Ouermi Timbwaoga Aime Judicael
course: CIS 315
Professor: Chris Willson
Homework7

1) Consider a greedy strategy for the following problem: We have a company with n workers. Worker w_i works a shift (s_i , f_i), where si is that worker's start time and fi the finish time. We want to form a small committee C $\subseteq$ {w_1 , . . . , w_n } with the following property: for every worker w_i there exists a worker $w_c \in$ C such that the shift of w_i overlaps with the shift of w_c . That is, the intervals (s_i , f_i) and (s_c , f_c) must intersect (they do not intersect if, say, $f_i = s_c$). So the problem is to find the smallest possible set C of workers whose shifts overlap with all workers.

(a) Describe the greedy choice. ("Choose the first worker with property P.")

- **sort the intervals work time by start time**
- **using a loop for a set of element overlapping we pick w_m such that f_m is the latest possible and we put in C. While we are going through the loop, we can use temporary variable w_m to keep track of the elements with latest finish among elements that are overlapping.**

(b) Show that if there is an optimal solution for which the greedy choice was not made, then an exchange can be made to conform with your greedy choice. ("Let schedule S use worker w_i who does not satisfy property P, and let wk be the worker that does. Here I show that the schedule S , which is obtained by exchanging worker w_i for w_k, is just as good as S ...")

Let C the solution we got from applying (a) S = {w_1, ... w_{k-1}, w_k, w_{k+1}, ... w_m}.

Let suppose that S is not an optimal solution because a greedy choice was not made at k. Let S' = {w_1, ... w_{k-1}, w_i, w_{k+1}, ... w_m} be an optimal solution. Assuming that a greedy choice has been made for {w_1, ... w_{k-1}} and {w_i, w_{k+1}, ... w_m} w_j must be overlapping with the elements that have not been covered but the greedy choices. Those elements are the same that are covered by the w_k S' is not a better solution than S. Therefore S is an optimal solution.

(c) Describe, in English, how to implement a greedy algorithm.

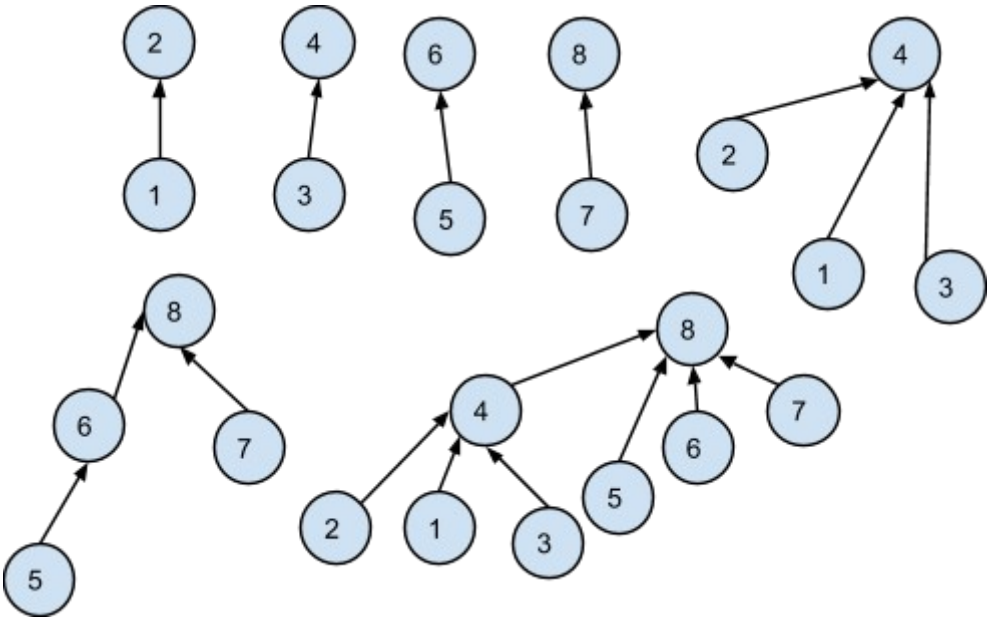
- **we'll sort all the element by start time**
- **we'll use a loop to make a greedy choice befoe we enter the loop we'll initialize k=1, temp= w_1, and C = empty**
- **for each step in the loop**
  - **if s[m]<f[k] and f[m]>f[k] then**
    update temp to W_m
  - **else**
    C = C U {temp}
    update temp to W_m

