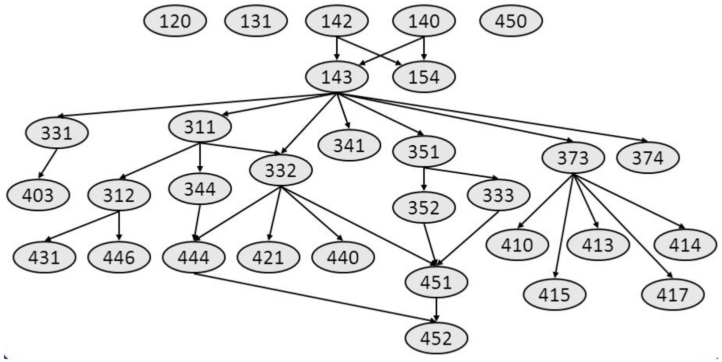
CSE 421

Applications of DFS(?)
Topological sort

Yin Tat Lee

Precedence Constraints

In a directed graph, an edge (i, j) means task i must occur before task j.



Applications

- Course prerequisite:

  course i must be taken before j

- Compilation:

  must compile module i before j

- Computing overflow:

  output of job i is part of input to job j

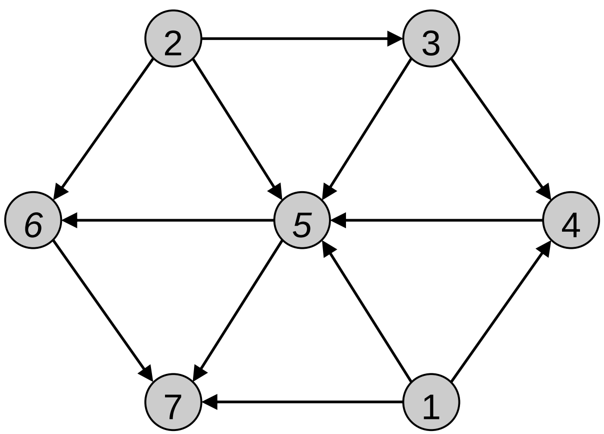
- Manufacturing or assembly:

  sand it before paint it

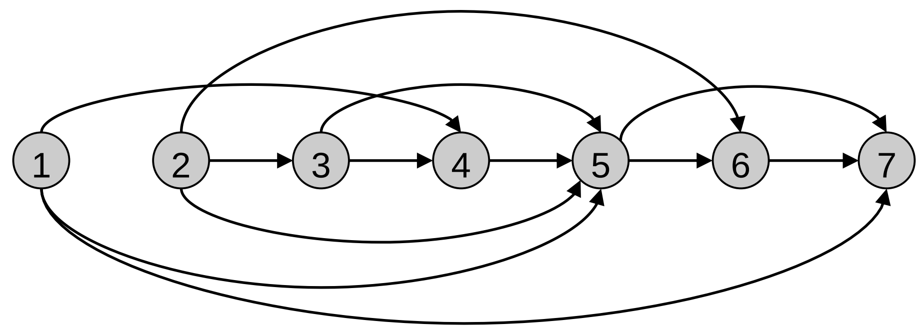
Directed Acyclic Graphs (DAG)

Def: A directed acyclic graph (DAG) is a graph that contains no directed cycles.

Def: A topological order of a directed graph $G = (V, E)$ is an ordering of its nodes as $v_1, v_2, \ldots, v_n$ so that for every edge (v_i, v_j) we have $i < j$.



a DAG



a topological ordering of that DAG– all edges left-to-right

DAGs: A Sufficient Condition

Lemma: If G has a topological order, then G is a DAG.

Proof. (by contradiction)

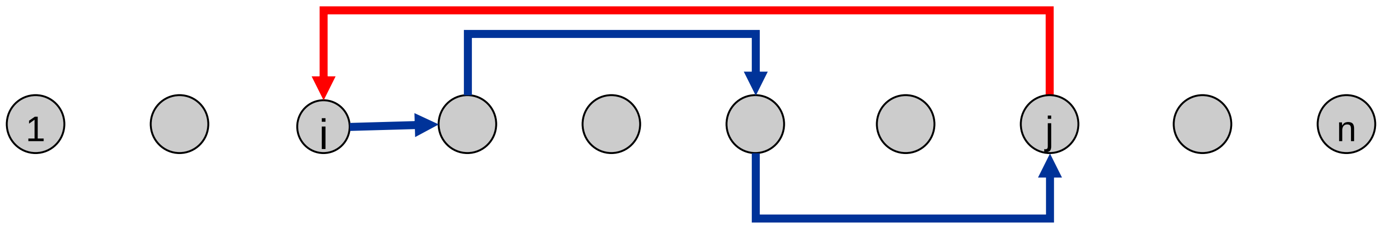
Suppose that G has a topological order $1,2,\ldots,n$ and that G also has a directed cycle C.

Let i be the lowest-indexed node in C, and let j be the node just before i; thus (j,i) is an (directed) edge.

