POLITECNICO DI TORINO

Algoritmi e programmazione

AP2021_IV_Programmazione

	ALBERTO HUGONIN 249803
Iniziato	mercoledì, 1 settembre 2021, 12:30
Terminato	mercoledì, 1 settembre 2021, 14:10
Tempo impiegato	1 ora 40 min.

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

In particolare:

- la domanda 2 fa parte dell'esercizio da 2 punti.
- le domande 3,4 fanno parte dell'esercizio da 4 punti.
- le domande 5,6,7 fanno parte dell'esercizio da 6 punti.

Le domande dalla 8 in poi sono dedicate all'esame completo (traccia 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 SABATO 04/09/2021, alle ore 23:59, mediante caricamento su Portale;
- Si ricorda di prestare attenzione a caricare la relazione sul corso di Algoritmi e Programmazione relativo all'A.A. 2020/21;
- 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

Punteggio max.: 1,00

Indicare quale tipologia di esame si intende svolgere.

- (a) 12 Punti - Semplificato
- (b) 18 Punti - Completo

Domanda 2

Completo

Punteggio max.: 100,00

Esercizio da 2 punti

Siano dati due vettori di interi positivi v1 e v2 di dimensioni d1 e d2 . Si scriva una funzione in grado di generare una matrice di dimensioni d1 x d2 in cui il contenuto della generica cella [i,j] sia la somma dell'elemento di posizione i di v1 e dell'elemento di posizione j di v2

Il prototipo della funzione sia quello presentato a seguire. L'allocazione della matrice deve avvenire dentro alla funzione. La matrice creata e i suoi contenuti devono essere disponili per chi invoca la funzione. Il prototipo va completato opportunamente in modo da poter rispondere alle specifiche della richiesta.

```
void f(int *v1, int *v2, int d1, int d2, ...);
```

Esempio:

$$v1 = \begin{pmatrix} 2 & 4 & 6 \end{pmatrix}$$
$$v2 = \begin{pmatrix} 1 & 3 & 5 & 7 \end{pmatrix}$$

Risultato:

$$Res = \begin{pmatrix} 3 & 5 & 7 & 9 \\ 5 & 7 & 9 & 11 \\ 7 & 9 & 11 & 13 \end{pmatrix}$$

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.

```
void f(int *v1, int *v2, int d1, int d2, int **matr) {
int i = 0;
int j=0;
//allochiamo il numero di righe della matrice
matr = malloc(d1*sizeof(int));
//allochiamo le colonne
for (i=0; i<d1; a++) {matr[i] = malloc(d2*sizeof(int));}

for (i=0; i<d1; i++) {
   for (j=0; j<d2; j++) {
       matr[i][j] = v1[i]+v2[j];
   }
}

}
```

Domanda 3

Completo

Non valutata

Definire una struttura dati adeguata a rappresentare una lista doppio linkata di interi come ADT I classe, dove il tipo lista si chiami LIST. **La lista definita non deve fare uso di sentinelle.**

Indicare esplicitamente in quale modulo/file appare la definizione dei tipi proposti.

La struttura così definita è quella da usare per completare la seconda domanda di questo esercizio.

Non è ammesso l'uso di funzioni di libreria.

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.

LIST.h

```
//LIST.h
//scrivere qui il codice

typedef struct lista_ *Lista;
```

LIST.c

```
//LIST.c
//scrivere qui il codice
#include <stdio.h>
#include <stdlib.h>

#include "list.h"

typedef struct nodo_ *link;

struct nodo_ {
int k;
link next;
link prev;
};

struct lista_ {
//puntatori
link testa;
link coda;
};
```

Domanda 4

Completo

Punteggio max.: 100,00

Implementare una funzione caratterizzata dal seguente prototipo:

```
void purge(LIST l, int div)
```

la funzione purge cancella dalla lista tutti i nodi che contengono un numero non divisibile per div

Non è ammesso l'uso di funzioni di libreria. Attenersi al prototipo dato.

Esempio

div = 3

Input

$9 \leftrightarrow 8 \leftrightarrow 7 \leftrightarrow 6 \leftrightarrow 5 \leftrightarrow 4 \leftrightarrow 3 \leftrightarrow 2 \leftrightarrow 1 \leftrightarrow 0$

