Corso: Algoritmi e strutture dati
Quiz: ASD2025 - Appello Straordinario - Programmazione

PORTALE ESAMI

[redacted]

Informazione

Attenzione!

Indicare quale tipologia di esame si intenda svolgere rispondendo alla domanda a scelta multipla seguente (domanda 1).
Si noti che è possibile leggere tutte o parte delle domande prima di effettuare la scelta e rispondere alla domanda 1.
Le domande dalla 2 alla 4 sono dedicate all'esame semplificato (traccia da 12pt).

In particolare:

- la domanda 2 fa parte dell'esercizio da 3 punti.
- la domanda 3 fa parte dell'esercizio da 4 punti.
- la domanda 4 fa parte dell'esercizio da 5 punti.

Le domande dalla 5 in poi sono dedicate all'esame completo (traccia da 18pt).

Un apposito separatore fa da intervallo tra le domande del compito da 12pt e le domande del compito da 18pt.

Informazione

PER ENTRAMBE LE PROVE DI PROGRAMMAZIONE (18 o 12 punti)

- se non indicato diversamente, è consentito utilizzare chiamate a funzioni standard, quali ordinamento per vettori, funzioni su FIFO, LIFO, liste, BST, tabelle di hash, grafi e altre strutture dati, considerate come librerie esterne
- gli header file riferiti dal codice dovranno essere inclusi nella versione del programma allegata alla relazione
- i modelli delle funzioni ricorsive **non** sono considerati funzioni standard
- consegna delle relazioni, per entrambe le tipologie di prova di programmazione: entro LUNEDI' 24/02/2025, alle ore 23:59, mediante caricamento su Portale
- assicurarsi di caricare l'elaborato nella **Sezione Elaborati relativa all'a.a. *2024/25***.
- le istruzioni per il caricamento sono pubblicate sul Portale nella sezione Materiale
- QUALORA IL CODICE CARICATO CON LA RELAZIONE NON COMPILI CORRETTAMENTE, VERRÀ APPLICATA UNA PENALIZZAZIONE. Si ricorda che la valutazione del compito viene fatta, senza la presenza del candidato, sulla base dell'elaborato svolto durante l'appello. Non verranno corretti i compiti di cui non sarà stata inviata la relazione nei tempi stabiliti.

Domanda 1
Completo
Non valutata

Indicare quale tipologia di esame si intende svolgere, selezionando UNA sola scelta tra a (12 punti) e b (18 punti).
E' possibile, in aggiunta, un'ulteriore opzione c, da selezionare SOLO nel caso di laureando/a che intenda (e possa) presentare domanda di laurea per la sessione di MARZO/APRILE 2025.

Il tal caso occorrerà, per poter anticipare la correzione della prova scritta e del successivo orale, CARICARE LA RELAZIONE NEGLI ELABORATI entro SABATO 22/2 alle 23:59.

Scegli una o più alternative:

- [ ] a. 12 Punti - Semplificato
- [x] b. 18 Punti - Completo
- [ ] c. Selezionare SOLO nel caso di laureando/a che intenda (qualora si superi l'esame) presentare domanda di laurea nella sessione di MARZO/APRILE 2025.
  Il tal caso occorrerà, per poter anticipare la correzione della prova scritta e del successivo orale.
  CARICARE LA RELAZIONE entro SABATO 22/2 alle 23:59.

Risposta parzialmente esatta.

Ignorare il feedback di questa domanda.

Le risposte corrette sono: 12 Punti - Semplificato, 18 Punti - Completo,
Selezionare SOLO nel caso di laureando/a che intenda (qualora si superi l'esame) presentare domanda di laurea nella sessione di MARZO/APRILE 2025.
Il tal caso occorrerà, per poter anticipare la correzione della prova scritta e del successivo orale. CARICARE LA RELAZIONE entro SABATO 22/2 alle 23:59.

