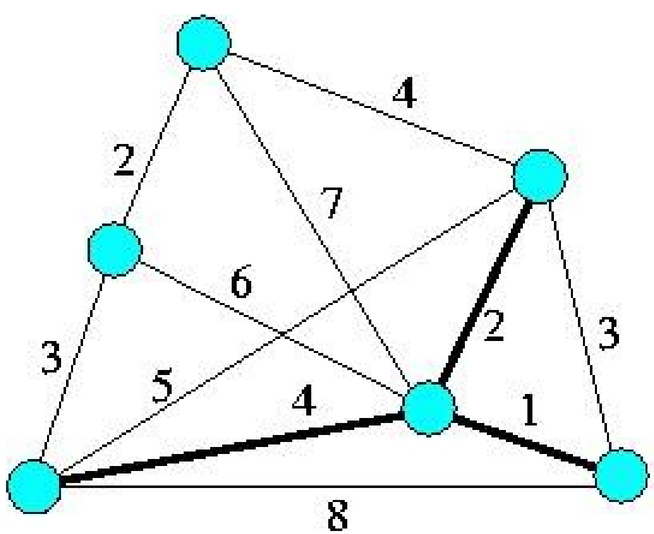
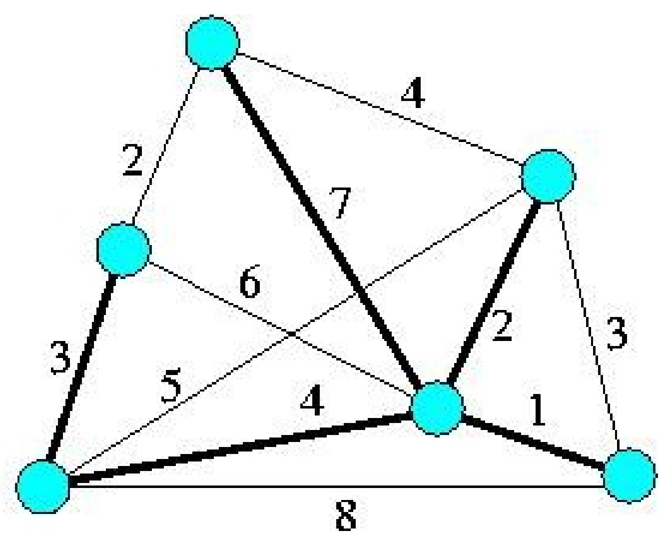
Minimum Spanning Trees

- Theoretical Background
- Prim's Algorithm
- Kruskal's Algorithm
- Application

Some Basic Definitions

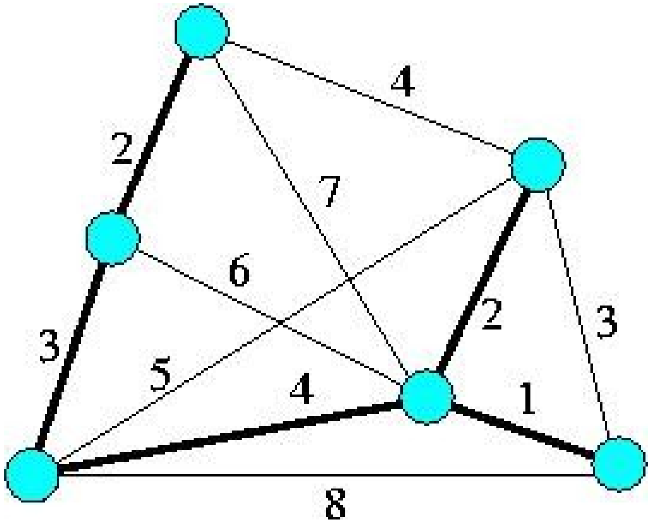
- Tree in a connected graph $G = (V,E,c)$.
- Spanning tree of a connected graph *G.*
- Every spanning tree of a connected graph G with *n.* vertices has *n*-1 edges (proof by induction on *n*).





Problem Formulation

- **Given**: Undirected, connected graph $G=(V,E,c)$ with positive edge costs.
- **Find**: A tree T spanning V with the total cost (sum of edge-costs) minimized.

