01OGD Algorithms and Programming

23/02/2015 – Part I: Theory (12 points)

1. (2 points)
Solve the following recurrence equation resorting to unfolding:

$$T(n) = 4T(n/3) + n \qquad n \geq 2$$
$$T(1) = 1$$

2. (1 point)
Sort in ascending order with counting-sort the following array of integers:

5 2 7 0 8 3 6 0 10 1 2 1 10 0 4

Show the contents of the data structures used at intermediate steps.

3. (2 points)
Given the following sequence of integers, assumed stored in an array:

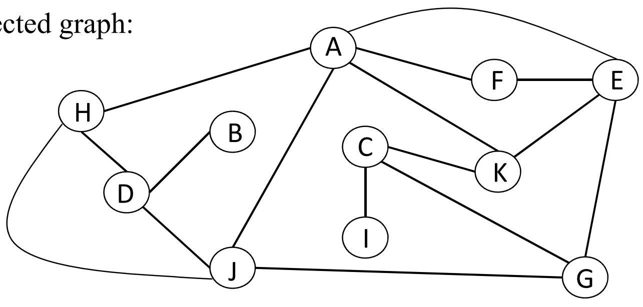
22 39 19 21 31 43 32 23 25 29 35 40

perform the first 2 steps of quicksort to obtain an ascending order. At each step indicate the pivot element you selected. NB: steps must be improperly considered in breadth on the recursion tree, rather than in depth. Return as a result the 2 partitions of the original array and the two partitions of the two partitions found at the previous step.

4. (2 points)
Execute in sequence the following operations on an initially empty BST (+ means insertion in leaf, − means deletion):

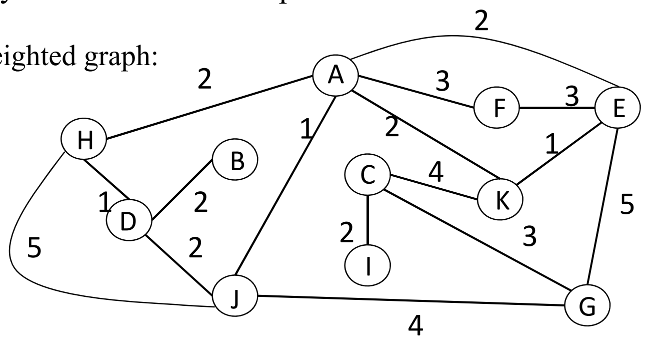
+3 +10 +4 +6 +1 +16 +21 +7 +11 -10 +2 +8 +12 -16 +5 -21

5. (1 point)
Find the articulation points of the following undirected graph:



Start at vertex **A.** Whenever necessary consider nodes in alphabetical order.

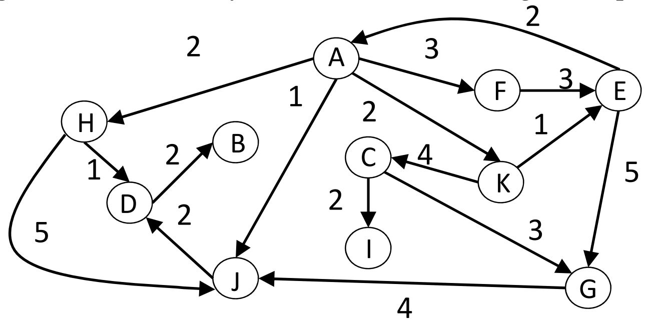
6. (2 points)
Given the following undirected and weighted graph:



find a minimum spanning tree applying Kruskal's algorithm. Show intermediate steps, draw the tree and compute the minimum weight.

8. (2 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 and edges in alphabetical order.

.



Algorithms and Programming

Examination Test

Programming Part
23 February 2015

No books or notes are allowed. Examination time: 100 minutes.

