

01OGD Algorithms and Programming

06/02/2014 – Part I: Theory (12 points)

1. (2 points)

Solve the following recurrence equation resorting to unfolding:

$$T(n) = 8T(n/2) + n^3 \quad n \geq 2$$

$$T(1) = 1$$

2. (2 points)

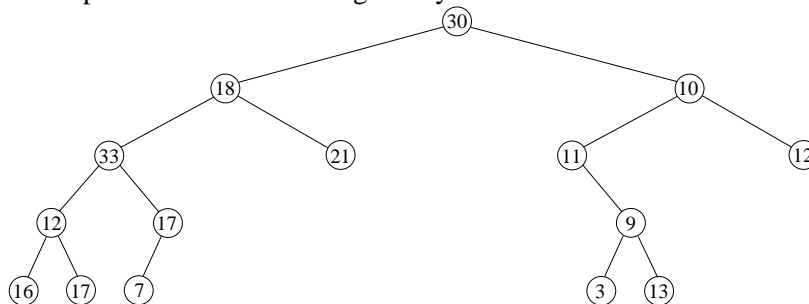
Given the following sequence of integers, assumed stored in an array:

12 29 9 11 21 33 22 13 15 19 25 30

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.

3. (3 x 0.5 points)

Visit in pre-order, in-order and post-order the following binary tree:

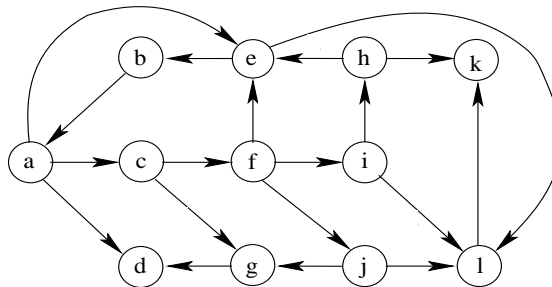


4. (2 points)

Given the sequence of keys SIDEFFECTS, where each character is identified by its progressive order in the alphabet (A=1, ..., Z=26), draw a hash table of size 23, initially empty, after the insertion of the above characters in sequence. Use open addressing with double hashing resorting to hash functions $h_1(k) = k \bmod 23$, $h_2(k) = 1 + (k \bmod 21)$.

5. (2.5 points)

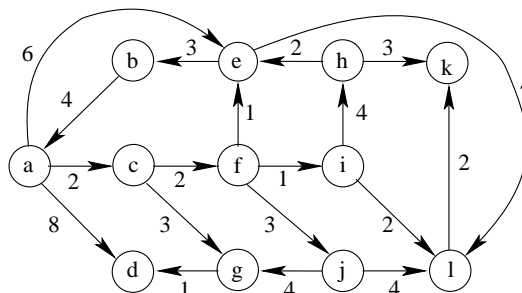
Given the following directed graph:



Find its Strongly Connected Components with Kosaraju's algorithm. Start from node **a** and, if necessary, consider nodes in alphabetical order.

6. (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

06 February 2014

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)

The *Verbal Arithmetic* is a mathematics game published for the first time by Henry Dudeney in the 1924 July issue of Strand Magazine. Let us suppose to have two integer numbers whose digits are encrypted by capital letters, and to know, again in encrypted form, the result of an addition between those two encrypted numbers.

The following is an example:

$$\begin{array}{rcccccc} & S & E & N & D & + \\ & M & O & R & E & = \\ \hline M & O & N & E & Y & \end{array}$$

The game consists in identifying for each letter the decimal digit satisfying the operation. For example, using the correspondence

$$O = 0 \quad M = 1 \quad Y = 2 \quad E = 5 \quad N = 6 \quad D = 7 \quad R = 8 \quad S = 9$$

we obtain

$$\begin{array}{rcccccc} & 9 & 5 & 6 & 7 & + \\ & 1 & 0 & 8 & 5 & = \\ \hline 1 & 0 & 6 & 5 & 2 & \end{array}$$

Let us make the following hypothesis:

- The only operation to be performed is addition.
- Each letter is associated to a single and unique decimal value.
- The two input strings include only lower or upper case letters.
- The most significant digit of each number cannot correspond to zero.
- The two strings do not necessary have the same length, and the resulting number has a number of digits which is dependent on the addition performed on the two numbers.
- Strings have a maximum length of 8 characters.
- Strings do not include more than 10 different characters.

Write a C program that, once received the three strings on the command line, is able to:

- Verifying a solution given by the user.
- Automatically generating a solution.

In the first case, the program has to read the solution from standard input (the input format is free), and to verify its correctness. In the second case, the program has to compute and print-out a possible letter-to-digit correspondence.

Suggestions

The problem of identifying the letter-to-digit correspondence can be solved adopting recursion with backtracking. The acceptance condition can be verified once reached a leaf of the recursion tree, without any preliminary check. During the acceptance check, it is possible to verify the operation working on single digits and performing the addition “by hand” or converting the strings into numbers and use standard computer arithmetic.

continue

2 (2.0 points)

Given 3 arrays of integer values of given length and ordered in ascending order, implement a merge operation.

The prototype of the function is the following:

```
int *merge3 (int *a, int *b, int *c, int na, int nb, int nc);
```

The parameters **a**, **b** and **c** are the 3 ordered input arrays of sizes **na**, **nb** and **nc**, respectively. The resulting array has to be returned by the function, and it has a size equal to **na+nb+nc**. This array has to be the result of a merge operation on the three original arrays.

The output array has to be dynamically allocated by the function. The **merge** operation has to be **unique**, i.e. it is not allowed to merge the three arrays by merging two of them and then the resulting array with the third one.

3 (4.0 points)

Write a C function able to **insert some anagraphical data, in the correct position in an ordered list**. Data include the surname and name of a person. Both fields are string **of** characters of **maximum length equal to 20**.

Use the **surname** as **primary ordering key**, and the **name** as **secondary ordering key**.

The prototype of the function is the following:


```
int orderInsert (list_t *list, char *surname, char *name);
```

It receives the list pointer (of type **list_t**), and the two strings storing surname and name. It returns an integer value which is **false** when the person was already present in the list, and true when the insertion operation succeeded.

Define the list_t type, considering that, for each **person**, surname and name have to **be copied in new dynamically allocated and separated fields**.


4 (6.0 points)

An **array of float of length equal to n** is given. Each float represents a bank **account cash flow**, i.e., a deposit, when positive, or a withdraw, when negative.

Given a **specific order** of those cash flow operations, for each of them, we define **current balance** the value resulting from the addition of the previous balance with the current value. The initial balance is equal to zero. **All cash flows values are unique.** 

Given an order of the cash flow operations there is a **maximum and a minimum** current balance, even if the final balance is (obviously) always the same.

For example, with cash flow equal to **(+10 -5 +7 -8)** the maximum and minimum current balances are **+12** and **+4** (with a difference equal to **+8**), whereas with **(-5 +10 -8 +7)** they are **+5** and **-5** (with difference **+10**).

Write a C program that, using a recursive algorithm, is able to **find an order** such that the difference between the maximum and the **minimum current balance is minimum.** 

The function has to return this order in the original array of bank cash flow movements.