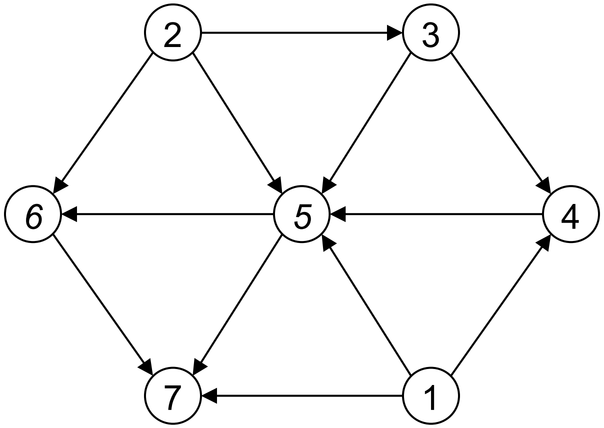
CSE 421: Introduction to Algorithms

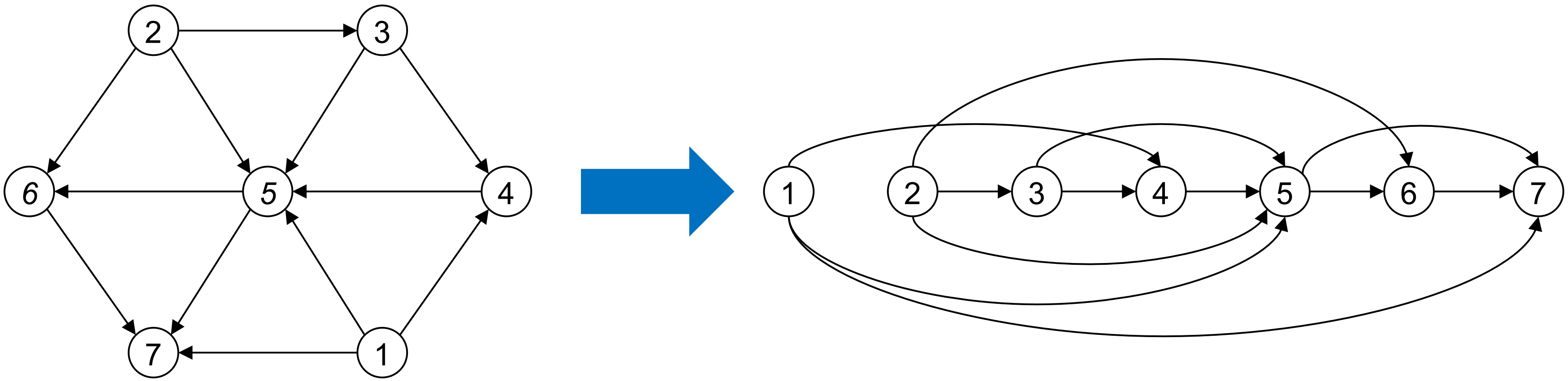
Greedy Algorithms

Shayan Oveis Gharan

Topological Order Algorithm: Example



Topological Order Algorithm: Example



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

Topological Sorting Algorithm

Maintain the following:

count[w] = (remaining) number of incoming edges to node w

S = set of (remaining) nodes with no incoming edges

Initialization:

count[w] = 0 for all w

count[w]++ for all edges (v,w) O(m + n)

S = S ∪ {w} for all w with count[w]=0

Main loop:

while S not empty

- remove some v from S
- make v next in topo order O(1) per node
- for all edges from v to some w O(1) per edge
  - –decrement count[w]
  - –add w to S if count[w] hits 0

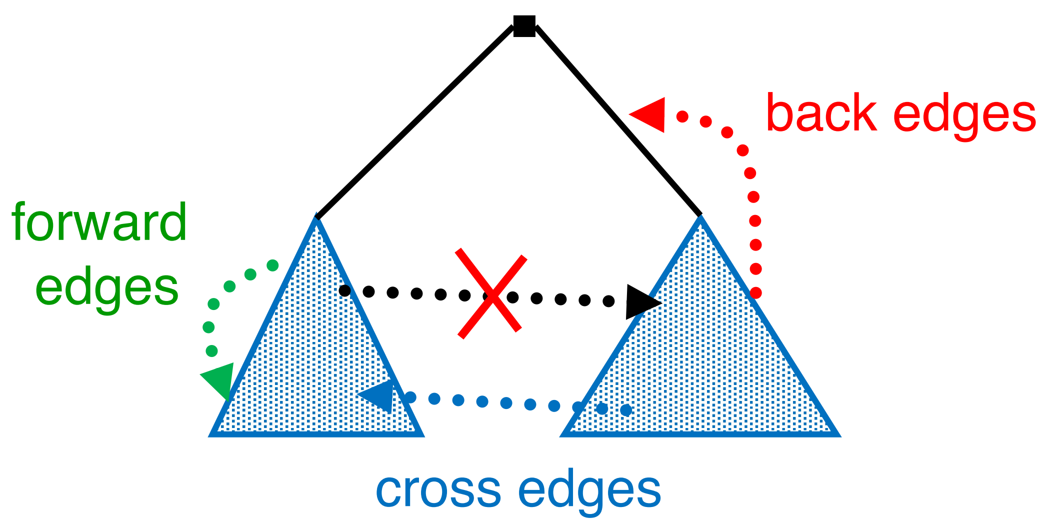
Correctness: clear, I hope

Time: O(m + n) (assuming edge-list representation of graph)

