Partizionamento di intervalli

In questo caso disponiamo di più risorse identiche tra di loro e vogliamo che vengano svolte tutte le attività in modo tale da usare il minor numero di risorse e tenendo conto del fatto che due attività non possono usufruire della stessa risorsa allo stesso tempo.

- Durante il suo svolgimento, ciascuna attività necessita di un'unica risorsa.
- Una risorsa può essere allocata ad al più un'attività alla volta

Nell'interval scheduling avevamo un'unica risorsa e volevamo che venissero svolte il massimo numero di attività

Un'istanza del problema (Input) consiste in un insieme di n intervalli $[s_1,f_1],\ldots,[s_n,f_n]$ che rappresentano gli intervalli durante i quali si svolgono le n attività.
Obiettivo: far svolgere le n attività utilizzando il minor numero possible di risorse e in modo che ciascuna risorsa utilizzata venga allocata ad al più un'attività alla volta.

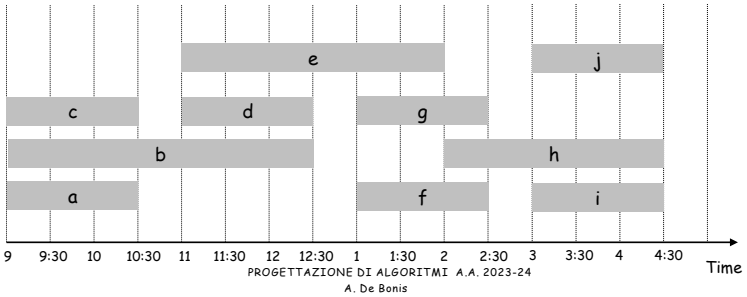
61

Partizionamento di intervalli

Esempio. Attività = lezioni da svolgere; Risorse= aule

- La lezione j comincia ad s_j e finisce ad f_j.
- Obiettivo: trovare il minor numero di aule che permetta di far svolgere tutte le lezioni in modo che non ci siano due lezioni che si svolgono contemporaneamente nella stessa aula.

Esempio: Questo schedule usa 4 aule (una per livello) per 10 lezioni: e, j nella prima aula; c,d,g nella seconda aula; b, h nella terza aula; a, f, i nella quarta aula

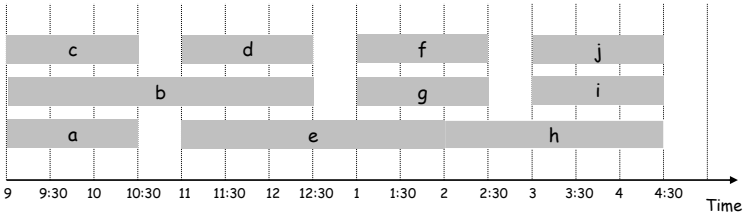


62

Partizionamento di intervalli

Esempio. Questo schedule usa solo 3 aule per le stesse attività: {c,d,f,j},{b,g,i},{a,e,h} .

Si noti che la disposizione delle lezioni lungo l'asse delle ascisse è fissato dall'input mentre la disposizione lungo l'asse delle y è fissato dall'algoritmo.



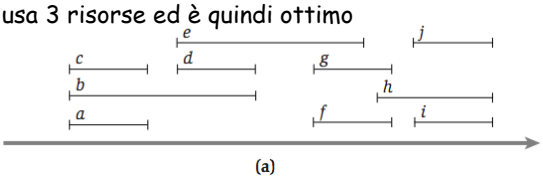
Partizionamento di intervalli: limite inferiore alla soluzione ottima

Def. Immaginiamo di disporre gli intervalli lungo l'asse delle ordinate in modo da non avere due intervalli che si sovrappongono alla stessa altezza (un qualsiasi schedule fa al nostro caso) . La profondità di un insieme di intervalli è il numero massimo di intervalli intersecabili con una singola linea verticale che si muove lungo l'asse delle ascisse.

Osservazione. Numero di risorse necessarie $\geq$ profondità.

Esempio. L'insieme di intervalli in figura (a) ha profondità 3. Lo schedule in figura (b) usa 3 risorse ed è quindi ottimo

Una risorsa per livello. Per un totale di 4 risorse



(a)

Una risorsa per livello. Per un totale di 3 risorse

(b)

Partizionamento di intervalli: soluzione ottima

Domanda. E' sempre possibile trovare uno schedule pari alla profondità dell'insieme di intervalli?

Osservazione. Se così fosse allora il problema del partizionamento si ridurrebbe a constatare quanti intervalli si sovrappongono in un certo punto.

- Questa è una caratteristica locale per cui un algoritmo greedy potrebbe essere la scelta migliore:
  - alloco una nuova risorsa solo se in un certo momento quelle gia` allocate sono tutte impegnate

Nel seguito vedremo un algoritmo greedy che trova una soluzione che usa un numero di risorse pari alla profondità e che quindi è ottimo

Idea dell'algoritmo applicata all'esempio delle lezioni:

- Considera le lezioni in ordine non decrescente dei tempi di inizio
- Ogni volta che esamina una lezione controlla se le può essere allocata una delle aule già utilizzate per qualcuna delle lezioni esaminate in precedenza. In caso contrario alloca una nuova aula.