By our choice of i, we have $i < j$.

On the other hand, since (j,i) is an edge and $1,2,\ldots,n$ is a topological order, we must have $j < i$, a contradiction

the directed cycle C



the supposed topological order: $1,2,\ldots,n$

DAGs: A Sufficient Condition

G has a topological order

?

G is a DAG

Every DAG has a source node

Lemma: If G is a DAG, then G has a node with no incoming edges (i.e., a source).

The proof is similar to "tree has $n-1$ edges".

Proof. (by contradiction)

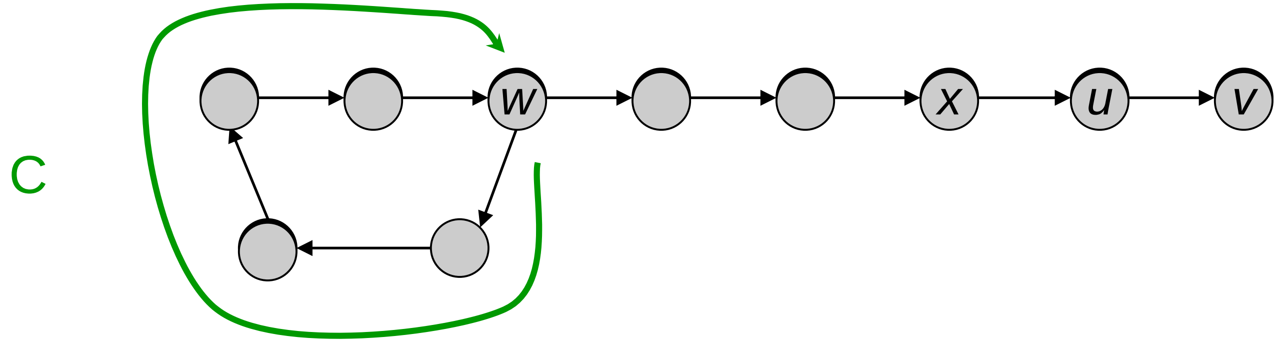
Suppose that G is a DAG and it has no source

Pick any node v, and begin following edges backward from v. Since v has at least one incoming edge (u, v) we can walk backward to u.

Then, since u has at least one incoming edge (x, u), we can walk backward to x.

Repeat until we visit a node, say w, twice.

Let C be the sequence of nodes encountered between successive visits to w. C is a cycle.



DAG => Topological Order

Lemma: If G is a DAG, then G has a topological order

Proof. (by induction on n)

Base case: true if $n = 1$.

Hypothesis: Every DAG with $n - 1$ vertices has a topological ordering.

Inductive Step: Given DAG with $n > 1$ nodes, find a source node v.

$G - \{v\}$ is a DAG, since deleting v cannot create cycles.

Reminder: Always remove vertices/edges to use IH

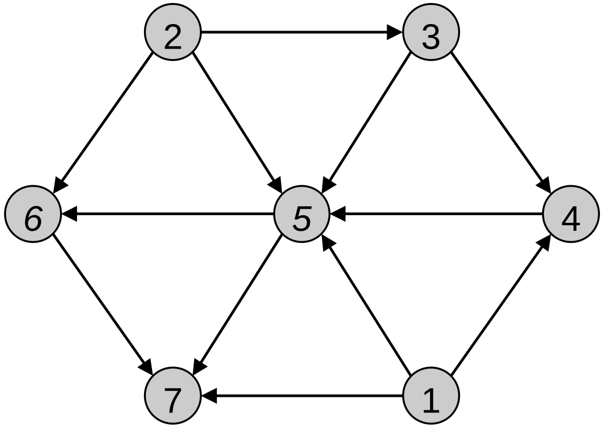
By hypothesis, $G - \{v\}$ has a topological ordering.

Place v first in topological ordering; then append nodes of $G - \{v\}$ in topological order. This is valid since v has no incoming edges.

