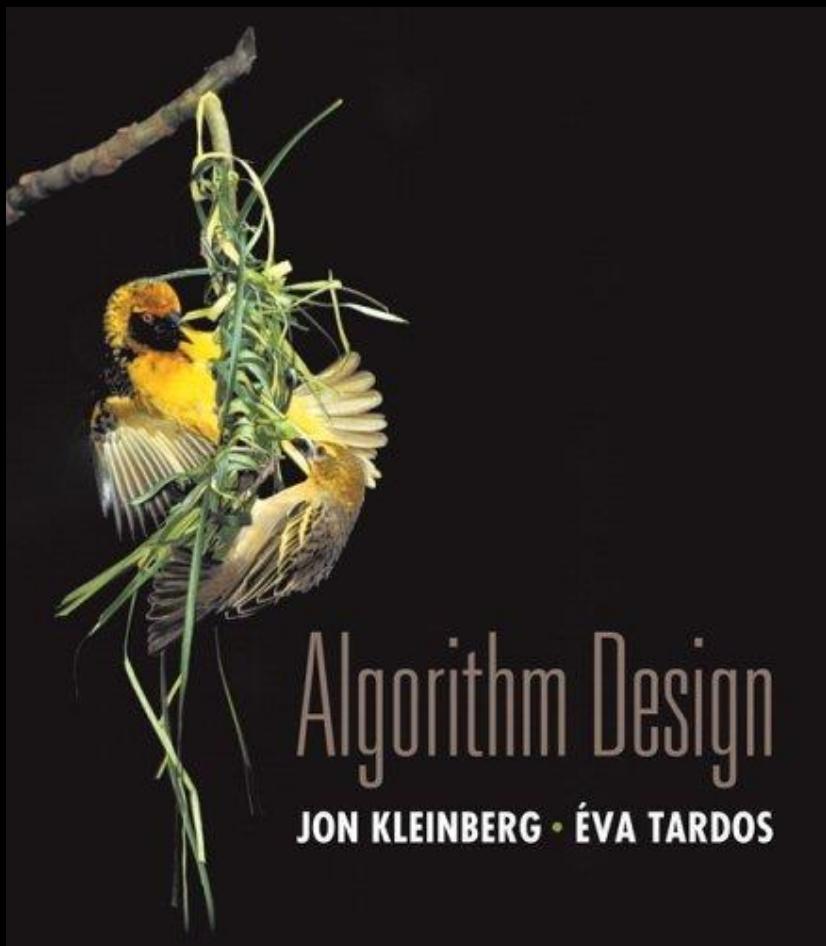


Chapter 4

Greedy Algorithms



Slides by Kevin Wayne.
Copyright © 2005 Pearson-Addison Wesley.
All rights reserved.

Greedy Algorithms

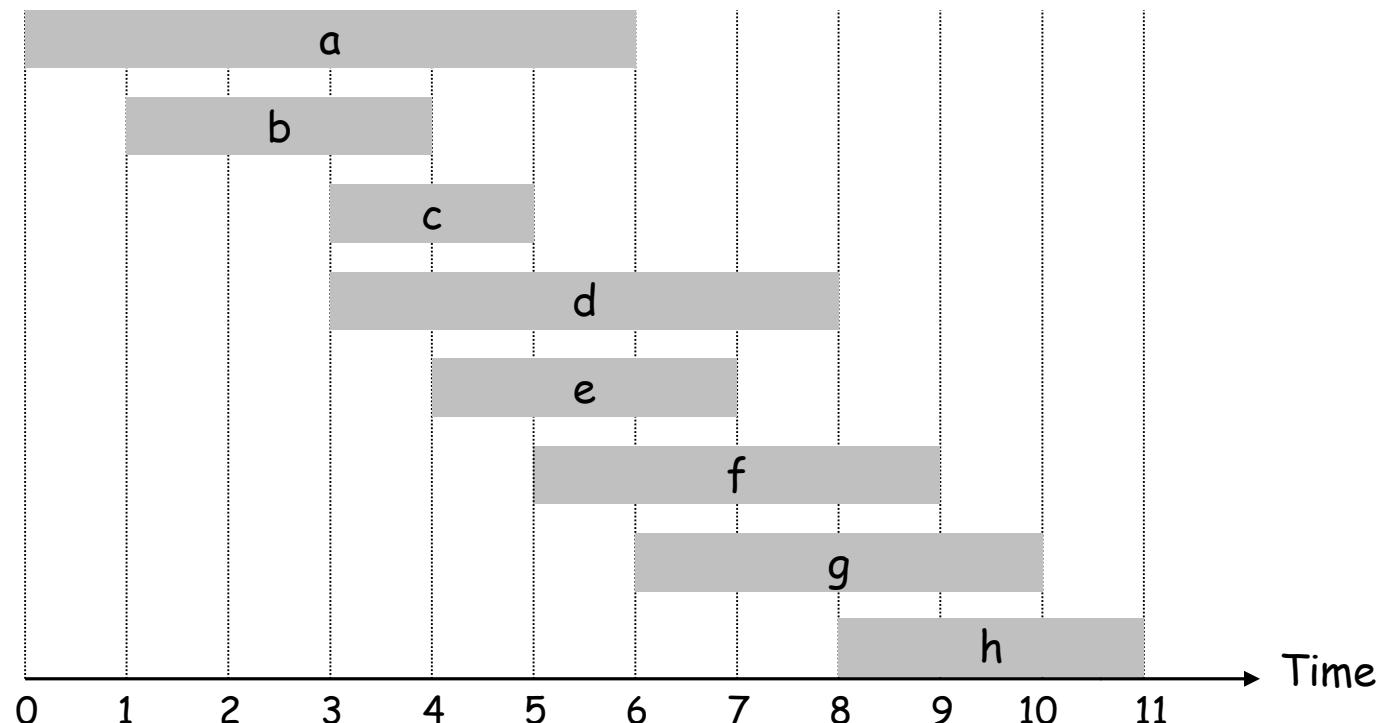
A greedy algorithm is an algorithm that **makes the locally optimal choice** at each step.

4.1 Interval Scheduling

Interval Scheduling

Interval scheduling.

- Job j starts at s_j and finishes at f_j .
- Two jobs **compatible** if they don't overlap.
- Goal: find maximum subset of mutually compatible jobs.



Interval Scheduling: Greedy Algorithms

Greedy template. Consider jobs in some order. Take each job provided it's compatible with the ones already taken.

take or not

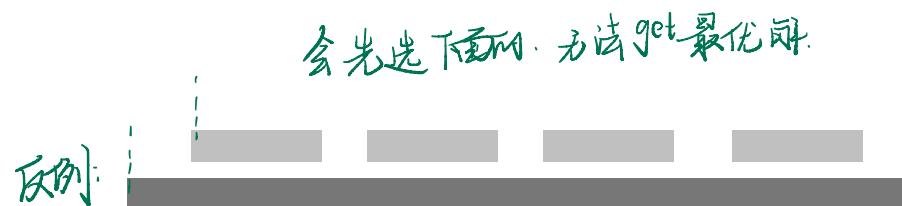
和前面无冲突就选

- [Earliest start time] Consider jobs in ascending order of start time s_j .
Sort by start time
- [Earliest finish time] Consider jobs in ascending order of finish time f_j .
- [Shortest interval] Consider jobs in ascending order of interval length $f_j - s_j$.
Sort by job duration
- [Fewest conflicts] For each job, count the number of conflicting jobs c_j . Schedule in ascending order of conflicts c_j .
冲突数

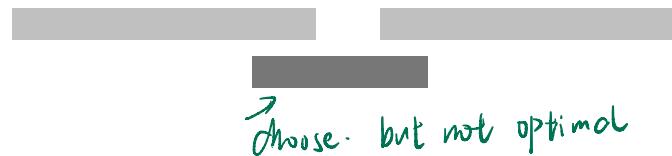
Interval Scheduling: Greedy Algorithms

Greedy template. Consider jobs in some order. Take each job provided it's compatible with the ones already taken.

earliest start time?



shortest interval?



fewest conflicts?



Interval Scheduling: Greedy Algorithm

Greedy algorithm. Consider jobs in increasing order of finish time.
Take each job provided it's compatible with the ones already taken.

Sort jobs by finish times so that $f_1 \leq f_2 \leq \dots \leq f_n$.

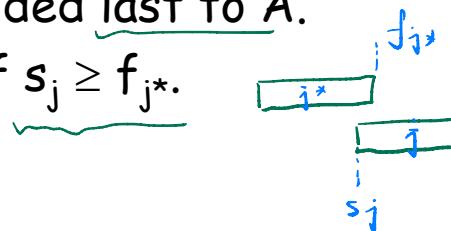
jobs selected
 \leftarrow
 $A \leftarrow \emptyset$
for $j = 1$ to n {
 if (job j compatible with A)
 $A \leftarrow A \cup \{j\}$
}
return A

