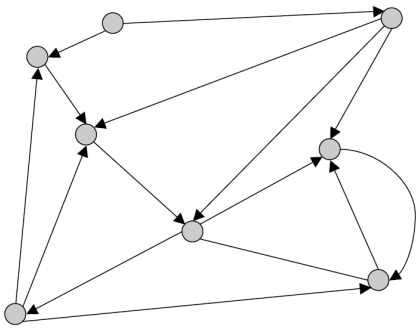
COS 460 – Algorithms (week 12)
Directed Graphs
Digraph. Set of objects with oriented pairwise connections.

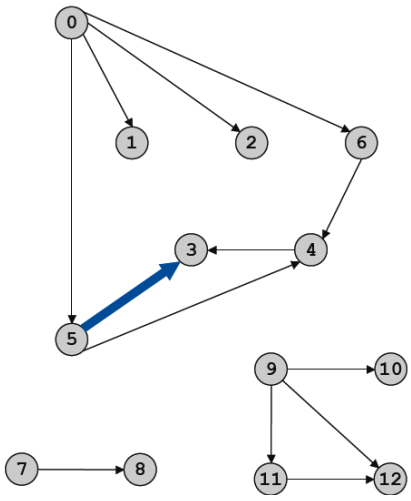


Digraph	Vertex	Edge
financial	stock, currency	transaction
transportation	street intersection, airport	highway, airway route
scheduling	task	precedence constraint
WordNet	synset	hypernym
Web	web page	hyperlink
game	board position	legal move
telephone	person	placed call
food web	species	predator-prey relation
infectious disease	person	infection
citation	journal article	citation
object graph	object	pointer
inheritance hierarchy	class	inherits from
control flow	code block	jump

Some Digraph Problems
Transitive closure. Is there a directed path from v to w?
Strong connectivity. Are all vertices mutually reachable?
Topological sort. Can you draw the graph so that all edges point from left to right?
PERT/CPM. Given a set of tasks with precedence constraints, what is the earliest that we can complete each task?
Shortest path. Given a weighted graph, find best route from s to t?

Digraph Representation

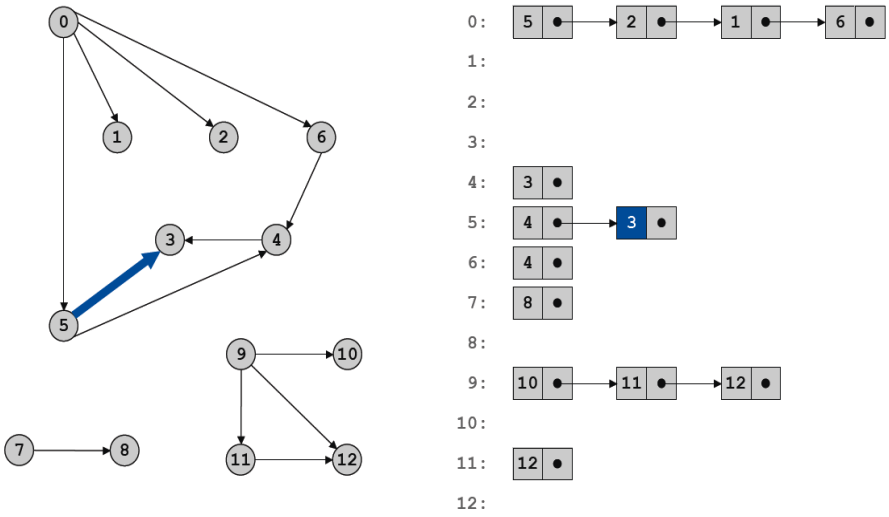
Adjacency matrix representation.
Two-dimensional V by V boolean array.
Edge v → w in graph: adj[v][w] = true.



to

from	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	1	1	0	0	1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	1	0	0	0	0	0	0	0	0	0
5	0	0	0	1	1	0	0	0	0	0	0	0	0
6	0	0	0	0	1	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	1	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	1	1	1
10	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	1
12	0	0	0	0	0	0	0	0	0	0	0	0	0

Adjacency list representation. Vertex indexed array of lists.



