Midterm I Review

- *stable matching*
- *complexity analysis*
- *graph algorithms*
- *greedy algorithms*

Midterm I Review

- *stable matching*
- *complexity analysis*
- *graph algorithms*
- *greedy algorithms*

Stable matching

Goal. Given a set of preferences among hospitals and med-school students, design a self-reinforcing admissions process.

Unstable pair. Hospital h and student s form an unstable pair if both:

- h prefers s to one of its admitted students.
- s prefers h to assigned hospital.

Def. A matching M is a set of ordered pairs h–s with $h \in H$ and $s \in S$ s.t.

- Each hospital $h \in H$ appears in at most one pair of M.
- Each student $s \in S$ appears in at most one pair of M.

Def. A matching M is perfect if $|M| = |H| = |S| = n$.

Def. A stable matching is a perfect matching with no unstable pairs.

Stable matching

An intuitive method that guarantees to find a stable matching.

GALE–SHAPLEY (*preference lists for hospitals and students*)

INITIALIZE M to empty matching.

WHILE (some hospital h is unmatched and hasn't proposed to every student)

  $s \leftarrow$ first student on h's list to whom h has not yet proposed.

  IF (s is unmatched)

    Add h–s to matching M.

  ELSE IF (s prefers h to current partner h')

    Replace h'–s with h–s in matching M.

  ELSE

    s rejects h.

RETURN stable matching M.

Stable matching

Theorem. [Gale–Shapley 1962] The Gale–Shapley algorithm guarantees to find a stable matching for any problem instance.

Def. Student s is a valid partner for hospital h if there exists any stable matching in which h and s are matched.

Hospital-optimal assignment. Each hospital receives best valid partner.

- Is it a perfect matching?
- Is it stable?

Claim. All executions of Gale–Shapley yield hospital-optimal assignment.

Corollary. Hospital-optimal assignment is a stable matching!

Student-pessimal assignment. Each student receives worst valid partner.

Midterm I Review

- *stable matching*
- *complexity analysis*
- *graph algorithms*
- *greedy algorithms*

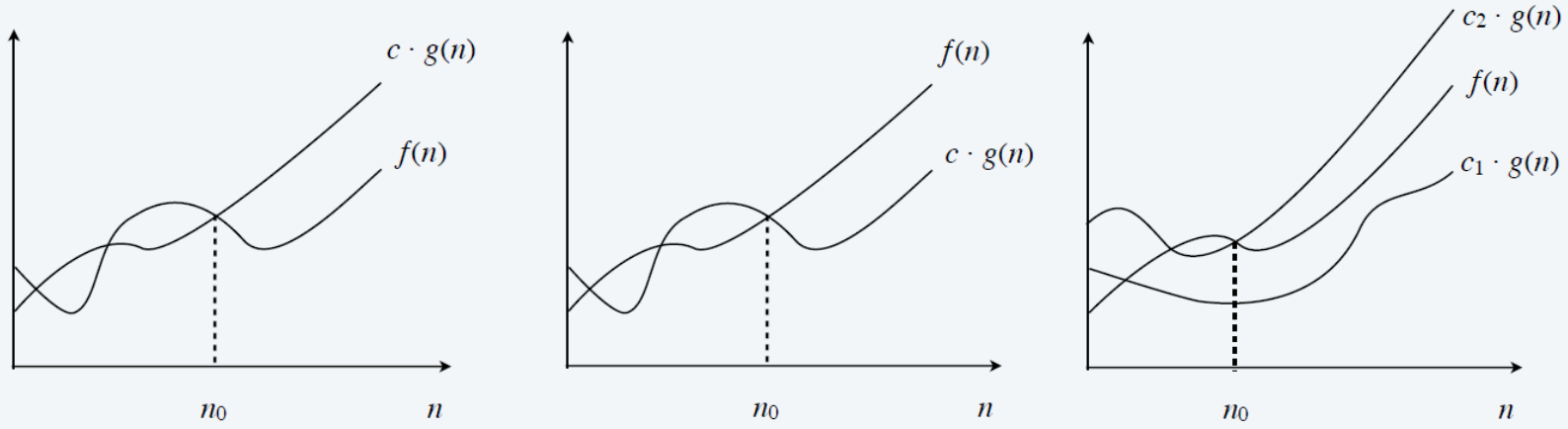
Complexity analysis

Worst case. Running time guarantee for any input of size n.

Upper bounds. $f(n)$ is $O(g(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that $0 \leq f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.

Lower bounds. $f(n)$ is $\Omega(g(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that $f(n) \geq c \cdot g(n) \geq 0$ for all $n \geq n_0$.

Tight bounds. $f(n)$ is $\Theta(g(n))$ if there exist constants $c_1 > 0$, $c_2 > 0$, and $n_0 \geq 0$ such that $0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ for all $n \geq n_0$.