Sort need $O(n \log n)$

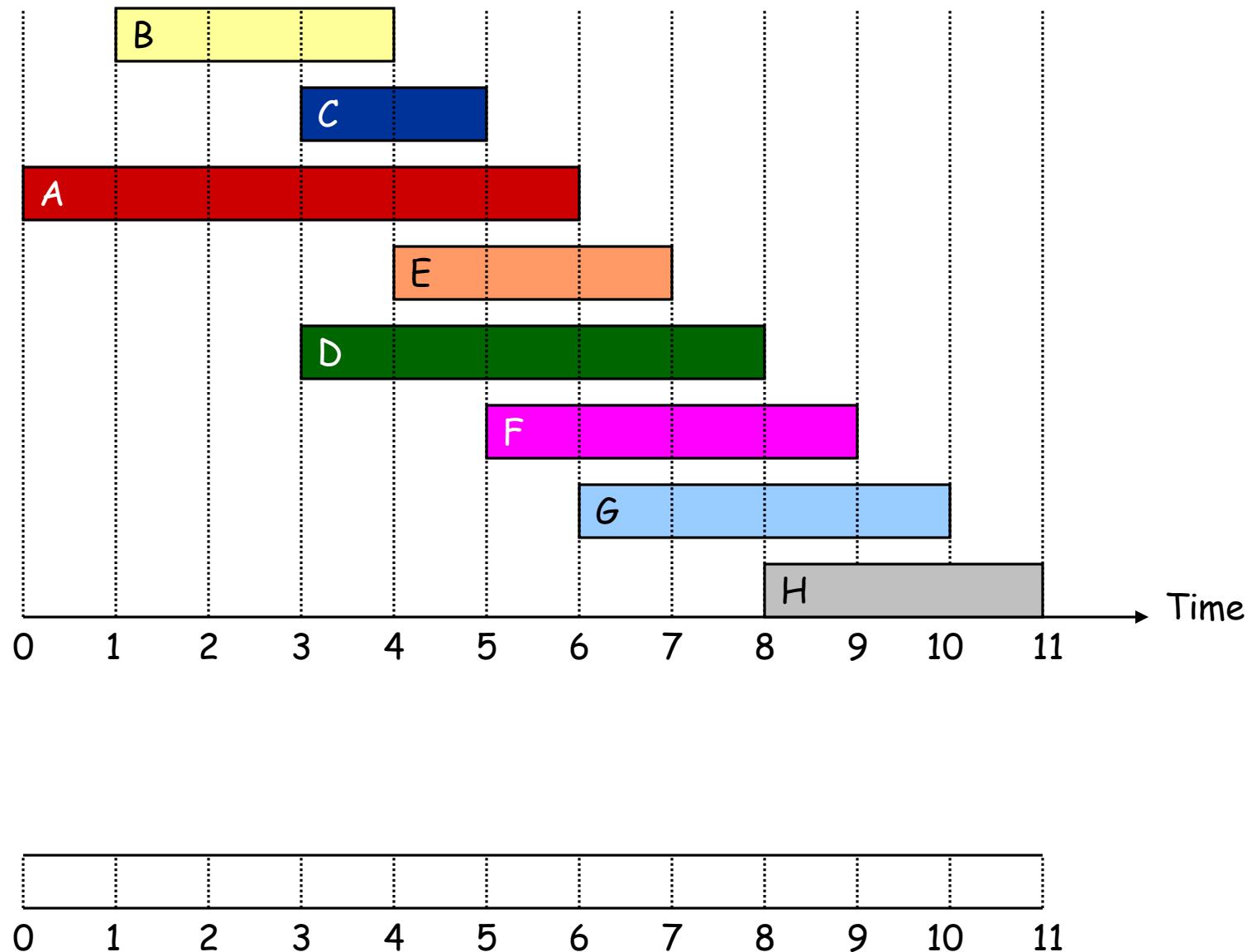
Implementation. $O(n \log n)$.

j^* - \rightarrow job added last to A .

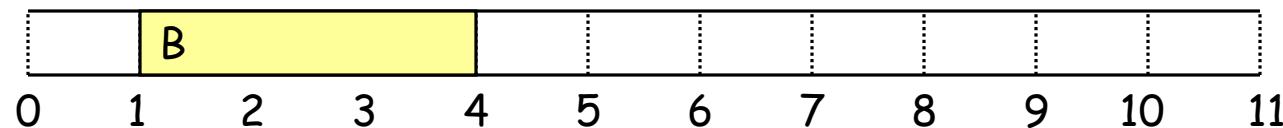
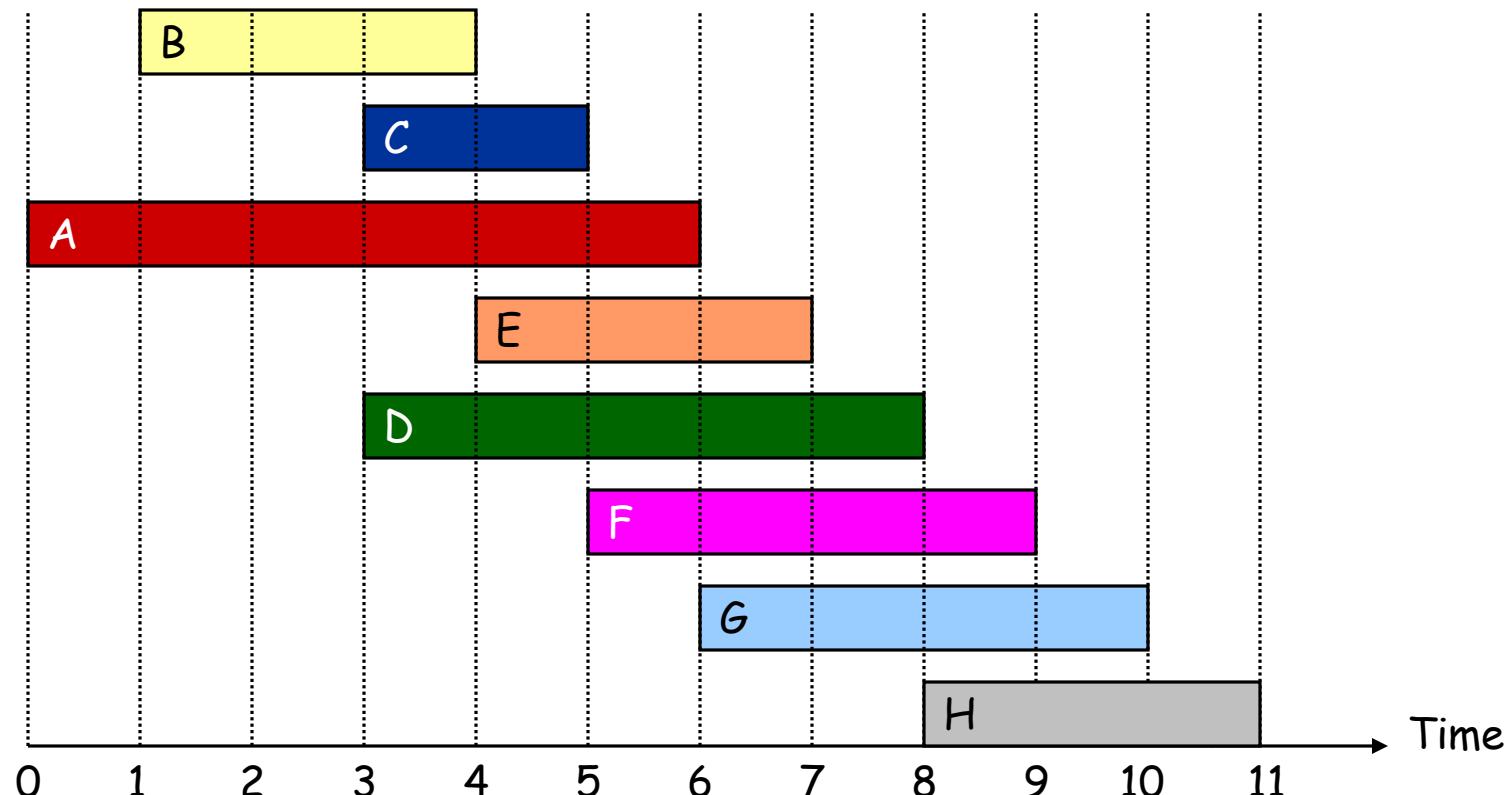
- Remember job j^* that was added last to A .
- Job j is compatible with A iff $s_j \geq f_{j^*}$.



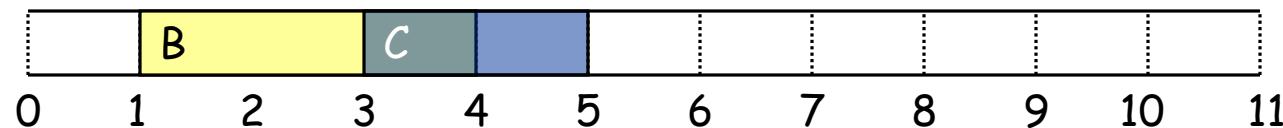
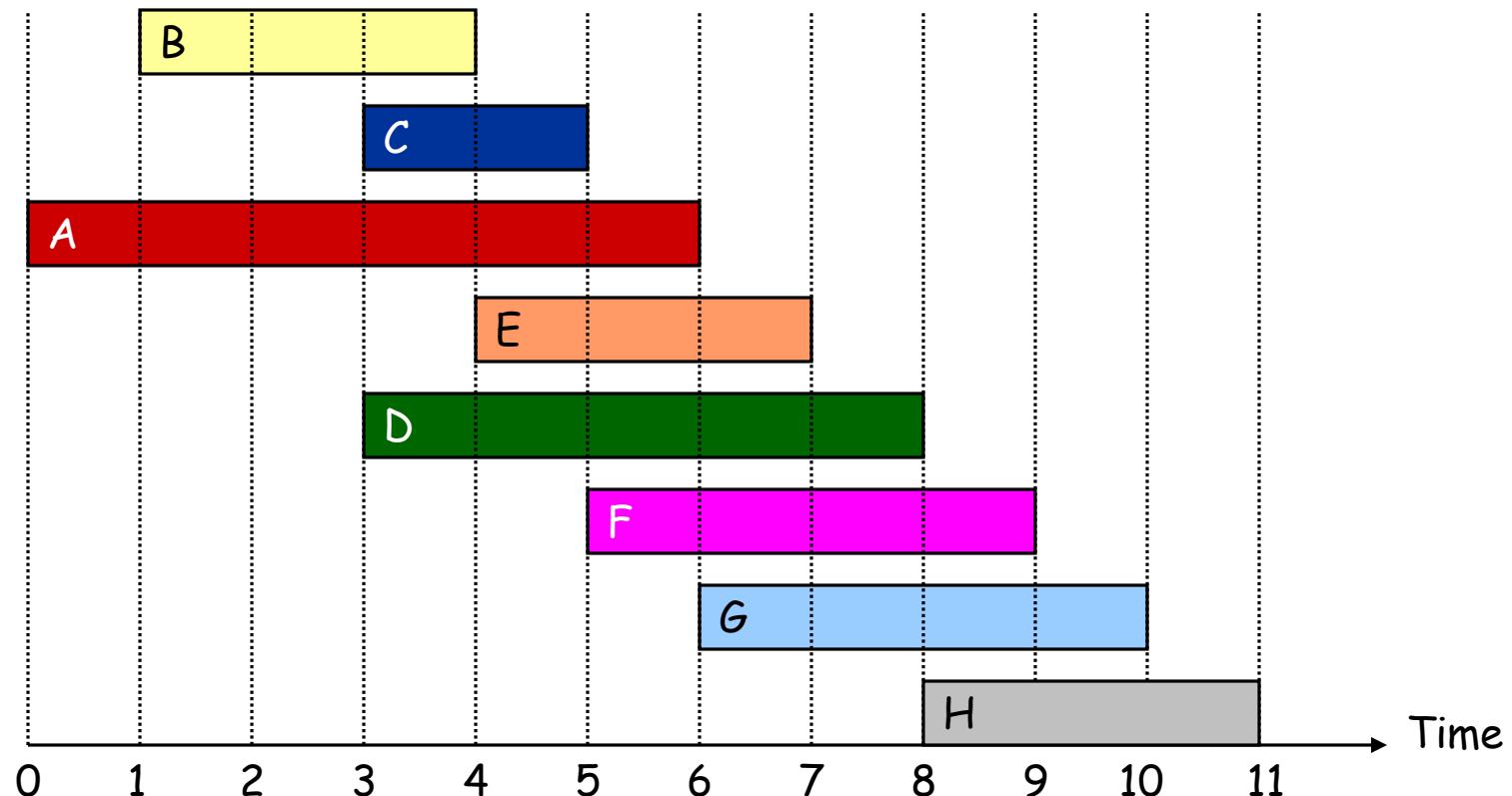
Interval Scheduling