Domanda 2

Risposta non data

Punteggio max.: 3,00

È dato un ADT List, in grado di rappresentare una lista ordinata di interi. Si modifichi la funzione listDoubleItem che, data la lista e una chiave (un valore intero), cerchi la chiave nella lista e, se la trova, raddoppi il valore della chiave e sposti di conseguenza l'elemento nella nuova posizione corretta in lista. Benché tale operazione possa essere realizzata da una cancellazione seguita da un inserimento, si chiede di realizzarla mediante un solo attraversamento della lista (quindi una sola funzione). La funzione ritorna un risultato logico (vero/falso) per indicare se la chiave è stata trovata oppure no.

```
int listDoubleItem (List l, int k) {
link x, p;
int found=0, done=0;

for (x=l->head, p=NULL; x!=NULL&&!done;              ) {
 if (!found && x->val==k) { // rimuovi
  found = 1;

 }
 else if (found &&          ) { // inserisci
  done=1;

 }
 else {

 }
}
return        ;
}
```

Domanda 3

Risposta non data

Punteggio max.: 4,00

È dato un BST (tipo ADT BST). Si scriva una funzione che verifichi se la proprietà funzionale è rispettata, cioè che si tratti “proprio” di un BST e non “solo” di un albero binario.

```
int BSTcheckOrder (BST b) {


}
```

Domanda 4

Risposta non data

Punteggio max.: 5,00

È dato un Grafo orientato realizzato con liste adiacenza (tipo ADT Graph). Si scriva una funzione GRAPHcheckHam che, ricevuti come parametro un grafo e un elenco di vertici (un vettore di interi e la sua dimensione) verifichi se l’elenco di vertici (secondo l’ordine proposto) rappresenti un cammino Hamiltoniano. Si ricorda che un cammino Hamiltoniano è un cammino aciclico contenente tutti i vertici del grafo. Non è necessario definire la struttura dati.

```
//scrivere qui il codice...
```

```
int GRAPHcheckHam (Graph g, int *p, int n) {



}
```

Informazione

PAGINA DI INTERMEZZO

La prova da 12 punti termina prima di questa pagina di intermezzo.

La prossima domanda rappresenta l'inizio della prova da 18 punti.

Informazione

Descrizione del problema

Un crucipuzzle è un gioco enigmistico che consiste nel cercare delle parole, generalmente attinenti ad un tema e presenti in un elenco, all'interno di una griglia di lettere.

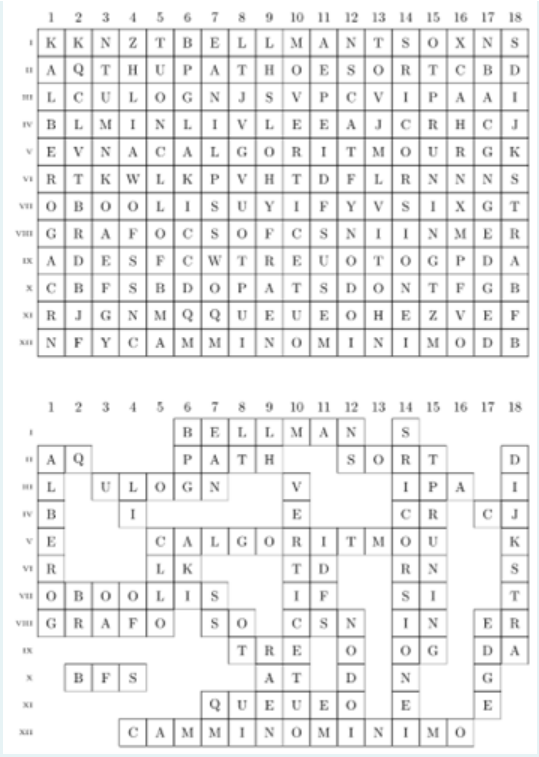
La griglia di gioco è una scacchiera rettangolare di dimensione R x C contenente solo lettere maiuscole, memorizzata nel file griglia.txt con il seguente formato:

- Sulla prima riga appare la coppia di interi R e C
- Seguono R righe di C caratteri, senza spazi, a rappresentare la griglia di gioco.

