Metodo *Greedy* (avido, goloso)

11 maggio 2023

Punto della situazione

Tecniche di progettazione di algoritmi:

- forza bruta, esaustivi, naif
- divide et impera;
- programmazione dinamica;
- tecnica greedy
- Algoritmi esaustivi intelligenti: backtracking e branch-and-bound

Esempi:

1. Selezione di intervalli (versione non pesata) ([KT] par. 4.1)
2. Partizionamento di intervalli ([KT] par. 4.1)
3. Scheduling che minimizza il ritardo ([KT] par. 4.2)
4. Codici di Huffman
5. e altri sui grafi: MST, cammini minimi, …

Tecnica greedy

- Serve per risolvere problemi di ottimizzazione = ricerca di una soluzione ammissibile che ottimizzi (max/min) un valore associato.

- Non sempre è possibile utilizzarla per ottenere una soluzione ottimale (nel caso, provare con la programmazione dinamica)

- Schema molto semplice

- Prova della correttezza… di meno

Schema algoritmo greedy

- Ordino secondo un criterio di convenienza
- Inizializzo SOL = insieme vuoto
- Considero ogni oggetto in ordine di convenienza
  - Se è compatibile con SOL lo aggiungo a SOL
- Restituisco SOL

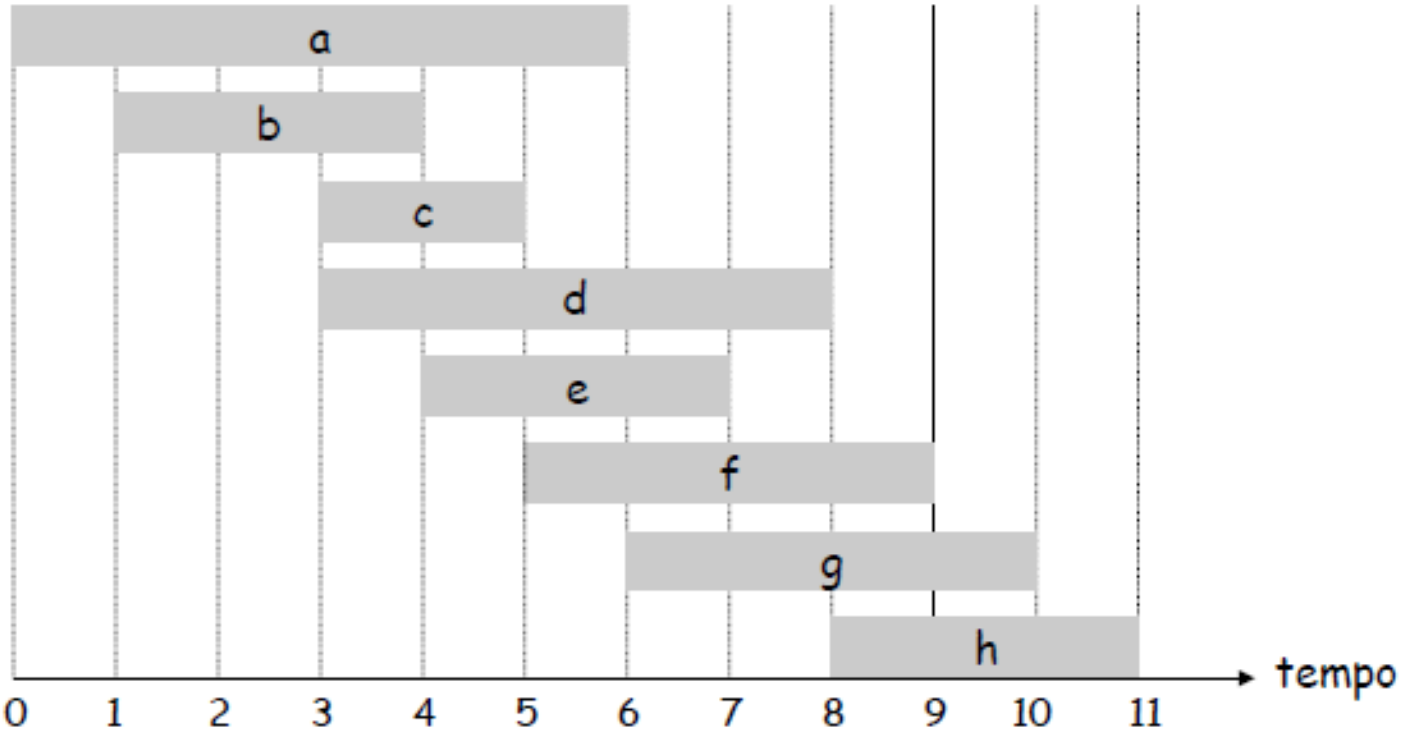
4.1 Interval Scheduling

(non pesato)

Schedulazione intervalli

Schedulazione intervalli.

- Job j inizia a s_j e finisce a f_j.
- Due job sono compatibili se hanno intersezione vuota.
- Obiettivo: trovare sottoinsieme massimale di job mutuamente compatibili.



Schedulazione intervalli: Algoritmo Greedy

Algoritmo Greedy. Considera job in ordine crescente del tempo di fine f_j. Prendere job se è compatibile con quelli già presi.

```
Ordina job per tempo di fine f1 ≤ f2 ≤ ... ≤ fn.
```

insieme job scelti

```
A ← {1}
for j = 2 to n {
   if (job j è compatibile con A)
      A ← A ∪ {j}
}
return A
```

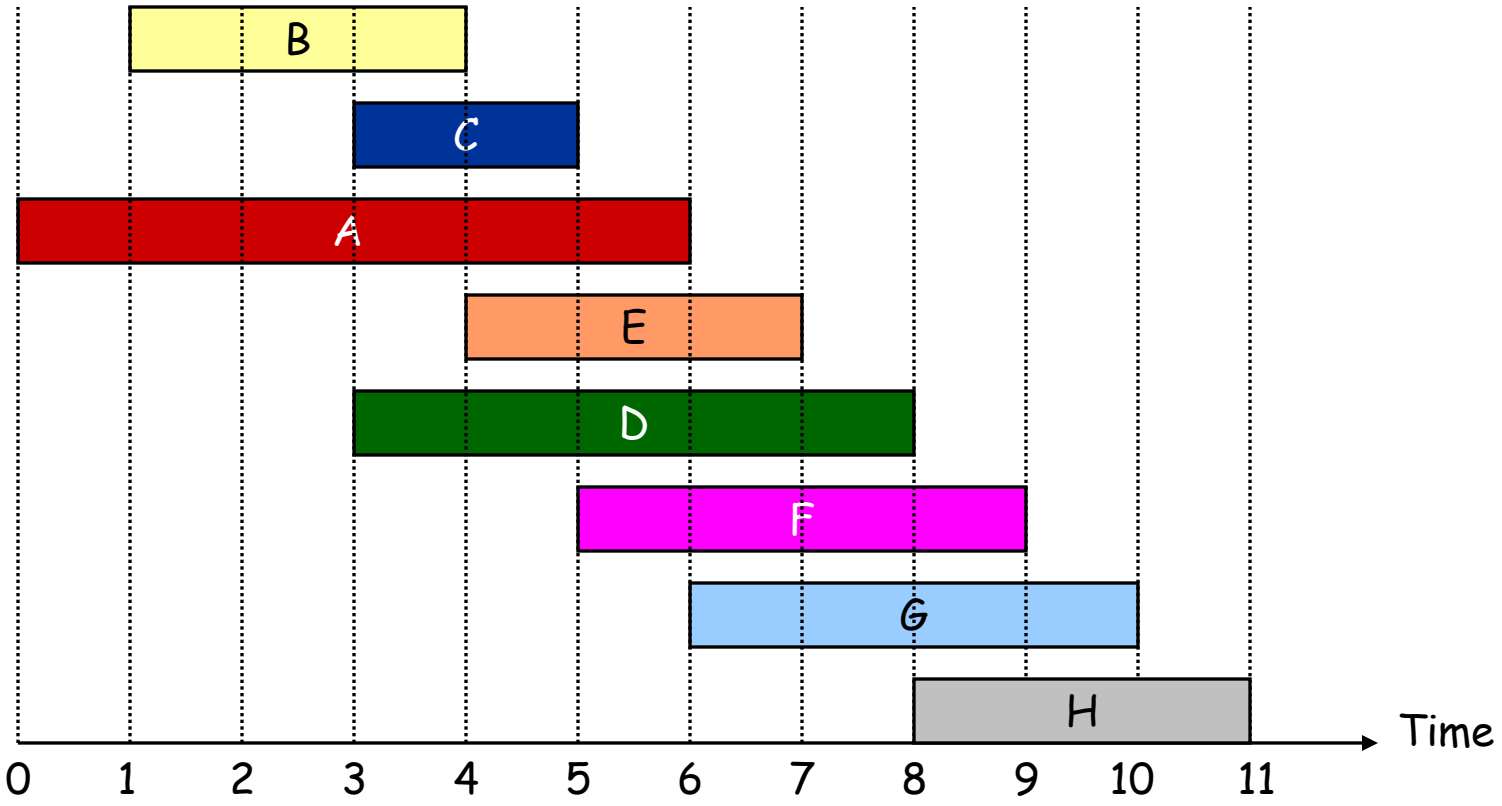
Schedulazione intervalli: Implementazione Algoritmo Greedy

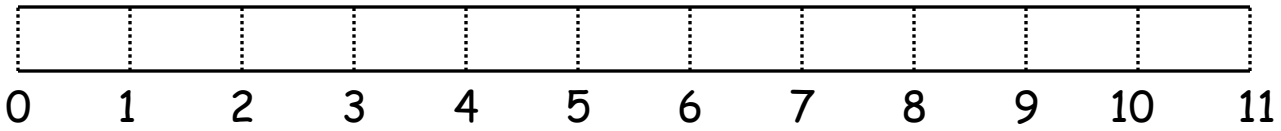
Implementazione Algoritmo Greedy. $O(n \log n)$.

