Laboratorio di Algoritmi e Strutture Dati

Docenti: M. Torelli, S. Aguzzoli

Appello del 22 gennaio 2007

Progetto "Templi"
Consegna entro l'11 febbraio 2007

Il problema

In una regione lontana si erigono templi votivi per le divinità venerate dagli indigeni. Un tempio è costituito da un certo numero di piani, ciascun piano è formato da un unico blocco. Quando un tempio è costruito, può essere dedicato a una divinità.

Formalmente, sia $\mathcal{A}$ l'alfabeto $\{a, b, \ldots, z\}$. Un *blocco* è una tripla $b = (\sigma_b, p(b), r(b))$, dove:

- σ_b è una stringa su $\mathcal{A}$ che denota il nome del blocco.
- $p(b)$ è un intero positivo che denota il *peso* del blocco.
- $r(b)$ è un intero positivo che denota la *resistenza* del blocco.

La resistenza di un blocco indica il peso complessivo che un blocco è in grado di reggere.

Un *tempio* è una coppia $T = (\sigma_T, P_T)$ dove:

- σ_T è una stringa su $\mathcal{A}$ che denota il nome del tempio.
- P_T è una successione di blocchi $P_T = (b_1, b_2, \ldots, b_k)$ tale che, per ogni $1 \leq i \leq k$, b_i è il blocco che forma il piano i del tempio.

Perché il tempio non collassi, ogni blocco deve essere in grado di sopportare il peso complessivo dei blocchi nei piani soprastanti, quindi, per ogni $i \in \{1, 2, \ldots, k-1\}$, deve valere:

$$r(b_i) \quad \geq \quad \sum_{j=i+1}^{k} p(b_j)\,.$$

L'*altezza* di T è il numero k dei piani che lo compongono. Quando un tempio viene costruito a partire da un insieme di blocchi $\mathcal{B}$, viene costruito il tempio che ha *altezza massima* fra tutti i templi costruibili usando i blocchi in $\mathcal{B}$ (ogni blocco può essere usato al massimo una volta). Nel caso esista più di una soluzione, ne viene scelta una.

Esempio 1

Supponiamo di avere a disposizione i blocchi a, b, c, d ed e, i cui pesi e resistenze sono descritti nella tabella sottostante.

Blocco	Peso	Resistenza
a	10	31
b	20	17
c	8	1
d	16	1
e	2	35

Consideriamo l'insieme dei blocchi $\mathcal{B}_1 = \{a, b\}$. L'altezza massima di un tempio costruibile da $\mathcal{B}_1$ è 2. È infatti possibile costruire i templi

$$T_1 \ = \ (\sigma_1, \, (a, b)) \qquad\qquad T_2 \ = \ (\sigma_2, \, (b, a))$$

Ad esempio, T_1 è un tempio di nome σ_1 formato dai piani a (primo piano) e b (secondo piano). Sia $\mathcal{B}_2 = \{c, d, e\}$. I templi costruibili da $\mathcal{B}_2$ hanno altezza massima 2 e sono:

$$T_3 \ = \ (\sigma_3, \, (e, c)) \qquad T_4 \ = \ (\sigma_4, \, (e, d))$$

Infine, se l'insieme dei blocchi è $\{a, b, c, d, e\}$, l'altezza massima è 4 e sono costruibili due templi:

$$T_5 \ = \ (\sigma_5, \, (a, b, e, c)) \qquad T_5 \ = \ (\sigma_5, \, (a, e, b, c))$$

Una *divinità* D è una coppia $D = (\sigma_D, B_D)$, dove:

- σ_D è una stringa finita su $\mathcal{A}$ che denota il *nome* della divinità.
- B_D è un insieme finito di stringhe su $\mathcal{A}$ tale che, per ogni $\beta \in B_D$, β è il nome di un blocco. B_D può anche essere l'insieme vuoto.

Diciamo che la divinità D *protegge* un tempio T se esiste $\beta \in B_D$ tale che β è il nome di un blocco che è un piano di T.