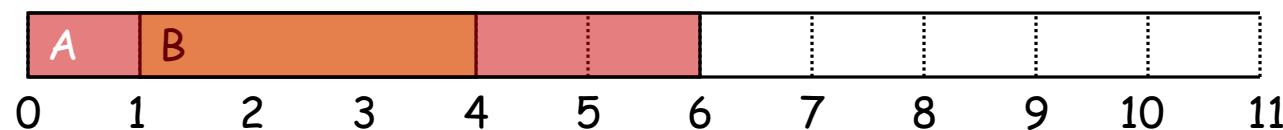
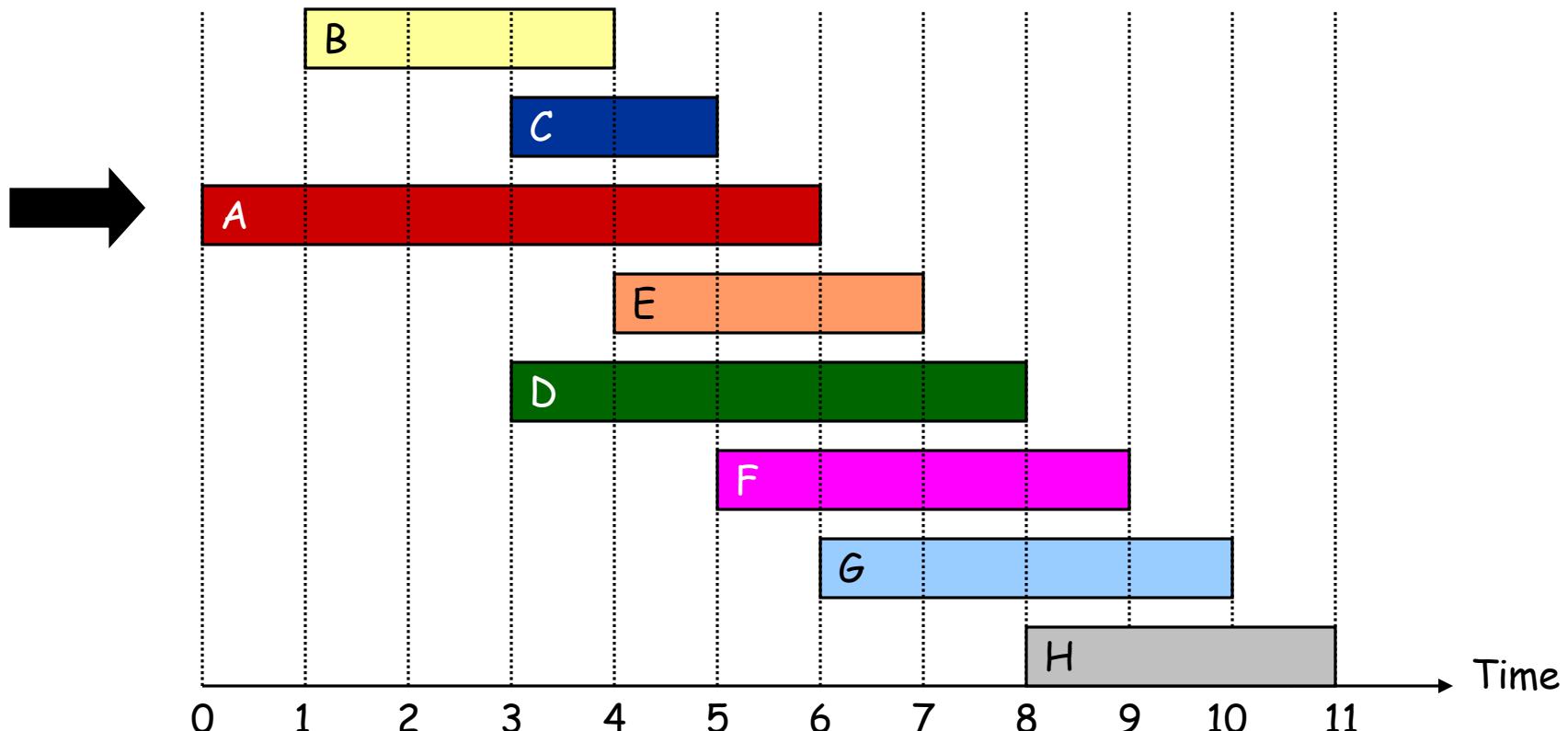
Interval Scheduling



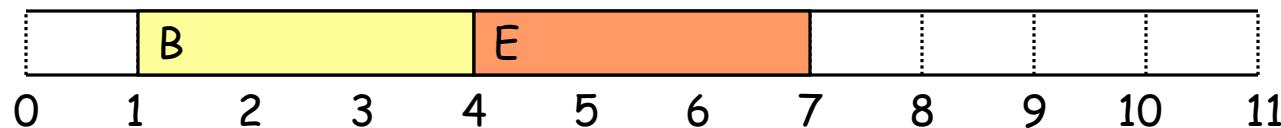
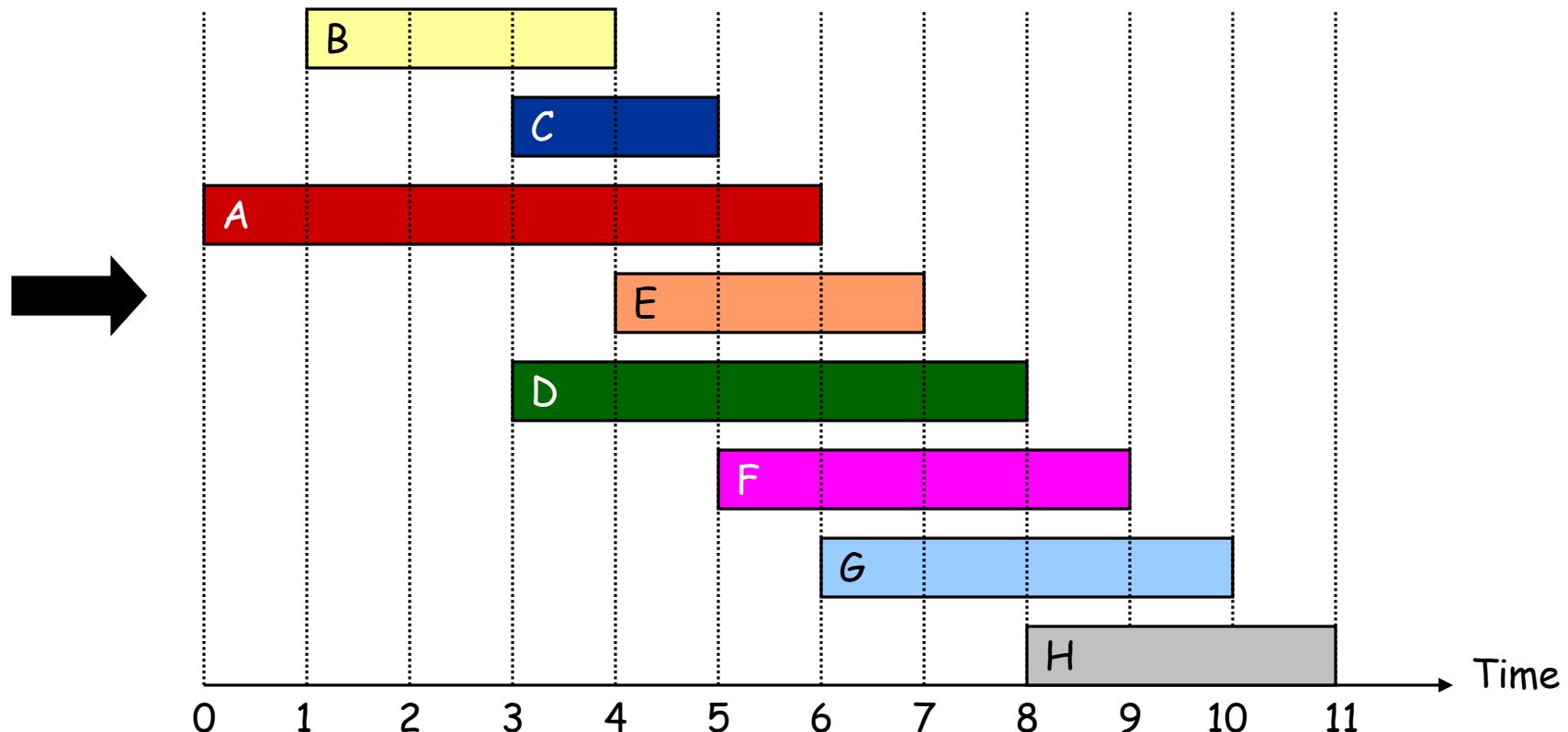
Interval Scheduling