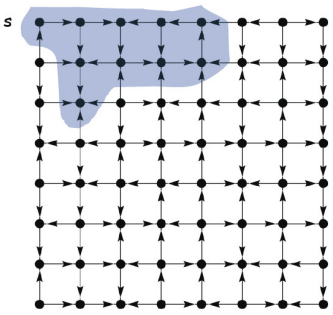
Implementation is the same as for undirected graphs, but only insert one copy of each edge.

Digraphs are abstract mathematical objects.

- ADT implementation requires specific representation.
- Efficiency depends on matching algorithms to representations.

Representation	Space	Edge from v to w?	Iterate over edges leaving v?
List of edges	E + V	E	E
Adjacency matrix	V^2	1	V
Adjacency list	E + V	outdegree(v)	outdegree(v)

Digraph Search. Reachability: Find all vertices reachable from s along a directed path.



Depth first search. Same as for undirected graphs.

DFS (to visit a vertex v)

- Mark v as visited.
- Visit all unmarked vertices w adjacent to v.

Running time: O(E) since each edge examined at most once.
Implementation remark:
Same as undirected version, except using Digraph ADT instead of Graph ATD.

Example: Control Flow Graph

- Vertex = basic block (straight-line program).
- Edge = jump.

Problem: Dead code elimination. Find (and remove) code blocks that are unreachable during execution. (dead code can arise from compiler optimization or careless programming)
Problem: Infinite loop detection. Exit block is unreachable from entry block.
Caveat. Not all infinite loops are detectable.

Mark-Sweep Garbage Collector.
Live objects. Objects that the program could get to by starting at a root and following a chain of pointers.
Mark-sweep algorithm. [by McCarthy, 1960]

- Mark: run DFS from roots to mark live objects.
- Sweep: if object is unmarked, it is garbage, so add to free list.

Depth First Search
DFS enables direct solution of simple digraph problems.

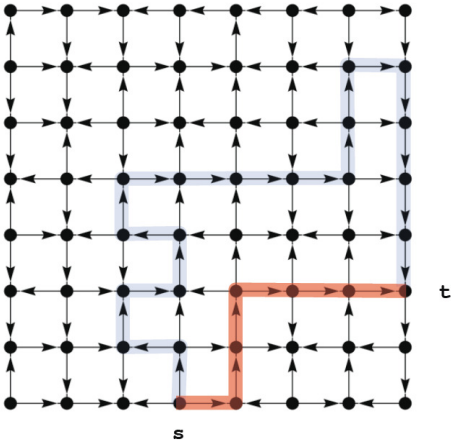
- Reachability.
- Cycle detection.
- Topological sort.
- Transitive closure.
- Find path from s to t.

Basis for solving difficult digraph problems.

- Directed Euler path.
- Strong connected components.

Breadth First Search

- Shortest path. Find the shortest directed path from s to t.
- BFS. Analogous to BFS in undirected graphs.



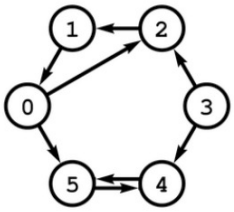
Application: Web Crawler. Web graph. Vertex = website, edge = hyperlink.
Goal. Crawl Internet, starting from some root website.
Solution. BFS with implicit graph.
BFS.

- Start at some root website, say http://www.princeton.edu.
- Maintain a Queue of websites to explore.
- Maintain a SET of discovered websites.
- Dequeue the next website, and enqueue websites to which it links (provided you haven't done so before).

Q. Why not use DFS?

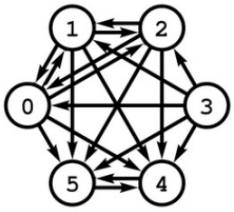
Transitive closure. Is there a directed path from v to w?

G



	0	1	2	3	4	5
0	1	0	1	0	0	1
1	1	1	0	0	0	0
2	0	1	1	0	0	0
3	0	0	1	1	1	0
4	0	0	0	0	1	1
5	0	0	0	0	1	1

Transitive closure



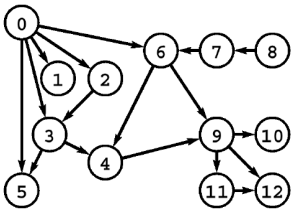
	0	1	2	3	4	5
0	1	1	1	0	1	1
1	1	1	1	0	1	1
2	1	1	1	0	1	1
3	1	1	1	1	1	1
4	0	0	0	0	1	1
5	0	0	0	0	1	1

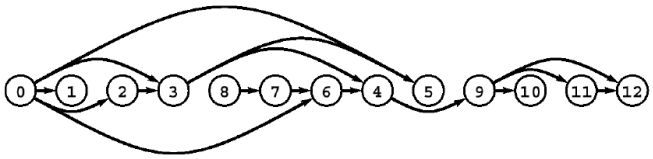
tc[v][w] = 1 iff path from v to w

Lazy. Run separate DFS for each query.
Eager. Run DFS from every vertex v; store results.

