Algorithms and Data Structures

05.09.2023

Exercise 1
Report the C implementation of quicksort, including the wrapper function, the recursive procedure, and the Hoare partition scheme.
Suppose the algorithm sorts an array of N integers in ascending order. Describe what a pivot is and which are the strategies to select it. Which is the worst-case complexity of the algorithm? Motivate your answer with an example.

Exercise 2
The following expression is given in in-fix notation. Using its representation as a binary tree, convert it into pre-fix (Polish) and post-fix (Reverse Polish) notation.

```
( A * ( B - C ) + D ) - ( ( E - F ) * ( G / ( H + I ) ) )
```

Report the expression in pre-fix notation and in post-fix notation. Notice that this is an open question that will be manually evaluated.

```
### Expression evaluation
Original expression: ( A * ( B - C ) + D ) - ( ( E - F ) * ( G / ( H + I ) ) )
Infix notation     : A * B - C + D - E - F * G / H + I
Prefix notation    : - + * A - B C D * - E F / G + H I
Postix notation    : A B C - * D + E F - G H I + / * -
```

Exercise 3
A BST contains integer values included in the range 1-1000. Suppose we are looking for the value 0499 is such a BST. Consider the following sequences of values generated during a search.

```
788 196 517 208 488 508 490 502 491 498 500 499
79 586 393 406 577 445 563 475 491 536 493 503 501 500 490 496 499
328 424 655 574 535 446 492 525 522 507 494 496 504 501 497 500 498 499
276 314 736 629 520 386 404 478 486 517 509 504 488 489 503 493 494 495 502 497 501 498 499
```

Indicate which ones of the following sequences (numbered 1, 2, 3, and 4) are correct. Notice that wrong responses imply a mark penalty.

```
### BST verify sequence
Searching: 499
Sequence  0: 788 196 517 208 488 508 490 502 491 498 500 499
            788 196 517 208 488 508 490 502 491 498 500 499 Sequence OK.
Sequence  1: 79 586 393 406 577 445 563 475 491 536 493 503 501 500 490 496 499
             79 586 393 406 577 445 563 475 491 Error: 490<491
Sequence  2: 328 424 655 574 535 446 492 525 522 507 494 496 504 501 497 500 498 499
            328 424 655 574 535 446 492 525 522 507 494 496 504 501 497 500 498 499
             ↪  Sequence OK.
Sequence  3: 276 314 736 629 520 386 404 478 486 517 509 504 488 489 503 493 494 495
 ↪  502 497 501 498 499
            276 314 736 629 520 386 404 478 486 517 509 504 488 489 503 493 494 495
             ↪  502 497 501 498 499 Sequence OK.
```

Exercise 4

Insert the following sequence of keys into an initially empty hash table. The hash table has a size equal to M=19. Insertions occur character by character using open addressing with quadratic probing (with c1=1 and c2=1). Each character is identified by its index in the English alphabet (i.e., A=1, ..., Z=26). Equal letters are identified by a different subscript (i.e., A and A become A1 and A2).

```
B A Z A A R
```

Indicate in which elements are placed the last three letters of the sequence, i.e., A, A, and R, in this order. Please, report your response as a sequence of integer values separated by one single space. No other symbols must be included in the response. This is an example of the response format: 3 4 11

```
### hash-table
Character keys
Hash table size = 19
Number of keys  =  6
Quadratic probing: c1=1.000000, c2=1.000000.
Key:  B A Z A A R
Key=B k=002 Final=02
Key=A k=001 Final=01
Key=Z k=026 Final=07
Key=A k=001 Coll.= 1 Final=03
Key=A k=001 Coll.= 1 Coll.= 3 Coll.= 7 Final=13
Key=R k=018 Final=18
00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18
-  A  B  A  -  -  -  Z  -  -  -  -  -  A  -  -  -  -  R
```

Exercise 5

Given the following sequence of integers stored into an array, turn it in a heap, assuming to use an array as an underlying data structure. Assume that, in the end, the largest value is stored at the heap's root. Then, execute the first two steps of heapsort on the heap built at the previous step.

```
11 23  2  4  6  3 13  9  5
```

Report the final content of the entire array at the end of the above process. Please, show the entire content of the array as a sequence of integer values separated by a single space. No other symbols must be included in the response. This is an example of the response: 0 3 2 6 8 etc.

