

Chapter 4

Greedy Algorithms



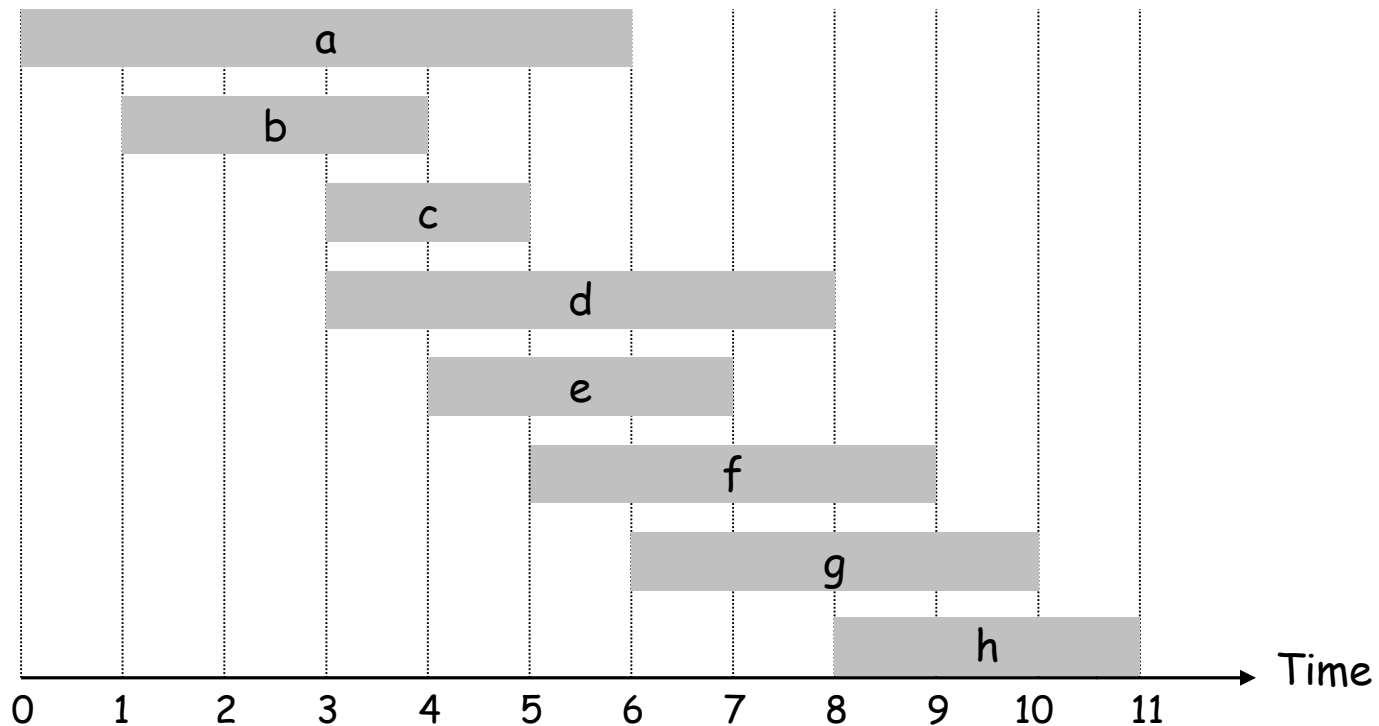
Slides by Kevin Wayne.
Copyright © 2005 Pearson-Addison Wesley.
All rights reserved.

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.

- [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 .

Interval Scheduling: Greedy Algorithms

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



breaks earliest start time



breaks shortest interval



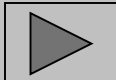
breaks fewest conflicts

Exercise: Show counter examples to each

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  
A ←  $\phi$   
for j = 1 to n {  
    if (job j compatible with A)  
        A ← A  $\cup$  {j}  
}  
return A
```



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

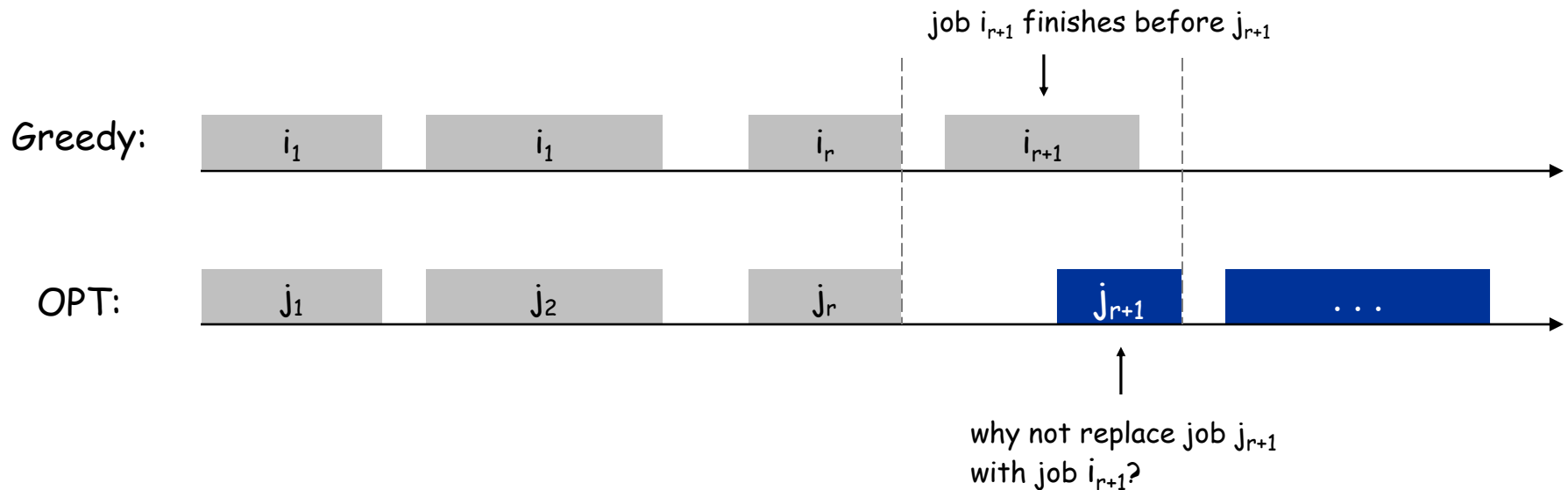
- Remember job j^* that was added last to A.
- Job j is compatible with A if $s_j \geq f_{j^*}$.

Interval Scheduling: Analysis via Greedy Stays Ahead Technique

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 .

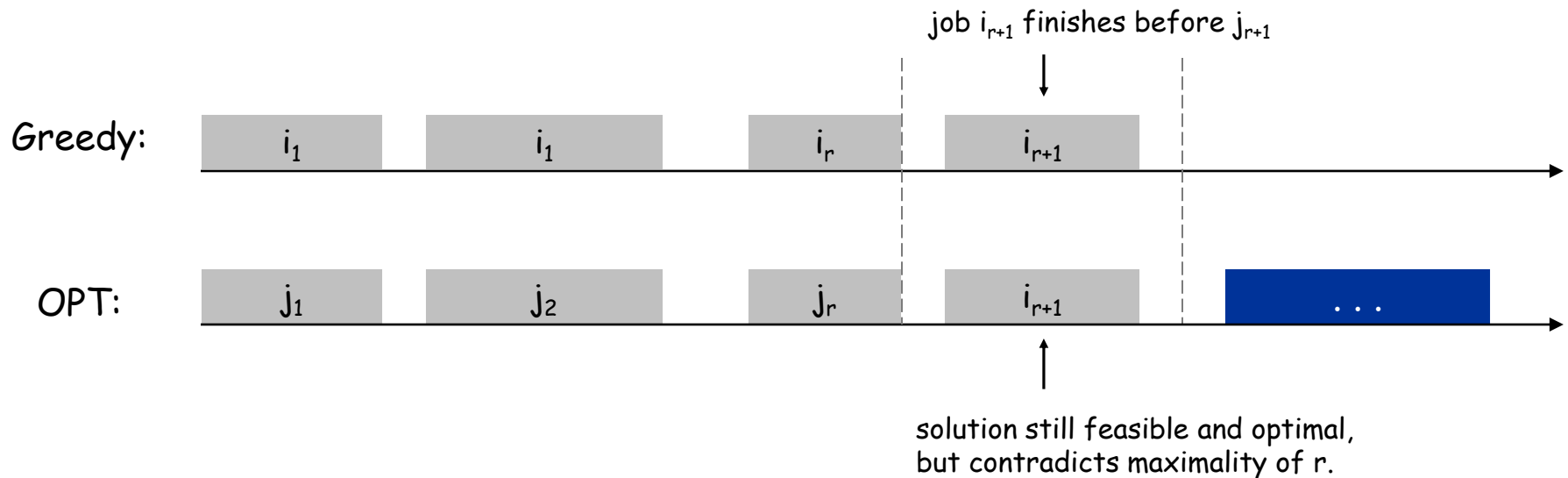


Interval Scheduling: Analysis via Greedy Stays Ahead Technique

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 .



Interval Scheduling: Analysis via Induction

- Induction statement

$P(k)$: There is an optimal solution that agrees with the greedy solution in the first k jobs.

- $P(n)$ is what we want to prove.
- Base case: $P(0)$
- We essentially proved the induction step...

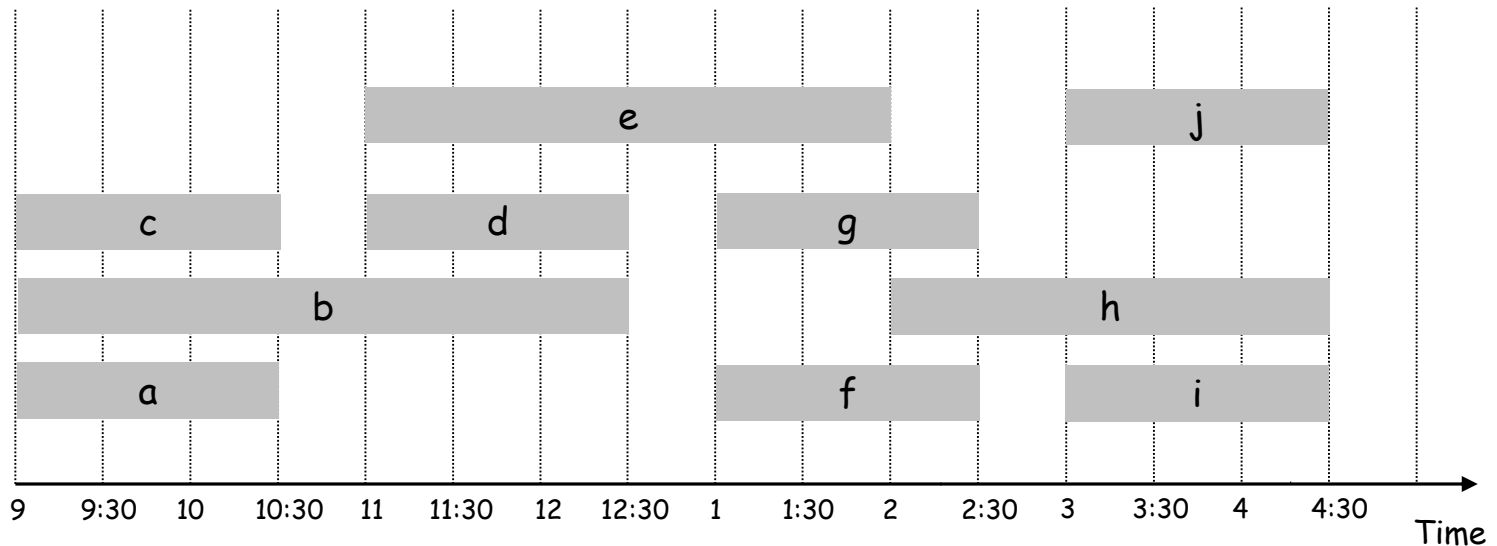
4.1 Interval Partitioning

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 4 classrooms to schedule 10 lectures.

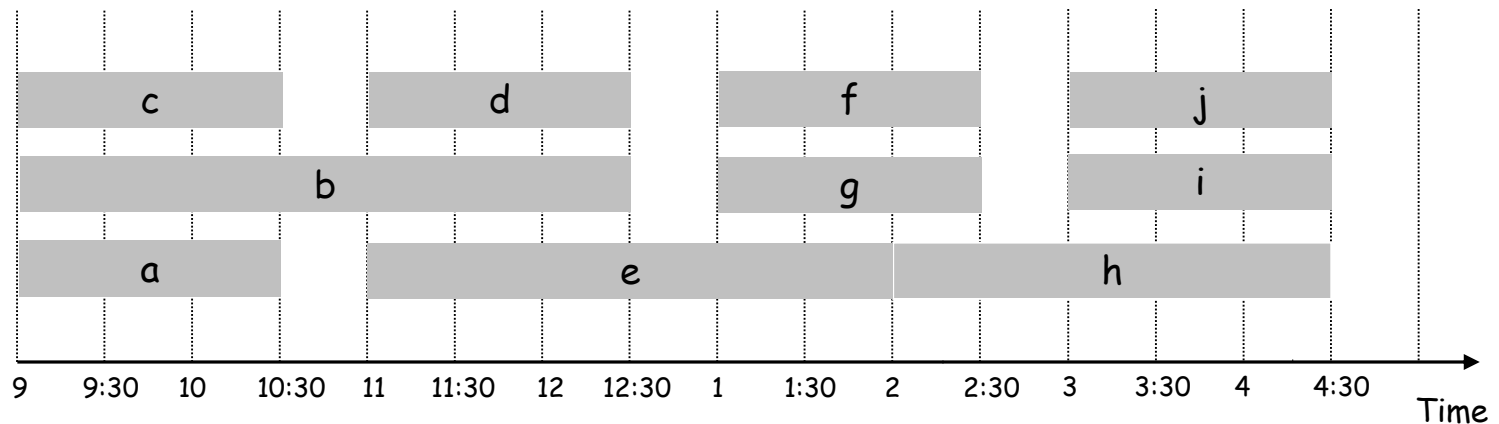


Interval Partitioning

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

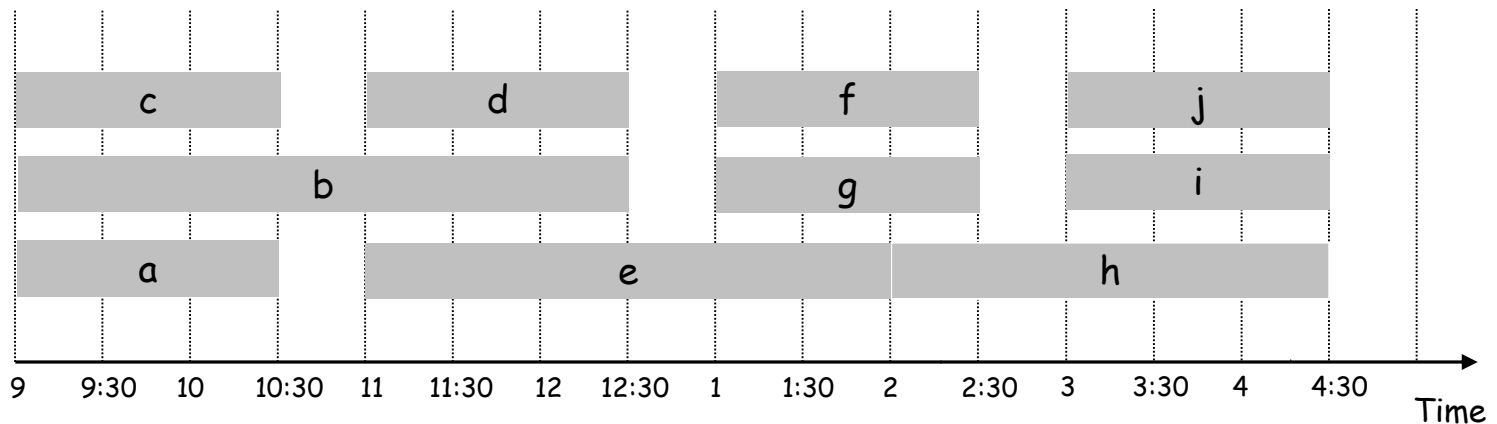
Def. The **depth** of a set of open intervals is the maximum number that contain any given time.

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?



Interval Partitioning: 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  $s_1 \leq s_2 \leq \dots \leq s_n$ .  
 $d \leftarrow 0$   $\leftarrow$  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  $d + 1$   
         $d \leftarrow d + 1$   
}
```