Method	Preprocess	Query	Space
DFS (lazy)	1	E + V	E + V
DFS (eager)	E V	1	V^2

Topological Sort
DAG. Directed acyclic graph.





Redraw DAG so all edges point left to right. Observation: Not possible if graph has a directed cycle.

Application: **Scheduling**
Given a set of tasks to be completed with precedence constraints, in what order should we schedule the tasks?
Graph model.

- Create a vertex v for each task.
- Create an edge v → w if task v must precede task w.
- Schedule tasks in topological order.

Topologically sort a DAG.

- Run DFS.
- Reverse postorder numbering yields a topological sort.

Proof of correctness: When DFS backtracks from a vertex v, all vertices reachable from v have already been explored. Running time: O(E + V).

Q. If not a DAG, how would you identify a cycle?

Topological sort applications.

- Causalities.
- Compilation units.
- Class inheritance.
- Course prerequisites.
- Deadlocking detection.
- Temporal dependencies.
- Pipeline of computing jobs.
- Check for symbolic link loop.
- Evaluate formula in spreadsheet.

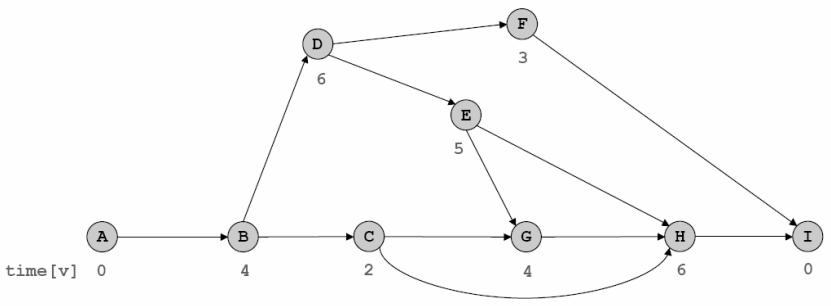
Program Evaluation and Review Technique / Critical Path Method

PERT/CPM.

- Task v takes time[v] units of time.
- Can work on jobs in parallel.
- Precedence constraints: must finish task v before beginning task w.
- What's earliest we can finish each task?

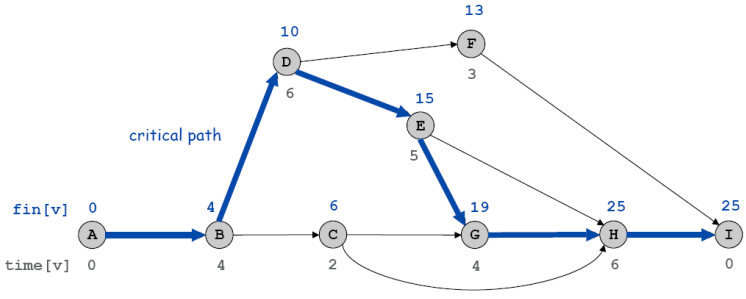
Example:

index	task	time	prereq
A	begin	0	-
B	framing	4	A
C	roofing	2	B
D	siding	6	B
E	windows	5	D
F	plumbing	3	D
G	electricity	4	C, E
H	paint	6	C, E
I	finish	0	F, H

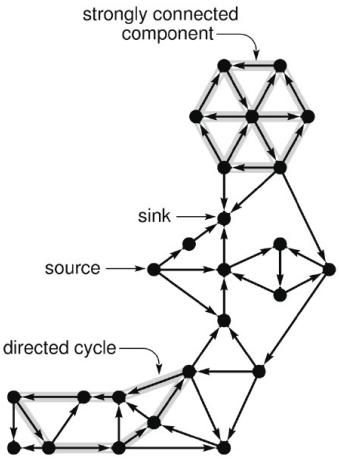


PERT/CPM algorithm.