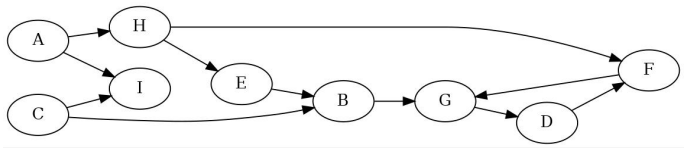
```
### Heap sort
                 0  1  2  3  4  5  6  7  8
-----------------------------------------
Initial array: 11 23  2  4  6  3 13  9  5
heap-build
---------------
heapify         [3<->7]
                11 23  2  9  6  3 13  4  5
heapify         [2<->6]
                11 23 13  9  6  3  2  4  5
heapify         No swap
                11 23 13  9  6  3  2  4  5
heapify         [0<->1]
                23 11 13  9  6  3  2  4  5
heap-sort
---------------
heapify         [0<->2]
                13 11  5  9  6  3  2  4 23
heapify         [0<->1][1<->3]
                11  9  5  4  6  3  2 13 23
heapify         [0<->1][1<->4]
                 9  6  5  4  2  3 11 13 23
heapify         [0<->1][1<->3]
                 6  4  5  3  2  9 11 13 23
heapify         [0<->2]
                 5  4  2  3  6  9 11 13 23
heapify         [0<->1]
                 4  3  2  5  6  9 11 13 23
heapify         [0<->1]
                 3  2  4  5  6  9 11 13 23
heapify         no-swaps
                 2  3  4  5  6  9 11 13 23
```

Exercise 6

Visit the following graph depth-first, starting at node A. Label nodes with discovery and end-processing times. Start with the discovery time set to 1 on A. When necessary, consider nodes and edges in alphabetic order.

```
  A B C D E F G H I
A 0 0 0 0 0 0 0 1 1
B 0 0 0 0 0 0 1 0 0
C 0 1 0 0 0 0 0 0 1
D 0 0 0 0 0 1 0 0 0
E 0 1 0 0 0 0 0 0 0
F 0 0 0 0 0 0 1 0 0
G 0 0 0 1 0 0 0 0 0
H 0 0 0 0 1 1 0 0 0
I 0 0 0 0 0 0 0 0 0
```

Display the end-processing time of all vertices. Please, indicate the end-processing time of all vertices sorted in alphabetic order (i.e, display the end-processing time for A B C D etc.). Report a sequence of integer values separated by one single space. No other symbols must be included in the response. This is an example of the response: 15 13 2 16 8 etc.