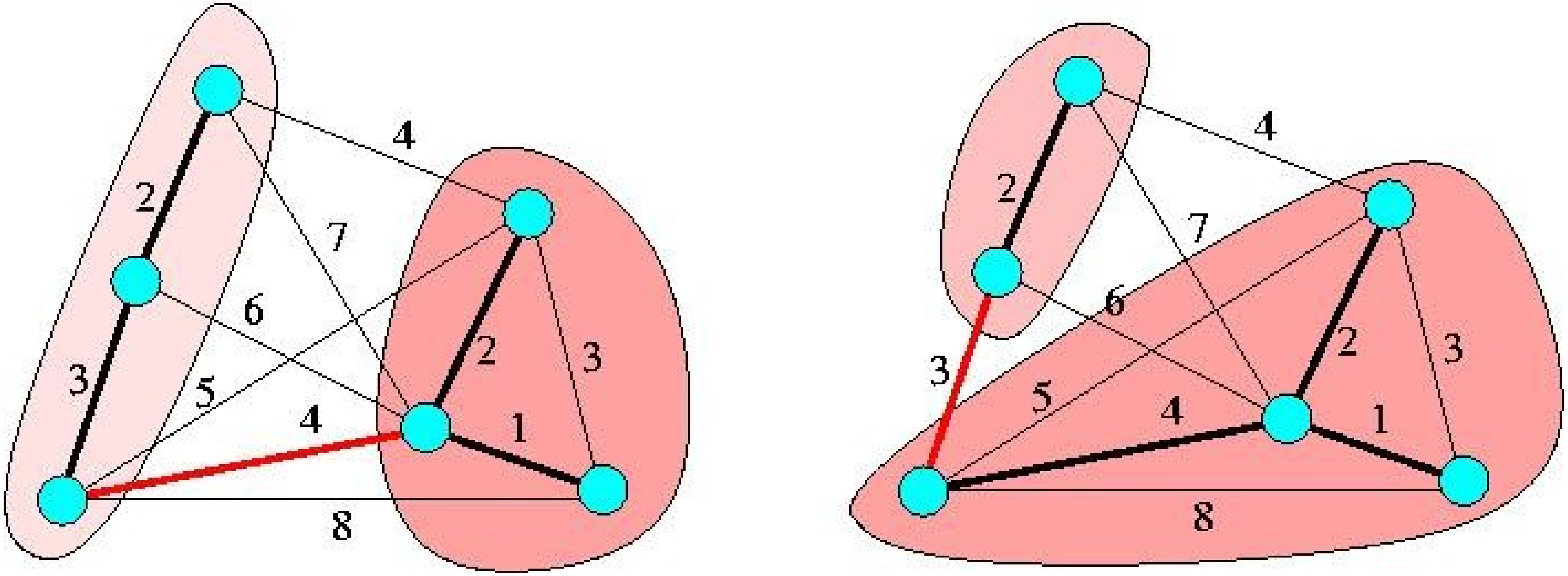


Trivial Algorithm

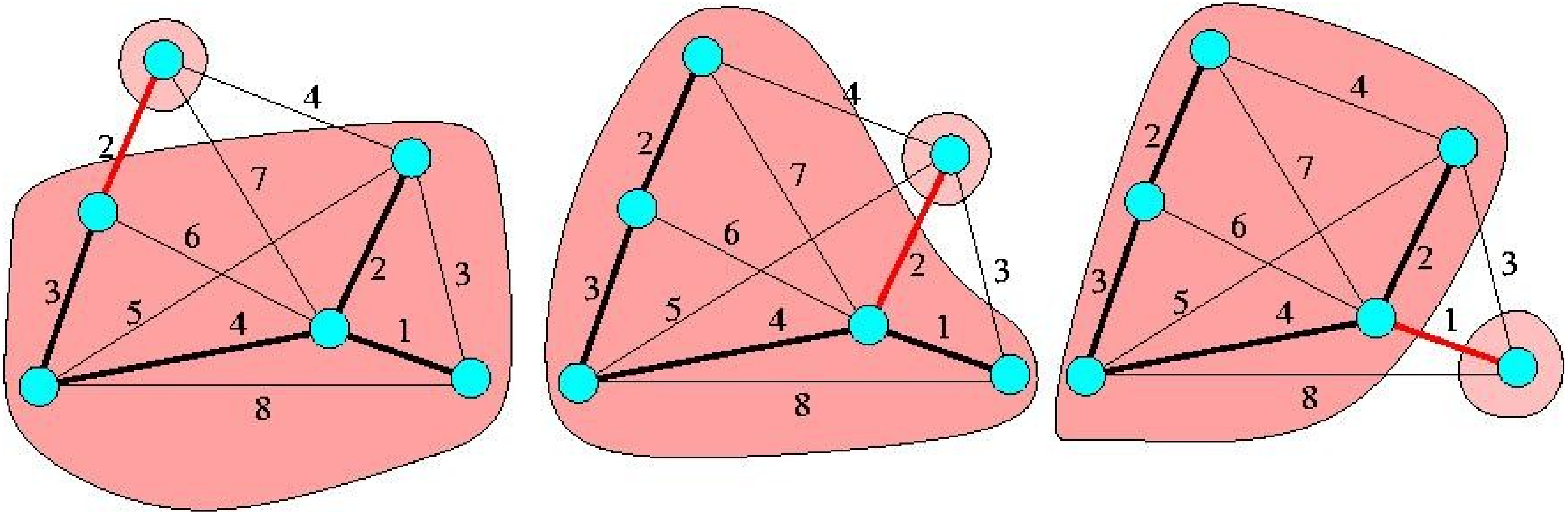
- Select n-1 edges.
- If it is a tree spanning G, compute its length and compare with the length of the best tree found so far. Save if the new tree is shorter.
- Repeat for another subset of n-1 edges.
- G can have up to $n(n-1)/2$ edges.
- Number of subset of size n-1 is exponential.

How to Get an Idea?

- Look at a solution for a small problem instance!

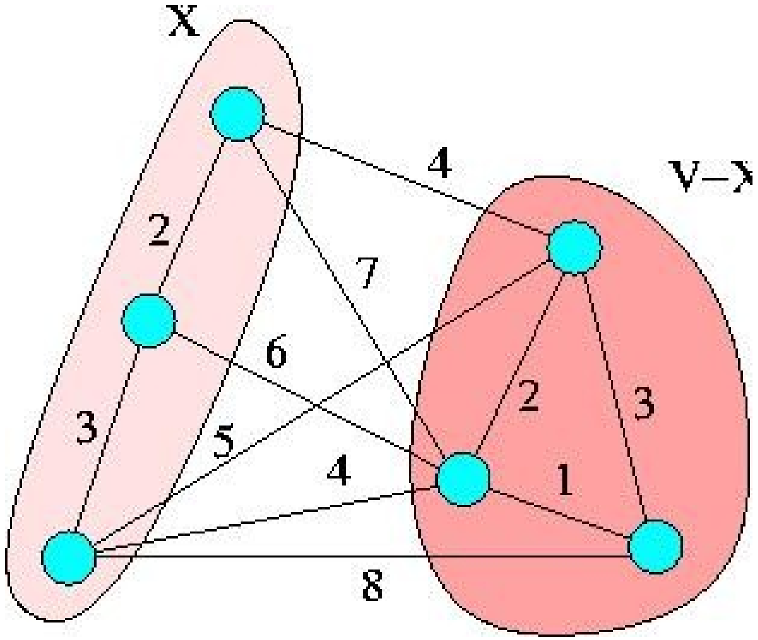


How to Get an Idea?



Cuts

- C={ X, V-X }, $\varnothing \subset X \subset V$, is a **cut** of G.
- An edge **crosses** a cut if it has one endvertex in X and one endvertex in V-X.
- An edge crossing cut C is a **light edge** if it has minimum weight among all edges crossing C.



Basic Property of Light Edges

- **Claim**: A light edge e of any cut $C = \{ X , V\text{-}X \}$ belongs to some minimum spanning tree of G=(V,E,c).
- Proof by contradiction: assume that e=(u,v) is a light edge of some cut C but it belongs to no MST.
- Let T be an MST. It contains a path from u to v. At least one edge, denoted by f, on this path crosses C.
- Remove f and add e to T. $T' = T - f + e$ is a tree spanning G. Since $|e| \leq |f|$, then $|T'| \leq |T|$.
- T is a MST, hence $|T'| = |T|$. T' is therefore also a MST. It contains e. This is a contradiction!!

Incorrect Algorithm

- Keep selecting cuts and their light edges until a minimum spanning tree is obtained.
- Why will this algorithm not work?
- How to avoid selecting the same light edge (belonging to two different cuts) more than once?
- Light edges of two cuts do not necessarily belong to the same minimum spanning tree!
- How to ensure that selected light edges belong to the same minimum spanning tree?

