $S$. However, when you enter a town that has obstacles, your car will be delayed by a fixed amount of time $D$.
   Your goal is to find a path to go from town $X$ to town $Y$ as quickly as possible.

   a) Construct the graph associated to the problem. 10

   **Solution:** Construct a weighted directed graph on the $N$ towns, with a directed edge from $u$ to $v$ when there is a road connecting $u$ and $v$; if the difficulty of town $u$ is higher than town $v$, weight the edge by its length divided by $S$, while if the difficulty of town $v$ is higher (and $v$ is not $Y$), weight the edge by its length divided by $S$ and add $D$ to it. This graph will have $N$ vertices and $2M$ edges, which can be constructed in $O(N + M)$ time.

   b) Describe and justify the graph algorithm applied to plan your route. $6 + 6$

   **Solution:** Because all the distances are positive, and so are the edge weights, we can run Dijkstra to compute shortest paths from $X$ to $Y$. The shortest path from $X$ to $Y$ will have the total weight equal to the shortest time for you to reach $Y$. Using a $parent$ array, we can keep track of the path.
   Here, we cannot use linear time approaches as the graph has cycles and non-integral weights. Because all weights are positive, we can use Dijkstra. It will be faster than Bellman-Ford.

   c) State the running time of your algorithm in terms of the nodes and edges in your graph. 3

**Solution:** It will take $O(N \log(N) + M)$ time.