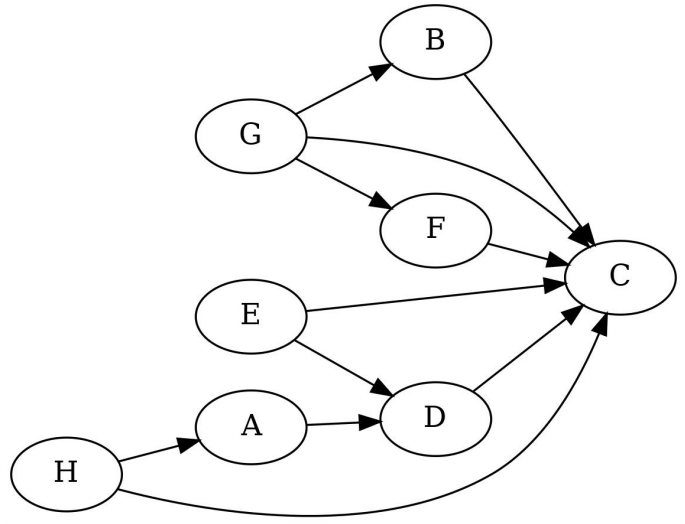



```
Directed matrix ... stated as such.
UN-weighted matrix ...stated as such.
Vertex list: [ 0] A [ 1] B [ 2] C [ 3] D [ 4] E [ 5] F [ 6] G [ 7] H [ 8] I
Adjacency Matrix:
 0 0 0 0 0 0 0 1 1
 0 0 0 0 0 0 1 0 0
 0 1 0 0 0 0 0 0 1
 0 0 0 0 0 1 0 0 0
 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 1 0 0
 0 0 0 1 0 0 0 0 0
 0 0 0 0 1 1 0 0 0
 0 0 0 0 0 0 0 0 0

### Graph visit
BFS:
Node=[ 0] A Discovery_time= 0 Predecessor=[-1]
Node=[ 1] B Discovery_time= 3 Predecessor=[ 4]
Node=[ 2] C Discovery_time= - Predecessor=[-1]
Node=[ 3] D Discovery_time= 4 Predecessor=[ 6]
Node=[ 4] E Discovery_time= 2 Predecessor=[ 7]
Node=[ 5] F Discovery_time= 2 Predecessor=[ 7]
Node=[ 6] G Discovery_time= 3 Predecessor=[ 5]
Node=[ 7] H Discovery_time= 1 Predecessor=[ 0]
Node=[ 8] I Discovery_time= 1 Predecessor=[ 0]

DFS:
List of edges:
[ 0] A -> [ 7] H : (T) Tree
[ 7] H -> [ 4] E : (T) Tree
[ 4] E -> [ 1] B : (T) Tree
[ 1] B -> [ 6] G : (T) Tree
[ 6] G -> [ 3] D : (T) Tree
[ 3] D -> [ 5] F : (T) Tree
[ 5] F -> [ 6] G : (B) Back
[ 7] H -> [ 5] F : (F) Forward
[ 0] A -> [ 8] I : (T) Tree
[ 2] C -> [ 1] B : (C) Cross
[ 2] C -> [ 8] I : (C) Cross
List of vertices:
[ 0] A: dist_time= 1 endp_time=16 pred=[-1] -
[ 1] B: dist_time= 4 endp_time=11 pred=[ 4] E
[ 2] C: dist_time=17 endp_time=18 pred=[-1] -
[ 3] D: dist_time= 6 endp_time= 9 pred=[ 6] G
[ 4] E: dist_time= 3 endp_time=12 pred=[ 7] H
[ 5] F: dist_time= 7 endp_time= 8 pred=[ 3] D
[ 6] G: dist_time= 5 endp_time=10 pred=[ 1] B
[ 7] H: dist_time= 2 endp_time=13 pred=[ 0] A
[ 8] I: dist_time=14 endp_time=15 pred=[ 0] A
Reachable nodes from vertex [ 0] A:
[ 0] A - [ 1] B - [ 2] C - [ 3] D - [ 4] E - [ 5] F - [ 6] G - [ 7] H - [ 8] I -
UN-reachable nodes from vertex [ 0] A:
none
```

Exercise 7

Given the following unweighted DAG, find the topological order of its vertices. Start the DFS visit from vertex A. If necessary, consider nodes in alphabetical order.

	A	B	C	D	E	F	G	H
A	0	0	0	1	0	0	0	0
B	0	0	1	0	0	0	0	0
C	0	0	0	0	0	0	0	0
D	0	0	1	0	0	0	0	0
E	0	0	1	1	0	0	0	0
F	0	0	1	0	0	0	0	0
G	0	1	1	0	0	1	0	0
H	1	0	1	0	0	0	0	0



Report the direct topological order of the vertices. Report vertices on the same line separated by a single space. No other symbols must be included in the response. This is an example of the response format: F A C H B etc.

```
Directed matrix ... stated as such.
UN-weighted matrix ...stated as such.
Vertex list: [ 0] A [ 1] B [ 2] C [ 3] D [ 4] E [ 5] F [ 6] G [ 7] H
Adjacency Matrix:
 0 0 0 1 0 0 0 0
 0 0 1 0 0 0 0 0
 0 0 0 0 0 0 0 0
 0 0 1 0 0 0 0 0
 0 0 1 1 0 0 0 0
 0 0 1 0 0 0 0 0
 0 1 1 0 0 1 0 0
 1 0 1 0 0 0 0 0

### Graph DAG
DFS (discovery and end-processing times):
[ 0] A: dist_time= 1 endp_time= 6
[ 1] B: dist_time= 7 endp_time= 8
[ 2] C: dist_time= 3 endp_time= 4
[ 3] D: dist_time= 2 endp_time= 5
[ 4] E: dist_time= 9 endp_time=10
[ 5] F: dist_time=11 endp_time=12
[ 6] G: dist_time=13 endp_time=14
[ 7] H: dist_time=15 endp_time=16

Topological sort (inverse):
[ 2] [ 3] [ 0] [ 1] [ 4] [ 5] [ 6] [ 7]
  C    D    A    B    E    F    G    H

Topological sort (direct):
[ 7] [ 6] [ 5] [ 4] [ 1] [ 0] [ 3] [ 2]
  H    G    F    E    B    A    D    C

Shortest Paths:
  H    G    F    E    B    A    D    C
infy infy infy infy infy   0    1    2

Longest Paths:
  H    G    F    E    B    A    D    C
infy infy infy infy infy   0    1    2
```