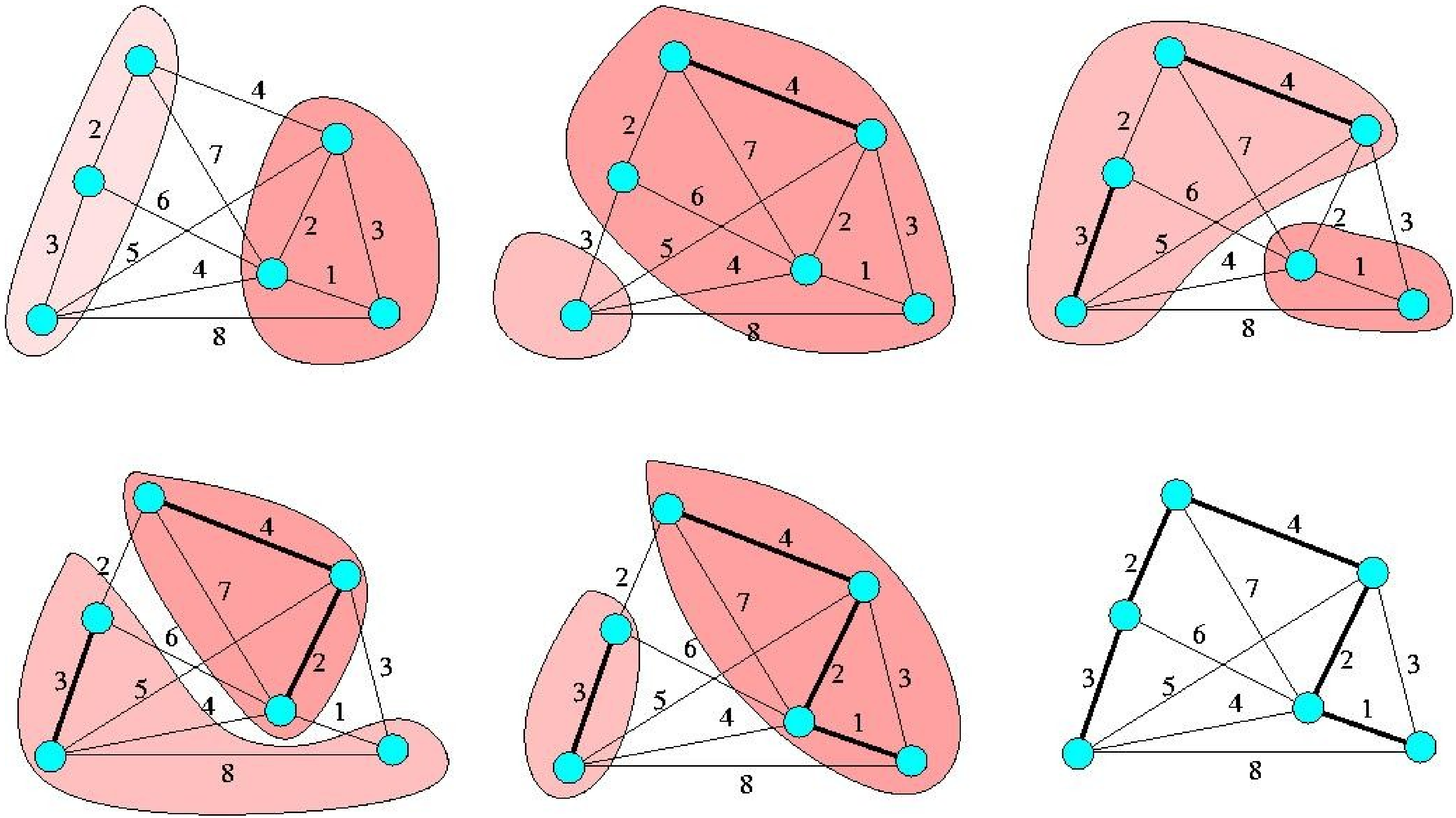
Cuts Respecting Set of Edges

- Let A be a subset of edges in $G=(V,E,c)$.
- A cut $C = \{ X, V\text{-}X \}$ **respects** A if none of the crossing edges of C belongs to A.

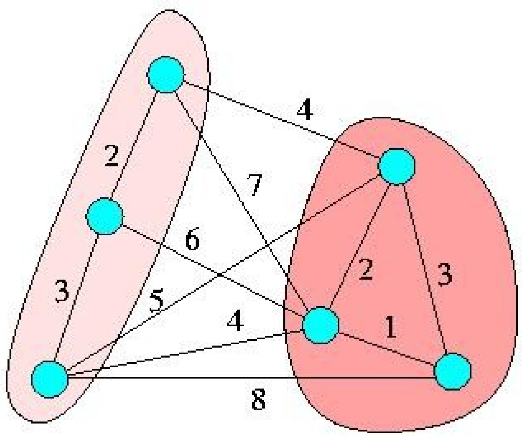
Correct Algorithm

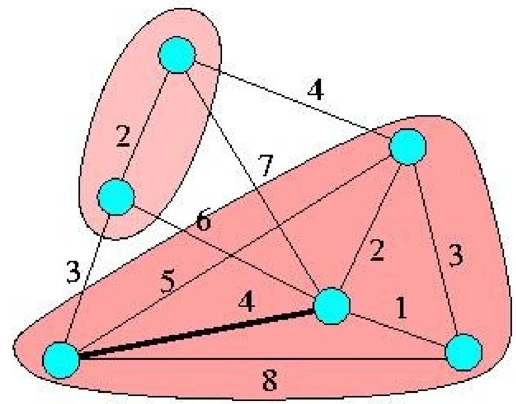
- Let A be the set of selected edges.
- Initially, $A = \emptyset$.
- Select a cut $C = \{ X , V\text{-}X \}$ respecting A.
- Select a light edge $e=(u,v)$ crossing C and add it to A.
- Repeat until there is no cut respecting A.

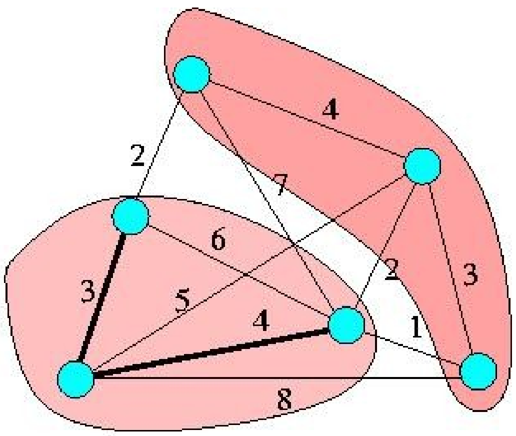
Example

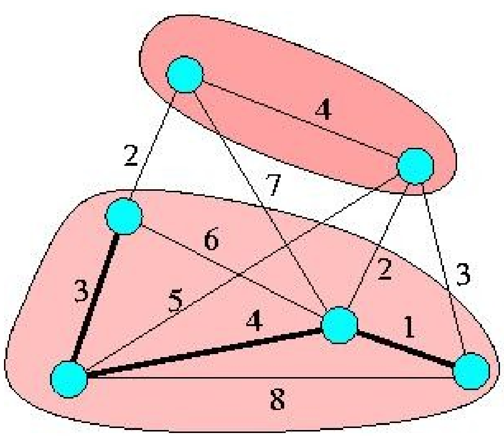


Another Example









What Do We Need to Prove?

- Let A be a subset of edges that belongs to some minimum spanning tree T.
- Let $C = \{ X , V\text{-}X \}$ be a cut respecting A.
- Let $e=(u,v)$ be a light edge crossing C.
- **Claim**: $A+e$ is a subset of edges that belongs to some minimum spanning tree T'.
- Proof by contradiction.

