Sum Yi Li
(UID: 505146702)

Discussion 1A
10:00am-11:50am
Goldstein Orpaz

(1)

(1) We will use BFS to find the shortest path from node $v$ to node $w$ in a graph $G$

(2) According to the definition of BFS, "For each $j \geq 1$, layer $L_j$ produced by BFS consists of all nodes at distance exactly $j$ from $s$. There is a path from $s$ to $t$ if and only if $t$ appears in some layer". It means that each new layer from the graph $G$ will add one node to the path that we want to find.

(3) Define a set of layers $W_0, W_1, W_2, \ldots$, such that node $v$ is located at layer $W_0$.

(4) First, we want to compute the smallest number of nodes that takes us from node $v$ to every other node $a$. This means that we want to compute the number of shortest path from node $v$ to node $a$, denoted by $P(C)$

(5) When $P(C) = 1$, this means that this is the shortest path for every node at Layer $W_1$ from node $v$ to node $a$.

(6) Let assume there is a node $b$ some Layer $L_i$, $i > 1$

(7) Before we getting to node $b$, all the shortest paths from node $v$ to node $b$ contains a shortest path from node $v$ to node $c$ that locates at Layer $L_{i-1}$.

⑧ This means that it only take one more layer to get to node b.

⑨ Therefore, we can sum up all $P(c)$ for all nodes c up to $\text{Layer}_{i-1}$ plus one more layer (edge) to get to node b.

⑩ So, the time it takes to run this algorithm is the total time it takes to traverse all the edges & nodes between node v & node w.

$$\underbrace{O(m}_{\text{edges}}+\underbrace{n)}_{\text{nodes}}$$

②

- First, We will prove by contradiction.
- Let's assume that there exist an edge that does not exist in the tree T that is built by both BFS and DFS. The edge is specified as {x, y}
- According to the definition of DFS, if there is a path between two vertices x & y, then one of the vertex must be the parent of the other vertex. In our case, let's assume vertex x is the parent of vertex y.
- According to the definition of BFS, the distance of two different vertices starting from the same vertex w in the tree can only different up to one which is the difference between the two paths is one node or one level.
- If there is a vertex t in the tree and vertex x is the parent of vertex y, which means that the distance between vertex t & y is greater than the distance between vertex x & y by one.
- We have shown that x is for sure a parent of y.

- Therefore, there is a path or edge from vertex x to vertex y.

- However, this contradicts what we assume before that the edge $\{x, y\}$ does not exists in the tree

③

- We will prove the claim by contradiction
- According to the problem statement, we have $G$ be a graph on $n$ nodes, where $n$ is an even number, and every node of $G$ has degree at least $n/2$
- Let's assume that the graph $G$ is not connected.
- Let assume we have a smallest component and it is connected. Since the graph is not connected, it means that we have more than one connected component (at least two components)
- Let $N$ be the nodes from the smallest component
- For the degree of nodes, we have $N \leq n/2$
- For every node inside $N$, all of its adjacent nodes are also inside $N$.
- The degree of each nearby node in $N$ is $\frac{n}{2} - 1$ which is less than $\frac{n}{2}$
- However, this result contradict what we assume before which is every node of $G$ has degree at least $n/2$

■

(4)

- First, we want to show our greedy algorithm use the smallest number of trucks that needed.
- According to the problem statement, each truck has a weight limit $W$ that it can carry.
- Let assume we have $X$ number of boxes & each box has arrived in the order such that $x_1, x_2, \ldots, x_n$. Each box has it own weight such as $W_i$ according to problem statement.
- Every box is being assigned to one of the total number of trucks.
- There are 2 specifications about the assignment of boxes to trucks.
- All the carried box weight on each truck is either less than or the same as the weight limit $W$.
- The company requires boxes to ship in the order they arrive. This means that if a box $d_a$ is being shipped before another box $d_b$, then it means that box $d_a$ arrive before
- Then, we have showed that the greedy algorithm we have establish the optimality of using the less amount of trucks for delivering the box

④ cont.

- However, let's assume that there is another solution other than our greedy algorithm solution.
- Let $t$ be the number of trucks that our greedy algorithm has used to pack our boxes.
- Let assume our greedy algorithm solution is able to pack all $y_1, y_2, \ldots y_n$ into the $t$ number of trucks
- Let assume the other solution is able to pack all $y_1, y_2, \ldots, y_m$ into $t$ number of trucks.
- However, $m \leq n$ according to our definition of the greedy algorithm.
- We will prove the algorithm by induction.
- The first case when $z=1$, the first truck is being stuffed with as much boxes as we could according to our greedy algorithm.
- Let assume the situation is the same for $z-1$, which means that our greedy algorithm solution can fit $n'$ boxes into the $z-1$ trucks. The other solution can fit $m'$ boxes into the $z-1$ trucks. However, $m' \leq n'$ according to our definition of our greedy algorithm.