Complexity analysis

Proposition. If $\lim_{n \to \infty} \frac{f(n)}{g(n)} = c$ for some constant $0 < c < \infty$ then $f(n)$ is $\Theta(g(n))$.

Proposition. If $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$, then $f(n)$ is $O(g(n))$ but not $\Omega(g(n))$.

Proposition. If $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$, then $f(n)$ is $\Omega(g(n))$ but not $O(g(n))$.

Polynomials. Let $f(n) = a_0 + a_1 n + \ldots + a_d n^d$ with $a_d > 0$. Then, $f(n)$ is $\Theta(n^d)$.

Logarithms. $\log_a n$ is $\Theta(\log_b n)$ for every $a > 1$ and every $b > 1$.

Logarithms and polynomials. $\log_a n$ is $O(n^d)$ for every $a > 1$ and every $d > 0$.

Exponentials and polynomials. n^d is $O(r^n)$ for every $r > 1$ and every $d > 0$.

Factorials. $n!$ is $2^{\Theta(n \log n)}$.

Complexity analysis

Linear time. Running time is $O(n)$.

Merge two sorted lists. Combine two sorted linked lists $A = a_1, a_2, \ldots, a_n$ and $B = b_1, b_2, \ldots, b_n$ into a sorted whole.

Logarithmic time. Running time is $O(\log n)$.

Search in a sorted array. Given a sorted array A of n distinct integers and an integer x, find index of x in array.

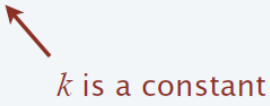
Linearithmic time. Running time is $O(n \log n)$.

Sorting. Given an array of n elements, rearrange them in ascending order.

Complexity analysis

Polynomial time. Running time is $O(n^k)$ for some constant $k > 0$.

Independent set of size k. Given a graph, find k nodes such that no two are joined by an edge.



Exponential time. Running time is $O(2^{n^k})$ for some constant $k > 0$.

Independent set. Given a graph, find independent set of max size.

Midterm I Review

- *stable matching*
- *complexity analysis*
- *graph algorithms*
- *greedy algorithms*

Graph algorithms

Notation. $G = (V, E)$

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.

Adjacency lists. Node-indexed array of lists.

- Two representations of each edge.
- Space is $\Theta(m + n)$.

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

Graph algorithms

Def. A cycle is a path $v_1, v_2, \ldots, v_k$ in which $v_1 = v_k$ and $k \geq 2$.

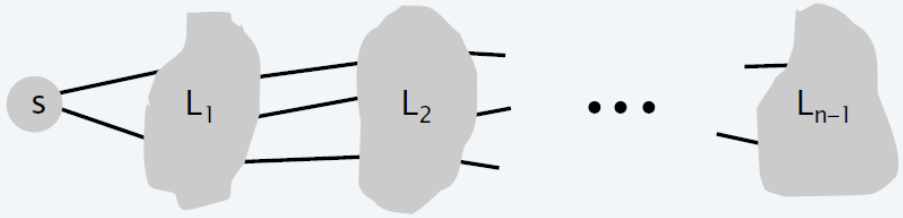
Def. A cycle is simple if all nodes are distinct (except for v_1 and v_k).

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.

BFS intuition. Explore outward from s in all possible directions, adding nodes one "layer" at a time.



BFS algorithm.

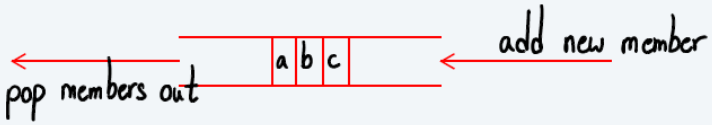
- $L_0 = \{ s \}$.
- L_1 = all neighbors of L_0.
- L_2 = all nodes that do not belong to L_0 or L_1, and that have an edge to a node in L_1.
- L_{i+1} = all nodes that do not belong to an earlier layer, and that have an edge to a node in L_i.

Theorem. For each i, L_i consists of all nodes at distance exactly i from s. There is a path from s to t iff t appears in some layer.

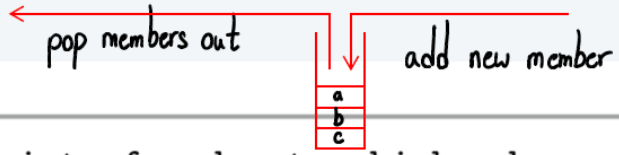
Graph algorithms

Connected component. Find all nodes reachable from s.