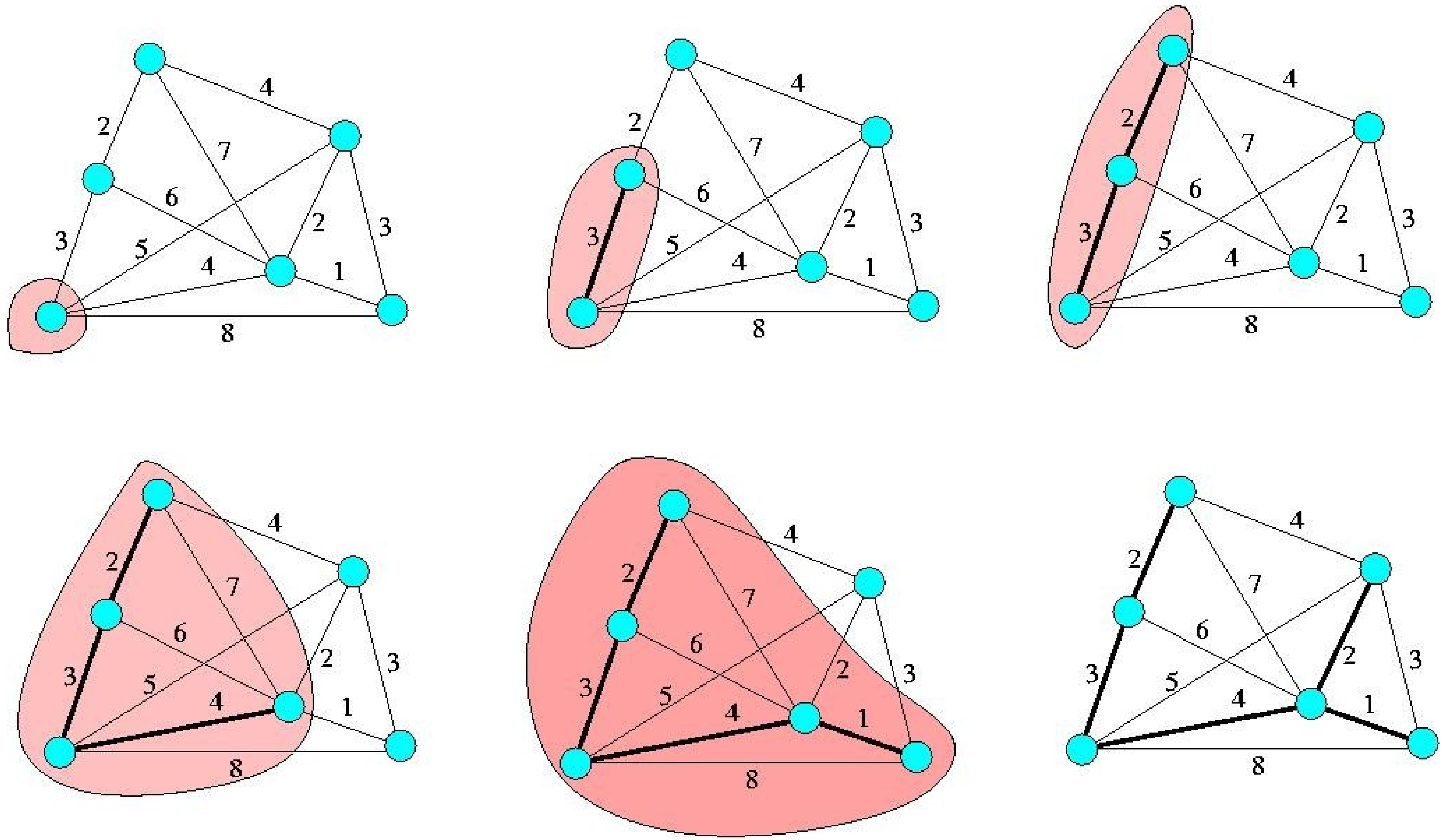
Proof by Contradiction

- Assume that no MST contains $A+e$.
- Consider T that contains A. It does not contain $e=(u,v)$.
- T has a path from u to v. This path must cross the cut C used to select e.
- Let f be the crossing edge on this path. Then $|e| \leq |f|$. Remove f and add e. Let $T' = T - f + e$.
- T' is spanning G and $|T'| \leq |T|$. Furthermore, T' contains $A+e$, a contradiction.

Prim's Algorithm

- Select a start vertex s in G.
- Let $T = (\{s\} , A)$ with $A = \emptyset$
- Keep adding to T vertices of G closest to T. Add the connecting edges to A.

Example



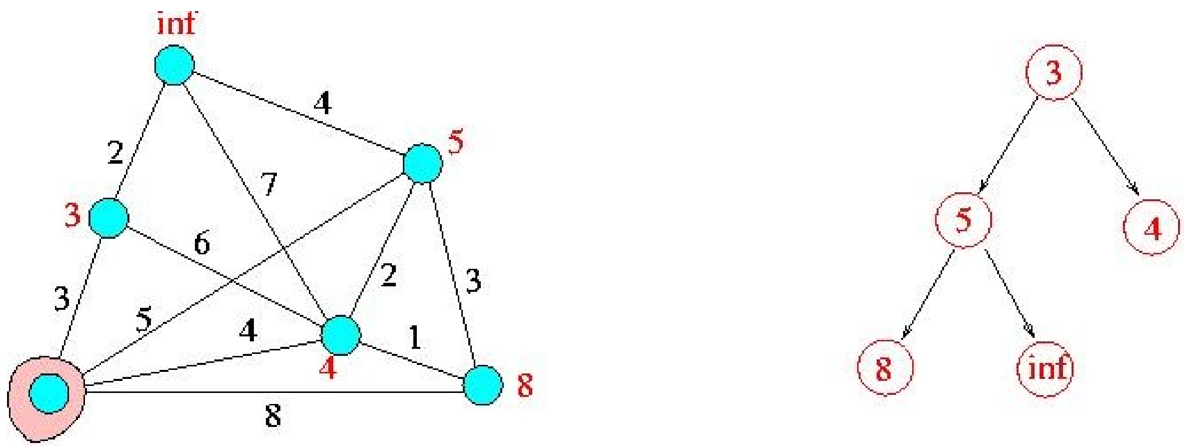
Prim's Algorithm is a Special Case of the General Algorithm

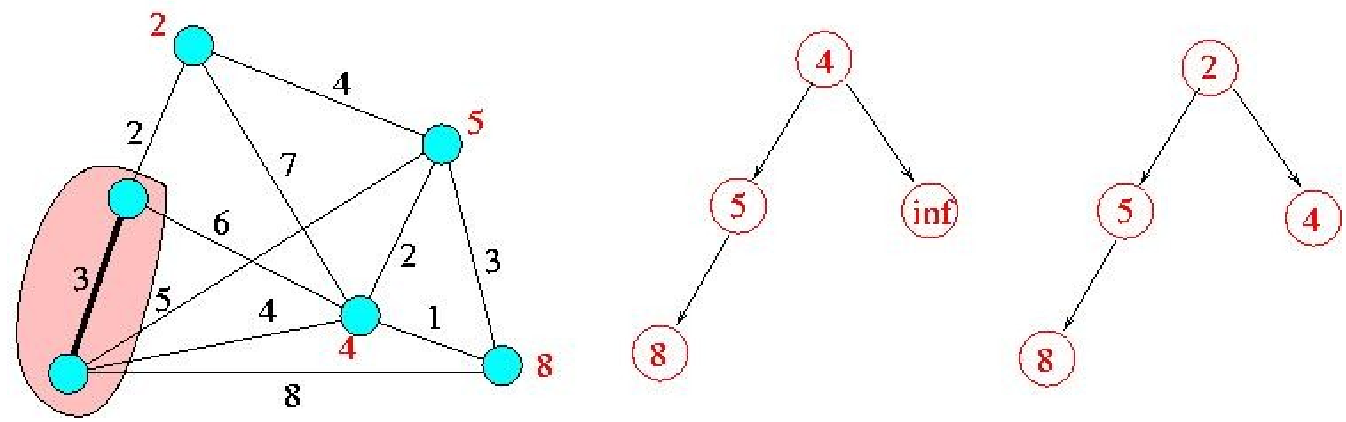
- During each iteration we select a cut separating vertices already in *T* from all other vertices.
- In particular, this cut respects the set of edges in *A*.
- The edge added to *A* is the shortest crossing edge.

