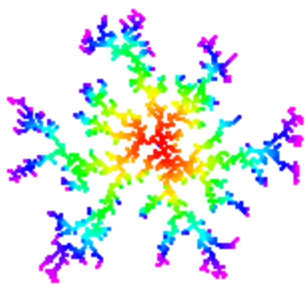
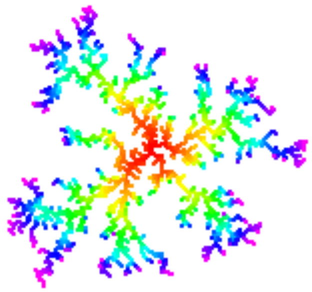
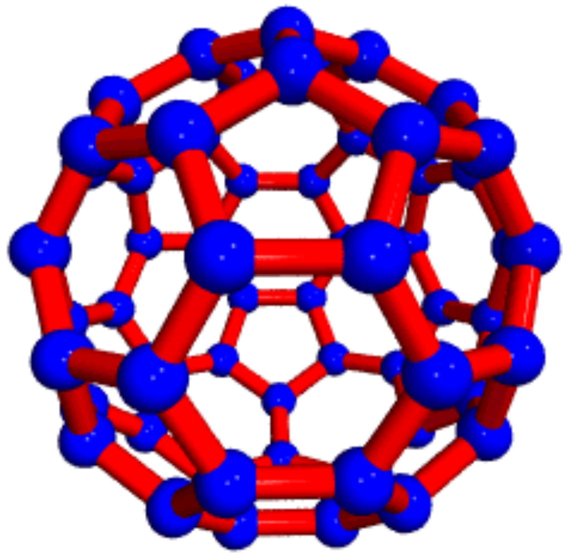
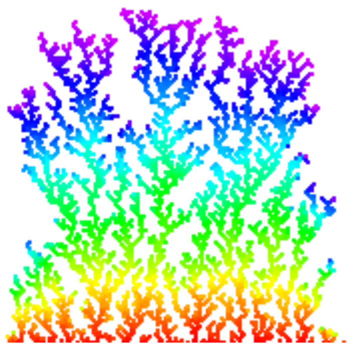


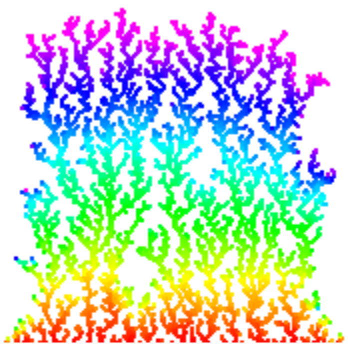
Data Structures and Algorithms







Minimum Spanning Trees



Minimum Spanning Trees

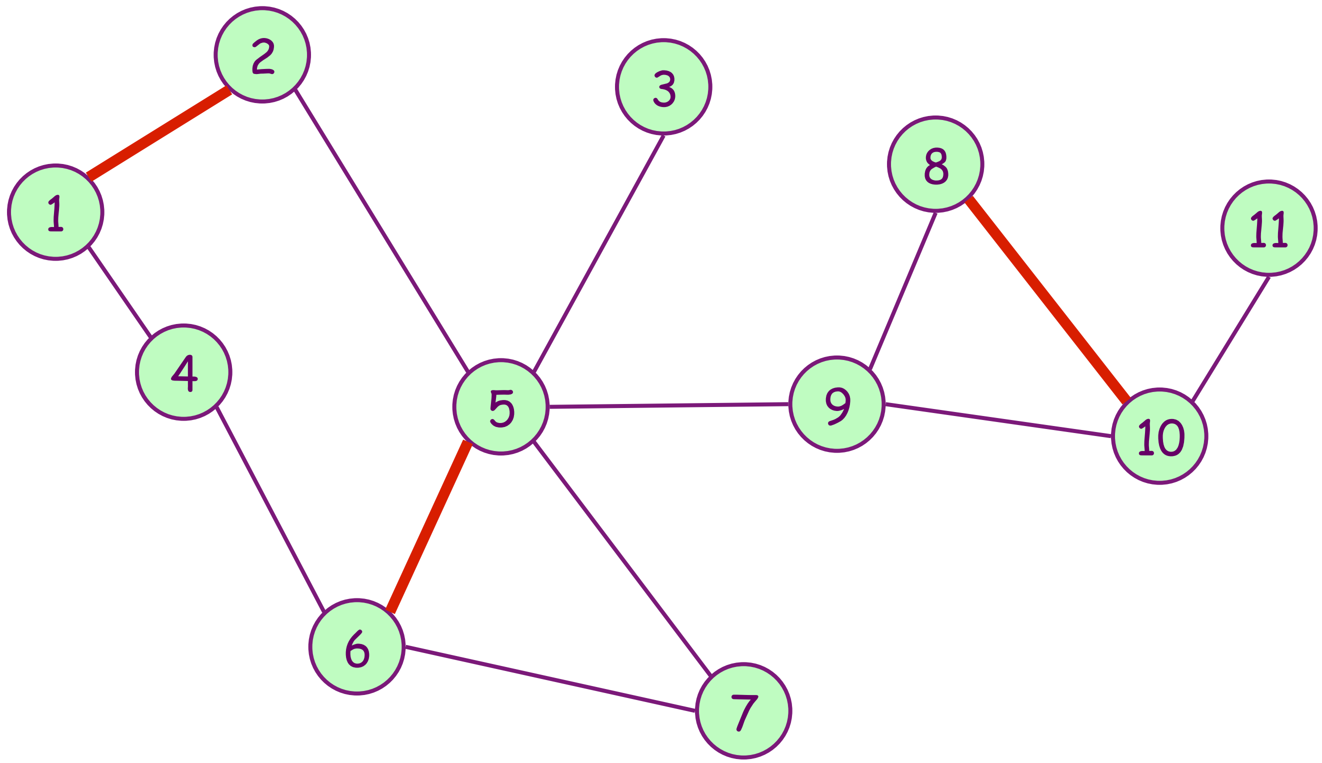
Prim's algorithm

Kruskal's algorithm

Greedy Algorithms

Example: Communication Network

- Removal of an edge that is on a cycle does not affect connectedness.



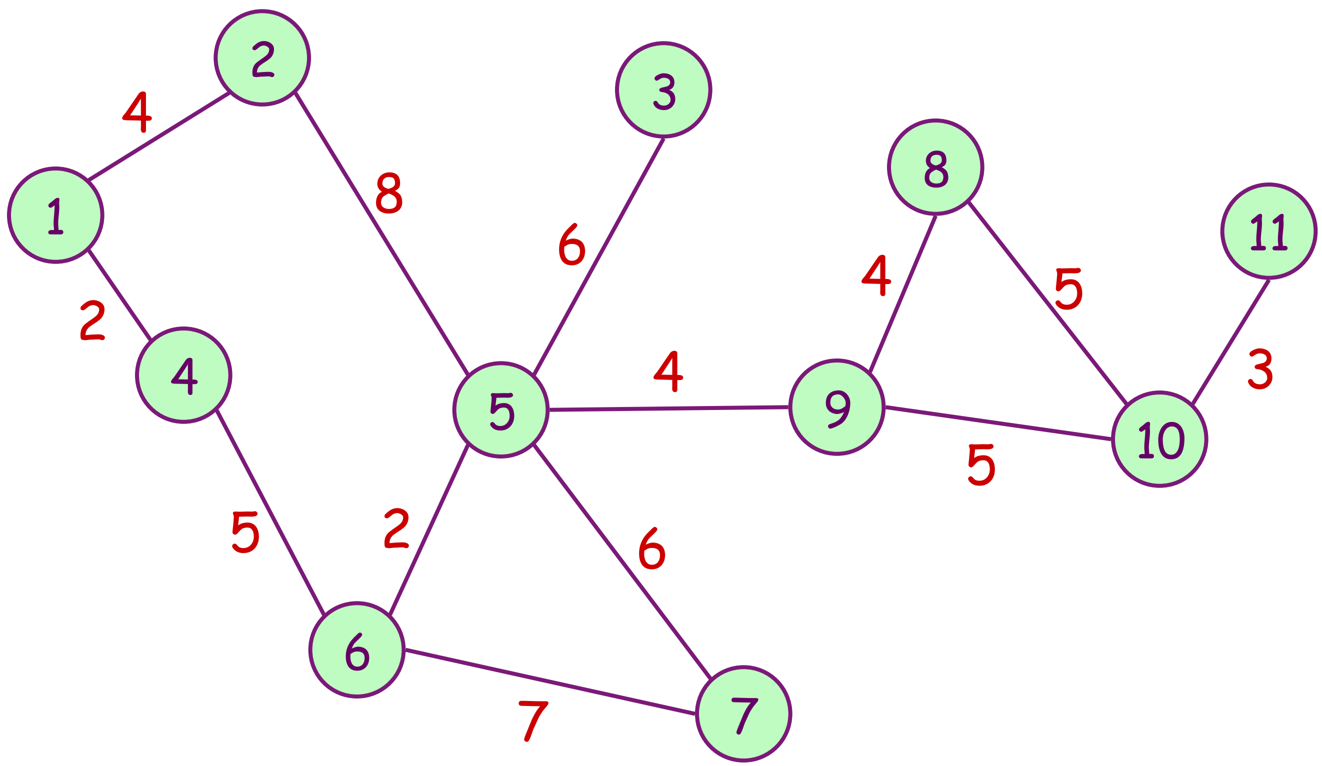
- Connected subgraph with all vertices and minimum number of edges has no cycles.

Example: Communication Network

- Tree
  - Connected graph that has no cycles.
  - n vertex connected graph with n-1 edges.

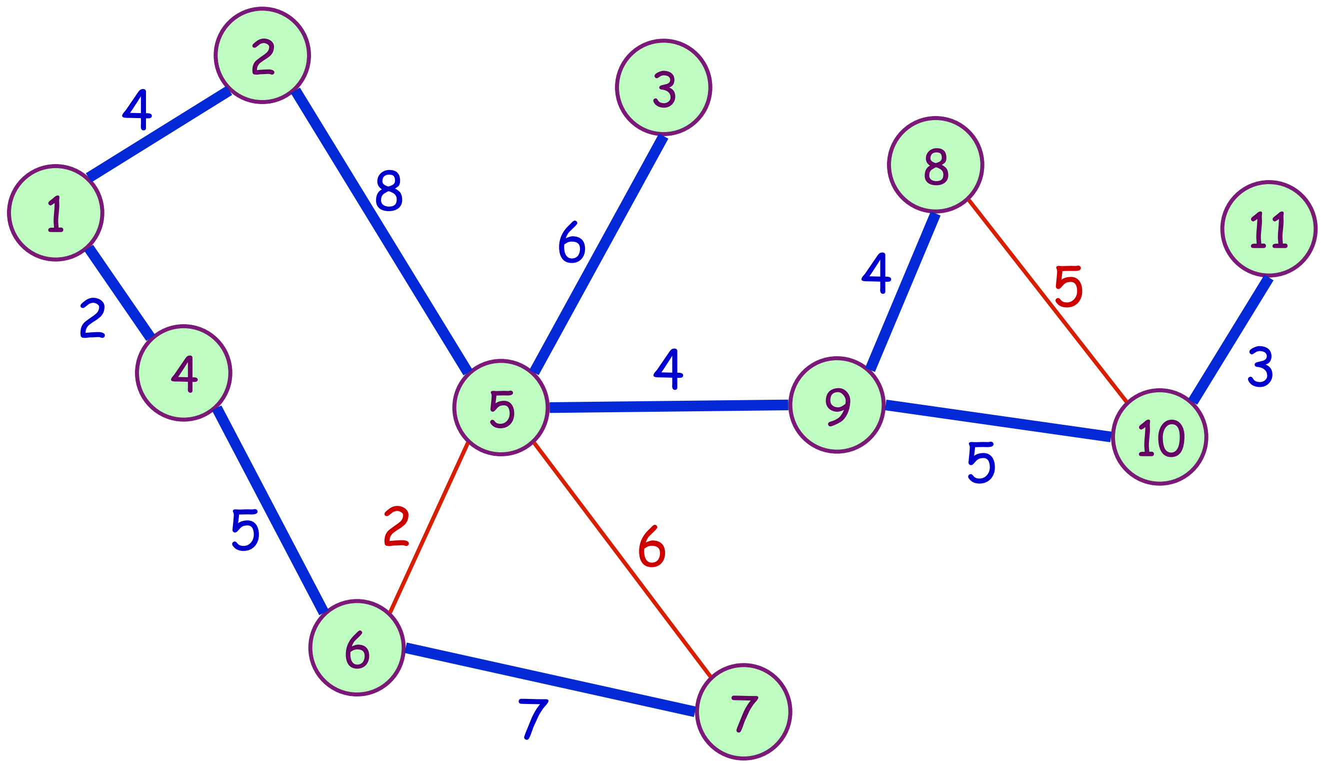
- Spanning Tree
  - Subgraph that includes all vertices of the original graph.
  - Subgraph is a tree.
    - If original graph has n vertices, the spanning tree has n vertices and n-1 edges.