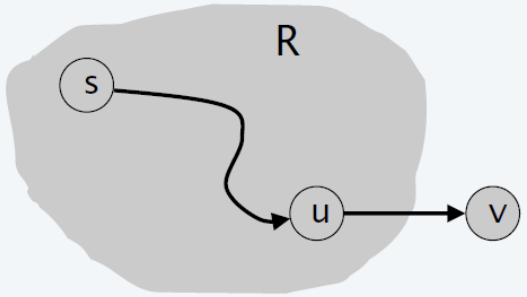
Recall: queue:



stack:




```
R will consist of nodes to which s has a path
Initially R = {s}
While there is an edge (u, v) where u ∈ R and v ∉ R
  Add v to R
Endwhile
```



it's safe to add v

Depending on how we pick the edges, we have different algorithms.

① BFS: put vertices in a queue, and pick a vertex and add all its neighbors to queue

Theorem. Upon termination, R is the connected component containing s.

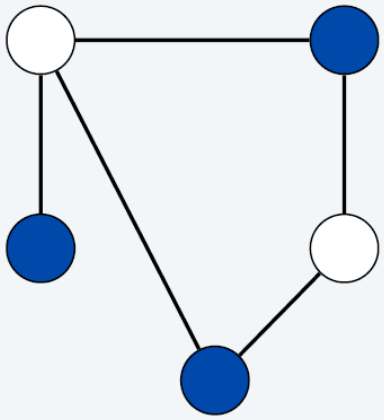
- BFS = explore in order of distance from s.
- DFS = explore in a different way.

② DFS: put vertices in a stack, and pick a vertex and add all its neighbors to stack

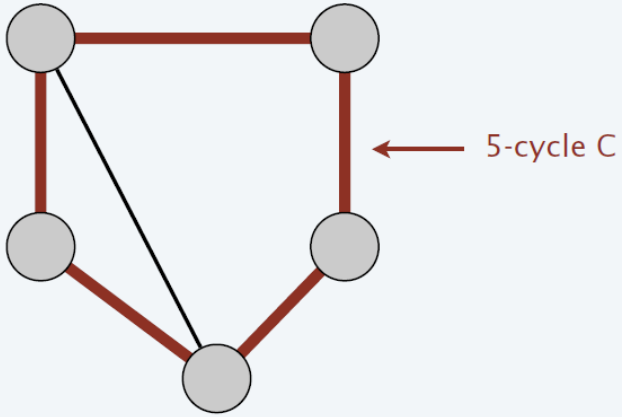
Graph algorithms

Def. An undirected graph $G = (V, E)$ is bipartite if the nodes can be colored blue or white such that every edge has one white and one blue end.

Lemma. If a graph G is bipartite, it cannot contain an odd-length cycle.



bipartite
(2-colorable)



not bipartite
(not 2-colorable)

Graph algorithms

Notation. $G = (V, E)$.

- Edge (u, v) leaves node u and enters node v.

Def. Nodes u and v are mutually reachable if there is both a path from u to v and also a path from v to u.

Def. A graph is strongly connected if every pair of nodes is mutually reachable.

Lemma. Let s be any node. G is strongly connected iff every node is reachable from s, and s is reachable from every node.

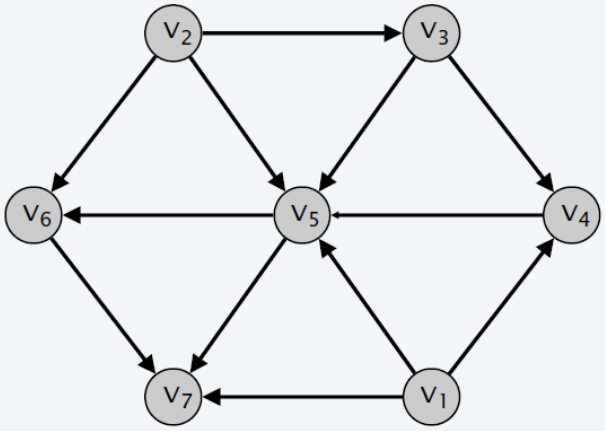
Graph algorithms

Def. A DAG is a directed graph that contains no directed cycles.

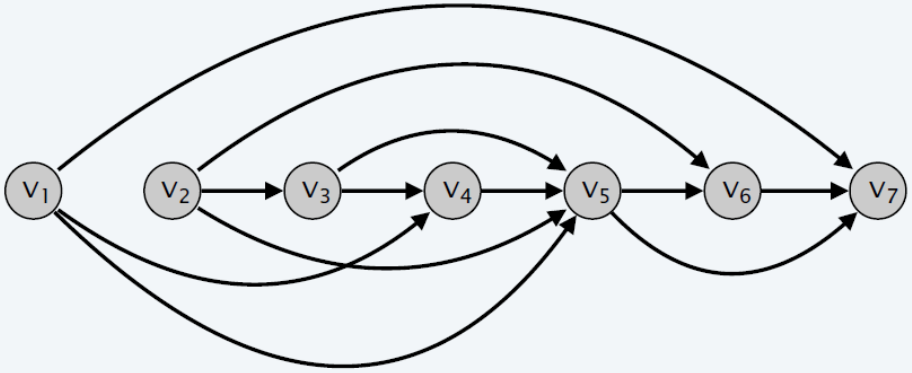
Def. A topological order of a directed graph $G = (V, E)$ is an ordering of its nodes as $v_1, v_2, \ldots, v_n$ so that for every edge (v_i, v_j) we have $i < j$.

