# Faculty of Science
# Final Examination

## Computer Science 308-251B
## *Data Structures and Algorithms*

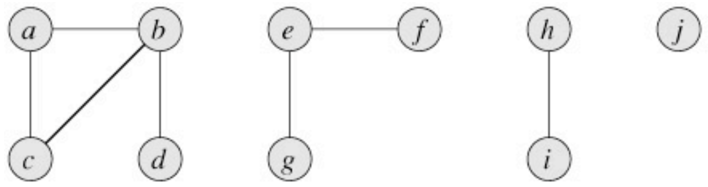
**Examiner:** Prof. Claude Crépeau **Date:** April 24, 2008
**Associate Examiner:** Prof. Patrick Hayden **Time:** 14:00 – 17:00

**INSTRUCTIONS:**

This examination is worth 50% of your final grade.
The total of all questions is 100 points.
Each question is assigned a value found in brackets next to it.
O P E N • B O O K S •/• O P E N • N O T E S
Faculty standard calculator permitted only.
This examination consists of 4 pages including title page.
This examination consists of 6 questions.

**SUGGESTION : read all the questions and their values before you start.**

[20%] **1)** Remember the application of the Disjoint-Sets data structure to find connected components of a graph and the related example we saw in class :



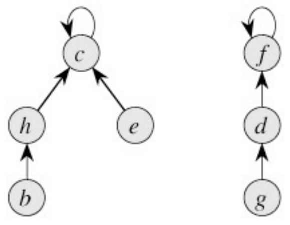
(a)

| Edge processed | Collection of disjoint sets | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| initial sets | {*a*} | {*b*} | {*c*} | {*d*} | {*e*} | {*f*} | {*g*} | {*h*} | {*i*} | {*j*} |
| (*b*,*d*) | {*a*} | {*b*,*d*} | {*c*} | | {*e*} | {*f*} | {*g*} | {*h*} | {*i*} | {*j*} |
| (*e*,*g*) | {*a*} | {*b*,*d*} | {*c*} | | {*e*,*g*} | {*f*} | | {*h*} | {*i*} | {*j*} |
| (*a*,*c*) | {*a*,*c*} | {*b*,*d*} | | | {*e*,*g*} | {*f*} | | {*h*} | {*i*} | {*j*} |
| (*h*,*i*) | {*a*,*c*} | {*b*,*d*} | | | {*e*,*g*} | {*f*} | | {*h*,*i*} | | {*j*} |
| (*a*,*b*) | {*a*,*b*,*c*,*d*} | | | | {*e*,*g*} | {*f*} | | {*h*,*i*} | | {*j*} |
| (*e*,*f*) | {*a*,*b*,*c*,*d*} | | | | {*e*,*f*,*g*} | | | {*h*,*i*} | | {*j*} |
| (*b*,*c*) | {*a*,*b*,*c*,*d*} | | | | {*e*,*f*,*g*} | | | {*h*,*i*} | | {*j*} |

(b)

Illustrate the execution of algorithm **CONNECTED-COMPONENTS(*G*)** on this example, from the initial collection of disjoint sets, to the final configuration, showing the evolution of the data structure after the processing of each new edge, in the context of the ***Disjoint-Set Forest*** representation together with the ***union by rank*** and ***path compression*** heuristics.

To be more precise about my expectations for this question, I expect a bunch of drawings like the following, representing the data structure after processing of each edge, as well as the table of ranks :



[15%] **2)** Prove using constructive induction that the following T(n) is Θ(n):

$$T(n) = \begin{cases} 1 & \text{if } n=1,2 \\ 2T(\lceil n/4 \rceil) + T(\lceil n/3 \rceil) + \Theta(n) & \text{if } n>2 \end{cases}$$

[15%] **3)** Consider the set of edges defined by a ***Single-Source shortest path*** algorithm from a given source s.

a) Show that these edges form a tree.

b) Explain why this tree is not necessarily a ***Minimum Spanning Tree*** and give an example where the Minimum Spanning Tree is distinct from the tree defined by the ***Single-Source Shortest Path*** algorithm.

c) Show that in any graph, there exists a source such that the two notions coincide.

