POLITECNICO DI TORINO

Algorithms and data structures

07 February 2023 - On-Site Exam

	TOMMASO CERRUTI 282754
Iniziato	martedì, 7 febbraio 2023, 11:07
Terminato	martedì, 7 febbraio 2023, 14:07
Tempo impiegato	3 ore

Domanda 1

Completo

Punteggio max.: 4,00

Define what a Minimum Spanning Tree is and specify the differences from a Single Source Shortest Path. Define what a cut, a light edge, and a safe edge are. Clarify why we define safe edges and how the algorithms of Prim and Kruskal use this concept. Are they greedy algorithms? Do they find an optimal solution? There is only one or more than one solutions to the MST problem?

Given a weighted graph G, a Minimum Spanning Tree (MST) is the set of edges E that minimizes the cost
(i.e. the sum of the edges weights wi) to span all the vertices V like a tree (i.e. no loops).
It is different from a Single Source Shortest Path (SSP) because SSP does not necessiraly include all the vertices
belonging to the graph (just those of interest to find the shortest path between two given nodes).
A cut is when, after having selected some vertices, we "cut" the edges dividing what is already part of the tree and the rest of the graph with a continuos line.
After we perform a cut we may define the edges that have been cut as a light edge, if among all the other edges its weight is minimum.
Prim and Krusal algorithms are both algorithms used to find the MST.
They differ in their approach: Prim starts with the minimum weighted edge and looks for edges only connected to the present tree (so performing a cut in such a way
these two sets are separated), Kruskal instead looks at the minimum weighted edge of the all graph (so performing a cut in such a way that we group what we already inserted).
Of course, they both do this while avoiding to form any loop!.
A greedy algorithm is an algorithm adopted in optimization problem that, instead of making difficult computations,
or expensive computations (brute-force approach), it assumes that the global optimal solution will coincide with the local one.
As a matter of fact, both algorithms look at the best local solution assuming it will lead to best one, so they indeed are greedy!
To note is that the MST problem may have more than one solution, but as long they both satify the MST properties
(i.e. they both should have the minimum and same sum-weight and not forming any loops).

Domanda 2

Completo

Non valutata

If you want to withdraw from the exam, please select true. Otherwise, i.e., you want to take the exam, select false.

- (a) False (No, I do not want to withdraw)
- (b) True (Yes, I want to withdraw)

Domanda 3

Completo

Punteggio max.: 1,50

The following capital letters are given with their absolute frequency.

A:4 B:7 C:8 D:10 E:17 F:5

Find an optimal Huffman code for all symbols in the set using a greedy algorithm. Indicate which one of the following encoding is correct.

- (a)
  A(000) B(100) C(101) D(01) E(11) F(001)
- (b)
  A(11) B(10) C(110) D(111) E(1110) F(00)
- (c) A(010) B(10) C(110) D(1111) E(1110) F(00)
- (d)
  A(101) B(110) C(11) D(1111) E(1110) F(00)
- (e) A(01) B(100) C(110) D(1111) E(1110) F(00)
- (f) A(01) B(10) C(110) D(1111) E(1110) F(00001)

Domanda 4

Completo

Punteggio max.: 5,00

A file stores an undefined number of rows. Each row includes two fields: An integer value and a string of at most 100 characters. These fields are separated by one or more spaces.
Write a C function to organize the file content into a BST in which each node points to a list. The keys of the BST are integer values, and the lists store strings. More specifically, all records of the file with the same integer value must be stored in the list of the same BST element.
The function:

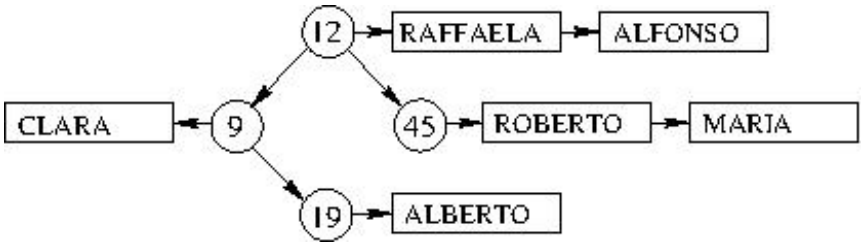
```
bst_t *insert (char *name);
```

receives the file name and returns the pointer to such a BST. The lists do not have to be sorted and may contain repeated elements (i.e, the strings of the record with the same integer value). Define the type **bst_t** for the BST nodes, and the type **list_t** for the list nodes. For the sake of simplicity, the string within the type **list_t** can be statically allocated.