Implementation of Prim's Algorithm

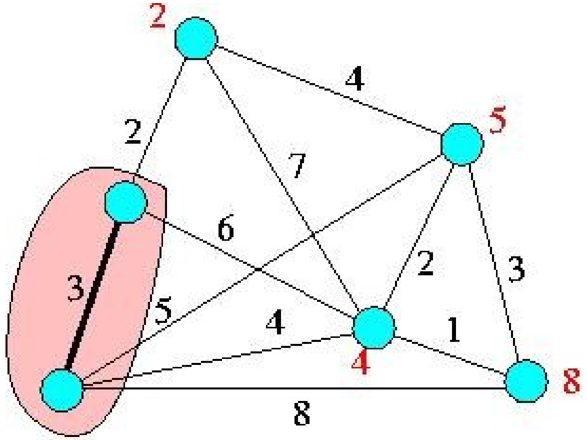
- Vertices not yet in *T* are on a min-heap.
- Each such vertex *v* has as its key the length of the of the shortest edge connecting *v* with *T*.
- The vertex *u* to be added at each iteration is at the root.
- When *u* and the connecting edge is added to *T,* some of the vertices not yet in *T* get closer to *T*. So the heap has to be updated.

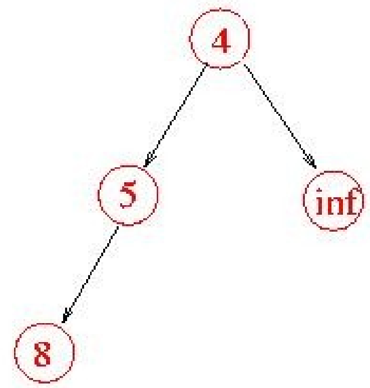
Using Heaps in Prim's Algorithm

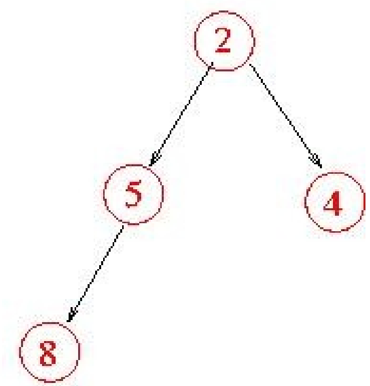


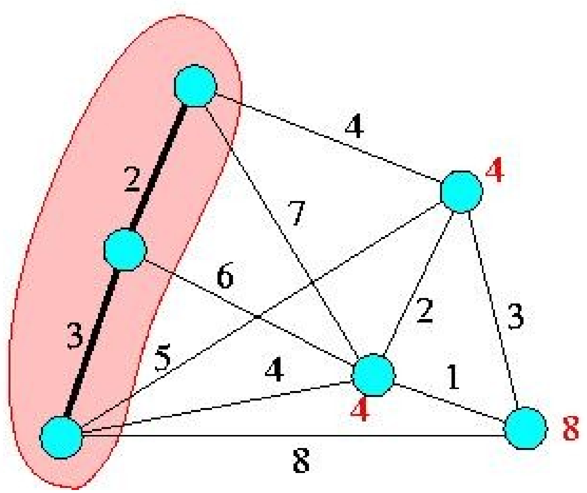


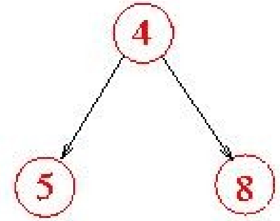
Using Heaps in Prim's Algorithm

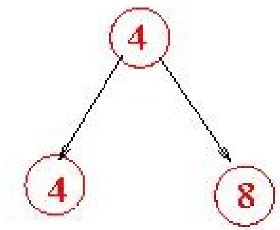




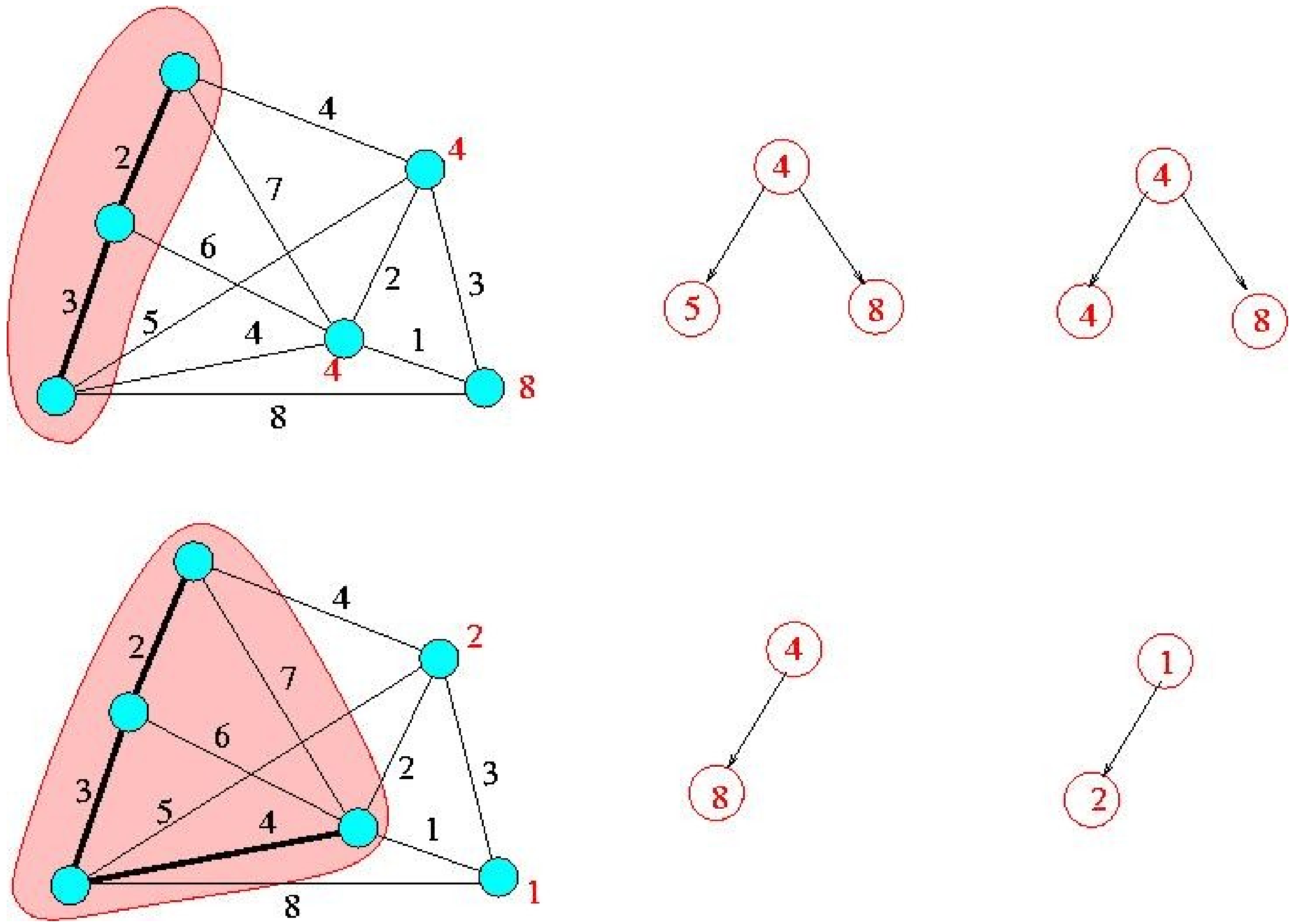








