CS180 Midterm

Xilai Zhang

TOTAL POINTS

55 / 100

QUESTION 1

1 Problem 1 5 / 25

✓ - 20 pts partial credit (incorrect answers; use some sorting, but wrong answer, e.g.n^2 method; or understand it as interval scheduling; give a divide and conquer solution)

QUESTION 2

2 Problem 2 25 / 25

✓ - 0 pts Correct

QUESTION 3

3 Problem 3 5 / 25

✓ - 20 pts Moderate attempt. Misunderstood the problem and solved it as 0-1 knapsack or coin-change with only one coin per type. Note that in this problem (1) we have INFINITE multiplicity for each type and (2) we should fill up exactly volume V.

QUESTION 4

4 Problem 4 20 / 25

✓ - 0 pts (a). Correct

✓ - 5 pts (b). The algorithm has some minor error

CS 180: Introduction to Algorithms and Complexity
Midterm Exam

Feb 20, 2019

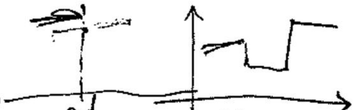
Name	Xilai Zhang
UID	804796478
Section	Friday 12-1:50

1	2	3	4	Total

★ **Print your name, UID and section number in the boxes above, and print your name at the top of every page.**

★ **Exams will be scanned and graded in Gradescope. Use Dark pen or pencil. Handwriting should be clear and legible.**

- The exam is a closed book exam. You can bring one page cheat sheet.
- There are 4 problems. Each problem is worth 25 points.
- Do not write code using C or some programming language. Use English or clear and simple pseudo-code. Explain the idea of your algorithm and why it works.
- Your answer are supposed to be in a simple and understandable manner. Sloppy answers are expected to receiver fewer points.
- Don't spend too much time on any single problem. If you get stuck, move on to something else and come back later.



Name: Xilai Zhang   UID: 804796478   CS180 Midterm Exam

1. A water utility has to adjust its pressure according to the maximum rate of flow any of the customers need at the time, i.e., at time 3 pm the pressure has to be proportional to accommodate the maximum flow rate among the customers flow-rate demands. The Utility wants to plan ahead for the next day. The clients are n companies. Each submits a triple (start-time$_i$, end-time$_i$, flow-rate-required$_i$), $i = 1, \ldots, n$. The output of the utility produces is a graph whose axis is time, say 12 AM to 11:59 PM of the pressure at any time t that corresponds to the maximum flow-rate-required$_i$ over all i such that start-time$_i \leq t \leq$ end-time$_i$. Since the function jumps from fixed value to another fixed value (piece-wise constant), it can be described by at most about $3n$ values just telling the next value at the next point of time the value switches to another value, and the time of the switch.

At perhaps the cost of sorting at the beginning, produce the graph of the function as described above for the Utility, incrementally proceeding from 12 AM to 11:59 AM.
The cost of your algorithm should be $O(n \log n)$. (25 pts)

sort the demands based on magnitude of flow rate, without loss of generality, assume after nlogn operations. flow rate is sorted so that $n_1 > n_2 > n_3 > \cdots n_i$.

look at start and end time of n_1, mark the period from $n_{1,start}$ to $n_{1,end}$ with n_1. Then look at start and end time of n_2, if some portion ~~of~~ from $n_{2,start}$ to $n_{2,end}$ has been marked, ignore that period, mark the rest of the period as n_2.

For each n_k of $\{n_3, n_4 \cdots n_i\}$: (sequentially)

look at the period of $n_{k,start}$ to $n_{k,end}$, ignore the portion that has already been marked on the graph, mark the portion that has not been marked on the graph with n_k.

This gurantees the maximum at each time as we always find the previously uncovered period for the current maximum.