DFS on Directed Graphs

- Before DFS(s) returns, it visits all previously unvisited vertices reachable via directed paths from s

- Every cycle contains a back edge in the DFS tree



Summary

- Graphs: abstract relationships among pairs of objects
- Terminology: node/vertex/vertices, edges, paths, multi-edges, self-loops, connected
- Representation: Adjacency list, adjacency matrix
- Nodes vs Edges: $m = O(n^2)$, often less
- 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

Greedy Algorithms

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.

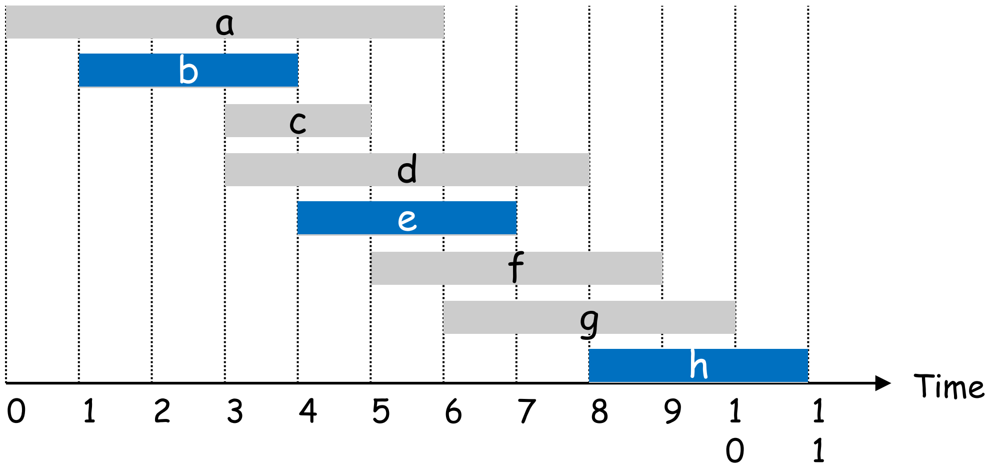
Greedy Algorithms Outline

Pros
- Intuitive
- Often simple to design (and to implement)
- Often fast

Cons
- Often incorrect!

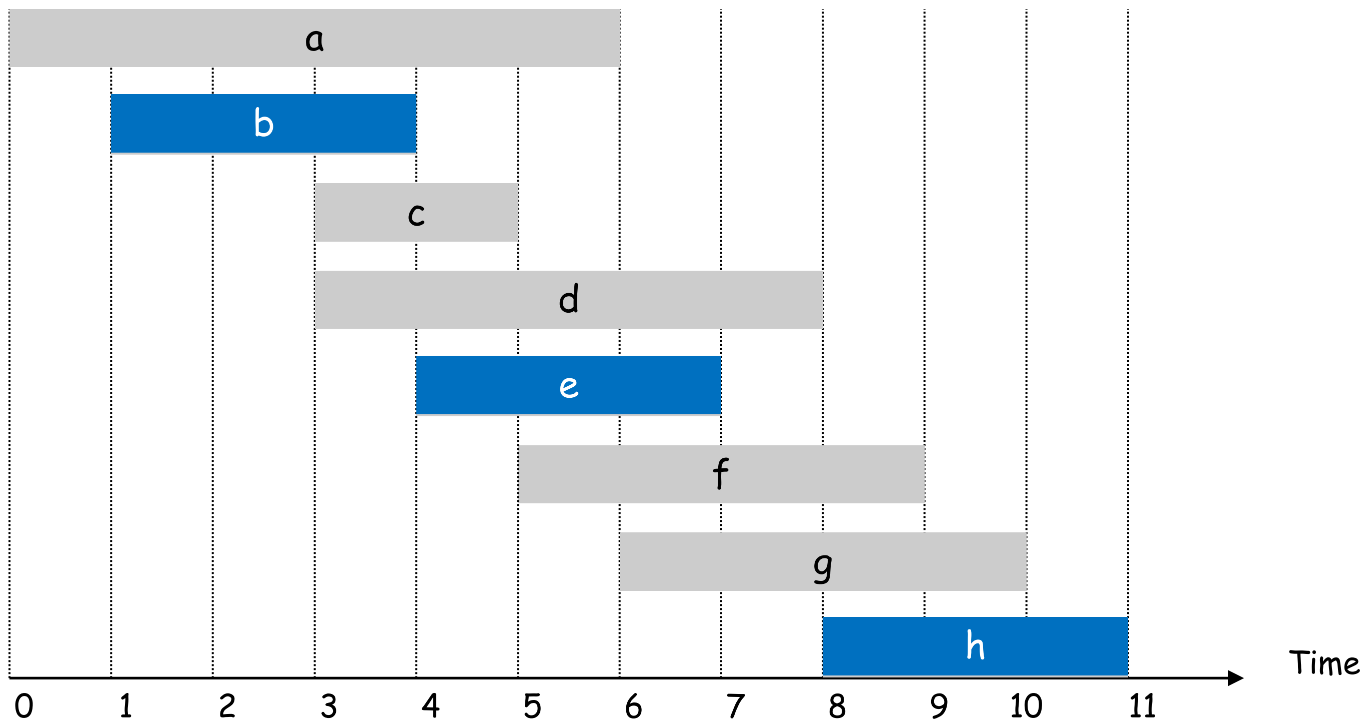
Proof techniques:
- Stay ahead
- Structural
- Exchange arguments

