

Graphs, Problems, and Greedy Algorithms

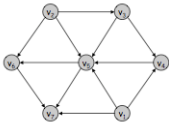
CMSC 451, Summer 2009
Sorelle Friedler

Reminders

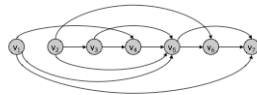
- Homework 1 due on Monday
- Homework write-up guide online
- We begin with some review from last time...

Directed Acyclic Graphs (DAGs)

- A directed graph with no cycles
- Models precedence constraints
- **Topological order:** An ordering of a directed graph's nodes v_1, v_2, \dots, v_n , so that for every edge (v_i, v_j) , $i < j$



a DAG



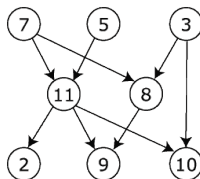
a topological ordering

Topological Ordering Algorithm

To compute a topological ordering of G :
Find a node v with no incoming edges and order it first
Delete v from G
Recursively compute a topological ordering of $G - \{v\}$
and append this order after v

- Initialization: For each node, determine the number of incoming edges. Create a set of edges for which this is 0. One scan through the graph – $O(n+m)$.
- When deleting v , update the counts for all adjacent nodes and add to the set if the count is 0 – $O(\deg(v))$
- Total time: $O(n+m)$

Find a Topological Ordering



Is there more than one?

Check-in

- On your index card please write one of these three things:
 1. Too fast: If the review of graphs was too fast and you don't feel ready to use them
 2. Too slow: If the review of graphs was too slow and you're bored
 3. Just right: If you got what you needed out of the review and are ready to move on

Some Sample Problems

- Demonstrate a few types of problems that we'll see during this course...

Interval Scheduling

- There is some resource
- n requests are made to use the resource
- Requests come in the form of some time interval
- The resource can only serve one request at once

Goal: Maximize the number of requests accepted
 Solution strategy: a simple pass through a carefully sorted version of the data (Greedy Algorithms)

Weighted Interval Scheduling

- Interval scheduling, but each request is given some weight
- Same as interval scheduling for all weights equal to one

Goal: Maximize the total weight of the scheduled requests

Solution strategy: Build up all possible solutions to determine the optimal (Dynamic Programming)

Bipartite Matching

- Bipartite graph: the nodes can be partitioned into two sets that have no edges between them
- Matching: a set of pairs where each pair contains exactly one node from each of the two sets and no node appears more than once

Goal: Find the maximum matching given some bipartite graph

Solution strategy: Build up larger matchings by doing selective backtracking (Network Flow)

Independent Set Problem

- Independent set: a set of nodes in a graph such that no two nodes have an edge between them

Goal: Find the largest independent set in a given graph

Solution strategy: Hard to find a solution efficiently since the power set is large. Easy to check the solution.

Competitive Facility Location

- Two players alternately choose nodes to occupy in a node-weighted graph
- Chosen nodes must form an independent set with all other chosen nodes

Goal: Given some bound, is there a strategy so that a player can always occupy nodes with weights that sum to that bound

Solution strategy: Even hard to check the solution – it needs a case by case analysis.

Greedy Algorithms

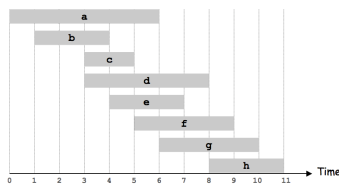
- An algorithm that builds a solution in small steps
- Each small step makes a decision to come closest to the goal
 - E.g. Max algorithm
- Optimal greedy algorithms don't exist for all problems
- Problems that have optimal greedy solutions have nice local properties

Greedy Proof Techniques

- How do we prove that a greedy algorithm is optimal?
- Show that the greedy algorithm has a better solution after every step than any other algorithm could
 - Show that any other solution can be transformed to the greedy solution without hurting its quality

Interval Scheduling

- Two requests are compatible if they don't overlap
- Goal: Find maximum subset of mutually compatible requests



Interval Scheduling: Greedy Algorithm

- Basic idea: For each request in the list, use a simple rule to decide if it should be accepted. Once accepted, all other requests must be compatible with it to be accepted.
- What is the simple rule?