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

Lemma. If G is a DAG, then G has a node with no entering edges.



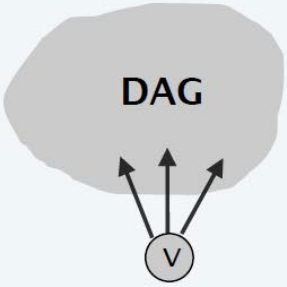
a DAG



a topological ordering

Graph algorithms

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



Theorem. Algorithm finds a topological order in $O(m+n)$ time.

Midterm I Review

- *stable matching*
- *complexity analysis*
- *graph algorithms*
- *greedy algorithms*

Greedy algorithms

Interval scheduling

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

EARLIEST-FINISH-TIME-FIRST $(n, s_1, s_2, \ldots, s_n, f_1, f_2, \ldots, f_n)$

SORT jobs by finish times and renumber so that $f_1 \le f_2 \le \ldots \le f_n$.

$S \leftarrow \varnothing$. ← set of jobs selected

FOR $j = 1$ TO n

  IF (job j is compatible with S)

    $S \leftarrow S \cup \{ j \}$.

RETURN S.

Proposition. Can implement earliest-finish-time first in $O(n \log n)$ time.

Theorem. The earliest-finish-time-first algorithm is optimal.

Greedy algorithms

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 lectures occur at the same time in the same room.

EARLIEST-START-TIME-FIRST $(n, s_1, s_2, \ldots, s_n, f_1, f_2, \ldots, f_n)$

SORT lectures by start times and renumber so that $s_1 \leq s_2 \leq \ldots \leq s_n$.

$d \leftarrow 0$. ← number of allocated classrooms

FOR $j = 1$ TO n

  IF (lecture j is compatible with some classroom)

    Schedule lecture j in any such classroom k.

  ELSE

    Allocate a new classroom $d + 1$.

    Schedule lecture j in classroom $d + 1$.

    $d \leftarrow d + 1$.

RETURN schedule.

Greedy algorithms

Proposition. The earliest-start-time-first algorithm can be implemented in $O(n \log n)$ time.

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

Observation. The earliest-start-time first algorithm never schedules two incompatible lectures in the same classroom.

Theorem. Earliest-start-time-first algorithm is optimal.

Scheduling to minimizing lateness

- 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 $L = \max_j \ell_j$.

EARLIEST-DEADLINE-FIRST $(n, t_1, t_2, \ldots, t_n, d_1, d_2, \ldots, d_n)$

SORT jobs by due times and renumber so that $d_1 \le d_2 \le \ldots \le d_n$.

$t \leftarrow 0$.

FOR $j = 1$ TO n

  Assign job j to interval $[t, t + t_j]$.

  $s_j \leftarrow t$; $f_j \leftarrow t + t_j$.

  $t \leftarrow t + t_j$.

RETURN intervals $[s_1, f_1], [s_2, f_2], \ldots, [s_n, f_n]$.

Greedy algorithms

Observation 1. There exists an optimal schedule with no idle time.

Observation 2. The earliest-deadline-first schedule has no idle time.

Def. Given a schedule S, an inversion is a pair of jobs i and j such that: $i < j$ but j is scheduled before i.

Observation 3. The earliest-deadline-first schedule is the unique idle-free schedule with no inversions.

Theorem. The earliest-deadline-first schedule S is optimal.

Greedy algorithms

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.

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.

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

Other greedy algorithms. Gale–Shapley, Kruskal, Prim, Dijkstra, Huffman, ...