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.

[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.

[Shortest interval] Consider jobs in ascending order of interval length $f_j - s_j$.

[Fewest conflicts] For each job, count the number of conflicting jobs c_j. Schedule in ascending order of conflicts c_j.

Greedy Alg: Earliest Finish Time

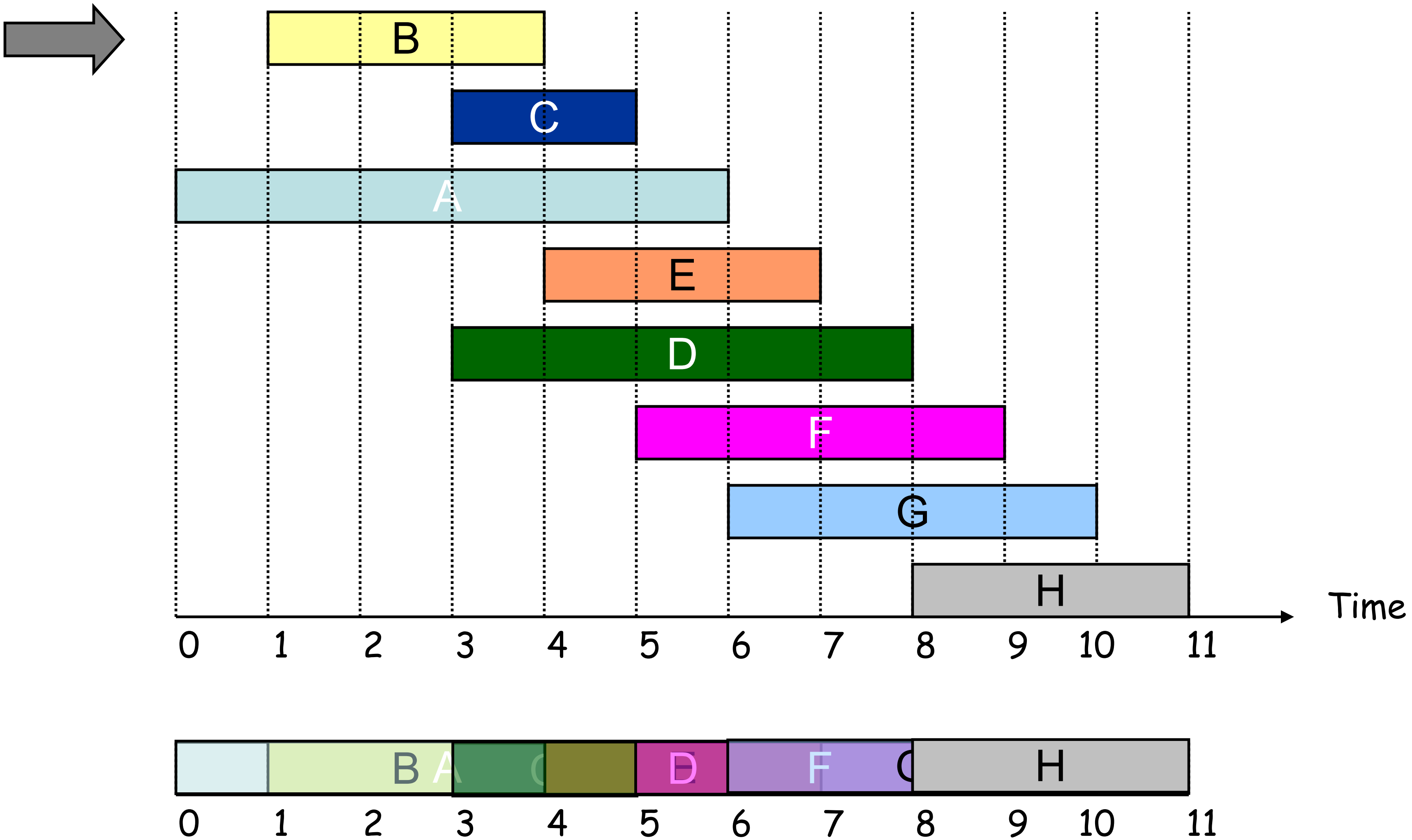
Consider jobs in increasing order of finish time. Take each job provided it's compatible with the ones already taken.

```
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

Theorem: Greedy algorithm is optimal.

Pf: (technique: "Greedy stays ahead")

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

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

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