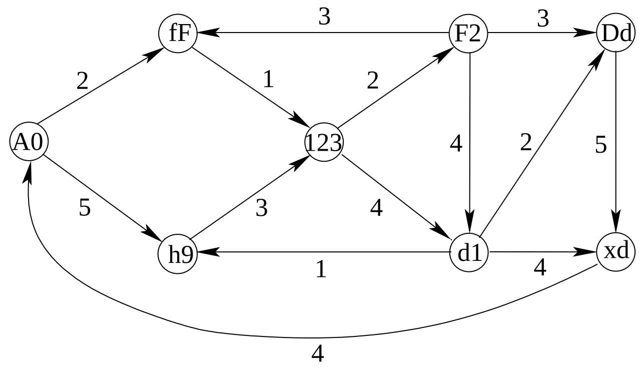
The programming part includes two *possibilities*:

- **Exercise number 1 (traditional) of a maximum value equal to 18 points.**
- **Exercises 2, 3 and 4 (easier) with a total maximum value of 12 points.**

Intermediate solutions are forbidden.

1 (18.0 points)
A weighted directed graph is stored in a file with the format represented on the left-hand side of the following example. The number of vertices is unknown so its their order. Each vertex is represented by an alphanumeric string of length equals to 20 characters at most. The graphical representation of the graph is reported on the right-hand side of the figure.

```
fF 1 123
A0 2 fF
A0 5 h8
h9 3 123
123 2 F2
123 4 d1
F2 3 Dd
F2 4 d1
d1 2 Dd
d1 4 xd
d1 1 h9
Dd 5 xd
xd 4 A0
F2 3 fF
```



Write a C program that:

- Receives the name of two files on the command line.
- Reads the graph from the first file and stores it in a proper data structure.
- Reads from the keyboard the identifiers of two vertices, s and d, and two integer values, k and p.
- Stores, on the o second file, the path
  - Starting in s.
  - Ending in d.
  - With at most k vertices traversed more than once.
  - With vertices traversed more than once globally traversed at most p times.
  - For which **the sum of the weights is maximum**.

For example for the previous graph, with $s = A0$, $d = fF$, $k = 1$, and $p = 1$ the required path, and the output, is

```
A0  h9  123  F2  d1  Dd  xd  A0  fF
```

with a total weight equal 27.

At the same time, with $s = A0$, $d = fF$, $k = 6$, and $p = 7$ the required path, and the output, is 1

```
A0  h9  123  F2  d1  Dd  xd  A0  h9  123  d1  Dd  xd  A0  fF
```

with a total weight equal 50.

continue

2 (2.0 points)
Write function

```
void invertSequence (int *v1, int n, int *v2);
```

where `v1` and `v2` are arrays of size `n`, and such that the function receives `v1`, and it stores into `v2` the same numbers stored in `v1` but such that all ascending sub-sequences appearing in `v1` are transformed into descending sub-sequences in `v2`.
For example, if `v1` is the following:

```
1 2 3 4 5 0 12 13 14 2
```

the function has to store into `v2` the following numbers:

```
5 4 3 2 1 14 13 12 0 2
```

3 (4.0 points)
A binary tree is defined using the following C structure:

```
struct node {
  int key;
  struct node *left;
  struct node *right;
};
```

Write function

```
void printPath (struct node *root, int h, ...);
```

that, given a tree of height equal to `h`, prints out all keys along all of its root-to-leaf paths. Notice that, each path has to be printed out entirely from root to leaf, i.e., the "path so far" needs to be communicated between recursive calls.
If it is necessary, it is possible to add arguments to the `printPath` function.

4 (6.0 points)
A password is a string of 5 characters, where:

- The first 3 characters are capital alphabetic letters ($A, \ldots, Z$).
- The last 2 characters are digits ($0, \ldots, 9$).
- The same letter or digit may appear in the password at most n times, where n is given by the user.

For example, with $n = 2$, passwords $AAB11$, $ZDZ09$, and $ABC34$ are acceptable, whereas $XXX12$ is not.
Write a recursive function able to generate all passwords satisfying the previous constraints and to write them on standard output.