  when we reach the end of the loop we return C

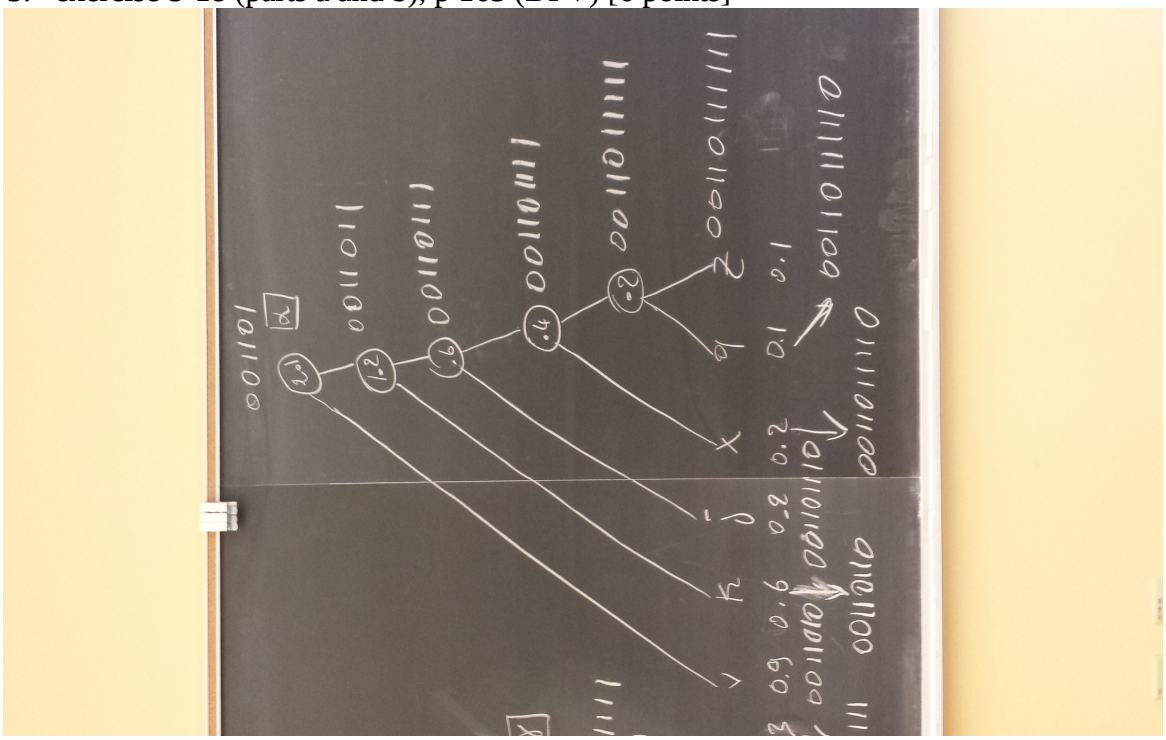
(d) How long would your algorithm take?

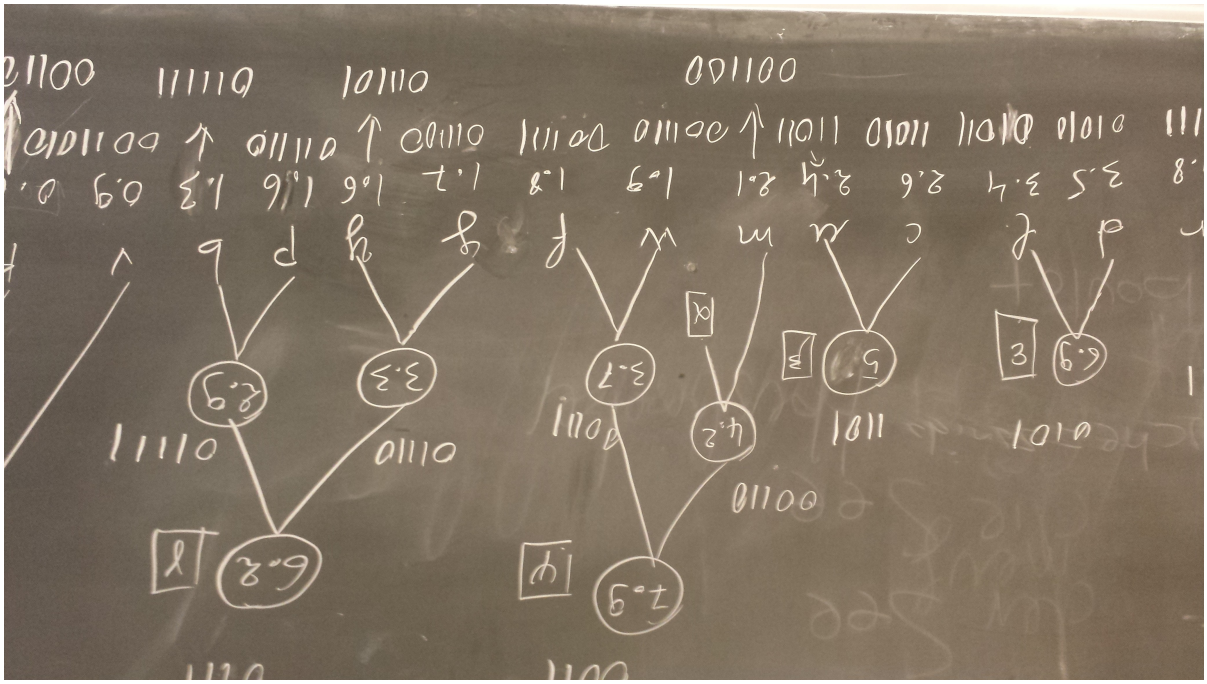
The algorithm will take O(nlog(n)) because the to sort the elements by start time it takes O(nlog(n)) and to make the choice it takes O(n) time. Therefore the time complexity is O(nlog(n))

