OIS 2018/2019

Soluzione ai problemi

Aldini Valeriani Sirani, 8 febbraio 2019

Staff OIS

Statistiche

- Sottoposizioni: ~1300
- 90 in C, 1200 in C++, 4 in Pascal
- 1785778 bytes totali
- Parentesi tonde aperte: 22770, chiuse: 22771
- Parentesi graffe aperte: 10857, chiuse: 10838
- Parole chiave:
  - int: 11526
  - float: 2
  - short: 149
  - if: 5655
  - bug: 5
  - todo: 11
  - ois: 0

Primi punti: Peaky Blinders (16min, 60/100)
Prima soluzione: hackerinoTopolino (20min, 100/100)

Long Columns (columns2)

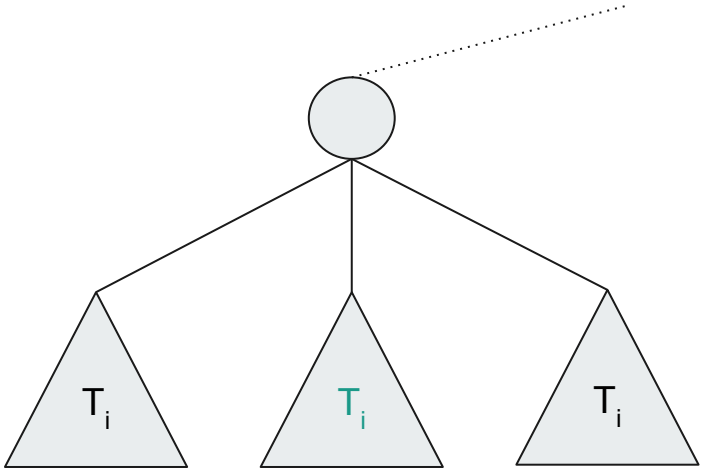
Osservazione chiave
L'ordine con cui vengono effettuate le operazioni di taglio sulle colonne è irrilevante.

Per ogni colonna lunga L, *senza guardare le altre*, decido quante colonne lunghe K posso ricavare:

- se è lunga esattamente 11K, la posso tagliare dieci volte (e ottengo 10+1 colonne lunghe K)
- se è più lunga di 10K, la posso tagliare dieci volte (ottengo 10 colonne lunghe K e una inutile da L-10K)
- se è più corta di 10K, ottengo L/K colonne lunghe K e una inutile da L%K.

Primi punti: Zente Mala (3min 49sec, 100/100)
Prima soluzione: Zente Mala (3min 49sec, 100/100)

Chief Executive Officer (ceo)



$$T_{i+1} = T_i * A_i + 1$$

Primi punti: I Discepoli di P.P. (1h07, 30/100)
Prima soluzione: hackerinoTopolino (1h21, 100/100)

Code Refactoring (refactor)

Non c'è alcuna idea profonda, ma è importante implementare bene il riconoscimento degli identificatori.

Allo scopo si può definire uno *stato* del programma, a seconda del quale facciamo cose diverse.

Possibili stati sono:

- stato "normale"
- all'interno di commento multiriga /* */
- all'interno di un commento //
- all'interno di una stringa (*string literal*)

Solo nello stato "normale" si cerca l'inizio del prossimo identificatore e si verifica se corrisponde **interamente** alla stringa cercata.

Primi punti: bocciodromi (8min, 40/100)
Prima soluzione: Sample_text (12min, 100/100)

Gasoline Stations (gasoline)

Osservazione chiave

Dopo aver passato un distributore con costo X non è sensato fermarsi ad uno con costo Y>X.

Quindi, ad ogni momento va acquistato il carburante che mi serve dalla stazione più economica.

Primi punti: C++opernico (16min, 55/100)
Prima soluzione: Sushi Squad (27min, 100/100)

Air Traffic Control II (atc2)

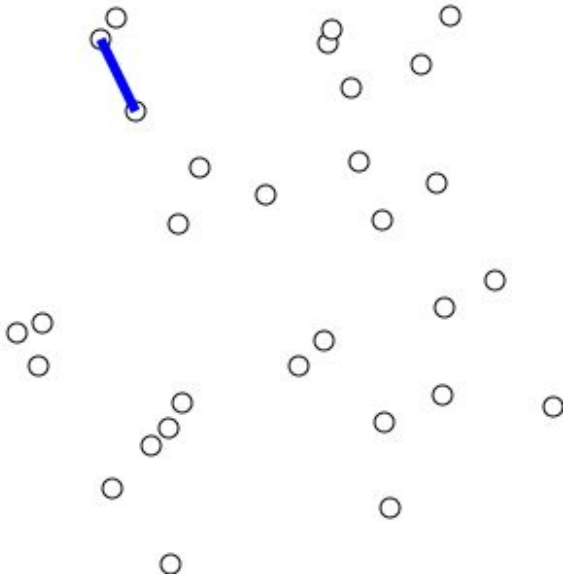
Astrazione del problema: grafo *completo* dove i nodi sono le torri.

