Algoritmi e strutture dati

Docente: Violetta Lonati

Prova di laboratorio - appello del 31 gennaio 2023

Note importanti

- Si leggano attentamente i testi degli esercizi e le indicazioni su come svolgerli. Se ci sono dubbi sul significato delle richieste, è opportuno chiedere chiarimenti!
- Attenzione agli esercizi in cui si chiede di rispondere a delle domande: anche queste sono parte integrale dello svolgimento della prova e sono soggette a valutazione! Si raccomanda di rileggere con cura quanto scritto, prima di consegnare.
- Dopo essersi autenticati, si carichino sul sito `upload.di.unimi.it` i file contenenti gli svolgimenti degli esercizi.
- Si leggano attentamente anche le indicazioni su come preparare i file da consegnare. Per ogni esercizio è richiesto di preparare un programma Go oppure un file di testo con estensione `.txt`. Alla fine del testo di ogni esercizio viene indicato il nome con cui salvare il file; è importante rispettare questa indicazione.
- Nella prima riga di tutti i file consegnati è necessario **scrivere nome, cognome e matricola**.
- Per collaudare i vostri programmi potete usare i file di test allegati. Test analoghi saranno usati nella correzione automatica degli svolgimenti. Per eseguire i test è necessario:
  - entrare della directory dell'esercizio
  - salvare nella directory il file col <nome> indicato
  - creare l'eseguibile (go build <nome>.go)
  - lanciare il comando go mod init <nome>
  - lanciare il comando go test -v (e osservare l'output)

I nomi dei file consegnati devono essere i seguenti:
`es1-foresta.go`
`es2-interruttori.txt`
`es3-interruttori.go`

1 Foresta

Federico, appassionato di enigmi, ha un negozio di oggettistica e arredamento di seconda mano, e ha un modo tutto particolare di indicare i prezzi degli oggetti in vendita. Ad ogni oggetto, associa un indizio che serve per stabilire il prezzo dell'oggetto.

L'indizio associato ad un oggetto può essere di due tipi:

- un numero,
- oppure una scritta di questo tipo: "*a* OP *b*", dove *a* e *b* sono nomi di altri oggetti e OP è il simbolo di una operazione aritmetica (`+`, `-`, `*`, `/`).

Usando gli indizi, si possono calcolare i prezzi degli oggetti:

- se l'indizio associato all'oggetto *x* è un numero, allora quello è il prezzo dell'oggetto *x*;
- se l'indizio associato all'oggetto *x* è dato da un'operazione "*a* OP *b*", allora il prezzo di *x* si ottiene combinando con l'operazione OP i prezzi dei due oggetti *a* e *b*.

Ogni oggetto compare al massimo in un indizio associato ad altri oggetti.

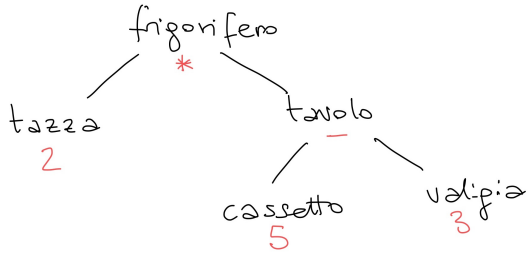
Esempio. Ecco gli indizi che ha predisposto Federico:

- al tavolo ha associato l'indizio `cassetto - valigia`
- al frigorifero ha associato l'indizio `tazza * tavolo`
- alla tazza ha associato l'indizio `2`
- alla valigia ha associato l'indizio `3`
- al cassetto ha associato l'indizio `5`,

Calcoliamo il prezzo del frigorifero, cui è associato l'indizio `tazza * tavolo`. Alla tazza e al tavolo sono associati rispettivamente gli indizi `2` e `cassetto - valigia`. Al cassetto e alla valigia sono associati rispettivamente gli indizi `5` e `3`. La differenza tra 5 e 3 è 2, e questo è il prezzo del tavolo. Dunque il prezzo frigorifero è 2 * 2 = 4.

Modellazione. Modelliamo il problema usando una foresta di alberi binari. Ogni oggetto è rappresentato da un nodo; ogni nodo ha per chiave una stringa: il nome dell'oggetto che il nodo rappresenta. Gli oggetti che hanno per indizio un numero saranno le foglie; gli oggetti che non compaiono in nessun indizio sono radici; gli oggetti che hanno per indizio un'operazione saranno nodi interni. Se un nodo rappresenta un oggetto che ha per indizio l'operazione "*a* OP *b*", allora il figlio sinistro del nodo ha per chiave *a*, il figlio destro ha per chiave *b*.

