Corso: Algoritmi e strutture dati
Quiz: ASD2024 - IV Appello - Programmazione

PORTALE ESAMI

	Mattia Micheletta Merlin **s310522**
Iniziato	13 settembre 2024, 10:01
Stato	Completato
Terminato	13 settembre 2024, 11:41
Tempo impiegato	1 ora 40 min.
Valutazione	Non ancora valutato
Riepilogo del tentativo	i i 1 2 3 4 i i 5 6 7

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 6 sono dedicate all'esame semplificato (traccia da 12pt).

In particolare:

- la domanda 2 fa parte dell'esercizio da 2 punti.
- la domanda 3 fa parte dell'esercizio da 4 punti.
- le domande 4,5,6 fanno parte dell'esercizio da 6 punti.

Le domande dalla 7 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 13/09/2024, alle ore 23:59, mediante caricamento su Portale
- assicurarsi di caricare l'elaborato nella **Sezione Elaborati relativa all'a.a. *2023/24***.
- 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.

Scegli un'alternativa:

a. 18 Punti - Completo

b. 12 Punti - Semplificato

Risposta corretta.

Ignorare il feedback di questa domanda.

La risposta corretta è: 18 Punti - Completo

Domanda 2

Risposta non data

Punteggio max.: 2,00

Sono dati due vettori di interi ordinati in modo crescente e privi di ripetizioni. Si scriva una funzione che generi un vettore (allocato dinamicamente) contenente gli interi appartenenti al primo vettore e non al secondo. La funzione deve essere chiamata come segue

c = diffVett(a,na,b,nb,&nc);

a e b sono i due vettori, na e nb il numero di dati che contengono; c è il vettore risultato, allocato dinamicamente nella funzione, nc il numero di interi nel vettore risultato.

Si richiede di realizzare (SOLO) la funzione diffVett (quindi non del programma chiamante).

Domanda 3

Risposta non data

Punteggio max.: 4,00

E’ dato un BST avente come valori delle stringhe, che fungono anche da chiave di ricerca. Si scriva una funzione che determini la foglia a profonditá massima MAXF (in caso di uguaglianza, si selezioni la foglia con chiave maggiore). La funzione stampi a ritroso (quindi da foglia a radice) le chiavi sul cammino che connette la foglia MAXF alla radice. Il prototipo della funzione deve essere:

void BSTprintDeepest(BST b);

Si richiede, oltre alla funzione, la definizione del tipo BST (ADT di prima classe) e del tipo usato per il nodo.

Domanda 4

Risposta non data

Punteggio max.: 6,00

Una superficie piana rettangolare viene suddivisa in N righe e M colonne e rappresentata da una matrice NxM di caratteri, in cui si usa in carattere ‘0’ per rappresentare una casella libera (utilizzabile) e il carattere ‘1’ per una casella occupata (non utilizzabile).

Si vuole realizzare una funzione che, data la matrice e le coordinate di due caselle libere (di coordinate (r0,c0) e (r1,c1)), determini il percorso di lunghezza minima per connetterle: un percorso è dato da una sequenza di caselle libere adiacenti (aventi riga o colonna in comune) a due a due: detto in altri termini, un percorso si ottiene mediante tratti orizzontali o verticali comprendenti solo caselle libere. La lunghezza di un percorso è data dal numero di caselle che lo compongono. Del percorso ottimo va solo calcolata e ritornata come risultato la lunghezza.

La funzione abbia prototipo:

```
int minPath(char **area, int N, int M, int r0, int c0, int r1, int c1);
```

Esempio (piccolo): matrice 4x5 in cui si è rappresentato con delle il percorso ottimo (di lunghezza 5) che connette la casella (1,0) alla (2,3)

```
01000
xxx10
01xx0
00000
```

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

La Regione deve decidere la localizzazione di un servizio di Pronto Soccorso per N città. Tra queste occorre selezionare M < N città che possono essere sede di Pronto Soccorso. Obiettivo del problema è determinare le sedi di pronto soccorse ed assegnare ad ognuna di esse il servizio di altre città, dati vincoli e criteri per valutare costi e benefici.