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

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

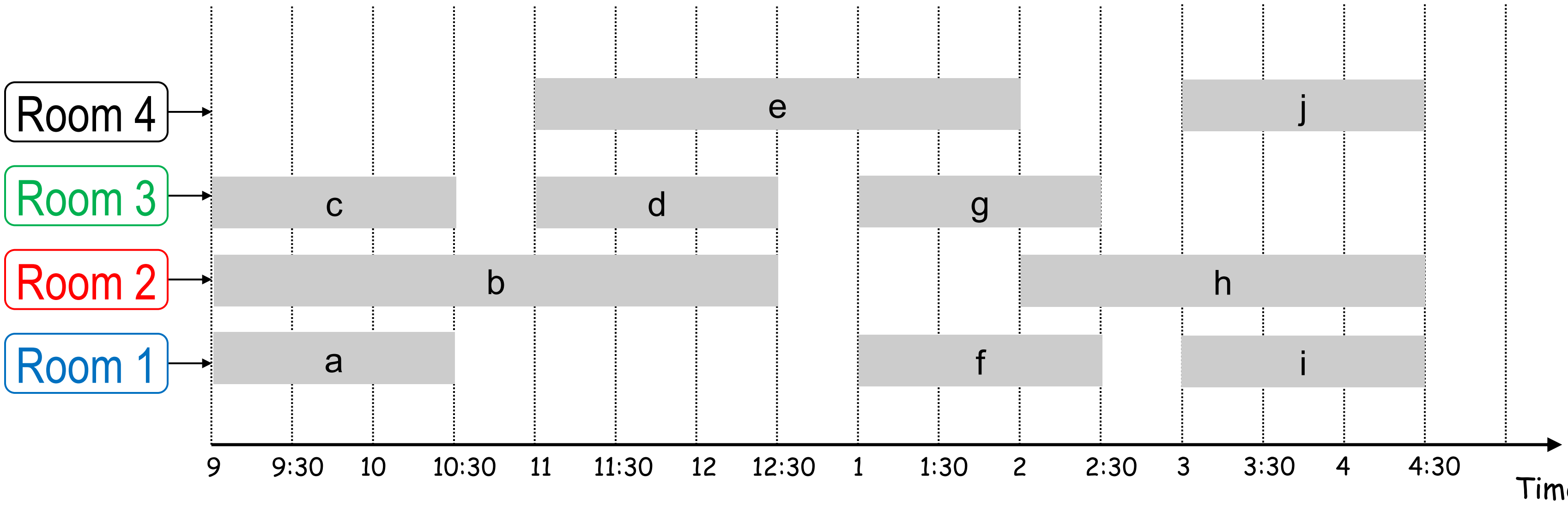
Observe that we must have $k \ge m$, else j_{k+1} is among (nonempty) set of candidates for i_{k+1}

Interval Partitioning
Technique: Structural

Interval Partitioning

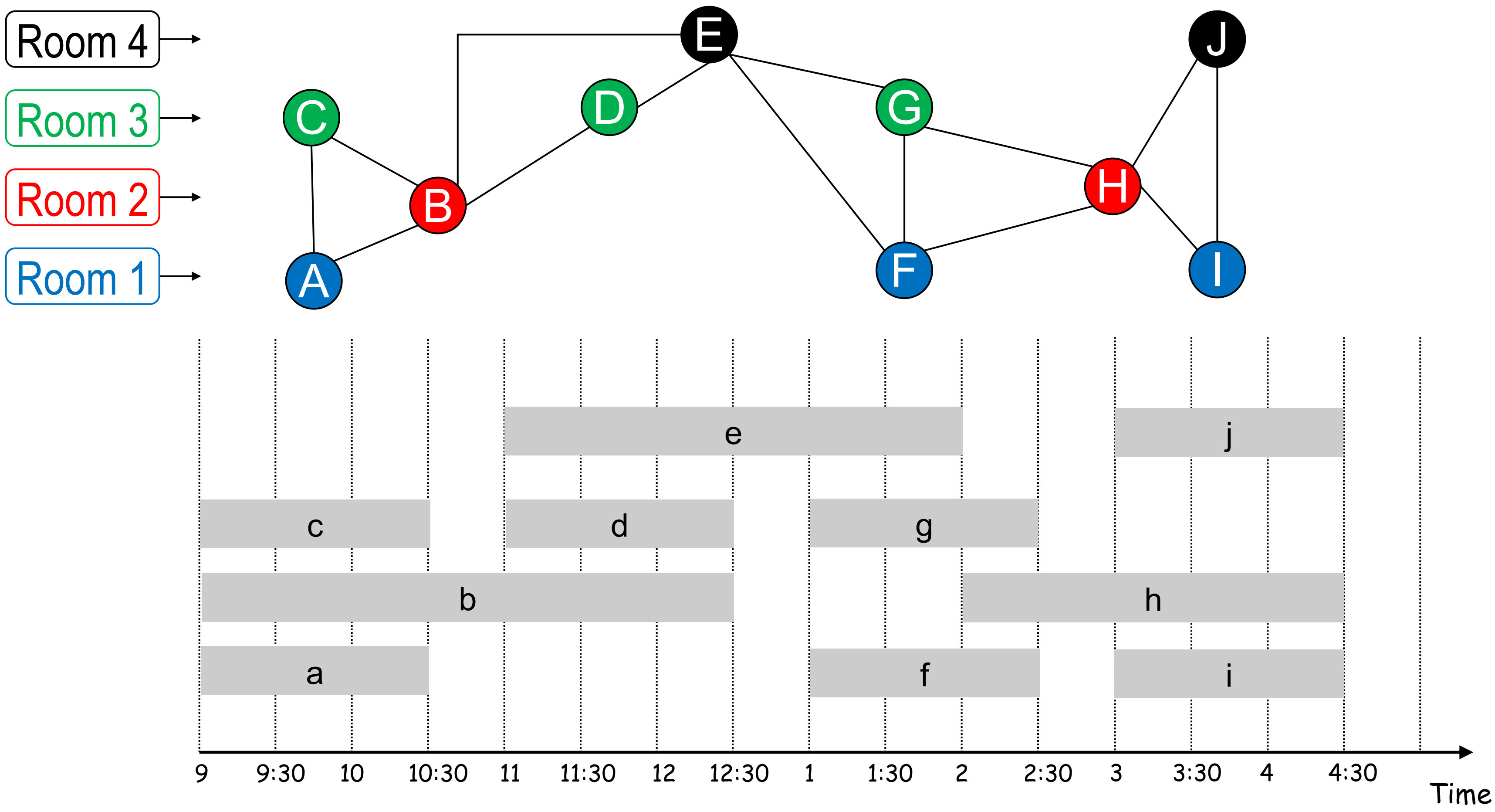
Lecture j starts at s(j) and finishes at f(j).

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



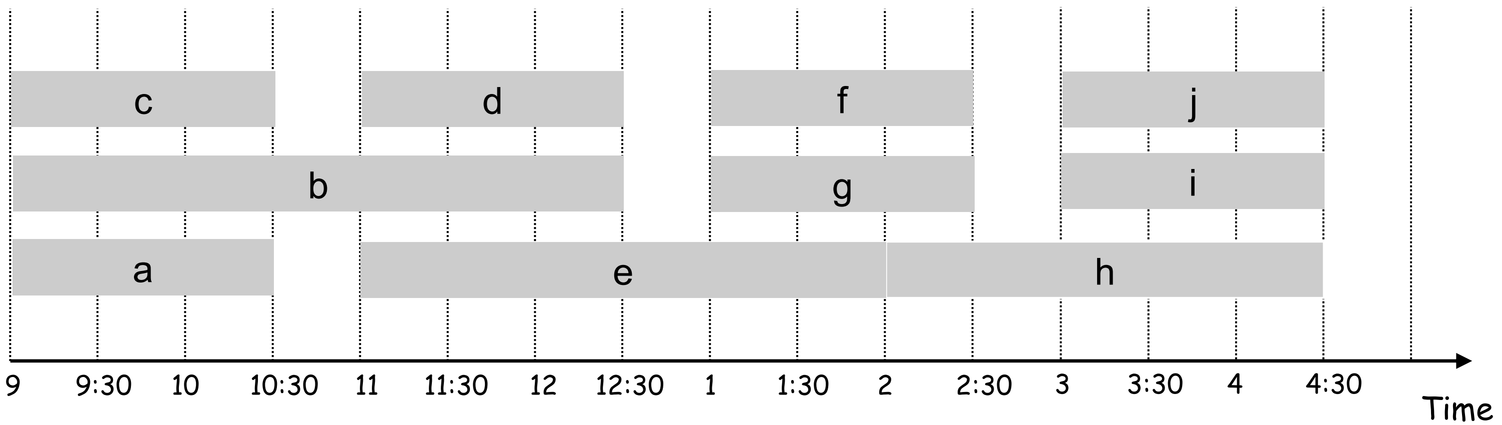
Interval Partitioning

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



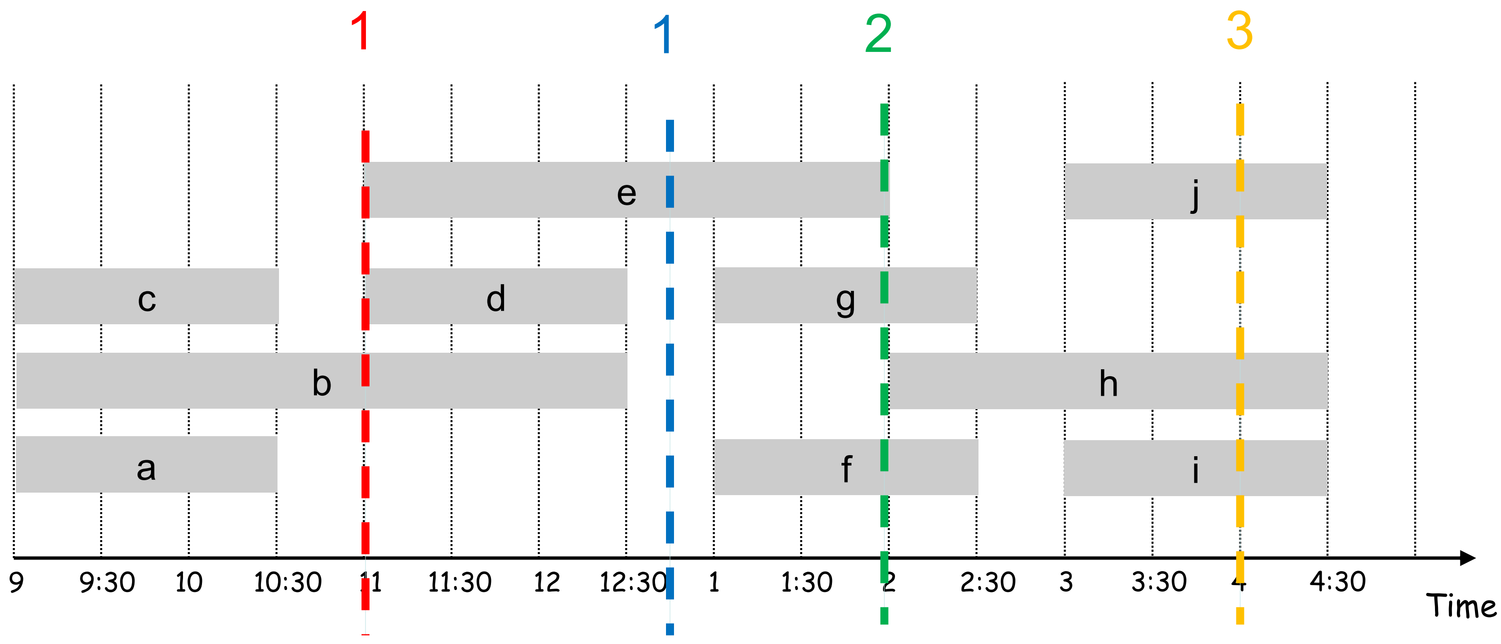
A Better Schedule

This one uses only 3 classrooms



A Structural Lower-Bound on OPT

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



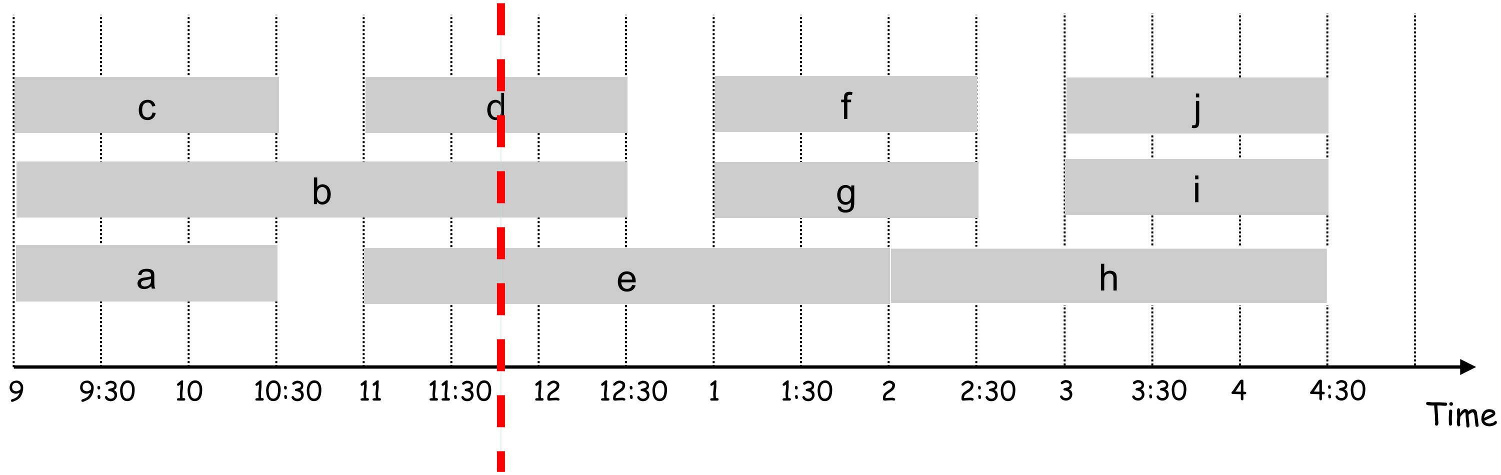