Using Heaps in Prim's Algorithm



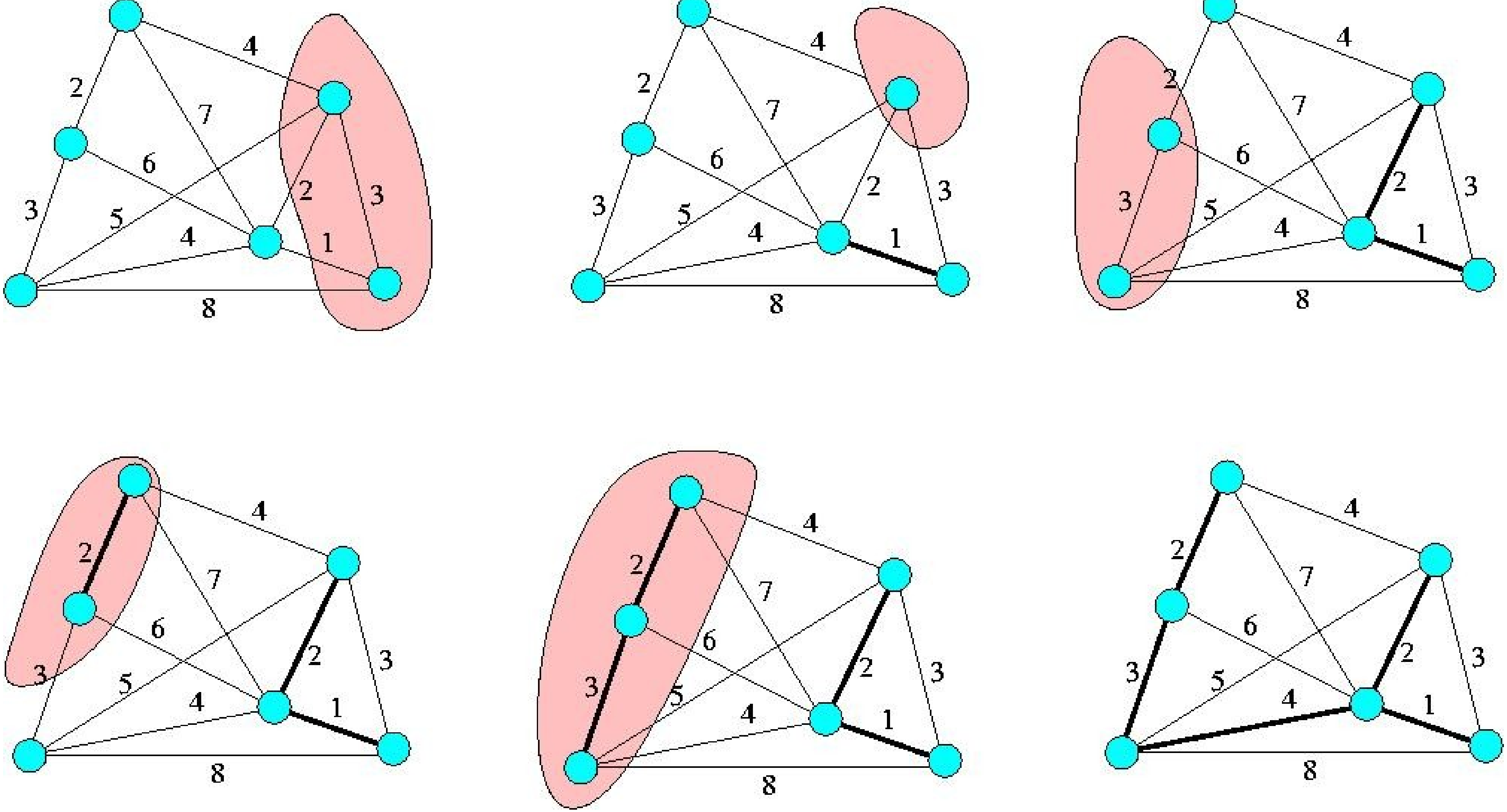
Complexity of Prim's Algorithm

- $G = (V,E,c)$ has n vertices and m edges.
- Min-heap can be constructed in $O(n)$ time.
- Each of the n-1 edges added to A are deleted from the min-heap in $O(\log n)$ time.
- When a vertex v is added to T, it has to be checked if the edges from v bring their end-vertices closer to T. If so, such end-vertex has to move up the min-heap in $O(\log n)$ time.
- Each edge is inspected once during the entire algorithm. In total $O(m \log n)$.

Kruskal's Algorithm

- Sort the edges of G in non-decreasing order.
- Start with a solution $T=(V,A)$ where $A = \emptyset$.
- Keep adding to A edges of G provided that they do not create a cycle in A.