Le parole da ricercare sono memorizzate in un file parole.txt, di lunghezza non nota.

Ogni riga del file parole.txt riporta una coppia parola valore, dove parola è una stringa senza spazi di massimo 15 caratteri e valore è un intero positivo a rappresentare il valore della parola.

La figura seguente rappresenta la griglia e un insieme di parole selezionate dalla griglia stessa. Non si rappresenta il file parole.txt.



Le regole del gioco solo le seguenti:

- Le parole possono essere cercate nel solo verso di lettura in orizzontale (da sinistra a destra), verticale (dall'alto in basso) o diagonale (da sinistra a destra e dall'alto in basso). Una parola deve essere trovata nella sua interezza: non sono ammessi match parziali
- Data una coppia di parole trovate nello schema, le due parole possono incrociarsi condividendo al massimo una lettera. Non sono ammesse sovrapposizioni di sottostringhe di lunghezza superiore a un carattere
- Il valore della "soluzione" è pari alla somma dei valori delle parole trovate nella griglia
- Non è detto che sia possibile individuare tutte le parole dell'elenco nello schema di gioco

Nell'esempio in figura è presentata una generica griglia di gioco e una sua versione "risolta" in cui sono state individuate 21 parole. Prestare attenzione a come la parola SORT (Oriz.,II,12) non possa essere trovata in (Diag.,VII,7) poiché la sovrapposizione con QUICKSORT (Diag.,II,2) violerebbe uno dei vincoli di cui sopra (massimo un carattere in comune data una coppia di parole).

Domanda 5

Completo

Punteggio max.: 4,00

Struttura dati

Definire e implementare delle opportune strutture dati per rappresentare:

- la griglia di gioco (tipo GRID)
- l'elenco di parole da cercare (tipo WORDS)
- una soluzione al problema di ricerca (tipo SOL)

In aggiunta alle tre strutture dati esplicitamente richieste, è possibile definire tutti i tipi ausiliari ritenuti opportuni, a supporto delle tre strutture principali. NON è richiesta la funzione di lettura da file.

Attenzione! Si consiglia l'uso degli spazi al posto delle tabulazioni per l'indentazione del codice, dal momento che il carattere TAB viene utilizzato per la navigazione della pagina da parte della piattaforma.

```
//scrivere qui il codice...
```

//GRID.h

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct grid_s{

  int nr, nc;

  char **mat;

}Grid;


Grid GRIDinit(int nr, int nc);

Grid GRIDfile(FILE *fp);

void GRIDfree(Grid g);
```

```
//GRID.c

#include "GRID.h"
```

```
Grid GRIDinit(int nr, int nc){

  int i;

  Grid g = malloc(sizeof(*g));

  g.nr = nr;

  g.nc = nc;

  g.mat = (char **) malloc (nr * sizeof(char *));

  for(i=0; i< nr; i++)

    g.mat[i] = malloc (nc * sizeof(char));

  return g;

}

Grid GRIDfile(FILE *fp); //NON richiesta

void GRIDfree(Grid g){

  if(g==NULL)

    return;

  int i;

  for(i=0; i<g.nr; i++)

    free(g.mat[i]);
```

```
  free(g.mat);

  free(g);

}
```

```
//WORDS.h
```

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>
```

#define MAXL 16

```
typedef struct word_s{

  char parola[MAXL];

  int lun, valore;

}Word;

typedef struct words_s{

  Word *vettW;

  int numW;

}Words;

Words WORDSinit(int n);

Words WORDSfile(FILE *fp);

void WORDSfree(Words w);

```

```
//WORDS.c

#include "WORDS.h"
```

```
Words WORDSinit(int n){

  Words w = malloc (sizeof(*w));

  w.numW = n;

  w.vettW = malloc (n * sizeof(Word));

  return w;

}