- Denota j^* l'ultimo job aggiunto ad A.
- Job j è compatibile con A se $s_j \geq f_{j^*}$.

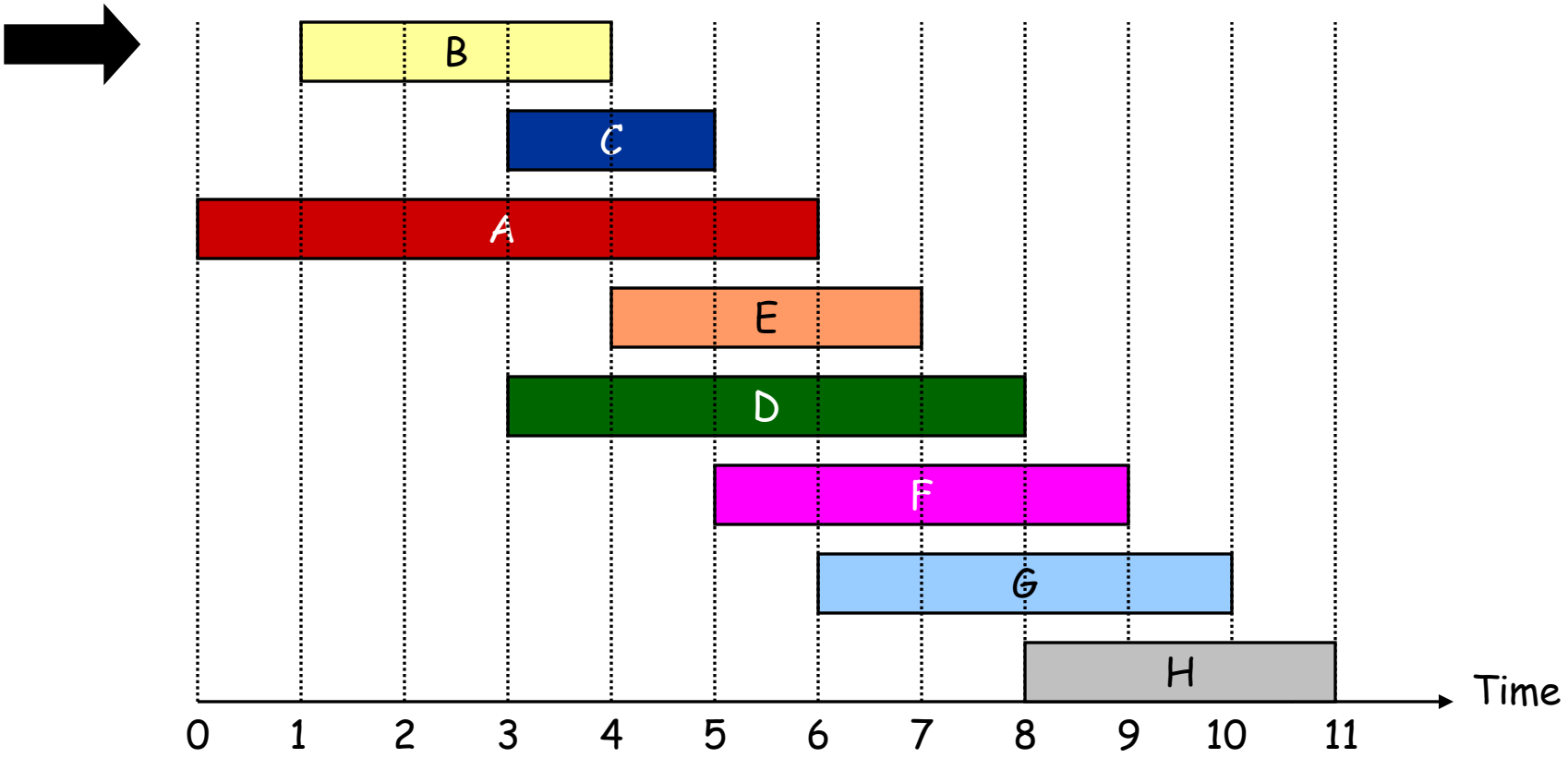
```
Ordina job per tempo di fine f1 ≤ f2 ≤ ... ≤ fn.

A ← {1}
j* = 1
for j = 2 to n {
   if sj ≥ fj*
      {A ← A ∪ {j}
       j* = j
      }
}
return A
```