Weight of a Spanning Tree



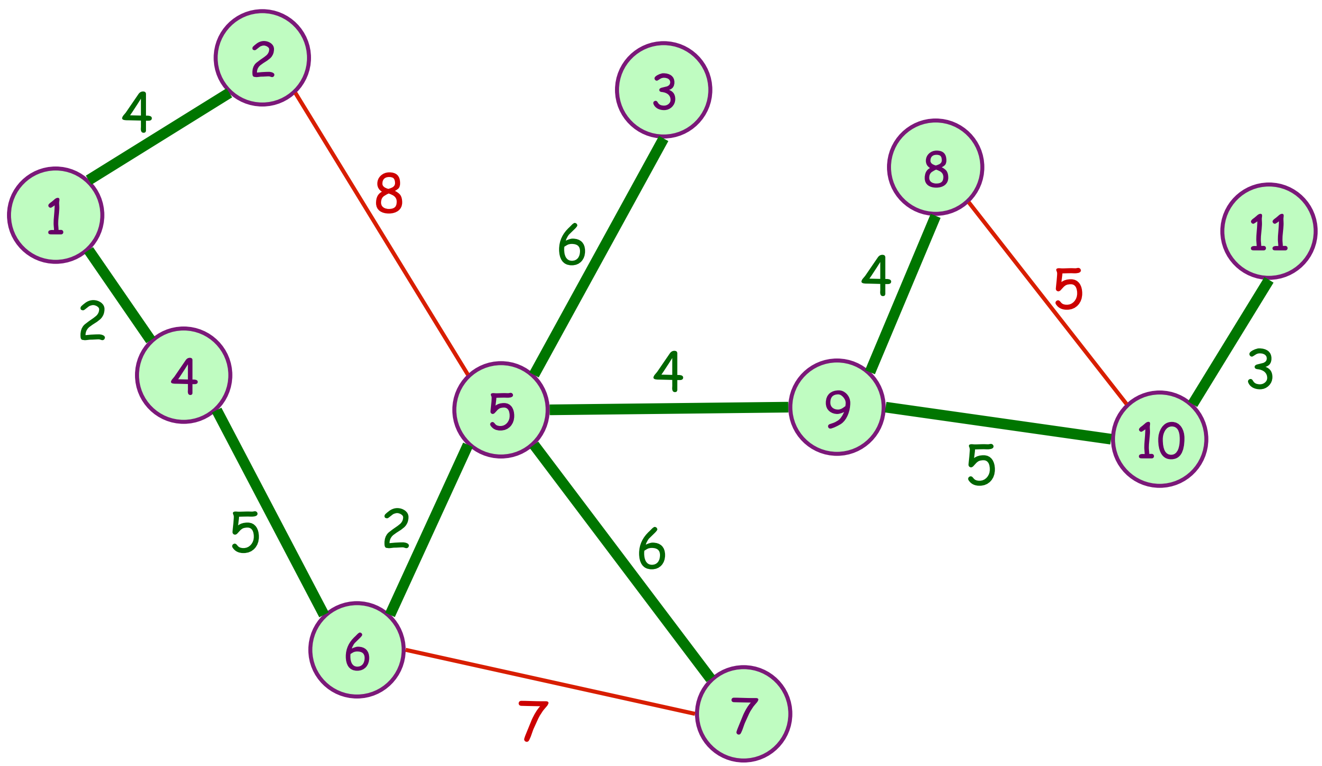
- Tree cost is sum of edge weights.

Weight of a Spanning Tree



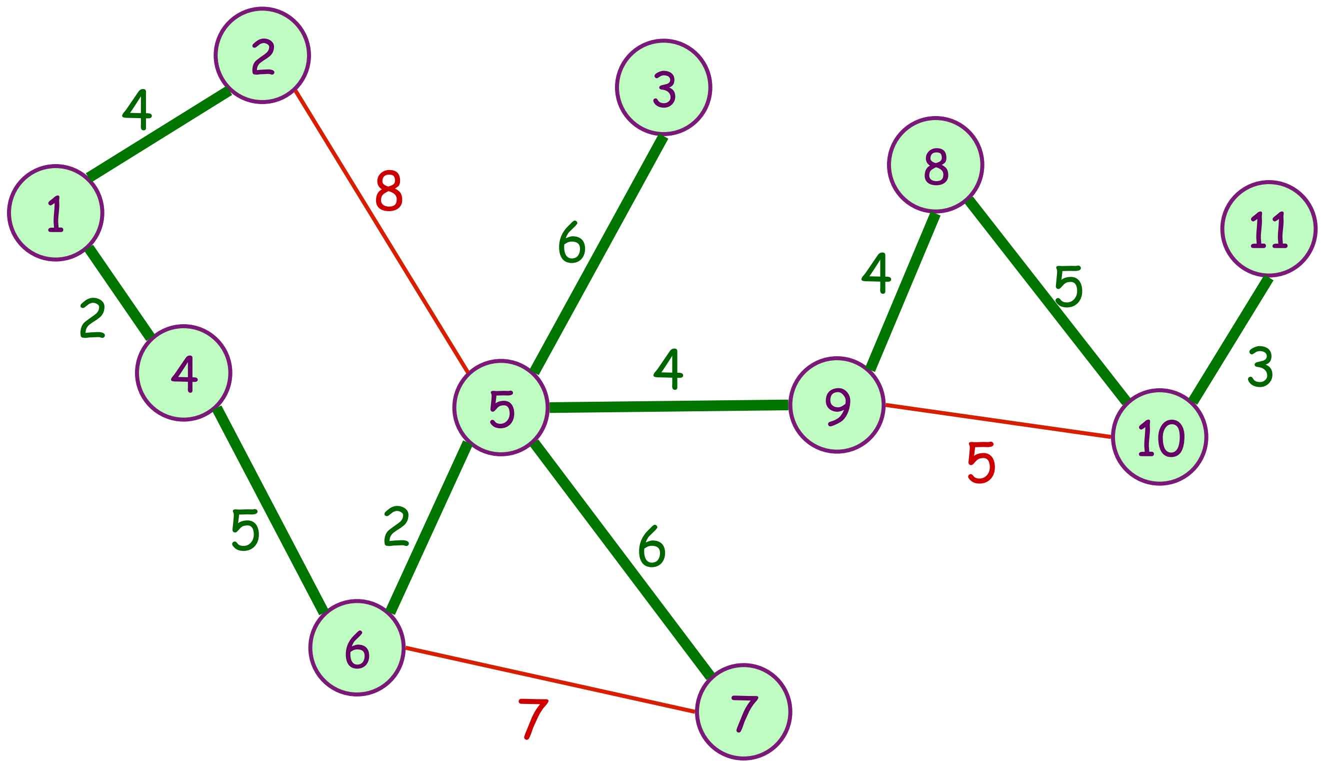
- Spanning tree weight = 48

Minimum Spanning Tree



- Spanning tree weight = 41 … minimum weight for given graph

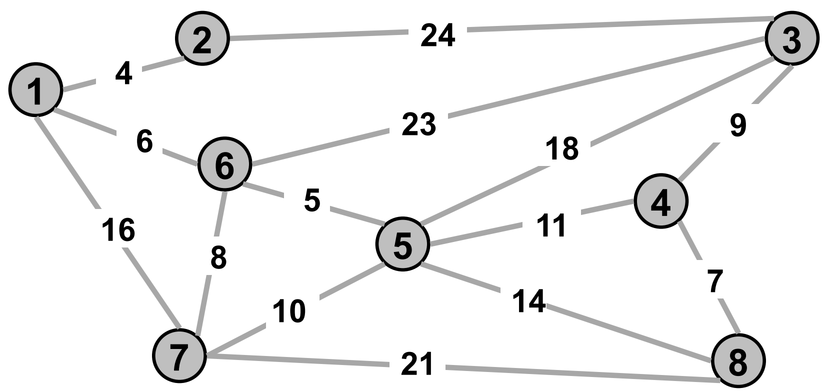
Another Minimum Spanning Tree



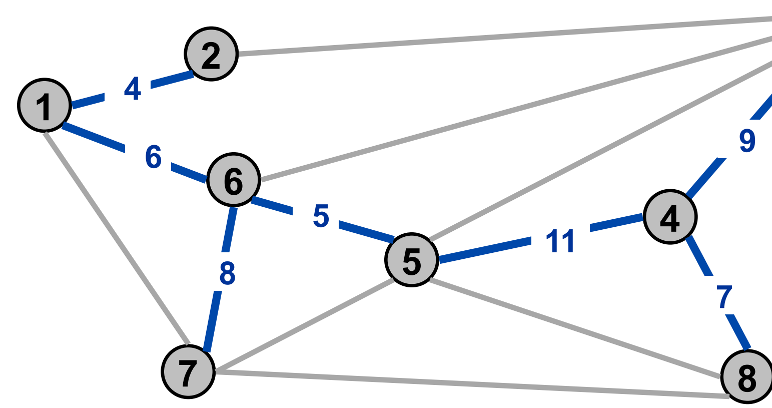
- Spanning tree weight = 41 … minimum weight for given graph

Minimum Spanning Tree

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



G = (V, E)



T = (V, F)

w(T) = 50

- Cayley's Theorem (1889): There are V^{V-2} spanning trees on the complete graph on V vertices.

  ⇒ Can't solve MST by brute force.

MST Origin

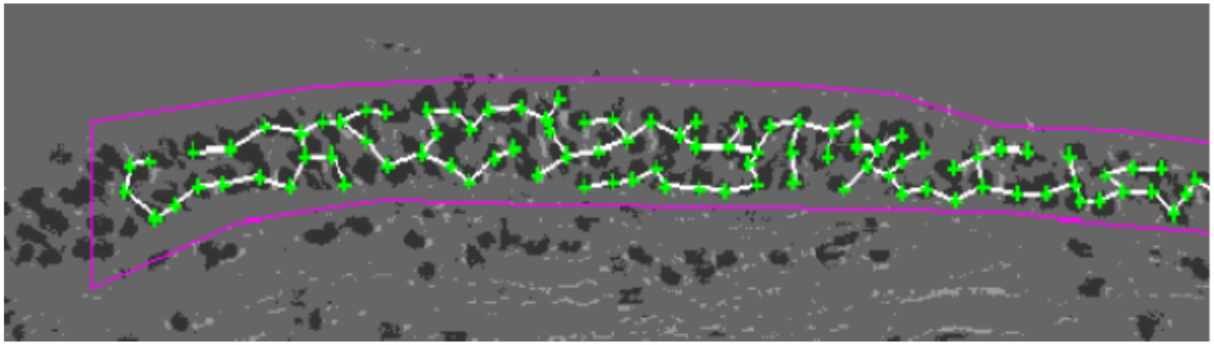
- Otakar Boruvka (1926).
  - Electrical Power Company of Western Moravia in Brno (Czech).
  - Most economical construction of electrical power network.
  - Concrete engineering problem is now a cornerstone problem in combinatorial optimization.

