Bowen Kruse

CSCI 232

5/30/2019

Homework 5 - Algorithms

**Q1.**

Take the national pool of previously incompatible recipients, and break the population down regional pools.

Each organ will be ranked on criteria such as the following:

* Blood type
* Tissue type
* location

Each hopeful recipient will be ranked on the following criteria:

* Blood type
* Tissue type
* Location
* Size of the current possible donor pool

1. Rank Order List
   1. Each organ will rank patients based on preference created by blood/tissue compatibility, location of the possible recipient, and the number of organs available for a donation that would also be compatible.
   2. Each hopeful recipient will create a ranked list of patients based on blood type, tissue type, and location of the organ.
2. Attempt to match the organ to the first choice recipient, if the match is not guaranteed, the organ will be matched to the second choice and so on. All proposals are Constant O(1)
   1. Matches will be considered tentative to ensure better matches with other recipients in the future are possible.
   2. Direct matches will be definite.
   3. For a match to occur, both the organ and recipient must rank each other.
3. The algorithm favors the party that actively proposes the matches (organs in this case), and because of the organ pool to the recipient waiting list, the organ needs to find its best match.
4. Total time: O(n^2)

This problem can be represented in a computer:

function OrganDonerMatching {

Initialize all Organs and Hopeful recipients to free

**while** O free organ who still can pair with R recipient {

R = O's highest ranked such organ to whom it has not yet paired with

if R is free

(O, R) become matched

else some pair (O', R) already exists

**if** R prefers O to O'

(O, R) become engaged

O' becomes free

**else**

(O', R) remain engaged

}

}

**Q2.**

The algorithm below utilizes the mergesort algorithm and runs in O(nLogn) time

Function EquivalenceTester() {

* Divide the card set into C1 and C2 where C1 has floor (n/2) and C2 has the rest
* If EquivalenceTester(C1) returns a card:
  + Iterate over the card set and count how many cards are similar to it
  + If the count is greater than the card set’s half-length
    - Return that card
    - Else return nothing
* If EquivalenceTester(C2) returns a card:
  + Iterate over the card set and count how many cards are similar to it
  + If the count is greater than the card set’s half-length
    - Return that card
    - Else return nothing

}

The algorithm below utilizes the Boyer-Moore algorithm and run in O(n) time (Faster than merge sort type above)

Function EquivalenceTester() {

* First iteration:
  + Iterate through the array and maintain the count of Integer Majority
  + If the next element is the same
    - Majority++
  + If the next element is not the same
    - Majority--
  + If Majority becomes == 0
    - Majority element = current element
    - Count = 1
  + Majority Element == candidate that might have a count higher than half
* Second Iteration
  + Iterate through the array and get the count of the element found in the First iteration
  + If the count is less than half of the total card quantity, there is no card number that accounts for more than half the total card set.

}