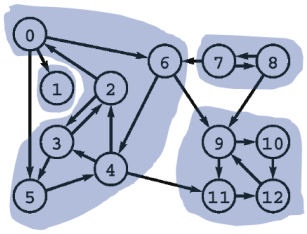
- Compute topological order of vertices.
- Initialize fin[v] = 0 for all vertices v.
- Consider vertices v in topological order.
  - for each edge v → w, set fin[w]= max(fin[w], fin[v] + time[w])



Strongly Connected Components. Terminology:

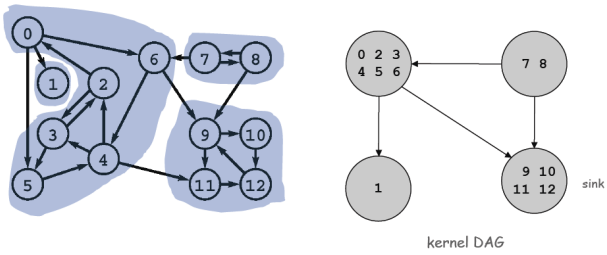


Strong Components.
Def.: Vertices v and w are strongly connected if there is a path from v to w and a path from w to v.
Properties. Symmetric, transitive, reflexive.
Strong component. Maximal subset of strongly connected vertices.
Brute force: O(EV) time using transitive closure.

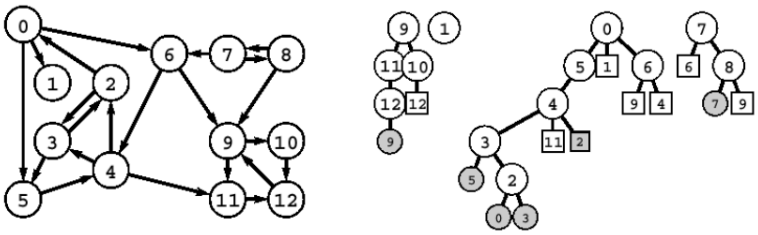


	0	1	2	3	4	5	6	7	8	9	10	11	12
sc	2	1	2	2	2	2	2	3	3	0	0	0	0

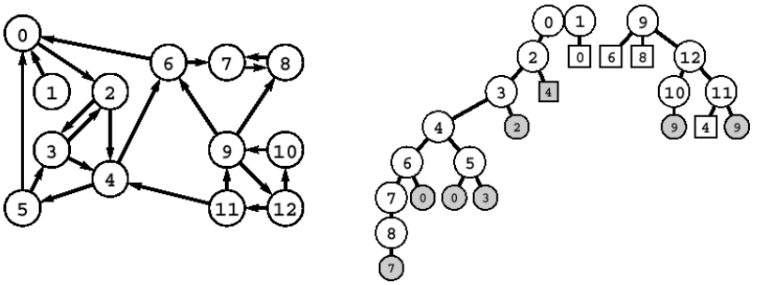
Computing Strongly Connected Components
Observation 1. If you run DFS from a vertex in sink strong component, all reachable vertices constitute a strong component.
Observation 2. If you run DFS on G, the node with the highest postorder (in order of completion of recursive calls) number is in source strong component.
Observation 3. If you run DFS on G^R, the node with the highest postorder number is in sink strong component.



Kosaraju's algorithm.
- Run DFS on G^R and compute postorder.
- Run DFS on G, considering vertices in reverse postorder.

