Homework 6

Tejas Kamtam

305749402

Discussion 1B - Friday, 10 am

TA: Parshan

Problem 1

> Exercise 19 on page 329

Algo

- Given the signal s of length n, create repetitions of each signal such that the repetitions are length n, we call these x', y' respectively
- Initialize $OPT[0,0] = True$ s.t. the indices i, j refer to the position in the signal and are used to determine the position of the memoized array
- For each subsequence length of the signal, l, in the range of n, do the following:
  - loop over all pairs i, j s.t. $i + j = l$ and for each pair:

$$OPT[i,j] = \Big((s[i+j] = x'[i]) \wedge (OPT[i-1,j]) \Big)$$

$$\vee \Big((s[i+j] = y'[j]) \wedge (OPT[i,j-1]) \Big)$$

- Return $OPT(i,j)$ s.t. $i + j = n$ else return `False`

Time Complexity

- looping over the length of the signal is $O(n)$
- for each iteration, we loop over all integer pairs that meet the condition which can be at worst also $O(n)$
- thus, the overall algo is $O(n^2)$

Proof by Induction

- The non-trivial part of our algorithm is the memoization which we need to prove is an inductively exhaustive case search
- Base Case: $n = 1$
  - if $x' = s$, then $OPT[1, 0] = True$ which is correct
  - if $y' = s$, then $OPT[0, 1] = True$ which is correct
  - else, $OPT[1, 0]$ and $OPT[0, 1]$ are False which is correct
- Inductive hypothesis: we assume our algo works correctly for a signal of length $n - 1$
- Inductive step: $n = n$
  - we observe that if the signal interleaves the signals, its last character (working backwards in this example) will be either from x' or y' where these variables refer to a sequence length repetition of the signal emissions
  - if the seq is interleaving the signals, sps. the last character in the string (seq) is from signal x , then that character will be found in x'
  - then, we know that $s[i + j] = x'[i]$ (because i tracks the position in x')
  - thus, $OPT[i, j]$ is true only if the above is true AND the value at $OPT[i - 1, j] = True$ (for the received sequence to ensure the values at positions $< i$ are from x')

    - a symmetric observation exists for the last signal emitting from signal y
    - if none of these are true, then this character of the string must not have been from either signal, and $OPT[i, j] = False$
    - these cover all the possible cases and each is correctly observed, thus our algo must be true inductively

Problem 2

> Exercise 22 on page 330

Algo

- Given a graph $G = (V, E)$ and we want to find the number of shortest paths between $v, w \in V$, we initialize the memoization $OPT(x, y, n)$ to represent the optimal cost of traveling from $x \to y$ in n steps
- We also initialize $NUM(x, y, n)$ to store the number of shortest paths $x \to y$ possible in n steps
- We initialize these objects for the source s.t.
    - $OPT(x, v, n) = 0$ for all $x \in V$
    - $NUM(x, v, n) = 1$ for all $x \in V$
- Then for all valid n, we can find the following:

$$OPT(v, y, n) = \min_{x \in V,\, (x,y) \in E} \{\infty,\ w_{x,y} + OPT(v, x, n-1)\}$$

$$NUM(v, y, n) = \sum_{x \in V, (x,y) \in E}$$

$$\begin{cases} NUM(x, y, n-1) & \text{if } OPT(v, y, n) = w_{x,y} + OPT(v, x, n-1) \\ 0 & \text{otherwise} \end{cases}$$

- The optimal solution has a cost of $W_{\min} = \min_n \{OPT(v, w, n)\}$

- Then, we can find the number of shortest paths $v \to w$ as

$$NUM(v,w,n) = \sum_n \begin{cases} NUM(v,w,n) & \text{if} \quad OPT(v,w,n) = W_{\min} \\ 0 & \text{otherwise} \end{cases}$$

Run Time

- for each valid path length (# of steps), we may look at at worst some finite $O(n)$ steps
- then for each iteration, our recurrence calculates the minimum across all $n-1$ path lengths which is also $O(n)$
- the count sums are also $O(n)$
- thus, overall the algo is $O(n^2)$

Proof

- We must prove our recurrence relation is correct, so by induction:
- Base Case: $n = 1$ steps
  - Our algo finds $OPT(v,w,1) = w_{v,w} + 0$ which correctly finds the shortest path length
  - NUM is then updated to 1 in the case selection in the sum
  - this is correct
- Inductive hypothesis: Our algo correctly works for $n-1$ steps
- Inductive Step: $n = n$ steps
  - Our OPT has 2 cases, to take infinity or the OPT for $n-1$ steps plus the value of the edge to w (either there is at least one path with 1 more edge that leads to w or we reach a dead end for n steps path)
  - This is an exhaustive consideration for our OPT because we consider all paths across all nodes

immediately before w and so on continuously; and, this is correct because we are able to represent the problem for n steps in $n-1$ steps which we know to be true by the inductive hypothesis
    - Similarly, the NUM is updated by the sum of the total number of paths that lead to w in $n-1$ steps or 0 if it is a dead end
    - this is an exhaustive consideration and is correct because the number of paths to w in n steps is the same as the overall number of paths to any nodes immediately before w and so on recursively
    - So our recurrence relation is correct
- We also return the number of paths as the sum over all possible number of paths that are conditioned to have the same optimal path length, which is correct