Exercise 8

Analyze the following program and indicate the exact output it generates. Please, report the exact program output with no extra symbols.

```c
#include <stdio.h>
#include <string.h>
void f (char []);
int main(void) {
  char s[] = "ABCABCDEF";
```

```
  f (s);
  fprintf (stdout, "%s", s);
  return (1);
}
void f (char s[]) {
  int i, j, k;
  char c;
  i = 0;
  while (i<strlen(s)-2) {
    j = i;
    while (j<strlen(s)-1 && s[j]<s[j+1]) {
      j++;
    }
    for (k=0; k<=(j-i)/2; k++) {
      c = s[i+k];
      s[i+k] = s[j-k];
      s[j-k] = c;
    }
    i = j+1;
  }
  return;
}
```

Exercise 9

The following function performs a Depth-First Search of a graph stored as an adjacency list. Indicates which ones of the following statements are correct. Note that incorrect answers imply a penalty in the final score.

```
int graph_dfs_r(graph_t *g, vertex_t *n, int currTime) {
  edge_t *e;
  vertex_t *t;
  n->color = GREY;
  n->disc_time = ++currTime;
  e = n->head;
  while (e != NULL) {
    t = e->dst;
    switch (t->color) {
      case WHITE: printf("%d -> %d : T\n", n->id, t->id);
                  break;
      case GREY : printf("%d -> %d : B\n", n->id, t->id);
                  break;
      case BLACK:
        if (n->disc_time < t->disc_time) {
          printf("%d -> %d : ???\n",n->disc_time,t->disc_time); // LINE 1A
        } else {
          printf("%d -> %d : ???\n", n->id, t->id);             // LINE 1B
        }
    }
    if (t->color == BLACK) {  // LINE 2
      t->pred = n;
      currTime = graph_dfs_r (g, t, currTime);
    }
    e = e->next;
  }
  n->color = BLACK;  // LINE 3
  n->endp_time = ++currTime;
  return currTime;
}
void graph_dfs (graph_t *g, vertex_t *n) {
  int currTime=0;
  vertex_t *t1, *t2;
  printf("List of edges:\n");
  currTime = graph_dfs_r (g, n, currTime);
  for (t1=g->g; t1!=NULL; t1=t1->next) {
    if (t1->color == WHITE) {
      currTime = graph_dfs_r (g, t1, currTime);
    }
  }
  printf("List of vertices:\n");
  for (t1=g->g; t1!=NULL; t1=t1->next) {
    t2 = t1->pred;
    printf("%2d: %2d/%2d (%d)\n",
      t1->id, t1->disc_time, t1->endp_time,
      (t2!=NULL) ? t1->pred->id : -1);
  }
}
```

1. The function has a complexity limited by $|V| + |E|$.

2. In LINE 1A the node is a Forward (F) edge and in LINE 1B a Cross (C) edge.

3. In LINE 2 the constant "BLACK" must be "WHITE".

4. The function has a complexity limited by $|V| \cdot |E|$.

5. In LINE 1B the node is a Forward (F) edge and in LINE 1A a Cross (C) edge.

6. In LINE 2 the constant "BLACK" must be "GREY".

7. In LINE 3 the constant "BLACK" must be "GREY".

Exercise 10
Analyze the following program and indicate the exact output it generates. Please, report the exact program output with no extra symbols.

```
#include <stdio.h>
void f1 (int n);
void f2 (int n);
void f3 (int n);
void f1 (int n) {
  if (n<=0) {
    return;
  }
  printf ("1");
  f2 (n-1);
  f3 (n-1);
  return;
}
void f2 (int n) {
  if (n<=0) {
    return;
  }
  printf ("2");
  f1 (n-1);
  f3 (n-1);
  return;
}
void f3 (int n) {
  if (n<=0) {
    return;
  }
  printf ("3");
  f1 (n-1);
  f2 (n-1);
  return;
}
int main () {
  f1(3);
  return 1;
}
```

