CS180 Midterm

Jason Less

TOTAL POINTS

54 / 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 20 / 25

✓ - 5 pts Partial credit (merge part unclear or incorrect)

QUESTION 3

3 Problem 3 22 / 25

✓ - 0 pts Correct

✓ - 3 pts No time complexity analysis/Wrong time complexity analysis

QUESTION 4

4 Problem 4 7 / 25

✓ - 7 pts (a) wrong property

✓ - 11 pts (b). Wrong algorithm

CS 180: Introduction to Algorithms and Complexity
Midterm Exam

Feb 20, 2019

Name	Jason Less
UID	404-640-158
Section	Shirley

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.

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$_i \le t \le$ 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)

(s_i, e_i, v_i), n items

[(0,2,1), (0,2,2), (1,3,1), (2,4,1)]

sort by fin time (increa.)

- A brute force solution is trivial => for each tuple (s_i, e_i, v_i) just iterate from s_i to e_i in rs array and increment each index by v_i
  - This is an O(N^2) solution
- Instead can set up some temporary values in an array where each index is a tuple corresponding to end time and the value to increment by
  - After this step, we will have an array where each elem corresponds to the start time and the tuple indicates how far to iter into rs array and how much to incr. each index by => O(N)
- After, just iter through the temp array, to create the rs array

Ex: [(0,2,1)(0,2,2)(1,3,1)]

temp: [_, _, _, _]

(0,2,1) => [(2,1), _, _, _]

(0,2,2) => [(2,3), _, _, _]

(1,3,1) => [(2,3), (3,1), _, _]

rs: [0, 0, 0, 0]

[3, 1, 1, 1]

Alg: write algorithm(jobs of tuples (s_i, e_i, v_i))

for each job in jobs:
  go to temp[s_i] and place tuple (e_i, v_i) there
for each i in temp:
  iter from i to e_i in rs array and incr by v_i
return rs

- The arr returned corresponds to index being the time and arr[i] being the max flow at that time
- Just iter through arr to plot the graph

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

[(0,2,2),(0,2,1),(1,3,1),(0,4,1)]

[(0,2,2)(0,2,1)] [(1,3,1)(0,4,1)]

[(0,2,2)] [(0,2,1)] [(1,3,1)] [(0,4,1)]

- This problem can be solved by divide and conquer by continuously halving the input arr until just one company tuple (s_i, e_i, v_i) remains
  - At this step, it is trivial => just iterate from start time to end time and increment each position in the output arr by the flow value
- There will be log n levels, and at each lvl, n work is done => O(N log N)

```
waterUtility ( arr jobs containing tuples (s_i, e_i, v_i))
    sort by decr. end time to get final end time
    create arr rs of size end time
    call Helper ( jobs, rs)

Helper(jobs, rs)
    if just one job to process
        iterate from that job's start to end time and incr corresponding indices of rs arr
    else
        Helper (1st half of remaining jobs, rs)
        Helper (2nd half of remaining jobs, rs)

    return rs
```

3. You are given n item types $x_1, \ldots, x_n$ 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.

n items : (v_i, wt_i), fill knapsack w/ vol V but min wt

[(1,1)(2,2)(5,3)(4,4)] V=6 => 1+1+4 => wt = 1+1+4 = 6 v = [1,2,5,4]

or 1+5 => wt = 1+3 = 4 wt = [1,2,3,4]

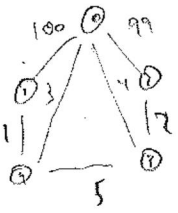
2+4 => wt = 2+4 = 6

- This problem can be solved using 2D Dynamic programming
- This is an extension of the coin change problem, but instead of each item having wt=1, they have varying wts
- Also, can apply the combination sum problem to retrieve all combinations of the n items that add up to volume V (via backtracking recursion)
- Then, with each of those combos, compute the wts of each and choose the combo w/ the min wt

```
func(vals, wts, V)
  helper(vals, rs, V, 0)
  for each arr in rs
    calculate total wt to get combo w/ smallest wt
    if combo is new min
      update minWt
      replace combo w/ new min wt combo

  return minWt combo
helper(vals, rs, arr, i, V)
  if (V < 0) return
  if (V == 0) {
    add arr to rs
    return
  }

  for (i to vals.size)
    add vals[i] to arr
    helper(vals, rs, arr, V-vals[i])
    pop back from vals[i]
```



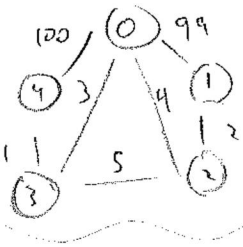
4. Given a connected undirected graph $G = (V, E)$, with edge costs that you may assume are all distinct. A particular edge $e = \{v, w\}$ of G is specified. Give an algorithm with running time $O(|V| + |E|)$ to decide whether e is contained in the unique (why?) minimum spanning tree (MST) of G, or not.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

(a) . The MST is a tree rooted at some node s that has a path to all other nodes in the graph s.t. that path has min weight

- The MST can be found by several algorithms: Prim or Kruskal
  - Kruskal sorts the edges by incr. weight and then uses the Union find algorithm to add edges to the unique MST set
  - However, this algorithm runs in $O(m \log m)$ time complexity, so not really useful here
- In class, we learned about a few graph traversal algorithms that run in $O(|V| + |E|)$ time
  - BFS, DFS, topological sort, is graph bipartite
  - The last two are irrevelant, but DFS could be useful
- Basically, if an edge has the min. weight in the entire graph, then it is guaranteed to be in the MST
  - This is because from the root s to some pt $v \in V$ not s, it will be useful to connect on the shortest path to such a node
- Take the graph =>



- Here the edge 3-4 has min wt = 1
- If not 0, then this edge isn't useful for 0-4 or 0-4-3
  - However, it will be for 0-3 and 0-3-4
- This can be said if rooted at any pt in the graph

(b) Algorithm => idea is to DFS from any pt in the graph and record the edge wt. Only keep the min wt edge encountered

```
findMinEdge(g, src)
    keep track of tuple edge (u,v) and min wt encountered
    DFS from src node
    if new min wt encountered, then
        update min wt
        replace edge (u,v) w/ new min wt edge
```