Arthur Zhou

Dis 1A, 10:00am-11:50am
Parshan

1) BWOC, assume there's edge e that is in G but not T.

3 Cases arise: Say e is edge connecting nodes a & b

1. a and b are on same layer
Literally cannot happen for DFS. If a is visited first b is a subtree of a or vice versa. As a result this is a contradiction

2. a & b are a layer apart
Assume T is a DFS tree. No matter what gets visited first (a or b) the other must get visited by the definition of a DFS search, thus forming an edge

3. a & b are MORE than 1 layer apart
Since T is a BFS-based tree, it it selects one of (a, b), the other one gets added to the NEXT layer, contradicting the statement that they're more than 1 layer apart.

∴ Since all 3 cases are a contradiction for G containing e while T doesn't, $\boxed{G = T}$

2) Every node should be initialized w/ 3 variables: shortestDist, numOfShortestDist, and a boolean called visited. Initialize them to INT_MAX, 0, and FALSE respectively.

Set shortestDist and numOfShortestDist to both 1 for our startNode since both are 1 for our initial node

Perform BFS from our startNode:

- For every vertex v:
  - If NOT visited,
    - Set shortestDist of v to shortestDist of v's parent +1
    - Set numOfShortestDist of v to numOfShortestDist of v's parent +1
    - Set visited = TRUE
  - ELSE (has been visited),
    - IF shortestDist of v = shortestDist of v's parent + 1,
      - a) Increment v's numOfShortestDist by v's parent's numOfShortestDist (numOfShortestDist(v) += numOfShortestDist(v's parent))
    - If shortestDist of v is > than shortestDist of v's parent +1,
      - a) Set shortestDist of v to shortestDist of v's parent +1
      - b) Set numOfShortDist of v to numOfShortDist to v's parent
    - Else, (shortestDist of v is < shortestDist of v's parent +1)
      - a) Continue
- By the time BFS finishes, it would looking for node n just look at its numOfShortestDist

Time Complexity: $\boxed{O(v+e)}$ since we are performing BFS. Every other operation we perform as we visit the nodes is just O(1)

Proof By Induction:

BASE CASE: for # of layers = 1, the # of shortest Dist and the shortestDist are both 1 ✓

ASSUME: For # of layers from starting node to layer k-1, assume we have the shortest Path for ALL nodes from layers 1 to k-1

For each node n in layer k, our algorithm will correctly identify the shortest Distances and the # of them for every node in layer k. Since we have all previous layers identified and since the graph is connected,

$$\text{num Of Shortest Dist for any node in layer } k = \sum_{\substack{\text{every node in layer } k-1 \\ \text{where an edge} \\ \text{exists to } n}} \#\text{ of shortest Dist}$$

3) Prove by Induction: Assume greedy isn't optimal and assume the company's other method on sending on # of trucks is both

BASE CASE: For only 1 box both algorithms send only 1 truck

ASSUME: The greedy algorithm sent n boxes, while the other one sent m, where $m \leq n$ thus making greedy more efficient for being able to send more in the k^{th} step.

INDUCTION: For the k+1 step, assume both algorithms send x boxes total. The greedy will send $x-(n+1)$ more boxes while the other one sends $x-(m+1)$ more boxes. BUT! we know the sum of the weights of the $x-(n+1)$ more boxes is LESS than the max. weight of the truck be k+1 should be the LAST truck we send.

This means 2 things:

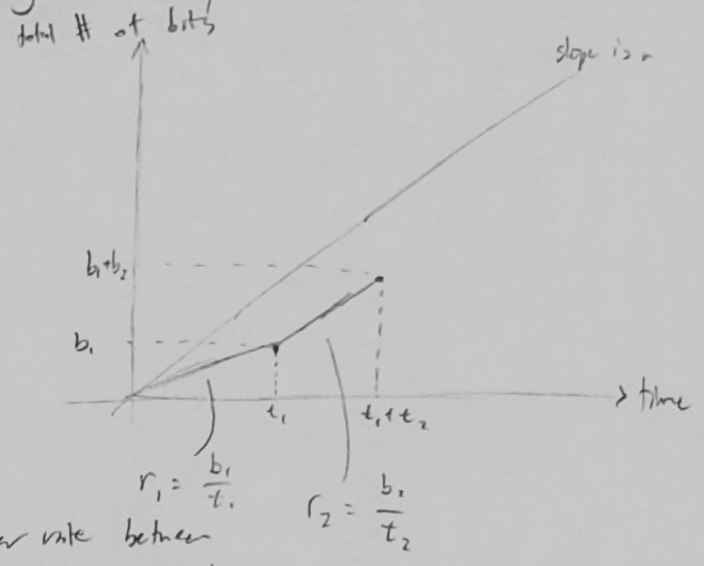
1) Boxes m+1 to x will at BEST send an EQUAL amount of weight as the greedy algorithm since $m+1 \leq n+1$ can't be equal to each other

2) Boxes m+1 to x is heavier than W because if m+1 is less than n+1, there's a chance it exceeds the max. weight of a truck, thus leading to us needing more

Thus Induction Method is contradicted and the greedy algorithm is more efficient bc the other algorithm sends at LEAST an equal amt of trucks as greedy.

b) Claim: Sort streams by their bit rate, or $r_i = \frac{b_i}{t_i}$ in increasing order. Play them in that order