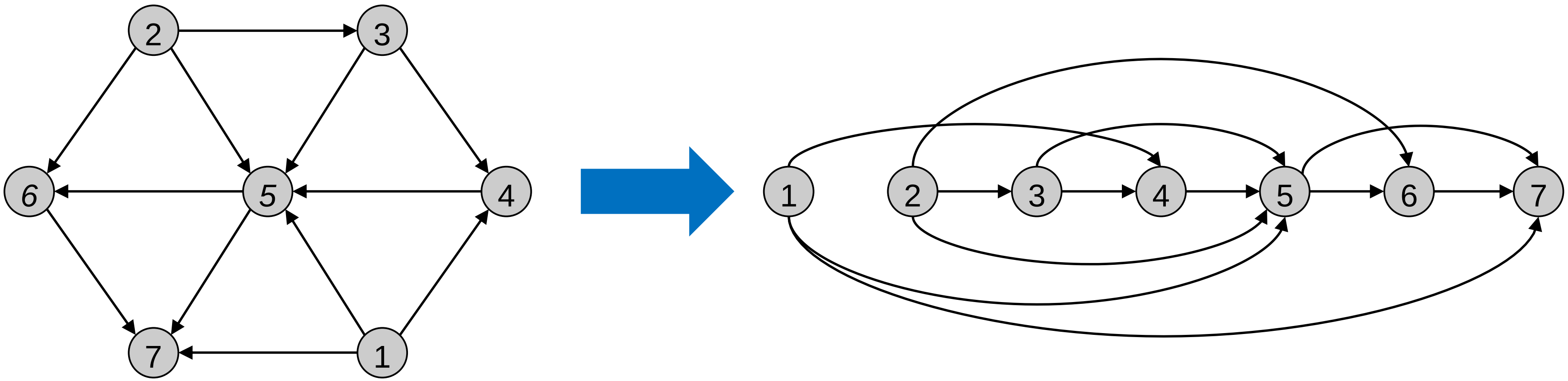
A Characterization of DAGs

G has a topological order $\rightleftarrows$ G is a DAG

Example



Example



Topological order: 1, 2, 3, 4, 5, 6, 7

Summary for last few classes

- Terminology: vertices, edges, paths, connected component, tree, bipartite…
- Vertices vs Edges: $m = O(n^2)$ in general, $m = n - 1$ for trees
- BFS: Layers, queue, shortest paths, all edges go to same or adjacent layer
- DFS: recursion/stack; all edges ancestor/descendant
- Algorithms: Connected Comp, bipartiteness, topological sort
- Techniques: Induction on vertices/layers

CSE 421

Interval Scheduling

Yin Tat Lee

Greedy Algorithms

- Hard to define exactly but can give general properties
  - Solution is built in small steps
  - Decisions on how to build the solution are made to maximize some criterion without looking to the future
    - Want the ‘best’ current partial solution as if the current step were the last step
- May be more than one greedy algorithm using different criteria to solve a given problem

Greedy Strategy

Goal: Given currency denominations: 1, 5, 10, 25, 100, give change to customer using *fewest* number of coins.

Ex: 34¢.

Cashier's algorithm: At each iteration, give the *largest* coin valued ≤ the amount to be paid.

Ex: $2.89.

Greedy is not always Optimal

Observation: Greedy algorithm is sub-optimal for US postal denominations: 1, 10, 21, 34, 70, 100, 350, 1225, 1500.

Counterexample. 140¢.

Greedy: 100, 34, 1, 1, 1, 1, 1, 1.

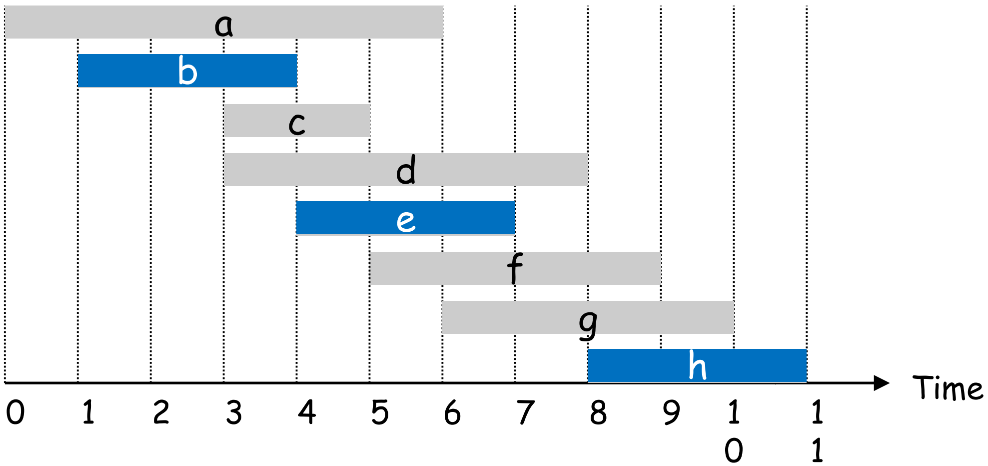
Optimal: 70, 70.

Lesson: Greedy is short-sighted. Always chooses the most attractive choice at the moment. But this may lead to a dead-end later.