Output

$9 \leftrightarrow 6 \leftrightarrow 3 \leftrightarrow 0$

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.

```
void purge(LIST l, int div) {
if (l->n == 0) { return null; }
link corrente = l->testa;

//se la divisione per div ritorna del resto allora cancelliamo il nodo testa
if ((l->testa->k)%div != 0) {
   l->testa->next->prev = NULL;
   l->testa = corrente->next;

   if (corrente == l->coda) {
   l->coda = l->testa;
  }
  free(corrente);
   }

corrente = l->testa;

while (corrente->next != NULL) {
   corrente = corrente->next;
   link back = corrente;
   if (corrente->k)%div != 0) {
       //possiamo essere per forza al centro della lista, l'ultimo elemento non viene m
ai preso in considerazione dal ciclo
       corrente->next->prev = corrente->prev;
       corrente->prev->next = corrente->next;
       corrente = back->next;
       free(back);
   }
}
if (corrente->k)%div == 0) {
return;
}
//il ciclo esce all'ultimo elemento che dobbiamo cancellarlo a parte
free(corrente->next);
corrente->next = NULL;
l->coda = corrente;
}
```

Domanda 5

Completo

Punteggio max.: 100,00

Data una funzione wrapper con il seguente prototipo

```
void solve(char *target, part *P, int nParts);
```

Dove target è una stringa obiettivo da costruire sfruttanto le stringhe disponibili nel vettore di strutture P, di tipo part, lungo nParts.

part è una struttura definita come segue:

```
typedef struct {
    char *s;
    int pos;
    int costo;
} part;
```

usata per rappresentare una possibile "parte" della stringa obiettivo.

Una parte è caratterizzata da una sottostringa s, che può essere posizionata solo in posizione pos. Ogni parte ha un certo costo associato.

Scrivere una funzione ricorsiva che determini se è possibile generare la stringa target sfruttando le stringhe disponibili in P minimizzando il costo totale di costruzione. A parità di costo prediligere la soluzione che fa uso di meno parti.

Es. completo:

```
target = "persona"

P = {
    {"p", 0, 1},    // p0
    {"pers", 0, 5},  // p1
    {"er", 1, 4},   // p2
    {"ers", 1, 4},  // p3
    {"sa", 3, 1},   // p4
    {"so", 3, 2},   // p5
    {"ato", 0, 1},  // p6
    {"on", 4, 2},   // p7
    {"ona", 4, 3},  // p8
    {"a", 6, 1}     // p9
}

Usando p0, p3, p7, p9: costo 8, cardinalità soluzione 4

Usando p0, p3, p8: costo 8, cardinalità soluzione 3

Usando p1, p8: costo 8, cardinalità soluzione 2 (soluzione ottima)
```

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.

```
void solve(char *target, part *P, int nParts) {

int *val = malloc(nParts*sizeof(int));
for (int a=0; a<nParts; a++) {
val[a] = a;
}
int *sol = malloc(strlen(target)*sizeof(int));
int *mark = calloc(nParts,sizeof(int));
int best_costo = MAX_INT; //ricordare di includere limits.h
int costo = 0;
int *best_sol = malloc(strlen(target)*sizeof(int));
SolveR(val,mark,nParts,sol,strlen(target),0,P,target,costo, *best_costo, best_sol);

}
```

```
//modello con disposizioni semplici
void solveR( int *val,  int *mark, int n, int *sol, int k, int pos, parts *P, char *target, in
t *best_costo, int costo, int *best_sol) {

int i = 0;
if (pos>=k) {
//caso terminale
if (cost > *best_costo) {
*best_costo = cost;
for (int a=0; a<
//sovra
}
}

for (i=0; i<n; i++) {
if (mark[i] == 0) {
   //pruning, prima di prendere elemento vediamo se va bene se no lo saltiamo
   if (check(sol,pos,P,targer) == 0) {
   mark[i]=1;
   //sol[pos] contiene l'indice del vettore P dell'elemento preso
   sol[pos] = val[i];
   solveR(val,mark,n,sol,k,pos+1,P,target,best_costo,costo+P[sol[pos]]->costo);
   mark[i] = 0;
   }

}
return void;
}
```

```
int check(int *sol, int pos, parts *P, char *target) {
if (strlen(parts[sol[pos]]) > strlen (target)) {
return 1;
}
char *stringa = parts[sol[pos]]->s;
int inizio = parts[sol[pos]]->pos;

//andiamo a vedere se la sottostringa è compatibile con parte della stringa iniziale
for (int a=0; a<strlen(stringa); a++) {
if (target[inizio+a] != stringa[a]) {
return 1;
}
}
//adesso vediamo se il posto era già occupato, questo controllo è meglio metterlo a
ll'inizio
for (int a=0; a<pos; a++) {
if (parts[sol[a]]->pos == inizio) {
return 1;
}
}

return 0;
}
```

Domanda 6

Completo

Non valutata

Giustificare la scelta del modello combinatorio adottato.

Utilizzo le disposizioni semplici, dal vettore P estraggo senza reinserimento un elemento, ne estraggo fino al massimo alla lunghezza della stringa perchè minimo le sottostringe sono di grandezza 1

Si poteva anche valutare le combinazioni semplici ma in questo caso l'ordine conta ad esempio se prendo in ordine p, pers, e infine er prendere p esclude pers, se avessi preso prima er sarebbe stato lui a escludere pers.