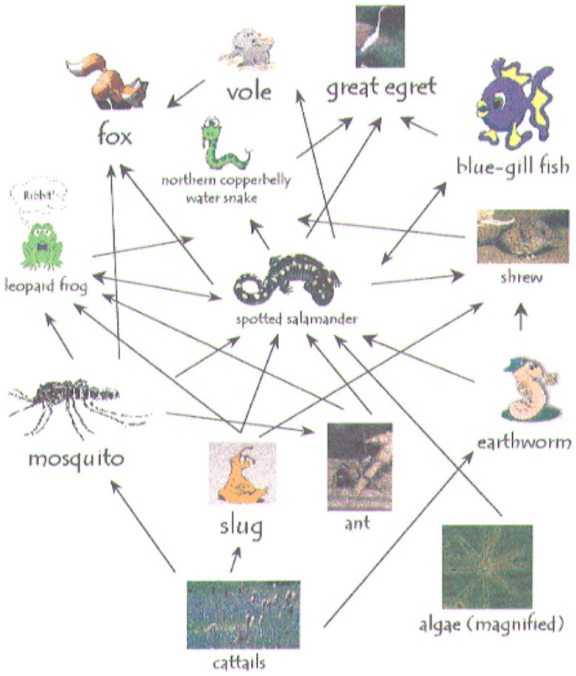


	0	1	2	3	4	5	6	7	8	9	10	11	12
post	8	7	6	5	4	3	2	0	1	11	10	12	9



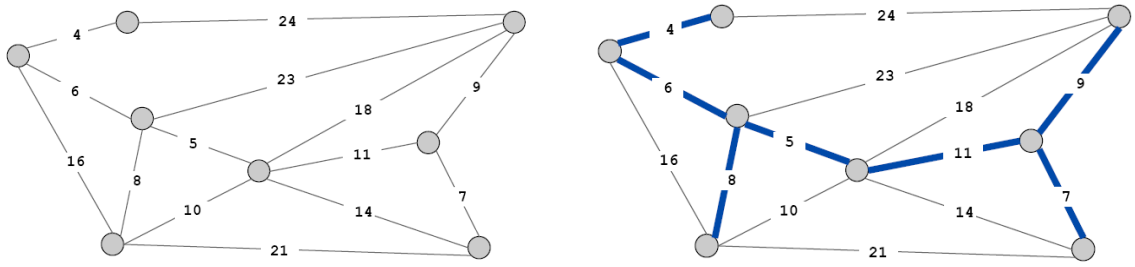
	0	1	2	3	4	5	6	7	8	9	10	11	12
sc	2	1	2	2	2	2	2	3	3	0	0	0	0

Theorem. Trees in second DFS are strong components.
Example: Ecological food web.
- Vertex = species.
- Edge = from producer to consumer.
- Strong component = subset of species for which energy flows from one another and back.



Weighted Graphs

Minimum Spanning Tree
MST. Given connected graph G with positive **edge weights**, find a minimum weight set of edges that connects all of the vertices.



G

cost(T) = 50

MST Origin (Otakar Boruvka, 1926).

- Electrical Power Company of Western Moravia in Brno.
- Most economical construction of electrical power network.
- Concrete engineering problem is now a cornerstone problem in combinatorial optimization.

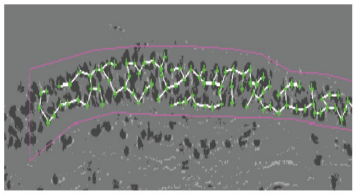
MST is fundamental problem with diverse applications.

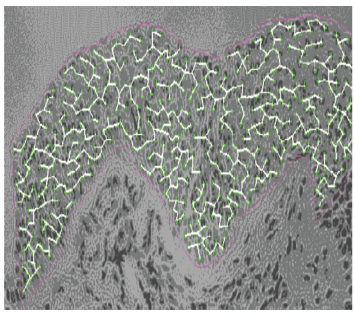
- Network design.
  - telephone, electrical, hydraulic, TV cable, computer, road
- Approximation algorithms for NP-hard problems.
  - traveling salesperson problem, Steiner tree

Indirect applications.

- max bottleneck paths
- codes for error correction
- 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.

Medical Image Processing
MST describes arrangement of nuclei in the epithelium for cancer research





Two Greedy Algorithms

Kruskal's algorithm. Consider edges in ascending order of cost. Add the next edge to T unless doing so would create a cycle.

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

Theorem. Both greedy algorithms compute an MST.

Def. A **spanning tree** of a graph G is a subgraph T that is connected and acyclic.

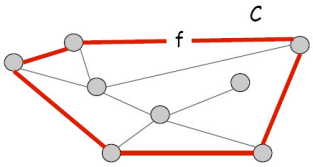
MST. Given connected graph G with positive edge weights, find a **min weight set of edges** that connects all of the vertices.

Property. MST of G is always a spanning tree.