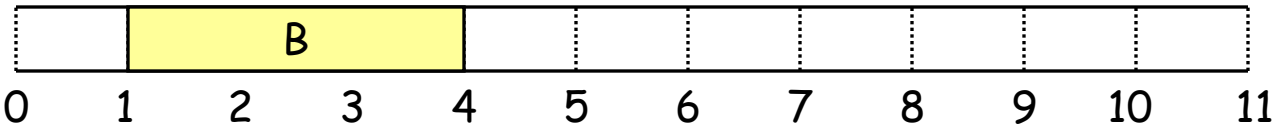
Interval Scheduling



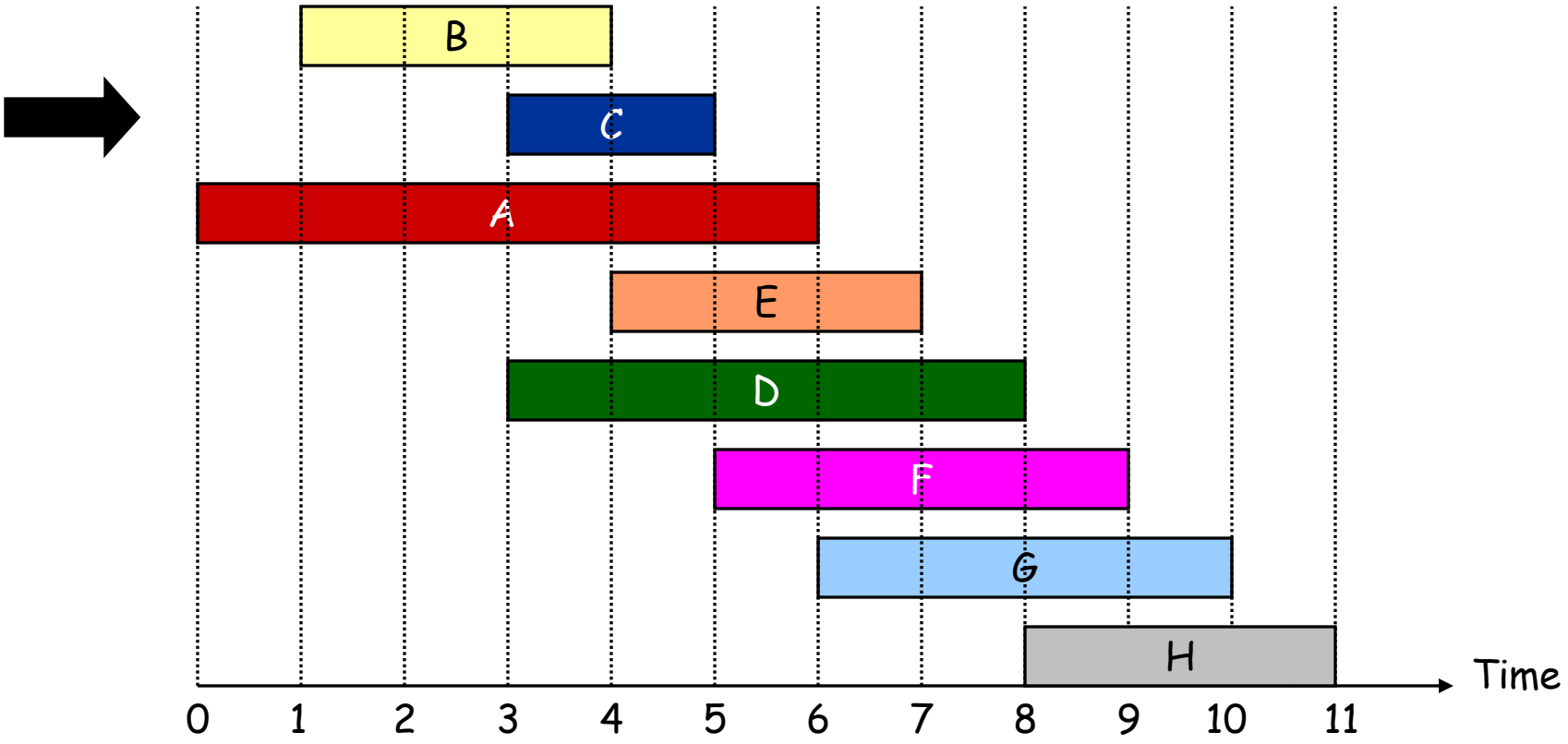


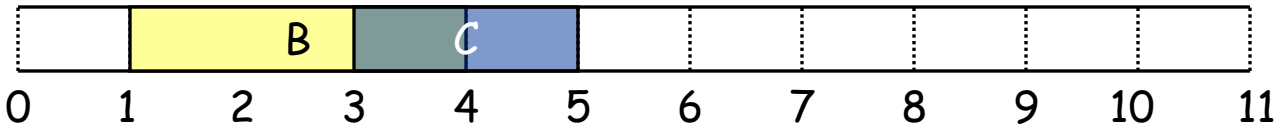
Interval Scheduling



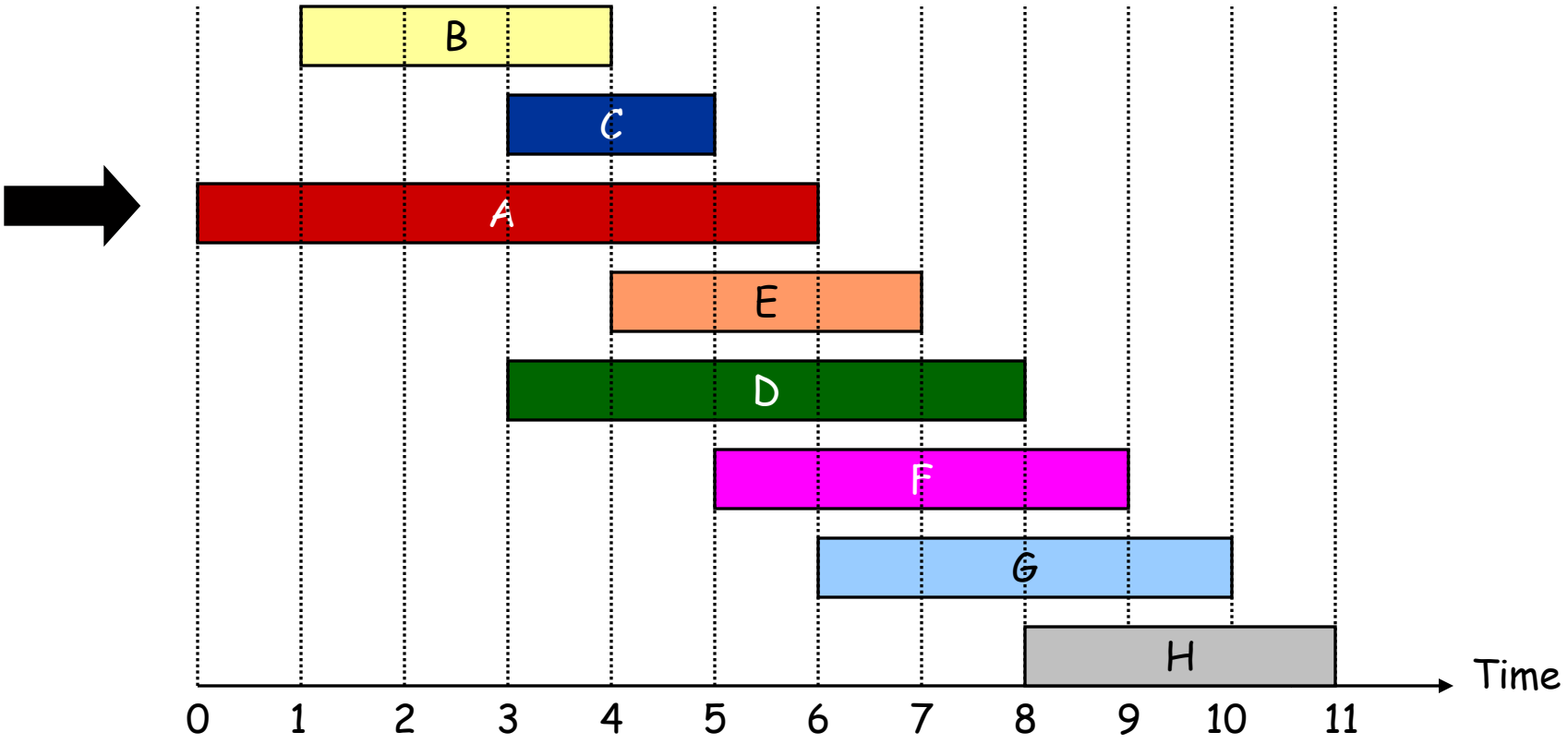


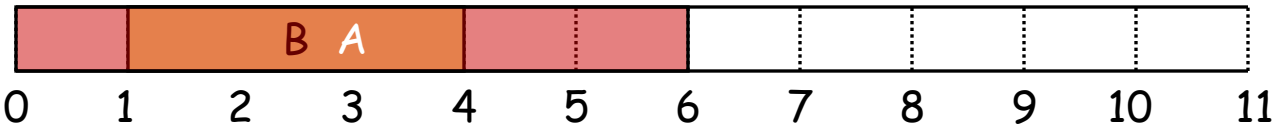
Interval Scheduling



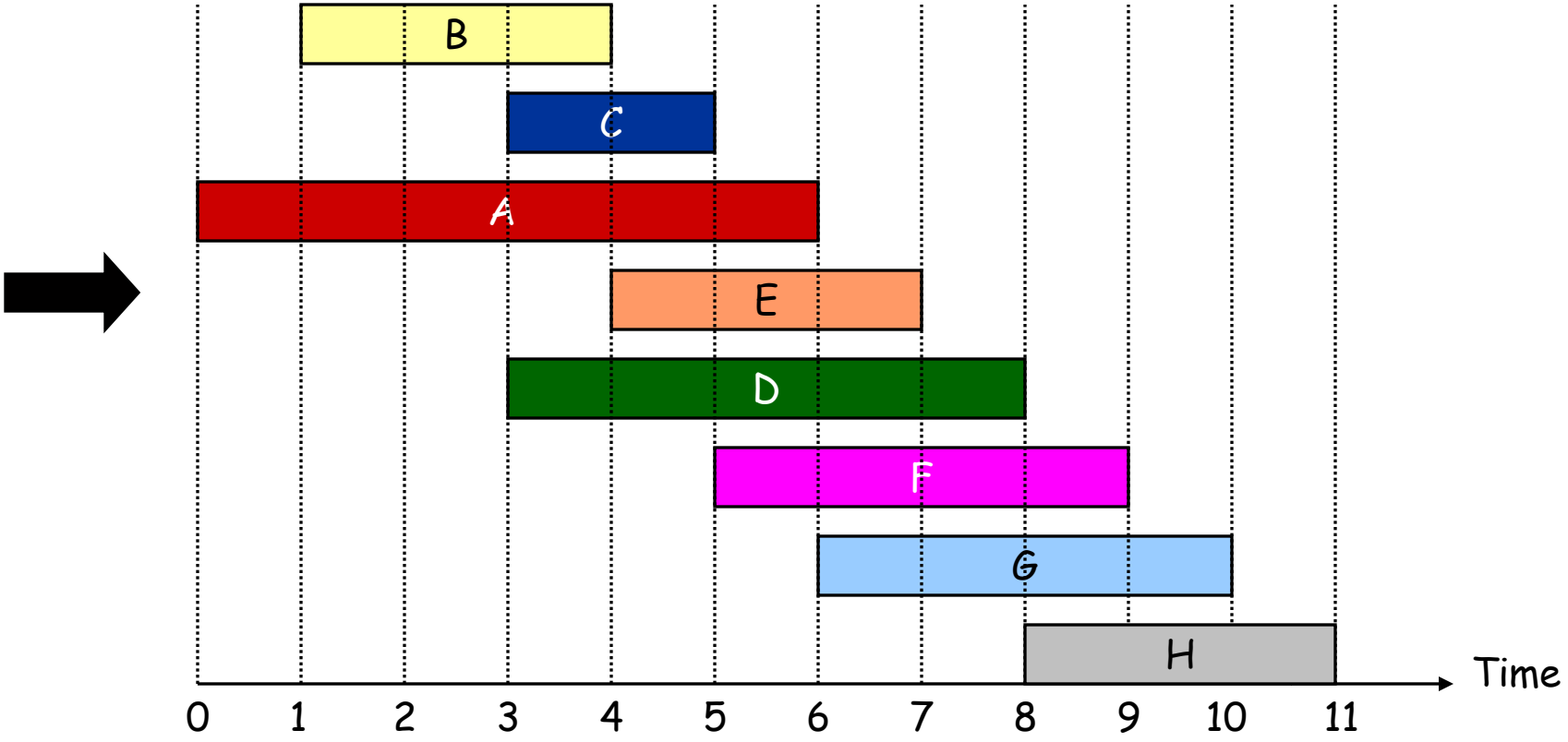


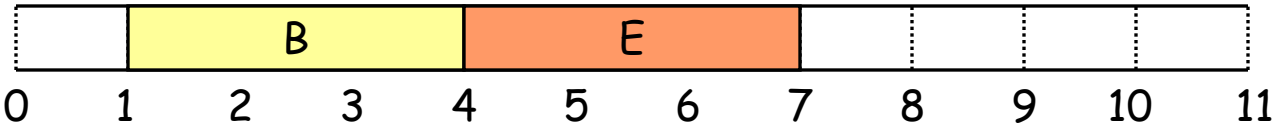
Interval Scheduling



