# Ve203 Discrete Mathematics (Fall 2016)

## Assigment 10: Graphs

**Date Due: 4:00 PM, Thursday, the 8th of December 2016**

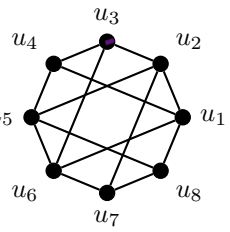
JOINT INSTITUTE
交大密西根学院

This assignment has a total of **(33 Marks)**.

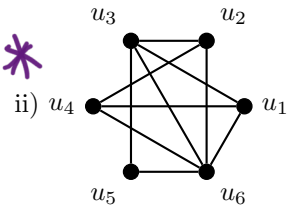
**Exercise 10.1**

Determine whether each given graph is planar. Either draw an isomorphic graph without crossing edges, or prove that the graph is non-planar.

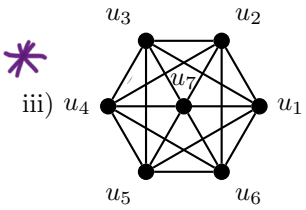
i)




ii)




iii)




**(6 Marks)**

**Exercise 10.2**

Show that a full $m$-ary balanced tree of height $h$ has more than $m^{h-1}$ leaves. Deduce the second statement of Corollary 3.4.16 regarding the height of full and balanced trees.
**(3 Marks)**

**Exercise 10.3**

One of four coins may be counterfeit. If it is counterfeit, it may be lighter or heavier than the others. How many weighings are needed, using a balance scale, to determine whether there is a counterfeit coin, and if there is, whether it is lighter or heavier than the others? Describe an algorithm to find the counterfeit coin (or establish its non-existence) and determine whether it is lighter or heavier using this number of weighings.
**(4 Marks)**

**Exercise 10.4**

Show that Huffman codes are optimal in the sense that they represent a string of symbols using the fewest bits among all binary prefix codes.
**(3 Marks)**

**Exercise 10.5**

Consider the three symbols A, B, and C with frequencies A: 0.80, B: 0.19, C: 0.01.

i) Construct a Huffman code for these three symbols.

**(2 Marks)**

ii) Form a new set of nine symbols by grouping together blocks of two symbols, AA, AB, AC, BA, BB, BC, CA, CB, and CC. Construct a Huffman code for these nine symbols, assuming that the occurrences of symbols in the original text are independent.

**(2 Marks)**

iii) Compare the average number of bits required to encode text using the Huffman code for the three symbols in part i) and the Huffman code for the nine blocks of two symbols constructed in part ii). Which is more efficient?

**(2 Marks)**

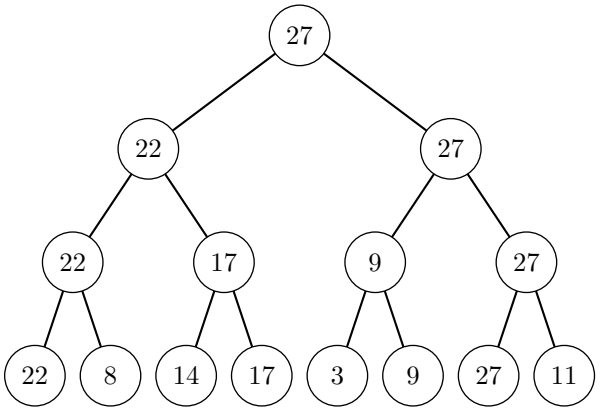
**Exercise 10.6**

The *tournament sort* is a sorting algorithm that works by building an ordered binary tree. We represent the elements to be sorted by vertices that will become the leaves. We build up the tree one level at a time as we would construct the tree representing the winners of matches in a tournament.

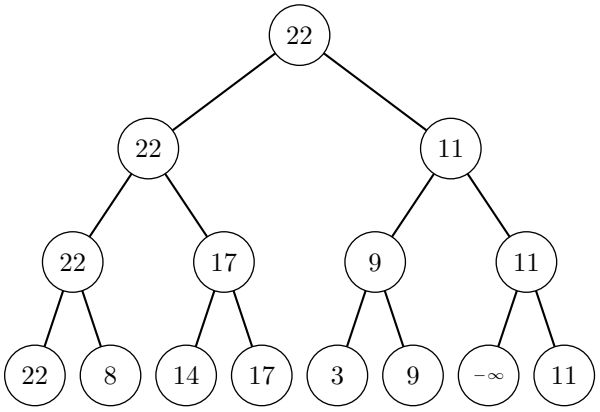
Working left to right, we compare pairs of consecutive elements, adding a parent vertex labeled with the larger of the two elements under comparison. We make similar comparisons between labels of vertices at each level until we reach the root of the tree that is labeled with the largest element.

The tree constructed by the tournament sort of 22, 8, 14, 17, 3, 9, 27, 11 is illustrated in part (a) of the figure at right. Once the largest element has been determined, the leaf with this label is relabeled by $-\infty$, which is defined to be less than every element. The labels of all vertices on the path from this vertex up to the root of the tree are recalculated, as shown in part (b) of the figure. This produces the second largest element. This process continues until the entire list has been sorted.

(a) 27 is the largest element.




(b) 22 is the second largest element.




i) Use the tournament sort to sort the list 17,4, 1, 5, 13, 10, 14, 6. Draw the tree at each step.

**(2 Marks)**

ii) Assuming that $n$, the number of elements to be sorted, equals $2^k$ for some positive integer $k$, determine the number of comparisons used by the tournament sort to find the largest element of the list using the tournament sort.

**(3 Marks)**

iii) How many comparisons does the tournament sort use to find the second largest, the third largest, and so on, up to the $(n-1)$st largest (or second smallest) element?

**(3 Marks)**

iv) Show that the tournament sort requires $\Theta(n \log n)$ comparisons to sort a list of n elements. [Hint: By inserting the appropriate number of dummy elements defined to be smaller than all integers, such as $-\infty$, assume that $n = 2^k$ for some positive integer $k$.]

**(3 Marks)**

(The tournament sort is a variant of the more widely known *heap sort*.)