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

- For each classroom k , maintain the finish time of the last job added.
- Keep the classrooms in a priority queue.

Interval Partitioning: Analysis via Structural Argument

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

Theorem. Greedy algorithm is optimal.

Pf.

- 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 $d-1$ other 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 $s_j + \varepsilon$.
- Key observation \Rightarrow all schedules use $\geq d$ classrooms. ▪

Interval Partitioning: Example of Duality

Our first example of “duality”!

High-level overview of proof of correctness:

- Identify obstacles to scheduling in few classrooms
 - Sets of overlapping lectures
- Show that our algorithm's solution matches some obstacle
 - If our solution uses d classrooms, then there is a set of d overlapping lectures
- Conclude that our solution cannot be improved

4.2 Scheduling to Minimize Lateness

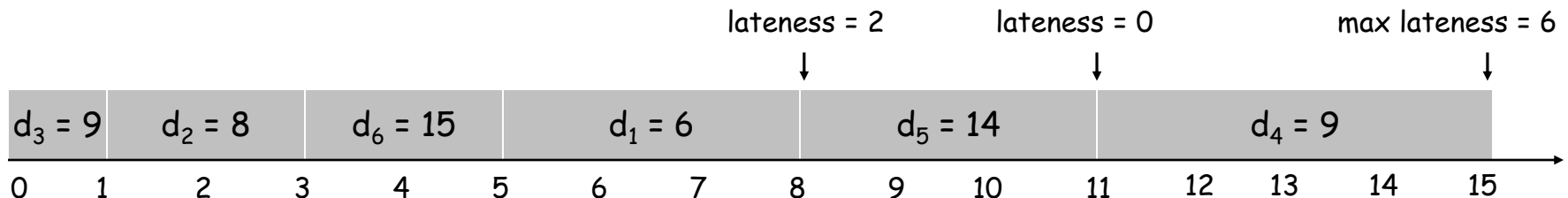
Scheduling to Minimizing Lateness

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 $f_j = s_j + t_j$.
- 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 $d_j - t_j$.

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 $d_j - t_j$.

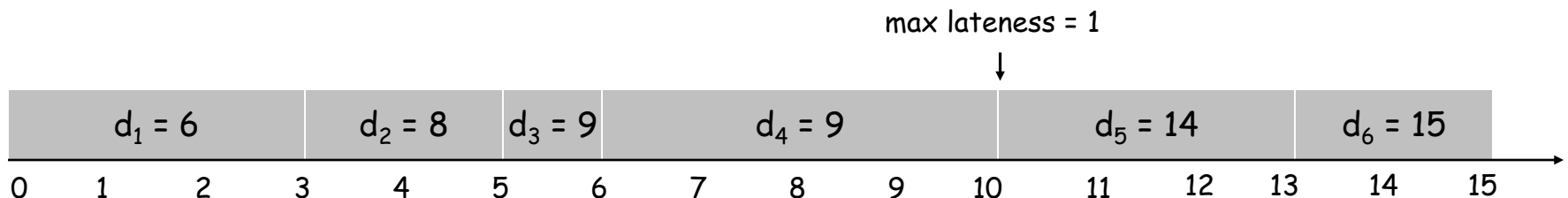
	1	2
t_j	1	10
d_j	2	10

counterexample

Minimizing Lateness: Greedy Algorithm

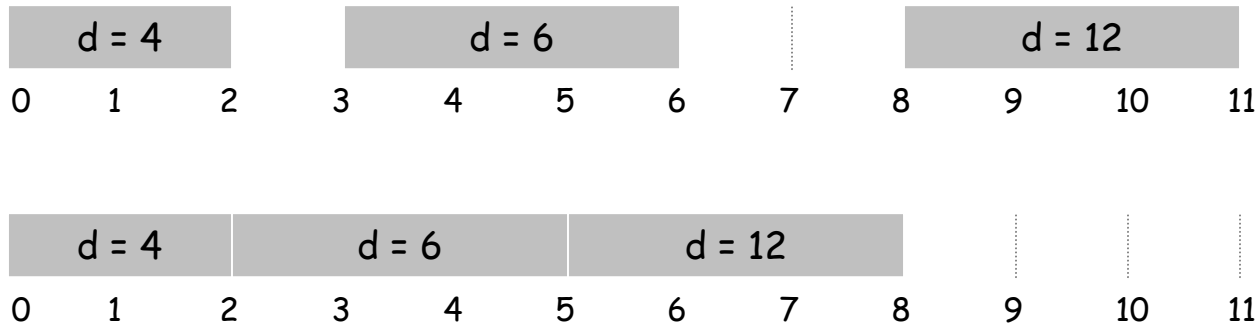
Greedy algorithm. Earliest deadline first.

```
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  $[s_j, f_j]$ 
```



Minimizing Lateness: No Idle Time

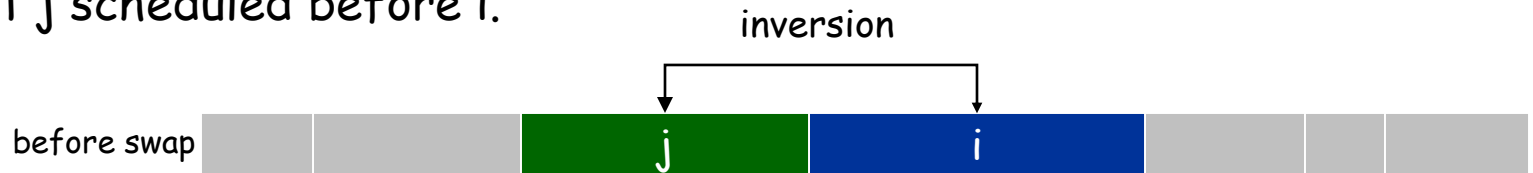
Observation. There exists an optimal schedule with no **idle time**.



Observation. The greedy schedule has no idle time.

Minimizing Lateness: Inversions

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

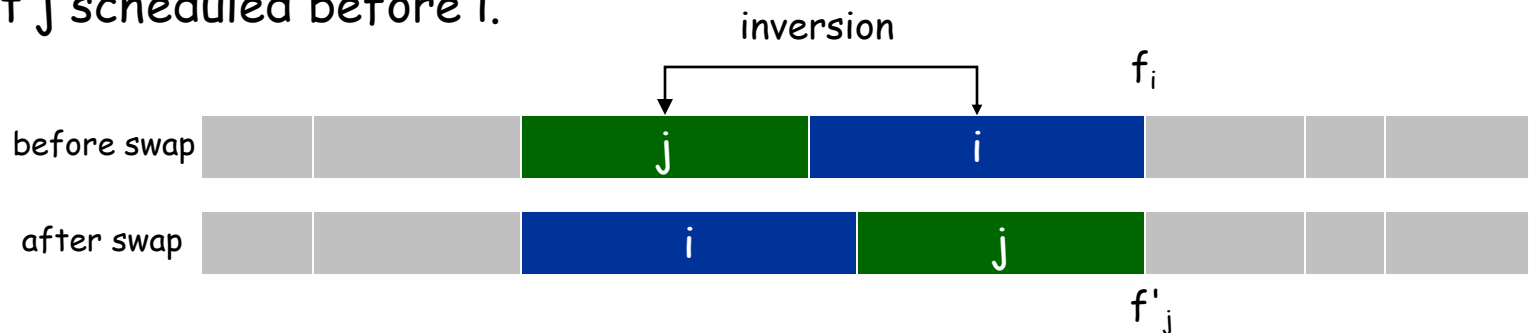


Observation. Greedy schedule has no inversions.

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

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



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 $k \neq i, j$
- $\ell'_i \leq \ell_i$
- If job j is late:

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

Minimizing Lateness: Analysis via Exchange Argument

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^* ▪

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.