Greedy Algorithms

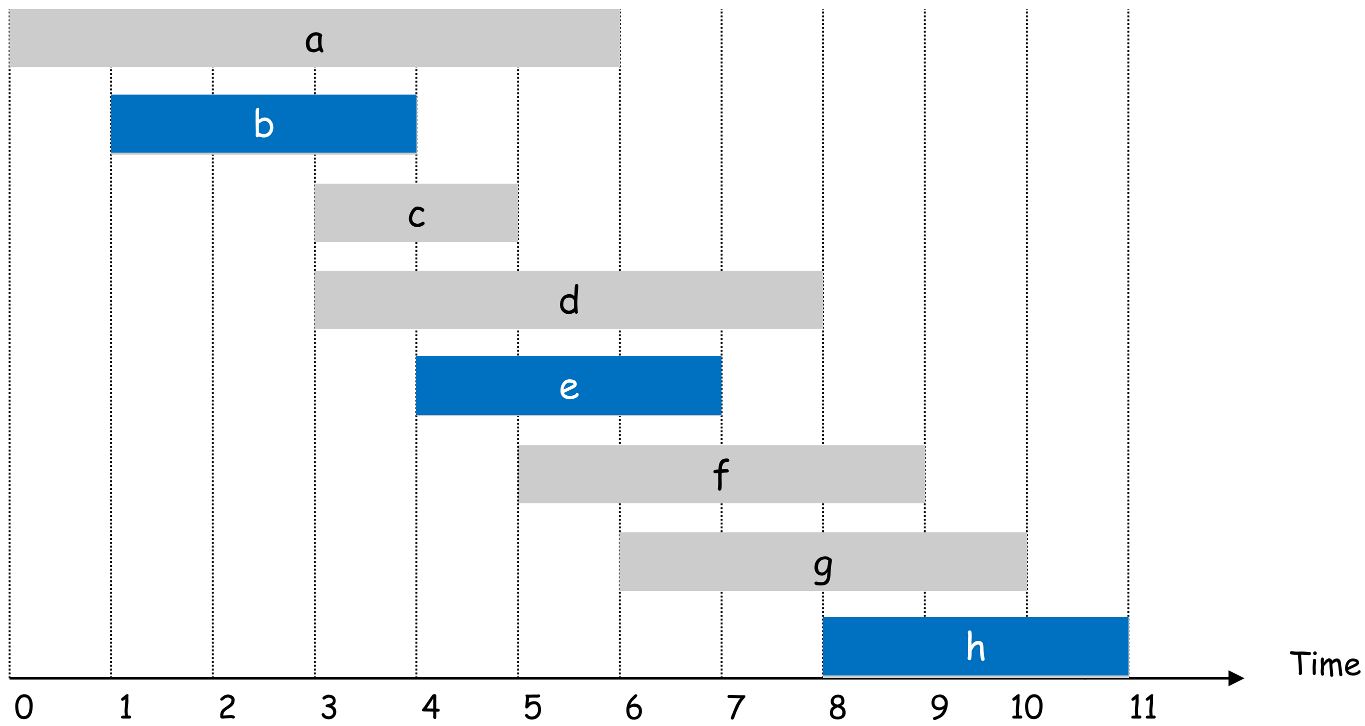
- Greedy algorithms
  - Easy to produce
  - Fast running times
  - Work only on certain classes of problems
    - Hard part is showing that they are correct
- Two methods for proving that greedy algorithms do work
  - Greedy algorithm stays ahead
    - At each step any other algorithm will have a worse value for some criterion that eventually implies optimality
  - Exchange Argument
    - Can transform any other solution to the greedy solution at no loss in quality

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.



Greedy Strategy

Sort the jobs in some order. Go over the jobs and take as much as possible provided it is compatible with the jobs already taken.

Main question:

- What order?
- Does it give the Optimum answer?
- Why?

Possible Approaches for Inter Sched

Sort the jobs in some order. Go over the jobs and take as much as possible provided it is compatible with the jobs already taken.

[Shortest interval] Consider jobs in ascending order of interval length $f(j) - s(j)$.

[Earliest start time] Consider jobs in ascending order of start time $s(j)$.

[Earliest finish time] Consider jobs in ascending order of finish time $f(j)$.