- For the sitution of $z$ trucks, our greedy algorithm could fit $y_{n'+1}, \dots, y_m$ boxes while the other solution can $y_{m'+1}, \dots, y_m$ boxes
- This means that our greedy algorithm solution is able to fit more boxes into the same number of trucks.

∎

⑤

- Let assume for the construction purpose, we construct the base stations starting from western point to eastern point.
- Our algorithm will aim at stretching out the covered distance as much as we could. Therefore, we will place a station where there is a house a that is 4 miles away from the western endpoint. The process will be repeated by placing a base station every 4 miles until we hit the eastern end point.
- This means that we place our first base station at far east as possible at location $b_1$ in order to cover the houses that are from 0 to $b_1$ distance.
- If we have already placed $\{b_1, \dots, b_x\}$ base station, then our $b_{x+1}$ base station will be placed at a stretched out distance which could cover all the houses from $b_x$ to $b_{x+1}$.
- We are gonna prove this claim by induction.
- Let assume that our greedy algorithm has set up all the base stations $B = \{b_1, \dots, b_j\}$
- Let assume that the other optimal solution has set up all the base stations $C = \{C_1, .. C_k\}$
- We first claim that $b_\ell \geq C_\ell$ for every $\ell$ which means we claim that everytime our greedy algorithm

- generates a base station that could cover a longer distance compared with the other optimal solution
- For our base case when $\ell = 1$, we could decide to stretch as much as we could for placing the first base station. We need to first show up to a point, j & k are the same.
- Next, we let the situation is also true for $\ell > 1$. Our greedy algorithm station position $\{b_1, \dots, b_\ell\}$ could cover all the houses that are covered by the other optimal solution first $\{C_1, \dots, C_\ell\}$ station positions.
- When we add the next base station $C_{\ell+1}$ to our set of $\{b_1, \dots, b_\ell\}$, we will cover every house within the distance $b_\ell$ to $C_{\ell+1}$.
- According to our greedy algorithm, we will choose $b_{\ell+1}$ to stretch as far as possible when we are placing our $\ell+1$ base station in order to cover the houses that are located within the distance $b_\ell$ & $b_{\ell+1}$.
- Therefore, we have shown that the greedy algorithm base station position $b_{\ell+1} \geq C_{\ell+1}$ the optimal solution station position.

⑤ cont.

- On the other hand, if we have more base station from our greedy algorithm, it means that even combining part of the base station from optimal solution, it will not cover the houses.
- At the same time, the optimal solution alone also will not cover the houses because $b_K \geq C_K$, this creates a contradiction

■

6a

- No, it is impossible to design an algorithm that finds the longest path in a directed graph
- It is because there are many different longest paths between 2 nodes under no restrictions.
- For example, there are **two** longest paths possibilities, from node q to node t.

$$\begin{tikzcd}
q \arrow[r] \arrow[d] & r \arrow[d] \\
s \arrow[r] & t
\end{tikzcd}$$

- 1st path : $q \rightarrow r \rightarrow t$
- 2nd path : $q \rightarrow s \rightarrow t$

- First, initialize distances to all vertices as infinite and the distance to the source as zero.
- Second, process the vertices in topological order of graph G
    - Find a node v with no incoming edges and order it first in order to compute the topological ordering
    - Delete v from G
    - Recursively compute a topological ordering of G - {v} and append the order after v.
- For every vertex u in the topological list
    - For every adjacent vertex v of u
        - if the current path to vertex v is less than the path to vertex u plus the edge from u to v
            - Update the current path to vertex v to be the path from vertex u plus the edge from u to v.

- Loop through the list that store all the longest path to each vertex from the source vertex
- For i in list :

```
        if longestPath < list[i]
                longestPath = list[i]
```

- return longestPath.

66 cont.

- Time complexity is $O(\overset{\text{vertices}}{V}+\overset{\text{edges}}{E})$
- The time complexity to compute the topological ordering is $O(V+E)$
- The algorithm process all vertices and for every vertex, it loop through all the adjacent vertices.
- The outer loop runs $O(V)$ & the inner loop runs $O(E)$
- The overall complexity of the algorithm is $\boxed{O(V+E).}$