Reserved Cells

| | |
|---|---|
| Ex. 1 | |
| Ex. 2 | |
| Ex. 3 | |
| Ex. 4 | |
| Ex. 5 | |
| Ex. 6 | |
| Tot. | |

# Algorithms and Programming

## 21 June 2018

## Part I: Theory

Register Number _______________ Family Name ________________________ First Name ________________________

Course: ○ 10 credit course (01$OGDLP$) ○ 12 credit course (02$OGDLM$)

**No books or notes are allowed. Solve exercises directly within the reserved space. Additional sheets are accepted only when strictly necessary. Examination time:** 50 **minutes.**

1. (2.0 **points)**
   Given the following sequence of pairs, where the relation $i$-$j$ means that node $i$ is adjacent to node $j$:

   2-7 5-3 1-7 6-2 5-9 5-6 10-9 3-5 6-8 10-0

   apply an on-line connectivity algorithm with weighted quick-union, showing at each step the content of the array and the forest of trees at the final step. Node names are integers in the range from 0 to 10.

2. **(**1.0 **points)**

10 **credit course** ($01OGDLP$)
Sort in ascending order with merge sort the following array of integers:

21 19 2 14 43 3 79 23 29 17 51 10 15 16 8

Show relevant intermediate steps.

12 **credit course** ($02OGDLM$)
Sort in descending order with merge sort the following array of integers:

21 19 2 14 43 3 79 23 29 17 51 10 15 16 8

Show relevant intermediate steps.

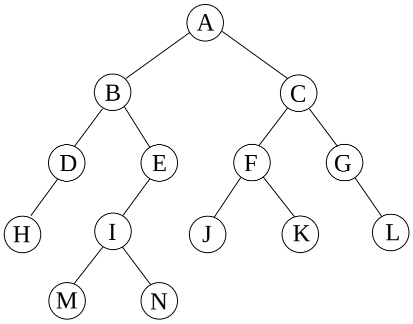
3. **(2.5 point)**
   Given the following sequence of integers stored in an array:

   11 29 2 11 31 3 39 17 29 17 41 10 15 12 8

   turn it into a heap, assuming to use an array as underlying data structure. Draw each step of the heap-building process, as well as the final result. Assume that, at the end, the largest value is stored at the heap's root. Execute the first three steps of the heap-sort algorithm on the heap built at the previous step.

   Assume that the sequence is already stored in the array and that it represents an intermediate configuration on which the heap property doesn't necessarily hold.

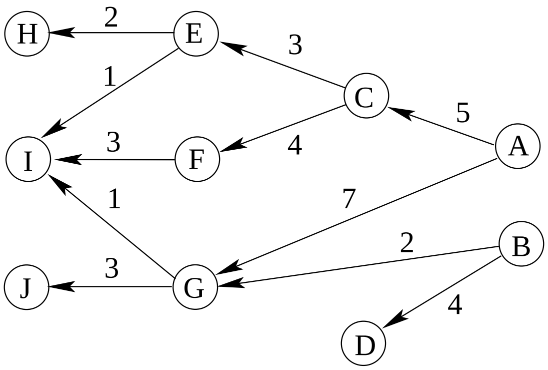
4. **(1.5 points)**
Visit the following binary tree in pre-order, in-order and post-order.



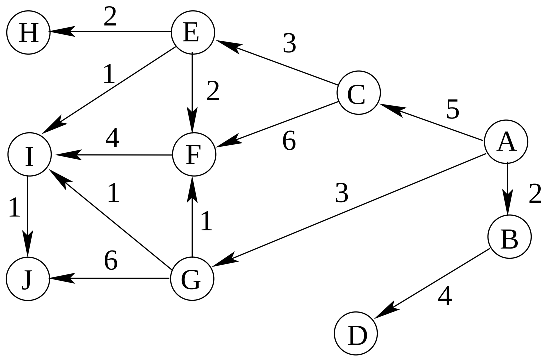
5. **(2.5 points)**
10 **credit course** ($01OGDLP$)
Consider the following weighted and directed graph. Visit it in breadth-first starting at node $A$. Label nodes with discovery times.

Redraw it, and visit it in depth-first starting at node $A$. Label nodes with discovery and end-processing times in the format $time_1/time_2$. Redraw it labeling each edge as $T$ (tree), $B$ (back), $F$ (forward), $C$ (cross). If necessary, consider nodes in alphabetical order.

12 **credit course** ($02OGDLM$)
Consider the following weighted DAG. Starting from $A$, find all longest paths connecting node $A$ with all the other nodes resorting to the algorithm for the longest paths on DAGs. If necessary, consider nodes in alphabetical order.



6. **(2.5 points)**
On the following directed and weighted graph, find all shortest paths connecting node A with all the other nodes resorting to Dijkstra's algorithm. If necessary, consider nodes in alphabetical order.