Estraggo elemento da p verifico se si puo' inserire se si allora lo metto se prima non ho messo già qualcos'altro che occupa il suo posto

Domanda 7

Completo

Non valutata

Giustificare la scelta del/dei criteri di pruning adottati, o il motivo della loro assenza.

prima di inserire un elemento verifico che sia valido e che non ci sia già un elemento scelto prima che occupa il suo posto

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

Inizio prova da 18 punti

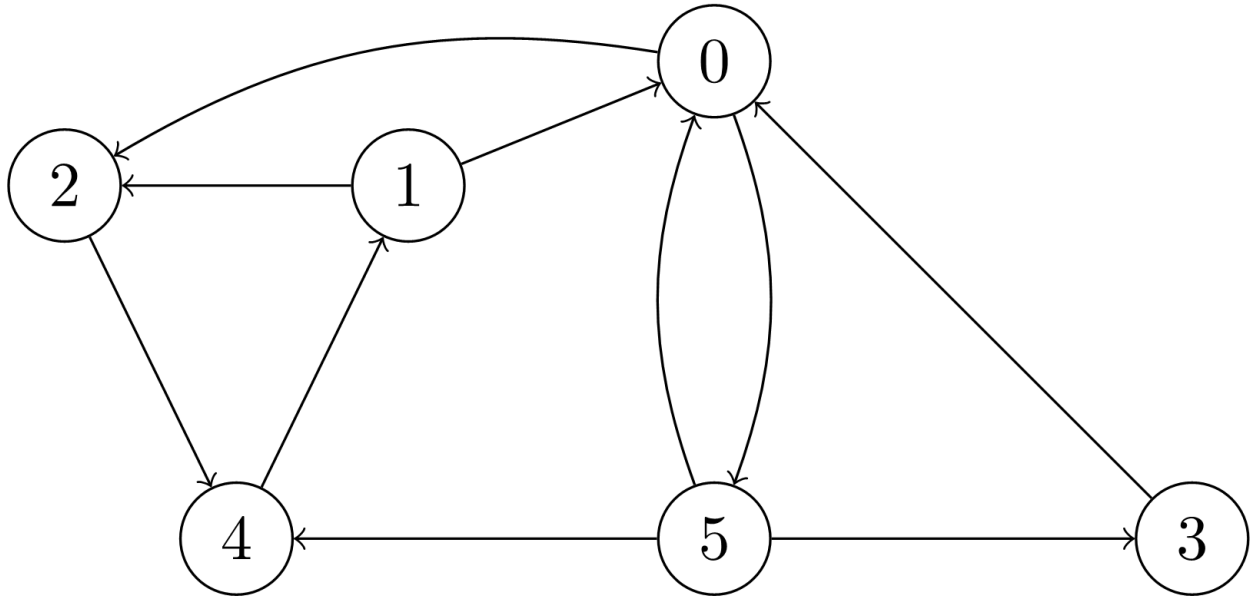
Descrizione

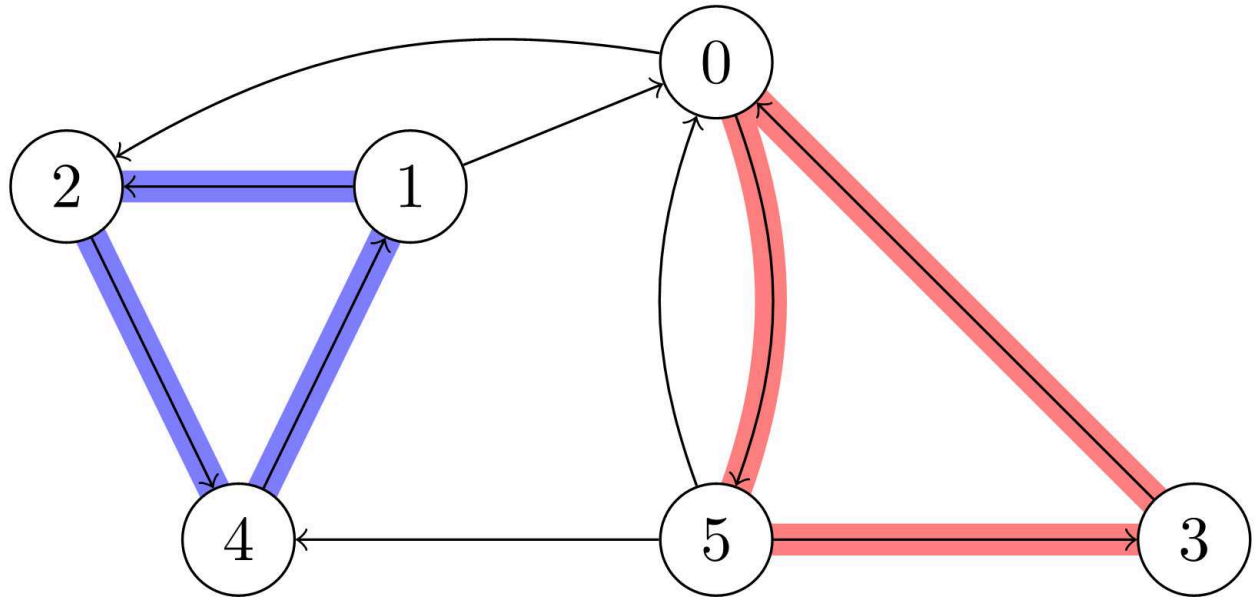
Dato un grafo G = (V,E) orientato e connesso, si definisce *copertura ciclica* un insieme di cicli la cui unione copre tutti i vertici del grafo. Perché un vertice sia ritenuto coperto, deve quindi far parte di almeno uno dei cicli inclusi nell'insieme preso in considerazione. Nel contesto dell'appello corrente, quando di parla di cicli, si intende sempre e solo ***cicli semplici***.