Proof: Assume we ignore the algorithm, we must prove it doesn't work

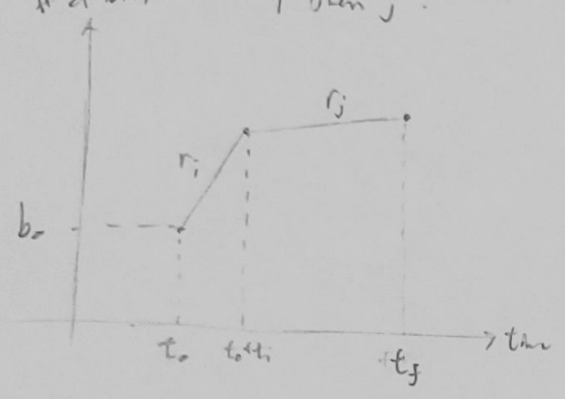


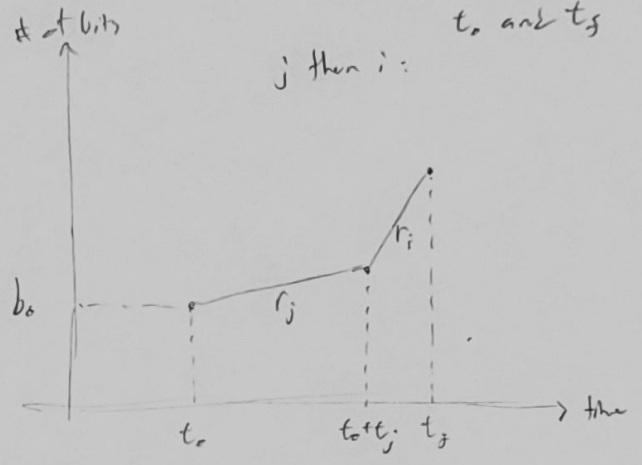
Let's assume we send stream i before j, such that $r_i > r_j$

Send i at t_0, i and j both finish sending at t_f

So $t_f = t_0 + t_i + t_j$.

REVERSE both i and j, claim that the reverse will have a LOWER transfer rate between t_0 and t_f





The total # of bits for each t in the "j then i" graph is LOWER than our original activity. It also becomes $r_i > r_j$

This is a contradiction bc an increasing cant exist and the optimal solution has to be in increasing order!

Time Complexity: we have to calculate the slope of the line. It's given by $\sum_{i=1}^{n} \frac{b_i}{t_i}$ which is $\boxed{O(n)}$. Interestingly enough it would be $O(n \log n)$ if we are asked to sort the array but we aren't so it's $O(n)$

6) • Begin by putting all coordinates of rotten oranges into a queue called q[i] where i starts at i=0.
• Iterate thru the entire matrix and sum up the total # of fresh oranges, called fOranges

Start BFS

• Pop first coordinate in q[i]
  ◦ Add all adjacent coordinates that are fresh oranges to q[i+1]
    ▪ Change all 1's to a 2
    ▪ Subtract # of changed 1's from fOranges
• Repeat until q[i] is empty, then i++ and repeat for new q[i]
  ◦ IF q[i+1] is empty
    ▪ If fOranges = 0 return i+1, bc all oranges rotted and you used an extra day for the last ones to rot
    ▪ If fOranges > 0, return -1 bc it's impossible for all oranges to rot

4) - Swimming MUST be solo
- Biking + Running can be done simultaneously among campers

So Biking + running is the most important factor!

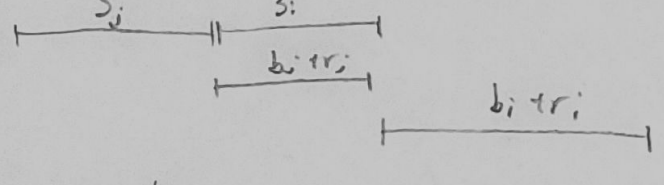
Solution: Sort the total biking + running times of all campers in a decreasing order. Have the ones with the longest (so first in our sort) go first to minimize time.

Time Complexity: Iterating thru all competitors is n, but sorting them all in decreasing order via heapsort is $O(n \log n)$ $O(n + n\log n) = O(n\log n)$

Proof: Let's say competitors i and j are sorted by biking + running $b+r$ time. i's $b+r$ time is LONGER than j's $b+r$ time. Let's say $b_i + r_i > b_j + r_j$ for short. To prove our algorithm works we must contradict that sending contestant j then i is better than sending i then j.

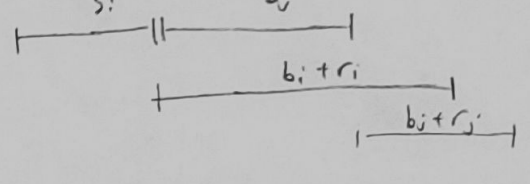
Send j then i:

swimming



time: $s_j + s_i + b_i + r_i$

Send i then j:



time: whichever's greater between $s_i + b_i + r_i$ or $s_i + s_j + b_j + r_j$

In both scenarios, $s_j + s_i + b_i + r_i$ > than sending i then j

(CASE 1)

$s_j + s_i + b_i + r_i < s_i + b_i + r_i$

$s_j < 0$, CONTRADICTED b2/c time must be positive

(CASE 2) $s_i + s_j + b_i + r_i < s_i + s_j + b_j + r_j$

$b_i + r_i < b_j + r_j$

FALSE! We stated $b_i + r_i > b_j + r_j$

Since both cases are contradicted we know sending the longest time first is more efficient

5)

a) No, here's a counter example:

if $r = 5000$

$(b_1, t_1) = (2000, 1)$

$(b_2, t_2) = (6000, 1)$

$b_2 \leq r t_2$

$6000 \leq 5000$ doesn't hold

but our schedule of airing 1 then 2 does!

Time Complexity: We iterate thru each coordinate once max., so our time complexity is $O(MN)$ for the total # of elements

Proof: From how BFS works if there's a path from a Rotten/Dirty orange to a fresh one that's of length x it should not cut $x+1$. If there isn't a path we return -1.