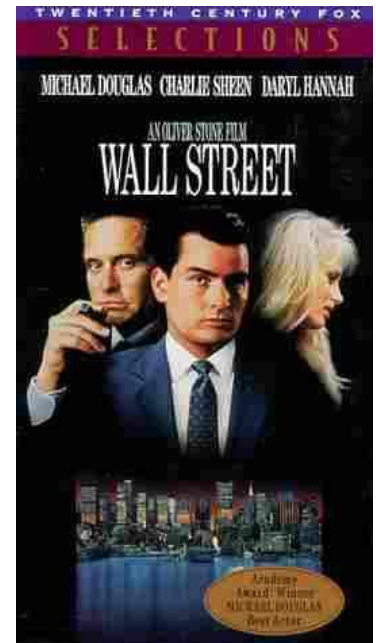
Exchange argument. Gradually transform any solution to the one found by the greedy algorithm without hurting its quality.

Structural. Discover a simple "structural" bound asserting that every possible solution must have a certain value. Then show that your algorithm always achieves this bound.

Coin Changing

*Greed is good. Greed is right. Greed works.
Greed clarifies, cuts through, and captures the
essence of the evolutionary spirit.*

- Gordon Gecko (Michael Douglas)



Coin Changing

Goal. Given currency denominations: 1, 5, 10, 25, 100, devise a method to pay amount to customer using fewest number of coins.

Ex: 34¢.



Cashier's algorithm. At each iteration, add coin of the largest value that does not take us past the amount to be paid.

Ex: \$2.89.



Coin-Changing: Greedy Algorithm

Cashier's algorithm. At each iteration, add coin of the largest value that does not take us past the amount to be paid.

```
Sort coins denominations by value:  $c_1 < c_2 < \dots < c_n$ .
```

```
    ↙ coins selected  
S ←  $\phi$   
while (x ≠ 0) {  
    let k be largest integer such that  $c_k \leq x$   
    if (k = 0)  
        return "no solution found"  
    x ← x -  $c_k$   
    S ← S  $\cup$  {k}  
}  
return S
```

Q. Is cashier's algorithm optimal?

Coin-Changing: Analysis of Greedy Algorithm

Theorem. Greedy is optimal for U.S. coinage: 1, 5, 10, 25, 100.

Pf. (by induction on x)

- Consider optimal way to change $c_k \leq x < c_{k+1}$: greedy takes coin k .
- We claim that any optimal solution must also take coin k .
 - if not, it needs enough coins of type c_1, \dots, c_{k-1} to add up to x
 - table below indicates no optimal solution can do this
- Problem reduces to coin-changing $x - c_k$ cents, which, by induction, is optimally solved by greedy algorithm. ▪

k	c_k	All optimal solutions must satisfy	Max value of coins 1, 2, ..., $k-1$ in any OPT
1	1	$P \leq 4$	-
2	5	$N \leq 1$	4
3	10	$N + D \leq 2$	$4 + 5 = 9$
4	25	$Q \leq 3$	$20 + 4 = 24$
5	100	no limit	$75 + 24 = 99$

Coin-Changing: Analysis of Greedy Algorithm

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.



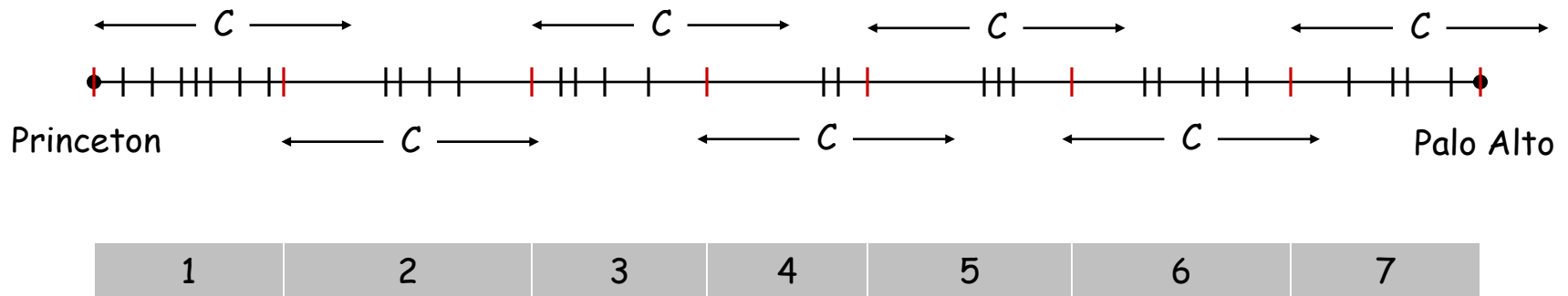
Selecting Breakpoints

Selecting Breakpoints

Selecting breakpoints.

- Road trip from Princeton to Palo Alto along fixed route.
- Refueling stations at certain points along the way.
- Fuel capacity = C .
- Goal: makes as few refueling stops as possible.

Greedy algorithm. Go as far as you can before refueling.



Selecting Breakpoints: Greedy Algorithm

Truck driver's algorithm.

```
Sort breakpoints so that:  $0 = b_0 < b_1 < b_2 < \dots < b_n = L$ 
```

```
 $S \leftarrow \{0\}$   $\leftarrow$  breakpoints selected
```

```
 $x \leftarrow 0$   $\leftarrow$  current location
```

```
while ( $x \neq b_n$ )
```

```
    let  $p$  be largest integer such that  $b_p \leq x + C$ 
```

```
    if ( $b_p = x$ )
```

```
        return "no solution"
```

```
     $x \leftarrow b_p$ 
```

```
     $S \leftarrow S \cup \{p\}$ 
```

```
return  $S$ 
```

Implementation. $O(n \log n)$

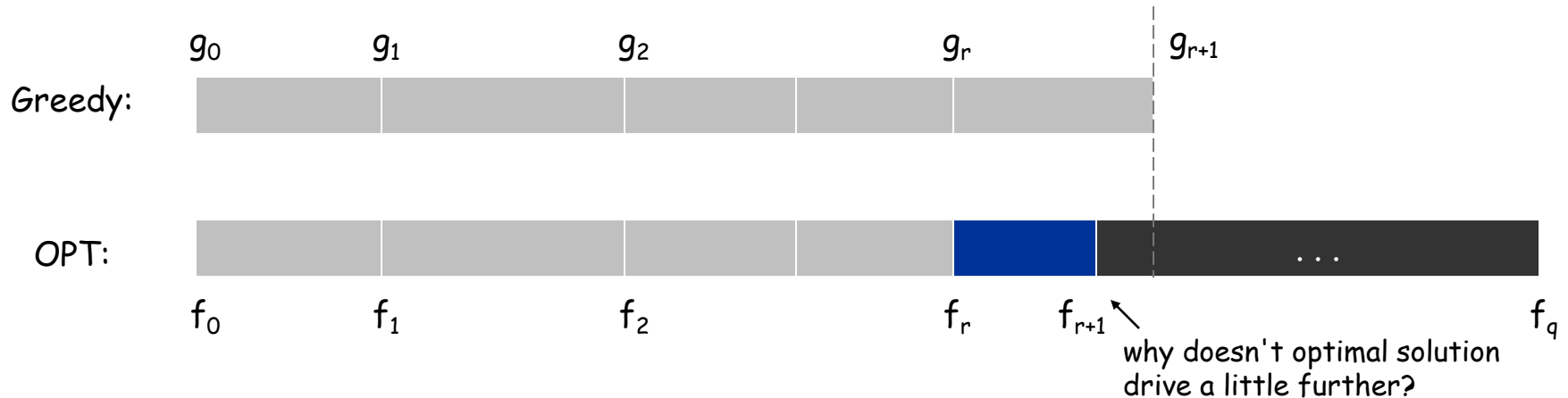
- Use binary search to select each breakpoint p .

Selecting Breakpoints: Correctness

Theorem. Greedy algorithm is optimal.

Pf. (by contradiction)

- Assume greedy is not optimal, and let's see what happens.
- Let $0 = g_0 < g_1 < \dots < g_p = L$ denote set of breakpoints chosen by greedy.
- Let $0 = f_0 < f_1 < \dots < f_q = L$ denote set of breakpoints in an optimal solution with $f_0 = g_0, f_1 = g_1, \dots, f_r = g_r$ for largest possible value of r .
- Note: $g_{r+1} > f_{r+1}$ by greedy choice of algorithm.

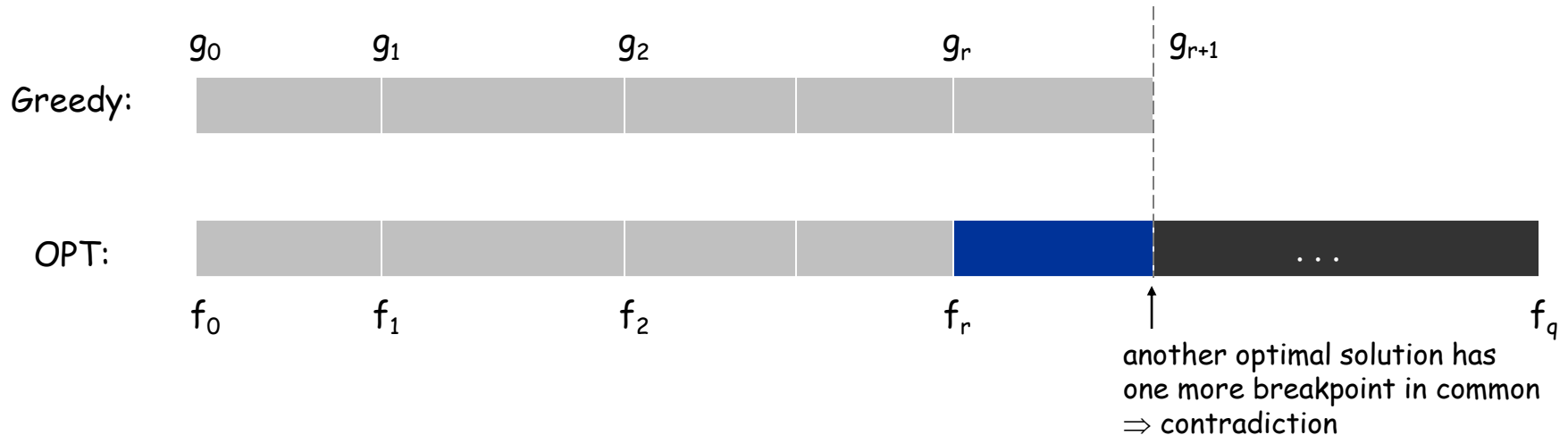


Selecting Breakpoints: Correctness

Theorem. Greedy algorithm is optimal.

Pf. (by contradiction)

- Assume greedy is not optimal, and let's see what happens.
- Let $0 = g_0 < g_1 < \dots < g_p = L$ denote set of breakpoints chosen by greedy.
- Let $0 = f_0 < f_1 < \dots < f_q = L$ denote set of breakpoints in an optimal solution with $f_0 = g_0, f_1 = g_1, \dots, f_r = g_r$ for largest possible value of r .
- Note: $g_{r+1} > f_{r+1}$ by greedy choice of algorithm.

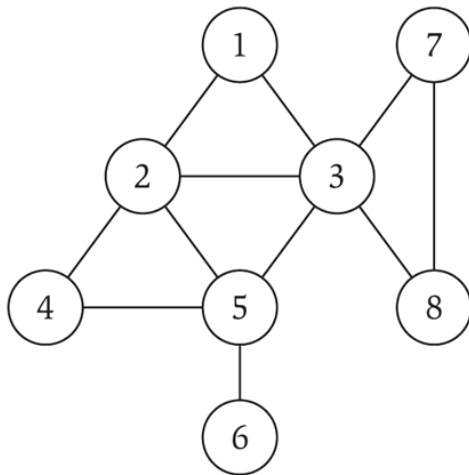


3.1 Graphs: Basic Definitions

Undirected Graphs

Undirected graph. $G = (V, E)$

- V = nodes.
- E = edges between pairs of nodes.
- Captures pairwise relationship between objects.
- Graph size parameters: $n = |V|$, $m = |E|$.