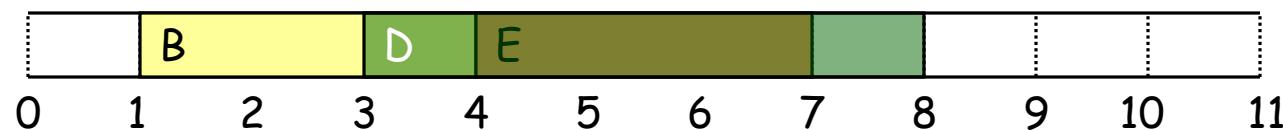
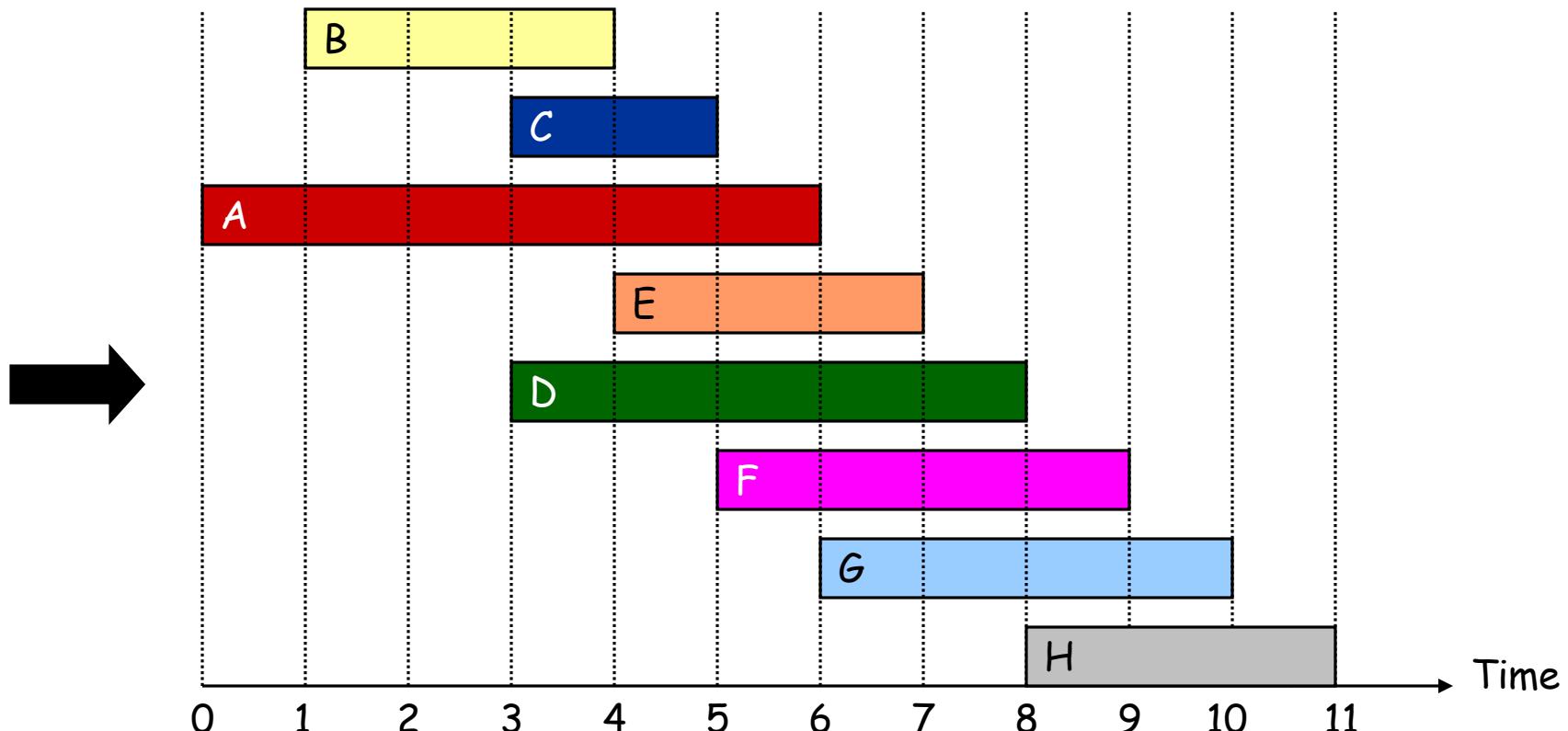
Interval Scheduling



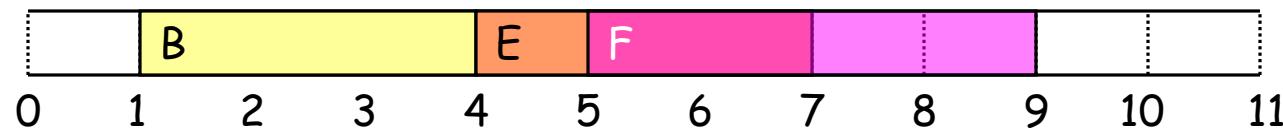
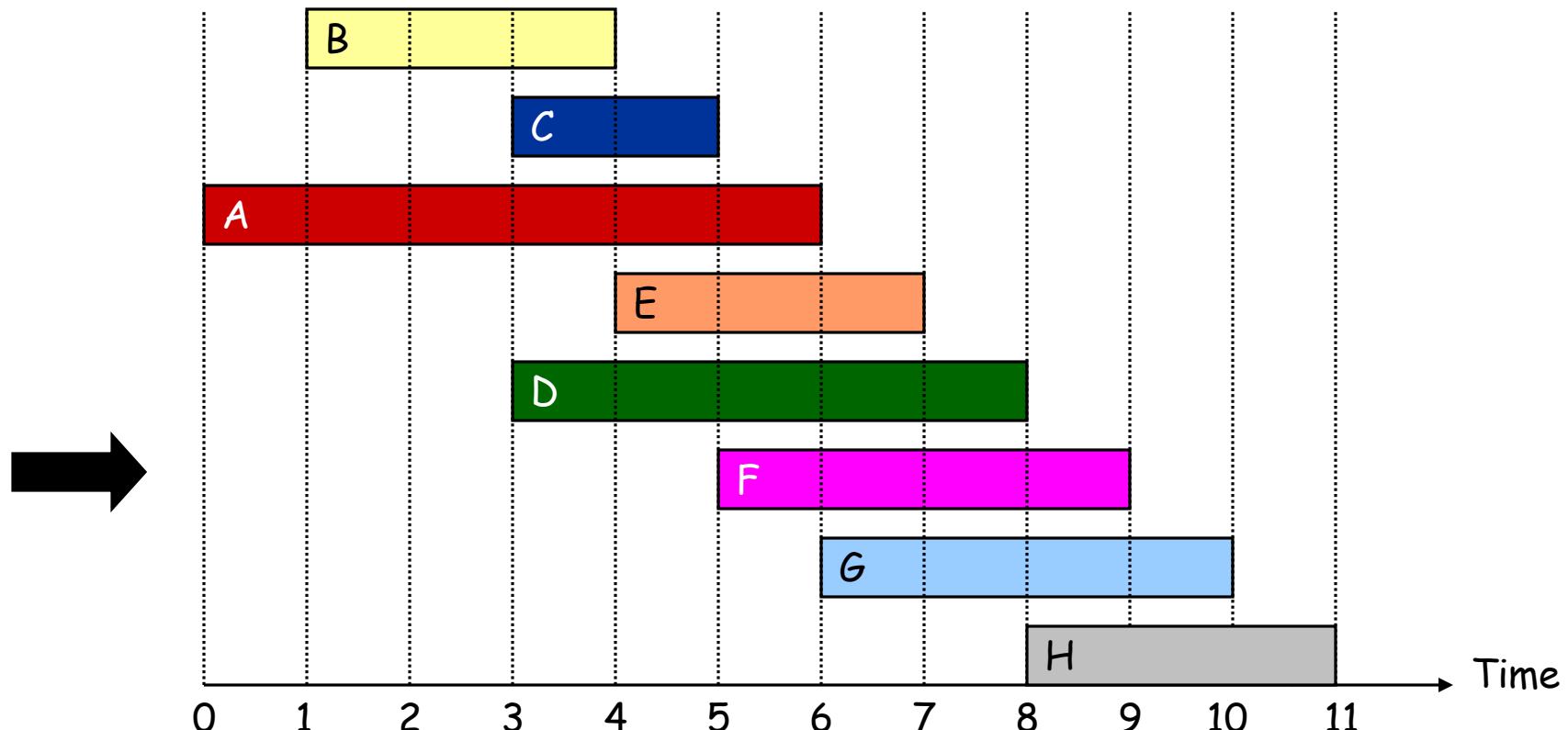
Interval Scheduling



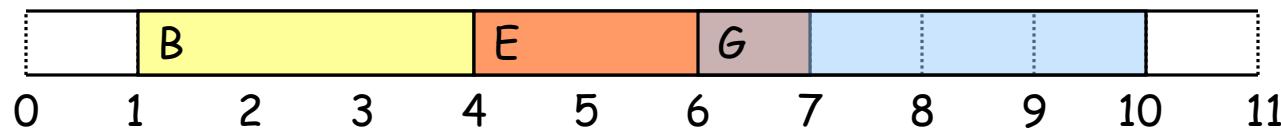
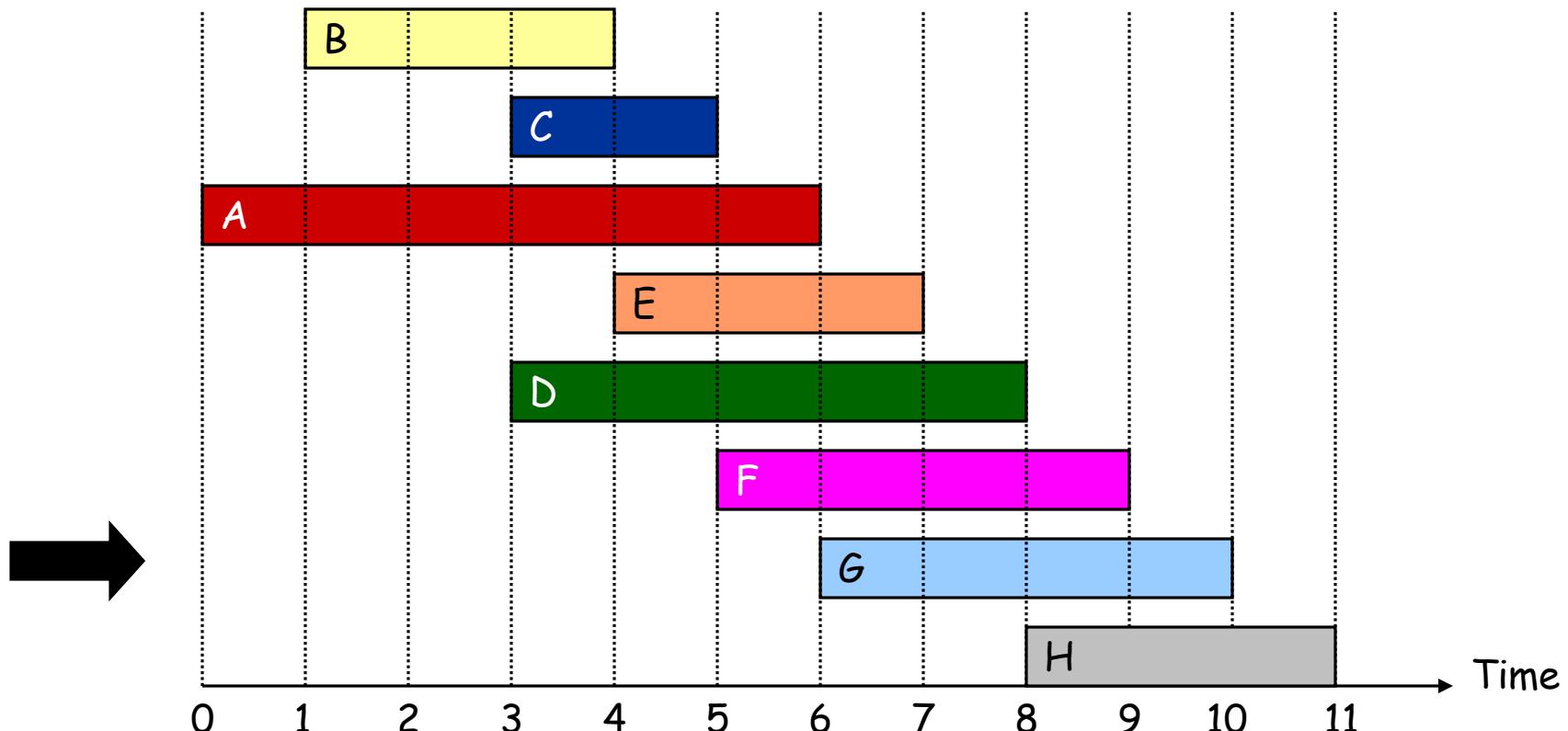
Interval Scheduling