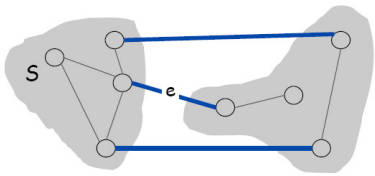
Simplifying assumption. All edge costs c_e are distinct.

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

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



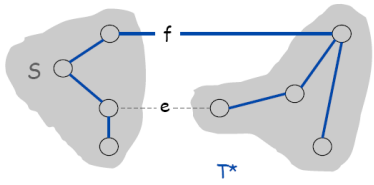
f is not in the MST



e is in the MST

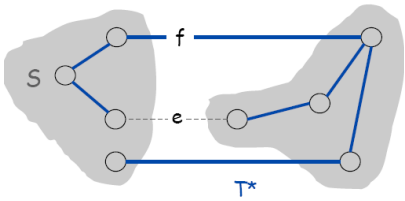
Proof of *Cycle property* [by contradiction]

- Suppose f belongs to T*.
- Deleting f from T* disconnects T*. Let S be one side of the cut.
- Some other edge in C, say e, has exactly one endpoint in S.
- T = T* + { e } – { f } is also a spanning tree.
- Since $c_e < c_f$, cost(T) < cost(T*).
- This is a contradiction.

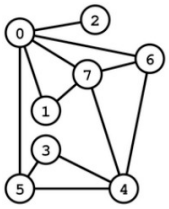


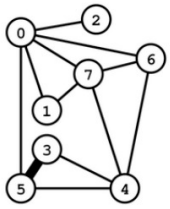
Proof of *Cut property* [by contradiction]

- Suppose e does not belong to T*.
- Adding e to T* creates a (unique) cycle C in T*.
- Some other edge in C, say f, has exactly one endpoint in S.
- T = T* + { e } – { f } is also a spanning tree.
- Since $c_e < c_f$, cost(T) < cost(T*).
- This is a contradiction.

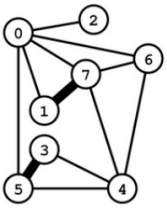


Kruskal's Algorithm Example: Consider edges in ascending order of cost (length). Add the next edge to T unless doing so would create a cycle.

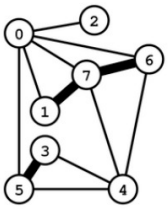




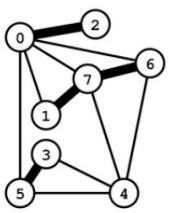
3-5



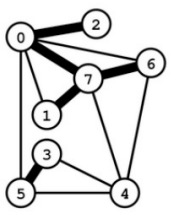
1-7



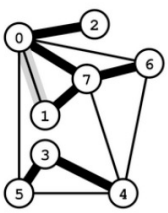
6-7



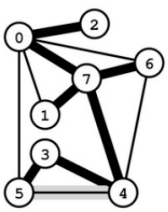
0-2



0-7

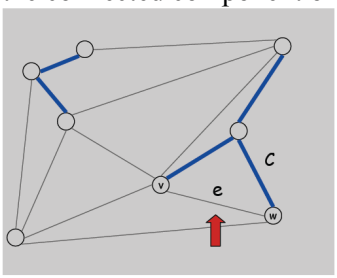


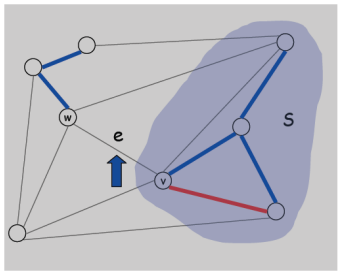
0-1 3-4



4-5 4-7

Theorem. Kruskal's algorithm computes the MST.
Proof.
Case 1: If adding e to T creates a cycle C, then e is the max weight edge in C. The cycle property asserts that e is not in the MST.
Case 2: If adding e = (v, w) to T does not create a cycle, then e is the min weight edge with exactly one endpoint in S, so the cut property asserts that e is in the MST (Here S denotes the set of vertices in the connected component of v).



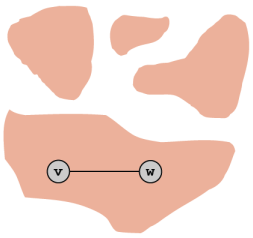


Kruskal's Algorithm: Implementation
Q. How to check if adding an edge to T would create a cycle?
A1. Naïve solution: use DFS.