Greedy Alg: Earliest Finish Time

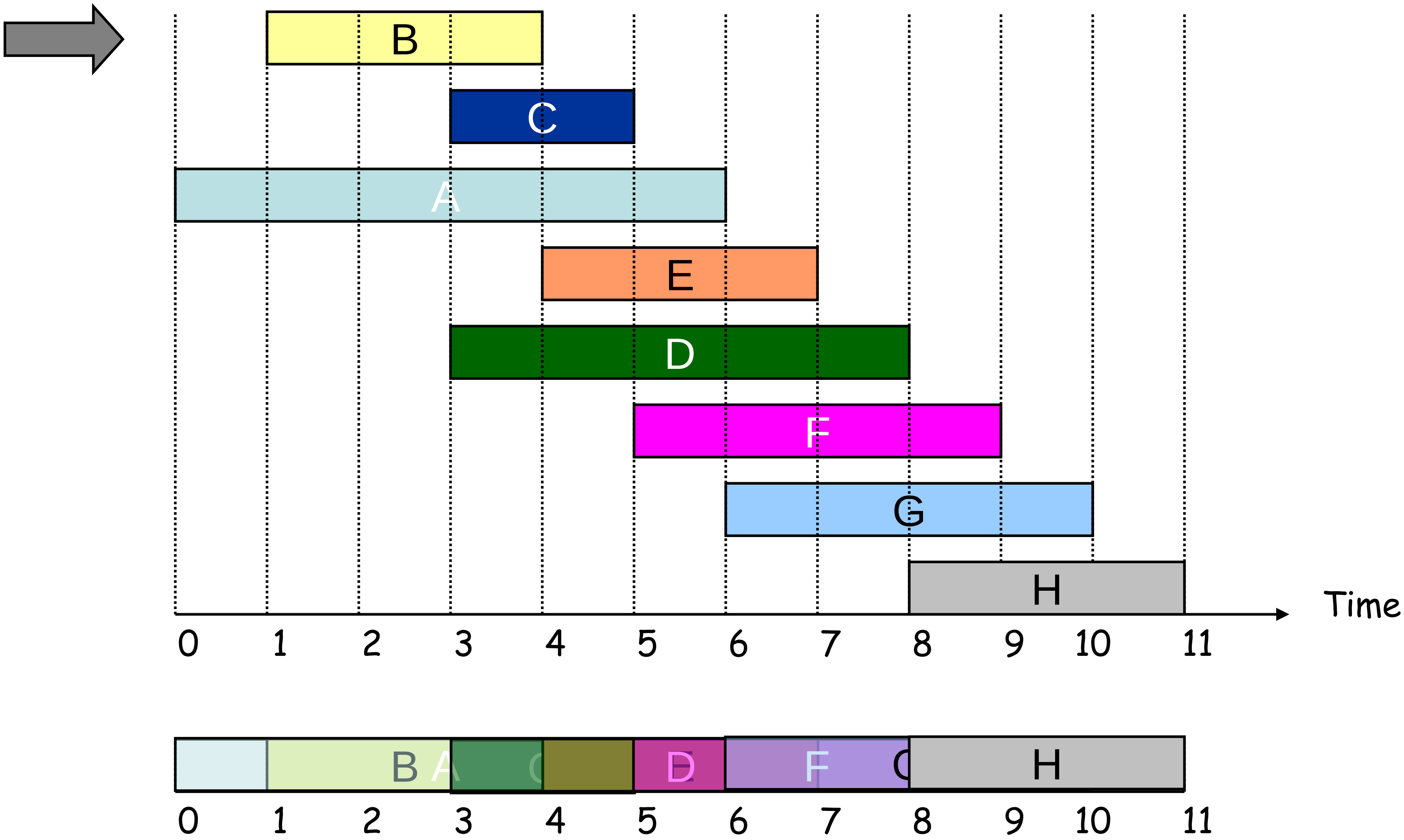
Consider jobs in increasing order of finish time. Take each job provided it's compatible with the ones already taken.

```
Sort jobs by finish times so that f(1) ≤ f(2) ≤ ... ≤ f(n).
A ← ∅
for j = 1 to n {
   if (job j compatible with A)
      A ← A ∪ {j}
}
return A
```

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

- Remember job j^* that was added last to A.
- Job j is compatible with A if $s(j) \geq f(j^*)$.

Greedy Alg: Example



Correctness

- The output is compatible. (This is by construction.)

How to prove it gives maximum number of jobs?

Let $i_1, i_2, i_3, \cdots$ be jobs picked by greedy (ordered by finish time)

Let $j_1, j_2, j_3, \cdots$ be an optimal solution (ordered by finish time)

How about proving $i_k = j_k$ for all k?

No, there can be multiple optimal solutions.

Idea: Prove that greedy outputs the “best” optimal solution.

Given two compatible orders, which is better?

The one finish earlier.

How to prove greedy gives the “best”?

Induction: it gives the “best” during every iteration.

Correctness

Theorem: Greedy algorithm is optimal.

Proof: (technique: "Greedy stays ahead")

Let $i_1, i_2, i_3, \cdots, i_k$ be jobs picked by greedy, $j_1, j_2, j_3, \cdots, j_m$ those in some optimal solution in order.

We show $f(i_r) \leq f(j_r)$ for all r, by induction on r.

Base Case: i_1 chosen to have min finish time, so $f(i_1) \leq f(j_1)$.

IH: $f(i_r) \leq f(j_r)$ for some r

IS: Since $f(i_r) \leq f(j_r) \leq s(j_{r+1})$, j_{r+1} is among the candidates considered by greedy when it picked i_{r+1}, & it picks min finish, so $f(i_{r+1}) \leq f(j_{r+1})$