```

```
Words WORDSfile(FILE *fp); //NON richiesta

void WORDSfree(Words w){

  if(w!=NULL){

    free(w.vettW);

    free(w);
```

```
  }

  return;

}
```

```
//SOL.h

#include "GRID.h"

#include "WORDS.h"

typedef enum{O, V, D, invalid} direzione;

typedef enum{VERO, FALSO} boolean;

typedef struct sol_s{

  WORDS parole;

  boolean valido;

  int valore;

}Sol;

Sol SOLinit(int n);

Sol SOLfree(Sol s);
```

```
//SOL.c

#include "SOL.h"
```

```
Sol SOLinit(int n){

  Sol s = malloc (sizeof(*s));

  s.parole = WORDSinit(n);

  s.valido = FALSE;

  s.valore = 0;

  return s;

}

Sol SOLfree(Sol s){

  if(s!=NULL){

    WORDSfree(s.parole);

    free(s);

  }

}
```

Domanda 6

Completo

Punteggio max.: 6,00

Problema di verifica

Un insieme di parole è valido se le parole che lo compongono si trovano nella griglia di gioco e rispettano i vincoli di orientamento e sovrapposizioni ammessi indicati in precedenza. Si scriva una funzione di verifica che, dato un elenco di parole (tipo WORDS) verifichi se sia un insieme di parole valido.

Attenzione! Si consiglia l'uso degli spazi al posto delle tabulazioni per l'indentazione del codice, dal momento che il carattere TAB viene utilizzato per la navigazione della pagina da parte della piattaforma.

```
//scrivere qui il codice...

//verificare se parola è interna alla griglia secondo ricerca orizz. vert. diag.
//massimo 1 cella condivisa con ogni altra parola,
//se tutte le parole sono presenti

//SOL.c
//matrice ausiliaria per verificare se una cella è gia stata occupata
int **matriceINIT(int nr, int nc){
  int m[nr][nc];
  int i, j;
  for(i=0; i<nr; i++)
    for(j=0; j<nc; j++)
      m[i][j] = 0;
  return m;
}



int ricercaW2(Grid g,Word w, int **occ,direzione dir, int i, int j){
int pos, trovato = 1;
  switch(dir){
  case O:{
   for(pos=0; pos< w.lun; pos++){
  if(mat[i][j+pos] != w.parola[pos])
    trovato = 0;
  }
  }break;
```

```
  case V:{
   for(pos=0; pos< w.lun; pos++){
  if(mat[i+pos][j] != w.parola[pos])
    trovato = 0;
  }
  }break;
```

```
  case D:{
   for(pos=0; pos< w.lun; pos++){
  if(mat[i+pos][j+pos] != w.parola[pos])
    trovato = 0;
  }
  }break;
 default:
  break;
}
return trovato;
}
```

```
int ricercaW(Grid g, Word w, int **occ){
  direzione dir = invalid;
  int i, j, pos=0;
  for(i=0; i<g.nr; i++){
    for(j=0; j<nc; j++){
      pos=0;
      if(g.mat[i][j] == w.parola[pos]){
        if(g.mat[i+1][j] == w.parola[pos+1])
          dir = V;

         if(g.mat[i][j+1] == w.parola[pos+1])
            dir = O;
```

```
            if(g.mat[i+1][j+1] == w.parola[pos+1])
              dir = D;
            if(ricercaW2(g, w, occ, dir, i, j))
```

```
            return 1;
        }


    }

  }

return 0;
}
boolean checkValidita(Grid g, Sol s){
 int i, cnt = 0, **occupato = matriceINIT(g.nr, g.nc);
 boolean ok = FALSO;
 for(i=0; i<s.parole.numW ; i++){
   if( ricercaW(g, s.parole.vettW[i], occupato){
     cnt++;
   }
 }
 if(cnt == s.parole.numW)
   ok=VERO;

 return ok;
}
```

Domanda 7

Risposta non data

Punteggio max.: 8,00

Problema di ricerca e ottimizzazione

Scrivere una funzione ricorsiva in grado di individuare, a partire da una griglia e da un elenco di parole, la soluzione a valore massimo possibile, cioè l'insieme ottimo di parole (tra quelle in elenco). A fronte di soluzioni dal valore equivalente, si prediliga quella composta dal maggior numero di parole.