A Structural Lower-Bound on OPT

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.

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



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!

Correctness

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

Theorem: Greedy algorithm is optimal.

Pf (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).

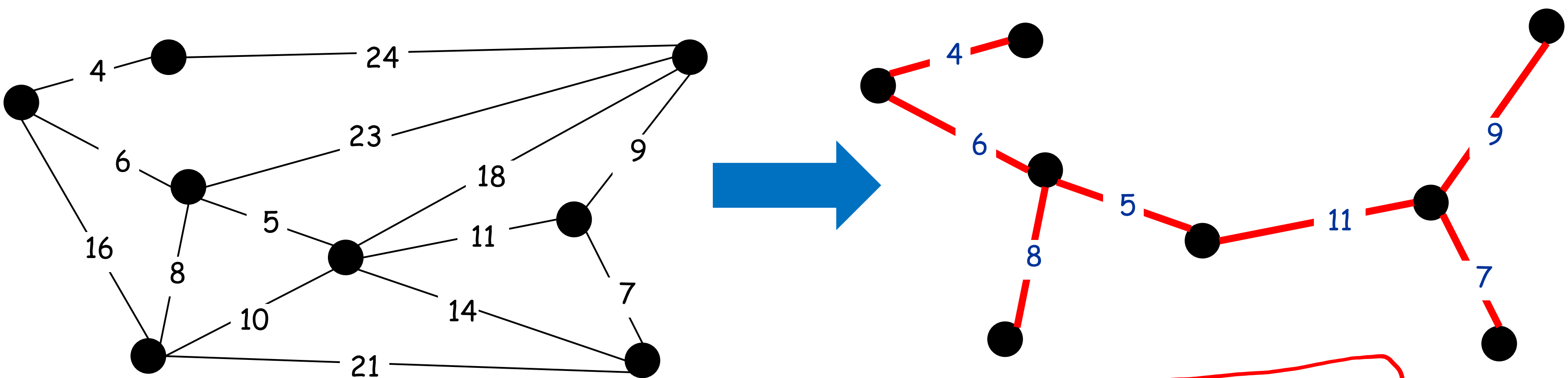
Thus, we have d lectures overlapping at time $s(j) + \epsilon$, i.e. depth ≥ d

"OPT Observation" ⇒ all schedules use ≥ depth classrooms, so d = depth and greedy is optimal ▪

Minimum Spanning Tree Problem

Minimum Spanning Tree (MST)

Given a connected graph $G = (V, E)$ with real-valued edge weights c_e, an MST is a subset of the edges $T \subseteq E$ such that T is a spanning tree whose sum of edge weights is minimized.