Fra le divinità è definita una gerarchia descritta da una *relazione d'ordinamento stretto* $<$. Questo significa che $<$ deve verificare le seguenti proprietà degli ordinamenti stretti:

(P1). Per ogni divinità D_1 e D_2, se $D_1 < D_2$, allora non vale $D_2 < D_1$.

(P2). Per ogni divinità D_1, D_2 e D_3, se $D_1 < D_2$ e $D_2 < D_3$, allora $D_1 < D_3$.

La gerarchia fra le divinità è stabilita dall'utente e il programma deve controllare che le asserzioni del tipo $D_1 < D_2$ inserite dall'utente non violino le proprietà scritte sopra.

Esempio 2

Supponiamo che la relazione fra divinità sia così definita:

```
nettuno < mercurio      mercurio < giunone     mercurio < minerva    mercurio < venere

saturno < venere        marte < minerva        minerva < giove       venere < giove

nettuno < urano         urano < giove
```

In questa situazione l'utente non può aggiungere a $<$ l'asserzione `minerva < nettuno`. Infatti, per la proprietà (P2) si ha che `nettuno < minerva`, quindi per (P1) non può valere `minerva < nettuno`. È possibile invece inserire ad esempio `minerva < venere`, così come `nettuno < marte`, `saturno < nettuno`, `giove < giunone`, ecc.

Per dedicare un tempio $T = (\sigma_T, P_T)$ a una divinità, evitando di creare discordie, gli indigeni seguono la seguente regola:

- Sia $\mathcal{D}$ l'insieme delle divinità che proteggono T.
  - Se in $\mathcal{D}$ esiste una divinità D^M superiore alle altre rispetto all'ordinamento $<$ (ossia: per ogni divinità $D \in \mathcal{D}$, se $D \neq D^M$ allora $D < D^M$), allora T viene dedicato a D^M.
  - Se in in $\mathcal{D}$ non esiste la divinità massima, T non viene dedicato.

Esempio 3

Supponiamo che la gerarchia sia quella descritta all'inizio dell'Esempio 2 e che le associazioni siano:

Divinità	Blocchi
`giove`	b, e
`giunone`	d
`marte`	c, e
`mercurio`	a
`minerva`	e
`nettuno`	a, e
`saturno`	a
`urano`	
`venere`	b

Le divinità che proteggono il tempio T_1 (dell'Esempio 1) sono `giove`, `mercurio`, `nettuno`, `saturno` e `venere`. Essendo la divinità massima `giove`, il tempio T_1 viene dedicato a `giove` (stesso discorso per T_2).
Analogamente, il tempio T_3, protetto da `giove`, `marte`, `minerva` e `nettuno`, viene dedicato a `giove`.
Il tempio T_4 è invece protetto da `giove`, `giunone`, `marte`, `minerva` e `nettuno`. Poiché non esiste una divinità superiore alle altre, il tempio non viene dedicato.
Infine, i templi T_5 e T_6 sono dedicati a `giove`.

Si richiede di implementare una struttura dati efficiente che permetta di eseguire le operazioni seguenti.

- **blocco** (α, p, r)

  Definisce un nuovo blocco di nome α, peso p e resistenza r. Se esiste già un blocco di nome α non compie alcuna operazione.

- **tempio** $(\alpha, \beta_1, \ldots, \beta_n)$

  Se esiste già un tempio di nome α non compie alcuna operazione. Se per qualche β_i (con $1 \leq i \leq n$) non esiste alcun blocco di nome β_i non compie alcuna operazione. Altrimenti, per ogni $1 \leq i \leq n$,

sia b_i il blocco di nome β_i (assumiamo che i nomi siano tutti distinti fra loro). Viene costruito un tempio di altezza massima fra quelli costruibili con l'insieme dei blocchi $\{b_1, \ldots, b_n\}$ e viene stampata su una nuova linea il messaggio

α k

dove k è l'altezza del tempio.