For example, if we suppose the file includes the following rows

```
12 ALFONSO
45 MARIA
9 CLARA
12 RAFFAELA
45 ROBERTO
19 ALBERTO
```

the function must create the following data structure



and return the pointer to the root node 12.

Write the entire function using only C standard libraries and 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.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define N 100

//first we define the data structure

typedef struct bst_s bst_t;
typedef struct list_s list_t;
```

```
// bst_t defines the struct of the nodes of the BST tree containing:
//its key (int key), the pointers to the children (bst_t *left, *right) and the pointer to the list (list_t
*next)
struct bst_s{
   int key;
   bst_t *left, *right;
   list_t *next;
};
//list_t defines the struct of the nodes of the list containing:
//its key (char *name) that we need to dynamically allocate and the pointer to the next node in the
list (list_t *next)
// NOTE: the text explicitly says that we can allocate the strings statically, but allocating them
dynamically saves memory!
struct list_s{
   char *name;
   list_t *next;
};

//function declaration
bst_t *insert(char *);
list_t *new_node_list(char *, int);
bst_t *new_node_tree(int, list_t *);
void insert_r(bst_t *, list_t *, int);

//wrapper that opens the file, reads it and calls insert_r that recursively inserts the nodes properly
in the BST
bst_t *insert(char *name){
   bst_t *root,*node_bst;
   list_t *node_list;
   FILE *fp;
   char pname[N+1]; //buffer
   int key;
   if ((fp=fopen(name,'r'))==NULL){
      fprintf(stderr,"File opening error");
      exit(1);
   }
   root=NULL;
   while(fscanf(fp,"%s %d",pname,&key)==1){
      //note: we always have to create a node for the list but not always for the bst since it might be
already present
      node_list=new_node_list(pname,key);
      insert_r(root,node_list,key);
   }
   fclose(fp);
   return root;
}
```

```
list_t *new_node_list(char *name, int key){
  list_t *node;
  node=malloc(sizeof(list_t));
  if (node==NULL){
    fprintf(stderr,"Memory allocation error");
    exit(1);
  }
  node->name=strdup(name);
  if (node->name==NULL){
    fprintf(stderr,"Memory allocation error");
    exit(1);
  }
  node->next=NULL;
  node->key=key;
  return node;
}

bst_t *new_node_tree(int key, list_t *list){
  bst_t *node;
  node=malloc(sizeof(bst_t));
  if (node==NULL){
    fprintf(stderr,"Memory allocation error");
    exit(1);
  }
  node->key=key;
  node->left=NULL;
  node->right=NULL;
  node->next=list;
  return node;
}

void insert_r(bst_t *node_tree, list_t *node_list, int key){
  if (node_tree==NULL){//the node does not exist in the tree (or the tree is empty)
    node_tree=new_node_tree(key,node_list);
    return;
  }
  if (node_tree->key>key)//recur to the left
    insert_r(node_tree->left,node_list,key);
  else if (node_tree->key<key)//recur to the right
    insert_r(node_tree->right,node_list,key);
  else{//same node, head insert on the list
    list_t *tmp;
    tmp=node_tree->next;
    node_tree->next=node_list;
    node_list->next=tmp;
    return;
```

```
    }
    //never reached but for safety
    return;
}
```

Domanda 5

Completo

Punteggio max.: 2,00

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