[10pts] 4) For each statement, say if it is *true* or *false*.

**Correct = +1 pt, Incorrect = -0.5 pt, No answer = 0 pt, Minimum Total= 0 pt.**

**(a)** Let $f:N \rightarrow N$ be an arbitrary function of the natural integers to themselves, such that $f(n) \leq n$ for all $n \in N$. In an array of n unsorted integers we can find the element of rank f(n) in worst case time O(n).

**(b)** Any comparison-based sorting algorithm of an array of n integers of k bits each must use, in the worst case, at least $\Omega(k\ n\ \lg n)$ time if we take into account the time necessary to compare integers.

**(c)** For any $\varepsilon>0$, we know how to multiply two nxn matrices using only $O(n^{1+\varepsilon})$ time.

**(d)** We can always construct an optimal binary search tree by putting the key with the greatest probability at the root.

**(e)** The greedy algorithm for making change without nickels presented by professor Hayden doesn't necessarily minimize the number of coins returned.

**(f)** The longest common subsequence of "patato" and "tomato" is "ato".

**(g)** The Bellman-Ford algorithm is preferable to Dijkstra's when negative edges are present.

**(h)** A theoretical speedup obtained for matrix multiplication would translate to a speedup for matrix inversion.

**(i)** Any algorithm that solves the ***All-Pairs Shortest Path*** problem must use time at least $\Omega(n^3)$.

**(j)** The ***Longest Simple path problem*** is an example where the optimal substructure property does not apply.

[20%]

**5)** The ***edit distance*** of two strings, $s_1$ and $s_2$, is defined as the ***minimum*** number of *point mutations* required to change $s_1$ into $s_2$, where a point mutation is one of:

A. change a letter,
B. insert a letter or
C. delete a letter.

The following recurrence relations define the edit distance, $d(s_1, s_2)$, of strings $s_1$ and $s_2$:

$d(s, \text{“”}) = d(\text{“”}, s) = |s|$ -- i.e. length of s, where “” = empty string.

$d(s_1+ch_1, s_2+ch_2) =$

$$\min\left(d(s_1, s_2)+\begin{cases}0 & \text{if } ch_1=ch_2\\ 1 & \text{if } ch_1\neq ch_2\end{cases},\ d(s_1+ch_1, s_2)+1,\ d(s_1, s_2+ch_2)+1\right)$$

The first rule above is obviously correct, so it is only necessary to consider the second. Here, neither string is empty, so each has a last character, $ch_1$ and $ch_2$ respectively. Somehow, $ch_1$ and $ch_2$ have to be explained in an *edit* of $s_1+ch_1$ into $s_2+ch_2$.

If $ch_1=ch_2$, they can be *matched* at no cost, i.e. 0, and the edit distance is $d(s_1, s_2)$.

If $ch_1\neq ch_2$, then $ch_1$ *could* be changed into $ch_2$, i.e. cost=1, giving an overall edit distance $d(s_1, s_2)+1$. Another possibility is to delete $ch_1$ and edit $s_1$ into $s_2+ch_2$, yielding an edit distance of $d(s_1, s_2+ch_2)+1$.The last possibility is to edit $s_1+ch_1$ into $s_2$ and then insert $ch_2$, yielding an edit distance of $d(s_1+ch_1, s_2)+1$.

There are no other alternatives. We take the least expensive, i.e. min, of these.

A. Explain why $d(s_1, s_2)$ depends only on $d(s_1', s_2')$ where $s_1'$ is shorter than $s_1$, or $s_2'$ is shorter than $s_2$, or both.

B. Given two input strings, $s_1$,$s_2$, describe a ***dynamic programming*** algorithm to compute a table of the edit distances necessary to determine $d(s_1, s_2)$.

C. Analyze the running time of your algorithm.

[20%]

**6) Sorting. (Copyright © 1999 Luc Devroye.)**

The input is a sequence of n integers with many duplications, such that the number of distinct integers in the sequence is O(log n). Design a sorting algorithm (based on comparisons only) to sort such sequences using at most O(n log log n) comparisons in the worst case (justify the running time).

**Hint:** A red-black tree may be useful here!