Example



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

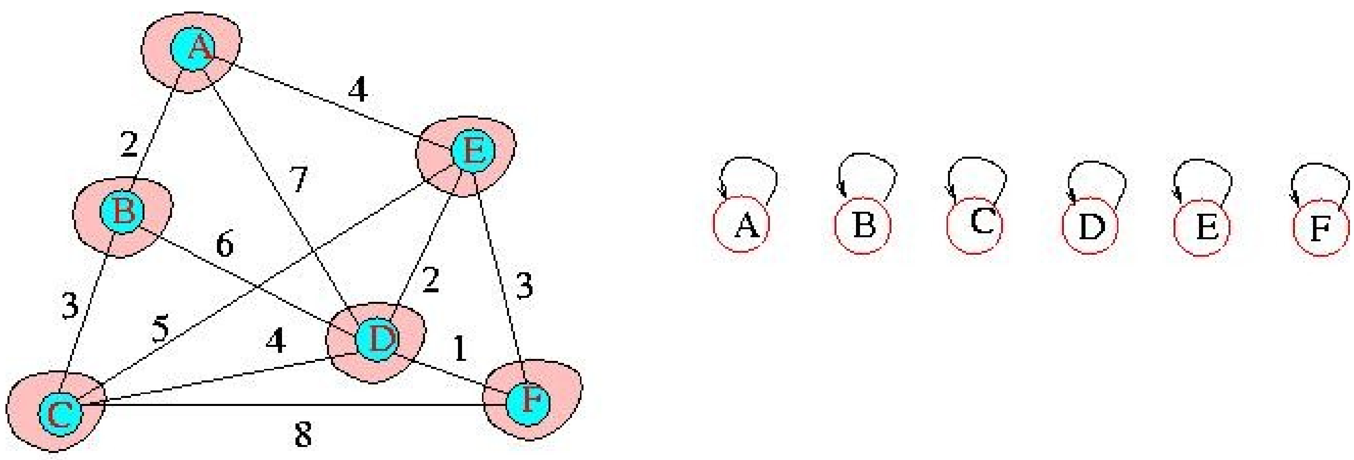
Kruskal's Algorithm is a Special Case of the General Algorithm

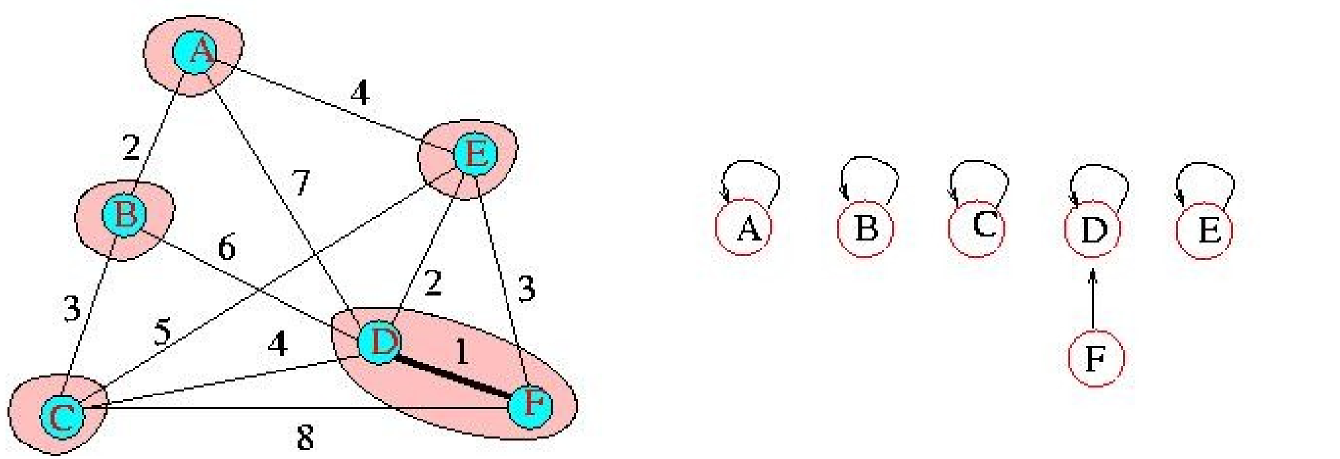
- Since we always select edges that do not close a cycle, these edges must connect vertice in two different components.
- A cut separating one of these components from the other vertices respects *A*.
- The edge selected crosses this cut and is the shortest among all edges crossing this cut.

Implementation of Kruskal's Algorithm

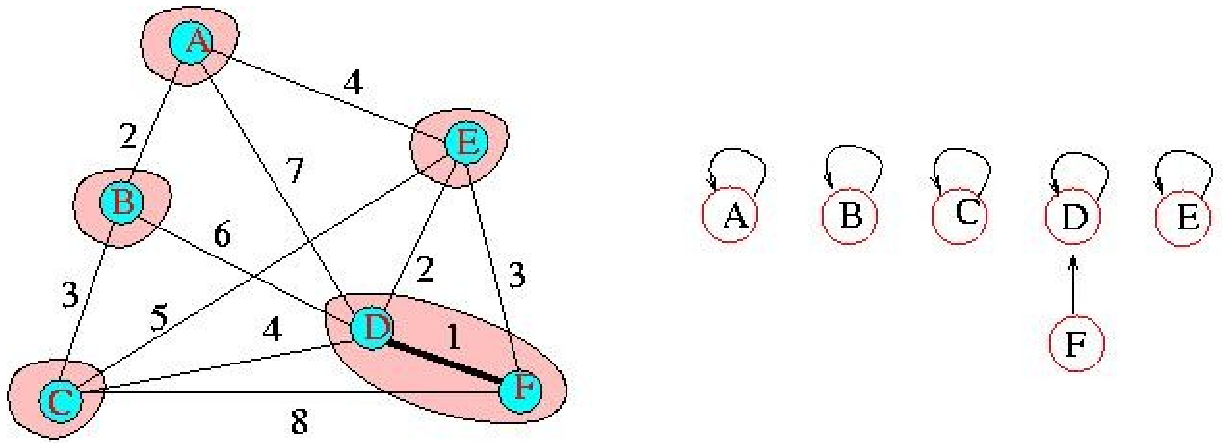
- We know how to sort.
- How do we check if an edge of G closes a cycle in *A*?
- Disjoint sets?

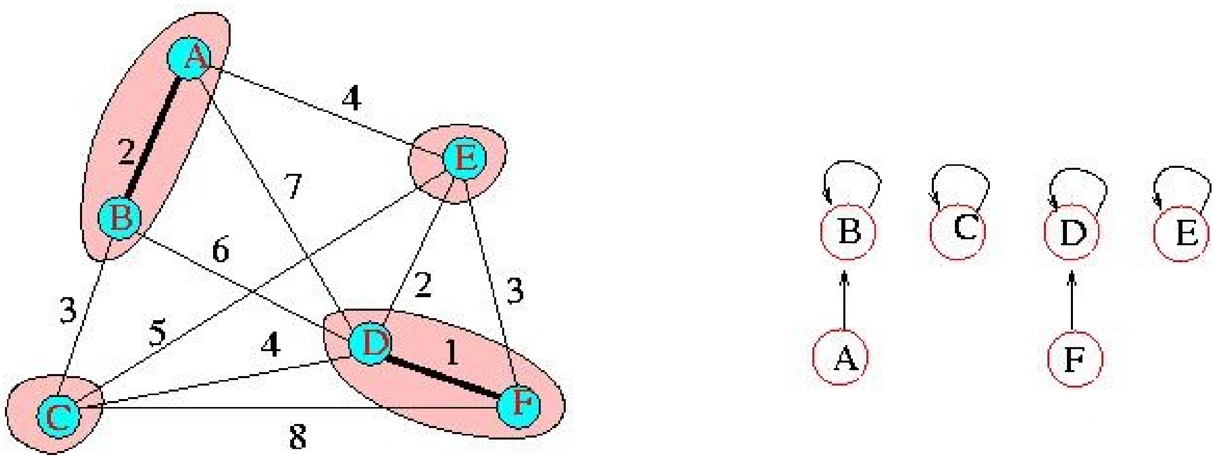
Using Disjoint Sets to Implement Kruskal's Algorithm