Interval Scheduling: Greedy Algorithm

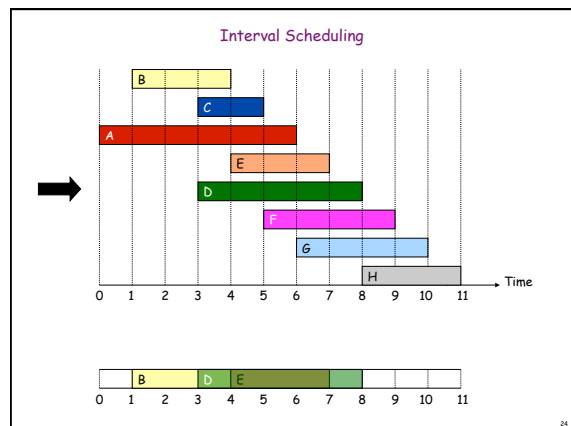
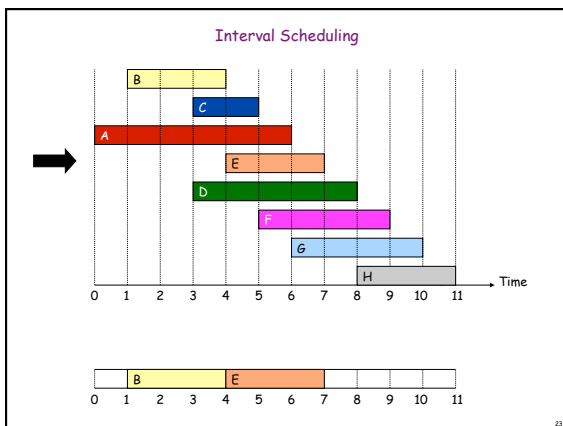
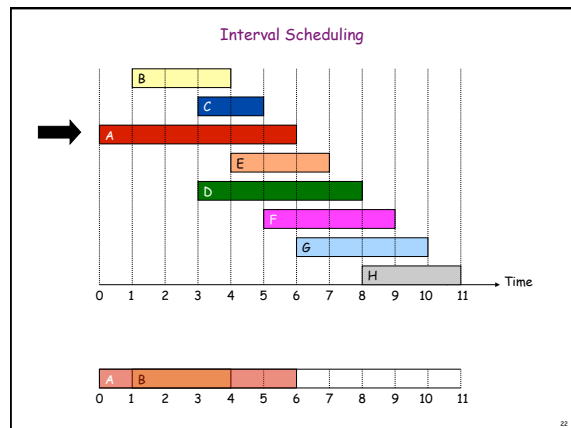
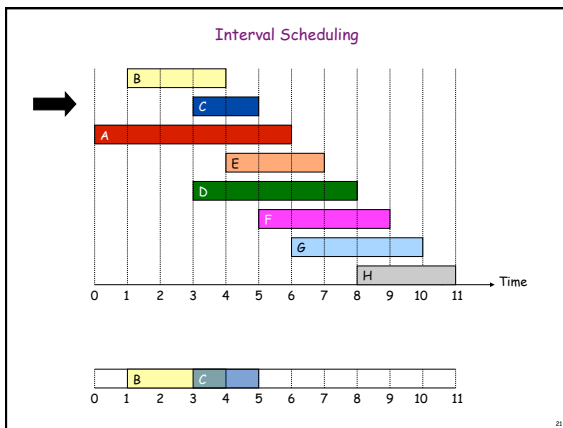
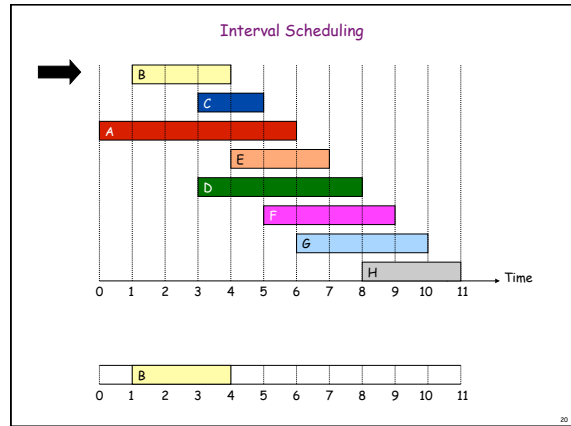
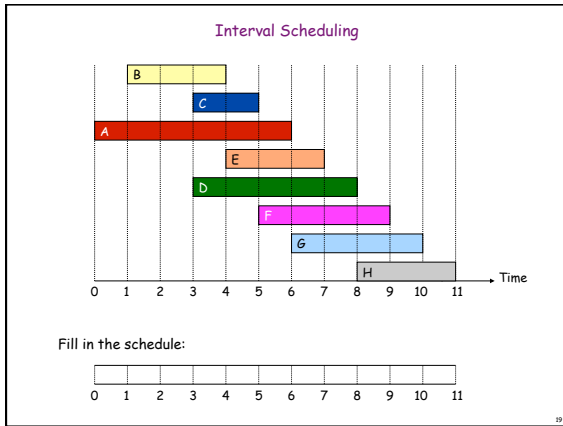
Simple rule options that don't work:

