HW3 CS 180

ASHER CHRISTIAN 006-150-286

1. Exercise 6 P 108

$G = (V, E)$, $u \in V$ and bfs T = dfs T show that $G = T$ Suppose there exists some edge $e = (v, w) \in G$ and e not in T. Then v and w must differ in layer by at most one level since T is a bfs tree. In particular since $(v, w) \notin T$ v is not an ancestor of w and w is not an ancestor of v. This contradicts the fact that T is a dfs tree where each edge must connect elements such that one is the ancestor of the other in the tree structure. THus a contradiction and so no such e exists and $T = G$.

2. Exercise 4 P 190

I propose the following greedy algorithm assuming S and S' are in array form or linked list form.

- `maintain an index of` S `starting with the first element`
- `iterate through` S'
  - `using the index of` S `continue forward until the first match of` S `to the current element of` S'
  - `if all of` S `has been iterated over return false`
- `if all of` S' `has been iterated over successfully return true else return false`

In essence this greedy algorithm scans S matching elements in order of S' as quickly as possible and ending after searching all of S if it hasnt already matched all of S' in order. I propose that this algorithm works correctly. Firstly if the algorithm returns true then it has matched every element of S' to an element of S in strictly increasing order. This is the specification of the problem and so it holds true. Assume for contradiction that the algorithm returns false when it should have returned true then there exists some $\{s_1, s_2, ..., s_n\} \subset S$ elements such that their indexes are strictly increasing in S that satisfy S'. Consider the algorithm upon finding each of these elements inductively. Upon finding the first element it matches to the first element of S' so it matches and continues. Inductively, every prior element has not been found before since it comes at an index strictly greater than that of the prior elements and each element of this sequence comes directly after the previous element in S' so when the algorithm reaches this element it would process it. Thus the algorithm would have processed every element in this list and thus would have returned true which is a contradiction and so the algorithm is always correct. This algorithm runs in $O(n + m)$ time because it checkes each element of S and of S' $O(1)$ times.

Date: 30.01.25.

3. Exercise 12 P 193

I first claim that that claim: *There exists a valid schedule if and only if each stream i satisfies $b_i \leq rt_i$*
is false. Consider the $n = 2$ streams with

$$(b_1, t_1) = (6000, 1) \quad (b_2, t_2) = (1000, 1).$$

and $r = 5000$ Then $b_1 = 6000 > 5000 * 1 = rt_1$ but the streaming order $2, 1$ is valid because

$$t = 1 : 1000 < 5000 * 1$$
$$t = 2 : 7000 < 5000 * 2$$

I propose the following algorithm

- `Sort the streams in order of increasing` $\frac{b_i}{t_i}$
- `iterate through the sorted array keeping track of the sum of bits` B
  - `at each element of the array add` $b_i t_i$ `to` B `and compare` B `to` $r\sum_{n=0}^{i} t_i$
  - `If` B `is greater than this product return false`

I first claim that when the algorithm returns true that there is a ordering that works. Indeed By the loop to check for bits if at each ending time step (ending meaning one stream has ended) the sum of bits is lower than the required maximum bits to be sent in that time interval. Assume for contradiction that at some point during on of the streams the bits exceed the limit for that time. If that were the case pick the stream that the overflow occurs in, Since the sum was below the required before the stream started the bit rate of the stream $\frac{b_i}{t_i}$ must have been greater than r. In this case by the end of the stream the total bits would have only continued to exceed the maximum and thus at the end of the stream it could be checked and determined that the bit limit was exceeded. Thus if every endpoint is ok, every other point is also ok. If the program instead returns false, assume for contradiction that there is a possible ordering $S' = (s'_0, s'_1, ..., s'_n) \neq$ the sroted array in my algorithm. I propose that at each time step in this ordering the amount of bits sent is at least as many as my sorted array. Indeed consider the first time that the proposed ordering differs from mine. Let j be the index of S' that this occurs and let S by my ordering and B the total bits that both of our orderings have sent up until that point. Then at the next time step the total amount of bits sent by my algorithm is $B + \frac{s_j}{t_j}$ compared to $B + \frac{s'_j}{t'_j}$ by the sorting of my list, my sum must be less or equal. Inductively at every time step my algorith must have less than or equal bits sent out because the rates of bit addition is minimal by my sorting. Thus at the point where my ordering fails, the other proposed ordering must fail a contradiction and if my algorithm returns false then there truly is not possible ordering.
My algorithm runs in $O(n \log(n))$ time. It runs a $O(n \log(n))$ sort at the beginning of the algorithm once It then loops through the array performing an $O(1)$ operation at each point for a total of $O(n)$ time. Thus the total is the sum which is $O(n \log(n))$

4. Exercise 3 P 189

Let $S = \{s_1, s_2, ..., s_n\}$ be the ordering as described by the greedy algorithm wtih each

$$s_i = \{x_{i,1}, x_{i,2}, ..., x_{i,m}\}.$$

with each x_i being a package. and let $S' = \{s'_1, ..., s'_k\}$ be the alternative packaging scheme. Assume for contradiction that $k < n$. Consider the first element of s_1 and s'_1. They must contain the same element because $x_{1,1}$ must be in both. in particular $1 \leq 1$. for each extra package if it fits it will go into the same truck in S but may or may not go into the truck in S' so since packages start in less than or equal trucks, at each step the packages also go into trucks of index less than or equal to those in the newly proposed ordering. Inductively this holds for every package so the last package must go into a truck less than or equal in index to the final truck of the new ordering which is a contradiction since S' uses fewer trucks. So by contradiction S is optimal.

5. Exercise 6 P 191

I propose the following algorithm

- `Sort the players in terms of decreasing time to do the combined bike and run`
- `pick the players in order for longest to shortest time to bike and run`

I will prove the optimality of my solution by showing that is is better than any other solution. Consider any solution with two competitors x_i, x_{i+1} such that $x_{i,r} < x_{i+1,r}$ (the time to run and bike for x_i is less than that of x_{i+1}) and consider swaping the two comeptitors. Every competitor before x_i and after x_{i+1} will start at the same time and thus end at the same time.
in the original sorting x_{i+1} would have finished after x_i since it starts running abd biking after x_i and takes longer to do both. It suffices to show that in the swapped sorting both x_i and x_{i+1} finish faster than x_{i+1} in the original sorting.
Indeed since x_{i+1} starts earlier than it did in the original sorting it will finish earlier additionally x_i will start biking at the same time x_{i+1} would start biking in the original ordering and since it is faster at biking and running it will finish faster. Thus swapping two competitors so as to comply with my proposed ordering can only increase the optimality of a solution and thus starting with any solution executing multiple swaps to get to my ordering will only decrease total running time and so my ordering is optimal.

6. Exercise 6 From Handout

- `Pass through the matrix one time populating a weighted graph` G `with a node for each orange`
- `add an edge for each adjacency relation with weight one and label all roten oranges with distance 0`
- `Keep track of all visited (rotten) oranges and the minimum distance to them and other nodes in a priority queue`
- `Run Dijkstra's algorithm on this graph until all oranges have been found`
- `If all oranges have a path to a rotten orange return the node with the largest value (distance) if not return that it is impossible to rot every orange`

This algorithm works as specificed because every possible way a orange can turn rotten is contained in the edges of the graph including the time it owuld take for the rotting to happen Additionally, by nature of Dijkstra's algorithm it is guaranteed to find the shortest path since all edges have positive weight.