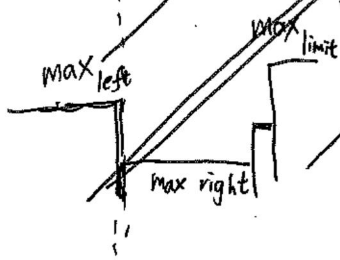
pairwise n^2

Name: Xilai Zhang UID: 804796478 CS180 Midterm Exam

2. Same as the problem above only that now you solve the same problem with the same complexity using divide-and-conquer. (25 pts)

Suppose two periods are ~~sorted~~ graphed correctly on the left and right side (of split) respectively

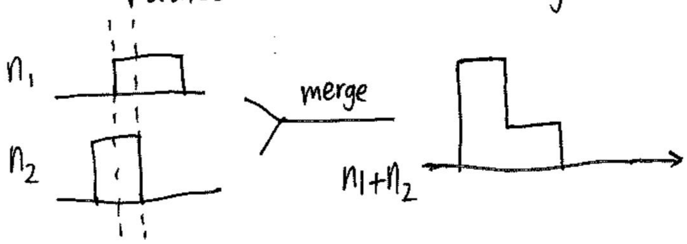
① if the max on the right Max_{right} is larger than max_{left}, extend max_{right} to the left until time becomes earlier than the start time of Max_{right}, ~~or until there~~ ignore the period when there exists a max_{limit} on the left such that $Max_{limit} > Max_{right}$.



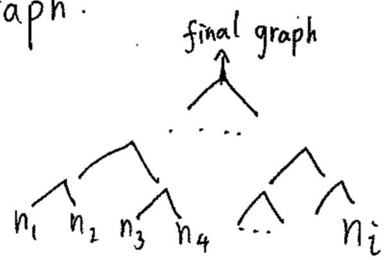
② if max on the left $Max_{left} > Max_{right}$, extend max

Base case: if there ~~are only two~~ is only one companies, graph ~~them be~~ its start time, end time, max value over the entire time.

For every two correct graphs over the entire period, compare the max values at ~~eat~~ each shift, keep the maximum and merge.



start from ~~sin~~ graphs of single companies on the entire time range, keep merging by groups of 2, after $\log_2 n$ rounds we will have the graph.



at each round there are at most $O(N)$ shifts so the complexity is $O(N)$. The complexity in total is $O(N \log N)$.

3. You are given n item types $x_1, \dots, x_n$ each of integer volume value, and each type has infinite multiplicity (as many items of the type as you wish). In addition to a volume, an item of type x_i has a weight $w_i > 0$. Item of type x_1 has a volume 1. You are asked to fill a knapsack of integer volume V to carry a total of V cumulative volume of items, but you want to minimize the total weight you carry.

   Give a pseudo-polynomial algorithm to solve the problem. Write the recursion, and argue that it is amenable to Dynamic-Programming treatment. Outline your algorithm and analyze its complexity. (25 pts)

   For an added protection, if you did not solve the problem or just made a mistake, you will get partial credit for naming the problem by a name that you might have heard for the case when $w_i = 1$ for all types.

~~opt (v) =~~ let V_i and W_i be the volume and weight of X_i, respectively.

$\cancel{opt(n, v) = opt(n-1, V), opt(n, V - W_n)}$

$$opt(n, V) = \min[opt(n-1, V),\ opt(n, V - V_i) + W_i]$$

look at $X_1 \cdots X_n$ — volume limit — minimum among — look at $X_1 \cdots X_{n-1}$, not take X_n — take X_n, volume limit decrease by V_i, Weight increase by W_i

to dynamic programming:

n \ V	0	1	2	...	V
opt(0,V):	∞	∞	∞	∞	∞
opt(1,V): $\longrightarrow$					
$\downarrow$ $\longrightarrow$					
$\longrightarrow$ opt(n,V)					

fill opt (0, V) to be all infinity, then fill the opt[n][v] table by row, ~~until~~ for each row, fill from left to right. the answer is at the lower right corner opt (n, V).

The complexity to fill the entire table is $O(mV)$.

added protection: continuous knapsack problem

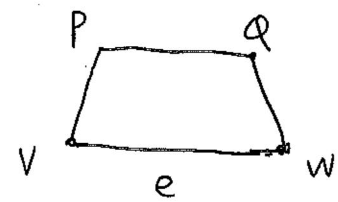
4. Given a connected undirected graph $G = (V, E)$, with edge costs that you may assume are all distinct. A particular edge $e = \{v, w\}$ of G is specified. Give an algorithm with running time $O(|V| + |E|)$ to decide whether e is contained in the unique (why?) minimum spanning tree (MST) of G, or not. Notice that the complexity required is too low to produce the MST and check whether e is in it, or not.