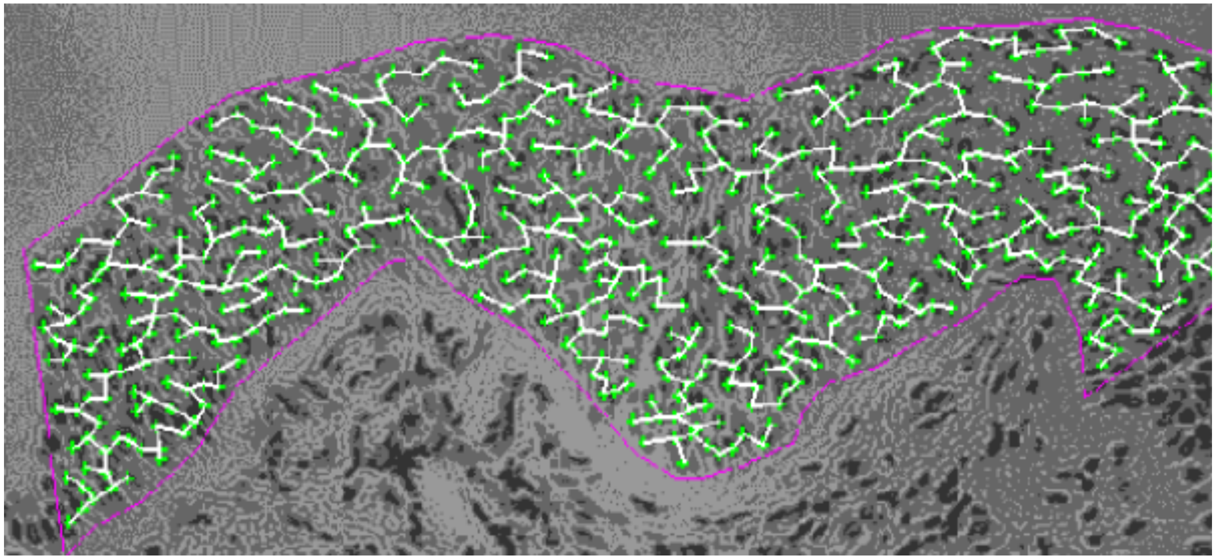
Applications

- MST is fundamental problem with diverse applications.
  - Network design
    - telephone, electrical, hydraulic, TV cable, computer, road
  - Cluster analysis
    - analyzing fungal spore spatial patterns
    - microarray gene expression data clustering
    - finding clusters of quasars and Seyfert galaxies
  - Approximation algorithms for NP-hard problems
    - traveling salesperson problem, Steiner tree
  - Indirect applications
    - max bottleneck paths
    - 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

Medical Image Processing

MST describes arrangement of nuclei
in the epithelium for cancer research

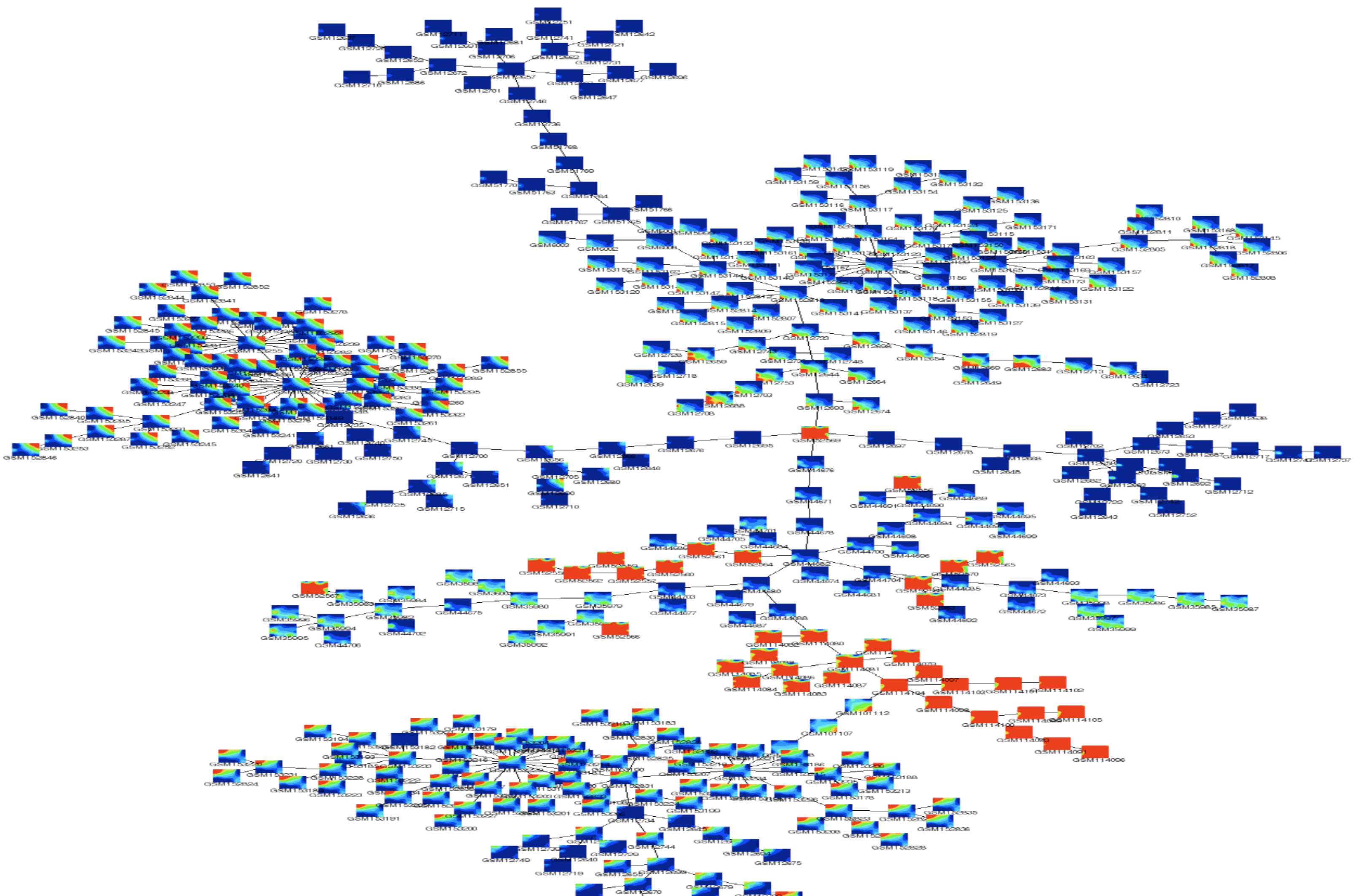




http://www.bccrc.ca/ci/ta01_archlevel.html

Genetic Research

MST of tissue relationships measured by gene expression correlation coefficient

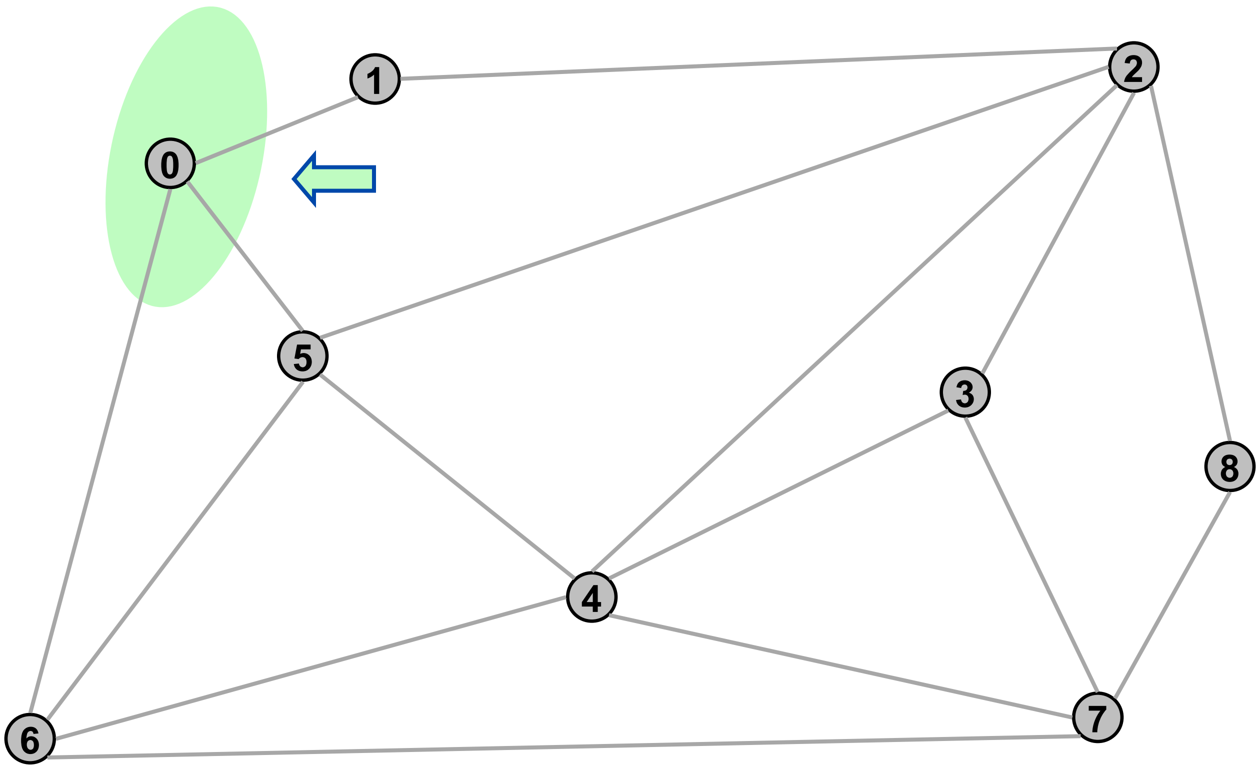


http://riodb.ibase.aist.go.jp/CELLPEDIA

Prim's Algorithm

- Prim's algorithm. (Jarník 1930, Dijkstra 1957, Prim 1959)
  - Initialize T = ϕ, S = {s} for some arbitrary vertex s.

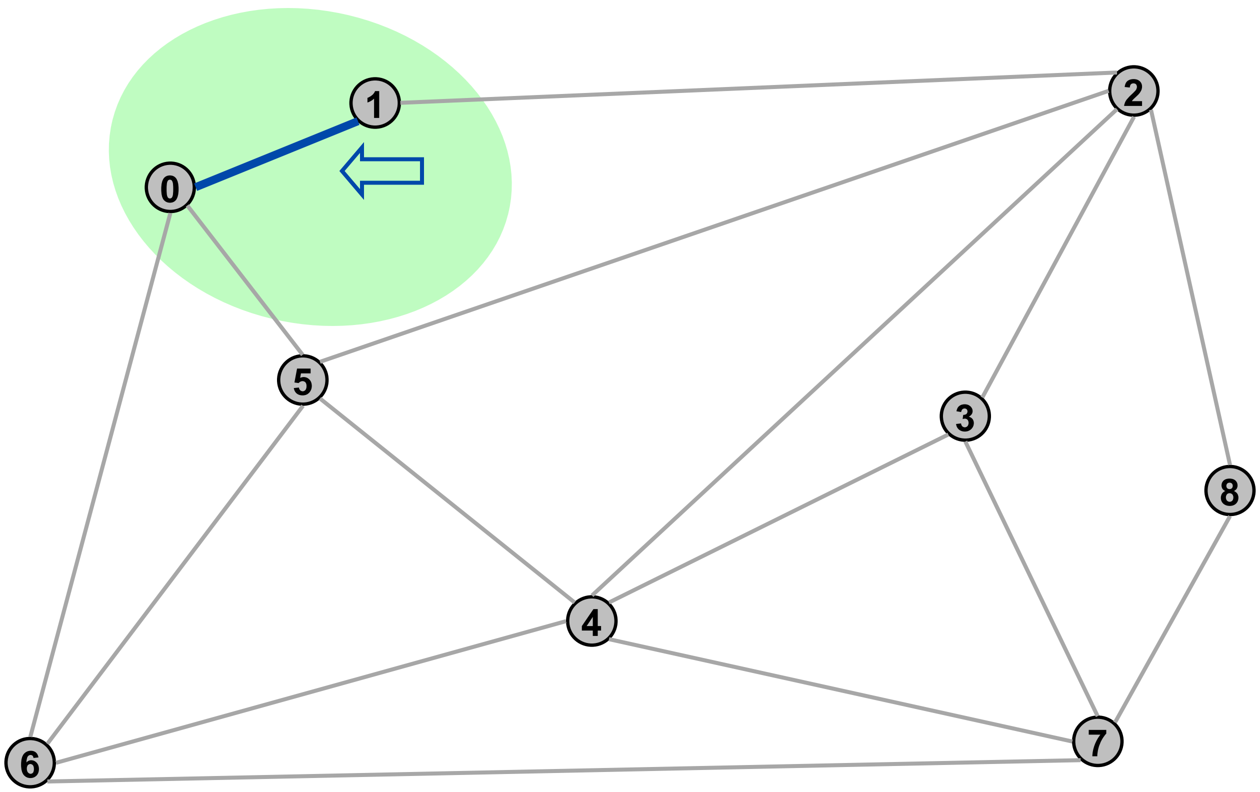
S	T
0	-



Prim's Algorithm

- Prim's algorithm. (Jarník 1930, Dijkstra 1957, Prim 1959)
  - Initialize T = ϕ, S = {s} for some arbitrary vertex s.
  - Grow S until it contains all of the vertices:
    - let f be smallest edge with exactly one endpoint in S