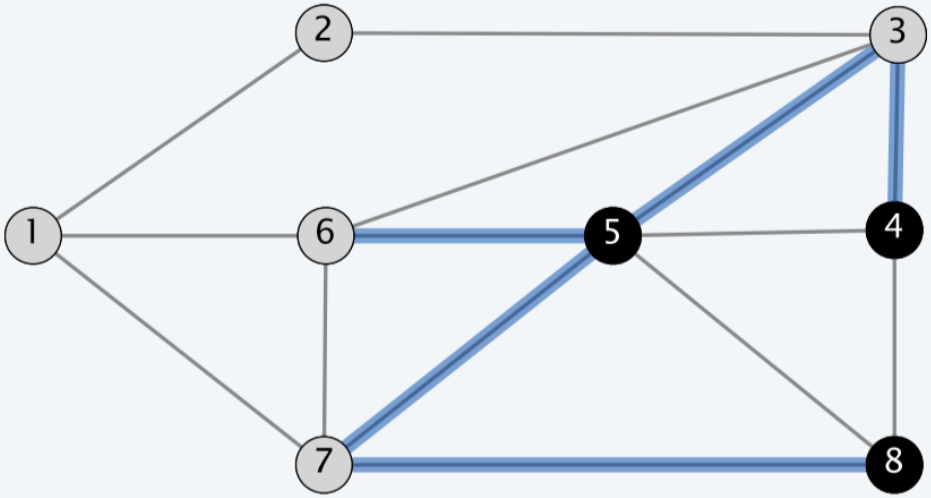
Greedy algorithms: cut and cutset

Def. A cut is a partition of the nodes into two nonempty subsets S and $V - S$.

note: These partitions need not to be connected, i.e., S & V-S might be disconnected themselves.

Def. The cutset of a cut S is the set of edges with exactly one endpoint in S.

note: If you remove edges in a cutset, there is no way to go from any vertex in S to any vertex in V-S



cut S = { 4, 5, 8 }

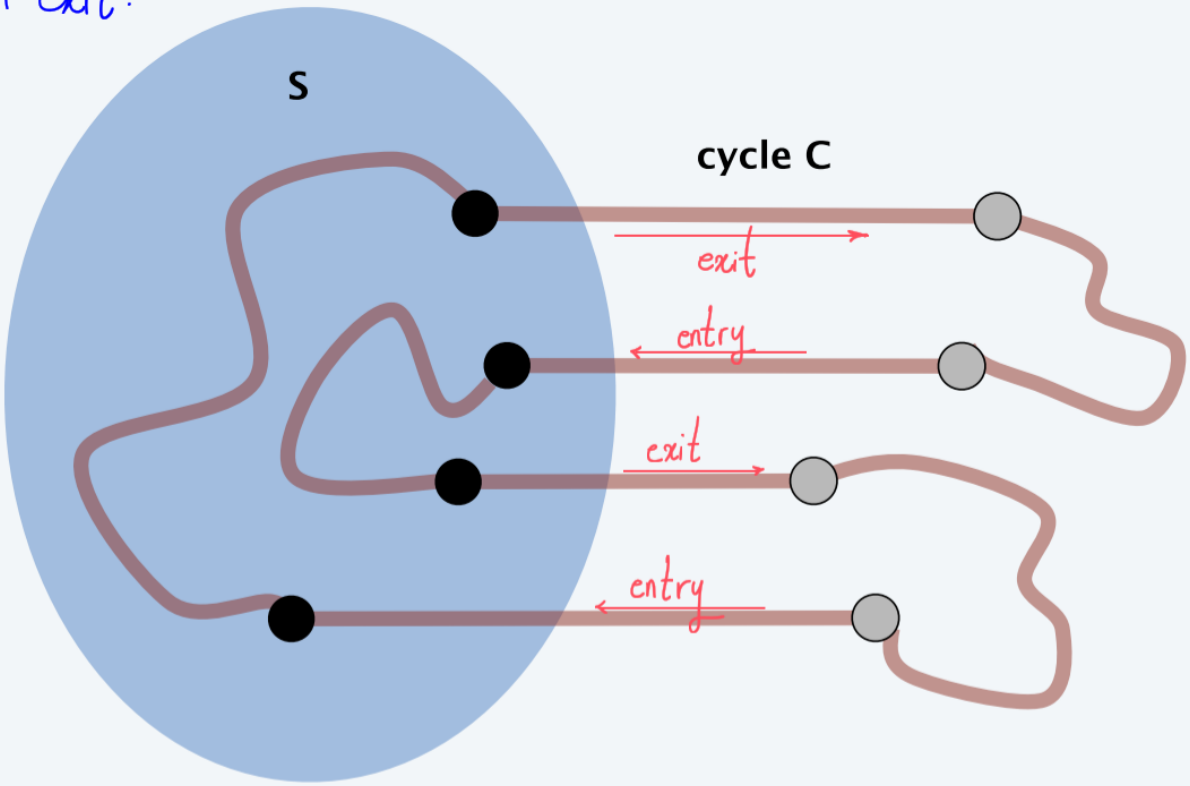
cutset D = { (3, 4), (3, 5), (5, 6), (5, 7), (8, 7) }

Greedy algorithms: cut and cutset

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

Pf. [by picture]