Basta trovare un insieme di archi di minimo costo che renda raggiungibile ogni nodo:
problema noto in letteratura come **Minimum Spanning Tree**.

Limite di memoria 8MB:

- Non serve memorizzare tutta la mappa in input ma solo la posizione delle torri
- Algoritmo di **Prim** (per risolvere MST), implementazione lineare in memoria:
  per ogni vertice aggiorno la distanza verso tutti gli altri (se mi conviene arrivarci da questo vertice)
  e alla fine scelgo il più vicino, espandendo l'insieme degli archi che costituiscono l'MST.

Air Traffic Control II (atc2)



Primi punti: ARACHIDI VELOCI (14min, 30/100)
Soluzione migliore: C++opernico (many, 98.96/100)

Full-Body Workout (workout)

Ricerca binaria sul risultato verificando con una greedy se è possibile ottenere *al più* quel valore.

Provo “per ogni” valore del risultato:

- Se è possibile aggiungere un esercizio faticoso senza sforare, prendo il più pesante,
- Altrimenti, se posso, aggiungo l’esercizio più riposante che non mi faccia andare sotto zero,
- Altrimenti metto l’esercizio riposante che mi manda il meno negativo possibile,
- Altrimenti non riesco ad ottenere quel valore.

Questa soluzione non è ottima, per ottenere il punteggio massimo è necessario verificare con una ricerca completa *almeno* i casi piccoli.

Primi punti: hackerinoTopolino (44min, 10/100)
Prima soluzione: LSFoligno2 (2h31min, 70/100)

Not Another Pet Fair (exhibitions2)

Prima osservazione
Non ha senso visitare un'esibizione più di una volta (non aggiunge né toglie niente alla somma).

Seconda osservazione
Se sono una guida *pro-gatti*, quando mi si presenta un gruppo ho due scelte:

- Mandarlo a casa
- Dirigerlo verso un'altra esibizione (ovviamente *non ancora visitata*).
  Poiché la guida *pro-gatti* vuole massimizzare la somma S, la mossa sicuramente migliore è mandare il gruppo verso l'esibizione non visitata di valore positivo massimo.

Simmetricamente una guida *pro-cani* nella strategia ottima o manda il gruppo a casa oppure lo manda verso l'esibizione non visitata di valore il più possibile negativo.

Not Another Pet Fair (exhibitions2)

Implementazione

Per ciascuno degli M gruppi, simulo dove viene mandato mantenendo **un'insieme ordinato** dei valori delle esibizioni non ancora viste. Ad ogni esibizione vedo se è controllata da una guida a favore dei cani o dei gatti, provando le due strategie descritte nella slide precedente (mando a casa oppure mando verso max/min). Questo procedimento si ripete al più N volte, quindi in tutto O(NM).

Da O(NM) a O(N^2+M)

Precalcolando la risposta per tutte le N possibili posizioni di inizio dei gruppi, la soluzione precedente si porta facilmente a complessità O(N^2+M) per risolvere anche il penultimo subtask.

Hint per la soluzione O(N)

Per risolvere in tempo lineare bisogna riuscire a spezzare la soluzione per un punto di partenza E_i nelle parti prima e dopo aver incontrato l'esibizione simmetrica nel vettore ordinato, e calcolarne dei valori opportuni di modo che possano essere combinati tra loro e con il valore $A[E_i]$ per ottenere la soluzione.

Primi punti: C++opernico (46min, 0.45/100)
Soluzione migliore: Sushi Squad (57min, 86.43/100)

Nation Infrastructures (streets)

Soluzione **Branch-and-Bound**, partendo dalla soluzione esponenziale banale cercare di evitare di percorrere branch inutili:

- Un nodo con grado zero non può avere archi: svuoto la sua lista di adiacenza;
- Un nodo con grado uguale al numero di archi disponibili: prendo tutti gli archi;
- Altrimenti provo *tutte* le combinazioni

Nation Infrastructures (streets)

Alcune considerazioni per velocizzare la BnB:

- Una volta raggiunta la soluzione massima possibile (prendo tutti gli archi in input) non ha senso fare altro;
- Se non ho scelte obbligate cerco di arrivare più velocemente possibile alla soluzione ottima: approccio euristico
    - Cerco l'arco `(i, j)` che minimizza la quantità `D[i] - adj[i].size() + D[j] - adj[j].size()`