$G = (V, E)$

T, $\sum_{e \in T} c_e = 50$

Applications

Network design:

- telephone, electrical, hydraulic, TV cable, computer, road

Approximation algorithms for NP-hard problems:

- traveling salesperson problem, Steiner tree

Indirect applications:

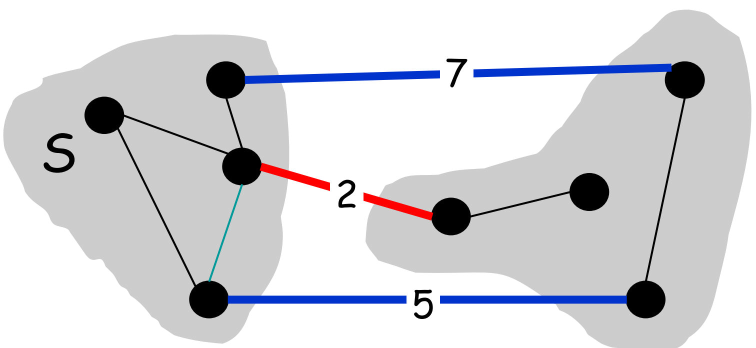
- Graph clustering
- max bottleneck paths
- LDPC codes for error correction
- image registration with Renyi entropy
- learning salient features for real-time face verification
- reducing data storage in sequencing amino acids in a protein
- model locality of particle interactions in turbulent fluid flows
- autoconfig protocol for Ethernet bridging to avoid cycles in a network

Properties of the OPT

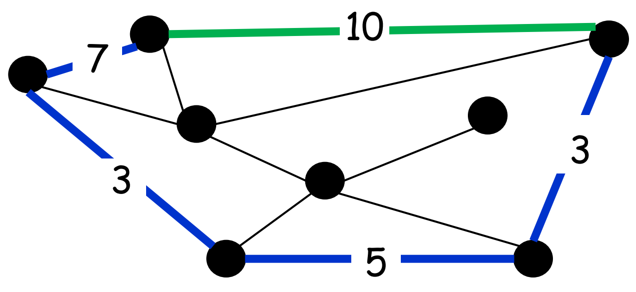
Simplifying assumption: All edge costs c_e are distinct.

