Reserved Cells

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

Algorithms and Programming

31 January 2019

Part I: Theory

Register Number ______________ Family Name ______________________ First Name ______________________

Course: ◯ 01*OGDLP* 10 credit ◯ 02*OGDLM* 12 credit

No books or notes are allowed. Solve exercises directly within the reserved space. Additional sheets are accepted only when strictly necessary. Available time: 60 minutes.

1. (2.0 **points)**
   Write a function implementing the algorithm of insertion sort and state its asymptotic complexity.

   After that, given the following sequence of integers stored in an array:

   6 19 2 1 12 5 7 10 13 3 0 9 15 17 8

   sort it in ascending order with shell sort. Use the Knuth's sequence $h = 3 \cdot h + 1$ (1, 4, 13, etc.). Show all relevant intermediate steps.

2. **(2.0 points)**
Write a function implementing a visit of a n-ary tree (i.e., a tree in which each node has at most n children). State a possible implementation for its nodes, considering strings as keys.

After that, consider a binary tree with 13 nodes. Its visits return the following sequences:

pre-order:	A	B	D	G	E	H	K	I	L	C	F	J	M
in-order:	D	G	B	K	H	E	I	L	A	C	J	M	F
post-order:	G	D	K	H	L	I	E	B	M	J	F	C	A

Draw the original binary tree.

3. **(2.0 point)**
Given the sequence of keys $INFINITYWAR$, where each character is identified by its index in the English alphabet ($A = 1, \ldots, Z = 26$), draw the final configuration of an initially empty hash table of size 23 where insertion of the previous sequence character by character occurs.

Assume open addressing with double hashing ($h_1(k) = k\%23$, $h_2(k) = 1 + k\%97$). Show all relevant intermediate steps.

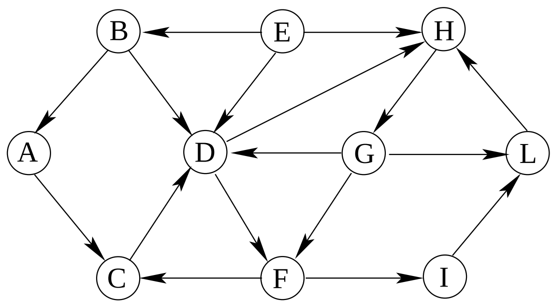
Which is the load factor of the hash-table after the previous sequence has been inserted?

4. **(2.0 points)**
Suppose to have an initially empty priority queue implemented with a minimum heap. Given the following sequence of integers and $\star$ character:

$$3 \quad 9 \quad 11 \quad 7 \quad 15 \quad 13 \quad 2 \quad * \quad * \quad 19 \quad 8 \quad 4 \quad *$$

where each integer corresponds to an insertion into the priority queue and character $\star$ corresponds to an extraction, show the priority queue after each operation and return the sequence of values extracted. At the end, change the priority of 15 into 1 and draw the corresponding priority queue.

5. **(2.0 points)**
10 credit course ($01OGDLP$)
Given the following unweighted directed graph:

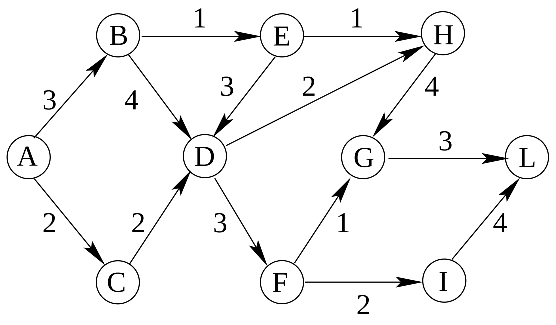


- Visit it in breadth-first starting from node A.
- Visit it in depth-first starting at node A. Label nodes with discovery and end-processing times in the format $time_1/time_2$.
- Redraw it labeling each edge as T (tree), B (back), F (forward), C (cross), starting at node A.

Whenever necessary consider nodes in alphabetical order.

12 **credit course** ($02OGDLM$)

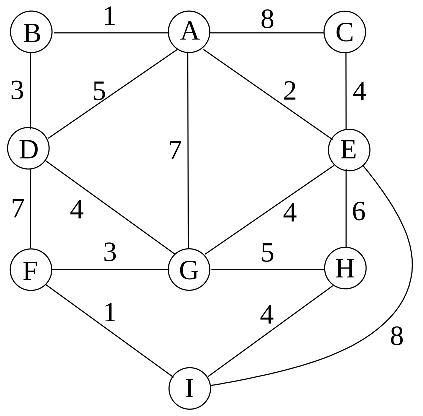
Given the following weighted DAG:



Find all shortest and all longest paths from vertex A, using the simplified algorithm for directed acyclic graphs.

If necessary, consider nodes in alphabetical order. Show all relevant intermediate steps.

