Soluzione Progetto 1 ASD a.a. 2017/2018

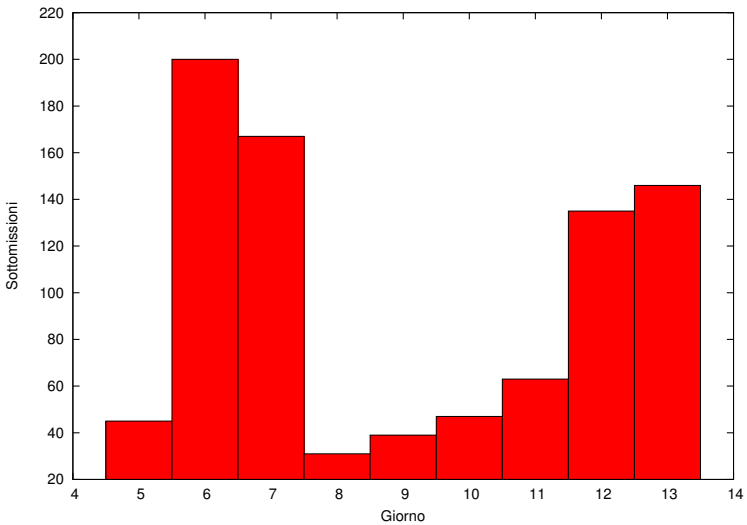
Alessio Guerrieri e Cristian Consonni

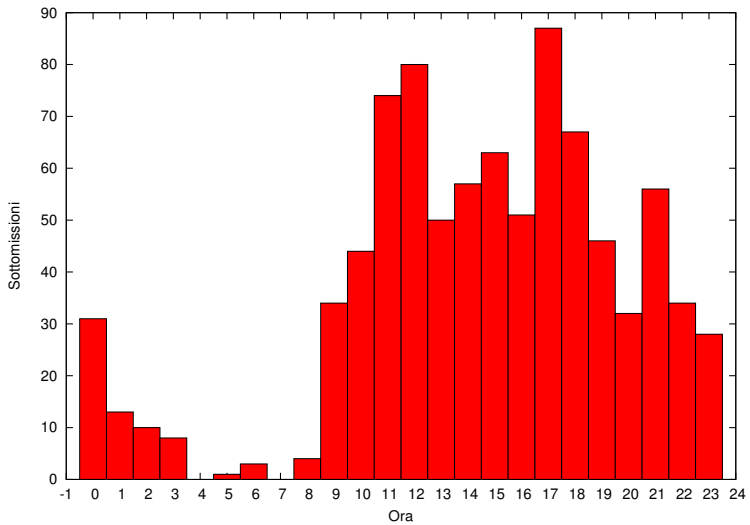
14 dicembre 2017

Statistiche

Statistiche

Numero sottoposizioni: 873





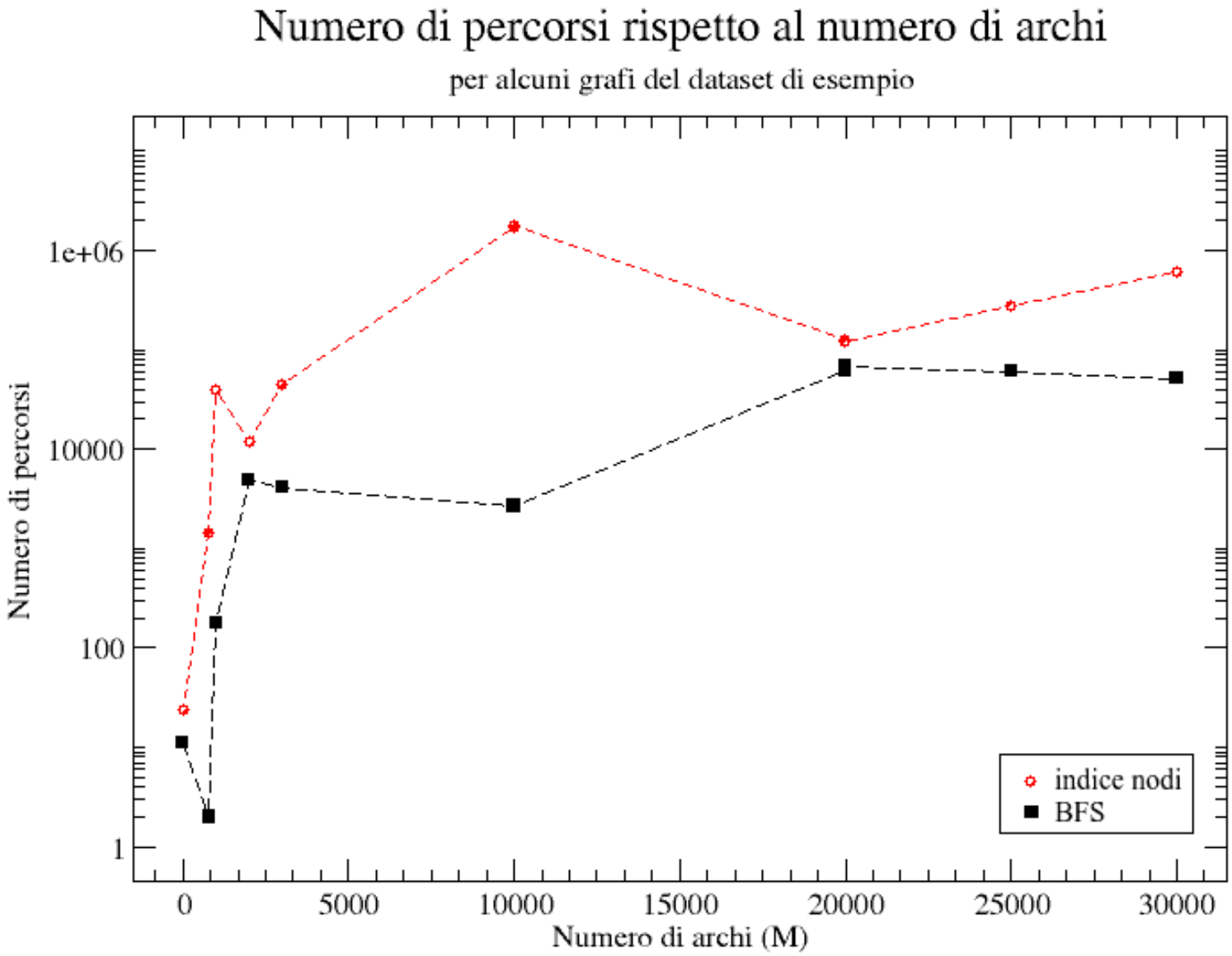
- 81 gruppi iscritti (l'ultimo alle **22:26 del 04/12**);
- 188 studenti;
- $\sim$ 13 ore di ricevimento (compresi i laboratori);
- 25 mail ricevute;

Risultati

Punteggi (classifica completa sul sito)

- $P < 30$ $\longrightarrow$ progetto non passato
- $30 \leq P < 40$ $\longrightarrow$ 1 punto bonus (6 gruppi)
- $40 \leq P < 100$ $\longrightarrow$ 2 punti bonus (8 gruppi)
- $P = 100$ $\longrightarrow$ 3 punti bonus (54 gruppi)

Anti-soluzione: enumerare i percorsi



Soluzione dei casi non pesati e non intermittenti ($w_i = 1$, $f_i = 0$, $y_i = T_{\max}$), solo T (o_1)

- trovare un cammino minimo da 0 a $N - 1$
- inizialmente $\text{dist}(0) = 0$ e $\forall n \neq 0\ \text{dist}(n) = T_{\max}$
- facciamo una visita in ampiezza partendo dal nodo 0
- se dal nodo u visitiamo il nodo v:

$$\text{dist}(v) = \text{dist}(u) + 1$$

- se $\text{dist}(N - 1) = T_{\max}$, allora $N - 1$ non è raggiungibile

$\Rightarrow$ soluzione: `bfs.cpp` (62 SLOC) (*Source Lines Of Code*)

$\Rightarrow$ complessità: $O(V + E)$

$\Rightarrow$ 20 punti (5 soluzioni parziali + 1 ottima)

Soluzione dei casi non pesati e non intermittenti ($w_i = 1$, $f_i = 0$, $y_i = T_{\max}$), T e percorso ($o_1 + o_2$)

