1	2	3	4	5
20	15	20	20	13

88

Name(last, first): [redacted]

UCLA Computer Science Department

CS 180 **Algorithms & Complexity** **ID (4 digit):** 2329

Midterm **Total Time: 1.5 hours** **October 24, 20101**

Each problem has 20 points.

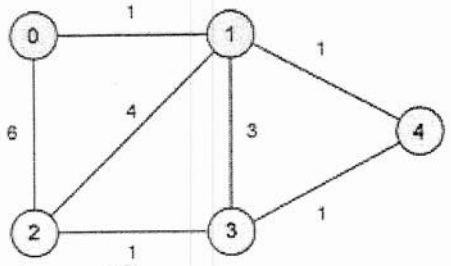
1 Describe Depth First Search (DFS) algorithm in an undirected connected graph (in English, bullet by bullet).

DFS(node of the graph):
Denote this node as r
For any neighbor n of r
If n is not marked as discovered
Apply DFS recursively on n
If r has no neighbors
Mark it as discovered

20

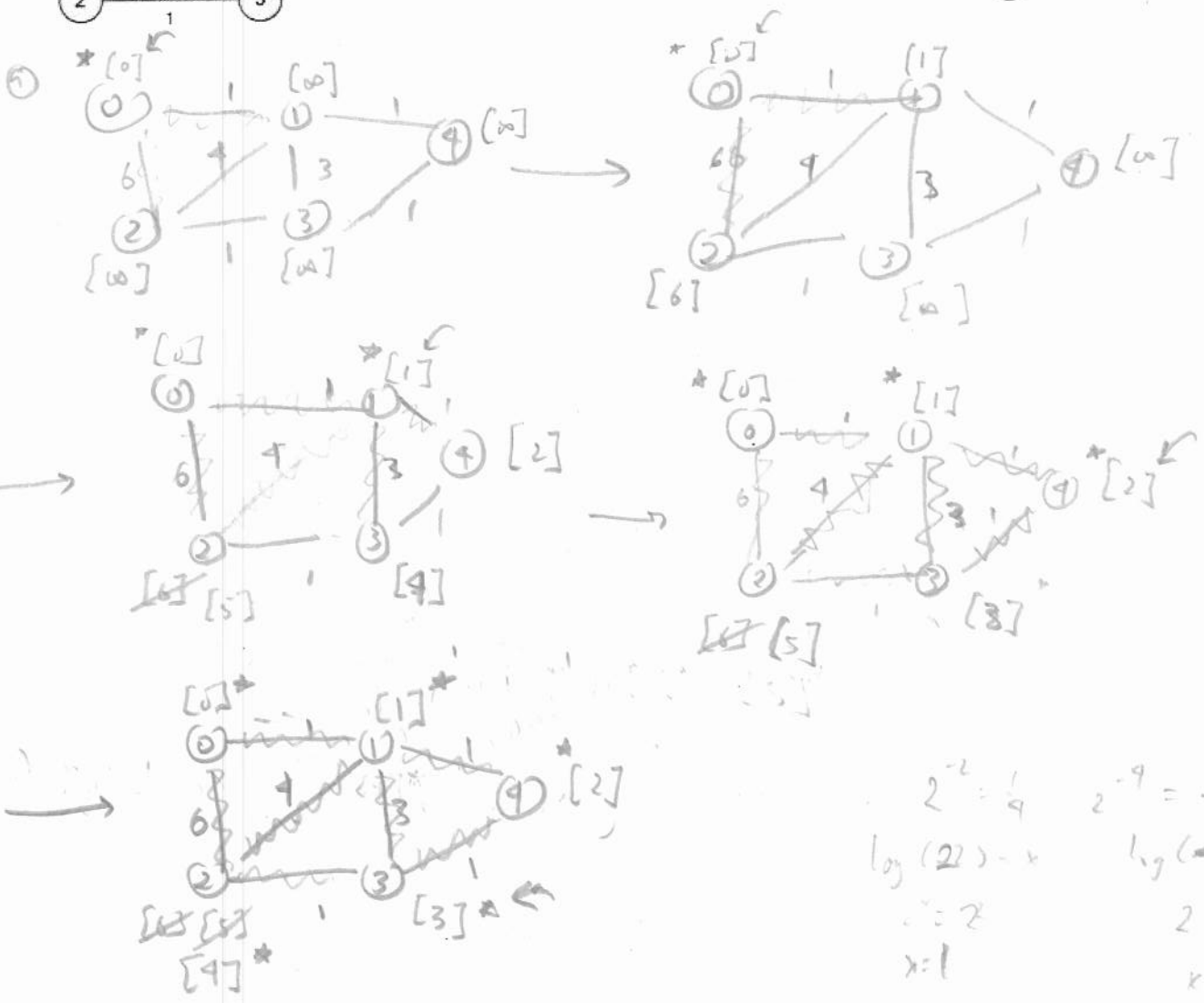
Name(last, first): [redacted]

2 . a. Use Dijsktra's shortest path algorithm to find a shortest path from vertex 0 to all vertices in the following graph. Show each step.
b. If all edges were negative of what they are shown below (e.g., -6 instead of 6) how would you find the shortest path?