65

Partizionamento di intervalli: Algoritmo greedy

```
Input: s1,s2, ... ,sn , f1,f2, ... ,fn
Sort intervals by starting time so that s1 ≤ s2 ≤ ... ≤ sn.
d ← 0
S ← Ø
for j = 1 to n {
   if (interval j can be assigned an already allocated resources v)
      assign resource v to interval j
      S ← S U {(j,v)}
   else
      allocate a new resource d + 1
      assign the new resource d+1 to interval j
      S ← S U {(j,d+1)}
      d ← d + 1
}
return S
```

- Osservazione. L'algoritmo greedy non assegna mai la stessa risorsa a due intervalli incompatibili
- Dimostreremo che alla j-esima iterazione del for il valore di d è pari alla profondità dell'insieme ordinato di intervalli {1,2,…,j}
- Il valore finale di d è quindi pari alla profondità dell'insieme di intervalli {1,…,n}

66

Partizionamento di intervalli: Ottimalità soluzione greedy

Lemma. Alla j-esima iterazione del for, il valore di d in quella iterazione è pari alla profondità dell'insieme di intervalli {1,2,…,j}.

Dim.

Caso in cui alla j-esima iterazione del for viene allocata una nuova risorsa

- Supponiamo che, alla j-esima iterazione del ciclo di for, d venga incrementato.
- La risorsa d è stata allocata perchè ciascuna delle altre d-1 risorse già allocate è assegnata al tempo s_j ad un intervallo che non è ancora terminato.
  → $f_i > s_j$ per ciascuna attività i che impegna una delle d-1 risorse
- **Siccome l'algoritmo considera gli intervalli in ordine non decrescente dei tempi di inizio**, i d-1 intervalli a cui sono assegnate le d-1 risorse iniziano non più tardi di s_j → $s_i \leq s_j$ e $f_i > s_j$, per ciascuna attività i che impegna una delle d-1 risorse
- Di conseguenza, questi d-1 intervalli e l'intervallo $[s_j,f_j]$ si sovrappongono al tempo s_j (sono cioè tutti intersecabili da una retta verticale che passa per s_j). Per definizione di profondità, d ≤ profondità di {1,2,…,j}
- Abbiamo osservato che il numero di risorse allocate per intervallo è maggiore uguale della sua profondità per cui d ≥ profondità di {1,2,…,j}
- Dagli ultimi due punti segue che d = profondità di {1,2,…,j}

67

Partizionamento di intervalli: Ottimalità soluzione greedy

Caso in cui alla j-esima iterazione del for non viene allocata una nuova risorsa:

- Sia j' l'ultima iterazione prima della j-esima in cui viene allocata una nuova risorsa.

1. Per quanto dimostrato nella slide precedente, d= profondità {1,2,…,j'}.
2. Siccome {1,2,…,j'} è contenuto in {1,2,…,j} allora si ha che profondità {1,2,…,j'} ≤ profondità di {1,2,…,j} .

- 1 e 2→ d ≤ profondità di {1,2,…,j}.
- Usiamo di nuovo il fatto che il numero di risorse allocate per un insieme di intervalli è maggiore o uguale della profondità dell'insieme. Si ha quindi d ≥profondità di {1,2,…,j}.
- Gli ultimi due punti implicano che d=profondità di {1,2,…,j}.

Partizionamento di intervalli: Ottimalità soluzione greedy

Teorema. L'algoritmo greedy usa esattamente un numero di risorse pari alla profondità dell'insieme di intervalli {1,2,…,n}

Dim. Per il lemma precedente, quando j=n il numero di risorse allocate è uguale alla profondità dell'intervallo {1,2,…,n}

Partizionamento di intervalli: Analisi Algoritmo greedy

Implementazione. $O(n \log n)$.

- Per ogni risorsa p, manteniamo il tempo di fine più grande tra quelli degli intervalli assegnati fino a quel momento a p. Indichiamo questo tempo con k_p
- Usiamo una coda a priorità di coppie della forma (p, k_p), dove p è una risorsa già allocata e k_p è l'istante fino al quale è occupata.
  - In questo modo l'elemento con chiave minima indica la risorsa v che si rende disponibile per prima
- Se $s_j \geq k_v$ allora all'intervallo j può essere allocata la risorsa v. In questo caso cancelliamo v dalla coda e la reinseriamo con chiave $k_v=f_j$. In caso contrario allochiamo una nuova risorsa e la inseriamo nella coda associandole la chiave f_j
- Se usiamo una coda a priorità implementata con heap binario ogni operazione sulla coda richiede $O(\log m)$, dove m è la profondità dell'insieme di intervalli. Poiché vengono fatte $O(n)$ operazioni di questo tipo, il costo complessivo del for è $O(n \log m)$.
- A questo va aggiunto il costo dell'ordinamento che è $O(n \log n)$. Siccome $m \leq n$ il costo dell'algoritmo è $O(n \log n)$