- ogni nodo nodo ha un predecessore
- se dal nodo u visitiamo il nodo v:

$$v.\text{pred} = u$$

- stampiamo i predecessori di ogni nodo, partendo da $N - 1$

$\Rightarrow$ soluzione: `bfs_path.cpp` (86 SLOC)

$\Rightarrow$ complessità: $O(V + E)$

$\Rightarrow$ 30 punti (6 soluzioni ottime)

Soluzione dei casi non intermittenti ($f_i = 0$, $y_i = T_{\max}$), solo T (o_1)

- trovare un cammino minimo da 0 a $N-1$ su un grafo pesato con pesi **positivi**
- algoritmo di *Dijkstra* o *Bellman-Ford*
- se dal nodo u visitiamo il nodo v:

$$\text{dist}(v) = \text{dist}(u) + \text{weight}(u, v)$$

$\Rightarrow$ soluzione: `dijkstra.cpp` (77 SLOC) o `bf.cpp` (69 SLOC)

$\Rightarrow$ complessità:

- $O((V+E)\log V)$ (Dijkstra)
- $O(V \cdot E)$ (Bellman-Ford)

$\Rightarrow$ 32 punti (9 soluzioni parziali + 1 ottima)

Soluzione del caso generico: intuizione

- se $y_i < w_i$: non si può mai passare, si elimina l'arco
- la distanza di un nodo da 0 corrisponde al momento in cui si arriva in quel nodo $t(u) = \text{dist}(u)$ (una volta giunti in un nodo non conviene mai tornare indietro)

$\Rightarrow$ dobbiamo calcolare se si più partire subito a visitare il nodo successivo o se bisogna aspettare, calcoliamo l'istante di **partenza**:

- se $t(u) < f_{(u,v)} \rightarrow \text{partenza}(u) = f_{(u,v)}$
  (si aspetta il primo spegnimento)
- se $t(u) > f_{(u,v)}$:

$$t^r_{(u,v)} = (t(u) - f_{(u,v)}) \mod \underbrace{(y_{(u,v)} + n_{(u,v)})}_{\text{durata ciclo laser}}$$

(tempo relativo rispetto al ciclo di spegnimento e accensione del laser nel tunnel (u, v))

Soluzione del caso generico: funzione partenza

```
1  int partenza(int t){
2    if (t < inizio)
3      return inizio;
4    int ciclo = vivo + morto;
5    int pos = (t - inizio) % ciclo;
6
7    if (vivo - pos >= peso){
8      // riesco a passare subito
9      return t;
10   }
11   else{
12     // aspetto la fine del ciclo (nuovo
           spegnimento)
13     return t - pos + ciclo;
14   }
15 }
```

Soluzione del caso generico, T e percorso ($o_1 + o_2$)

- algoritmo di *Dijkstra*
- se dal nodo u visitiamo il nodo v:

$$\text{dist}(v) = \text{partenza}(u) + \text{weight}(u, v)$$

$\Rightarrow$ soluzione: `laser.cpp` (100 SLOC)

$\Rightarrow$ complessità: $O((V + E) \log V)$

$\Rightarrow$ 100 punti

Note finali

- Classifiche e sorgenti sul sito (controllate i numeri di matricola):
  - `http://judge.science.unitn.it/slides/asd17/classifica_prog1.pdf`
  - Assumiamo gli stessi gruppi, in caso di cambiamenti scrivetemi (`cristian.consonni@unitn.it` e `a.guerrieri@unitn.it`)

Approfondimento: code con priorità

Struttura dati	INSERT	DEL-MIN	MIN	DELETE
array	1	n	n	1
binary heap	$\log(n)$	$\log(n)$	1	$\log(n)$
d-way heap	$\log_d(n)$	$d \cdot \log_d(n)$	1	$d \cdot \log_d(n)$
binomial heap	1	$\log(n)$	1	$\log(n)$
Fibonacci heap	1	$\log(n)^{\dagger}$	1	$\log(n)^{\dagger}$

$^{(\dagger)}$ costo ammortizzato

Table: Analisi di complessità per varie implementazioni di una coda con priorità.

$\Rightarrow$ implementazioni in Java: algs4 @ Princeton (link)

Credits

- Il conteggio delle linee si codice è stato realizzato usando il programma `SLOCCount` di A. Wheeler