Problem 3

> Exercise 24 on page 331

Algo

- Given n even precincts each with m even possible votes, we can consider A to be the majority party in the first go (and run it again to check for B)
- We define A as the number of votes party A received total across all precincts s.t. party A has a majority iff $A > \frac{mn}{2}$
    - within this majority, A wins both districts if A's votes in district 1, $A_{d1} > \frac{mn}{4}$ and district 2, $A_{d2} > \frac{mn}{4}$ s.t. $A_{p1} + A_{p2} = A$
- Thus the distribution is susceptible to gerrymandering if $A_{d'} - \frac{nm}{4} > A_d > \frac{nm}{4}$ where d is some district 1 that consists of $k = n/2$ precincts and d' is the complementary district 2

- Then, we can define the recurrence relation which can be used to memoize across all valid tuples in the operands of the recurrence relation for p the number of precincts, k the size of district 1 in terms of number of precincts on the iteration, A_p the number of votes for party A for the number of precincts on the iteration:

$$OPT(p,k,A_p) = \begin{cases} True & \text{base case: } p=2, k=1, A_p = A_d \\ True & \text{if } OPT(p-1,k,A_p) \\ True & \text{if } OPT(p-1,k-1,A_d-A_p) \\ False & \text{otherwise} \end{cases}$$

- We assign this by looping through the memoized matrix: looping p through the range of n, then inner looping k through the range of n, then inner looping A_p through the range of m, and inner operations on the recurrence relation that looks at values of previous operands up to the range of n
- Finally, we can check if $OPT(n, \frac{n}{2}, A_p)$ for all values of A_p s.t. $A_{p'} - \frac{nm}{4} > A_p > \frac{nm}{4}$
- We return true if any of those memoizations are true

Run Time

- All our loops are nested:
  - the outermost: $O(n)$
  - the second: $O(n)$
  - the third: $O(m)$
  - the innermost operations call recurrence on n: $O(n)$
- Thus, overall algo has time complexity of $O(mn^3)$

Proof

- The algo is an exhaustive case analysis

- We discuss the bounds in the algo above and check that our recurrence relation is correct:
    - the base case is correct since for 2 precincts, the majority would either have or not have the majority in both districts since each is made up of 1 partition
- the inductive cases are true because:
    - if we have the majority for 1 fewer precincts and we include 1 fewer precincts in our consideration, then we still have the majority
    - if we decrease the number of precincts in our consideration by 1 and maintain the size of our districts, if this is also true, then our evaluation at the current iteration is true
    - these are all the cases, so any other case is false
- we also check to ensure we are only returning the output if it is within the bounds so our return is good.

Problem 4

> Exercise 7 on page 417

Algo

- similar to the cell-phone tower problem in class, we want to create a directed graph $G = (V, E)$ with a source, s, and sink, t
- for each of the n clients, create a directed edge $s \rightarrow c_i$ with weight 1
- then for each of the clients, connect an edge from the client to each of the k base stations the client is in the radius of s.t. the directed path $c_i \rightarrow b_j$ has weight of 1

- Finally for each of the k base stations, create a directed edge $b_j \to t$ with a weight of that base station's load parameter
- Now run Ford-Fulkerson and find the max-flow
- If the max-flow, f, is equal to n the number of clients, then return true that every client can simultaneously be connected otherwise return false

Run Time

- creating the graph requires at worst $O(nk)$ time if each client is connected to every base station (this also means at worst $E = nk$ and $V = n + k + 2$)
- running Ford-Fulkerson is $O(|f| \cdot (V + E))$ where f is the max flow
- in our case the max-flow is at most the number of clients n, so the overall time complexity is at worst $O(n^2 k)$

Proof

- The proof of the Ford-Fulkerson algo is discussed in lecture. We must, however, prove that our graph is a valid max-flow problem and can be made in poly time (which we showed in the run time analysis)
- The graph we constructed is indeed a network flow graph because each edge is directed, there is a capacity ≥ 1 on each edge, and there is a clear source and sink
- this is the correct representation because there exists an edge between all clients and base stations only if the clients are within the range parameter of the base stations and thus accessible
- all of these edge weights are 1 so that each edge has a capacity (an edge weight 0 is akin to removing it from the graph altogether)