6. (2.0 **points**)
10 **credit course** (01*OGDLP*)
Given the following undirected and weighted graph find a minimum spanning tree using **Prim's** algorithm starting from vertex *A*.

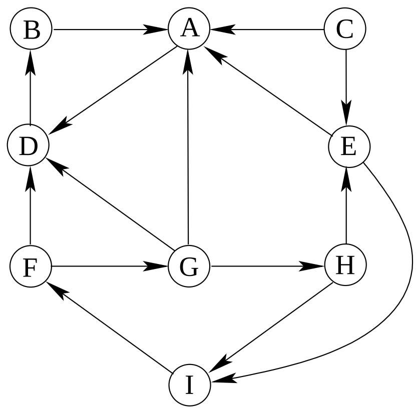


Draw the tree and return the minimum weight as a result. Show all relevant intermediate steps and all cuts generated.

Define and explain what a “light edge” and a “safe edge” are.

12 **credit course** ($02OGDLM$)

Given the following directed graph:



Find its strongly connected components using Kosaraju's algorithm. Start from node A. If necessary, consider nodes in alphabetical order. Show all relevant intermediate steps.

Algorithms and Programming
31 January 2019
Part II: Program (12 point version)

At most one C manual is allowed. Available time: 120 minutes. Final program due by 8.00 p.m. of Tuesday the 5th of February; use the course portal page ("Elaborati" section) to upload it.

1 (2.0 **points)**
Given an array `v` storing `n` integer values, write the function

```
void range_sum (int *v, int n, int c1, int c2);
```

which prints (on standard output) all sub-arrays of `v` including contiguous elements whose sum is strictly included in the range `[c1, c2]`.

For example if `v = {0, 5, 10, 15, 20, 25, 30, 35, 40, 50}`, `c1 = 20` and `c2 = 30` the program must print the following sub-arrays `{0, 5, 10, 15}`, `{5, 10, 15}`, `{10, 15}`, `{20}`, `{25}`, `{30}`.

2 (4.0 **points)**
Given an n-ary tree, write the function

```
void tree_max_level (node_t *root);
```

which prints the level of the tree which stores the maximum number of nodes. Report the C node structure to store the tree nodes.
Notice that the function may call other functions. Moreover, the average number of children is usually much smaller than the degree of the tree, and the implementation must optimize memory usage.

3 (6.0 **points)**
A basic calculator supports only 3 operations on integer decimal numbers: Multiplication, addition, and subtraction. Moreover operators have the same precedence and they are left associative. For example, the expression $22 - 21 * 79$ can be evaluated as $(22 - 21) * 79 = 79$.
Write the function

```
void calculator (int *v, int n, int result);
```

which receives an array `v` of size `n` of integer values, and the expression result, and it prints all possible expressions delivering that result.
For example, if `v = { 55, 3, 45, 33, 25 }`, `n = 5`, and `result = 404`, the required expression is `55 + 3 - 45 * 33 - 25`.

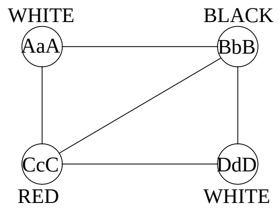
Algorithms and Programming
31 January 2019
Part II: Program (18 point version)

At most one C manual is allowed. Available time: 120 minutes. Final program due by 8.00 p.m. of Tuesday the 5th of February; use the course portal page ("Elaborati" section) to upload it.

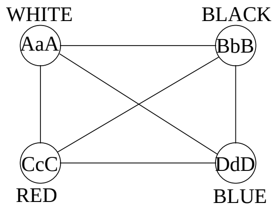
In graph theory, graph coloring is a special case of graph labeling. It is an assignment of labels traditionally called "colors" to elements of a graph subject to certain constraints. In its simplest form, it is a way of coloring the vertices of a graph such that no two adjacent vertices share the same color. This is called a **vertex coloring**.

An undirected unweighted graph is stored in a text file. The file reports the list of graph edges, one for each row. Identifiers of vertices are strings of unknown length, but limited to 100 characters. The following picture represents two input files, the corresponding graphs, and a final possible coloring.

```
AaA BbB
AaA CcC
BbB CcC
BbB DdD
CcC DdD
```



```
AaA BbB
AaA CcC
AaA DdD
BbB CcC
BbB DdD
CcC DdD
```



Write a C program able to:

- Receive the name of the file storing the graph on the command line, and to store the graph in a proper data structure.
- Solve the **vertex coloring** problem, i.e., find the minimum number of colors such that no two adjacent vertices share the same color. Print the correspondence vertex-color for all vertices, on standard output.

Notice that it is possible to use integer values to represent colors. Moreover, the candidate must present a proper implementation for all data structures used, the procedure to read the graph, and the core recursive function.