$V = \{ 1, 2, 3, 4, 5, 6, 7, 8 \}$

$E = \{ 1-2, 1-3, 2-3, 2-4, 2-5, 3-5, 3-7, 3-8, 4-5, 5-6 \}$

$n = 8$

$m = 11$

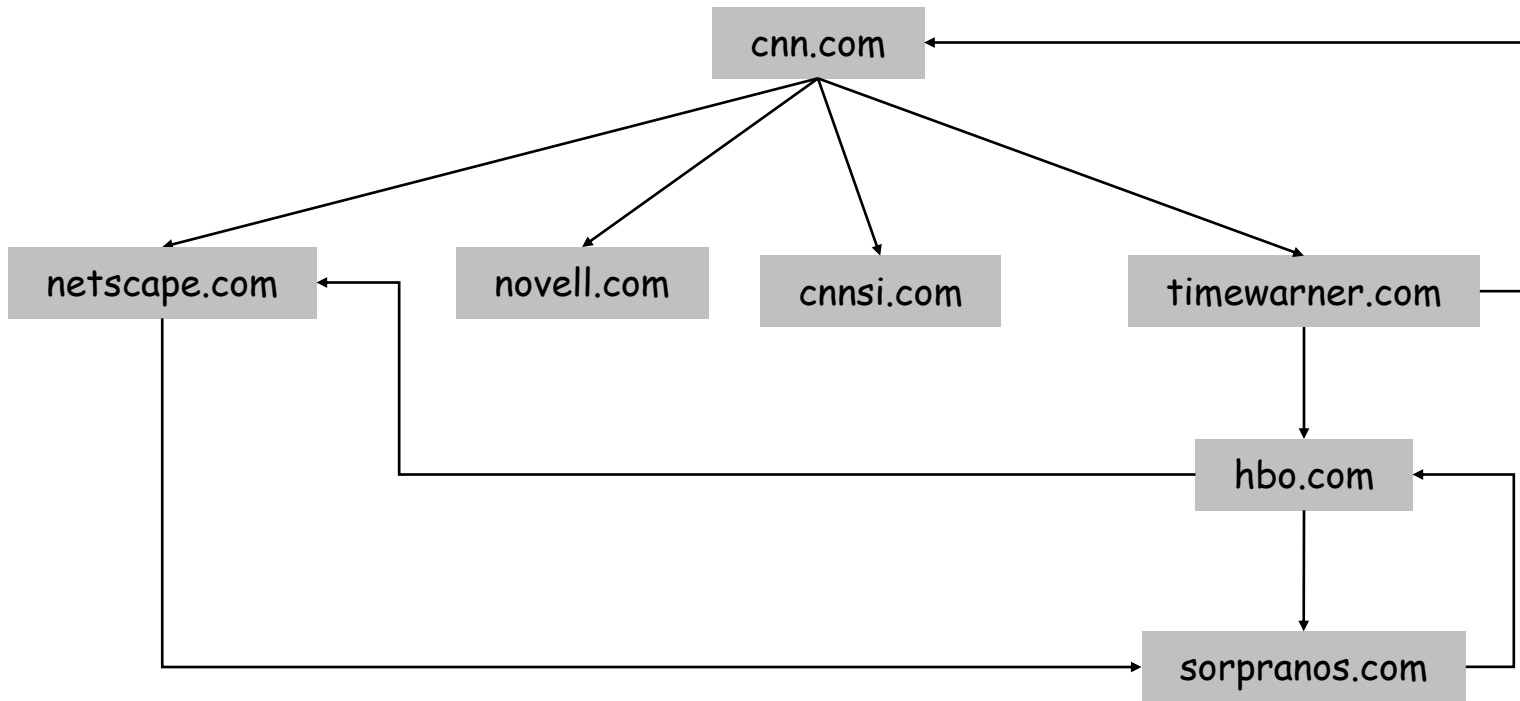
Some Graph Applications

<i>Graph</i>	<i>Nodes</i>	<i>Edges</i>
transportation	street intersections	highways
communication	computers	fiber optic cables
World Wide Web	web pages	hyperlinks
social	people	relationships
food web	species	predator-prey
software systems	functions	function calls
scheduling	tasks	precedence constraints
circuits	gates	wires

World Wide Web

Web graph.

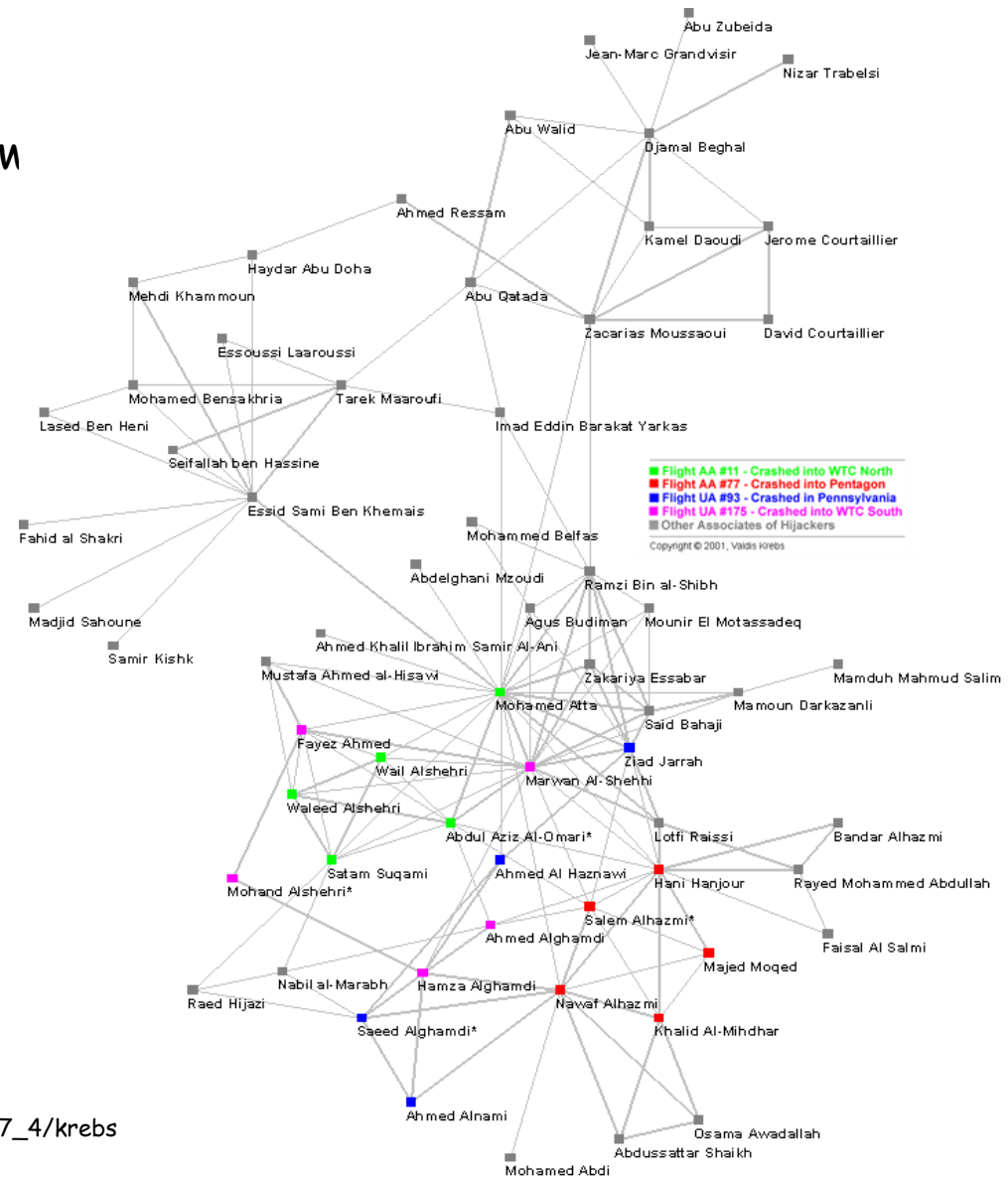
- Node: web page.
- Edge: hyperlink from one page to another.



9-11 Terrorist Network

Social network graph.

- Node: people.
- Edge: relationship between two

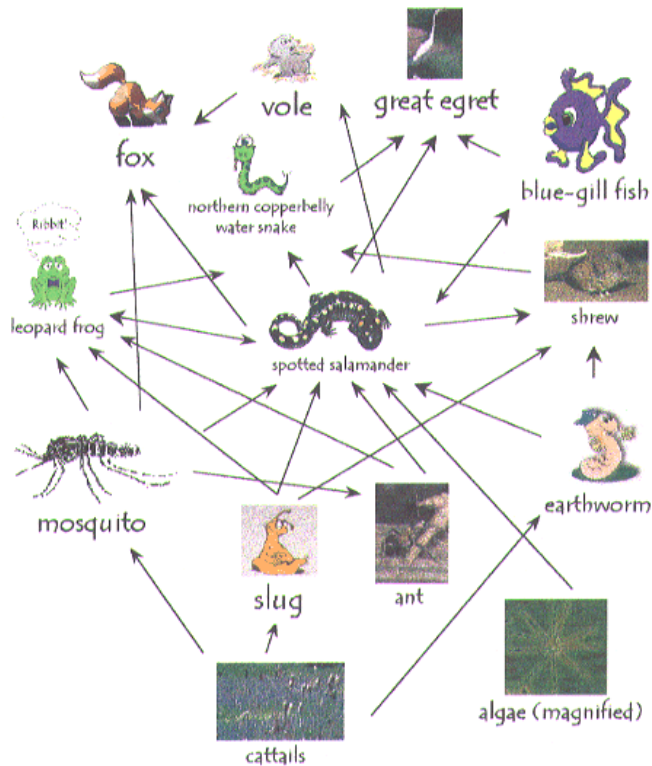


Reference: Valdis Krebs, http://www.firstmonday.org/issues/issue7_4/krebs

Ecological Food Web

Food web graph.

- Node = species.
- Edge = from prey to predator.

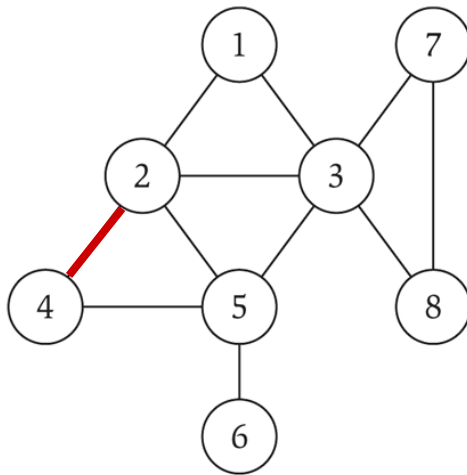


Reference: <http://www.twingroves.district96.k12.il.us/Wetlands/Salamander/SalGraphics/salfoodweb.gif>

Graph Representation: Adjacency Matrix

Adjacency matrix. n -by- n matrix with $A_{uv} = 1$ if (u, v) is an edge.

- Two representations of each edge.
- Space proportional to n^2 .
- Checking if (u, v) is an edge takes $\Theta(1)$ time.
- Identifying all edges takes $\Theta(n^2)$ time.



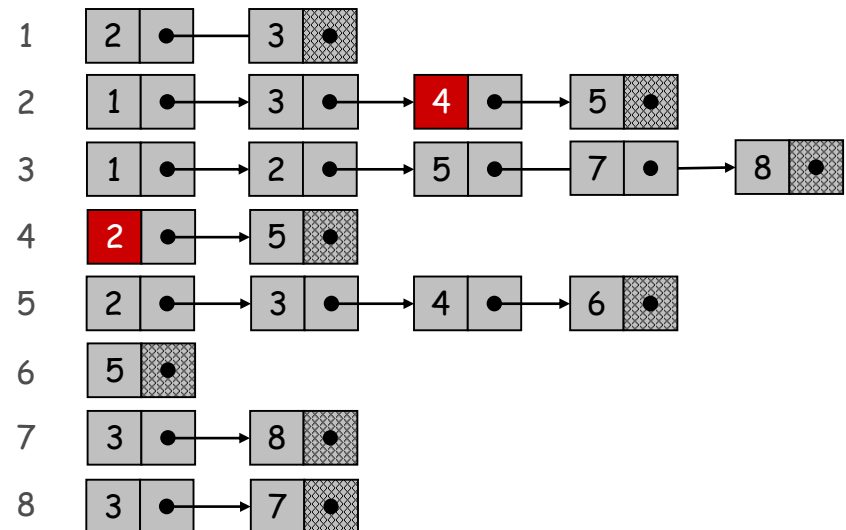
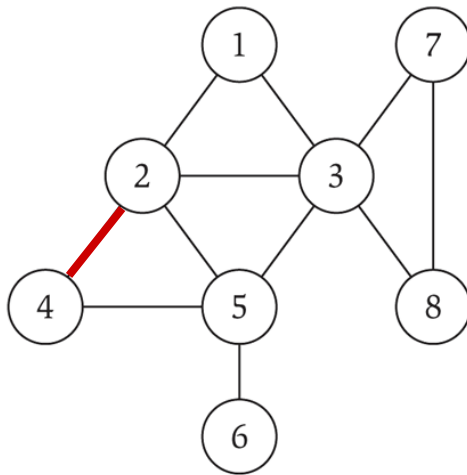
	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	1	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0

Graph Representation: Adjacency List

Adjacency list. Node indexed array of lists.