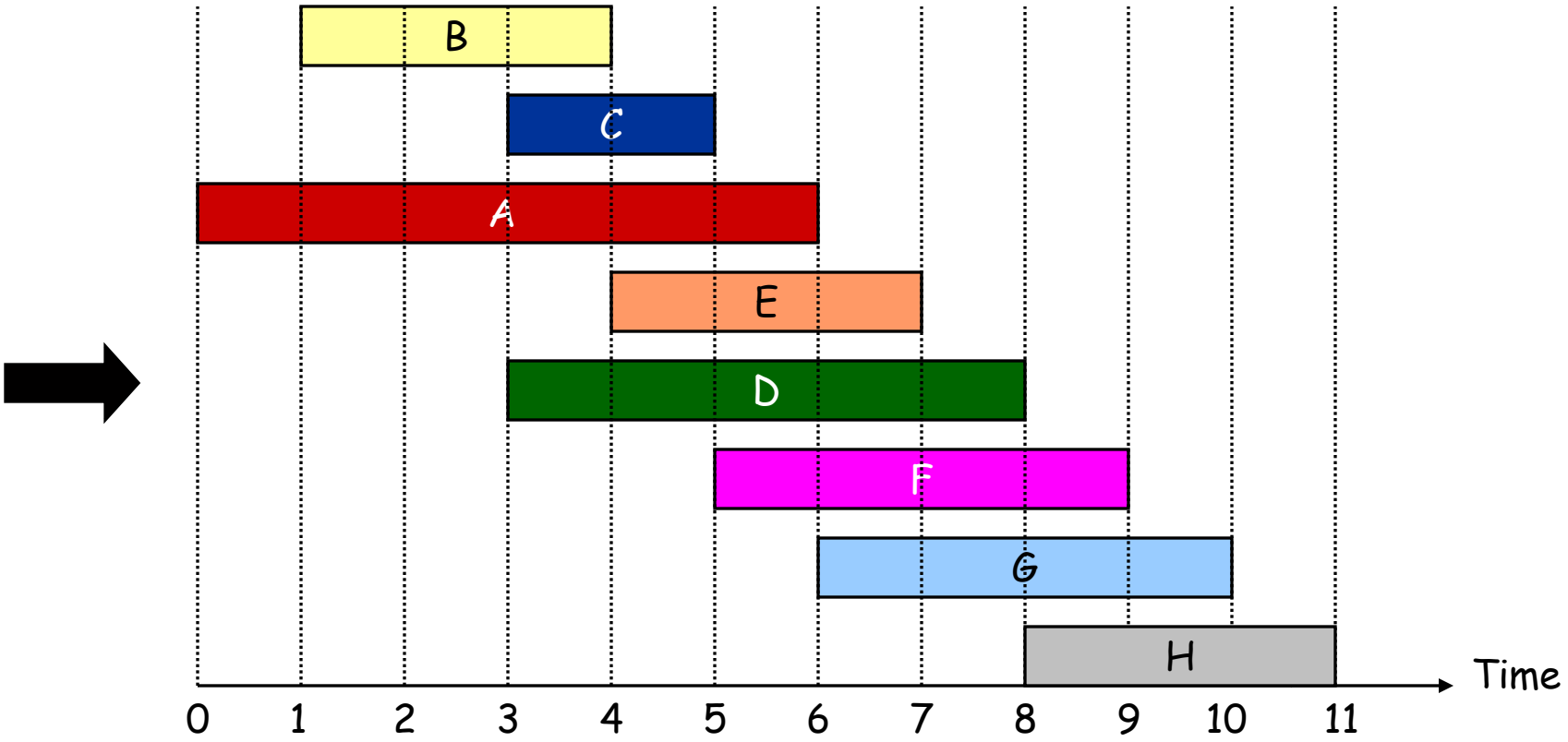


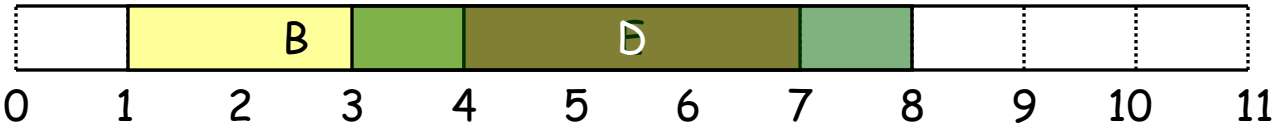
Interval Scheduling



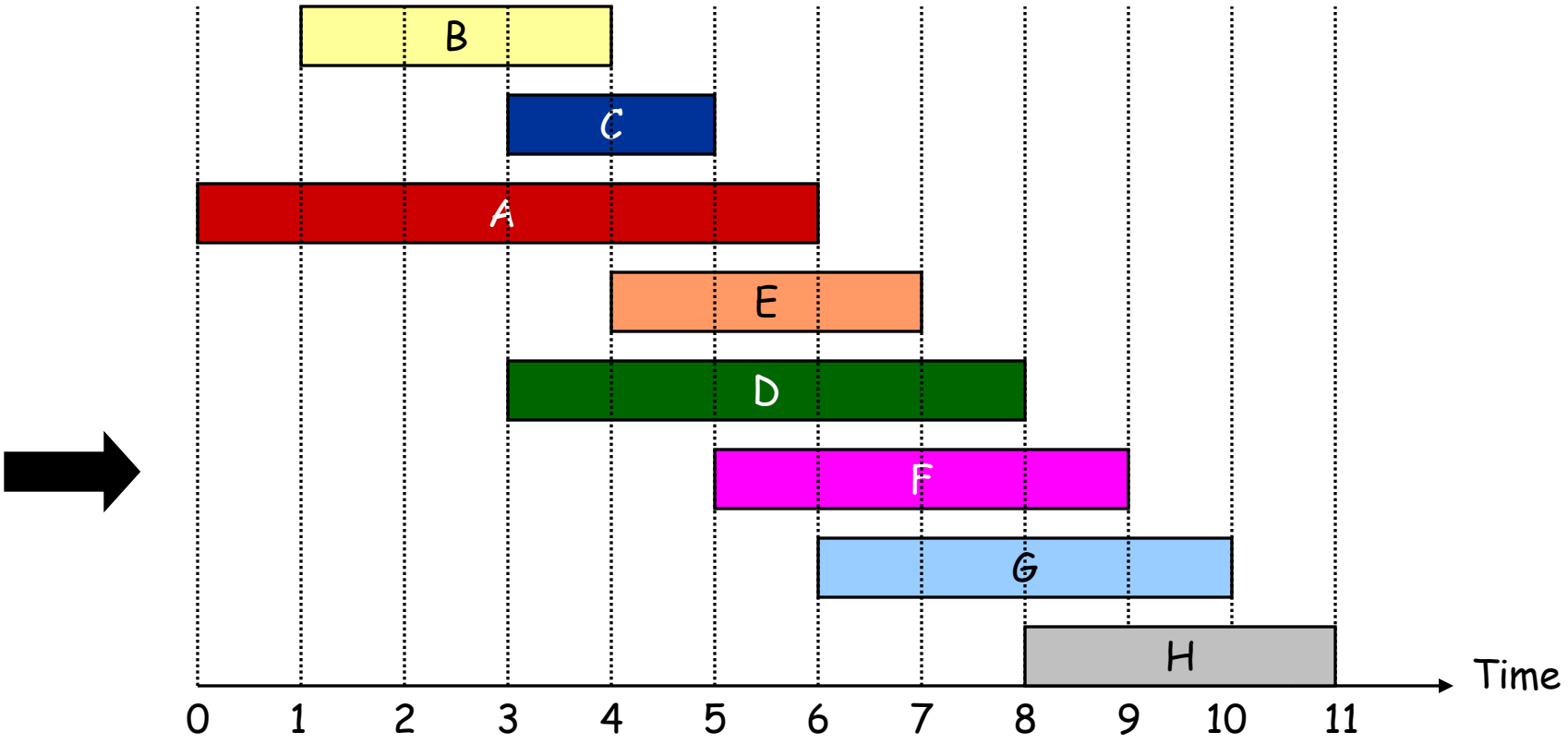


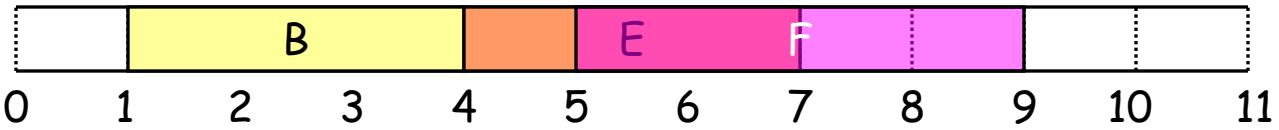
Interval Scheduling



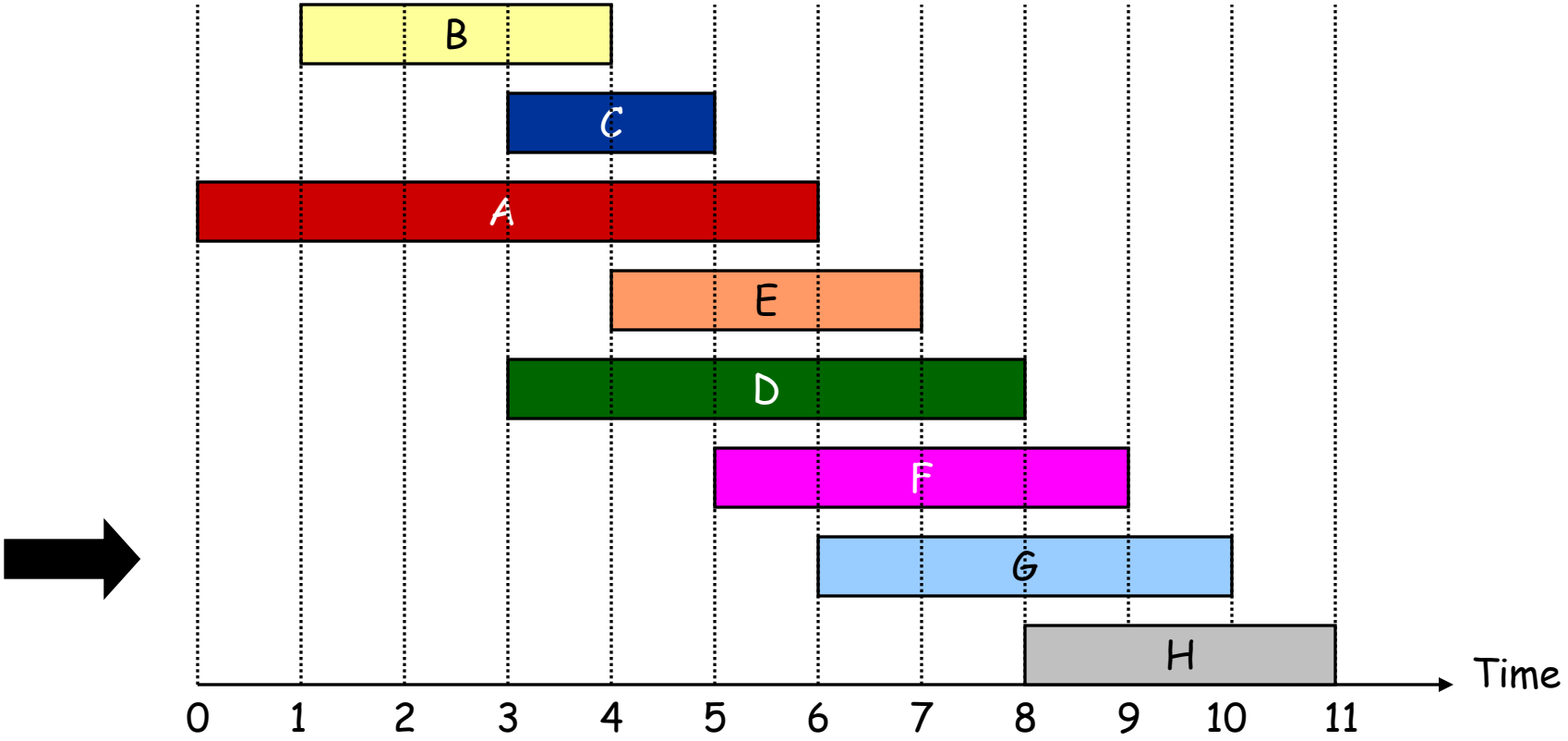


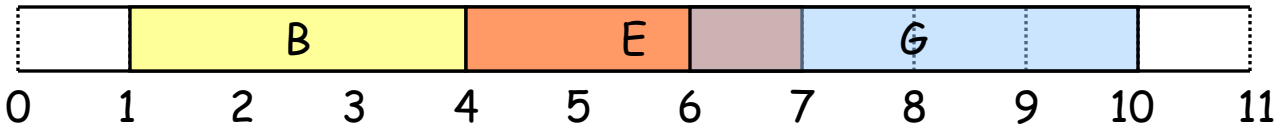
Interval Scheduling



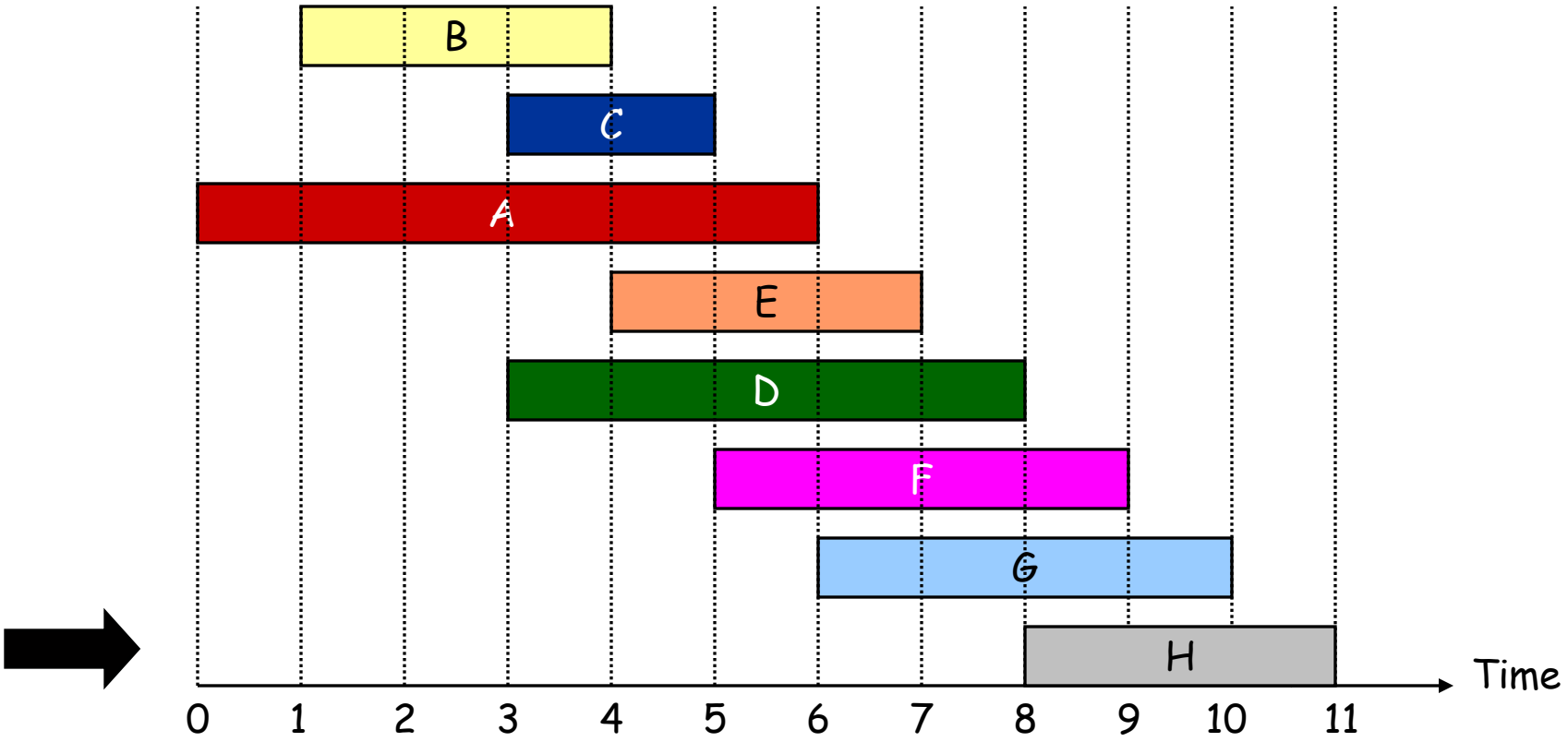