Date

- le distanze tra tutte le coppie di città, come matrice NxN di interi (ogni città dista 0 da se stessa),

- una distanza massima tollerabile MAXD, tra sede di un pronto soccorso e una città servita,

- un numero minimo MINS di città servibili da ogni sede di pronto soccorso (siccome una sede di pronto soccorso serve ovviamente se stessa, MINS tiene conto solo delle altre città NON sede di pronto soccorso)

un insieme di M città sede di pronto soccorso è accettabile se sono rispettati i vincoli seguenti:

- per ognuna delle altre N-M città esiste almeno una sede di pronto soccorso (tra le M scelte) a distanza <= MAXD

- ognuna delle sedi di pronto soccorso è in grado di servire almeno MINS città non sedi di pronto soccorso (rispettando per ognuna il vincolo precedente).

Richieste del problema

A seguire una sintesi delle richieste del problema. Per ogni richiesta si troverà una domanda dedicata nelle sezioni a seguire con una descrizione più dettagliata per le richieste.

Strutture dati e letture

Definire opportune strutture dati per rappresentare:

- L’elenco delle città (tipo ELENCO)
- La matrice delle distanze (DISTMATR)
- Un insieme di sedi di pronto soccorso (tipo SEDI, vedere i punti successivi)
- Un’assegnazione di città alle sedi (tipo SERVIZI, la soluzione del problema di ottimizzazione)

In particolare, tutte le strutture dati precedenti devono essere ADT di prima classe. Si scriva la funzione caricaDati che ritorni un elenco di città (tipo ELENCO) e una matrice delle distanze (tipo DISTMATR), leggendo un file in cui la prima riga contiene il valore N, le N righe successive i nomi di N città, le ulteriori N righe successive la matrice quadrata delle distanze: ogni riga contiene N interi separati da spazi.

Esempio di file (per semplicitá si usano nomi di un solo carattere):

```
6
A
B
C
D
E
F
0 2 5 4 2 6
2 0 6
6 2 5
5 6 0
7 4 3
4 6 7
0 6 3
2 2 4
6 0 6
6 5 3
3 6 0
```

Problema di verifica

Si assuma di voler enumerare i possibili insiemi di M città sedi di pronto soccorso, mediante un algoritmo basato su combinazioni semplici. Si scriva la funzione checkSedi, in grado di verificare, nel caso terminale, l'accettabilità di una soluzione: NON si chiede l'algoritmo per generare le combinazioni ma solo la checkSedi. Questa deve ricevere, tra gli altri parametri (da decidere) la matrice delle distanze, la distanza massima MAXD, il numero minimo MINS, l'insieme ottenuto nel caso terminale (la soluzione, di tipo SEDI, da verificare).

Problema di ricerca e ottimizzazione

Dato un insieme valido di M sedi (tipo SEDI) di pronto soccorso, già verificato mediante checkSedi, si vuole determinare un partizionamento ottimo delle città servite, in modo tale che si rispettino i criteri seguenti:

- ogni città non sede di pronto soccorso viene assegnata a una (sola) sede di pronto soccorso, a distanza <= MAXD

- la distanza media tra le città servite e le sedi di pronto soccorso è minima

- a ognuna delle sedi di pronto soccorso si assegnano almeno MINS città.

Si scriva la funzione bestPart, che, utilizzando un algoritmo di partizionamento, dato l'elenco delle città, la matrice delle distanze, un insieme di M sedi di pronto soccorso e i valori MAXD e MINS, trovi la soluzione ottima e la ritorni come struttura dati (tipo SERVIZI). Non è necessario realizzare la funzione wrapper: è sufficiente rappresentare la funzione bestPart, che al suo interno deve usare due funzioni (da scrivere anche queste):

- checkPart: funzione di verifica da chiamare in un caso terminale

- prunePart: chiamata per fare pruning (si dica esplicitamente di quali vincoli o criteri si può fare pruning e di quali no).