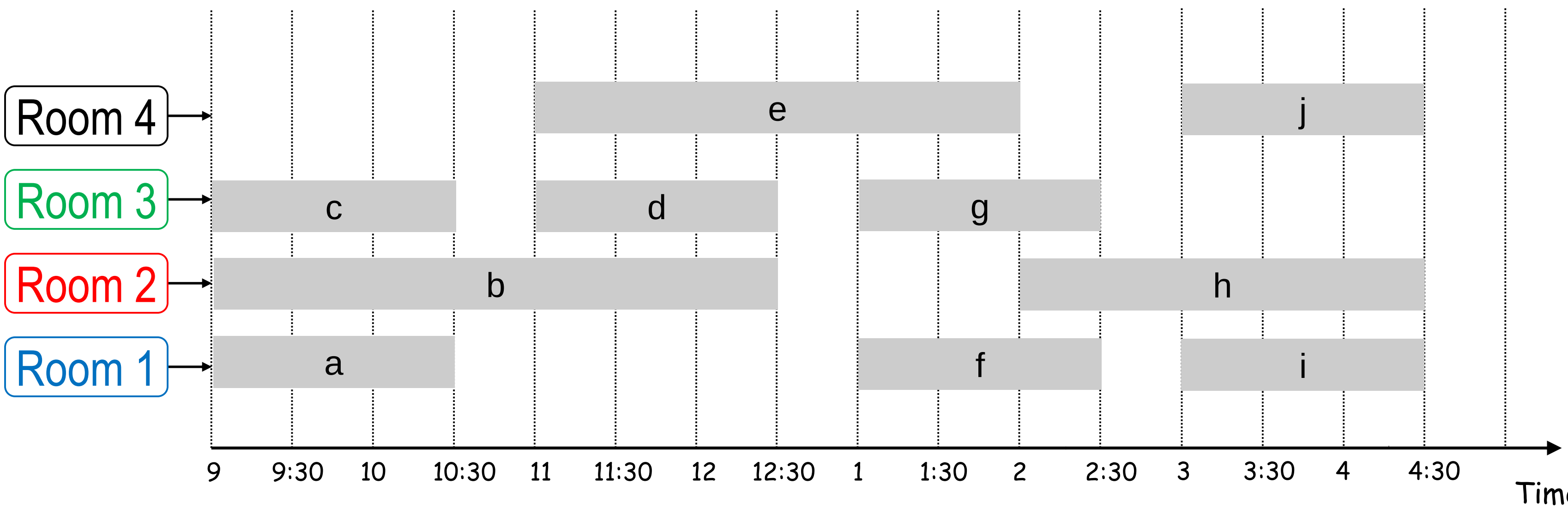
Observe that we must have $k \geq m$, else j_{k+1} is among (nonempty) set of candidates for i_{k+1}.

Interval Partitioning Technique: Structural

Interval Partitioning

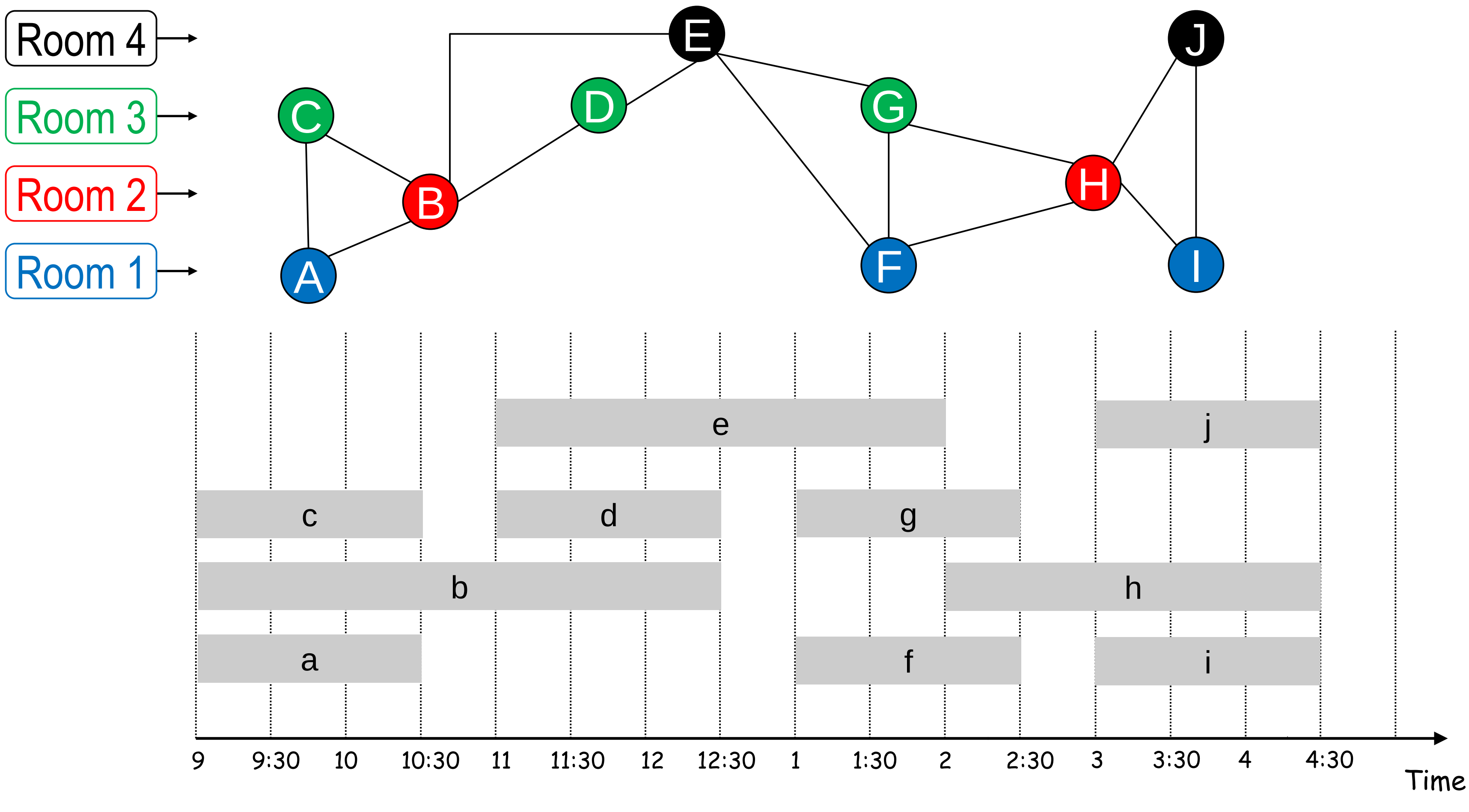
Lecture j starts at $s(j)$ and finishes at $f(j)$.