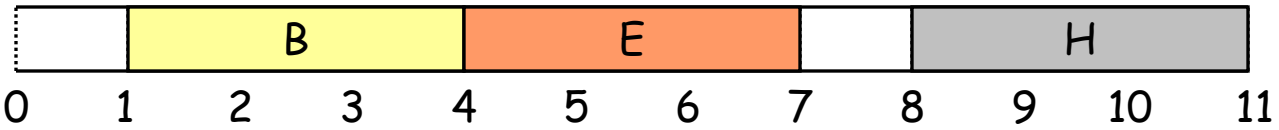
Interval Scheduling





Interval Scheduling





Correttezza

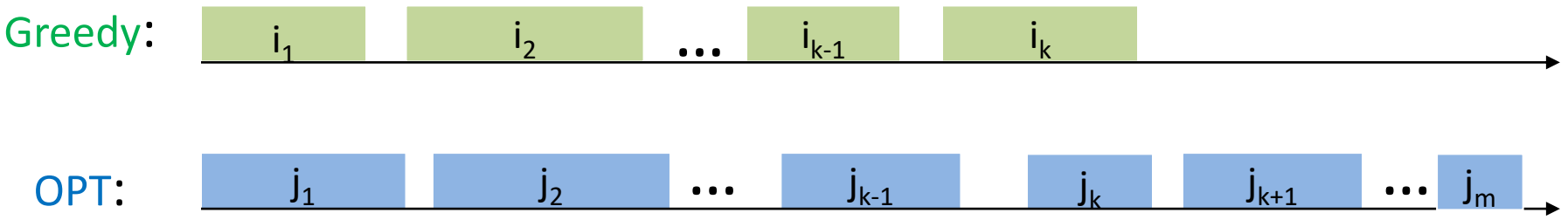
- A è una soluzione ammissibile (un insieme di intervalli compatibili).
  L'algoritmo aggiunge un intervallo se è compatibile con l'ultimo inserito, ma lo sarà pure con gli altri perché sono inseriti in ordine di tempo di fine.

