Soluzione Progetto 1 ASD a.a. 2018/2019

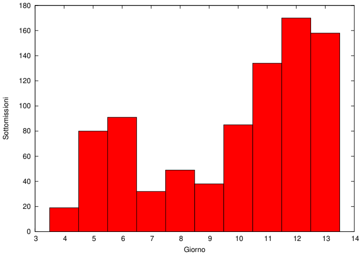
Montresor Va Alla Guerra

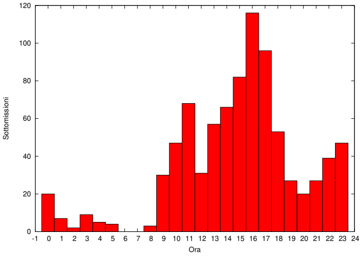
Cristian Consonni e Lorenzo Ghiro

18 dicembre 2018

Statistiche

Numero sottoposizioni: 856





- 87 gruppi partecipanti;
- 185 studenti;
- 12 ore di ricevimento (compresi i laboratori);
- 62 mail ricevute;

Risultati

Punteggi (classifica completa sul sito)

- $P < 30$ $\longrightarrow$ progetto non passato
- $30 \leq P < 40$ $\longrightarrow$ 2 punti bonus (17 gruppi)
- $40 \leq P < 80$ $\longrightarrow$ 3 punti bonus (35 gruppi)
- $P \geq 80$ $\longrightarrow$ 4 punti bonus (30 gruppi)

`https://judge.science.unitn.it/slides/asd18/classifica_prog1.pdf`

Considerazioni iniziali

- Nel caso generale, quando in campo ci sono più soldati e componenti, il problema assegnato si riconduce a quello noto in letteratura come *"Assegnamento massimale in grafo bipartito di costo minimo"*;
- L'algoritmo risolutivo è noto come **algoritmo ungherese**;
- Esistono algoritmi ad-hoc più semplici per il caso particolare in cui in campo si trovi un solo soldato;
- Risolvere correttamente questo sottocaso più semplice è stato considerato come requisito per raggiungere la sufficienza;
- Forniamo ora una traccia delle soluzioni per:
  1. Il caso particolare con 1 solo soldato;
  2. Il caso generale, da risolvere modellando il problema in forma di grafo bipartito per poi individuare un assegnamento impiegando l'algoritmo ungherese.

Approccio Forza Bruta

- Una rappresentazione (corretta!) del problema ha forma matriciale;
- Sia M la matrice, si definisce un elemento della matrice $M[s][c]$ come il costo associato al trasposto della componente c se affidata al soldato s.
- Si tenta quindi di assegnare tutte le C componenti una e una sola volta agli S soldati disponibili. Tra tutti i modi di creare questo assegnamento si cerca quello a costo minimo.
- Ma quanti sono gli assegnamenti possibili?
- Per la prima componente posso scegliere tra S soldati. Mi restano $S-1$ soldati che posso assegnare alla seconda, $S-2$ per la terza...
- La complessità di questo problema combinatorio è quindi $S \times (S-1) \times \ldots \times (S-C) \Rightarrow \mathcal{O}(S!)$, davvero impensabile!

Sottocaso semplice: 1 soldato

Per fortuna non è necessario esplorare tutti i possibili assegnamenti, soprattutto se c'è un solo soldato. La vita si semplifica grazie a questa osservazione:

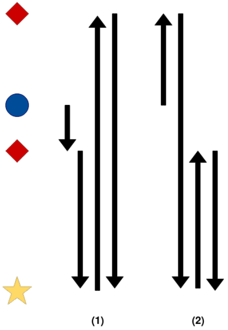
Osservazione

Un soldato da solo in campo ha un unico problema: scegliere la componente che **conviene** riportare al target **per prima**.

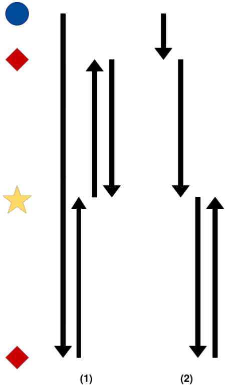
- Si noti infatti che, una volta riportata questa componente, il costo per riportare le altre componenti è fisso, indipendentemente dall'ordine con cui vengono recuperate;
- Il soldato quindi non deve fare più nessun altro ragionamento, deve soltanto fare bene la prima scelta;
- Ma qual è il *criterio di convenienza* corretto che deve stabilire il soldato?

Criterio di convenienza (I)

- componente più vicina?



- componente più lontana?