Exercise 11
Write the function

```
int common_substring (char *str1, char *str2);
```

which receives two strings (str1 and str2), and it:

- Displays the longest common substring of str1 and str2. Notice that substrings are sequences of contiguous characters.
- Return the length of such a substring.

For example, if str1 = "123ABCD34EFG" and str2 = "XXXABCE124YABCD" the function must display "ABCD" and it returns 4.

Write the entire program using standard C libraries but implement all required personal libraries. Modularize the program adequately, and report a brief description of the data structure and the logic adopted in plain English. Unclear or awkward programs, complex or impossible to understand, will be penalized in terms of the final evaluation.

Exercise 12

A sparse matrix, i.e., a matrix with many values equal to zero, is represented as a list of lists. The main list includes elements of type list_t1 storing the row index, and two pointers: One to elements of the list_t1, and the other to elements of type list_t2. The secondary lists include elements of type list_t2 storing the column index, the element value, and a pointer to elements of type list_t2.

Write the function:

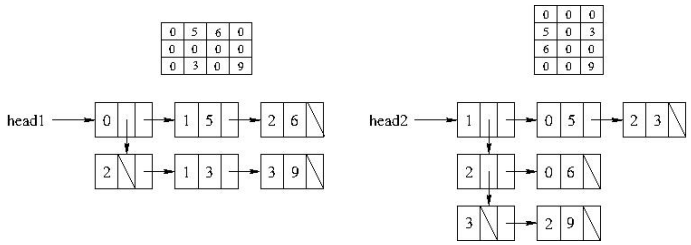
```
void transpose (list_t1 *head1, list_t1 **head2);
```

which receives the sparse matrix referenced by pointer head1 and creates the transpose sparse matrix referenced by pointer head2.

The following picture represents an example of an input matrix and the corresponding representation with a list of lists (pointed by head1, on the left-hand side) and an output matrix and its list of list representation (pointed by head2, on the right-hand side).

JPEG

Write the entire program using standard C libraries but implement all required personal libraries. Modularize the program adequately, and report a brief description of the data structure and the logic adopted in plain English. Unclear or awkward programs, complex or impossible to understand, will be penalized in terms of the final evaluation.



Exercise 13

Given an unweighted and undirected graph G, graph coloring is the procedure of assignment of colors to each vertex of G such that no adjacent vertices have the same color. Notice that colors can be represented with integer values, and a graph with n nodes can be colored with at most n different colors (all n colors are required only if the graph is wholly connected).

Given a graph G, write a function to color a graph using the minimum number of colors.

More.texifically, write the function

```
void color (int **graph, int n);
```

which receives the adjacency matrix of the graph (graph), and the number of vertices of G (n) and it displays the list of vertices with their color using the minimum number of colors.

For example, in the following figure

JPEG

the adjacency matrix on the left-hand side describes the unweighted and undirected graph represented on the left. Notice that the nodes labels, i.e., A, B, C, and D, are reported only to understand the correspondence between the rows and columns of the matrix and the vertices of the graph. The graph in the middle is colored with four colors, i.e., 1, 2, 3, and 4, whereas the graph on the right is colored with only two colors, i.e., 1 and 2. Two is also the minimum number of colors required to color that graph; thus, the function must display a solution similar to:

```
A 1
B 2
C 1
D 2
```

Write the required function program using standard C libraries but implement all required personal libraries. Modularize the program adequately, and report a brief description of the data structure and the logic adopted in plain English. Unclear or awkward programs, complex or impossible to understand, will be penalized in terms of the final evaluation.
Suggestion: Try to assign all colors (from 1 to n) to each vertex, then verify whether no adjacent nods have the same color, and finally retain the solution with the minimum number of colors.

	A	B	C	D
A	0	1	0	1
B	1	0	1	0
C	0	1	0	1
D	1	0	1	0