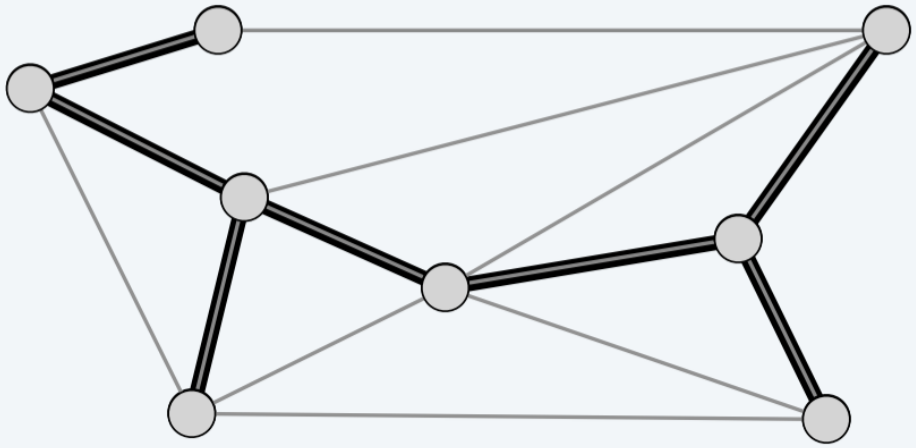
If you move in a specific direction on the cycle, for every entry to S there is an exit.



Greedy algorithms: spanning tree

Def. Let $H = (V, T)$ be a subgraph of an undirected graph $G = (V, E)$.

H is a spanning tree of G if H is both acyclic and connected.

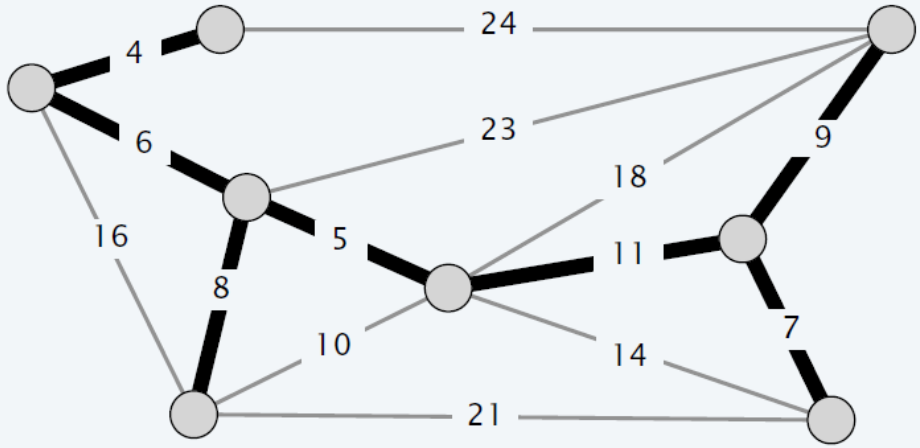


graph G = (V, E)

spanning tree H = (V, T)

Greedy algorithms: minimum spanning tree

Def. Given a connected, undirected graph $G = (V, E)$ with edge costs c_e, a minimum spanning tree (V, T) is a spanning tree of G such that the sum of the edge costs in T is minimized.



MST cost = 50 = 4 + 6 + 8 + 5 + 11 + 9 + 7

Cayley's theorem. The complete graph on n nodes has n^{n-2} spanning trees.

can't solve by brute force

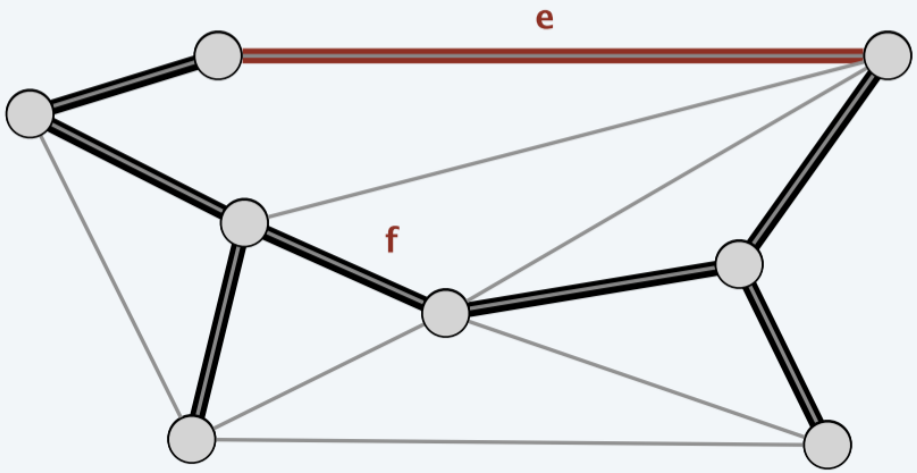
Greedy algorithms

Fundamental cycle. Let $H = (V, T)$ be a spanning tree of $G = (V, E)$.