Criterio di convenienza (II)

Per ogni componente, il soldato deve confrontare 2 possibili strade da percorrere, coprendo due distanze diverse:

- d_1: il costo per il recupero a partire dal nascondiglio del soldato, pari alla somma di due distanze: la distanza Soldato-Componente + quella tra Componente e Target;
- d_2: il costo per il recupero della componente a partire dal target, pari a 2 volte la strada dal target alla componente;

Criterio di convenienza (III)

Convenienza

Conviene prendere per prima la componente che, se presa partendo dal nascondiglio piuttosto che dal target, fa risparmiare più strada (rispetto alle altre).

- Formalmente, quella per cui la differenza $d_1 - d_2$ è negativa, ($\Rightarrow d_2 > d_1$ cioè costa più il recupero dal target che dal nascondiglio), e in particolare la differenza è, in valore assoluto, la più grande (tra tutte le componenti è quella che in assoluto non conviene prendere a partire dal target).
- Bisogna calcolare per tutte le componenti i costi d_1 e d_2, quindi calcolare la differenza $d_1 - d_2$ e ricordarsi la componente per cui è vero che $d_1 - d_2 < 0 \wedge max(|d_1 - d_2|)$.
- Non c'è bisogno di mantenere i costi d_1 e d_2 in una matrice, possono essere calcolati on the fly a partire dall'input.

Complessità

- I costi e le differenze si calcolano in tempo costante, la componente più conveniente si individua iterando una volta su tutte le componenti;
- La complessità è lineare nel numero di componenti: $\mathcal{O}(C)$
- Il calcolo del tempo minimo per la consegna di tutte le componenti non è ancora terminato; Al costo d_1 individuato per la componente che conviene prendere per prima bisogna ricordarsi di sommare i costi d_2 associati a tutte le altre componenti. Questo calcolo si fa in tempo costante;

⇒ soluzione: `unsoldato.cpp` (74 SLOC)

⇒ complessità: $\mathcal{O}(C)$

⇒ 35 punti (6+1 caso degenere)

Modello e algoritmo efficiente (I)

- Visto che ogni soldato può riportare una sola componente alla volta, possiamo ricondurre il nostro problema all'*Assegnamento massimale in grafo bipartito di costo minimo**;
- Introducendo $C - 1$ soldati virtuali che partono dal target, modelliamo la possiblità di avere soldati che recuperano più di una componente;
- I pesi degli archi tra soldati veri e componenti corrispondono ai costi d_1, mentre gli archi uscenti dai soldati virtuali hanno il corrispondente peso d_2;

*maximum bipartite matching problem of minimum weight

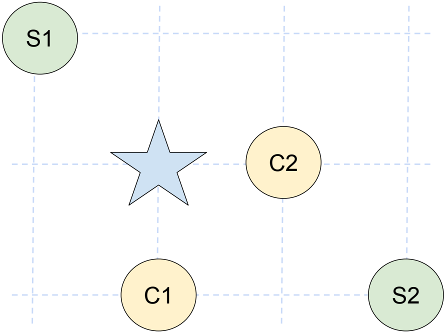
(a.k.a assignment problem)

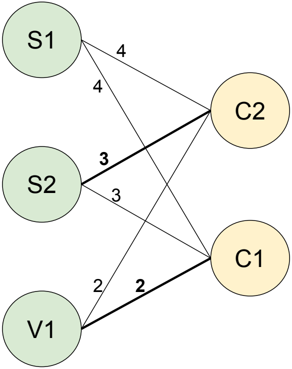
Modello e algoritmo efficiente (II)

- Sul grafo bipartito così costruito si individua un assegnamento (matching) esclusivo per ogni componente, minimizzando il costo dell'assegnamento;
- Tale assegnamento massimale e a costo minimo si individua in tempo $\mathcal{O}(C^3)$ utilizzando l'algoritmo di Khun-Munkres, conosciuto anche come algoritmo ungherese.

Esempio

A uno dei 2 soldati sarà affidato il recupero di entrambe le componenti.





	C1	C2
S1	4	4
S2	3	**3**
V1	**2**	2

Complessità

- formulazione matriciale (approfondimento)
  - $\Rightarrow$ soluzione: `hungarian_algorithm_matrix_On4.cpp` (219 SLOC)
  - $\Rightarrow$ complessità: $\mathcal{O}(C^4)$
  - $\Rightarrow$ 85 punti
- formulazione su grafo con augmenting paths (approfondimento)
  - $\Rightarrow$ soluzione: `hungarian_algorithm_graph_On3.cpp` (151 SLOC)
  - $\Rightarrow$ complessità: $\mathcal{O}(C^3)$
  - $\Rightarrow$ 100 punti

Note finali

- Classifiche e sorgenti sul sito (controllate i numeri di matricola):
  - `http://judge.science.unitn.it/slides/asd18/classifica_prog1.pdf`
  - Assumiamo gli stessi gruppi, in caso di cambiamenti scrivete a `cristian.consonni@unitn.it`

Credits

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