```
#define R 3
#define C 5
#define D 3

void f (int [][C]);

int main(void) {
  int mat[R][C] = {
    {1,2,3,4,5},
    {1,2,3,4,5},
    {1,2,3,4,5},
  };
  f (mat);
  printf ("\n");
  return (1);
}

void f (int mat[][C]) {
  int i1, i2, j1, j2, s;

  for (i1=0; i1<=R-D; i1++) {
    for (j1=0; j1<=C-D; j1++) {
      s = 0;
      for (i2=0; i2<D; i2++) {
        for (j2=0; j2<D; j2++) {
          s = s + mat[i1+i2][j1+j2];
        }
      }
      printf ("%d ", s);
    }
  }

  return;
}
```

Risposta: 18 27 36

Domanda 6

Completo

Punteggio max.: 1,50

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).

```
R  I  C  C  I  A
```

Indicate in which elements are placed the last three letters of the sequence, i.e., the last C, I, and A, 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

Risposta: 5 11 1

Domanda 7

Completo

Punteggio max.: 9,00

Write the recursive function

```
void generate (char *name, int n);
```

which stores in the file **name** all decimal integers of **n** digits for which:

- the digits of weight 10^0, 10^2, 10^4, 10^6, etc., have an even value (e.g., 0, 2, 4, 6, 8)
- the digits of weight 10^1, 10^3, 10^5, 10^7, etc., have an odd value (e.g., 1, 3, 5, 7, 9)
- the sum of all even digits equals the sum of all odd digits.

For example, with n=4 digits, the following numbers satisfy the specified properties: 1430 (with sum 0+4=3+1=4), 3652 (with sum 2+6=5+3=8), 5676 (with sum 6+6=7+5=12).
The solution must prune the recursion tree as soon as possible, i.e., solutions that generate all integer numbers of n digits and display only the numbers satisfying the requested properties will be penalized in terms of the final mark.

Write the entire function using only C standard libraries and 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.

```
/*
IDEA:
- The order of the digit matters
- We can repeat each digit (up to n), so repetitions are allowed
--> Use arrangements with repetitions!
*/

#include <stdio.h>
#include <stdlib.h>
#define N 9

void generate(char *, int );
int generate_r(int , FILE *, int , int , int);
int evaluate(int *, int);

//wrapper function:
- opening the file (and closing it)
- declaring the data structure we need (and free them if dynamically allocated)
- calling the recursive function generate_r

void generate(char *name, int n){
  FILE *fp;
  if ((fp=fopen(name,'w'))==NULL){
    fprintf(stderr,"File opening error");
    exit(1);
  }
  int *sol,count;
  sol=malloc(n*sizeof(int));
  if (sol==NULL){
    fprintf(stderr,"Memory allocation error");
    exit(1);
  }
  count=generate_r(n,fp,sol,0,0);
  free(sol);
  fclose(fp);
  return;
}

// the text says to avoid brute-force approach, i.e. having the evaluate function at the termination condition
```

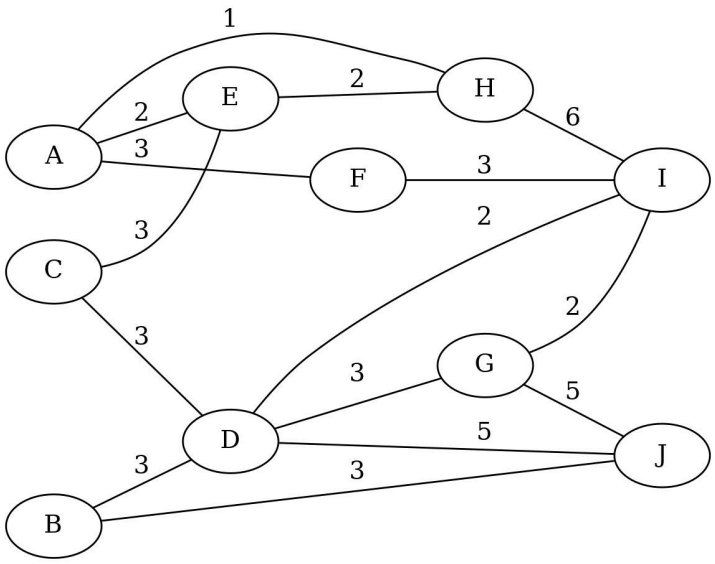
```
// indeed, for a large value n, this would cause overflow
// instead, at each step, we evaluate if we are doing the right choice (satisfying the first and second condition)
// to check this we want pos and i to be either both even or both odd
// only the third condition can be evaluated only at the end
//extra: we can also count the solution (for the particular wrapper given makes no sense since we loose its value, but just for the matter of the exercise)