- the edges from the base stations to the sink however have a weight equal to the base stations load parameter, this ensures each base station can only output as much flow as their load parameter and thus only accept that many clients due to the conservation of flow property of the network
- Finally, BWOC, consider our flow network is not correct and states false when in fact the state is true
- this means we must have not been able to send over the maximal flow equal to n, this is only possible if we missed creation of an edge, but this contradicts that our graph is correct and makes edges for all those in range
- or, the capacity of an edge is wrong, because the only non-trivial capacities are fro mthe base stations to the sink, these are determined by the load factor. Thus we could have only output false if our loads were not saturated.
- However, this contradicts that our graph is correctly connected as we should have n flow entering the "layer" of our base stations

Problem 5

> Exercise 9 on page 419

Algo

- similar to the previous question we want to create a directed graph $G = (V, E)$ with a source s and sink t
- We draw a directed edge $s \to p_i$ of capacity 1 to each of the n injured people
- then we draw an edge $p_i \to h_j$ of capacity 1 from each injured person to each of the k hospitals IFF they are in the 30 minute range of the person

- finally, we draw edges $h_j \to t$ of capacity n/k that we are trying to maintain
- Running Ford-Fulkerson, if the max-flow is equal to n then we return true since each person is able to reach a hospital in range while maintaining the balanced load factor of the hospitals
- else return false

Run Time

- constructing the graph is $O(nk)$ at worst if each person is in range of every hospital, implies $E = nk$ and $V = n + k + 2$
- running Ford-Fulkerson is $O(|f|(V + E))$ for max-flow f
- in our case the max-flow possible is n the number of injured people, so at worst the overall time complexity is $O(n^2k)$

Proof

- similar to the previous question, we've proved Ford-Fulkerson in class
- we also know this is a valid network flow representation by the symmetric considerations from the previous problem
- BWOC, sps our algo returns false when the true value is true, this means our algo either (1) missed creation of an edge or (2) placed the wrong load capacity on the edges from the hospital to the sink
- (1) is a contradiction because we could not have missed an edge if edges are only drawn to hospitals in range of injured people and all people are connected to the source and all hospitals to the sink
- (2) this is a contradiction since the load on the graph is precisely the load we want to check for balance and

is given by the problem statement, so all capacities are correct

Problem 6

> Given a sequence of numbers find a subsequence of alternating order, find the length where the subsequence is as long as possible. (That is, find a longest subsequence with alternate low and high elements).
> Example
> Input: 8, 9, 6, 4, 5, 7, 3, 2, 4
> Output: 8, 9, 6, 7, 3, 4 (of length 6)
> Explanation: 8 < 9 > 6 < 7 > 3 < 4 (alternating < and >)

Algo

- given a seq, s, of length n, initialize OPT to memoize the length of the sequence up to position i s.t. $OPT[i, G]$ represents the max subsequence length for a subseq ending on a value greater than the value before it and $OPT[i, L]$ represents the complement
- Then, for each sequence size up to n, do the following:

$$OPT[i, G] = \max_{j<i} \{1, 1 + OPT[j, L] \quad | \quad s[j] < s[i]\}$$

$$OPT[i, L] = \max_{j<i} \{1, 1 + OPT[j, G] \quad | \quad s[j] > s[i]\}$$

- At each calculation of OPT, save the character/integer we included to update our OPT at position i
- then for $m = \max\{OPT[n, G], OPT[n, L]\}$, trace back the sequence elements and output the sequence that led to m

Time Complexity

- We loop for each seq length up to n which is $O(n)$
- Then for each iteration we look at all values up to the subseq length which is $O(n)$
- tracing back is $O(2n)$ to find the subsequence which created the max
- thus, overall is $O(n^2)$

Proof

- the non-trivial proof occurs in the memoization of the arrays
- Base Case: $n = 1$
    - $OPT[1, G]$ and $OPT[1, L]$ both default to 1 as defined in their recurrence relation which is correct
- Inductive hypothesis: our algo works correctly for length $n - 1$
- Inductive Case: $n = n$
    - we observe that our maximal subarray will either end on a value that is greater or less than the value before it, so we have to track 2 possible sequences: G and L
    - for each, we observe that, for example WLOG if at position i we choose to update $OPT[i, G]$, then there are 2 cases:
    - we either choose to begin at that sequence, which will evaluate its recurrence max to 1, or we include the element at $s[i]$ and increment length for the corresponding value in the sequence $OPT[j, L]$ for all the possible values that could have come before it; with the condition that the value at $s[i] > s[j]$
    - this max takes the optimal value across all possible subsequences up to position i and maintains the condition for the value to be greater than the one before it

  - a symmetric observation exists if we choose the next value to be from $OPT[i, L]$
  - this exhaustively searches all possible cases (either G or L), and correctly does so QED
- We return the correct values as well by tracing back through OPT to find the actual sequence that contributed to the maximal length of the alternating subsequence