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



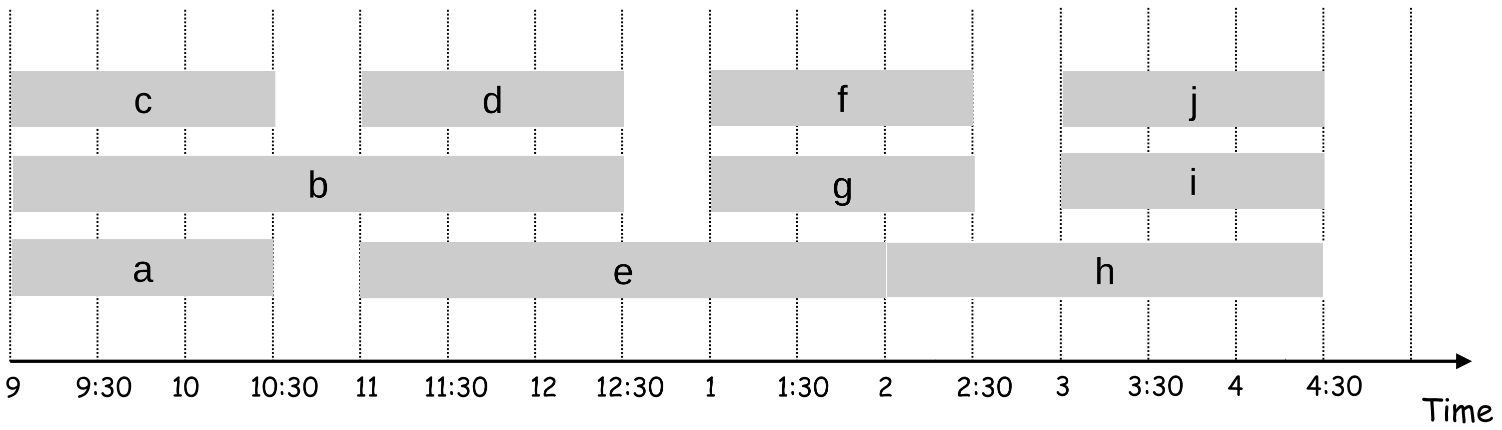
Interval Partitioning

Note: graph coloring is very hard in general, but graphs corresponding to interval intersections are simpler.



A Better Schedule

This one uses only 3 classrooms



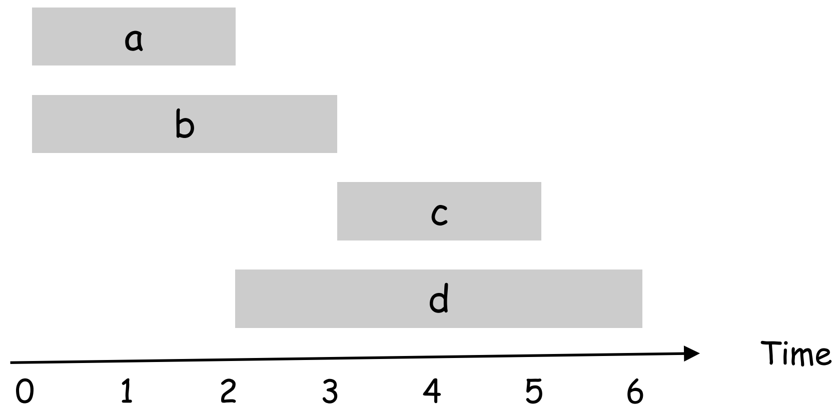
A Greedy Algorithm

Greedy algorithm: Consider lectures in increasing order of finish time: assign lecture to any compatible classroom.