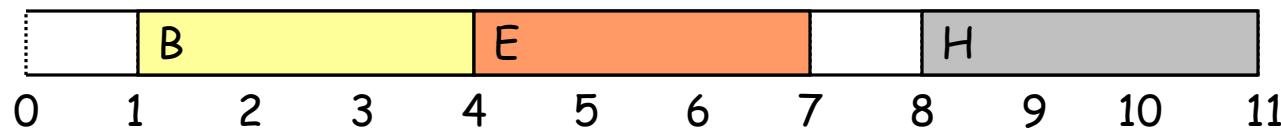
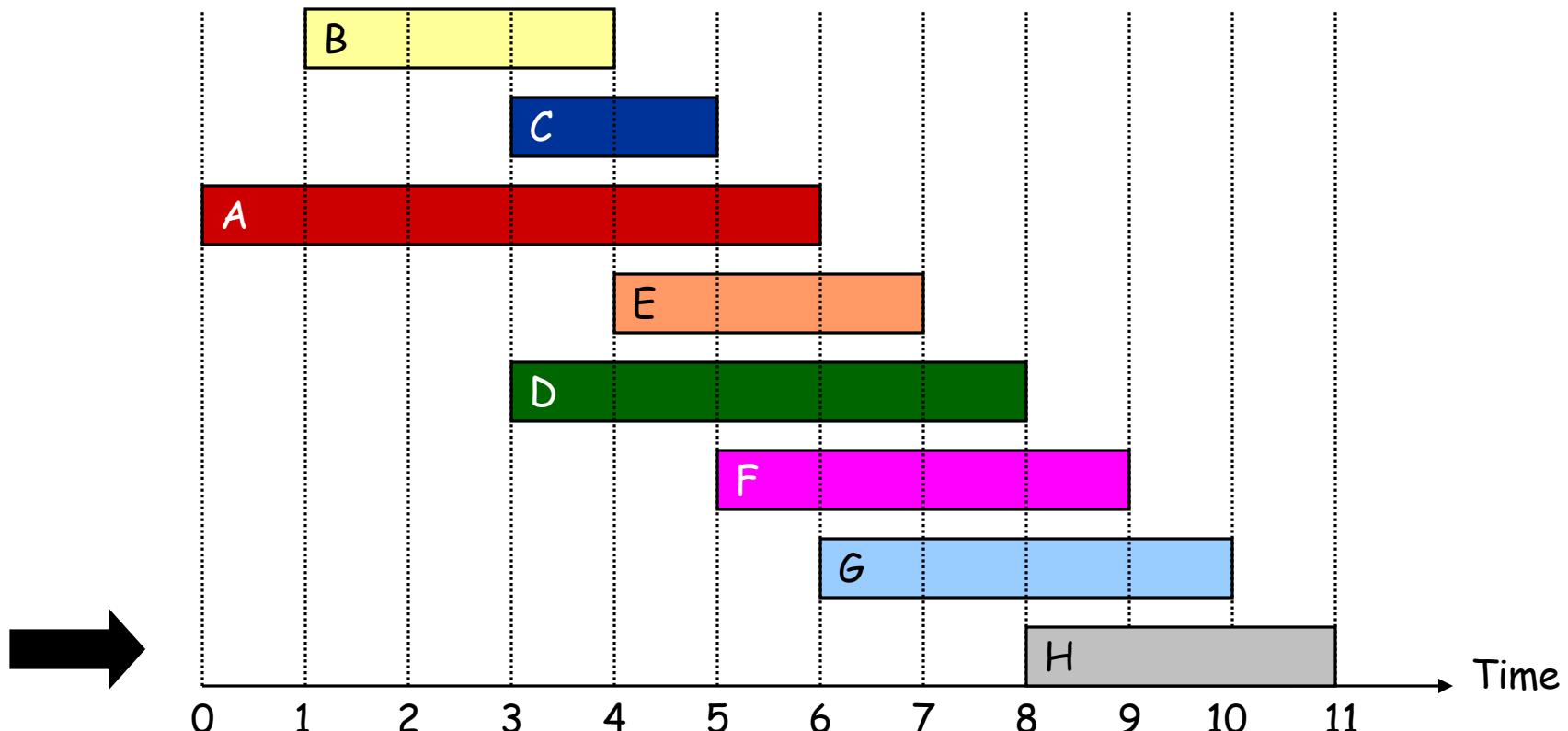
Interval Scheduling



Interval Scheduling



Interval Scheduling



Interval Scheduling: Analysis

Theorem. Greedy algorithm is optimal. 证明

Pf. (by contradiction) 反证法

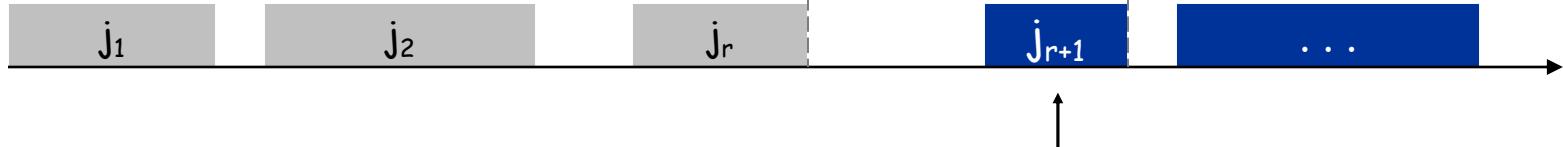
- Assume greedy is not optimal, and let's see what happens.
- Let i_1, i_2, \dots, i_k denote set of jobs selected by greedy.
- Let j_1, j_2, \dots, j_m denote set of jobs in the optimal solution with $i_1 = j_1, i_2 = j_2, \dots, i_r = j_r$ for the largest possible value of r .

i, j, 前一个选择 same. many opt. 设计序列 在所有opt中, r为r_{max}.

Greedy:



OPT:



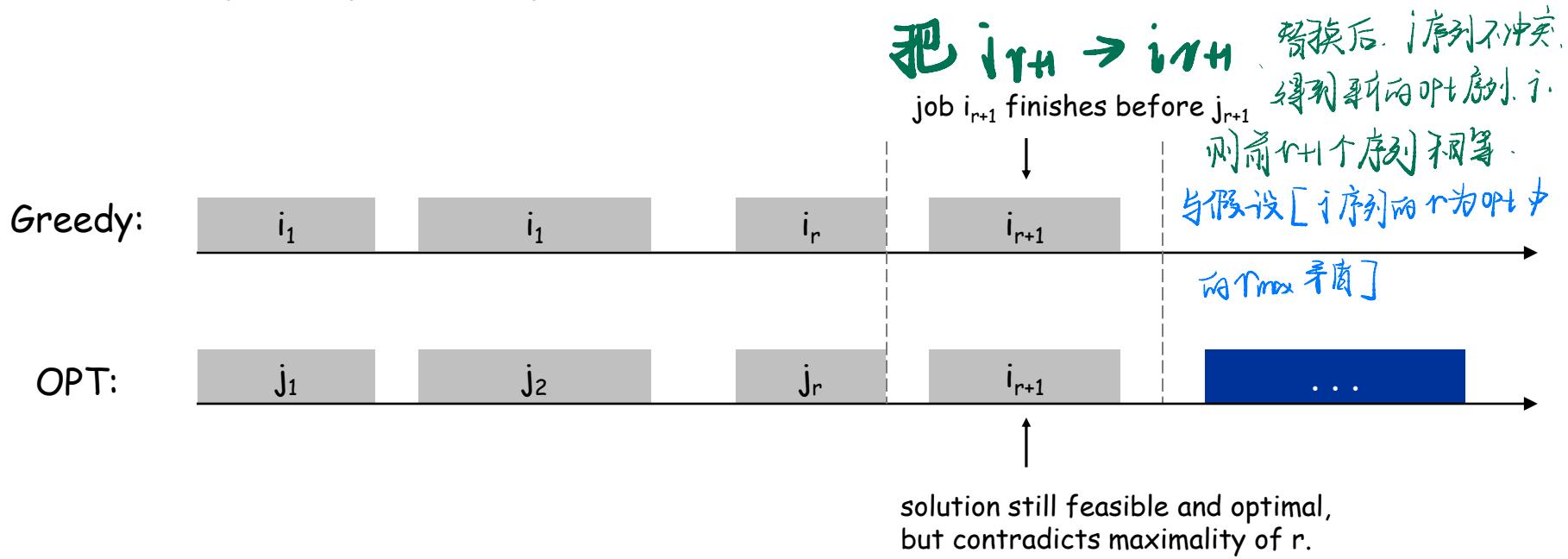
why not replace job j_{r+1} with job i_{r+1} ?