Una copertura può essere *sovrapposta*, se lo stesso vertice appare in più cicli, o *disgiunta*, nel caso contrario.

Esempio

A seguire è riportato un grafo di esempio e una sua copertura disgiunta che fa uso di due cicli.





Input

Il grafo è memorizzato in un file testualegrafo.txt organizzato come segue:

- Sulla prima riga appare una coppia di interi V E a rappresentare il numero di vertici e archi del grafo;
- Seguono E righe, ognuna della quale riporta una coppia v w a rappresentare i singoli archi del grafo;
- Si assuma che i vertici siano identificati univocamente con un valore intero nell'intervallo 0 .. V-1.

L'insieme dei cicli è anch'esso fornito, ed è memorizzato in un file testualecicli.txt organizzato come segue:

- I cicli sono riportati in ragione di uno per riga. Il numero di cicli non è noto a priori;
- Ogni i-esima riga presenta un primo valore intero len_i, rappresentante la lunghezza del ciclo, seguito da len_i valori interi a rappresentare i vertici che compongono il ciclo stesso.

A seguire sono riportati due file completi che riprendono i contenuti dell'esempio presentato in precedenza.

grafo.txt

```
6 10
0 2
0 5
1 0
1 2
2 4
3 0
4 1
5 0
5 3
5 4
```

cicli.txt

```
4 0 5 4 1
3 0 5 3
2 0 5
4 0 2 4 1
3 1 2 4
```

Richieste

A seguire è riportata una versione sintentica delle richieste dell'appello. Per ognuna delle richieste è prevista una domanda dedicata nelle pagine a seguire.

- Acquisire il grafo in una opportuna struttura dati leggendo il filegrafo.txt;
- Definire una struttura dati idonea a memorizzare l'elenco di cicli riportato incicli.txt e acquisire i contenuti del file;
- Dato un elenco di vertici in un formato a discrezione del candidato, scrivere una funzione che verifichi se questo rappresenta un ciclo;
- Dato un elenco di cicli in un formato a discrezione del candidato, scrivere una funzione che verifichi se questo rappresenta una copertura ciclica, disgiunta o meno;
- Individuare, se possibile, una copertura ciclica a cardinalità minima utilizzando l'elenco di cicli fornito in precedenza. A parità di cardinalità, prediligere una copertura disgiunta.

Domanda 8

Risposta non data

Punteggio max.: 1,00

Strutture dati: grafo

Acquisire in una opportuna struttura dati idonea i contenuti del file grafo.txt.

Se si fa uso della struttura grafo ADT vista a lezione, riportare le eventuali modifiche apportate alla sua definizione.

Domanda 9

Risposta non data

Punteggio max.: 1,00

Strutture dati: elenco di cicli

Definire una struttura dati idonea a memorizzare un numero arbitrario di cicli e implementare le funzioni opportune per acquisire i contenuti del file cicli.txt.

Domanda 10

Risposta non data

Punteggio max.: 1,00

Verifica Cicli

Dato un elenco ordinato di vertici, in un formato a discrezione del candidato, verificare se questo rappresenta un ciclo.

Domanda 11

Risposta non data

Punteggio max.: 1,00

Problema di ricerca e ottimizzazione

Individuare, se possibile, un insieme a cardinalità minima di cicli, scelti dall'elenco letto in precedenza, la cui unione copre tutti i vertici del grafo. A parità di cardinalità, prediligere una copertura disgiunta.