- A è una soluzione ottimale.

  Lo dimostriamo con la tecnica "Greedy algorithm stays ahead" che consiste nel confrontare la soluzione fornita dall'algoritmo greedy con una ottimale OPT, e dimostrare che, ad ogni passo, la soluzione greedy "stays ahead", sta avanti, è in qualche senso migliore di OPT.

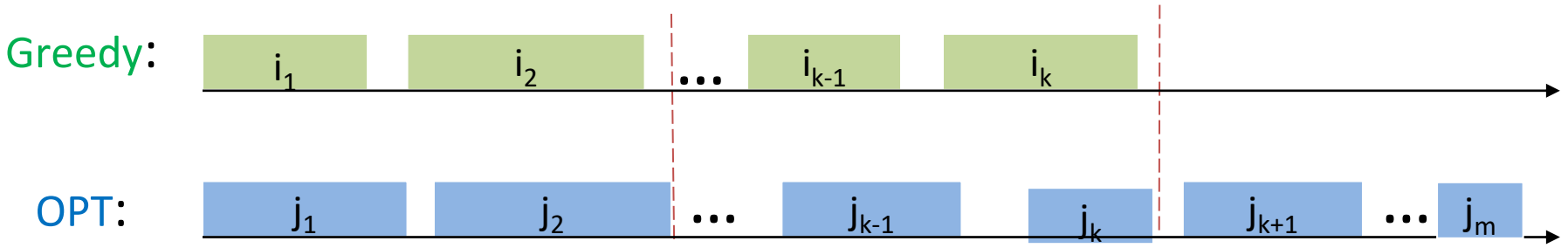
  Supponiamo Greedy abbia k intervalli, e OPT ne abbia m, con $m \geq k$.

  Tesi: k=m.



Greedy algorithm stays ahead

- A è una soluzione ottimale.



Ricorda: l'algoritmo greedy sceglie ad ogni iterazione il primo intervallo (in ordine di tempo di fine) fra quelli compatibili con l'ultimo appena aggiunto, finchè avrà esaminato tutti gli intervalli e termina.

Si ha che: $f(i_h) \leq f(j_h)$ per ogni $1 \leq h \leq k$ (segue dimostrazione per induzione)

In particolare $f(i_k) \leq f(j_k)$. Quindi, se esistesse j_{k+1} compatibile con j_k, a maggior ragione sarebbe compatibile con i_k e l'algoritmo greedy l'avrebbe considerato (e scelto), anziché terminare: assurdo!

Greedy algorithm stays ahead

$f(i_h) \leq f(j_h)$ per ogni $1 \leq h \leq k$.

Dimostrazione per induzione:

La proprietà è vera per h=1. Poichè l'algoritmo sceglie il primo intervallo in ordine di fine crescente: $f(i_1) \leq f(j_1)$.

Per ipotesi induttiva, supponiamo vera $f(i_h) \leq f(j_h)$, per $1 \leq h \leq k-1$, e dimostriamo che $f(i_{h+1}) \leq f(j_{h+1})$.

Confrontiamo i_{h+1} con j_{h+1}. Anche j_{h+1} è compatibile con i_h. Infatti $s(j_{h+1}) \geq f(j_h)$ e, per ipotesi induttiva, $f(j_h) \geq f(i_h)$. Essendo sia i_{h+1} che j_{h+1} sono compatibili con i_h dato che l'algoritmo ha scelto i_{h+1} , significa che $f(i_{h+1}) \leq f(j_{h+1})$.

Correttezza: seconda dimostrazione

Teorema. L'algoritmo greedy è ottimale.

Prova. (per assurdo)

- Assumiamo che la scelta greedy non sia ottimale.
- Siano $i_1, i_2, \ldots i_k$ job scelti in modo greedy.
- Siano $j_1, j_2, \ldots j_m$ job in una soluzione ottimale con $i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r$ dove r è il più grande valore possibile.

Correttezza: seconda dimostrazione

Teorema. L' algoritmo greedy è ottimale.

Prova. (per assurdo)

- Assumiamo che la scelta greedy non sia ottimale.
- Siano $i_1, i_2, \ldots i_k$ job scelti in modo greedy.
- Siano $j_1, j_2, \ldots j_m$ job in una soluzione ottimale con $i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r$ dove r è il più grande valore possibile.

4.1 Interval Partitioning

Interval Partitioning

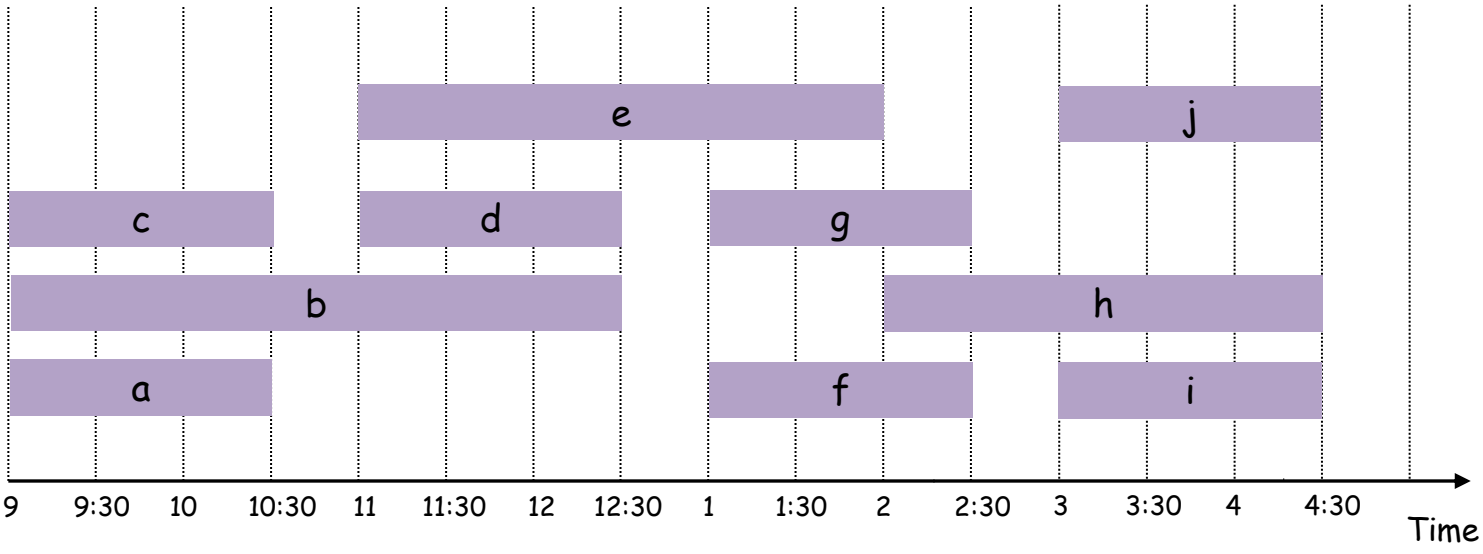
adesso abbiamo più risorse e vogliamo accontentare tutti (col minimo delle risorse….).

Interval partitioning

Lecture j starts at s_j and finishes at f_j.

Goal: find minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room.

Ex: This schedule uses 4 classrooms to schedule 10 lectures.