- For any non tree-edge $e \in E$: $T \cup \{ e \}$ contains a unique cycle, say C.
- For any edge $f \in C$: $(V, T \cup \{ e \} - \{ f \})$ is a spanning tree.

we can switch e with any edge on the fundamental cycle C.

Notice that $e \notin T$.



graph G = (V, E)

spanning tree H = (V, T)

Observation. If $c_e < c_f$, then (V, T) is not an MST.

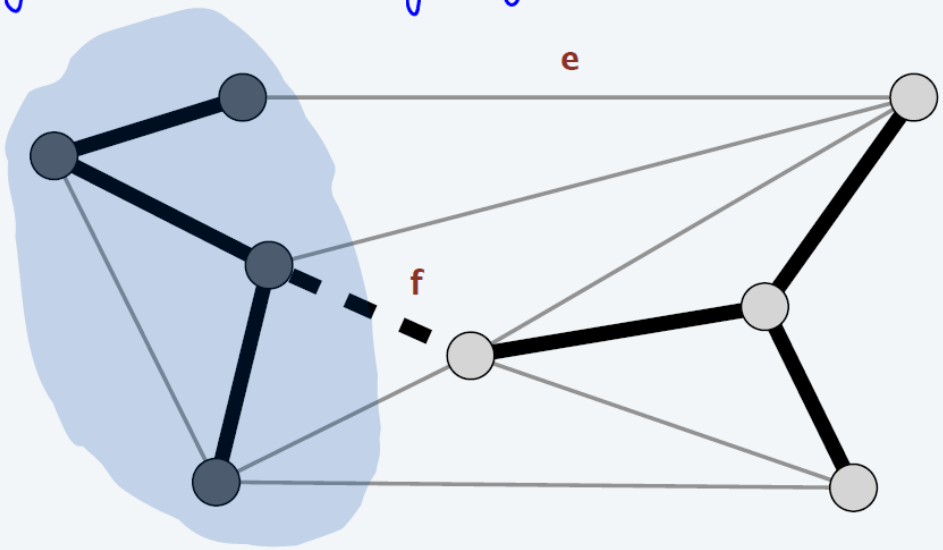
Greedy algorithms

Fundamental cutset. Let $H = (V, T)$ be a spanning tree of $G = (V, E)$.

- For any tree edge $f \in T$: $(V, T - \{ f \})$ has two connected components.
- Let D denote corresponding cutset.
- For any edge $e \in D$: $(V, T - \{ f \} \cup \{ e \})$ is a spanning tree.

you can switch every edge in the fundamental cutset with f.

Notice that $f \in T$.



graph G = (V, E)

spanning tree H = (V, T)

Observation. If $c_e < c_f$, then (V, T) is not an MST.

Greedy algorithms

Red rule.

- Let C be a cycle with no red edges.
- Select an uncolored edge of C of max cost and color it red.

Blue rule.

- Let D be a cutset with no blue edges.
- Select an uncolored edge in D of min cost and color it blue.

Greedy algorithm.

- Apply the red and blue rules (nondeterministically!) until all edges are colored. The blue edges form an MST.
- Note: can stop once $n - 1$ edges colored blue.

Greedy algorithms: Prim's algorithm

Initialize $S = \{ s \}$ for any node s, $T = \emptyset$.

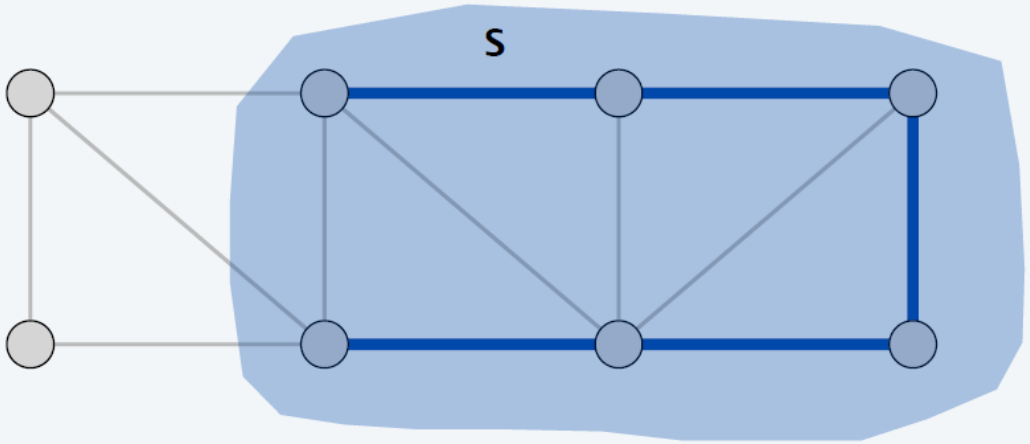
Repeat $n - 1$ times:

- Add to T a min-cost edge with exactly one endpoint in S.
- Add the other endpoint to S.

by construction, edges in cutset are uncolored

Theorem. Prim's algorithm computes an MST.