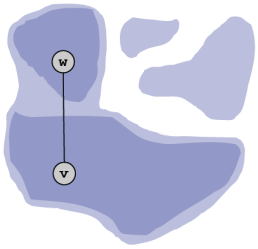
- O(V) time per cycle check.
- O(E V) time overall.

A2. Use the union-find data structure.

- Maintain a set for each connected component.
- If v and w are in same component, then adding v-w creates a cycle.
- To add v-w to T, merge sets containing v and w.



Case 1: adding v-w creates a cycle

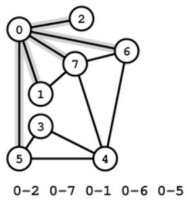


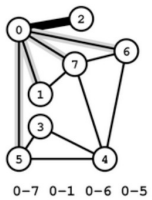
Case 2: add v-w to T and merge sets

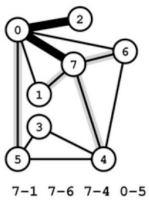
Kruskal running time: O(E log V) since

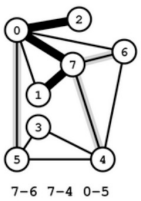
sort	1 times	E log V
union	V times	log V
find	E times	log V

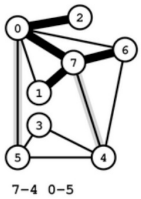
Prim's Algorithm Example: Start with vertex 0 and greedily grow tree T. At each step, add the cheapest edge that has exactly one endpoint in T.

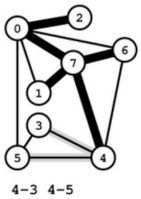


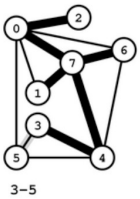


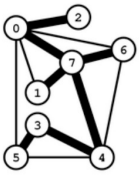












Theorem. Prim's algorithm computes the MST.
Proof:

- Let S be the subset of vertices in current tree T.
- Prim adds the cheapest edge e with exactly one endpoint in S.
- Cut property asserts that e is in the MST.

Prim's Algorithm: Implementation
Q. How to find cheapest edge with exactly one endpoint in S?
A1. Brute force: try all edges.

- O(E) time per spanning tree edge.
- O(E V) time overall.

A2. Maintain edges with (at least) one endpoint in S in a priority queue.

- Delete min to determine next edge e to add to T.
- Disregard e if both endpoints are in S.
- Upon adding e to T, add to PQ the edges incident to one endpoint.

Running time.

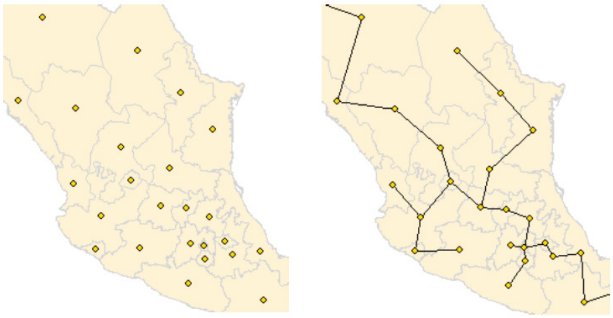
- O(log V) time per edge (using a binary heap).
- O(E log V) time overall.

Removing the *Distinct Edge Costs Assumption*: The correctness of both Prim and Kruskal don't actually rely on the assumption (only our proof of correctness does!).

One way to remove the assumption: When writing an implementation, Kruskal and Prim may only access edge weights through some compare function; suffices to introduce tie-breaking rule, e.g.:

```
Bool compare(edge e, edge f)
{ if (e.weight < f.weight) return true;
  if (e.v < f.v) return true;
  if (e.w < f.w) return true;
  return false;
}
```

Euclidean MST. Given N points in the plane, find MST connecting them. Distances between point pairs are Euclidean distances.



Brute force. Compute $O(N^2)$ distances and run Prim's algorithm.
Ingenuity. Exploit geometry and do it in O(N log N).

Key geometric fact. Edges of the Euclidean MST are edges of the Delaunay triangulation.
Euclidean MST algorithm.

- Compute Voronoi diagram to get Delaunay triangulation.
- Run Kruskal's MST algorithm on Delaunay edges.