int generate_r(int n, FILE *fp, int *sol, int pos, count){
  int i;
  if (pos>=n){//termination condition
    //check if it satisfies third condition
    if (evaluate(sol,n)){
      //if yes print it (since stored in the array we need to print it backwards, from the most significant digit)
      for (i=n-1;i>=0;i++){
        fprintf(fp,"%d",sol[i]);
      }
      fprintf(fp,"\n");
      return count+1;
    }
    return count;
  }
  for (i=0;i<=N;i++){
    if ((pos%2)==(i%2)){//both odd or both even
      sol[pos]=i;
      count = generate_r(n,fp,sol,pos+1,count);
    }
  }
  return count;
}

int evaluate(int *sol, int n){
  int i, sum_e=0, sum_o=0;
  for (i=0;i<n;i++){
    if ((i%2)==0)
      sum_e+=sol[i];
    else
      sum_o+=sol[i];
  }
  if (sum_e==sum_o)
    return 1;
  else
    return 0;
}
```

Domanda 8

Completo

Punteggio max.: 1,50

Given the following undirected and weighted graph find a minimum spanning tree using Kruskal algorithm.



Indicate the total weight of the final minimum spanning tree. Report one single integer value. No other symbols must be included in the response. This is an example of the response format: 13

Risposta: 22

Domanda 9

Completo

Punteggio max.: 3,00

A square matrix m of size n stores only capital alphabetic letters.
Write the function

```
int check (char **m, int n);
```

which receives the matrix m, its size n, and returns the index of the row that includes the same alphabetic letter more times. The function must return -1 if no rows contain the same letter more than once.
For example, if the matrix m is the following one (with size n=6)

```
X A E I O U
U X A E I O
O U X A E I
I O U X A E
E I O U X A
X Y Z X Y X
```

the last row includes three letters X; thus, the function must return its index, i.e., 5.

Write the entire function using only C standard libraries and 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.

```
/*
The idea is to use an array of size N=26 (size of english alphabet), where we count the occurances at at each row
While we count them, we check if the array is not set to zero for that index, if yes, just return the row index.
Of course, we need to reset the array at every row iteration to distinguish the repetitions between rows and those across the matrix.
*/

#include <stdio.h>
#include <stdlib.h>
#define N 26

int check(char **m, int n){
  int i,j,*vet,ind;
  for (i=0;i<n;i++){//at every row we allocate the vector with calloc so it is already intialized to zero
    vet=calloc(N,sizeof(int);
    if (vet==NULL){
      fprintf(stderr,"Memory allocation error");
      exit(1);
    }
    for(j=0;j<n;j++){
      ind=m[i][j]-'A'; //we find the index of interest
      if (vet[ind]==1){//we already found that char
        free(vet);//free the vector before returing the row index
        return j;}
      else
        vet[ind]++;
    }
    free(vet); //we need to free it before allocating it again (if not the last row, in that case we just end the program)
  }
```