Es: gli indizi elencati nell'elenco precedente possono essere modellati con una foresta di un solo albero, come nel disegno qui sotto:



Completate il programma `es1-foresta.go`

1. Scrivete una funzione con segnatura
   `leggiInput() map[string]*oggetto`
   che legga da standard input gli indizi (uno per riga, nel formato "oggetto: indizio"), crei un `oggetto` per ogni indizio e li memorizzi in una mappa.

2. Definite opportunamente un tipo `foresta` (potete definire tipi ausiliari, se lo ritenete opportuno) e scrivete una funzione con segnatura
   `costruisciForesta(mappa map[string]*oggetto) foresta`
   che costruisca una foresta a partire dalla mappa degli indizi.

3. Scrivete una funzione con segnatura
   `up(f foresta, n string) (string, bool)`
   che stampi la chiave del padre del nodo di chiave `n`. Scrivete funzioni simili di nome `dx` e `sx` che restituiscano, rispettivamente, la chiave del figlio destro e sinistro del nodo di chiave `n`. Se tali figli/padre non esistono, la funzione deve restituire `false` come secondo argomento.

4. Scrivete una funzione con segnatura
   `stampaAlbero(f foresta, n string)`
   che stampi, in ordine simmetrico, i nodi dell'albero della foresta che ha per radice il nodo di chiave `n`. Nel caso delle foglie, si deve stampare tra parentesi anche il numero associato. Ad esempio, nel caso dell'albero disegnato sopra, la funzione deve stampare:

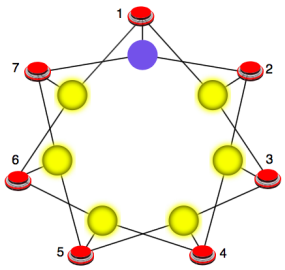
   ```
   tazza (val = 2)
   frigorifero
   cassetto (val = 5)
   tavolo
   valigia (val = 3)
   ```

5. Scrivete una funzione con segnatura
   `calcolaPrezzo(f foresta, n string) int`
   che calcoli il prezzo dell'oggetto di nome `n`.

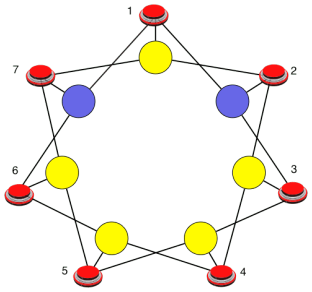
Note per la consegna. Salvate il vostro programma con nome `es1-foresta.go`. Per testare il vostro programma potete usare il file `es1-foresta_test.go`.

2 Interruttori - modellazione

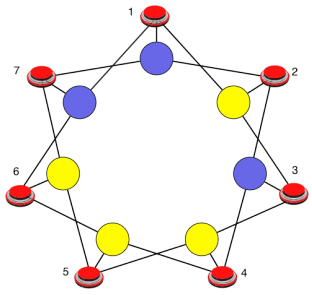
Nella rete di luci rappresentata in figura, ogni interruttore agisce sulle tre luci che collega (le luci sono disegnate in giallo o blu a seconda che siano accese o spente, rispettivamente, mentre gli interruttori sono in rosso).

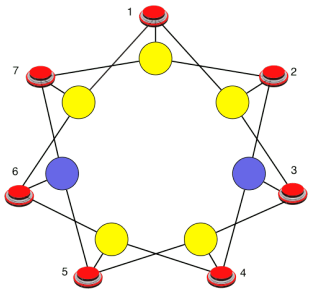


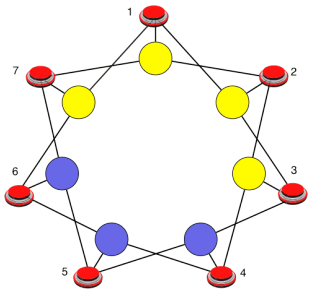
Ad esempio, se si parte con una sola luce spenta come nella figura sopra e si preme l'interruttore 1, si ottiene



Se poi si premono nell'ordine gli interruttori 2, 7 e 4, la situazione evolve come raffigurato qui sotto:







Premendo infine l'interruttore 5, tutte le luci risulteranno accese.

Generalizziamo la situazione considerando un numero arbitrario n di interruttori e di luci (nell'esempio sopra n è pari a 7). Indichiamo sia le luci che gli interruttori con numeri progressivi da 1 a n. Se i è compreso tra 2 e $n-1$, allora l'interruttore i è collegato con le luci $i-1$, i e $i+1$. Inoltre, l'interruttore 1 è collegato con le luci n, 1 e 2, mentre l'interruttore n è collegato con le luci $n-1$, n e 1. L'intero n è chiamato *dimensione della rete*.

