Reserved Cells

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

Algorithms and Programming

18 September 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**)
   Given the following sequence of integers stored in an array:

   3 1 2 4 5 9 0 0 1 3 4 8 7 3 2

   sort it in ascending order using counting sort.

   Show the content of arrays A, B and C and all relevant intermediate steps on the array C.

   Show how counting sort can be implemented in C language, and specify its asymptotic complexity. Motivate the conclusions intuitively.

2. (2.0 **points**)
Given the following program:

```
1  #include <stdio.h>
2
3  void f (int l, int r) {
4    int m;
5
6    if (l>=r) {
7      return;
8    }
9
10   m = (l+r) / 2;
11   fprintf (stdout, "call %d-%d\n", l, m);
12   f (l, m);
13   fprintf (stdout, "call %d-%d\n", m+1, r);
14   f (m+1, r);
15
16   return;
17 }
18
19 int main () {
20   f (0, 8);
21 }
```

draw the recursive tree generated by function `f`. For each recursive call, report the values of all parameters and the exact output generated by the procedure.

3. (2.0 **points**)

10 **credit course** (01$OGDLP$)

Consider a binary tree with 13-nodes. Its visits return the following sequences:

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

Draw the original binary tree.

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

Given an initially empty BST perform the following sequence of insertions on the BST **root**:

3 6 19 1 13 0 8

Report all relevant intermediate steps.

4. (2.0 **points**)
Given the sequence of integers:

31 123 19 101 13 9 42 7 20 88 54 33

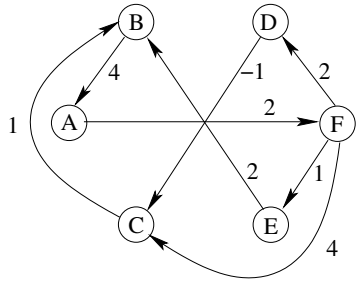
draw the final configuration of an initially empty hash table of size 23 where insertion of the previous sequence occurs. Assume open addressing with quadratic probing with $c_1 = 1$ and $c_2 = 1$. Show all relevant intermediate steps.

5. (2.0 **points**)
Suppose to have an initially empty priority queue implemented with a maximum heap. Given the following sequence of integers and $\star$ characters:

$$11 \quad 3 \quad 17 \quad 5 \quad 9 \quad 13 \quad 21 \quad 1 \quad 27 \quad * \quad * \quad 2 \quad *$$

where each integer corresponds to an insertion into the priority queue and each character $\star$ corresponds to an extraction, show the priority queue after each operation and return the sequence of values extracted.

6. **(2.0 points)**
Given the following directed and weighted graph, find all shortest paths connecting node A with all the other nodes resorting to Bellman-Ford's algorithm.



If necessary, consider nodes and edges in alphabetical order.

Would Dijkstra's algorithm find the same result? Justify your answer.

Algorithms and Programming
18 September 2019
Part II: Program (12 point version)

At most one C manual is allowed. Available time: 120 minutes. Final program due by 2.00 p.m. of Saturday the 21th of September; use the course portal page ("Elaborati" section) to upload it.

1 (2.0 **points**)
Write the function

```
void longest (char *in, int n, char ***out);
```

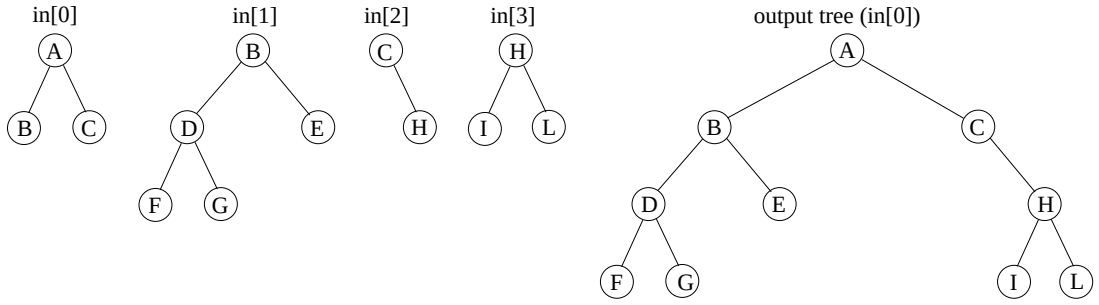
which receives the string `in`, an integer value `n`, and it returns the pointer to a dynamically allocated array of strings `out`. The value of `n` indicates the number of sub-strings within `in` including only uppercase consecutive letters. The array of strings `out` should contain all such sub-strings, and it must be allocated by the function.