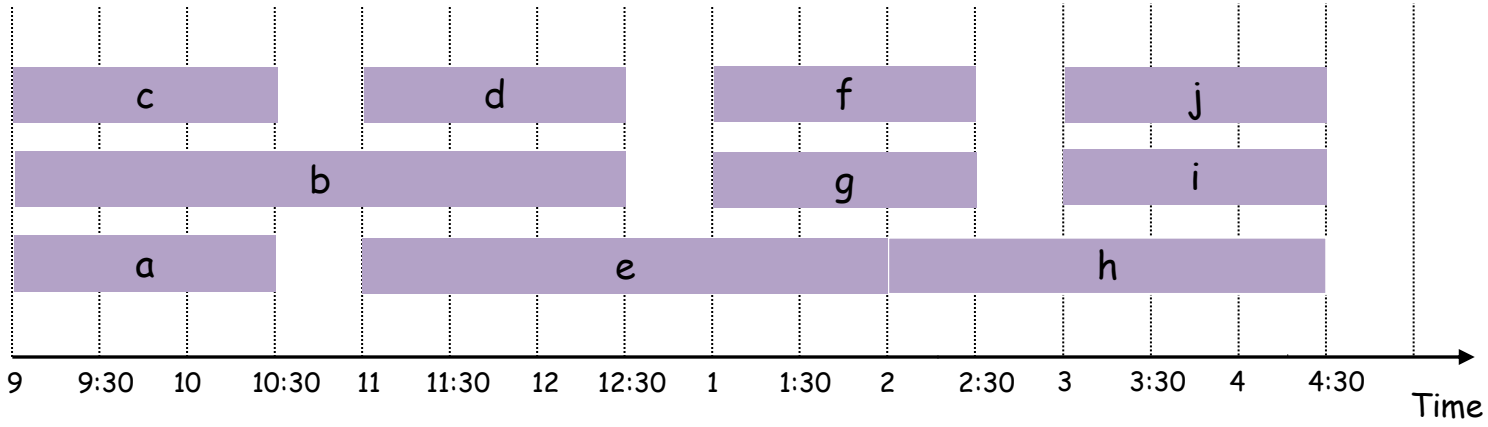
Interval Partitioning

Interval partitioning.

Lecture j starts at s_j and finishes at f_j.

Goal: find minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room.

Ex: This schedule uses only 3.



Interval Partitioning: Lower Bound on Optimal Solution

- Def. The depth of a set of open intervals is the maximum number that contain any given time.

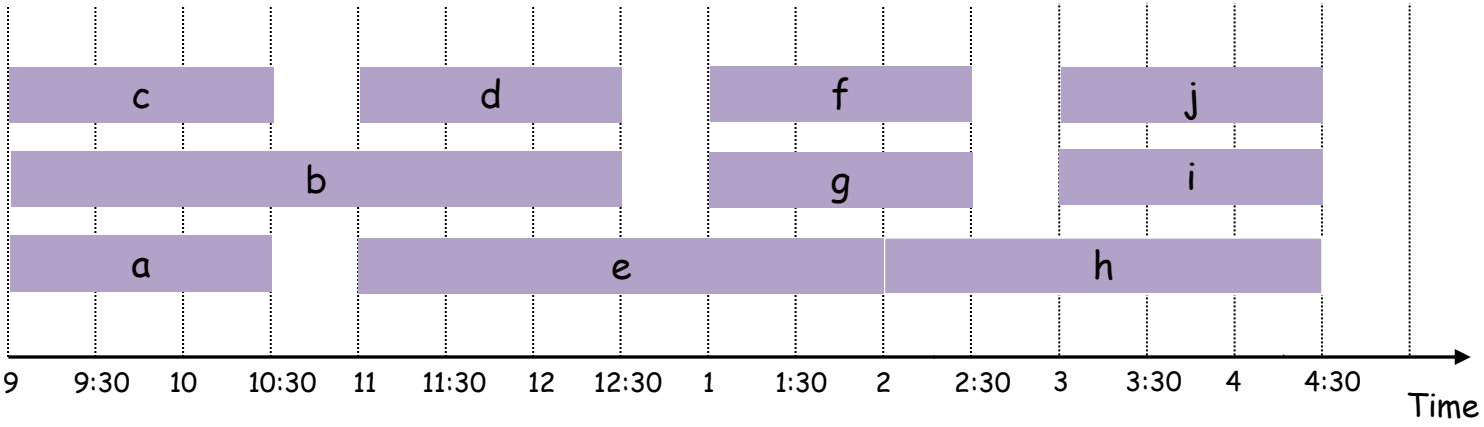
- Key observation. Number of classrooms needed $\geq$ depth.

- Ex: Depth of schedule below = 3 $\Rightarrow$ schedule below is optimal.

a, b, c all contain 9:30

- Q. Does there always exist a schedule equal to depth of intervals?

Osservazione chiave:
se d lezioni si accavallano all'istante t $\Rightarrow$ qualsiasi schedule usa almeno d aule



Interval Partitioning: Greedy Algorithm

Greedy algorithm.

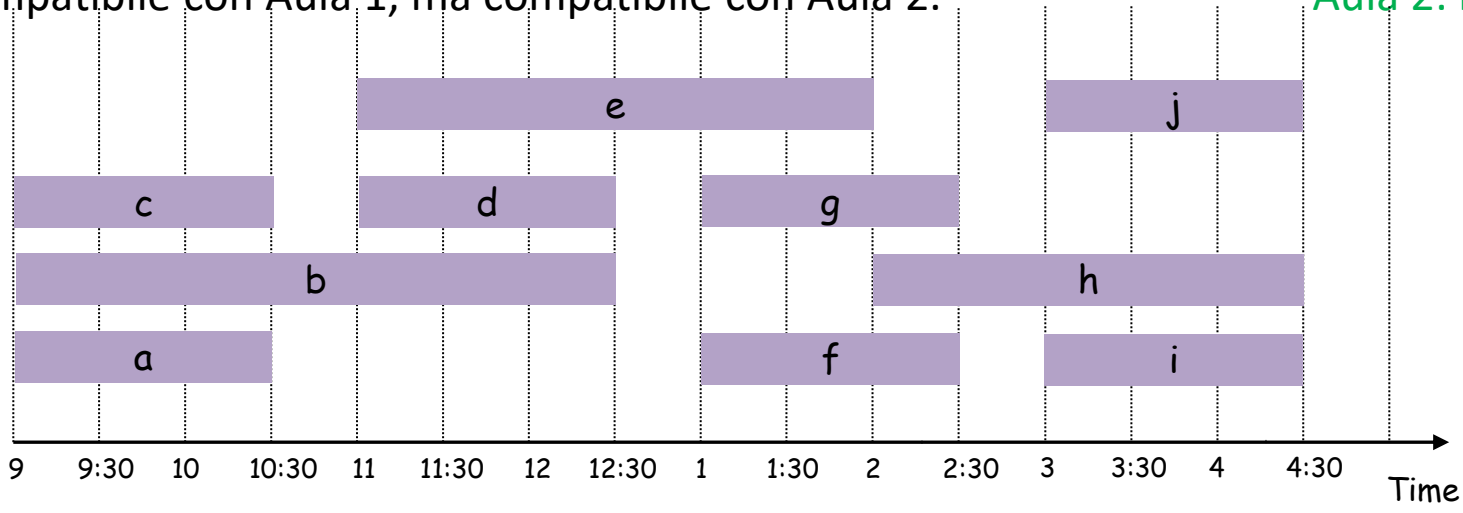
Consider lectures in increasing order of start time.

Assign lecture to any compatible classroom. If no compatible classroom exists, allocate a new classroom and schedule it in.

```
Sort intervals by starting time so that s_1 ≤ s_2 ≤ ... ≤ s_n.

d ← 0      // number of allocated classrooms

for j = 1 to n {
   if (lecture j is compatible with some classroom k)
      schedule lecture j in classroom k
   else
      allocate a new classroom d + 1
      schedule lecture j in classroom d + 1
      d ← d + 1
}
```

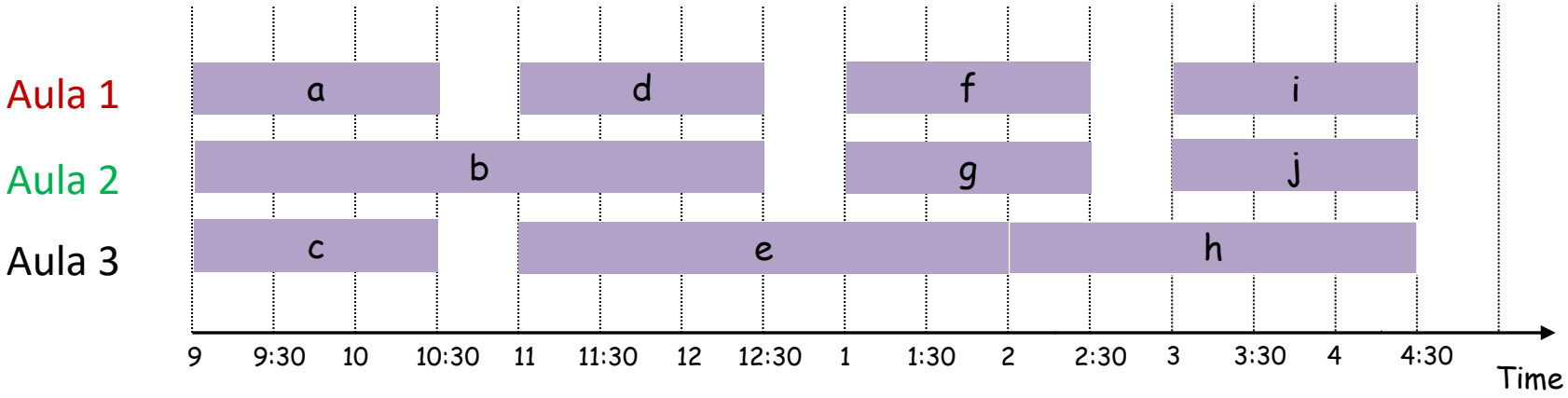
Esempio di esecuzione

- Ordina per start time: a, b, c, …, j.
- a in Aula 1: Aula 1: a
- b è incompatibile con Aula 1: apro Aula 2: Aula 2: b
- c è incompatibile con Aula 1 e Aula 2: apro Aula 3: Aula 3: c
- d è compatibile con Aula 1: Aula 1: a, d
- e è incompatibile con Aula 1 e Aula 2, ma compatibile con Aula 3: Aula 3: c, e
- f è compatibile con Aula 1: Aula 1: a, d, f
- g è incompatibile con Aula 1, ma compatibile con Aula 2: Aula 2: b, g
- h è incompatibile con Aula 1 e Aula 2, ma compatibile con Aula 3: Aula 3: c, e, h
- i è compatibile con Aula 1: Aula 1: a, d, f, i
- j è incompatibile con Aula 1, ma compatibile con Aula 2. Aula 2: b, g, j