- **elimina** (α)

  Distrugge, se esiste, il tempio di nome α, altrimenti non compie alcuna operazione.

- **divinità** (α, β)

  Se non esiste alcun blocco di nome β non compie alcuna operazione. Altrimenti, viene aggiunto β all'insieme dei blocchi B_D della divinità di nome α.

- **minore** (α, β)

  Stabilisce che la divinità di nome α è minore della divinità di nome β ($\alpha < \beta$), se questo non viola le condizioni (P1) e (P2) su $<$ precedenti. In caso contrario, viene stampato il messaggio

  `non e' possibile porre` $\alpha < \beta$

- **dedica** (α)

  Se non esiste alcun tempio di nome α oppure se il tempio di nome α è già stato dedicato a una divinità non compie alcuna operazione.
  Altrimenti, si dedica il tempio di nome α nel modo descritto. Se il tempio viene dedicato a una divinità di nome σ viene stampato il messaggio

  α `e' stato dedicato a` σ

  altrimenti viene stampato il messaggio

  `non e' possibile dedicare` α

- **stampa** (α)

  Se non esiste alcun tempio di nome α non compie alcuna operazione. Altrimenti stampa la descrizione del tempio di nome α nel modo descritto nell'apposita sezione.

Si noti che le operazioni richieste sono liberamente implementabili; in particolare, non vanno necessariamente intese come prototipi di funzioni.

Specifiche di implementazione

Il programma deve leggere dallo standard input (`stdin`) una sequenza di righe (separate da `\n`), ciascuna delle quali corrisponde a una riga della prima colonna della Tabella 1, dove $\alpha, \beta, \beta_1, \ldots, \beta_n$ sono stringhe finite di lunghezza arbitraria sull'alfabeto $\{a, b, \ldots, z\}$, n, p e r sono interi positivi. I vari elementi sulla riga sono separati da uno o più spazi. Quando una riga è letta, viene eseguita l'operazione associata; le operazioni di stampa sono effettuate sullo standard output (`stdout`), e ogni operazione deve iniziare su una nuova riga.

Riga di input	Operazione
`b` $\alpha\ p\ r$	**blocco** (α, p, r)
`t` $\alpha\ \beta_1\ \cdots\ \beta_n$	**tempio** $(\alpha, \beta_1, \ldots, \beta_n)$
`e` α	**elimina** (α)
`d` $\alpha\ \beta$	**divinità** (α, β)
`m` $\alpha\ \beta$	**minore** (α, β)
`D` α	**dedica** (α)
`s` α	**stampa** (α)
`f`	Termina l'esecuzione del programma

Tabella 1: Specifiche del programma

Note

1. Non devono essere presenti vincoli sul numero di elementi, divinità, templi, elementi associati a divinità, pesi, resistenze (se non quelli determinati dal tipo di dato intero). Non si richiede – anzi si sconsiglia – l'uso di grafica, se non per test personali: in modo particolare, non si usi `conio.h` e neppure `clrscr()`.

2. Per semplicità si suppone che l'input sia sempre conforme alle specifiche di Tabella 1, per cui non è necessario controllare la correttezza dell'input. Per leggere l'input si usino le funzioni standard ANSI C `getchar()` e/o `scanf()`.

3. **Formato per la visualizzazione di un tempio**

   Sia $T = (\alpha, (b_1, \ldots, b_k))$ il tempio da visualizzare, e sia β_i il nome del blocco b_i, per ogni $1 \leq i \leq k$. Se il tempio è dedicato a una divinità D, sia σ il nome di D, altrimenti σ è la stringa nulla. Allora l'output deve apparire nel seguente formato:

   $(\alpha\ \ k\ \ \sigma$
   β_k
   β_{k-1}
   $\vdots$
   β_2
   β_1
   $)$

   nell'ordine specificato.