- Two representations of each edge.
- Space proportional to $m + n$.
- Checking if (u, v) is an edge takes $O(\deg(u))$ time.
- Identifying all edges takes $\Theta(m + n)$ time.

degree = number of neighbors of u

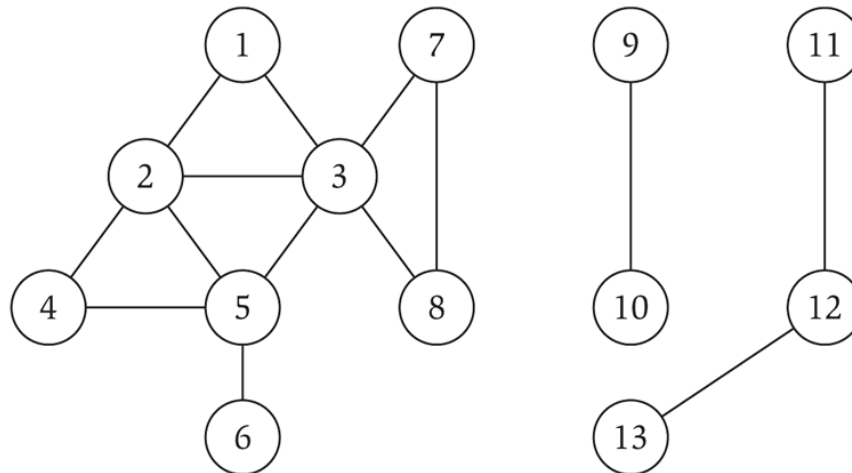


Paths and Connectivity

Def. A **path** in an undirected graph $G = (V, E)$ is a sequence P of nodes $v_1, v_2, \dots, v_{k-1}, v_k$ with the property that each consecutive pair v_i, v_{i+1} is joined by an edge in E .

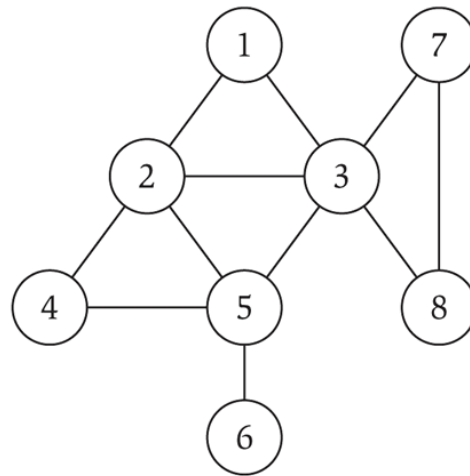
Def. A path is **simple** if all nodes are distinct.

Def. An undirected graph is **connected** if for every pair of nodes u and v , there is a path between u and v .



Cycles

Def. A **cycle** is a path $v_1, v_2, \dots, v_{k-1}, v_k$ in which $v_1 = v_k$, $k > 2$, and the first $k-1$ nodes are all distinct.



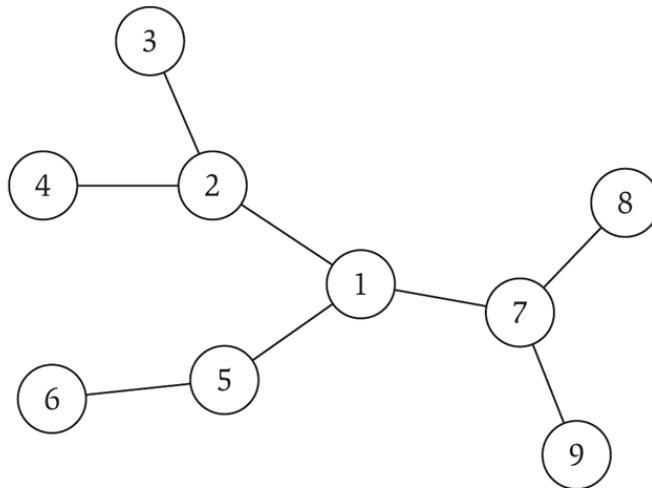
cycle $C = 1-2-4-5-3-1$

Trees

Def. An undirected graph is a **tree** if it is connected and does not contain a cycle.

Theorem. Let G be an undirected graph on n nodes. Any two of the following statements imply the third.

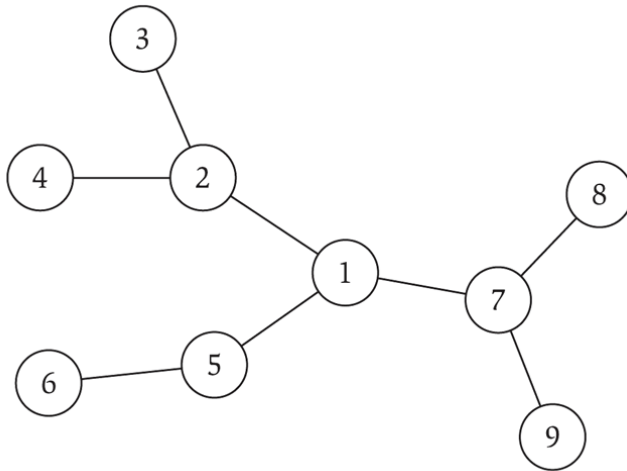
- G is connected.
- G does not contain a cycle.
- G has $n-1$ edges.



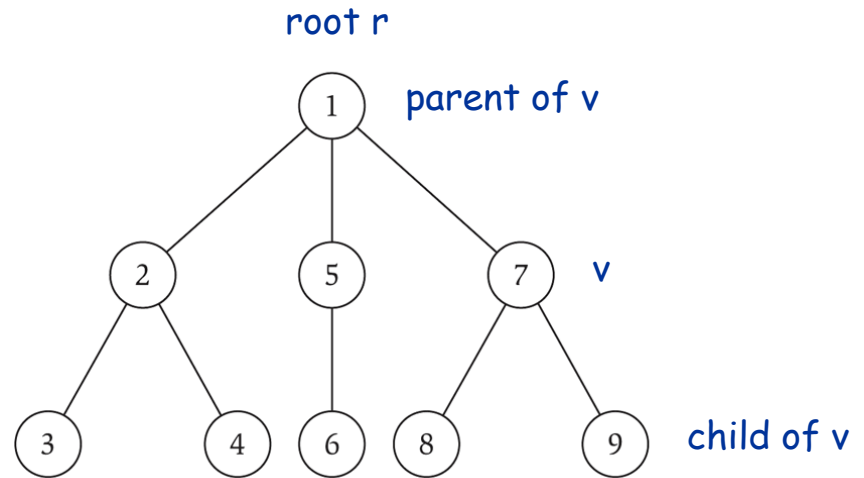
Rooted Trees

Rooted tree. Given a tree T , choose a root node r and orient each edge away from r .

Importance. Models hierarchical structure.



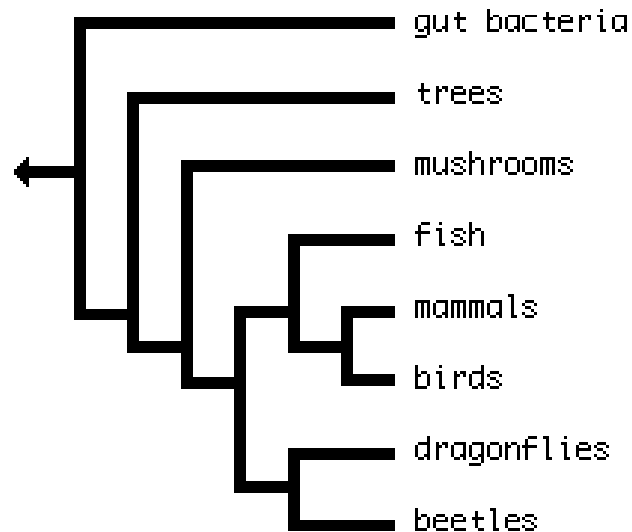
a tree



the same tree, rooted at 1

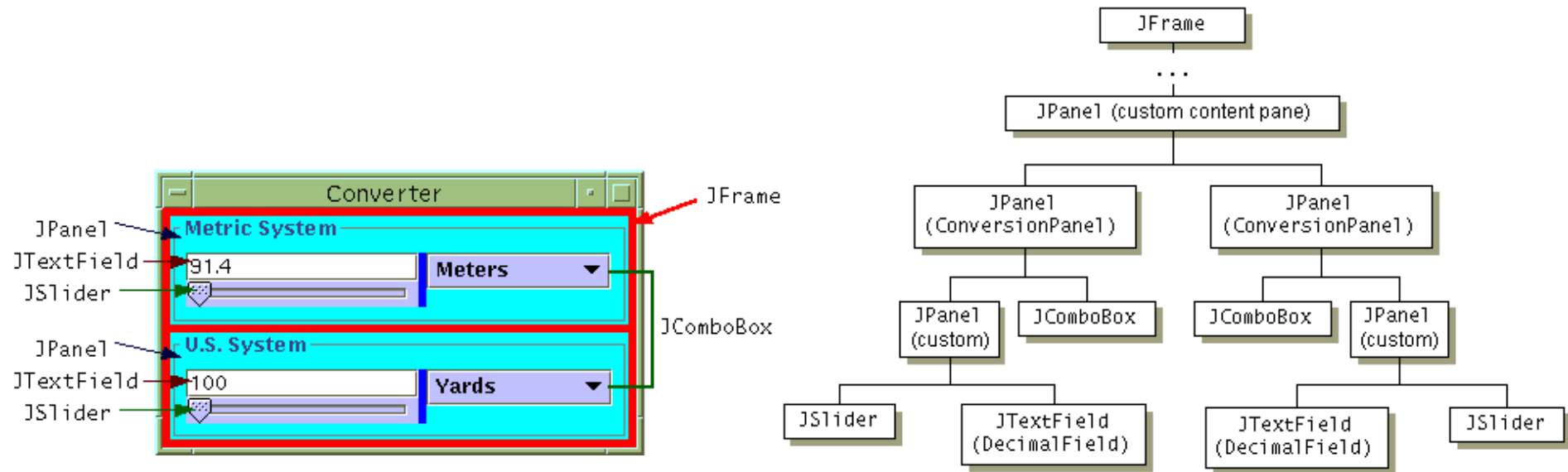
Phylogeny Trees

Phylogeny trees. Describe evolutionary history of species.