Nell'esempio proposto in precedenza, supponendo che si siano selezionate come sedi di pronto soccorso {A,C}, che MAXD valga 4 e MINS valga 2, la soluzione sarebbe quella di assegnare {B,D} alla sede A e {E,F} alla sede C. Si riporta la matrice delle distanze, avendo evidenziato in rosso (sulle righe) le sedi e in grassetto le distanze compatibili con MAXD. Per la soluzione proposta, la distanza media tra le citta servite e le sedi è (2+4+4+3)/4 = 13/4.

	A	B	C	D	E	F
A	0	**2**	5	**4**	**2**	6
B	2	0	6	6	2	5
C	5	6	0	7	**4**	**3**

D	4	6	7	0	6	3
E	2	2	4	6	0	6
F	6	5	3	3	6	0

Domanda 5

Completo

Punteggio max.: 5,00

Strutture dati e acquisizione/lettura

Definire opportune strutture dati per rappresentare:

- L'elenco delle città (tipo ELENCO)
- La matrice delle distanze (DISTMATR)
- Un insieme di sedi di pronto soccorso (tipo SEDI, vedere i punti successivi)
- Un'assegnazione di città alle sedi (tipo SERVIZI, la soluzione del problema di ottimizzazione)

In particolare, tutte le strutture dati precedenti devono essere ADT di prima classe. Si scriva la funzione caricaDati che ritorni un elenco di città (tipo ELENCO) e una matrice delle distanze (tipo DISTMATR), leggendo un file in cui la prima riga contiene il valore N, le N righe successive i nomi di N città, le ulteriori N righe successive la matrice quadrata delle distanze: ogni riga contiene N interi separati da spazi.

Esempio di file (per semplicitá si usano nomi di un solo carattere):

```
6
A
B
C
D
E
F
0 2 5 4 2 6
2 0 6
6 2 5
5 6 0
7 4 3
4 6 7
0 6 3
2 2 4
6 0 6
6 5 3
3 6 0
```

#include<stdio.h>

```
typedef struct ELENCO_s* ELENCO;

struct ELENCO_s{

int N; // numero totale di città
```

```
char** nome_città; // vettore di N stringhe contente i nomi delle città
};
char* getNome(ELENCO e, int i){
 return e->nome_città[i];
}

typedef struct DISTMATR_S* DISTMATR;
```

```
struct DISTMATR_S{
int N;
int** mat; // NxN matrix
};
int getM(DISTMATR m, int i, int j){
 return m->mat[i][j];
};

typedef struct SEDI_s* SEDI;
struct SEDI_s{
 int M;
 int* sedi; // elenco di M int contenenti l'indice delle città sede di p.s. nell'elenco
};

// liste usate per le città N-M da assegnare alle N città con p.s.
typedef struct node* link;
struct node{ int i; link next; }; // indice della città nell'elenco
// funzinoe per aggiungere un elemento al fondo di una lista (ritorna la nuova lista)
link addToList(link l, int i){
 link new = malloc(sizeof(struct node);
 new->i = i; new->next == NULL;
 if (l == NULL){ return new; }
 link t = l;
 while (t->next != NULL){ t = t->next; }
 t->next = new;
```

```
 return l;

}

// removes last element form list

link removeFromList(link l){

 // ...

}

struct SERVIZI_s{

 SEDI sedi_ps; // città sedi del p.s. di riferimento

 link* v; // vettore di M liste, ogni lista contiene le città senza p.s. assegnate alle rispettive città con p.s.

 int* l; // lunghezza delle liste

 int dist_media; // distanza media p.s. - città senza p.s.

}

SERVIZI serviziInit(SEDI s){

 SERVIZI new = malloc(sizeof(struct(SERVIZI_s));

 new->sedi_ps = s;

 new->l = malloc(sizeof(int) * s->M);

 for (int i=0; i<s->M; i++){ new->l[i] = NULL; }

 new->v = calloc(s->M, sizeof(link));

 new->dist_media = -1;

 return new;

}

// elenco e distmat passaty by reference e NON ancora inizializzati con malloc

void caricaDATI(FILE* fp, ELENCO* e, DISTMAT* m){

 e = malloc(sizeof(struct ELENCO_s));
```

```
m = malloc(sizeof(struct DISTMAT_s));

fscanf(fp, "%d", &(e->N));

m->N = e->N;

for (int i=0; i<e->N; i++){

  fscanf(fp, "%s", e->nome_città[i]);

}

for (int i=0; i<m->N; i++){

  for (int j=0; j<m->N; j++){

    fscanf(fp, "%d", m->mat[i][j]);

  }

}

}
```