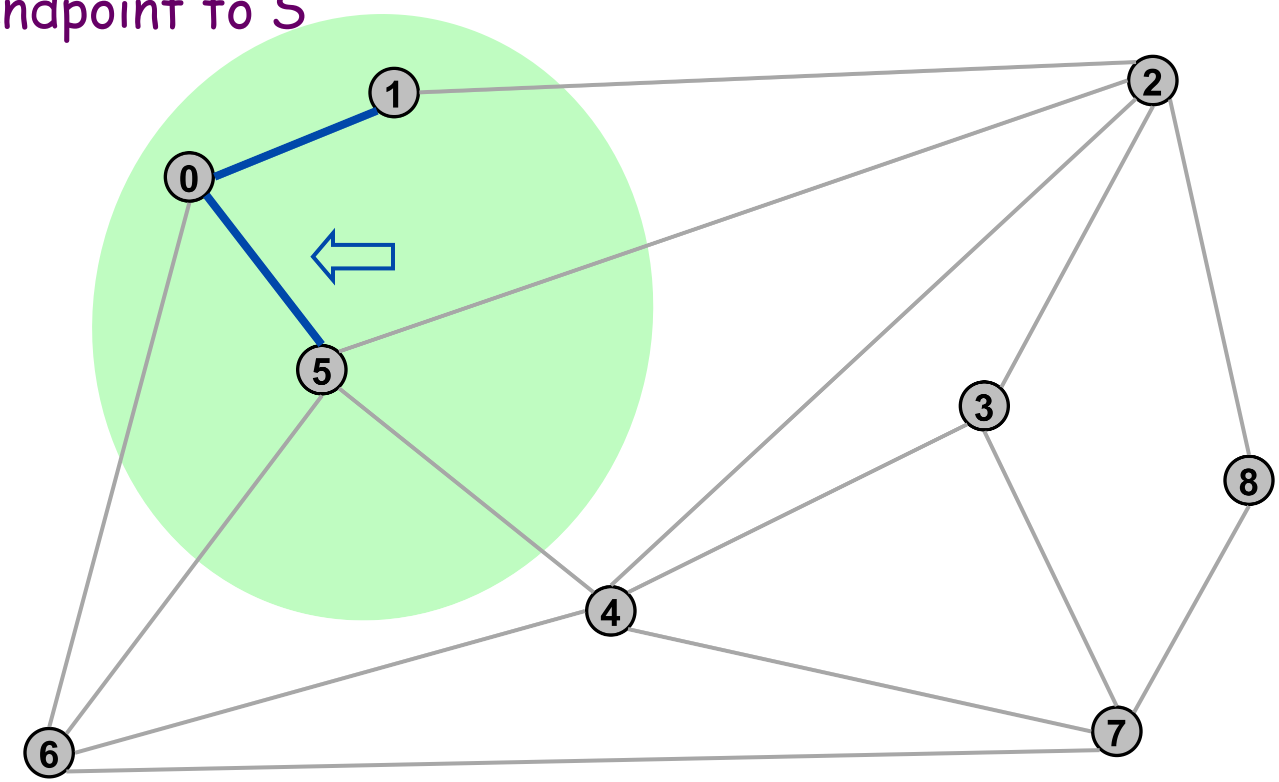
S	T
0	-
1	0-1



Prim's Algorithm

- Prim's algorithm. (Jarník 1930, Dijkstra 1957, Prim 1959)
  - Initialize T = ϕ, S = {s} for some arbitrary vertex s.
  - Grow S until it contains all of the vertices:
    - let f be smallest edge with exactly one endpoint in S
    - add edge f to T
    - add other endpoint to S

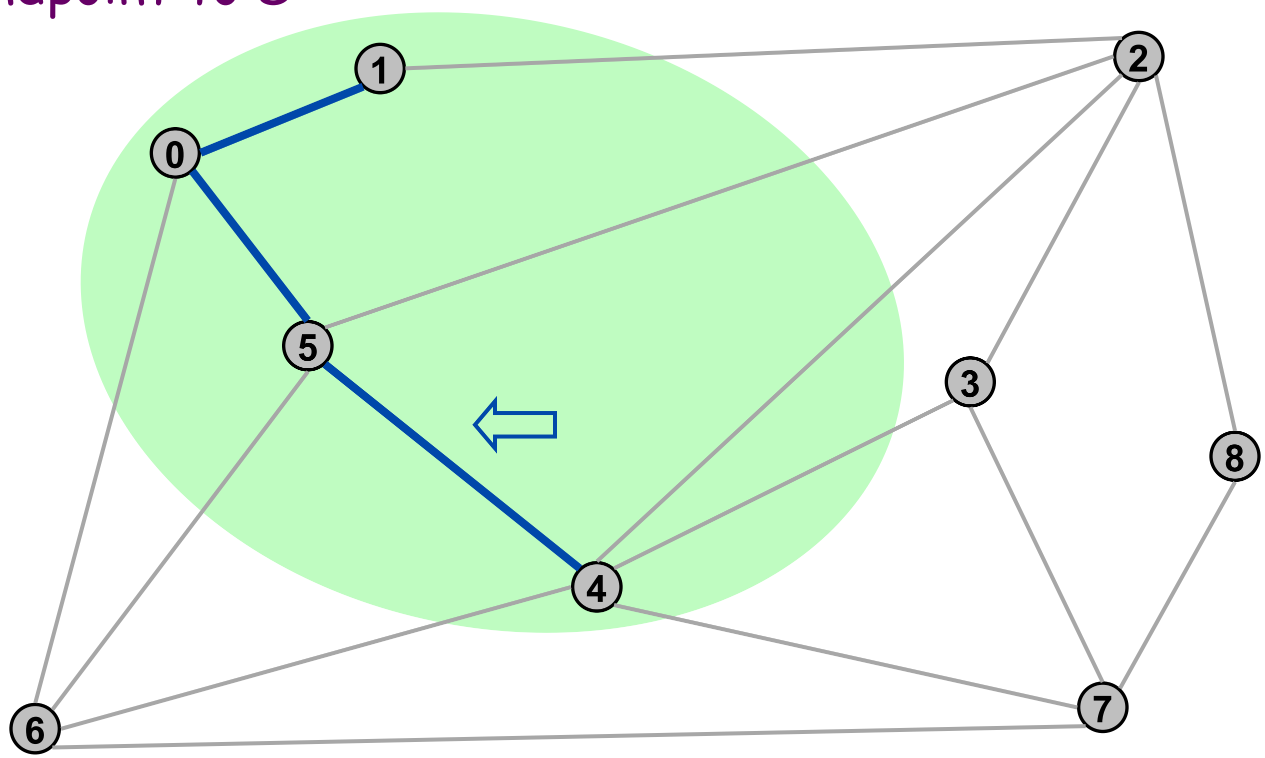
S	T
0	-
1	0-1
5	0-5



Prim's Algorithm

- Prim's algorithm. (Jarník 1930, Dijkstra 1957, Prim 1959)
  - Initialize T = ϕ, S = {s} for some arbitrary vertex s.
  - Grow S until it contains all of the vertices:
    - let f be smallest edge with exactly one endpoint in S
    - add edge f to T
    - add other endpoint to S

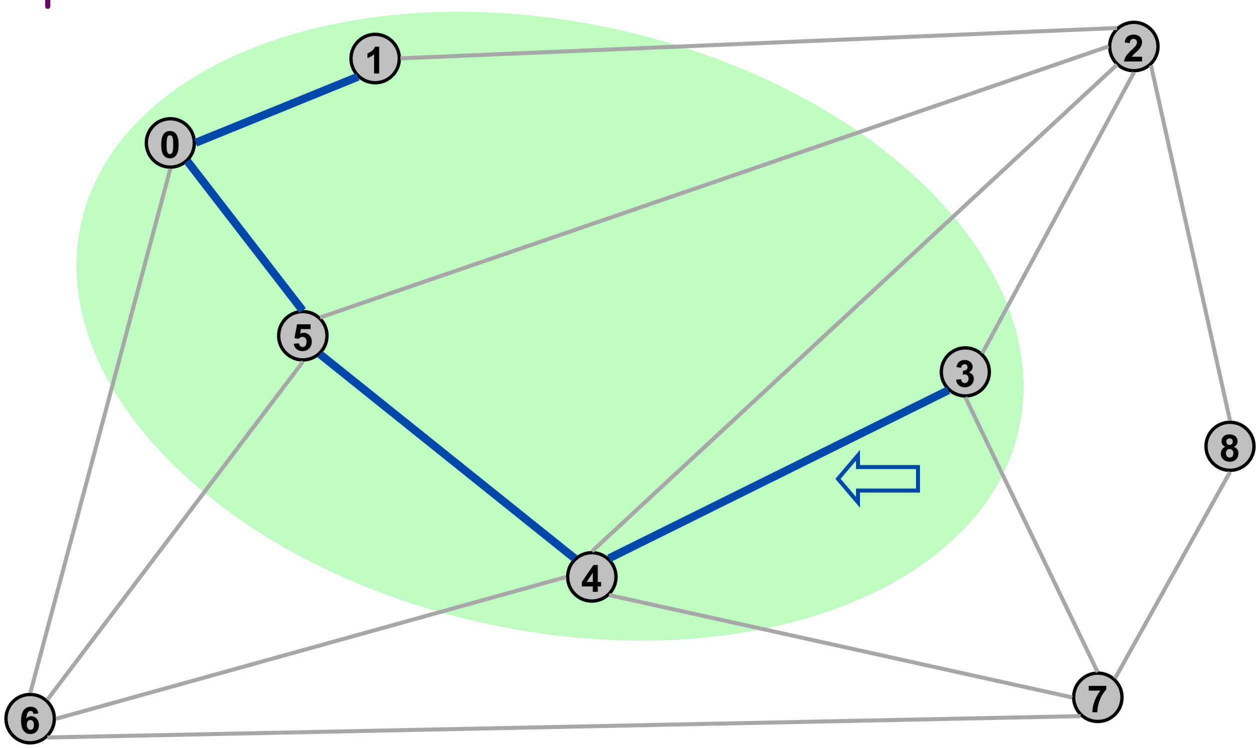
S	T
0	-
1	0-1
5	0-5
4	5-4



Prim's Algorithm

- Prim's algorithm. (Jarník 1930, Dijkstra 1957, Prim 1959)
  - Initialize T = ϕ, S = {s} for some arbitrary vertex s.
  - Grow S until it contains all of the vertices:
    - let f be smallest edge with exactly one endpoint in S
    - add edge f to T
    - add other endpoint to S