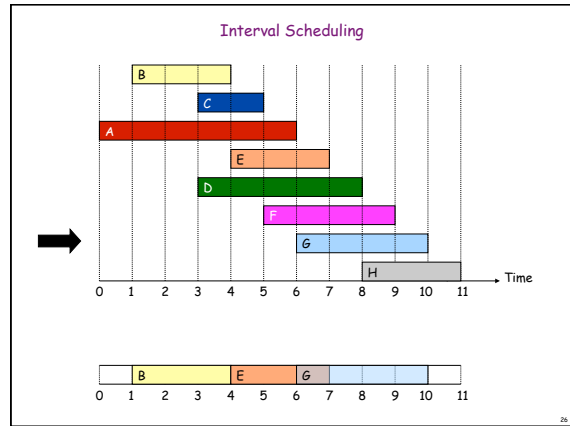
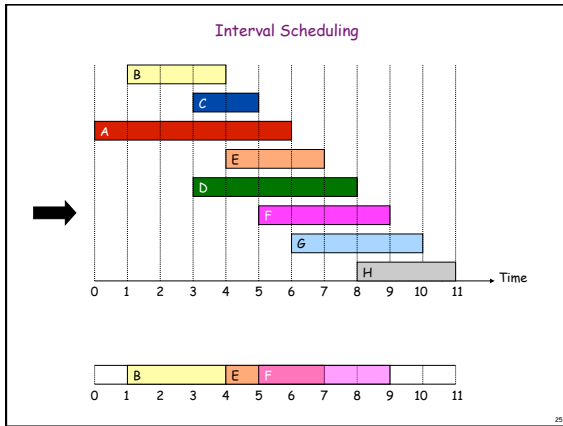


Interval Scheduling: Greedy Algorithm

- Simple rule that does work: accept the request that finishes first
- I.e., free up the resource as fast as possible

```
Sort jobs by finish times so that  $f_1 \leq f_2 \leq \dots \leq f_n$ .
set of jobs selected
A ← ∅
for j = 1 to n {
  if (job j compatible with A)
    A ← A ∪ {j}
}
return A
```





Interval Scheduling: Analysis

- Analysis: $O(n \log n)$ time
 - Sorting $O(n \log n)$
 - Check compatibility in $O(1)$ by remembering last finish time

```

Sort jobs by finish times so that  $f_1 \leq f_2 \leq \dots \leq f_n$ .
set of jobs selected
A ← ∅
for j = 1 to n {
  if (job j compatible with A)
    A ← A ∪ {j}
}
return A
    
```

Interval Scheduling: Analysis

Proof that the greedy algorithm is optimal:

- Show that the number of jobs scheduled is the same (not that the specific jobs are the same)
- Will try to “stay ahead” of the optimal solution at each step
- “Staying ahead” means that the r^{th} job chosen by the greedy algorithm doesn’t finish after the r^{th} job chosen by the optimal algorithm
- Note that by construction, the greedy solution is valid (i.e. all chosen jobs are compatible)

Interval Scheduling: Analysis

Lemma: Given finish time $f(i_r)$ of the r^{th} job scheduled by the greedy algorithm and finish time $f(j_r)$ of the r^{th} job scheduled by the optimal algorithm, $f(i_r) \leq f(j_r)$ for all r .

Proof (by induction):

Base case ($r=1$): Greedy algorithm chooses minimum finish time.

Induction Hypothesis: Assume true for $r-1$.

Induction Step: Since $f(i_{r-1}) \leq f(j_{r-1})$, the greedy algorithm has job j_r as a compatible possibility. It chose the minimum finish time, so $f(i_r) \leq f(j_r)$.

Interval Scheduling: Analysis

Lemma: Given finish time $f(i_r)$ of the r^{th} job scheduled by the greedy algorithm and finish time $f(j_r)$ of the r^{th} job scheduled by the optimal algorithm, $f(i_r) \leq f(j_r)$ for all r .

Proof (by induction):

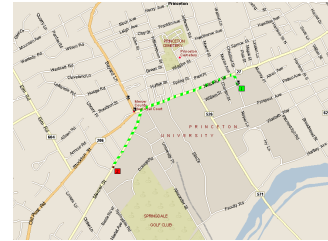
Induction Step: Since $f(i_{r-1}) \leq f(j_{r-1})$, the greedy algorithm has job j_r as a compatible possibility. It chose the minimum finish time, so $f(i_r) \leq f(j_r)$.

Interval Scheduling: Analysis

Theorem: The greedy algorithm is optimal.
 Proof (by contradiction): Let m be the number of jobs scheduled by the optimal algorithm and k the number scheduled by the greedy algorithm.