```
    //no rows contain the same letter twice
    return -1;
}
```

Domanda 10

Completo

Punteggio max.: 1,50

Given the following array of integer values, perform the first step of quicksort to sort the array in ascending order; then, from the initial array generate the right and the left partitions.

10 9 11 4 1 4 2 3 7 5

Report 3 integer values: The pivot selected on the original array, the pivot you would select on the left partition generated from the original array, and the pivot you would select on the right partition generated (again) from the original array. No other symbols must be included in the response. This is an example of the response format: 13 1 10

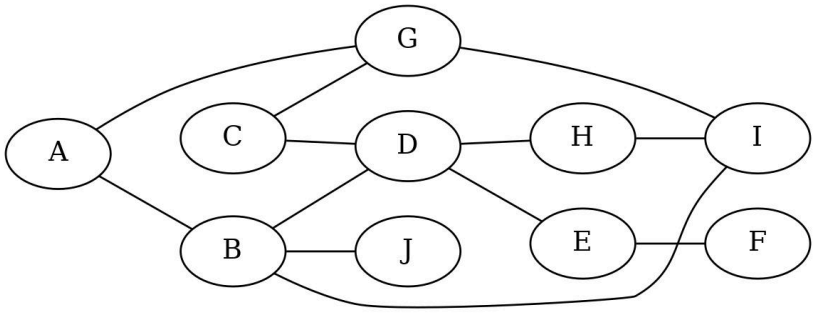
Risposta: 5 1 11

Domanda 11

Completo

Punteggio max.: 1,50

Given the following graph find all articulation points. If necessary, consider nodes and edges in alphabetical order.



Display all articulation points alphabetically. Please, report only the list of vertices separated by a single space. No other symbols must be included in the response. This is an example of the response format: A B C etc.

Risposta: B E

Domanda 12

Completo

Punteggio max.: 1,50

Consider a binary tree whose visits return the following sequences.

Pre-order:	A	B	D	F	H	G	E	C	I	L
In-order:	H	F	D	G	B	E	A	I	L	C
Post-order:	H	F	G	D	E	B	L	I	C	A

Report the sequence of keys stored on the leaves of the tree, moving on the tree from left to right. Please, report the list of keys 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: A X C Y etc.

Risposta: HGEL

Domanda 13

Completo

Punteggio max.: 2,00

The following function allocates a three-dimensional matrix (a two-dimensional matrix of strings) and reads the strings from standard input. The parameters r and c are the numbers of rows and columns of the matrix.
Indicates which ones of the following statements are correct.
Note that incorrect answers imply a penalty in the final score.

```
char ***read (int r, int c) {
  char *word, ***mx; // LINE 1
  int i, j;

  // LINE 2
  if (mx == NULL) {
    printf("Memory allocation error.\n");
    exit(EXIT_FAILURE);
  }
  for (i=0; i<r; i++) {
    mx[i] = (char **)malloc(c * sizeof(char *));
    if (mx[i] == NULL) {
      printf("Memory allocation error.\n");
      exit(EXIT_FAILURE);
    }
  }

  for (i=0; i<r; i++) {
    for (j=0; j<c; j++) {
      printf("String: ");
      fscanf("%s", word);
      LINE 3;
      if (mx[i][j] == NULL) {
        printf("Memory allocation error.\n");
        exit(EXIT_FAILURE);
      }
    }
  }

  return mx;
}
```

Scegli una o più alternative:

- [x] (a) LINE 2 can be: mx = (char ***)malloc(r * sizeof(char **));
- [x] (b) LINE 2 can be: mx = malloc(r * sizeof(char **));
- [ ] (c)
  In LINE 1 the array word must be either defined statically or allocated dynamically.
- [ ] (d)
  In LINE 1 the array word can be defined as: char word[r];
- [ ] (e) LINE 2 can be: mx = (char **)malloc(r * sizeof(char *));
- [ ] (f) LINE 2 can be: mx = malloc(r * sizeof(char *));
- [ ] (g) LINE 3 can be: strcpy (mx[i][j], word);

☑ (h)
LINE 3 can be: mx[i][j] = strdup (word);

Domanda 14

Completo

Punteggio max.: 2,00

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

```
void f (int);
void fp (int);
void fm (int);

void f (int n) {
  if (n%2==0) {
    fp (n);
  } else {
    fm (n);
  }
  return;
}

void fp (int n) {
  if (n<=0) {
    return;
  }
  printf ("+");
  f (n-1);
  return;
}

void fm (int n) {
  if (n<=0) {
    return;
  }
  printf ("-");
  f (n-1);
  return;
}

int main () {
  f(10);
  return 1;
}
```

Risposta: +-+-+-+-+-