Lo *stato di accensione* della rete è descritto da una sequenza di bit: il bit di posizione i indica se la luce i è accesa o spenta. Ad esempio, lo stato di accensione nell'ultima figura è `1110001`.

Vogliamo scrivere un programma che simuli l'uso di interruttori in questo tipo di reti, rappresentando in ogni istante lo stato di accensione della rete mediante una slice di booleani. Le operazioni da simulare sono le seguenti:

Operazioni:

(a) Dato un intero n, premere l'interruttore n-esimo.
(b) Data una sequenza di interruttori $i_1 i_2 \dots i_m$, premere la sequenza di interruttori (nell'ordine dato).
(c) Data una rete di luci in un certo stato di accensione iniziale, determinare una sequenza di interruttori che consenta di spegnere tutte le luci.
(d) Data una rete di luci in un certo stato di accensione iniziale, determinare quanti interruttori si devono premere, come minimo, per poter spegnere tutte le luci.

Esempio: Consideriamo la rete descritta precedentemente (vedi figure). Partendo dalla configurazione `1000000`, premendo gli interruttori 1, 2, 7, 4, 5 si spengono tutte le luci; ecco come evolve lo stato di accensione della rete:

```
0100001
1010001
0010010
0001110
0000000
```

Svolgete i seguenti punti, assumendo che il numero degli interruttori della rete sia molto grande:

1) *Modellate la situazione* con una struttura di dati opportuna, che consenta di svolgere efficientemente le operazioni enunciate sopra:
   - *descrivete* come si può rappresentare la rete mediante questa struttura di dati;
   - *riformulate*, nei termini della struttura di dati scelta, ciascuno delle operazioni enunciati sopra.

   Nota bene: non ci interessa modellare la topologia della rete, ma l'evolvere dello stato di accensione della rete!!!

2) Descrivete come è opportuno *implementare la struttura dati scelta*, specificando quali informazioni vanno tenute in memoria e come vanno rappresentate.

3) Per ciascun compito, *progettate e descrivete un algoritmo* che consente di svolgere il compito, sfruttando le scelte di progettazione e implementazione fatte precedentemente. Gli algoritmi possono essere descritti a parole o in pseudocodice; può essere opportuno fare riferimento ad algoritmi noti.

4) Discutete la complessità dei vostri algoritmi.

Note per la consegna. Scrivete le vostre risposte in un file con nome `es2-interruttori.txt`.

3 Interruttori - implementazione

Definite il tipo `conf` come slice di booleani. Scrivete quindi una funzione Go per ciascuna delle operazioni enunciate sopra. Le funzioni devono avere queste segnature:
(a) `premi(curr conf, x int) conf`
(b) `sequenza(curr conf, x []int) conf`
(c) `sequenzaSpegniTutto(curr conf) []int`
(d) `spegniTutto(curr conf) int`
Potete definire altri tipi e scrivere funzioni ausiliarie se lo ritenete opportuno.

Scrivete infine una funzione `main` che legge da standard input una sequenza di istruzioni, e simula l'operazione corrispondente. Le istruzioni in input sono scritte scritte una per riga, secondo il formato nella seguente tabella, dove n è un intero positivi, $\ell_1 \dots \ell_n$ sono bit, $i, i_1, \dots i_m$ sono numeri compresi tra 1 e n.

Istruzione	**Operazione**
`+` n	Crea una rete con n interruttori e n luci tutte spente. Se esiste già una rete, la elimina e ne costruisce una nuova.
`o` $\ell_1\ell_2\dots\ell_n$	Imposta la rete allo stato di accensione $\ell_1\ell_2\dots\ell_n$.
`p`	Stampa lo stato di accensione della rete.
`s` i	Preme l'interruttore i.
`S` $i_1 i_2 \dots i_m$	Preme in sequenza gli interruttori $i_1\ i_2\ \dots i_m$.
`x`	Stampa il minimo numero di interruttori che si devono premere per spegnere tutte le luci
`f`	Termina l'esecuzione

Esempio di esecuzione

Simuliamo il caso descritto precedentemente (vedi figure): ricevendo da standard input

```
f
+ 7
p
o 0111111
p
s 1
```

```
p
S 2 7 4
p
s 5
p
f
```

il programma deve stampare

```
0000000
0111111
1011110
1110001
1111111
```

Note per la consegna. Scrivete le funzioni richieste in un file con nome `es3-interruttori.go`. Per testare il vostro programma potete usare il file `es3-interruttori_test.go`.