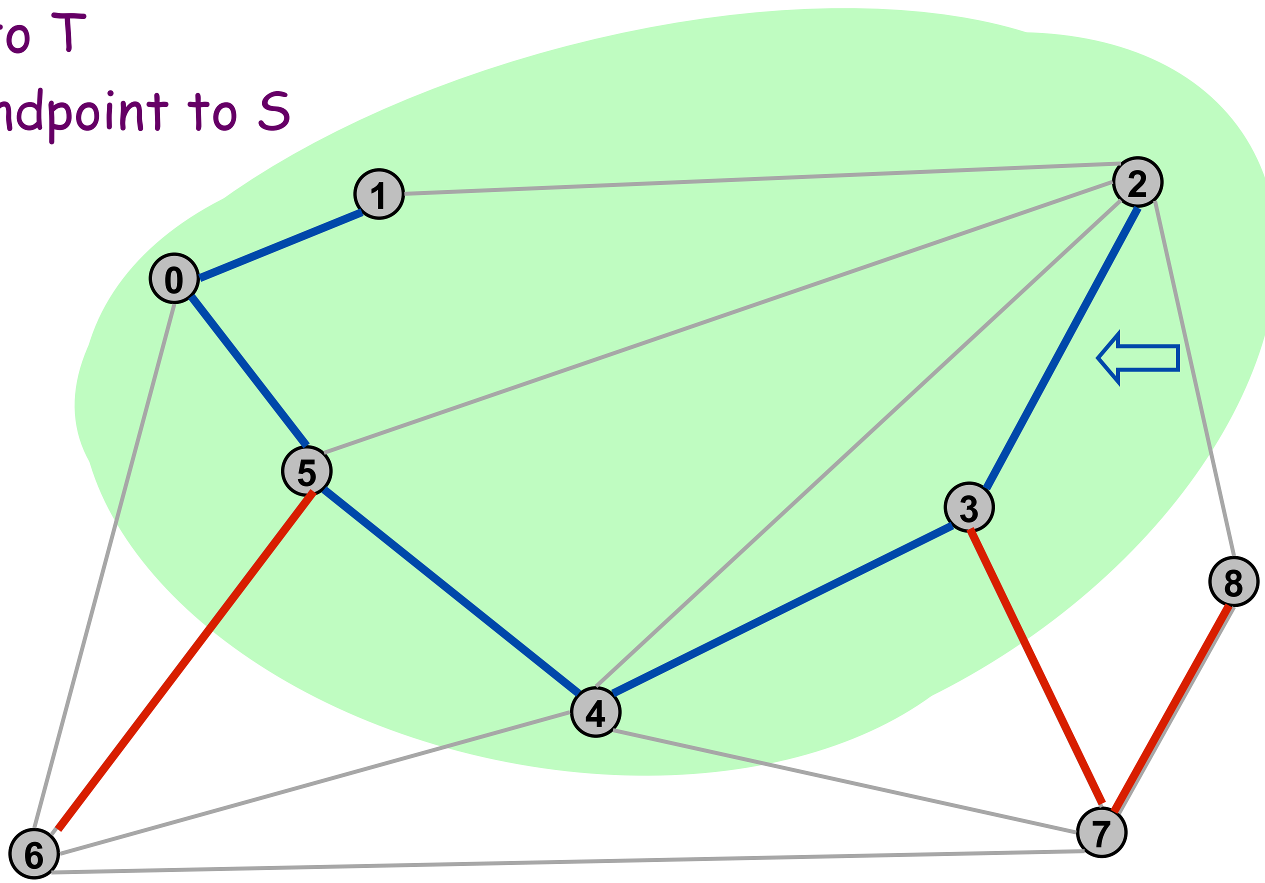
S	T
0	-
1	0-1
5	0-5
4	5-4
3	4-3



Prim's Algorithm

- Prim's algorithm. (Jarník 1930, Dijkstra 1957, Prim 1959)
  - Initialize T = ϕ, S = {s} for some arbitrary vertex s.
  - Grow S until it contains all of the vertices:
    - let f be smallest edge with exactly one endpoint in S
    - add edge f to T
    - add other endpoint to S

S	T
0	-
1	0-1
5	0-5
4	5-4
3	4-3
2	3-2



An Optimization Problem

- A problem in which some function is to be optimized (usually minimized or maximized) subject to some constraints.

- An example of an optimization problem: Machine Scheduling
  - Find a schedule that minimizes the finish time
    - optimization function …
      - finish time
    - constraints
      - each job is scheduled continuously on a single machine for an amount of time equal to its processing requirement
      - no machine processes more than one job at a time

Another Optimization Problem

- A problem in which some function is to be optimized (usually minimized or maximized) subject to some constraints.

- An example of an optimization problem: MST
  - Find a spanning tree that has minimum cost.
    - optimization function ...
      - sum of edge costs
    - constraints
      - must select n-1 edges of the given n vertex graph
      - the selected edges must form a tree

Greedy Method

- Solve (optimization) problem by making a sequence of decisions.
- Decisions are made one by one in some order.
- Each decision is made using a greedy criterion.
- A decision, once made, is (usually) not changed later.

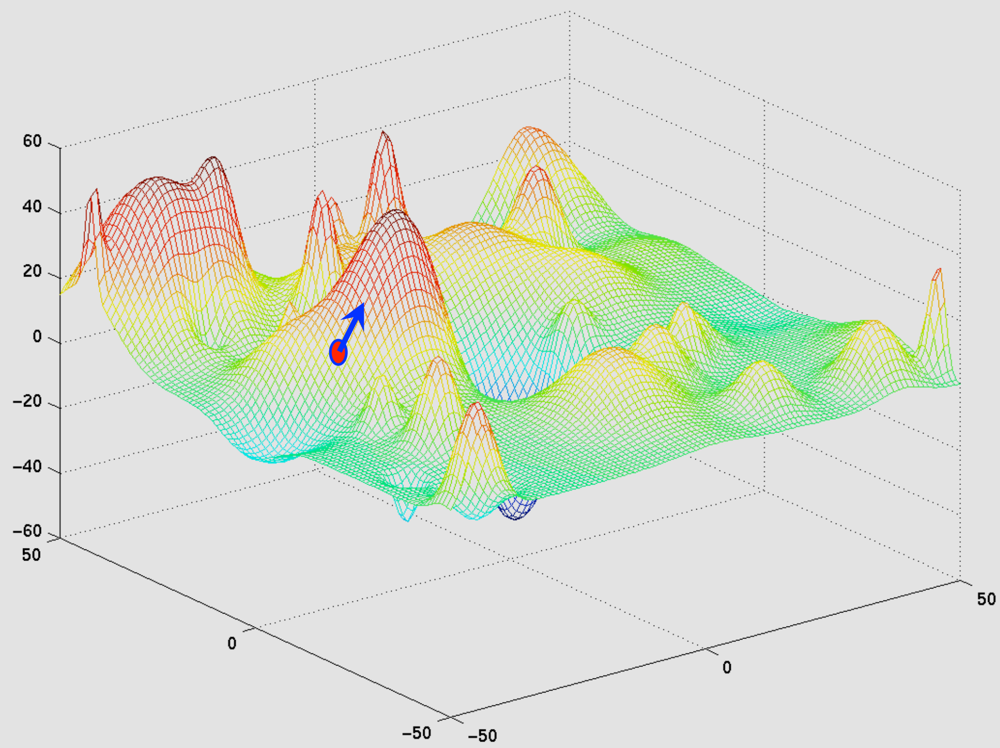
- Example: LPT Scheduling
  - Schedule jobs one by one and in decreasing order of processing time.
  - Each job is scheduled on the machine on which it finishes earliest.
  - Scheduling decisions are made serially using a greedy criterion (minimize finish time of this job).
  - LPT scheduling is an application of the greedy method.

- However, recall that LPT does not guarantee an optimal solution! Although, it does offer a heuristic approximate solution.

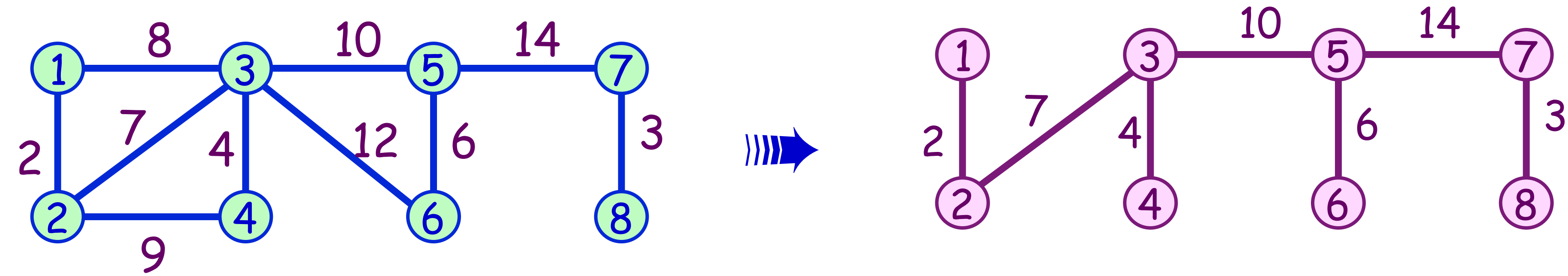
Greedy Method

- Solve problem by making a sequence of decisions.
- Decisions are made one by one in some order.
- Each decision is made using a greedy criterion.
- A decision, once made, is (usually) not changed later.
- Example: LPT Scheduling

So what is going on here?



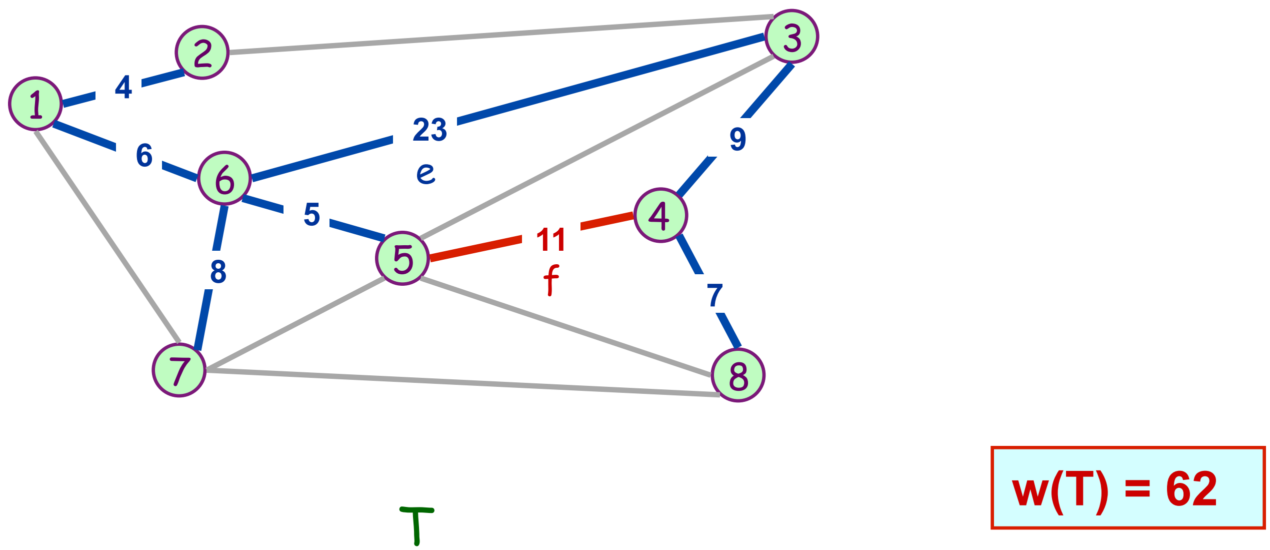
Prim's Algorithm: Example



- Start with any single vertex tree.
- Get a 2 vertex tree by adding a cheapest edge.
- Get a 3 vertex tree by adding a cheapest edge.
- Grow the tree one edge at a time until the tree has n - 1 edges (and hence has all n vertices).