Interval Scheduling: Analysis

Theorem. Greedy algorithm is optimal.

Pf. (by contradiction)

- Assume greedy is not optimal, and let's see what happens.
- Let i_1, i_2, \dots, i_k denote set of jobs selected by greedy.
- Let j_1, j_2, \dots, j_m denote set of jobs in the optimal solution with $i_1 = j_1, i_2 = j_2, \dots, i_r = j_r$ for the largest possible value of r .

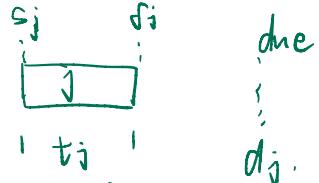


4.2 Scheduling to Minimize Lateness

Scheduling to Minimizing Lateness

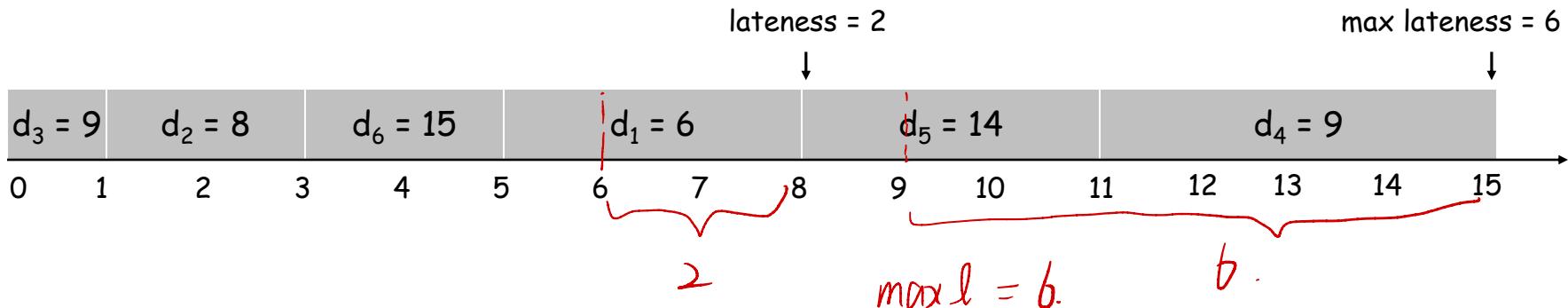
Minimizing lateness problem.

- Single resource processes one job at a time.
- Job j requires t_j units of processing time and is due at time d_j .
- If j starts at time s_j , it finishes at time $f_j = s_j + t_j$.
返回逾期了
- Lateness: $\ell_j = \max \{ 0, f_j - d_j \}$.
- Goal: schedule all jobs to minimize maximum lateness $L = \max \ell_j$.



Ex:

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15



Minimizing Lateness: Greedy Algorithms

Greedy template. Consider jobs in some order.

- [Shortest processing time first] Consider jobs in ascending order of processing time t_j .
- [Earliest deadline first] Consider jobs in ascending order of deadline d_j .
- [Smallest slack] Consider jobs in ascending order of slack $d_j - t_j$.
slack. 最晚多久 start · 可保证不超时 ·

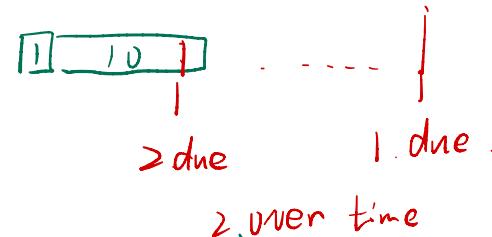
Minimizing Lateness: Greedy Algorithms

Greedy template. Consider jobs in some order.

- [Shortest processing time first] Consider jobs in ascending order of processing time t_j .

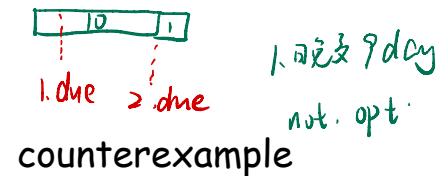
	1	2
t_j	1	10
d_j	100	10

counterexample



- [Smallest slack] Consider jobs in ascending order of slack $d_j - t_j$.

	1	2
t_j	1	10
d_j	2	10



counterexample

Minimizing Lateness: Greedy Algorithm

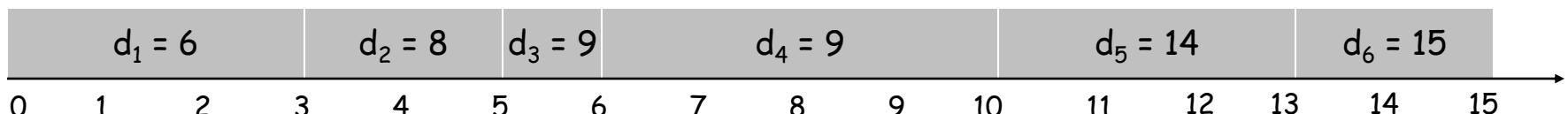
Greedy algorithm. Earliest deadline first.

```
Sort n jobs by deadline so that  $d_1 \leq d_2 \leq \dots \leq d_n$ 

 $t \leftarrow 0$ 
for  $j = 1$  to  $n$ 
    Assign job  $j$  to interval  $[t, t + t_j]$ 
     $s_j \leftarrow t, f_j \leftarrow t + t_j$ 
     $t \leftarrow t + t_j$ 
output intervals  $[s_j, f_j]$ 
```

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15

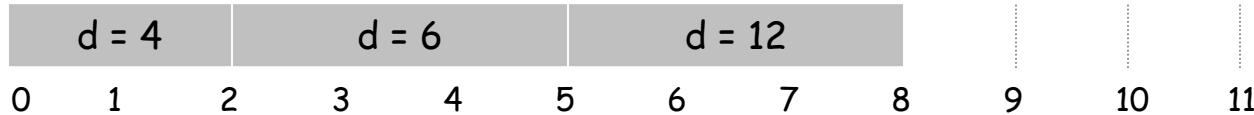
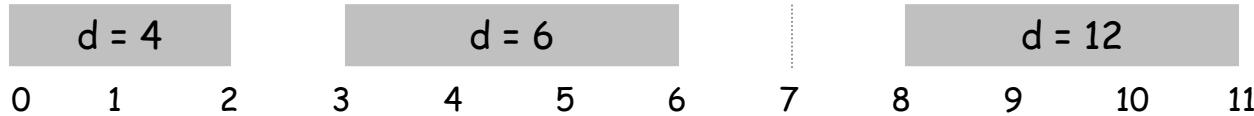
max lateness = 1



Minimizing Lateness: No Idle Time

存在 opt. 为 空闲时间

Observation. There exists an optimal schedule with no **idle time**.



Observation. The greedy schedule has no idle time.

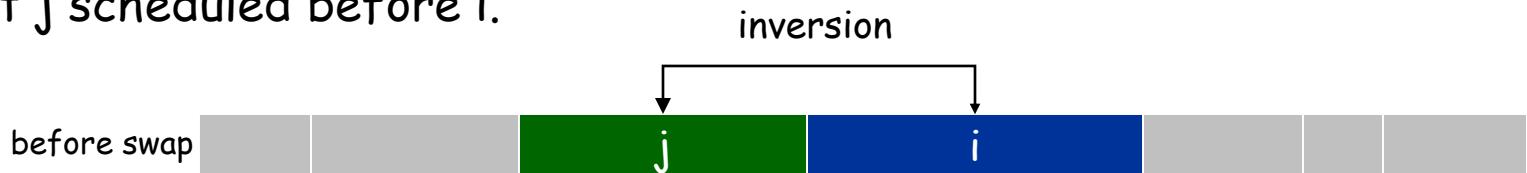
Minimizing Lateness: Inversions

→ 把最晚的 job 先做.

Def. An inversion in schedule S is a pair of jobs i and j such that:

$i < j$ but j scheduled before i .

$d_j < d_i$



Observation. Greedy schedule has no inversions.

Observation. If a schedule (with no idle time) has an inversion, it has one with a pair of inverted jobs scheduled consecutively.

Minimizing Lateness: Inversions

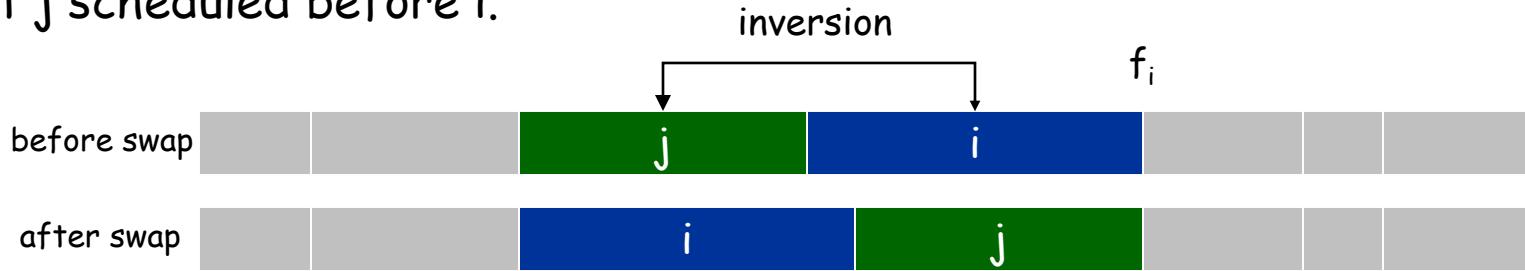
如果有 inversion. 一定有相邻 job. inversion.

证: if. $d_i < d_{i+1}$.

$d_{i+1} < d_{i+2} \dots$

那么就有 inversion.

Def. An **inversion** in schedule S is a pair of jobs i and j such that: $i < j$ but j scheduled before i .



Step: \exists opt no idle time \Rightarrow if \exists inversion. \rightarrow 一定有相邻 job. inversion \rightarrow 到这一步的证明 f'_j

Claim. Swapping two adjacent, inverted jobs reduces the number of inversions by one and does not increase the max lateness.

Pf. Let ℓ be the lateness before the swap, and let ℓ' be it afterwards.