Esempio

Si supponga che le righe di input siano:

```
b argento 4 5
b bronzo 7 4
b corallo 3 20
b diamante 50 15
b zinco 10 20
b pietra 3 4
b granito 5 3
b marmo 1 8
b faggio 20 8
b noce 7 1
b rovere 1 5
t atene argento noce
e atene
t atene argento bronzo corallo
t sparta zinco argento diamante
t corinto diamante zinco corallo bronzo
t delo diamante zinco argento corallo bronzo
s delo
t rodi marmo granito pietra
t tebe noce rovere faggio
m eos poseidon
m eos apollo
m cronos apollo
m cronos zeus
m gea zeus
m gea eos
m gea cronos
m gea urano
m selene cronos
m selene zeus
m apollo selene
d apollo argento
d apollo marmo
d apollo pietra
d cronos faggio
d eos zinco
d gea corallo
d gea noce
d gea granito
d selene rovere
d selene pietra
d poseidon bronzo
d urano noce
d zeus diamante
d zeus zinco
D atene
D sparta
D delo
D rodi
```

```
D tebe
s rodi
m apollo zeus
m cronos urano
D atene
D sparta
D delo
D tebe
f
```

L'output prodotto dal programma deve essere:

```
atene 1
atene 3
sparta 3
corinto 3
delo 4
(delo 4
argento
bronzo
corallo
diamante
)
rodi 3
tebe 3
non e' possibile porre apollo < selene
non e' possibile dedicare atene
non e' possibile dedicare sparta
non e' possibile dedicare delo
rodi e' stato dedicato a apollo
non e' possibile dedicare tebe
(rodi 3 apollo
pietra
granito
marmo
)
non e' possibile dedicare atene
sparta e' stato dedicato a zeus
non e' possibile dedicare delo
tebe e' stato dedicato a urano
```

Presentazione del progetto

Il progetto deve essere inviato per posta elettronica all'indirizzo `aguzzoli@dsi.unimi.it` entro l'11 febbraio 2007 (incluso).

Occorre presentare:

1. il codice sorgente (rigorosamente ANSI C, compilabile con **gcc**);
2. una sintetica relazione (formato pdf o rtf) che illustra le strutture dati utilizzate e analizza il costo delle diverse operazioni richieste dalla specifica.

I due o più file (file sorgenti C + relazione) devono essere contenuti in un unico file `.zip` il cui nome dovrà essere `cognome.zip`. La relazione e il codice devono riportare il vostro nome, cognome e matricola.

Una copia cartacea della relazione e del codice deve inoltre essere consegnata al dr. Aguzzoli entro il 12 febbraio 2007 (lasciandola eventualmente nella sua casella postale presso il dipartimento in via Comelico).

Si ricorda infine di presentarsi alla prova orale con una copia stampata della relazione e del codice.

La discussione del progetto e l'esame orale di Algoritmi e Strutture Dati si svolgeranno **indicativamente** nei giorni 15 febbraio, 20 febbraio, 23 febbraio.

Alla consegna del progetto, indicare nel *testo* della e-mail la data in cui si preferisce sostenere la prova orale; nei limiti del possibile si cercherà di tener conto di tali indicazioni (se non si hanno preferenze, non dare alcuna indicazione).

Il calendario degli esami orali sarà disponibile sulla pagina del corso qualche giorno dopo il termine di consegna del progetto.

Per ogni ulteriore chiarimento:
E-mail: `aguzzoli@dsi.unimi.it`
Ricevimento: il mercoledì, ore 15-16, stanza S204.

Avvisi

La versione aggiornata del progetto è pubblicata in .pdf sul sito:
`http://homes.dsi.unimi.it/~aguzzoli/algo.htm`.
Si consiglia di consultare periodicamente questo sito per eventuali correzioni e/o precisazioni relative al testo del progetto.

Si richiede allo studente di effettuare un adeguato collaudo del proprio progetto su numerosi esempi diversi per verificarne la correttezza e valutarne le prestazioni.

La realizzazione del progetto è una prova d'esame da svolgersi **individualmente**. I progetti giudicati frutto di **collaborazioni** saranno **estromessi** d'ufficio dalla valutazione.