For example with the string `in = "THIS is a STring incLUDing small and capITAL lETTERS"`, which includes `n = 5` sub-strings with only capital letters, the function has to create and return the pointer `out` referencing the following array of strings {`"THIS"`, `"ST"`, `"LUD"`, `"ITAL"`, `"ETTERS"`}.

2 (4.0 **points**)
Write the function:

```
void treeMerge (struct node **in, int n);
```

that receives an array `in` of size `n` of binary trees, and it merges those trees, i.e., merge all trees from `in[1]` to `in[n-1]` into the tree referenced by `in[0]`. Merging a tree `in[i]` into the tree `in[j]`, with $j < i$, means substituting the node of the tree j with the same key of the root of `in[i]` with the entire tree `in[i]`.
Each node of a tree contains a string key, and the two children pointers. Define an appropriate data structure for the node. Assume that the root of each tree, but the first one, appears only once as a leaf of one of the previous trees.
For example, if `n=4`, and the input array of trees in `in`, the output tree will be the one on the right-hand side:



3 (6.0 **points**)
In the school computer room a set of servers is responsible for processing all computing tasks, each one with its own time cost. In order to balance the time load for each server, it is necessary to assign all tasks to make the difference between the most loaded server and the least loaded server as small as possible. In other words it is necessary to minimize the expression $t_{max} - t_{min}$, where t_{max} (t_{min}) is the largest (smallest) sum of the running times of all tasks assigned to a server. Write the function:

```
void balance (int nTask, int *times, int nServer);
```

which

- Receives the number of tasks `nTask`, the array storing the time costs of all tasks `times`, and the number of servers `nServer` in the computer room.
- Assigns each task to a server such that the aforementioned cost function is minimized.

The function has to print the computer identifier (a number from 0 to $nServer - 1$) assigned to each task.
For example if there are `nTask=7` tasks, with time costs equal to `times` $= \{1, 2, 3, 4, 5, 6, 7\}$, and $nServer = 3$ servers, we can assign $\{2, 7\}$ to computer 0, $\{3, 6\}$ to computer 1, and $\{1, 4, 5\}$ to computer 3, with a cost difference of $10 - 9 = 1$.

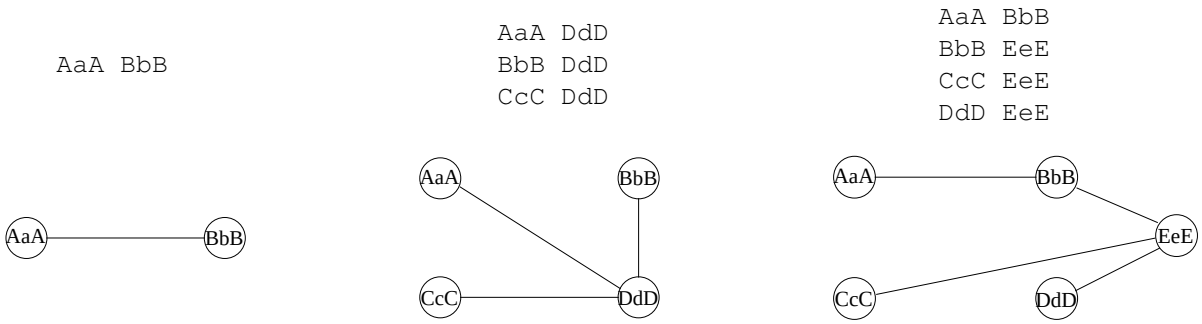
Algorithms and Programming
18 September 2019
Part II: Program (18 point version)

At most one C manual is allowed. Available time: 120 minutes. Final program due by 2.00 p.m. of Saturday the 21th of September; use the course portal page ("Elaborati" section) to upload it.

Solve the following "Vertex cover problem".
A vertex cover of an undirected graph is a subset of its vertices that covers every edge, that is for every edge (u, v) of the graph, either u or v is in vertex cover.
Given an undirected graph, the vertex cover problem is to find a vertex cover of minimum size.

The undirected unweighted graph is stored in a text file. The file reports the list of graph edges, one for each row. The identifiers of the vertices are strings of unknown length, but limited to 100 characters.

The following picture represents 3 different input files, and the corresponding graphs.



In the first example (left-hand side) the minimum vertex cover is either {`AaA`} or {`BbB`}. In the second one, it is the set {`DdD`}. In the third example (right-hand side) the minimum vertex cover is either {`BbB`, `EeE`} or {`AaA`, `EeE`}.

The C program must:

- Receive, on the command line, the name of the file storing the graph.
- Define a proper data structure to store the graph.
- Store the graph in a proper data structure.
- Solve the aforementioned problem, and print one possible solution on standard output.