# Algorithms and Programming

# 21 June 2018

## Part II: Program (12 point version)

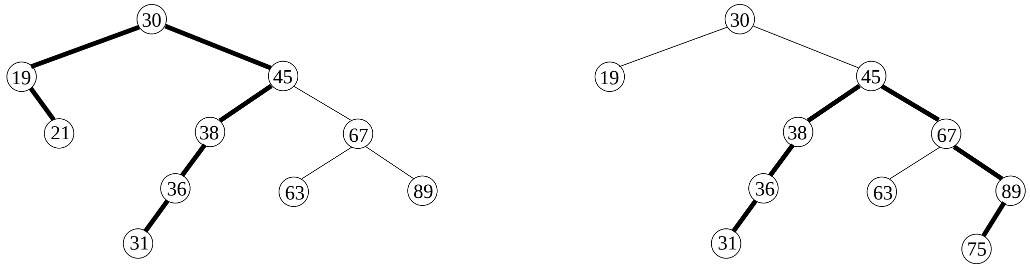
**At most one C manual is allowed. Examination time: 100 minutes. Final program due by midday of Tuesday the 26th; use the course portal page ("Elaborati" section) to upload it.**

**1 (2.0 points)**
Write function

```
int string_count (char *s, int n);
```

which receives a string `s`, and an integer `n`, and it returns the number of sub-string of `s` with length equal to `n` and at least two vowels (among *a*, *e*, *i*, *o*, and *u*).
For example, if `s` is "`ForExample`", and `n=4`, the desired sub-strings are "`ForE`", "`orEx`", "`rExa`", and "`Exam`".

**2 (4.0 points)**
The *diameter* of a binary tree is defined as the length of the longest path between any two nodes. The following pictures represent two trees with their longest paths, corresponding to a diameter equal to 6.



Write function

```
int tree_diameter (node_t *r);
```

which receives a pointer to the root of a binary tree `r` and it returns the diameter of the tree.
**Suggestion**: Visit both subtrees from each tree node, and compute the distance from that node to all reachable leaves.

**3 (6.0 points)**
Given a string $s$ of length $n$, a sub-sequence of characters of length $k$ of such a string is a set of $k$ characters $\{c_0 c_1 c_2 \dots c_k\}$ extracted from $s$, where $k \leq n$, the characters are not necessary contiguous, and they have increasing indices.
For example, given the string $AZCD$, if $k = 3$, then $AZC$, $AZD$, $ACD$, $ZCD$ are sub-sequences but $ADC$ is not.
Write function

```
void subsequences (char *s, int k);
```

to print all sub-sequences of string `s` of length `k` whose characters are strictly placed in alphabetical order.
For example, given the string $AZCD$, if $k = 3$, then $AZC$, $AZD$, $ACD$, $ZCD$ are sub-sequences but the function has to print only $ACD$.

# Algorithms and Programming

# 21 June 2018

## Part II: Program (18 point version)

**At most one C manual is allowed. Examination time: 100 minutes. Final program due by midday of Tuesday the 26th; use the course portal page ("Elaborati" section) to upload it.**

A game is played placing tiles on a rectangular board of $R \times C$ cells. Each tile contains a horizontal and a vertical pipe section. Pipe sections are characterized by a color (represented by a single character: 'B' for black, 'G' for green, etc.) and an integer value. Tiles can be placed into cells either with their given orientation or rotated by 90°. Adjacent pipe sections with the same color and orientation connect together to form longer pipes of the same color. Those pipes have a value equal to the sum of the values of all their individual pipe sections. The target of the game is to fill up the board by placing one tile at a time (assume that there are always enough tiles available to complete the board). Once the board is complete, the final board configuration is given a score that is equal to the sum of the values of all uninterrupted pipes (pipes with sections all of the same color) spanning an entire row or column.

A first file (`tiles.txt`) stores all tiles. The first row indicates the total number of available tiles. For each tile, a row of the file reports the color and value of its horizontal and vertical pipe section, in this order. Consider tiles as identified by integer indexes $i$, corresponding to their position (starting from 0) in the input file.

A second file (`board.txt`) stores an initial board configuration with the following format:

- The first row indicates the board size, i.e., the values of $R$ and $C$.
- The subsequent $R$ rows each contain $C$ elements with the format $i/r$. Each element corresponds to a cell of the board and describes the presence and the orientation of a tile in that cell, where:
  - $i$ indicates either that the tile with index $i$ is placed in that cell (if $i \geq 0$) or that no tile is placed in that cell (if $i = -1$).
  - $r$ indicates whether the tile is placed in that cell with its original orientation ($r = 0$) or is rotated by 90° ($r = 1$).

Write a program able to:

- Receive three file names on the command line. The first two are input files, i.e., `tiles.txt` and `board.txt`. The last one, is the output file.
- Verify whether the second file describes a final board configuration or not.
  - In the first case, the program has to compute and print (on standard output) the score obtained by the described board configuration.
  - In the second case, the program must complete the board using the remaining available tiles among those read from the first file. The program must find the final configuration with the maximum score among those that is possible to obtain from the initial board configuration using the available tiles. The final configuration must be stored in the output file with the same format of the input file.

**Example** The following figure reports a possible content for the two input files (`tiles.txt` and `board.txt`), the initial board configuration, the final one, and the output file. The final configuration has been obtained starting from the initial one and using the remaining tiles read from file `tiles.txt` to complete it. The output file stores the final configuration. The points obtained are $(8 + 0 + 0 + 4 + 11 + 12)$, i.e., 35 overall.

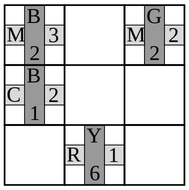
**File: tiles.txt**

```
9
M 3 B 2
M 2 G 1
M 2 G 2
B 1 C 2
M 3 Y 3
G 1 Y 2
R 1 Y 6
G 1 B 1
G 9 B 3
```

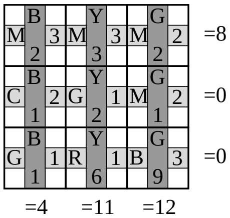
**File: board.txt**

```
3 3
0/0 -1/-1 2/0
3/1 -1/-1 -1/-1
-1/-1 6/0 -1/-1
```

**Initial configuration**



**Final configuration**



**Output file**

```
3 3
0/0 4/0 2/0
3/1 5/0 1/0
7/0 6/0 8/1
```