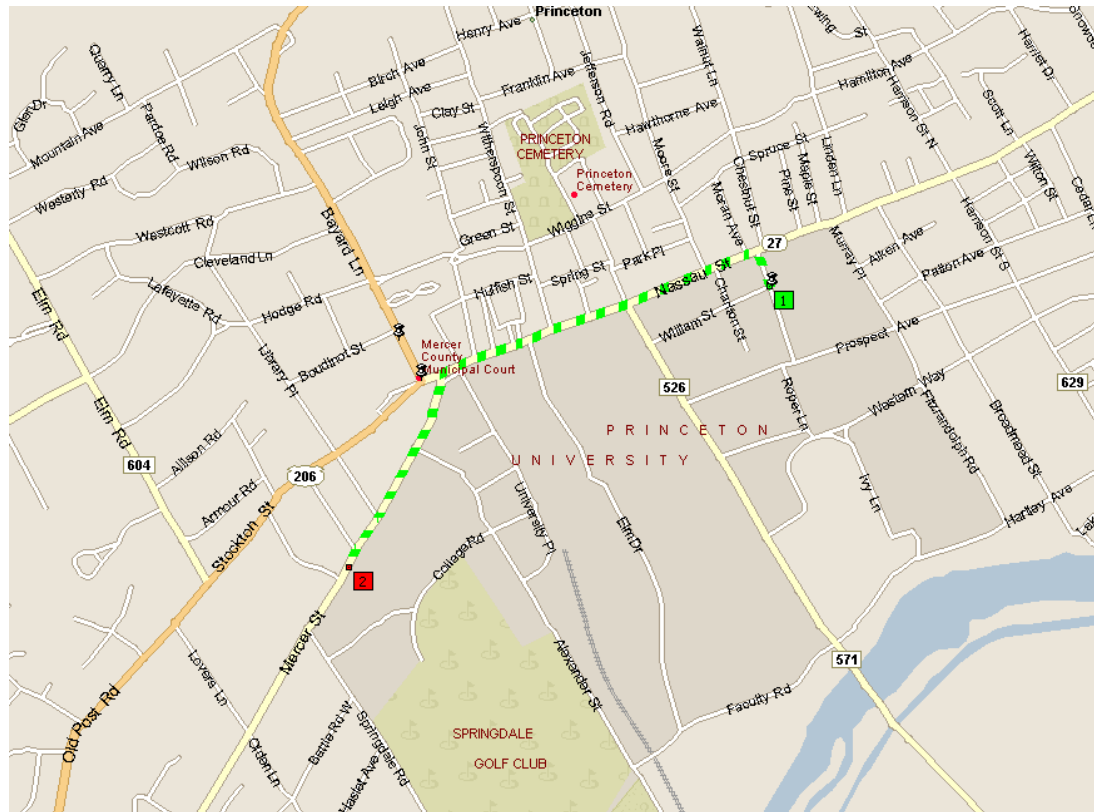
GUI Containment Hierarchy

GUI containment hierarchy. Describe organization of GUI widgets.



Reference: <http://java.sun.com/docs/books/tutorial/uiswing/overview/anatomy.html>

4.4 Shortest Paths in a Graph



shortest path from Princeton CS department to Einstein's house

Edsger W. Dijkstra

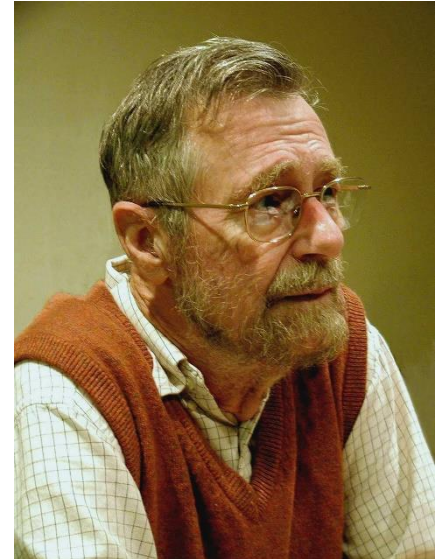
The question of whether computers can think is like the question of whether submarines can swim.

Do only what only you can do.

In their capacity as a tool, computers will be but a ripple on the surface of our culture. In their capacity as intellectual challenge, they are without precedent in the cultural history of mankind.

The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offence.

APL is a mistake, carried through to perfection. It is the language of the future for the programming techniques of the past: it creates a new generation of coding bums.



Shortest Path Problem

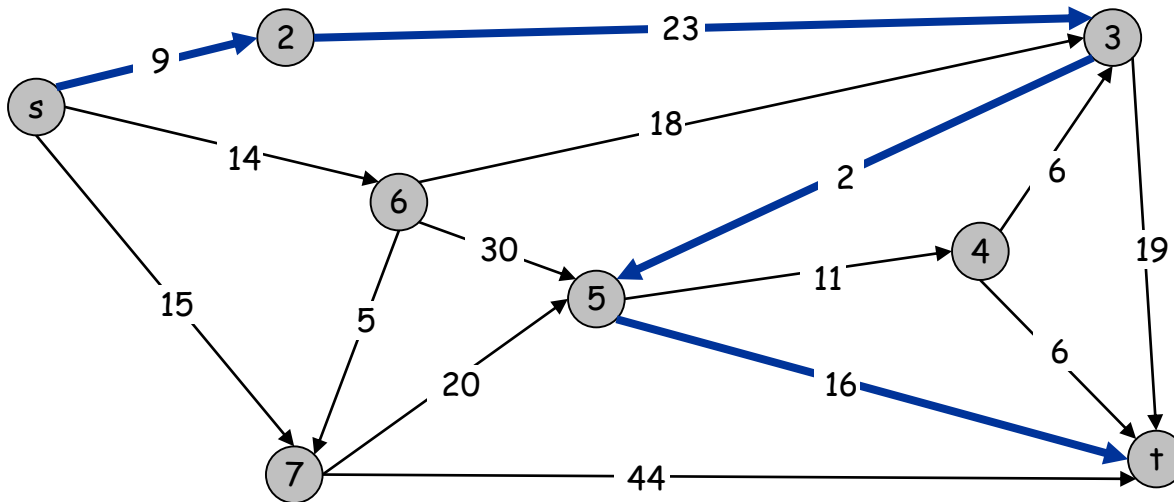
Shortest path network.

- 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 $s-2-3-5-t$
= $9 + 23 + 2 + 16$
= 48.

Dijkstra's Algorithm

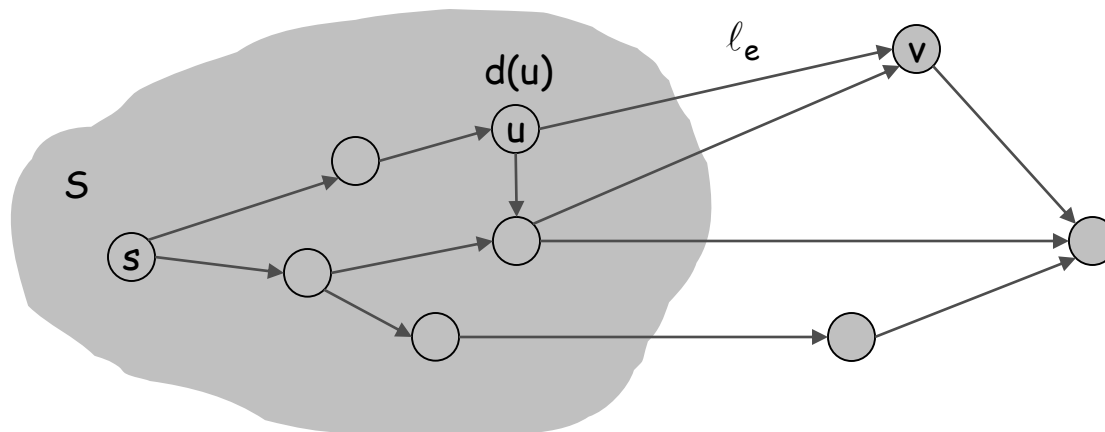
Dijkstra's algorithm.

- Maintain a set of **explored nodes** S for which we have determined the shortest path distance $d(u)$ from s to u .
- Initialize $S = \{s\}$, $d(s) = 0$.
- Repeatedly choose unexplored node v which minimizes

$$\pi(v) = \min_{e = (u,v) : u \in S} d(u) + \ell_e,$$

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)



Dijkstra's Algorithm

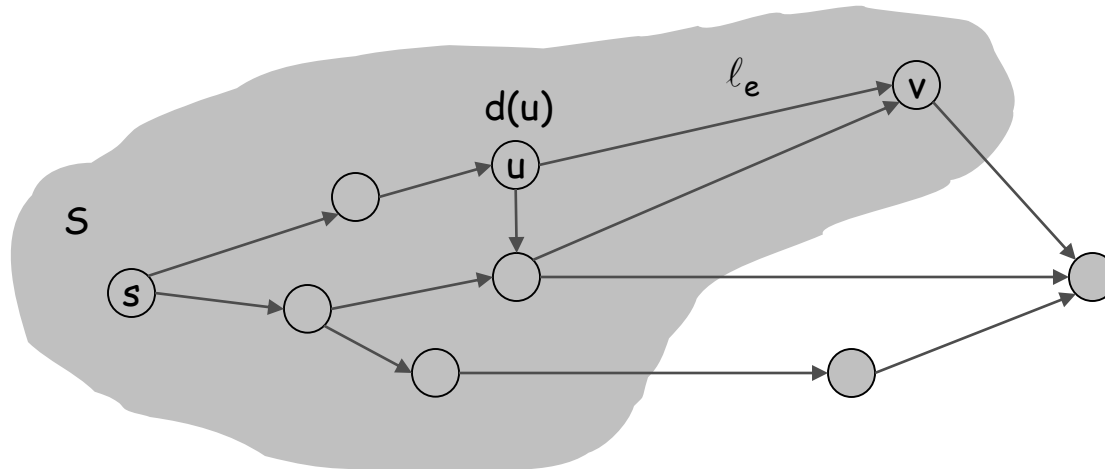
Dijkstra's algorithm.

- Maintain a set of **explored nodes** S for which we have determined the shortest path distance $d(u)$ from s to u .
- Initialize $S = \{s\}$, $d(s) = 0$.
- Repeatedly choose unexplored node v which minimizes

$$\pi(v) = \min_{e = (u,v) : u \in S} d(u) + \ell_e,$$

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)



Dijkstra's Algorithm: Proof of Correctness

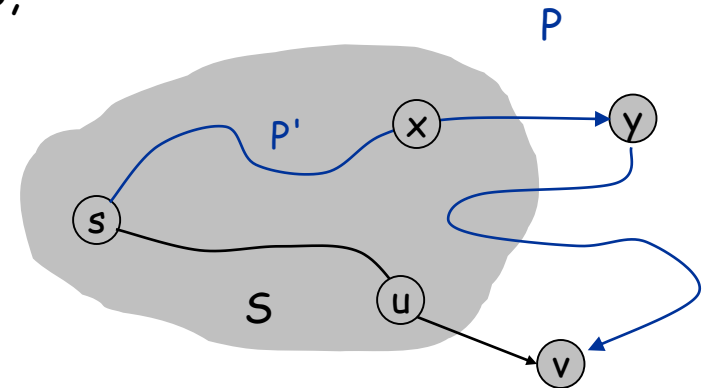
Invariant. For each node $u \in S$, $d(u)$ is the length of the shortest s - u path.

Pf. (by induction on $|S|$)

Base case: $|S| = 1$ is trivial.

Inductive hypothesis: Assume true for $|S| = k \geq 1$.

- Let v be next node added to S , and let u - v be the chosen edge.
- The shortest s - u path plus (u, v) is an s - v path of length $\pi(v)$.
- Consider any s - v path P . We'll see that it's no shorter than $\pi(v)$.
- Let x - y be the first edge in P that leaves S , and let P' be the subpath to x .
- P is already too long as soon as it leaves S .



$$\begin{array}{ccccccc} \ell(P) & \geq & \ell(P') + \ell(x, y) & \geq & d(x) + \ell(x, y) & \geq & \pi(y) \geq \pi(v) \\ \uparrow & & \uparrow & & \uparrow & & \uparrow \\ \text{nonnegative} & & \text{inductive} & & \text{defn of } \pi(y) & & \text{Dijkstra chose } v \\ \text{weights} & & \text{hypothesis} & & & & \text{instead of } y \end{array}$$

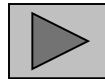
Dijkstra's Algorithm: Implementation

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 $\pi(v)$.