Cut property: Let S be any subset of nodes (called a cut), and let e be the min cost edge with exactly one endpoint in S. Then every MST contains e.

Cycle property. Let C be any cycle, and let f be the max cost edge belonging to C. Then no MST contains f.



red edge is in the MST

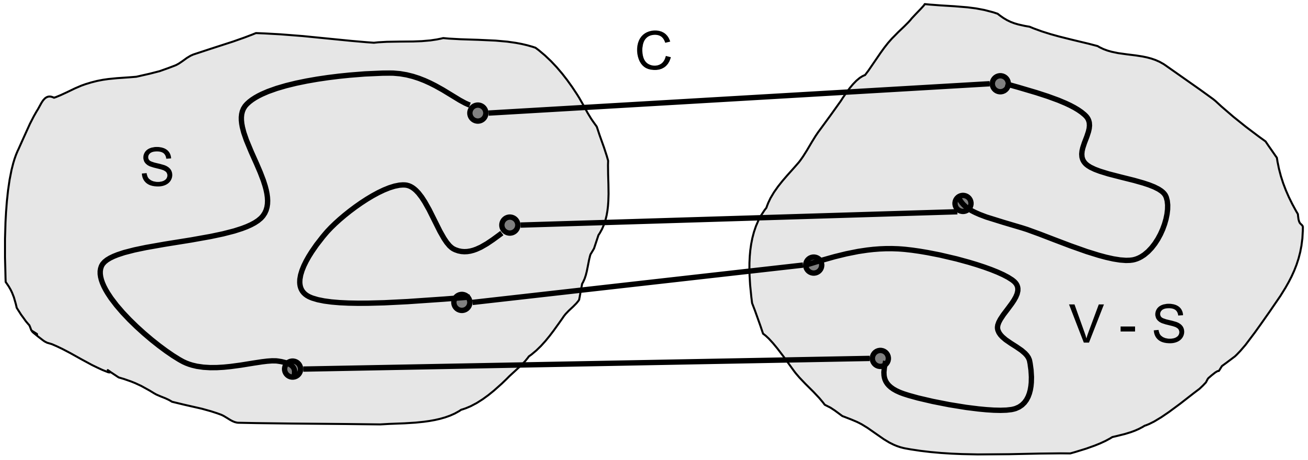


Green edge is not in the MST

Cycles and Cuts

Claim. A cycle crosses a cut (from S to V-S) an even number of times.

Pf. (by picture)



Cut Property: Proof

Simplifying assumption: All edge costs c_e are distinct.

Cut property. Let S be any subset of nodes, and let e be the min cost edge with exactly one endpoint in S. Then the T* contains e.

Pf. By contradiction

Suppose e = {u,v} does not belong to T*.

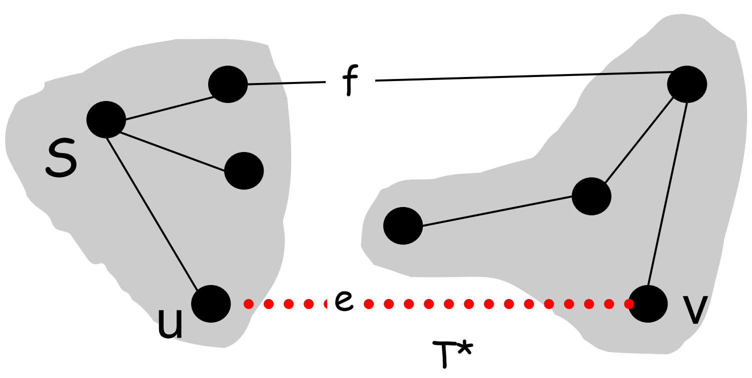
Adding e to T* creates a cycle C in T*.

There is a path from u to v in T* $\Rightarrow$ there exists another edge, say f, that leaves S.

$T = T^* \cup \{e\} - \{f\}$ is also a spanning tree.

Since $c_e < c_f$, $\text{cost}(T) < \text{cost}(T^*)$.

This is a contradiction.



Cycle Property: Proof

Simplifying assumption: All edge costs c_e are distinct.

Cycle property: Let C be any cycle in G, and let f be the max cost edge belonging to C. Then the MST T* does not contain f.

Pf. (By contradiction)

Suppose f belongs to T*.

Deleting f from T* cuts T* into two connected components.

There exists another edge, say e, that is in the cycle and connects the components.

$T = T^* \cup \{e\} - \{f\}$ is also a spanning tree.

Since $c_e < c_f$, $\text{cost}(T) < \text{cost}(T^*)$.

This is a contradiction.