Reserved Cells

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

Algorithms and Programming

13 February 2018

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. Added sheets are accepted only when strictly necessary. Examination time: 50 **minutes.**

1. **(**2.0 **points)**
   Sort in ascending order with shell sort the following array of integers:

   10 8 5 4 8 5 3 6 1 9 12 9 15 13 0

   Use the Knuth's sequence $h = 3 \cdot h + 1$ (1, 4, 13, etc.). Report all main steps to obtain the final order.

2. **(1.0 points)**
Sort in ascending order with counting sort the following array of integers:

$$1 \quad 4 \quad 5 \quad 3 \quad 2 \quad 7 \quad 8 \quad 2 \quad 1 \quad 9 \quad 0 \quad 2 \quad 6 \quad 9 \quad 3$$

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

3. (2.0 **points**)

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

A correct BST contain integer keys in the range 1-1000. The user searches for key 871. Among these sequences, which are the ones that cannot occur? Why?

14	250	490	530	750	971	889	817	613	871		
81	300	351	380	931	420	901	571	899	701	898	871
998	100	900	310	390	400	600	700	800	899	871	
500	991	687	792	988	810	757	880	885	871		

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

Using a greedy algorithm find an optimal Huffman code for the following symbols with the specified frequencies:

A:12 B:32 C:10 D:8 E:9 F:17 G:13 H:4 I:11 J:8

4. **(2.0 points)**
Given the sequence of keys

$$101 \quad 124 \quad 157 \quad 172 \quad 98 \quad 133 \quad 44 \quad 205 \quad 16 \quad 78 \quad 189$$

draw the final configuration of an initially empty hash table of size 23 where insertion of the previous sequence occurs. Assume open addressing with double hashing and hash functions $h_1(k) = k\%23$, $h_2(k) = 1 + k\%97$. Show relevant intermediate steps.

5. (1.0 **point**)

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

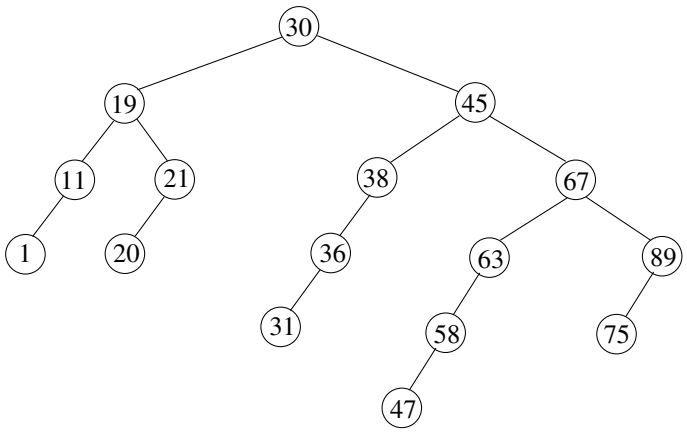
Given the following BST perform the following operations:

$$+3 \quad +27 \quad +98 \quad +43 \quad +39 \quad +60 \quad +32 \quad -30 \quad -45 \quad -67$$

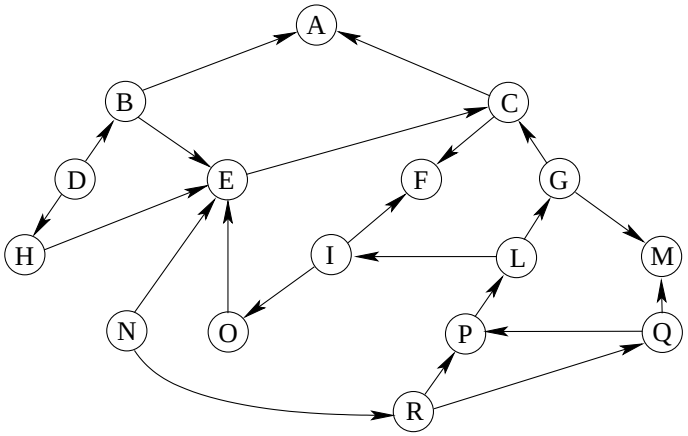
where each + symbol indicates and insertion in the leaves, and each − symbol represents an extraction.

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

Partition the following BST around key 58:



6. **($1.0 + 2.0 + 1.0$ points)**
Suppose to have the following directed graph:



- Represent it as an adjacency matrix.
- 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).

Whenever necessary consider nodes in alphabetical order.

Algorithms and Programming
13 February 2018
Part II: Program (12 point version)

At most one C manual is allowed. Examination time: 100 minutes. Final program due by midday of Friday the 16th; use the course portal page ("Elaborati" section) to up-load it.