* = done with this node
↖ = current node

(10)



+5 (b) Dijkstra's only operates on positive paths; a negative path would cause an infinite cycle. We can raise 2 by the power of the weight to make the weights positive and still preserve the relative ordering of weights. That is, ∀ weight w_i replace it with 2^{w_i}.

Example: $\{-1, -4, -5\} \Rightarrow \{2^{-1}, 2^{-4}, 2^{-5}\} = \{\frac{1}{2}, \frac{1}{16}, \frac{1}{32}\}$. The relative weight is still preserved. Then apply Dijkstra's. The path will be minimized.

$2^{-2} = \frac{1}{4}$ $2^{-4} = \frac{1}{16}$
$\log(2) = x$ $\log(4) = x$
$x = 1$ $2^x = 4$
$x = 2$

Name(last, first): [redacted]

3. Consider a sorted list of n integers and given integer L. We want to find two numbers in the list whose sum is equal to L. Design an efficient algorithm for solving this problem (note: an $O(n^2)$ algorithm would be trivial: considering all possible pairs).

List = our list of numbers
i = 1
j = n
Sum = 0
While (i ≠ j)
    Sum = List[i] + List[j] - L
    If (Sum < L)
        Increment i by 1
    If (Sum > L)
        Decrement j by 1
    If (Sum is L)
        We are done and our numbers are List[i] and List[j]
The numbers do not exist.

(20)

Notes
For my algorithm, I assumed that the list can be called List, and that it uses 1-based indexing.

Example: List = {a, b, c, d}
So, n = 4
List[1] = a
List[n] = d

Name(last, first): [redacted]

4. Consider a sequence x1, x2, ... , xn of (positive and negative) integers. We want to find two indices i and j such that xi + ... + xj is maximized. Describe in English (bullet by bullet) an O(n) time algorithm for solving this problem. For example, if the input is (-2, -2, 5, 7, -3, 4, -4) then i=3 and j = 6 (and the sum xi+…+xj is equal to 13).

Algorithm

Assume i is 1

Assume j is 1

Assume the current max is 0

Assume the old max is 0

Look at each item, x_n, in the list (in order).

Set the newly max to the current max.

Add x_n to our current max.

If the current max is less than 0.

Set the current max back to 0.

Set i to 1 plus the position of x_n (x_n's position is n)

If the current max is greater than the old max

Set j to be the position of x_n (x_n's position is n)

If 0 is greater than n (this means we have all negative numbers)

Set i to 1

Set j to 1

Set the current max to be the value of the first element

Look at each item x_n in the list (in order)

If x_n is bigger than the current max

Set i and j to the position of x_n (that is n)

Set the current max to x_n

The indices we want are i and j.

+10

Trial run:

	-2	-2	5	7	-3	4	-4
i 0	1	1	3	3	3	3	3
j 0	0	0	3	4	4	6	
cm 0	0	-2	5	12	9	13	
om 0	0	0	0	5	12	9	

Name(last, first): [redacted]

5. a. Let G be a directed graph with n vertices. Design an efficient algorithm to label all vertices with distinct integers 1 to n such that the label of each vertex is at least one greater than the label of at least one of its predecessors, or to determine that no such labeling is possible.
b. Analyze the time complexity of your algorithm.

(a) This is only possible if the graph has no cycles: *

← In the cyclical case, neither node can have a valid numbering as each node increments its value each time its predecessor does—forever.

So, we must have a DAG. In the DAG, there is always a source.

Algorithm:

Counter = 1
While (∃ some source)
- Number it with Counter's value
- Remove it and its adjacent edges
- Increment Counter by 1

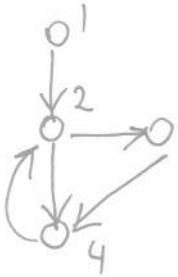
If (Counter == n+1)
  We are done
Else
  No such numbering is possible.

+5

(b) (Suppose our graph is modeled with linked lists)
This algorithm is similar to topological sort. Finding the source takes O(n) times each time although, where n = # of vertices. Removing it and rearranging pointers is an O(1) operation.
So, the algorithm runs in ~~O(n²)~~ O(n+m) time.

+8

(if source finding is efficiently implemented)



valid labeling

∃ cycle

*