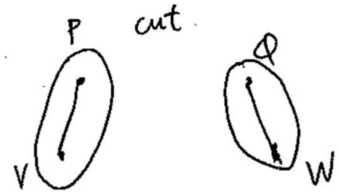
   (a) Give a property of the edge that determines if and only if the edge $e = \{v, w\}$ is in the MST. (10 pts) (Hint: Recall that we have seen in the homework that a MST is also the lexicographic MST and therefore in the MST, the unique path between v and w is the lexicographically smallest path.)

   (b) Give an algorithm and argue it is of the complexity required. (15 pts)

$\boxed{\text{also on the back of this page}}$

(a)





the edge VW is bottleneck smaller than all other paths from V to W. Namely, VW is smaller than the largest weighted edge on any path from V to W. To prove if and only if:

(i) VW is on MST but not bottleneck smallest.

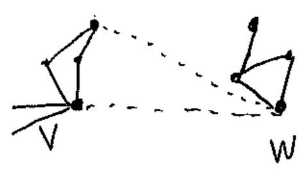
Assume another path $V \to P \to Q \to W$ has $PQ < VW$, then VW is not the smallest weight edge between cut P, V and Q, W, thus VW is not on MST, ~~contradi~~ contradiction.

(ii) VW is bottleneck smallest but not on MST

suppose MST connects V and W through another path $V \to P \to Q \to W$ has its largest weighted edge $PQ > VW$, then PQ is not the smallest weighted edge between cut P, V and Q, W, thus PQ is not on MST, contradiction.

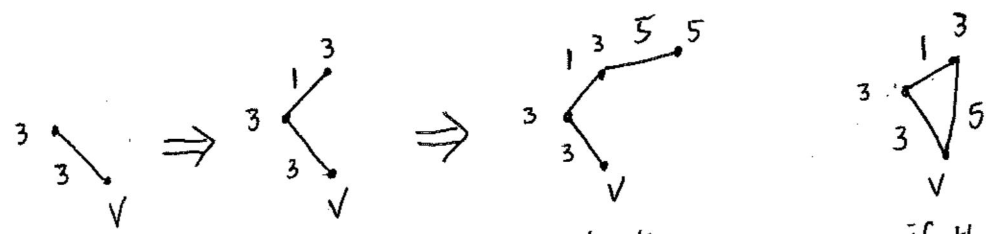
Thus, we know edge $e = \{v, w\}$ has to be bottleneck smallest among all paths from v to w, iff vw is on MST.

(b)



add edges to vertex V, (except VW), the added edges will give a connected component around V. If after an edge is added to the connected component of V, its other endpoint is also in the connected component we keep only the smaller of the bottleneck weight edge, otherwise we keep track of the current bottle neck weight edge.

illustration :



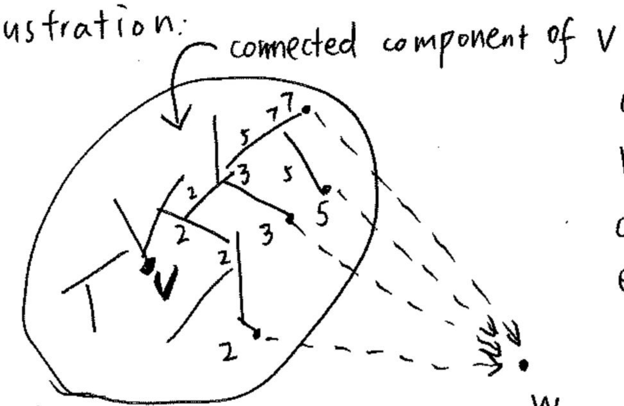
keeps 3 because 3 > 1

make the new value 5 because 5 > 3

if the endpoint is also in connected component, keep the smaller one, here we keep 3 because 3 < 5

now we grow the connected component of v to be big enough but also keeps track of the bottleneck weight edge, ~~until there~~ If we ever encounter an edge from the connected component of v to w, we record this value and compare it to e, if this value is smaller or equal to e, than $e=\{v,w\}$ is not on the unique MST. ~~Afte~~ If all edges are added and there is no such value, then $e=\{v,w\}$ is on the unique MST.

illustration:



compare all the bottleneck weight edge with vw after we discover a connection between connected component of v and w. at each endpoint is the current bottleneck weight edge

Since we iterate through each vertex and edge exactly once, the total complexity is $O(V+E)$.