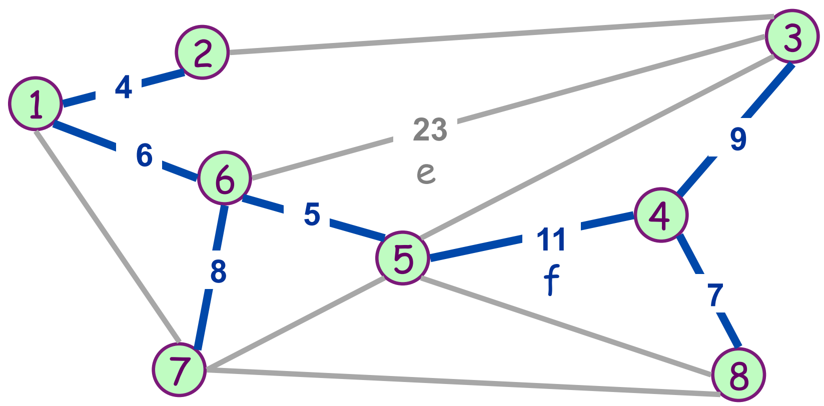
Prim's Algorithm: Intuition of Proof of Correctness

- Observation: Given a spanning tree T. Let f be an edge not in T. Adding f to T creates a unique cycle. If $c_f < c_e$ for some edge e of cycle, then $T \cup \{ f \} - \{ e \}$ is a tree of lower cost.



Prim's Algorithm: Intuition of Proof of Correctness

- Observation: Given a spanning tree T. Let f be an edge not in T. Adding f to T creates a unique cycle. If $c_f < c_e$ for some edge e of cycle, then $T \cup \{ f \} - \{ e \}$ is a tree of lower cost.



$T \cup \{ f \} - \{ e \}$

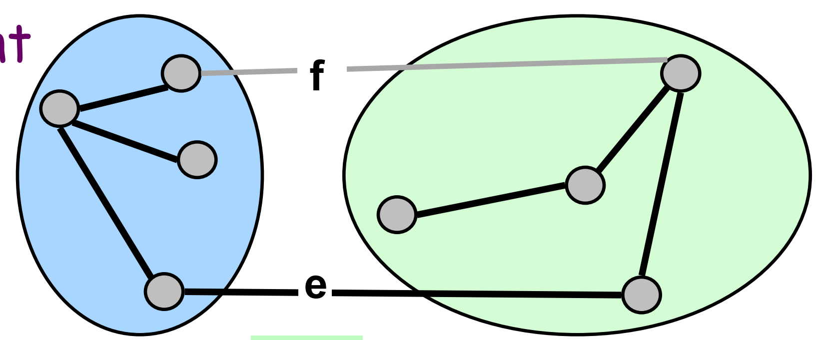
w(T) = 50

Prim's Algorithm: Proof of Correctness

- Theorem: Upon termination of Prim's algorithm, T is a MST.
- Proof: By induction on number of iterations

Invariant: There exists a MST T* containing all of the edges in T.

- Base case: $T = \phi$ every MST satisfies invariant.
- Induction step: assume invariant true at beginning of iteration i.
  - Let f be the edge that Prim's algorithm chooses.
  - If $f \in T^*$, T^* still satisfies invariant.
  - Otherwise, consider cycle C formed by adding f to T*
    - let $e \in C$ be another arc with exactly one endpoint in S
    - $c_f \leq c_e$ since algorithm chooses f instead of e
    - $T^* \cup \{ f \} - \{ e \}$ satisfies invariant

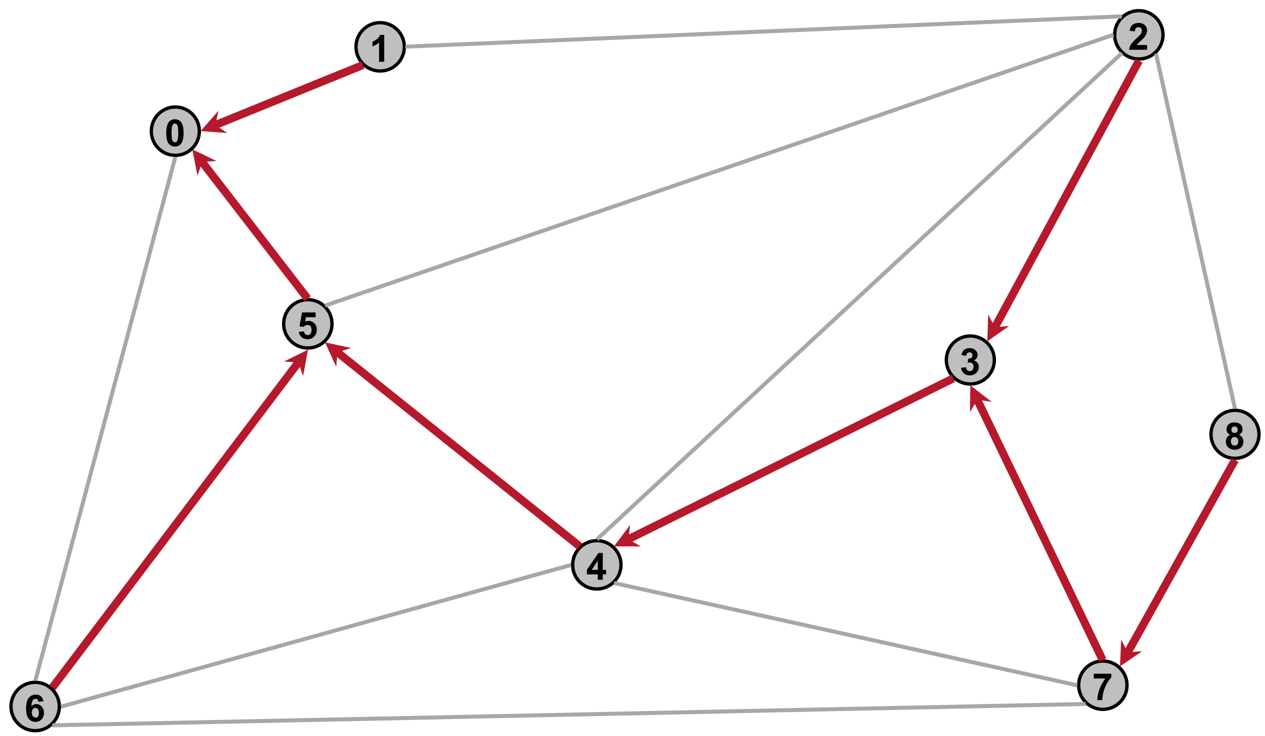


T*

Spanning Tree Representation

- How to represent a spanning tree?
  - List of edges: 0-1 0-5 2-3 3-4 3-7 4-5 5-6 7-8
  - Parent-link representation: vertex indexed array pred[v].

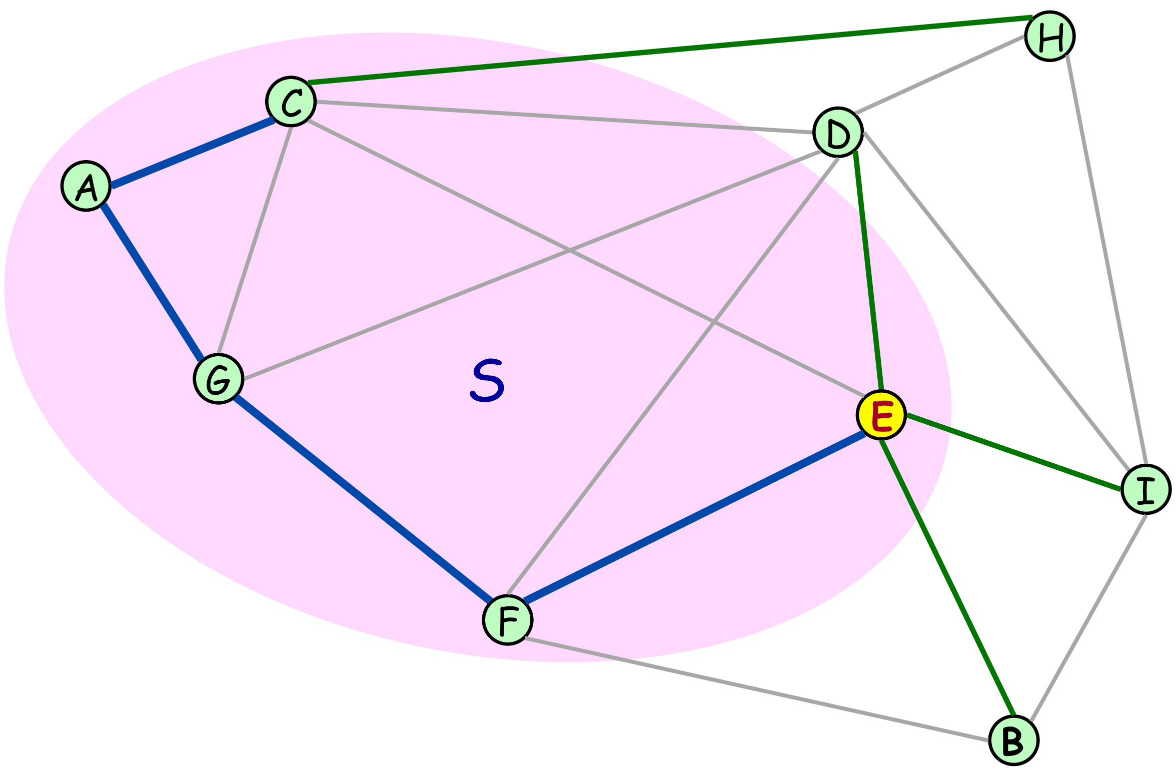
v	pred[v]
0	-
1	0
2	3
3	4
4	5
5	0
6	5
7	3
8	8



Prim's Algorithm

- Maintain S = set of vertices in current tree.
  - For each vertex not in S, maintain vertex in S to which it is closest.

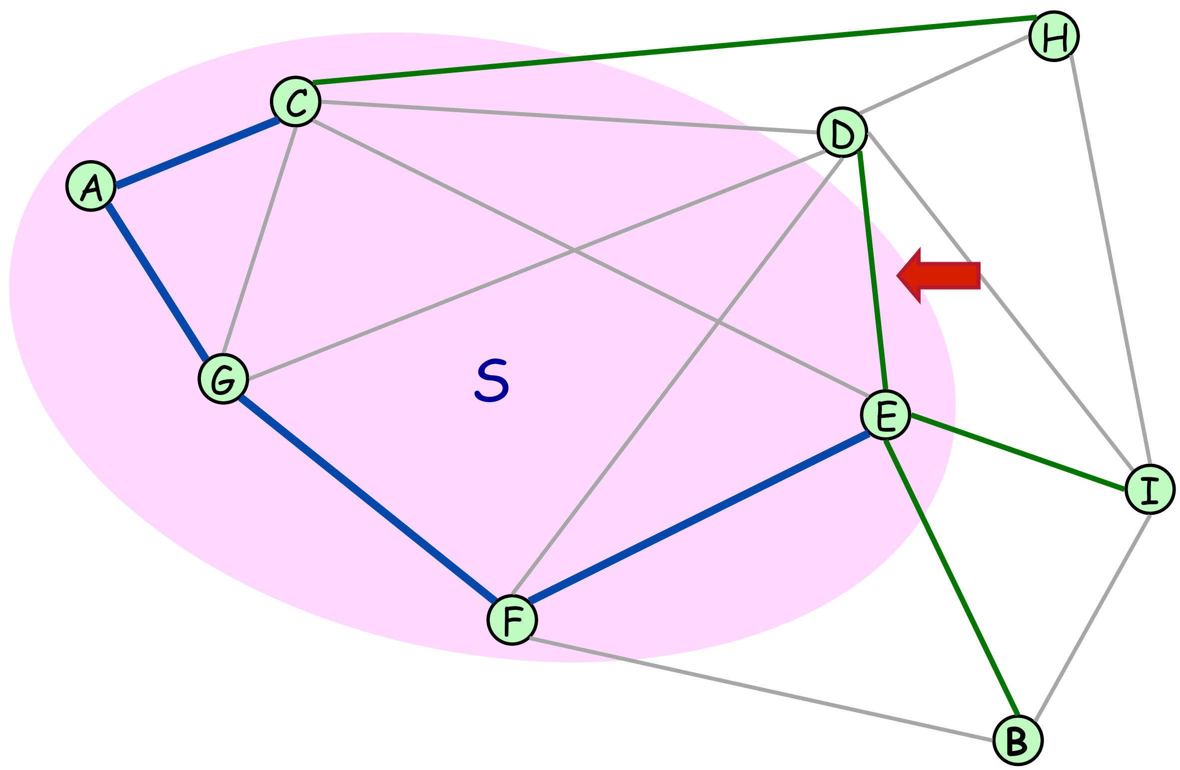
v	pred	dist
A	A	-
B	E	15
C	A	-
D	E	9
E	F	-
F	G	-
G	A	-
H	C	23
I	E	11



Prim's Algorithm

- Maintain S = set of vertices in current tree.
  - For each vertex not in S, maintain vertex in S to which it is closest.
  - Choose next vertex v to add to S with min dist[v].