- $\ell'_k = \ell_k$ for all $k \neq i, j$
- $\ell'_i \leq \ell_i$ i 被前 done. lateness 可能 \downarrow or 不变.
- If job j is late:

1. 交换后

$$i < j \Rightarrow d_i < d_j$$

$$\begin{aligned}\ell'_j &= f'_j - d_j && (\text{definition}) \\ &= f_i - d_j && (j \text{ finishes at time } f_i) \\ &\leq f_i - d_i && (i < j) \\ &\leq \ell_i && (\text{definition})\end{aligned}$$

Minimizing Lateness: Analysis of Greedy Algorithm

其他 job not change.

$$\therefore f'_j \leq f_{ij} \leq f_{\max} \therefore f_{\max} \text{ won't increase}$$

Theorem. Greedy schedule S is optimal.

Pf. Define S^* to be an optimal schedule that has the fewest number of inversions, and let's see what happens.

- Can assume S^* has no idle time.
- If S^* has no inversions, then $S = S^*$.
- If S^* has an inversion, let $i-j$ be an adjacent inversion.
- swapping i and j does not increase the maximum lateness and strictly decreases the number of inversions
- this contradicts definition of S^* .

设 S^* 为 opt 且 inversion \Rightarrow ①

那么, 调换 inversion 时 f_{\max} 不变.

那么, f_{\max} 不变, 且 inversion 减少.

与①矛盾.

∴ greedy is opt.

Greedy Analysis Strategies

Greedy algorithm stays ahead. Show that after each step of the greedy algorithm, its solution is at least as good as any other algorithm's.

Exchange argument. Gradually transform any solution to the one found by the greedy algorithm without hurting its quality. ←

将 solution 转换为 greedy solution. 不损害质量

4.3 Optimal Caching

Optimal Offline Caching

Caching.

- Cache with capacity to store k items.
- Sequence of m item requests d_1, d_2, \dots, d_m .
- Cache hit: item already in cache when requested.
- Cache miss: item not already in cache when requested: must bring requested item into cache, and evict some existing item, if full.

Goal. Eviction schedule that minimizes number of evictions.

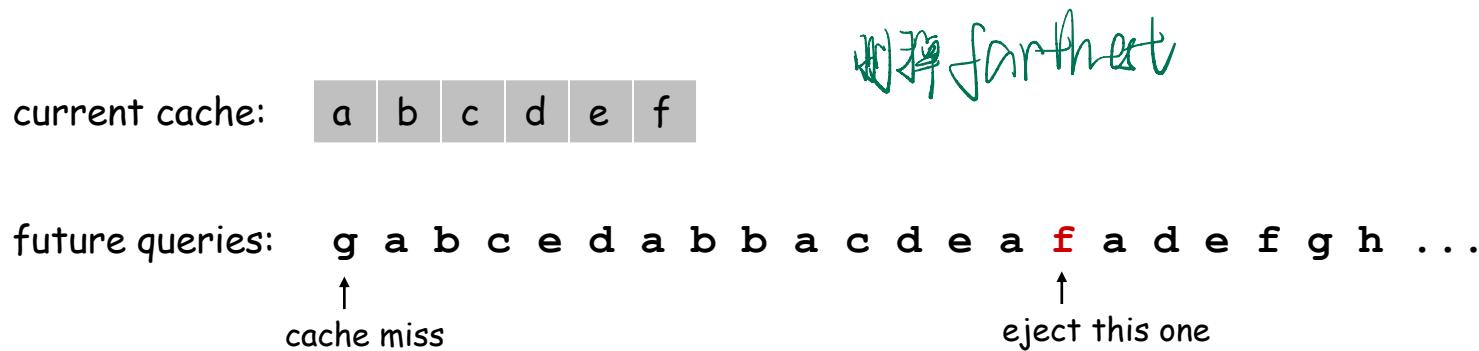
Ex: $k = 2$, initial cache = ab,
requests: a, b, c, b, c, a, a, b.

Optimal eviction schedule: 2 evictions.

requests	cache
a	a b
b	a b
c	c b
b	c b
c	c b
a	a b
a	a b
b	a b

Optimal Offline Caching: Farthest-In-Future

Farthest-in-future. Evict item in the cache that is not requested until farthest in the future.



Theorem. [Bellady, 1960s] FF is optimal eviction schedule.

Pf. Algorithm and theorem are intuitive; proof is subtle.

Reduced Eviction Schedules

Def. A **reduced** schedule is a schedule that only inserts an item into the cache in a step in which that item is requested.

Intuition. Can transform an unreduced schedule into a reduced one with no more evictions.

a	a	b	c
a	a	x	c
c	a	d	c
d	a	d	b
a	a	c	b
b	a	x	b
c	a	c	b
a	a	b	c
a	a	b	c

read a . delete b.
没必要. unreduced

a	a	b	c
a	a	b	c
c	a	b	c
d	a	d	c
a	a	d	c
b	a	d	b
c	a	c	b
a	a	c	b
a	a	c	b

an unreduced schedule

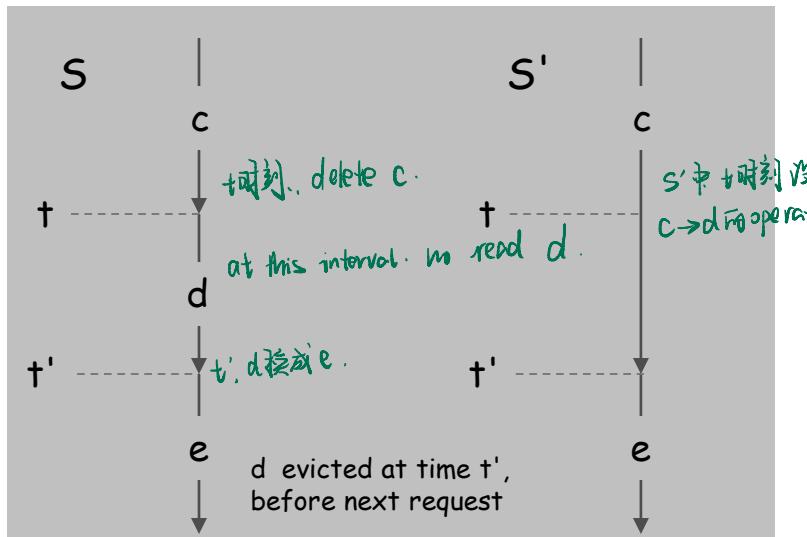
a reduced schedule

Reduced Eviction Schedules

Claim. Given any **unreduced schedule S** , can transform it into a reduced schedule S' with no more evictions.

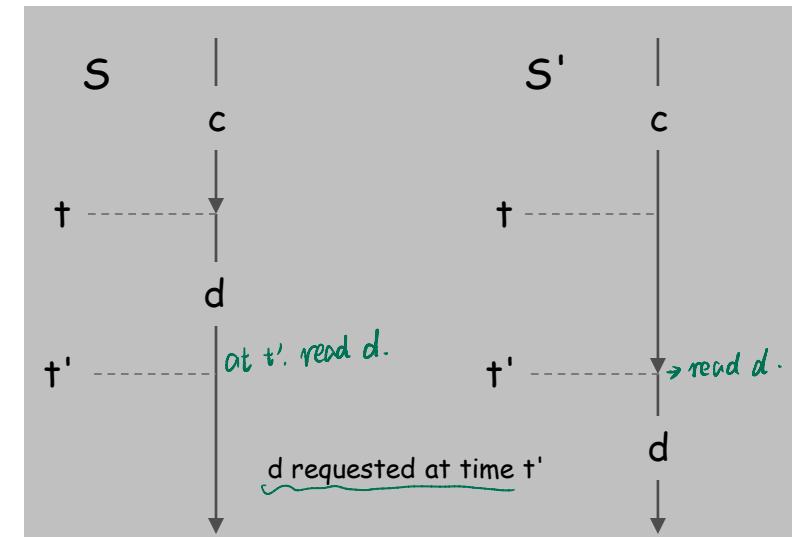
Pf. (by induction on number of unreduced items) ← doesn't enter cache at requested time

- Suppose S brings d into the cache at time t , without a request.
- Let c be the item S evicts when it brings d into the cache.
- Case 1: d evicted at time t' , before next request for d .
- Case 2: d requested at time t' before d is evicted.



Case 1

结论: 可把 unreduced S' 转为 reduced S .
而 eviction number 不增加



Case 2

Farthest-In-Future: Analysis

Lemma. Let S be a reduced schedule that makes the same schedule as S_{FF} through the first j requests. Then there is a reduced schedule S' that makes the same schedule as S_{FF} through the first $j+1$ requests, and incurs no more evictions than S does.

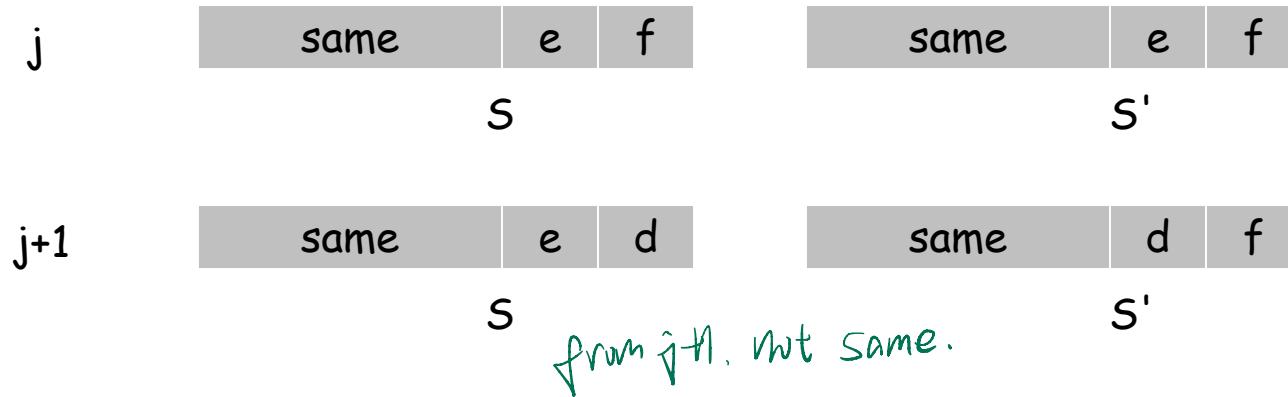
Pf.