Pf. Special case of greedy algorithm (blue rule repeatedly applied to S). ▪



Greedy algorithms: Prim's algorithm

Theorem. Prim's algorithm can be implemented to run in $O(m \log n)$ time.

Pf. Implementation almost identical to Dijkstra's algorithm.

PRIM (V, E, c)

$S \leftarrow \emptyset$, $T \leftarrow \emptyset$.

$s \leftarrow$ any node in V.

FOREACH $v \neq s$: $\pi[v] \leftarrow \infty$, $pred[v] \leftarrow null$; $\pi[s] \leftarrow 0$.

predecessor

Create an empty priority queue pq.

FOREACH $v \in V$: INSERT$(pq, v, \pi[v])$. ⇝ at this point $\pi(v) = \infty$ for all $v \in V - \{s\}$, & $\pi(s) = 0$

WHILE (IS-NOT-EMPTY(pq))

$\pi[v]$ = cost of cheapest known edge between v and S

$u \leftarrow$ DEL-MIN(pq).

$S \leftarrow S \cup \{ u \}$, $T \leftarrow T \cup \{ pred[u] \}$.

FOREACH edge $e = (u, v) \in E$ with $v \notin S$:

IF $(c_e < \pi[v])$

DECREASE-KEY(pq, v, c_e).

$\pi[v] \leftarrow c_e$; $pred[v] \leftarrow e$.

Greedy algorithms: Kruskal's algorithm

Consider edges in ascending order of cost:

- Add to tree unless it would create a cycle.

Theorem. Kruskal's algorithm computes an MST.

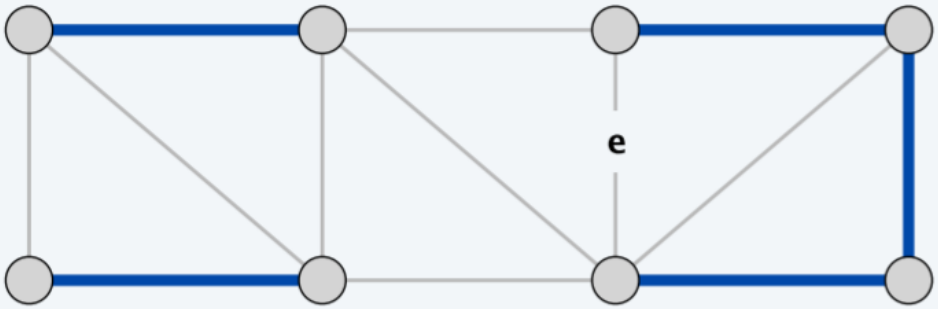
Pf. Special case of greedy algorithm. red rule

- Case 1: both endpoints of e in same blue tree.
  $\Rightarrow$ color e red by applying red rule to unique cycle.
- Case 2: both endpoints of e in different blue trees.
  $\Rightarrow$ color e blue by applying blue rule to cutset defined by either tree. ▪

blue rule

all other edges in cycle are blue

no edge in cutset has smaller cost
(since Kruskal chose it first)



Greedy algorithms: Kruskal's algorithm

Theorem. Kruskal's algorithm can be implemented to run in $O(m \log m)$ time.

- Sort edges by cost. → will take $O(m \log m)$
- Use union–find data structure to dynamically maintain connected components. → Recall that Union operation will take $\log n$

KRUSKAL (V, E, c) Note: for any connected graph, $O(m \log m) = O(m \log n)$

SORT m edges by cost and renumber so that $c(e_1) \le c(e_2) \le \ldots \le c(e_m)$.

$T \leftarrow \emptyset$.

FOREACH $v \in V$: MAKE-SET(v).

FOR $i = 1$ TO m

  $(u, v) \leftarrow e_i$.

  IF (FIND-SET$(u) \neq$ FIND-SET(v)) ← are u and v in same component? | $\log(n)$

    $T \leftarrow T \cup \{ e_i \}$.

    UNION(u, v). ← make u and v in same component | $\log(n)$

RETURN T.

Greedy algorithms: reverse delete

Start with all edges in T and consider them in descending order of cost:

- Delete edge from T unless it would disconnect T.

Theorem. The reverse-delete algorithm computes an MST.

Pf. Special case of greedy algorithm.

red rule

- Case 1. [deleting edge e does not disconnect T]

  $\Rightarrow$ apply red rule to cycle C formed by adding e to another path in T between its two endpoints

  no edge in C is more expensive
  (it would have already been considered and deleted)

blue rule

- Case 2. [deleting edge e disconnects T]

  $\Rightarrow$ apply blue rule to cutset D induced by either component ▪

  e is the only remaining edge in the cutset
  (all other edges in D must have been colored red / deleted)