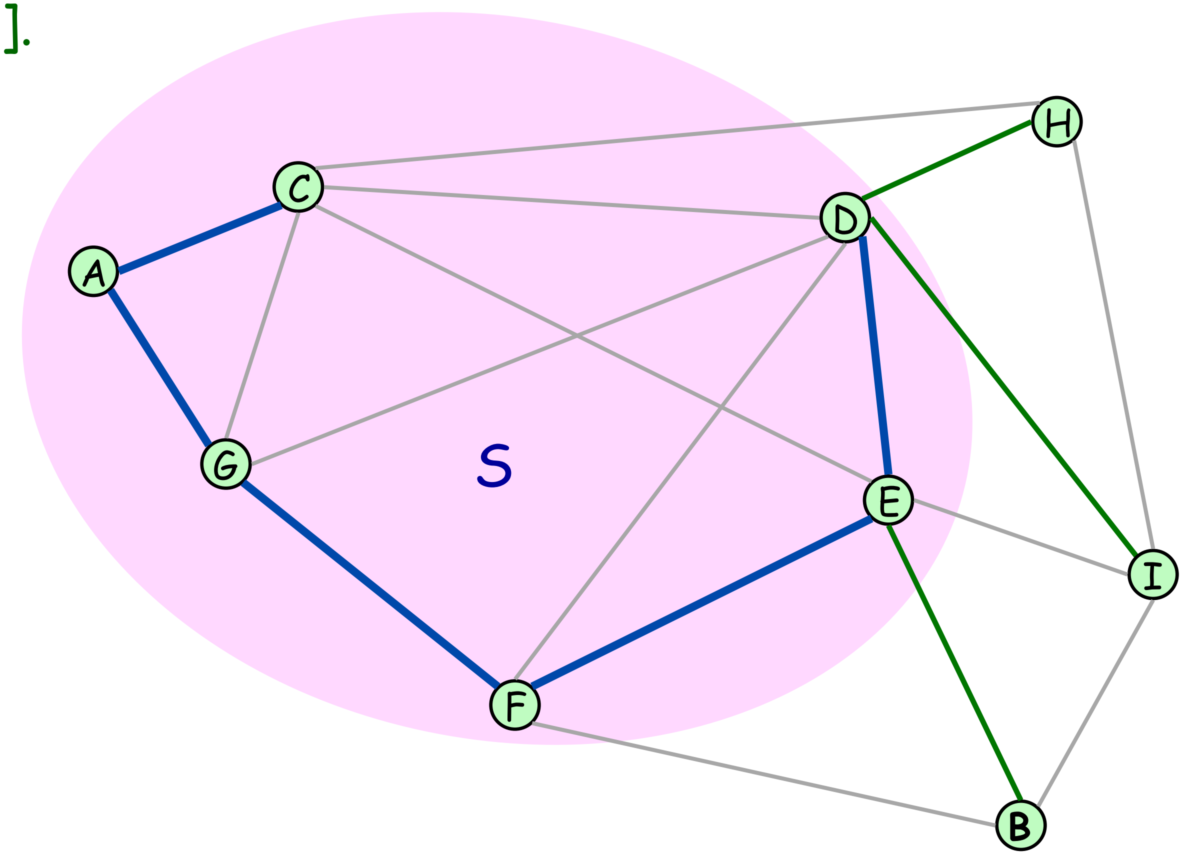
v	pred	dist
A	A	-
B	E	15
C	A	-
D	E	9
E	F	-
F	G	-
G	A	-
H	C	23
I	E	11



Prim's Algorithm

- Maintain S = set of vertices in current tree.
  - For each vertex not in S, maintain vertex in S to which it is closest.
  - Choose next vertex v to add to S with min dist[v].
  - For each neighbor w of v, if w is closer to v than current neighbor in S, update dist[w].

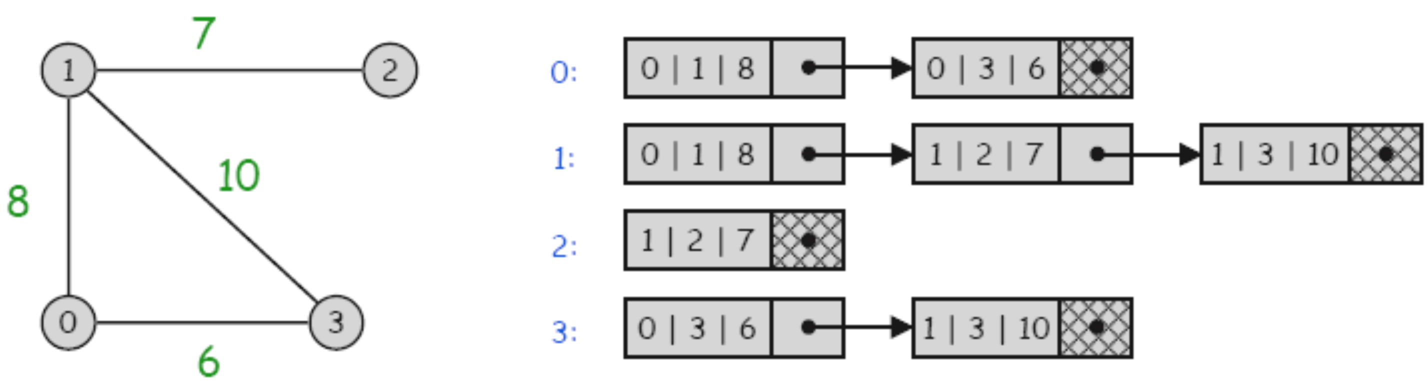
v	pred	dist
A	A	-
B	E	15
C	A	-
D	E	-
E	F	-
F	G	-
G	A	-
H	D	4
I	D	6



Weighted Graphs

- Weights:
  - Method 1: graph access function G.cost(v, w).
  - Method 2: modify adjacency list iterator to return Edge.
- Tradeoffs:
  - Method 1 is easier with adjacency matrix or Euclidean weights.

  Method 2 is more general.



adjacency list of Edge objects

Prim's Algorithm

- Adjacency list implementation.
  - Initialize, dist[v] = and dist[s] = 0.
  - Insert all vertices onto PQ.
  - Repeatedly delete vertex v from PQ with min dist[v].
    - for each v-w, if (dist[w] > G.cost(v, w)), update dist[w]

```
// main loop
while (!pq.isEmpty()) {
   int v = pq.delMin();
   IntIterator i = G.neighbors(v);
   while (i.hasNext()) {
      int w = i.next();
      if (dist[w] > G.cost(v, w)) {
         dist[w] = G.cost(v, w);
         pq.decrease(w, dist[w]);
         pred[w] = v;
      }
   }
}
```

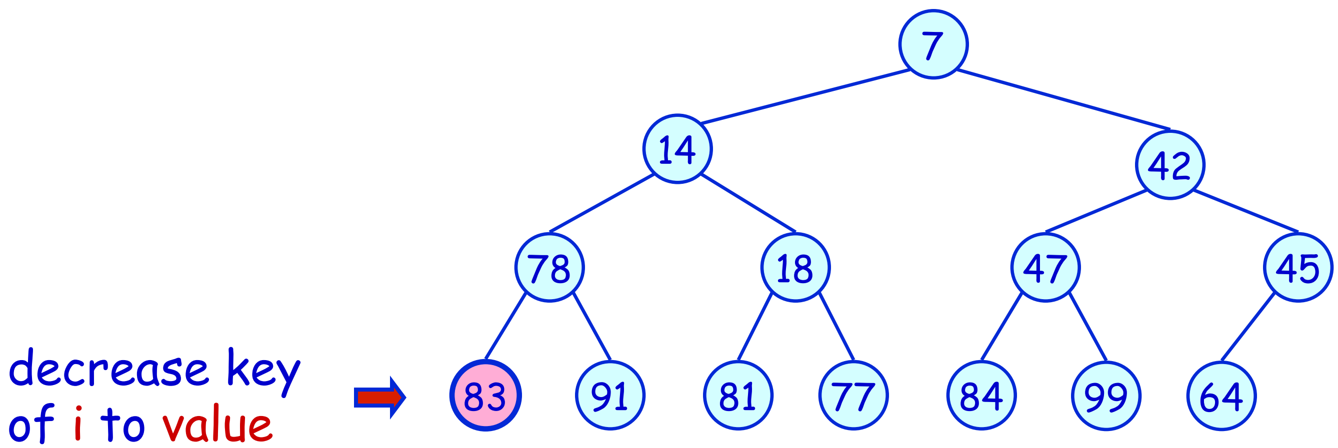
Priority Queues for Index Items

- Index heap-based priority queue:
  - Insert, delete min, test if empty
  - Decrease key
- Brute force array implementation:
  - Maintain vertex indexed array dist[w].
  - Decrease key: change dist[w].
  - Delete min: scan through dist[w] for each vertex w.

Operation	Prim	Array
insert	V	V
delete-min	V	V
decrease-key	E	1
is-empty	V	1
total		V^2

Priority Queues for Index Items

- Index heap-based priority queue:
  - Assumes elements are named 0 to N-1
  - Assumes priorities are of type double
  - Client: pq.decrease(i, value)



- How to decrease key of vertex i? Bubble it up.
- How to know which heap node to bubble up? Maintains an extra array qp[i] that stores the heap index of vertex i

Priority Queues for Index Items

- Design issues:
  - PQ maintains priorities; client accesses through PQ interface
  - Client maintains priorities; PQ accesses through client
  - Both maintain their own copy

```
public void insert(int k, double value) {
   N++;
   pq[N] = k;
   qp[k] = N;
   priority[k] = value;
   fixUp(pq, N);
}

public void decrease(int k, double value) {
   priority[k] = value;
   fixUp(pq, qp[k]);
}

private void exch(int i, int j) {
   int swap = qp[i]; qp[i] = qp[j]; qp[j] = swap;
   pq[qp[i]] = i; pq[qp[j]] = j;
}
```

Prim's Algorithm: Priority Queue Choice

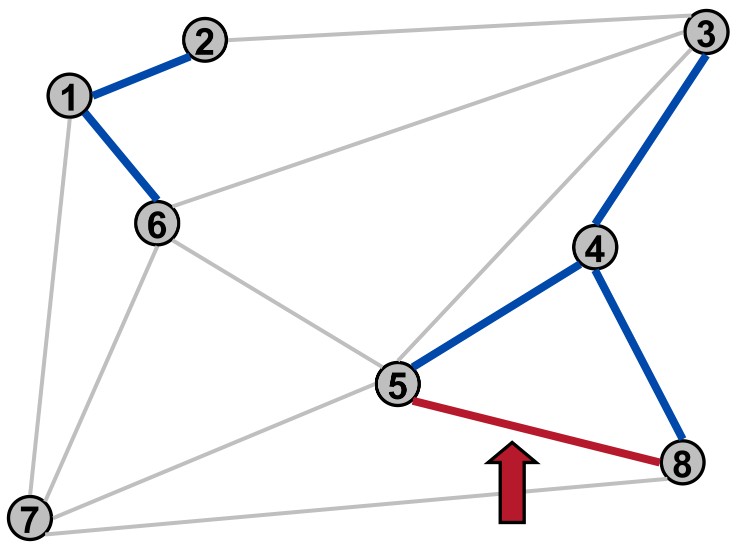
- The choice of priority queue matters in Prim implementation
  - Array: $\Theta(V^2)$
  - Binary heap: $O(E \log V)$
  - Fibonacci heap: $O(E + V \log V)$