- Assume $m > k$.
- By the lemma, $f(i_k) \leq f(j_k)$.
- Since $m > k$, the optimal algorithm schedules some job j_{k+1}
- But j_{k+1} is compatible with the greedy set

Shortest Paths in a Graph



shortest path from Princeton CS department to Einstein's house

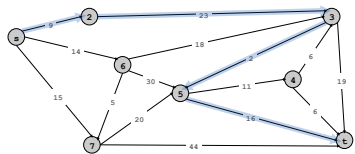
Shortest Path Problem

Shortest path network.

- Directed graph $G = (V, E)$.
- Source s , destination t .
- Length ℓ_e = length of edge e .

Shortest path problem: find shortest directed path from s to t .

cost of path = sum of edge costs in path



Cost of path $s-2-3-5-t$
 $= 9 + 23 + 2 + 16$
 $= 50$.

33

Dijkstra's Algorithm

Input: edge-weighted graph $G = (V, E, \ell)$, source s , sink t
 Let S be the set of explored nodes

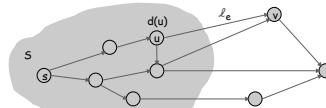
For each u in S , store a distance $d(u)$
 Initialize $S=\{s\}$ and $d(s)=0$

While $S \neq V$

Select a node v not in S with at least one edge from S such that $\pi(v) = \left(\min_{e=(u,v): u \in S} d(u) \right) + \ell_e$ is minimized

Add v to S and set $d(v) = \pi(v)$

EndWhile



34

Dijkstra's Algorithm

Input: edge-weighted graph $G = (V, E, \ell)$, source s , sink t
 Let S be the set of explored nodes

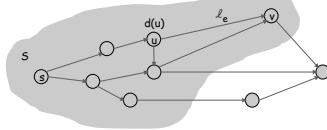
For each u in S , store a distance $d(u)$
 Initialize $S=\{s\}$ and $d(s)=0$

While $S \neq V$

Select a node v not in S with at least one edge from S such that $\pi(v) = \left(\min_{e=(u,v): u \in S} d(u) \right) + \ell_e$ is minimized

Add v to S and set $d(v) = \pi(v)$

EndWhile



35

Dijkstra's Algorithm: Data Structure

Input: edge-weighted graph $G = (V, E, \ell)$, source s , sink t
 Let S be the set of explored nodes

For each u in S , store a distance $d(u)$
 Initialize $S=\{s\}$ and $d(s)=0$

While $S \neq V$

Select a node v not in S with at least one edge from S such that $\pi(v) = \left(\min_{e=(u,v): u \in S} d(u) \right) + \ell_e$ is minimized

Add v to S and set $d(v) = \pi(v)$

EndWhile

Need to be able to:

- Determine minimum $\pi(v)$
- Update $d(v)$
- Update S

36

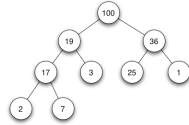
Dijkstra's Algorithm: Data Structure

Priority Queue:

- A data structure that maintains a set of elements S
- Each element has an associated key that indicates its priority
- Supports the following operations:
 1. Add an element to the queue
 2. Remove the element with the highest priority

Example Implementation: Heaps

- Tree-based data structure
- $key(parent) \geq key(child)$
- Operations:
 1. Find max $\Theta(1)$
 2. Delete max $\Theta(\log n)$
 3. Increase key $\Theta(\log n)$
 4. Insert key/value pair $\Theta(\log n)$
 5. Merge: combine two heaps $\Theta(n)$



37

Dijkstra's Algorithm: Implementation

For each unexplored node, explicitly maintain $\pi(v) = \min_{e=(u,v): u \in S} d(u) + \ell_e$.

- Next node to explore = node with minimum $\pi(v)$.
- When exploring v, for each incident edge $e = (v, w)$, update $\pi(w) = \min \{ \pi(w), \pi(v) + \ell_e \}$.

Efficient implementation. Maintain a priority queue of unexplored nodes, prioritized by $\pi(v)$.

38

Dijkstra's Algorithm: Time Analysis

Input: edge-weighted graph $G = (V, E, \ell)$, source s, sink t
 Let S be the set of explored nodes
 For each u in S, store a distance d(u)
 Initialize $S = \{s\}$ and $d(s) = 0$
 While $S \neq V$
 Select a node v not in S with at least one edge from S such that $\pi(v) = \left(\min_{e=(u,v): u \in S} d(u) \right) + \ell_e$ is minimized
 Add v to S and set $d(v) = \pi(v)$ (shortest path to some u in explored part, followed by a single edge (u, v))
 EndWhile

PQ Operation	Dijkstra	Array	Binary heap	d-way Heap	Fib heap †
Insert	n	n	log n	d log _d n	1
ExtractMin	n	n	log n	d log _d n	log n
ChangeKey	m	1	log n	log _d n	1
IsEmpty	n	1	1	1	1
Total		n^2	$m \log n$	$m \log_{m/d} n$	$m + n \log n$

† Individual ops are amortized bounds

39