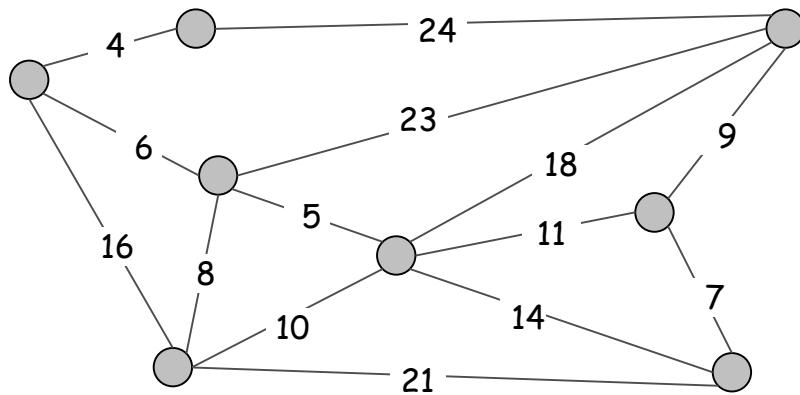
PQ Operation	Dijkstra	Array	Binary heap	d-way Heap	Fib heap [†]
Insert	n	n	$\log n$	$d \log_d n$	1
ExtractMin	n	n	$\log n$	$d \log_d n$	$\log n$
ChangeKey	m	1	$\log n$	$\log_d n$	1
IsEmpty	n	1	1	1	1
Total		n^2	$m \log n$	$m \log_{m/n} n$	$m + n \log n$

[†] Individual ops are amortized bounds

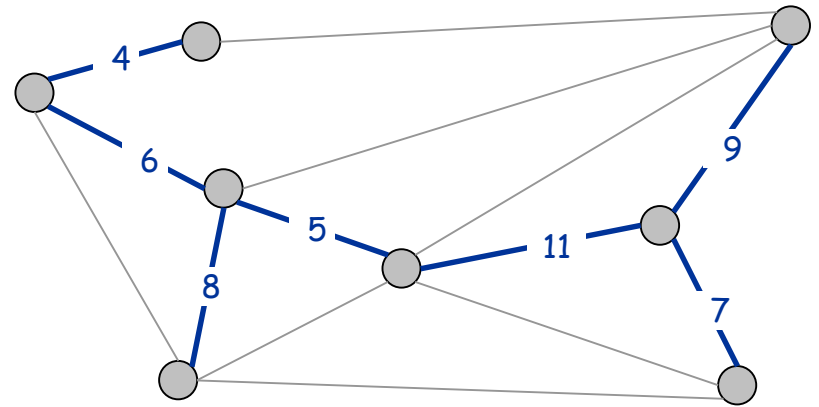
4.5 Minimum Spanning Tree

Minimum Spanning Tree

Minimum spanning tree. 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.



$G = (V, E)$



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

Cayley's Theorem. There are n^{n-2} spanning trees of K_n .

↑
can't solve by brute force

Applications

MST is fundamental problem with diverse applications.

- Network design.
 - telephone, electrical, hydraulic, TV cable, computer, road
- Approximation algorithms for NP-hard problems.
 - traveling salesperson problem, Steiner tree
- Indirect applications.
 - 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
- Cluster analysis.

Greedy Algorithms

Kruskal's algorithm. Start with $T = \emptyset$. Consider edges in ascending order of cost. Insert edge e in T unless doing so would create a cycle.

Reverse-Delete algorithm. Start with $T = E$. Consider edges in descending order of cost. Delete edge e from T unless doing so would disconnect T .

Prim's algorithm. Start with some root node s and greedily grow a tree T from s outward. At each step, add the cheapest edge e to T that has exactly one endpoint in T .

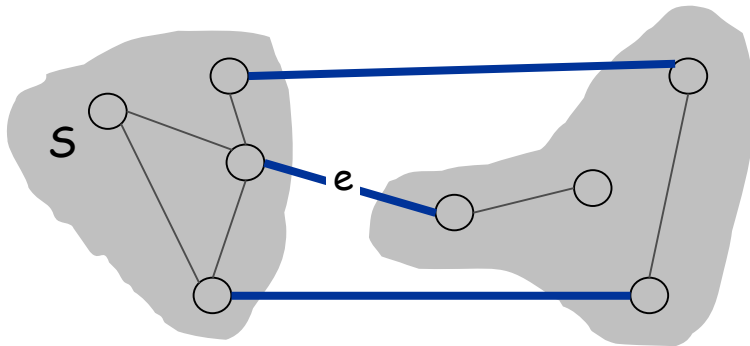
Remark. All three algorithms produce an MST.

Greedy Algorithms

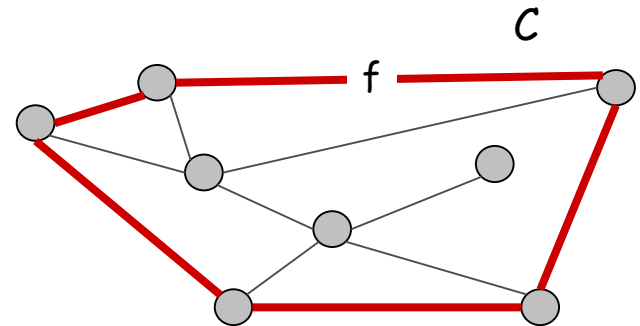
Simplifying assumption. All edge costs c_e are distinct.

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 MST contains e .

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



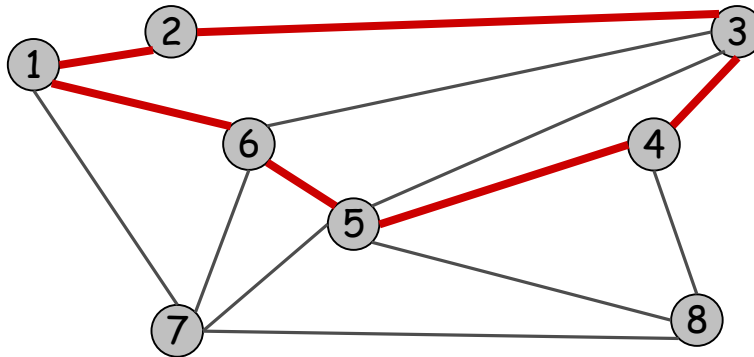
e is in the MST



f is not in the MST

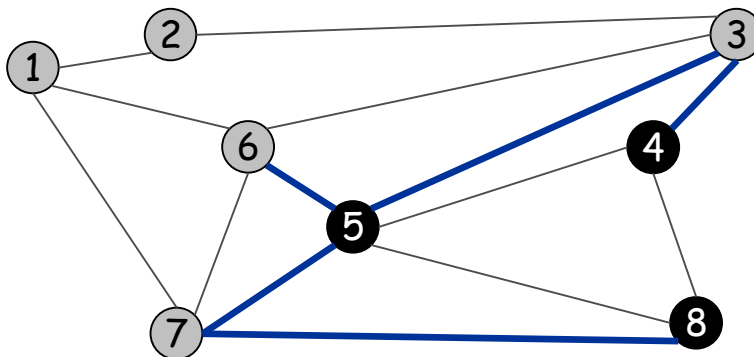
Cycles and Cuts

Cycle. Set of edges the form $a-b, b-c, c-d, \dots, y-z, z-a$.



Cycle $C = 1-2, 2-3, 3-4, 4-5, 5-6, 6-1$

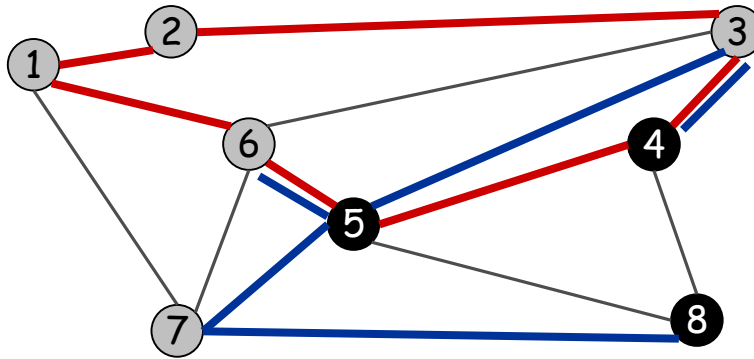
Cutset. A cut is a subset of nodes S . The corresponding cutset D is the subset of edges with exactly one endpoint in S .



Cut $S = \{4, 5, 8\}$
Cutset $D = 5-6, 5-7, 3-4, 3-5, 7-8$

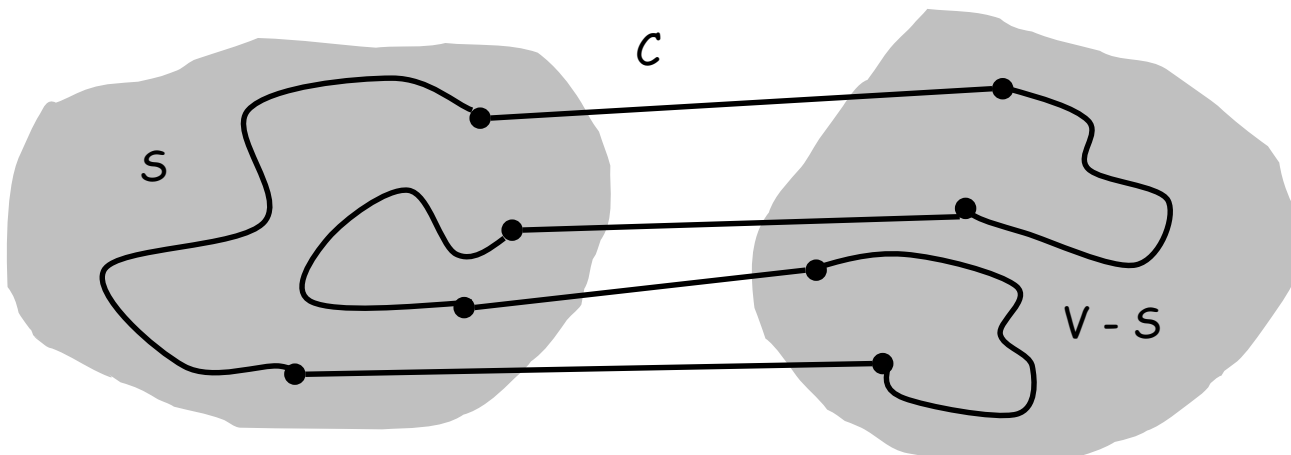
Cycle-Cut Intersection

Claim. A cycle and a cutset intersect in an even number of edges.



Cycle $C = 1-2, 2-3, 3-4, 4-5, 5-6, 6-1$
Cutset $D = 3-4, 3-5, 5-6, 5-7, 7-8$
Intersection = $3-4, 5-6$

Pf. (by picture)



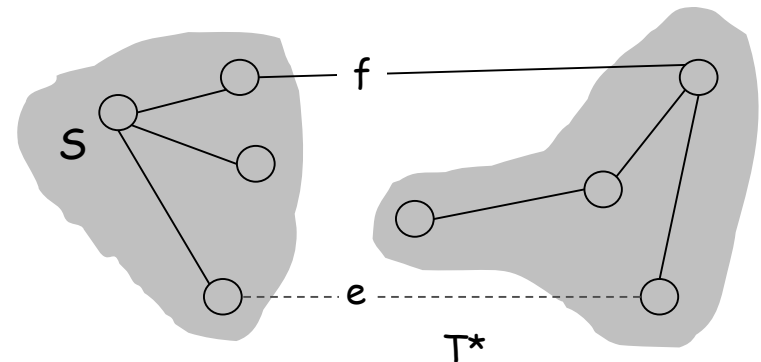
Greedy Algorithms

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 MST T^* contains e .