1 (2.0 points)
Write function

```
void submat_write (int **m, int r, int c);
```

which receives a matrix `m` storing `r` rows and `c` columns of integer values, and it prints-out (on standard output) all square sub-matrices included in `m` with size larger than 1.
Let us suppose that

1	2	3	4
6	7	8	9
10	11	12	13

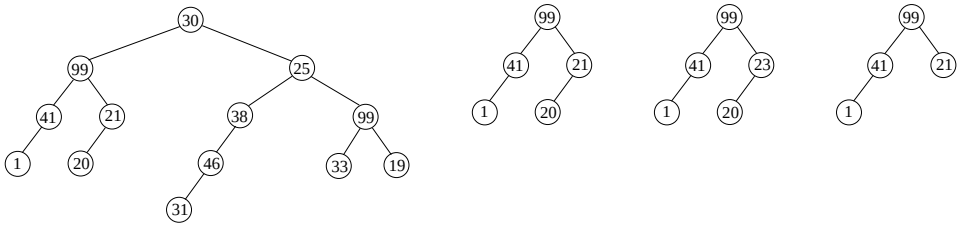
is the input matrix `m`, with `r=3` and `c=4`. The function has to print out the following sub-matrices:

1	2	2	3	3	4	6	7	7	8	8	9	1	2	3	2	3	4
6	7	7	8	8	9	10	11	11	12	12	13	6	7	8	7	8	9
												10	11	12	11	12	13

2 (4.0 points)
Write function

```
int subtree_check (node_t *r1, node_t *r2);
```

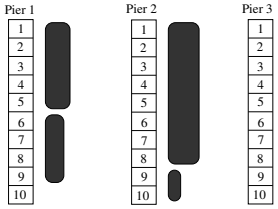
which receives two pointers to binary trees `r1` and `r2`, and it checks whether tree `r1` is a subtree of tree `r2`. It returns 1 when `r1` is a subtree of `r2`. Both trees store only integer values. Define the node structure.
Notice that `r1` is a subtree of tree `r2` if a subtree of `r2` has the same shape (is isomorphic) and stores the same values of `r1`. In the following example the second tree is a subtree of the leftmost one, whereas the other two are not.



3 (6.0 points)
A port has `N` piers of length `L`. In such a port `m` ships have to dock. Each ship i is characterized by its length $\texttt{l}_i$.
Write function

```
void ship_to_port (int N, int L, int m, int *l);
```

which finds a ship distribution on the piers such that the minimum number of piers is used to dock all ships. Such a placement has to be printed-out by the function. As already stated `N` is the number of piers, `L` it is their length (all piers have the same length), `m` is the number of ships, and the array `l` stores their length (in `m` elements).
For example, if $\texttt{N} = 3$, $\texttt{L} = 10$, $\texttt{m} = 4$, and $\texttt{l} = \{5, 8, 4, 2\}$ a possible placement for those ships is on pier 1 for ship 1 and 3 and pier 2 for ships 2 and 4. The next picture show a diagrammatical representation of such an example.



Algorithms and Programming
13 February 2018
Part II: Program (18 point version)

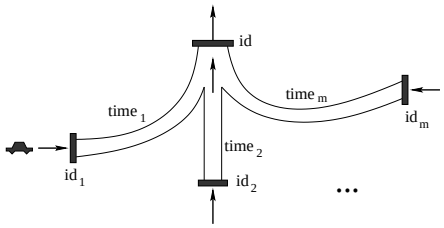
At most one C manual is allowed. Examination time: 100 minutes. Final program due by midday of Friday the 16th; use the course portal page ("Elaborati" section) to up-load it.

Write a C application able to manage the tutoring system of the highway network of a large country to check for errors and to issue fines (tickets) as follows.

The highway network is spread with a high number of tutoring stations. Each tutoring station stores the car plate numbers of all cars passing below the station itself and the time of the passage. As each station knows which other stations precede it on the highway, for each car passage it checks whether the car has already passed in a previous station and at which time. If the car has not been registered by a previous tutoring station the application issues an error message. If the car has driven too fast, exceeding the speed limits, the system issues a fine.

A first file stores all tutoring stations, using an adjacency list format where for each list incoming edges are stored instead of outgoing ones:

$$\begin{array}{l} n \\ id \quad m \quad time_1 \quad id_1 \quad time_2 \quad id_2 \quad \ldots \quad time_m \quad id_m \\ \ldots \end{array}$$



where n is the number of tutoring stations, id is the station identifier (a string of at most 20 characters), m is the number of stations that immediately precede that station in different highway directions, and all other id_i on the same row are their identifiers. Each $time_i$ indicates the minimum traveling time to drive from the tutoring station id_i to id as a string with format *hour.min.sec* (e.g., 01.11.00).

A second file stores all car passages below all tutoring stations. Each row of the file has the following format:
$id \quad plateNumber \quad time$
where id is the tutoring station identifier, $plateNumber$ is the car plate number (a string of 7 characters), and $time$ is the date and time in the format *year.month.day.hour.min.sec* (e.g., 2018.02.13.21.30.00). The file is ordered by increasing values of the time field.

After the initial set-up, the application has to:

- Upload the graph stored into the first file in a proper data structure.
- For each row of the second file the application must:
  - Register the new passage in the involved tutoring station.
  - Check whether the car has been registered by a previous tutoring station.
    - If there is a previous recorded passage, the system has to check whether the car drove below or above the speed limit. If the car drove too fast (i.e., the time difference between the new passage and the previous is higher than the minimum traveling time reported for those two stations) the system has to issue a fine. The previous passage must be erased by the system.
    - If there is not a previous passage, and the station has at least an incoming station, the system has to issue an error message on standard output.

    The fine and the error messages have to include the identifier of the station, the plate number, and the passage time.
- Once all records of the second file have been read, the application has to print on standard output the identifiers of all tutoring stations ordered by increasing number of registered passages, and the number of those passages.

Notice that the system has to be very time efficient as it has to be able to work in real-time (i.e., inserting new passages, checking for fines, and erasing old passages) with a high number of tutoring stations, and a huge number of car traveling on the highway network. For that reason, a proper design of the data structure and the application (modularity, etc.) is of up-most importance. Library functions can be used whenever necessary.