Domanda 6

Completo

Punteggio max.: 5,00

Problema di verifica

Si assuma di voler enumerare i possibili insiemi di M città sedi di pronto soccorso, mediante un algoritmo basato su combinazioni semplici. Si scriva la funzione checkSedi, in grado di verificare, nel caso terminale, l'accettabilità di una soluzione: NON si chiede l'algoritmo per generare le combinazioni ma solo la checkSedi. Questa deve ricevere, tra gli altri parametri (da decidere) la matrice delle distanze, la distanza massima MAXD, il numero minimo MINS, l'insieme ottenuto nel caso terminale (la soluzione, di tipo SEDI, da verificare).

```
// utility function: returns 1 if an int is in the vectro, 0 otherwise

int elInVector(int* v, int lenght; int el){

  for (int i=0; i<lenght; i++){

    if (v[i] = el){return 1;}

  }

  return 0;

}

int* getCittàSenzaPS(SEDI s, int N, int M){

  int N_M = N - M;

  int* città_no_ps = malloc(sizeof(int) * (N_M));

  int index = 0;

  // per ogni città (indice i)...
```

```
  for (int i=0; i<N; i++){

    // se non è nelle città con p.s. (sedi)

    if (!elInVector(s->sedi, sM, i)){ città_no_ps[index] = i; index++; }

  }

  return città_no_ps;

}

// per ogniuna delle altre N-M città esiste almeno una città con p.s. a distanza minore di MAXD
```

// ogni città con p.s. deve servire almeno MINS città senza p.s. (a distanza < di MAXD)

// returns 0 if the sol is not valid, 1 if valid

```
int checkSedi(DISTMATR m, int MAXD, int MINS, SEDI s){

 int N_M = m->N - s->M;

 int* città_no_ps = getCittàSenzaPS(s, m->N, s->M);

 int found;

 for (int i=0; i<N_M; i++){

   found = 0;

   for (int j=0; j<s->M; j++){

       if (m->mat[città_no_ps[i]][s->sedi[j]] <= MAXD){ found = 1; break; }

   }

    if (found = 0){ return 0; } // if there is not even one città con sedi at required distance

 }

  int n_servite;

 for (int i=0; i<s->M; i++){

   n_servite = 0;

   for (int j=0; j<N_M; j++){
     if (m->mat[città_no_ps[i]][s->sedi[j]] <= MAXD){ n_servite++; }

   }

   if (n_servite < MINS){ return 0; }

 }

 return 1; // se ha passato tutti i check...

}
```

Domanda 7

Completo

Punteggio max.: 8,00

Problema di ricerca e ottimizzazione

Dato un insieme valido di M sedi (tipo SEDI) di pronto soccorso, già verificato mediante checkSedi, si vuole determinare un partizionamento ottimo delle città servite, in modo tale che si rispettino i criteri seguenti:

- ogni città non sede di pronto soccorso viene assegnata a una (sola) sede di pronto soccorso, a distanza <= MAXD

- la distanza media tra le città servite e le sedi di pronto soccorso è minima

- a ognuna delle sedi di pronto soccorso si assegnano almeno MINS città.