- Consider $(j+1)^{st}$ request $d = d_{j+1}$.
- Since S and S_{FF} have agreed up until now, they have the same cache contents before request $j+1$. \Rightarrow their behavior is the same.
- Case 1: (d is already in the cache). $S' = S$
- Case 2: (d is not in the cache and S and S_{FF} evict the same element).
 $S' = S$ case2: d 不在 cache \nexists . S, S_{FF} delete same element
 $\Rightarrow S = S'$

Farthest-In-Future: Analysis

Pf. (continued)

- Case 3: (d is not in the cache; S_{FF} evicts e; S evicts $f \neq e$).
 - begin construction of S' from S by evicting e instead of f



- now S' agrees with S_{FF} on first $j+1$ requests
- From request $j+2$ onward, we make S' the same as S , but this becomes impossible when e or f is involved

Farthest-In-Future: Analysis

Pf. (continued)

Let j' be the **first** time after $j+1$ that S and S' take a different action, and let g be item requested at time j' .
must involve e or f (or both)

j'	same	e	same	f
------	------	---	------	---

S
 $S \neq S'$ 在 j' first not same \Rightarrow this case must 涉及到 e,f

- Case 3a: $g = e$. Can't happen with Farthest-In-Future since there must be a request for f before e.

j' 时刻 不可能读 e. \because for greedy, e is farthest.
before j' , 没有 read f.

$\Rightarrow f$ farther than e.
与 greedy 原则矛盾

Farthest-In-Future: Analysis

Pf. (continued)

Let j' be the **first** time after $j+1$ that S and S' take a different action, and let g be item requested at time j' .

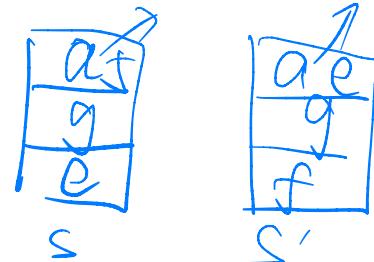


otherwise S' would take the same action

- Case 3b: $g \neq e, f$. S must evict e .
Make S' evict f ; now S and S' have the same cache.



Farthest-In-Future: Analysis



Pf. (continued)

Let j' be the **first** time after $j+1$ that S and S' take a different action, and let g be item requested at time j' .

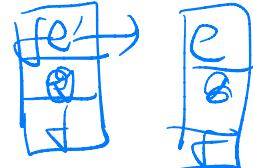
j'



$S \rightarrow \text{evict } e$



$S' \rightarrow \text{not } e$



- Case 3c: $g = f$. Element f can't be in cache of S , so let e' be the element that S evicts. $\Rightarrow S \xrightarrow{\text{evict } e'} f$

且 mW. S' evict < S evict num

- if $e' = e$, S' accesses f from cache; now S and S' have same cache
- if $e' \neq e$, S' evicts e' and brings e into the cache; now S and S' have the same cache. S' is no longer reduced, but can be transformed into a reduced schedule which
 - agrees with S_{FF} through step $j+1$
 - has no more evictions than S

$S' \rightarrow \text{reduced. 有 } \text{some part not change.}$

$$\textcircled{1} \quad \begin{array}{|c|c|} \hline \text{same} & e \\ \hline \end{array} \rightarrow \begin{array}{|c|c|} \hline \text{same} & f \\ \hline \end{array} = S' \quad \textcircled{2}$$

Farthest-In-Future: Analysis

Theorem. FF is optimal eviction algorithm.

Pf. (by induction on number of requests j)

Base case (trivial):

There exists an optimal reduced schedule S that makes the same schedule as S_{FF} through the first 0 requests.

farthes-in future

Inductive step (implied by the lemma): 顺序办法:

If there exists an optimal reduced schedule S that agrees with S_{FF} through the first j requests,

then there exists an optimal reduced schedule S' that agrees with S_{FF} through the first $j+1$ requests

Caching Perspective

Online vs. offline algorithms.

- Offline: full sequence of requests is known a priori.
- Online (reality): requests are not known in advance.
- Caching is among most fundamental online problems in CS.

LIFO. Evict page brought in most recently.

LRU. Evict page whose most recent access was earliest.

↑
FF with direction of time reversed!

Theorem. FF is optimal offline eviction algorithm.

- Provides basis for understanding and analyzing online algorithms.
- LRU is k-competitive. [Section 13.8]
- LIFO is arbitrarily bad.

4.7 Clustering

Clustering

photos, documents, micro-organisms



Clustering. Given a set U of n objects labeled p_1, \dots, p_n , partition into clusters s.t. objects in different clusters are far apart.

e.g., a large number of corresponding pixels whose intensities differ over some threshold



Fundamental problem. Divide into clusters so that points in different clusters are far apart.

- Routing in mobile ad hoc networks.
- Identify patterns in gene expression.
- Document categorization for web search.
- Similarity searching in medical image databases
- Skycat: cluster 10^9 sky objects into stars, quasars, galaxies.

Clustering of Maximum Spacing

k-clustering. Divide objects into k non-empty groups.

Distance function. Assume it satisfies several natural properties.

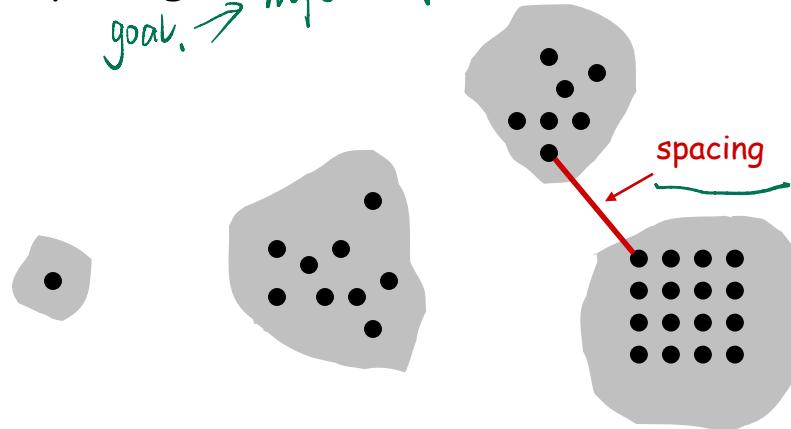
- $d(p_i, p_j) = 0$ iff $p_i = p_j$ (identity of indiscernibles)
- $d(p_i, p_j) \geq 0$ (nonnegativity)
- $d(p_i, p_j) = d(p_j, p_i)$ (symmetry)

不同 cluster 間的最短距離 min distance

Spacing. Min distance between any pair of points in different clusters.
间隔.

Clustering of maximum spacing. Given an integer k, find a k-clustering of maximum spacing.

goal. → hope to ~~最大化~~ space



$$k = 4$$

Greedy Clustering Algorithm

Single-link k-clustering algorithm. 一开始创建 n clusters, each point is a cluster

- Create n clusters, one for each object.
- Find the closest pair of objects such that each object is in a different cluster; add an edge between them and merge the two clusters. 我们两个点，距离最短 \rightarrow space \downarrow means 合并了这两个 cluster.
- Repeat $n-k$ times until there are exactly k clusters.
 $n-k$ 次后, 得到 k clusters

Key observation. This procedure is precisely Kruskal's algorithm
(except we stop when there are k connected components).

cluster 对应 Kruskal 中的连通分量

Remark. Equivalent to finding an MST and deleting the $k-1$ most expensive edges. single-link = 删掉 MST 中最长的 $k-1$ 条边

Greedy Clustering Algorithm: Analysis

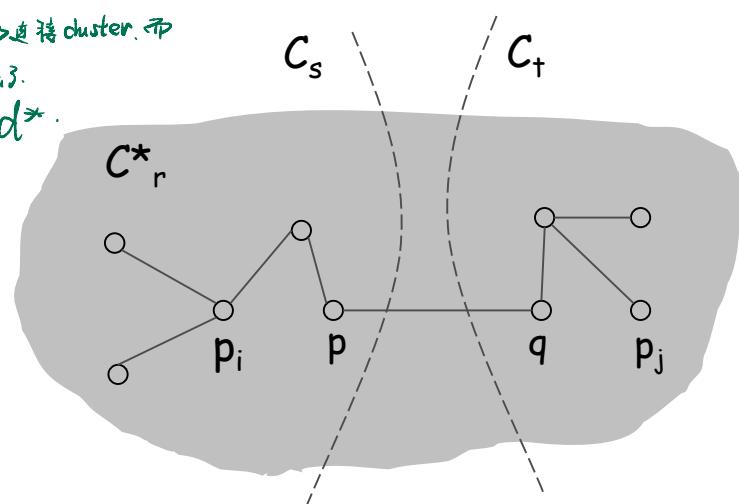
Theorem. Let C^* denote the clustering C^*_1, \dots, C^*_k formed by deleting the $k-1$ most expensive edges of a MST. C^* is a k -clustering of max spacing.

Pf. Let C denote some other clustering C_1, \dots, C_k .
↗ another method different with C^*

- The spacing of C^* is the length d^* of the $(k-1)^{st}$ most expensive edge in MST.
SL n-k 次選後, 如果 go on, get the 第 k-1 条最長. 這也就是 spacing
single-linked ∵ SL 7. add edge. 抽短列表
- Let p_i, p_j be in the same cluster in C^* , say C^*_r , but different clusters in C , say C_s and C_t .
→ 在 C^* 中 分屬 cluster C_s, C_t ; 在 C 中 屬 same cluster
- Some edge (p, q) on p_i-p_j path in C^*_r spans two different clusters in C .
- All edges on p_i-p_j path have length $\leq d^*$
在 C^* 中 下一次會加 d^* 會連接 cluster. 而
since Kruskal chose them. ↗ edge pq 已在之前被加上去了.
- Spacing of C is $\leq d^*$ since p and q are in different clusters.
∴ $pq \leq d^*$.

證明了, 有 $\text{spacing of } C < \text{spacing of } C^*$.

$\Rightarrow C^*$ have max spacing



Greedy Algorithms: Chapter Summary

Greedy Algorithms

Basic idea

- Make the locally optimal choice at each step.

Algorithms

4 examples help to understand greedy

Greedy feature

① Algorithm is easy.

② hard to prove it's right,
it's optimal.

证明的思想很重要。

- Interval Scheduling
 - Choose the job with the earliest finish time
- Scheduling to Minimize Lateness
 - Choose the job with the earliest deadline
- Optimal Caching
 - Evict item that is requested farthest in future
- Clustering
 - Single-link k-clustering