Esempio (continua)

- a in Aula 1: Aula 1: a
- b è incompatibile con Aula 1: apro Aula 2: Aula 2: b
- c è incompatibile con Aula 1 e Aula 2: apro Aula 3: Aula 3: c
- d è compatibile con Aula 1: Aula 1: a, d
- e è incompatibile con Aula 1 e Aula 2, ma compatibile con Aula 3: Aula 3: c, e
- f è compatibile con Aula 1: Aula 1: a, d, f
- g è incompatibile con Aula 1, ma compatibile con Aula 2: Aula 2: b, g
- h è incompatibile con Aula 1 e Aula 2, ma compatibile con Aula 3: Aula 3: c, e, h
- i è compatibile con Aula 1: Aula 1: a, d, f, i
- j è incompatibile con Aula 1, ma compatibile con Aula 2. Aula 2: b, g, j



Implementation

```
Sort intervals by starting time so that s1 ≤ s2 ≤ ... ≤ sn.

d ← 0     ←number of allocated classrooms

for j = 1 to n {
   if (lecture j is compatible with some classroom k)
      schedule lecture j in classroom k
   else
      allocate a new classroom d + 1
      schedule lecture j in classroom d + 1
      d ← d + 1
}
```

Implementation in O(n log n):

To test compatibility of j: keep the classrooms in a priority queue {(k, $maxf_k$)}, where for each classroom k, $maxf_k$ maintains the maximum finish time of a job in the class.

Job j is compatible with some classroom iff $\mathbf{s}_j$ is greater than or equal to the minimum priority.

If $\mathbf{s}_j$ is greater than or equal to the minimum priority, schedule j in a minimal classroom and update (increase) the priority of such a classroom. Otherwise, allocate a new classroom, i.e. insert (d+1,f_j) in the priority queue.

The priority queue has at most d element, with d=O(n). With a heap implementation, each operation requires O(log d), O(n log d) in total.

Correctness: structural bound technique

Greedy algorithm solution is admissible: Greedy algorithm never schedules two incompatible lectures in the same classroom.

Theorem. Greedy algorithm solution is optimal.

Proof.

Let d = number of classrooms that the greedy algorithm allocates.

Classroom d is opened because we needed to schedule a job, say x, that is incompatible with all d-1 other classrooms.

Since we sorted by start time, all these incompatibilities are caused by lectures that start no later than s_x.

Thus, we have d lectures overlapping at time $s_x + \varepsilon$.

Key observation $\Rightarrow$ all schedules use $\geq$ d classrooms. ▪

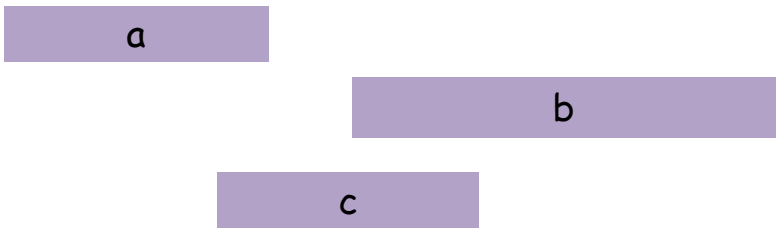
Nell'esempio: d = 3 è il numero di aule aperte dall'algoritmo.
L' Aula 3 è aperta perché la lezione x=c è incompatibile con le Aule 1 e 2, in cui ho già inserito a e b; a e b sono state considerate prima perché $s_a = s_b = 9 \leq s_c = 9$.
Quindi abbiamo d=3 lezioni che si accavallano al tempo s_c=9 (o anche 9:30).
Osservazione chiave:
se d lezioni si accavallano al tempo t $\Rightarrow$ qualsiasi schedule usa almeno d aule

Osserva…

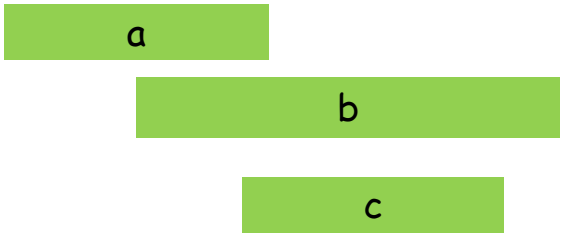
Dove interviene nella prova la scelta di un ordinamento basato sui tempi di inizio s_i crescenti?

«Since we sorted by start time, all these incompatibilities are caused by lectures that start no later than s_x , **thus**, we have d lectures overlapping at time $s_x + \varepsilon$»

Nota: se le lezioni non fossero selezionate secondo i tempi di inizio crescenti, si sarebbe potuto avere che, pur essendo che c si accavalla sia con a che con b, i compiti a, b e c non si accavallassero in uno stesso istante $s_c + \varepsilon$:



Ordine qualsiasi



Ordinate per tempo di inizio

Quesito 1 (*Scheduling attività*)
Si consideri un algoritmo greedy (goloso) per il problema della selezione delle attività (non pesato) basato sulla scelta dell'attività che **inizia per ultima**.
a) Se pensate che tale algoritmo restituisca sempre una soluzione ottimale, dimostratelo formalmente.
a') Se pensate che tale algoritmo non sempre restituisca una soluzione ottimale, mostrate un contro-esempio.

Quesito 2 (*Campi di calcetto*)
Dopo il successo del vostro primo campo di calcetto, avete aperto molti altri campi di calcetto, all'interno di un unico complesso. Ogni giorno raccogliete le richieste per utilizzare i vostri campi, ognuna specificata da un orario di inizio e un orario di fine. Oramai avete un numero di campi sufficiente ad accontentare sempre tutte le richieste. Volete però organizzare le partite nei campi in modo da accontentare tutti, senza che vi siano sovrapposizioni di orari, ma con il minimo numero possibile di campi (la manutenzione costa!).
Formalizzate il problema reale in un problema computazionale e risolvetelo nel modo più efficiente possibile. E' necessario giustificare la correttezza della vostra soluzione e valutarne l'efficienza.

Quesito 3 (*Cambio monete greedy*)
Si consideri il problema di determinare il minimo numero di monete da 1, 5, 10 centesimi necessarie per scambiare un ammontare di denaro D, supponendo di poter utilizzare un numero illimitato di monete dello stesso taglio.
a) Descrivere ed analizzare un algoritmo *greedy* che risolve tale problema
b) Dimostrare la correttezza dell'algoritmo proposto

MULTIPLE

1) 1 ☐

Si consideri il problema dello *scheduling* di attività (non pesato). Quale delle seguenti affermazioni è vera?

A. Ogni soluzione ottimale contiene l'intervallo che finisce per primo
B. Ogni soluzione ottimale contiene l'intervallo che inizia per primo
C. Esiste una soluzione ottimale che contiene l'intervallo che finisce per primo
D. Nessuna delle risposte precedenti

2) 2 ☐

Una comitiva di amici in vacanza vuole noleggiare delle moto per visitare la località. Ognuno vuole effettuare il giro in un orario a sua scelta, indipendentemente dagli altri. Mario che non sopporta il sole, dalle 8 alle 11, Giovanna che è dormigliona, dalle 13 alle 15, e così via. Poiché l'affitto delle moto si paga per l'intera giornata, decidono di organizzarsi in modo che ogni moto sia utilizzata da più persone nell'arco della giornata. E si chiedono: qual è il minimo numero di moto che dobbiamo noleggiare? Il loro è un problema di:

A. Scheduling di intervalli
B. Scheduling di intervalli pesato
C. Partizionamento di intervalli
D. Nessuna delle risposte precedenti

Calendario

- Lezione 29 *(12 maggio)*: Greedy 2
- *Settimana del 17 maggio*: Greedy 3 + Esercitazione + MST1
- *Settimana del 24 maggio*: MST2 + Cammini minimi greedy + Algoritmi esaustivi intelligenti. FINE programma!
- **Lezione 36** *(31 maggio)*: Esercitazione
- Lezione 36+1 *(1 giugno)*: Esercitazione extra?

- Giovedì 8 giugno ore 15, P3+P4:

  Seconda prova intercorso e pre-appello

- Tutorato
  Maggio: mercoledì ore 16-18 in F8 (?)
  Continuerà anche dopo la fine del corso, in preparazione agli appelli, in orari da stabilire