Si scriva la funzione bestPart, che, utilizzando un algoritmo di partizionamento, dato l'elenco delle città, la matrice delle distanze, un insieme di M sedi di pronto soccorso e i valori MAXD e MINS, trovi la soluzione ottima e la ritorni come struttura dati (tipo SERVIZI). Non è necessario realizzare la funzione wrapper: è sufficiente rappresentare la funzione bestPart, che al suo interno deve usare due funzioni (da scrivere anche queste):

- checkPart: funzione di verifica da chiamare in un caso terminale

- prunePart: chiamata per fare pruning (si dica esplicitamente di quali vincoli o criteri si può fare pruning e di quali no).

Nell'esempio proposto in precedenza, supponendo che si siano selezionate come sedi di pronto soccorso {A,C}, che MAXD valga 4 e MINS valga 2, la soluzione sarebbe quella di assegnare {B,D} alla sede A e {E,F} alla sede C. Si riporta la matrice delle distanze, avendo evidenziato in rosso (sulle righe) le sedi e in grassetto le distanze compatibili con MAXD. Per la soluzione proposta, la distanza media tra le citta servite e le sedi è (2+4+4+3)/4 = 13/4.

	A	B	C	D	E	F
A	0	**2**	5	**4**	**2**	6
B	2	0	6	6	2	5
C	5	6	0	7	**4**	**3**
D	4	6	7	0	6	3
E	2	2	4	6	0	6
F	6	5	3	3	6	0

```
// runnning out of time...
```

```
SERVIZI sol_from_appartenenza_c_n_ps(DISTMATR m, SEDI s, int* appartenenza_c_n_ps){
```

```
  int N_M = m->N - s->M;
```

```
  int* città_no_ps = getCittàSenzaPS(s, m->N, s->M);
```

```
  SERVIZI sol = serviziInit(s->M);
```

```
  sol->sedi_ps = s;
```

for(int i=0; i<N_M; i++){

```
    addToList(sol->v[appartenenza_c_n_ps[i]], città_no_ps[i]);

   sol->l[appartenenza_c_n_ps[i]]++;

  }

  // calcola dist_media

  for (int i=0; i<s->M; i++){



  }

  return sol;

}



// return 1 if you can prune, 0 otherwise

// prune if

int prunePart(DISTMATR m, SEDI s, int* appartenenza_c_n_ps, int MINS, int MAXD){

---

  SERVIZI cuurent_sol = sol_from_appartenenza_c_n_ps(m, s, appartenenza_c_n_ps);

  int MAXS = m->N - MINS * s->M;
```

for (int i=0 ; i<s->M; i++){

```
   if (current_sol->l[i] < MINS || current_sol->l[i] > MAXS){ return 1; }

  }

  return 0;

}

void bestpart_r(DISTMATR m, SEDI s, int MINS, int MAXD, int pos, SERVIZI* best_sol, int*
appartenenza_c_n_ps){

  if (prunePart(m, s, appartenenza_c_n_ps, MINS, MAXD)){ return; }

  // condizione terminazione

  if (pos == s->M - m->N){

    SERVIZI cuurent_sol = sol_from_appartenenza_c_n_ps(m, s, appartenenza_c_n_ps);
);

    if (checkPart(cuurent_sol) && cuurent_sol->dist_media < (*best_sol)->dist_media){

      (*best_sol) = current_sol; // doens't work, i should copy everthing, but i dont have time...

   }

   return;

}

  // per ogni possibile città con p.s. di appartenenza

  for (int i=0; i<s->M; i++){

---

   appartenenza_c_n_ps[pos] = s->sedi[i]; // assegnala ala città senza p.s. indicata da pos
```

```
    bestpart_r(m, s, MINS, MAXD, pos+1, best_sol, appartenenza_c_n_ps);
  }

}
```

SERVIZI bestpart(ELENCO e, DISTMATR m, SEDI s, int MINS, int MAXD){

```
  int M = s->M; int N = m->N;

  int* appartenenza_c_n_ps = malloc(sizeof(int) * (N-M));

  for (int i=0; i<N-M; i++){ appartenenza_c_n_ps[i] = -1; }
```

SERVIZI best_sol = serviziInit(s);

```
  bestpart_r(m, s, MINS, MAXD, 0, &best_sol, appartenenza_c_n_ps);

  // print bestsol
```

}