Pf. (exchange argument)

- Suppose e does not belong to T^* , and let's see what happens.
- Adding e to T^* creates a cycle C in T^* .
- Edge e is both in the cycle C and in the cutset D corresponding to S
 \Rightarrow there exists another edge, say f , that is in both C and D .
- $T' = T^* \cup \{e\} - \{f\}$ is also a spanning tree.
- Since $c_e < c_f$, $\text{cost}(T') < \text{cost}(T^*)$.
- This is a contradiction. ▪



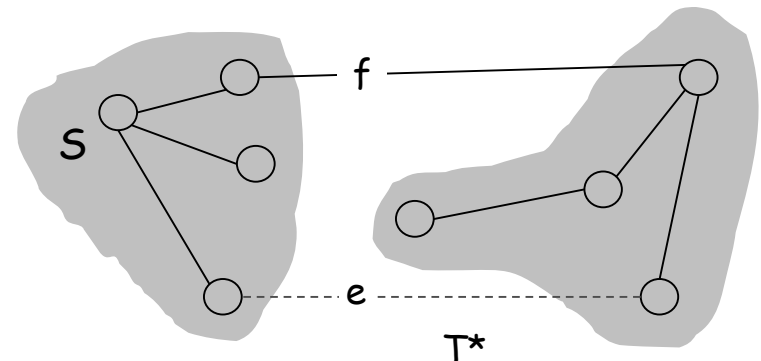
Greedy Algorithms

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. (exchange argument)

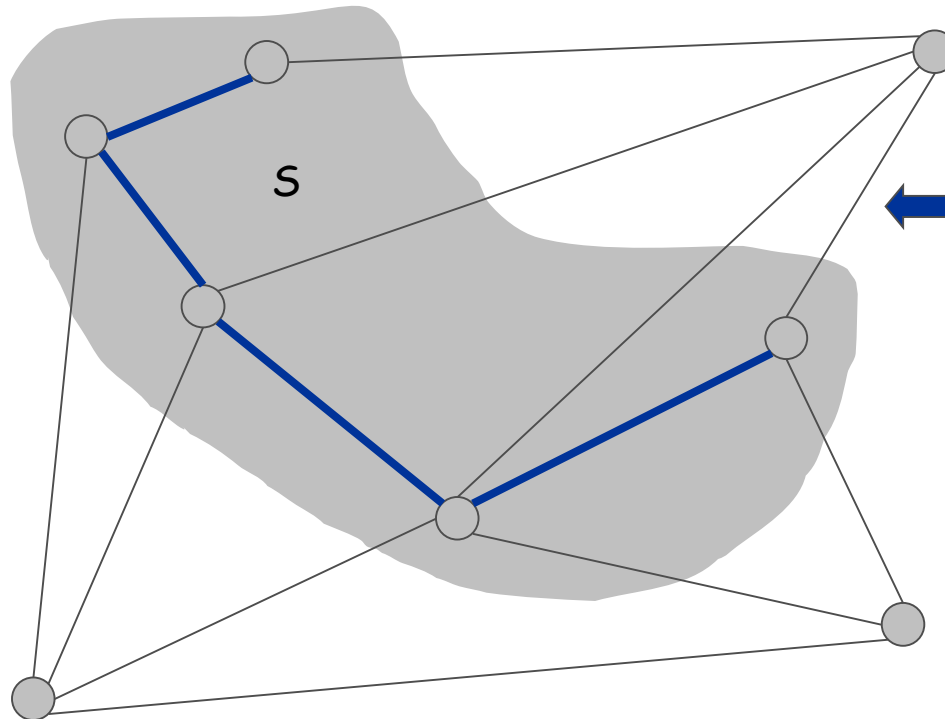
- Suppose f belongs to T^* , and let's see what happens.
- Deleting f from T^* creates a cut S in T^* .
- Edge f is both in the cycle C and in the cutset D corresponding to S
 \Rightarrow there exists another edge, say e , that is in both C and D .
- $T' = T^* \cup \{e\} - \{f\}$ is also a spanning tree.
- Since $c_e < c_f$, $\text{cost}(T') < \text{cost}(T^*)$.
- This is a contradiction. ▪



Prim's Algorithm: Proof of Correctness

Prim's algorithm. [Jarník 1930, Dijkstra 1957, Prim 1959]

- Initialize S = any node.
- Apply cut property to S .
- Add min cost edge in cutset corresponding to S to T , and add one new explored node u to S .



Implementation: Prim's Algorithm

Implementation. Use a priority queue ala Dijkstra.

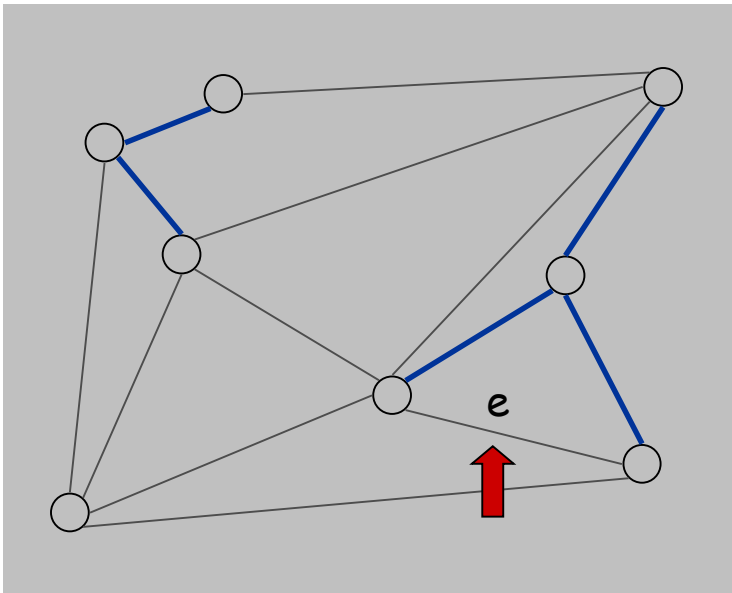
- Maintain set of explored nodes S .
- For each unexplored node v , maintain attachment cost $a[v]$ = cost of cheapest edge v to a node in S .
- $O(n^2)$ with an array; $O(m \log n)$ with a binary heap.

```
Prim(G, c) {  
    foreach ( $v \in V$ )  $a[v] \leftarrow \infty$   
    Initialize an empty priority queue  $Q$   
    foreach ( $v \in V$ ) insert  $v$  onto  $Q$   
    Initialize set of explored nodes  $S \leftarrow \phi$   
  
    while ( $Q$  is not empty) {  
         $u \leftarrow$  delete min element from  $Q$   
         $S \leftarrow S \cup \{u\}$   
        foreach (edge  $e = (u, v)$  incident to  $u$ )  
            if ( $(v \notin S)$  and ( $c_e < a[v]$ ))  
                decrease priority  $a[v]$  to  $c_e$   
    }  
}
```

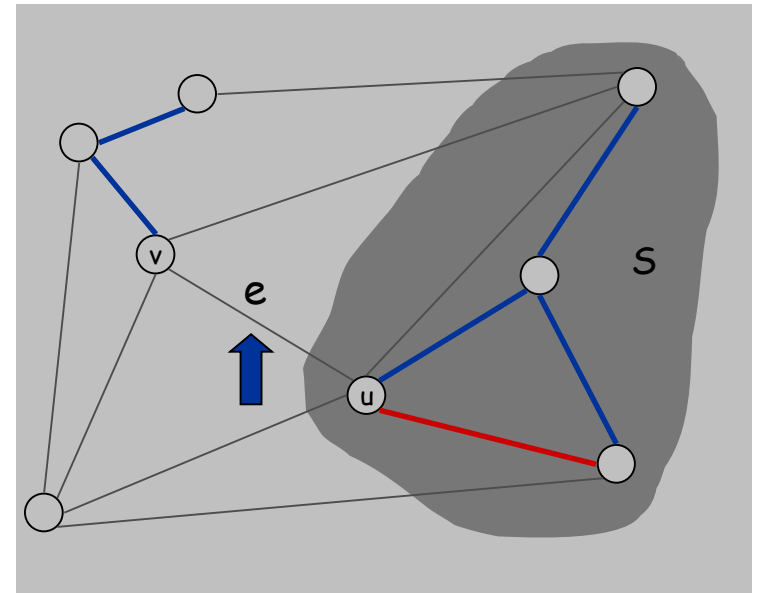
Kruskal's Algorithm: Proof of Correctness

Kruskal's algorithm. [Kruskal, 1956]

- Consider edges in ascending order of weight.
- Case 1: If adding e to T creates a cycle, discard e according to cycle property.
- Case 2: Otherwise, insert $e = (u, v)$ into T according to cut property where S = set of nodes in u 's connected component.



Case 1



Case 2

Implementation: Kruskal's Algorithm

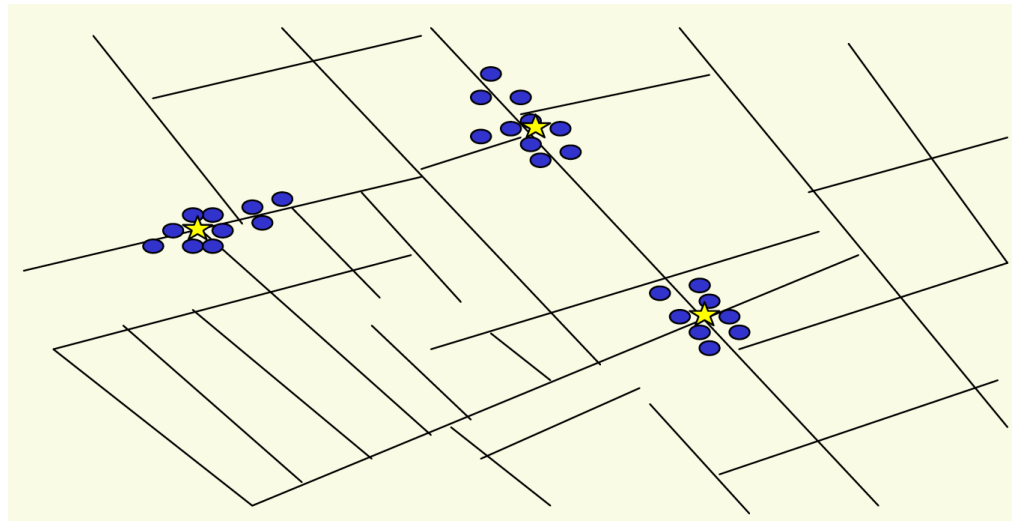
Implementation. Use the **union-find** data structure.

- Build set T of edges in the MST.
- Maintain set for each connected component.
- $O(m \log n)$ for sorting and $O(m \underbrace{\alpha(m, n)}_{\text{essentially a constant}})$ for union-find.

\swarrow $m \leq n^2 \Rightarrow \log m$ is $O(\log n)$ \searrow essentially a constant

```
Kruskal(G, c) {  
    Sort edges weights so that  $c_1 \leq c_2 \leq \dots \leq c_m$ .  
     $T \leftarrow \phi$   
  
    foreach ( $u \in V$ ) make a set containing singleton  $u$   
  
    for  $i = 1$  to  $m$     are  $u$  and  $v$  in different connected components?  
         $(u, v) = e_i$   $\swarrow$   
        if ( $u$  and  $v$  are in different sets) {  
             $T \leftarrow T \cup \{e_i\}$   
            merge the sets containing  $u$  and  $v$   
        }  $\nwarrow$  merge two components  
    return  $T$   
}
```

4.7 Clustering



Outbreak of cholera deaths in London in 1850s.
Reference: Nina Mishra, HP Labs

Clustering

Clustering. Given a set U of n objects labeled p_1, \dots, p_n , classify into coherent groups.

↑
photos, documents, micro-organisms

Distance function. Numeric value specifying "closeness" of two objects.

↑
number of corresponding pixels whose intensities differ by 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

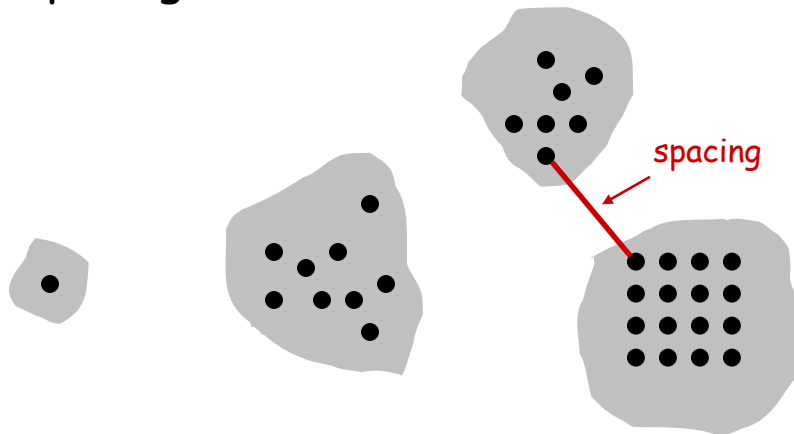
k-clustering. Divide objects into k non-empty groups.

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)

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

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



Greedy Clustering Algorithm

Single-link k-clustering algorithm.

- Form a graph on the vertex set U , corresponding to n clusters.
- Find the closest pair of objects such that each object is in a different cluster, and add an edge between them.
- Repeat $n-k$ times until there are exactly k clusters.

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

Remark. Equivalent to finding an MST and deleting the $k-1$ most expensive edges.

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 .

- The spacing of C^* is the length d^* of the $(k-1)^{\text{st}}$ most expensive edge.
- 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 .
- 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 $\leq d^*$ since Kruskal chose them.
- Spacing of C is $\leq d^*$ since p and q are in different clusters. ▪