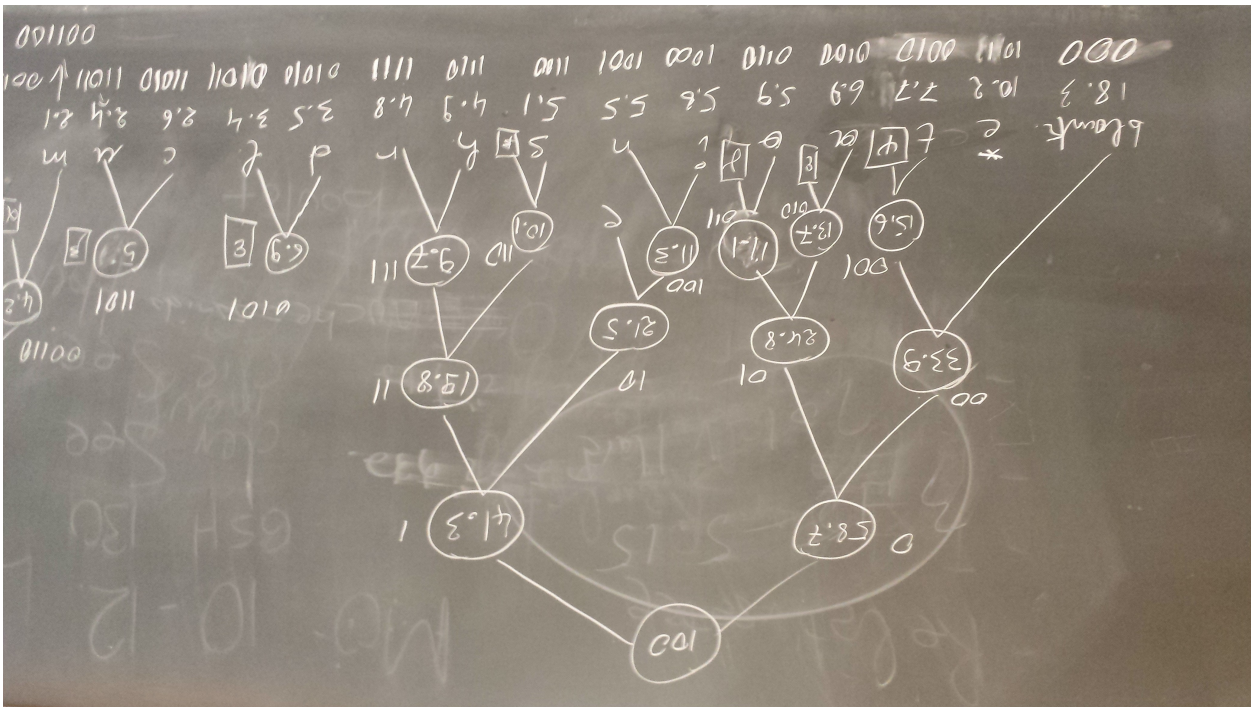
2)exercise 5.11Give the state of the disjoint-sets data structure after the following sequence of operations, starting from singleton sets {1}, . . . , {8}. Use path compression. In case of ties, always make the lower numbered root point to the higher numbered one.
union(1, 2), union(3, 4), union(5, 6), union(7, 8), union(1, 4), union(6, 7), union(4, 5), find(1)



3. exercise 5-18 (parts a and b), p 163 (DPV) [6 points]







(a) optimum coding of this alphabet
blank=000; e =1011; t=0010; a=0100; o=0110; i=1000; n=1001; s=1100; h=1110; r=1111; d=01010; l=01011; c=11010; u=11011; m=00110; w=001110; f=001111; g=011100; y=011101; p=011110; b=011111; v=0011010; k=00110110; j=001101110; x=0011011110; q=00110111110; z=001101111110;

(b) The expected number of bits per letter
The number of bits expected per letter corresponds to depth of the letter. For instance 11 bits is expected to represent z.

4. Here's a problem that occurs in automatic program analysis. For a set of variables x1 , x2 , . . .xn you are given some equality constraints of the form $x_i = x_j$ and some disequality constraints
of the form $x_i = x_j$. Is it possible to satisfy all of them?
For example, the constraints
x1 = x2 , x2 = x3 , x3 = x4 , x1 = x4

cannot be satisfied. Give an efficient algorithm that takes as input m constraints over n variables and decides whether the constraints can be satisfied. (Of course, make it efficient and give the time bound.)
We can solve this by creating sets of values with the same equality constraints an then check for the inequality constraints

- **For every x_i**
  MakeSet(x_i)
- **For every edges (x_i,x_j) such that x_i=x_j do**
  Find (x_i) U Find(x_j) // creates a set of of vertices with same equality constraint

- **For every edges (x_i,x_j) such that x_i!=x_j do**
  if(Find (x_i) = Find(x_j))
    set boolean to True
  else
    set boolean to False and break out of the loop

<u>Time complexity</u>

The Find(x) has a time complexity of log*(n) and we go through the loop m (number of constraint) times in the worst case. Therefore the time complexity of this algorithm is mlog*(n)