- Best choice depends on whether graph is SPARSE or DENSE
  - 2,000 vertices, 1 million edges. Heap: 2-3x slower
  - 100,000 vertices, 1 million edges. Heap: 500x faster
  - 1 million vertices, 2 million edges. Heap: 10,000x faster

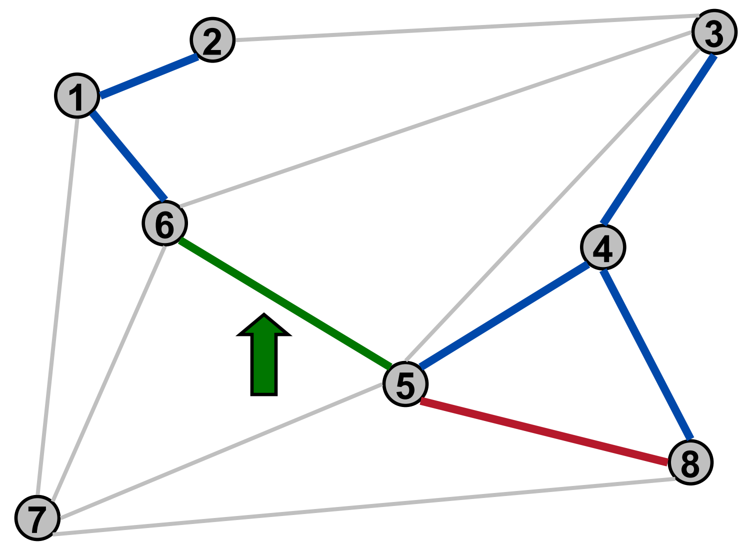
- Bottom line
  - Array implementation optimal for dense graphs.
  - Binary heap far better for sparse graphs.
  - Fibonacci heap best in theory, but not in practice.

Kruskal's Algorithm

- Kruskal's algorithm (1956).
  - Initialize forest F = ϕ.
  - Consider edges in ascending order of weight.
  - If adding edge e to forest F does not create a cycle, then add it. Otherwise, discard e.

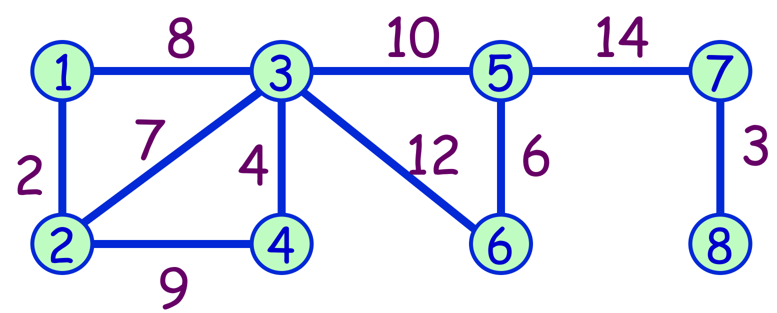


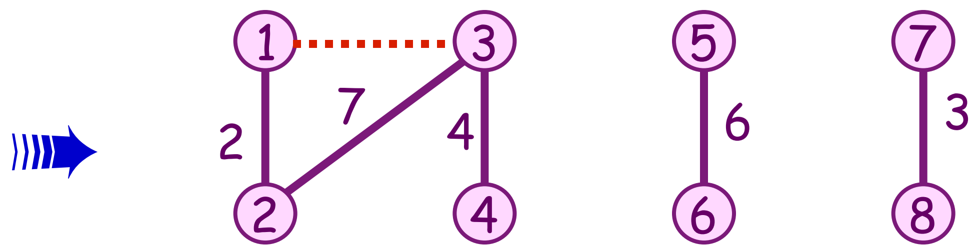
Case 1: adding 5-8 creates a cycle



Case 2: adding 5-6 connects 2 components

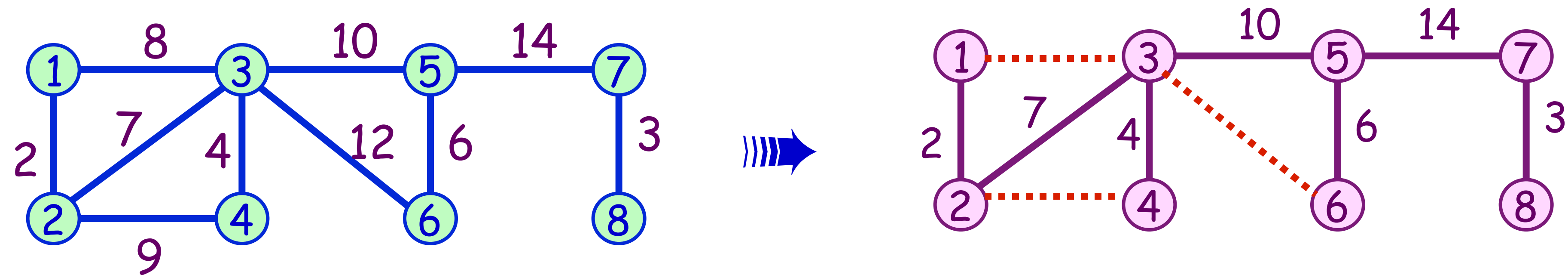
Kruskal's Algorithm: Example





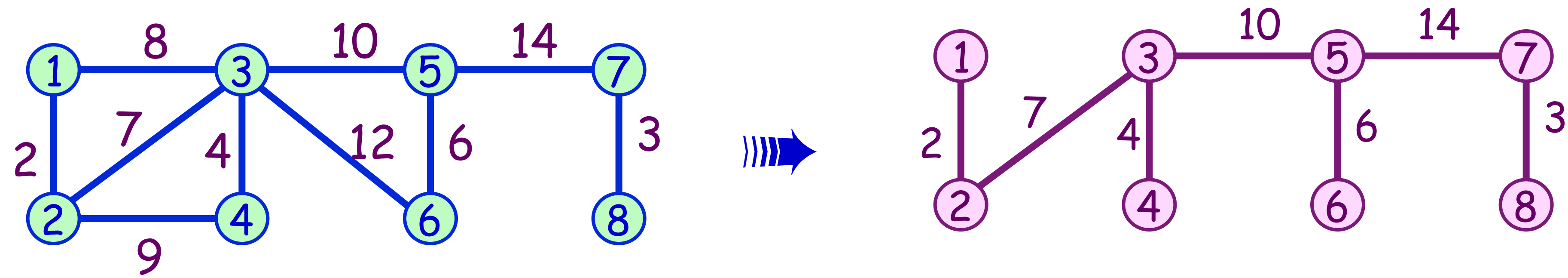
- Start with a forest that has no edges.
- Consider edges in ascending order of cost.
  - Edge (1,2) is considered first and added to the forest.
  - Edge (7,8) is considered next and added.
  - Edge (3,4) is considered next and added.
  - Edge (5,6) is considered next and added.
  - Edge (2,3) is considered next and added.
  - Edge (1,3) is considered next but rejected because it creates a cycle

Kruskal's Algorithm: Example



- Start with a forest that has no edges.
- Consider edges in ascending order of cost.
  - Edge (2,4) is considered next but rejected because it creates a cycle.
  - Edge (3,5) is considered next and added to the forest.
  - Edge (3,6) is considered next but rejected.
  - Edge (5,7) is considered next and added to the forest.

Kruskal's Algorithm: Example



- n - 1 edges have been selected and no cycle formed.
- So we must have a spanning tree.
- Cost is 46.
- Min-cost spanning tree is unique when all edge costs are different.

Kruskal's & Prim's Algorithms

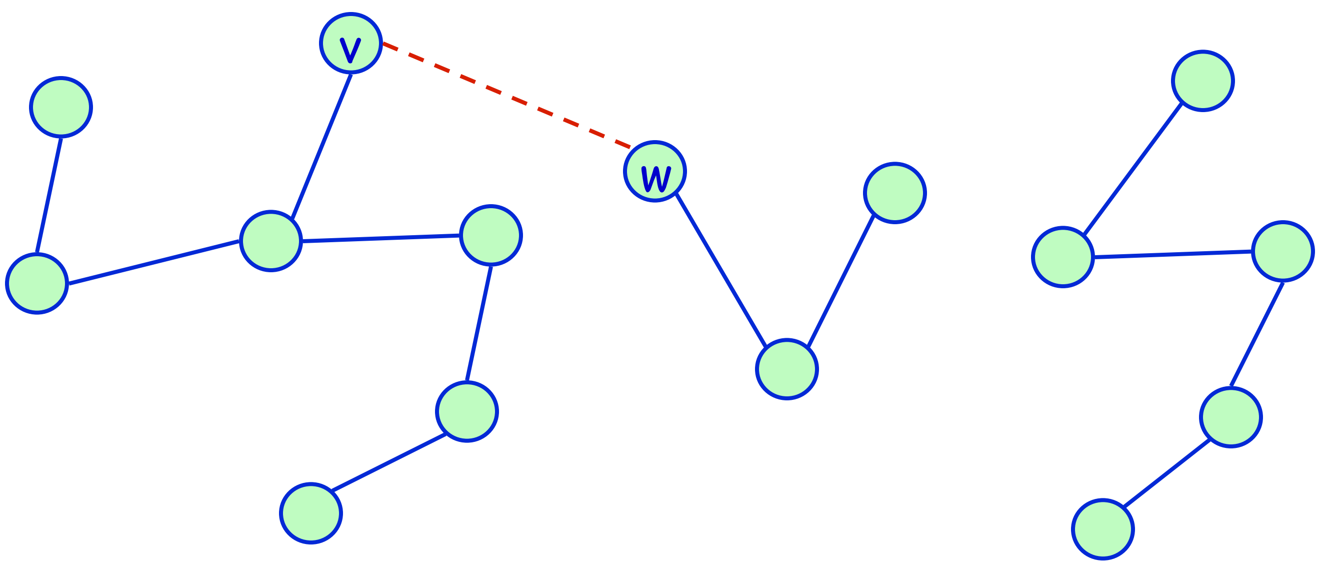
"Greed is good. Greed is right. Greed works. Greed clarifies, cuts through, and captures the essence of the evolutionary spirit."

- Gordon Gecko

Unlike with LPT – the Greedy approach yields an optimal solution for MST problem

Kruskal's Algorithm: Implementation

- How to check if adding an edge to F would create a cycle?
  - Naïve solution: DFS in O(V) time.
  - Clever solution: union-find in O(log* V) amortized time.
    - each tree in forest F corresponds to a set
    - adding v-w creates a cycle if v and w are in same component
    - when adding v-w to forest F, merge sets containing v and w



Kruskal's Algorithm: Implementation

```
public class MST {
   private Edge[] mst;               // list of all edges in mst

   public MST(Graph G) {
      mst = new Edge[G.V()];
      Edge[] edges = G.edges();   // list of all edges in G
      Arrays.sort(edges);         // sort them by weight

      UnionFind uf = new UnionFind(G.V());
      for (int i = 0, k = 1; i < G.E(); i++) {
         int v = edges[i].v();
         int w = edges[i].w();
         if (!uf.find(v, w)) {    // v-w does not create a cycle
            uf.unite(v, w);       // merge v and w components
            mst[k++] = edges[i]; // add edge to mst
         }
      }
   }
}
```

Kruskal's Algorithm: Running Time

Operation	Frequency	Cost
sort	1	E log E
union	V - 1	log* V
find	E	log* V

Amortized bound using weighted quick union with path compression

- Kruskal running time: O(E log V).
- If edges already sorted. O(E log* V) time.
  - recall: log* V ≤ 5 in this universe!