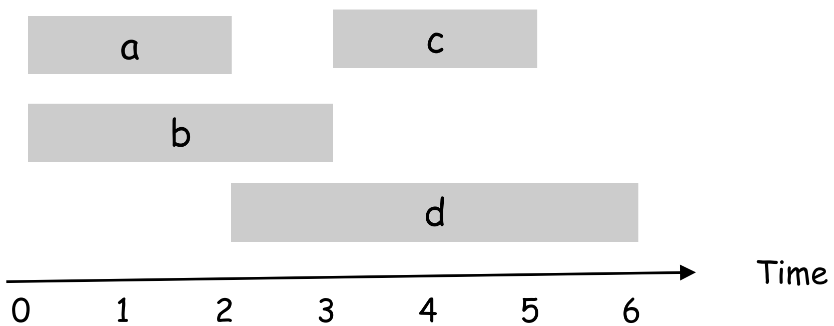
```
Sort intervals by finish time so that f1 ≤ f2 ≤ ... ≤ fn.
d ← 0

for j = 1 t
   if (lect j                              oom k, 1 ≤ k ≤ d)
      schedule le
   else
      allocate a
      sched
      d ← d
}
```

Correctness: This is wrong!

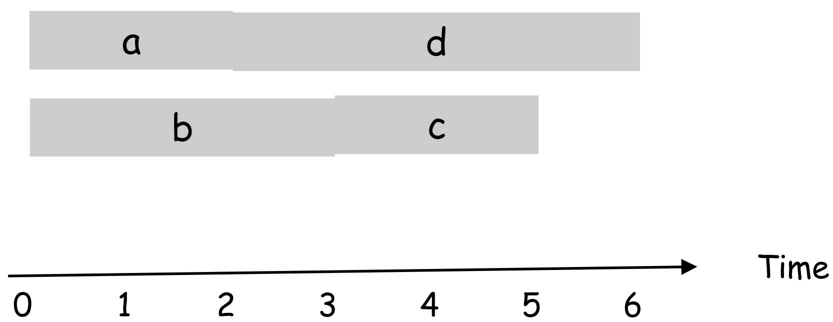
Example



Greedy by finish time gives:



OPT:



A Greedy Algorithm

Greedy algorithm: Consider lectures in increasing order of start time: assign lecture to any compatible classroom.

```
Sort intervals by starting time so that s1 ≤ s2 ≤ ... ≤ sn.
d ← 0

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

Implementation: Exercise!

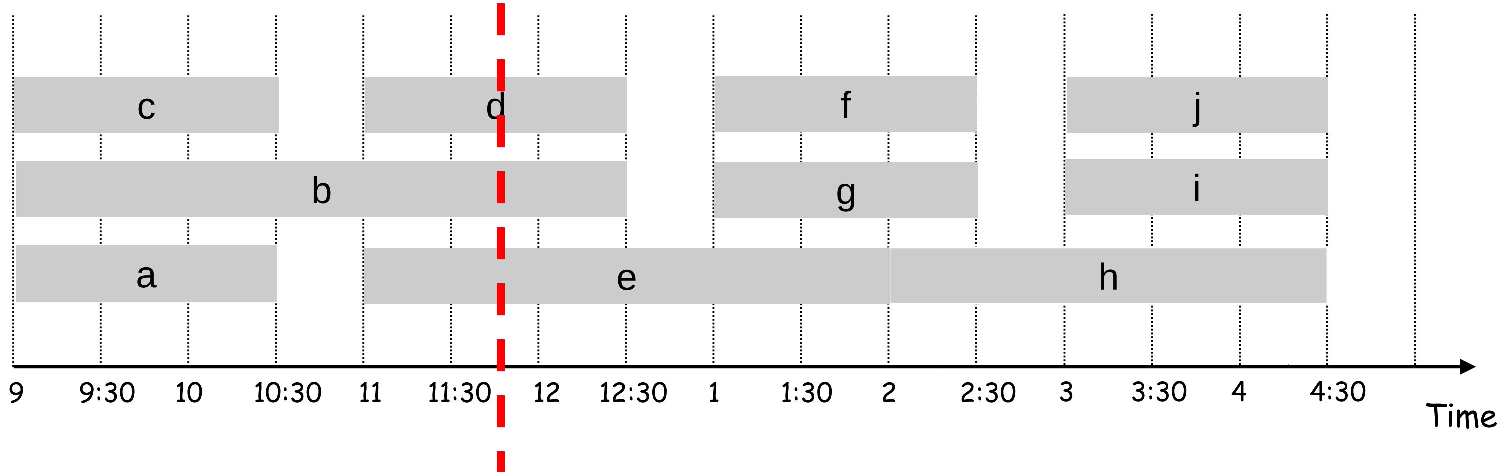
A Structural Lower-Bound on OPT

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

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

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

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



Correctness

Observation: Greedy algorithm never schedules two incompatible lectures in the same classroom.

Theorem: Greedy algorithm is optimal.

Proof (exploit structural property).

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

Classroom d is opened because we needed to schedule a job, say j, that is incompatible with all $d - 1$ previously used classrooms.

Since we sorted by start time, all these incompatibilities are caused by lectures that start no later than $s(j)$.

Thus, we have d lectures overlapping at time $s(j) + \epsilon$, i.e. depth $\geq d$

"OPT Observation" $\Rightarrow$ all schedules use $\geq$ depth classrooms,

so $d =$ depth and greedy is optimal ▪