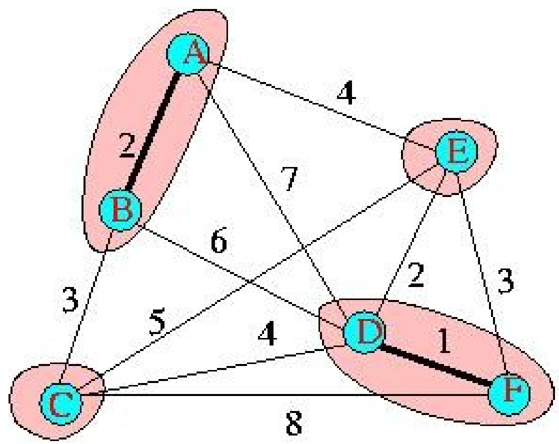


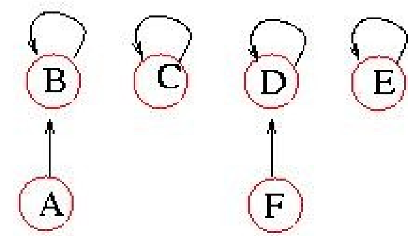
Using Disjoint Sets to Implement Kruskal's Algorithm

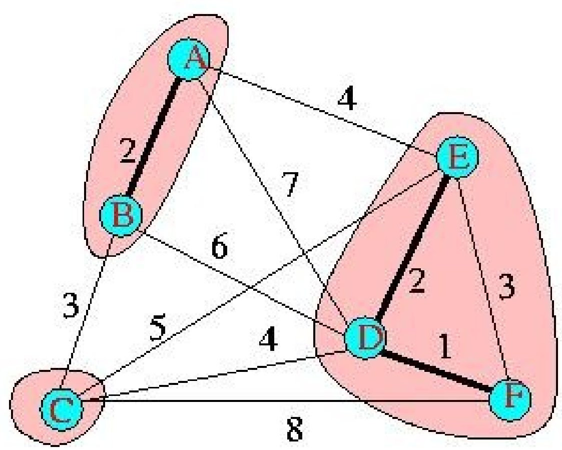


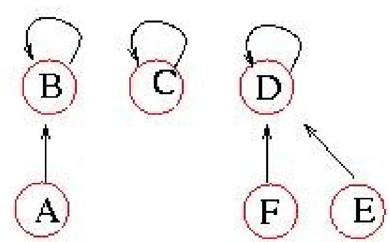


Using Disjoint Sets to Implement Kruskal's Algorithm









Complexity of Kruskal's Algorithm

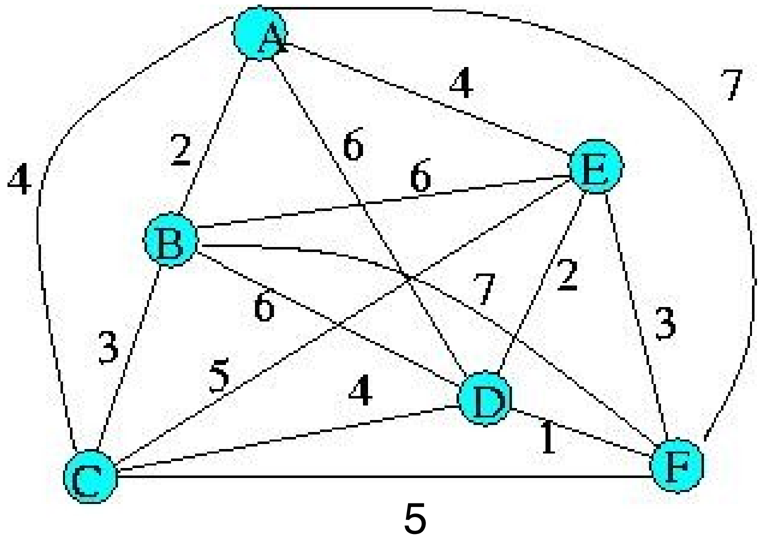
- Sorting of m edges: $O(m \log m)$
- Find-Union:
  - Creation of n sets with one vertex each: $O(n)$
  - n-1 union operations: $O(n)$
  - $2m$ find operations: $O(m \log^* n)$

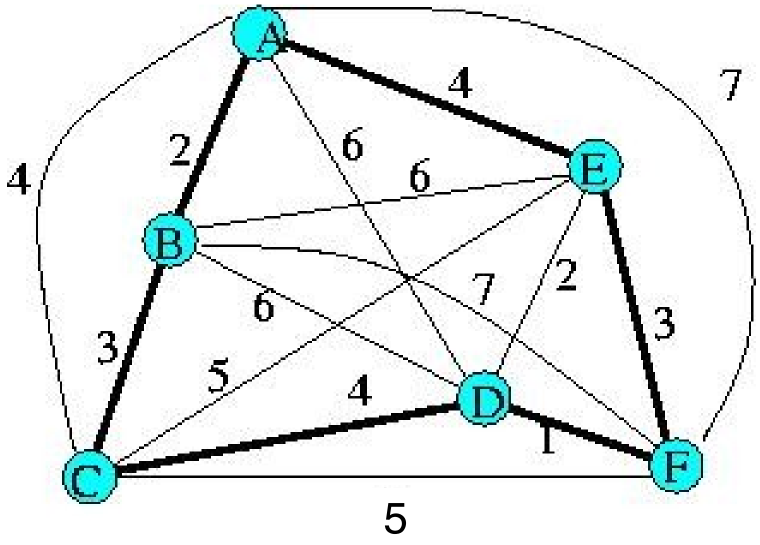
Travelling Salesman Problem

- Given *n* cities represented by an *n* x *n* matrix of intercity flight distances.
- A salesman has to visit all *n* cities and return to his home city. He wants to minimize his total travel length.

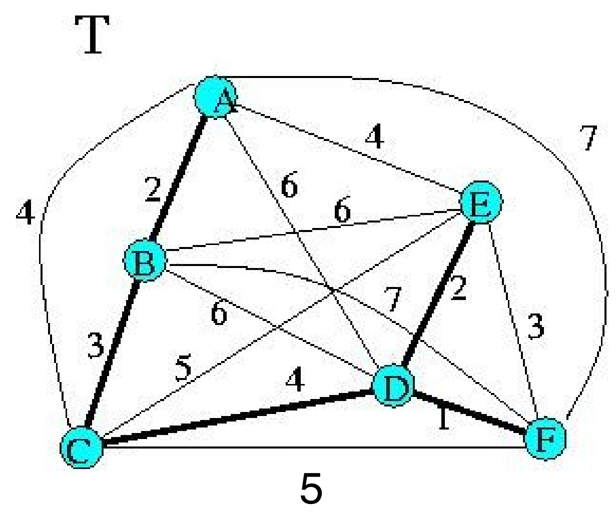
TSP - Example

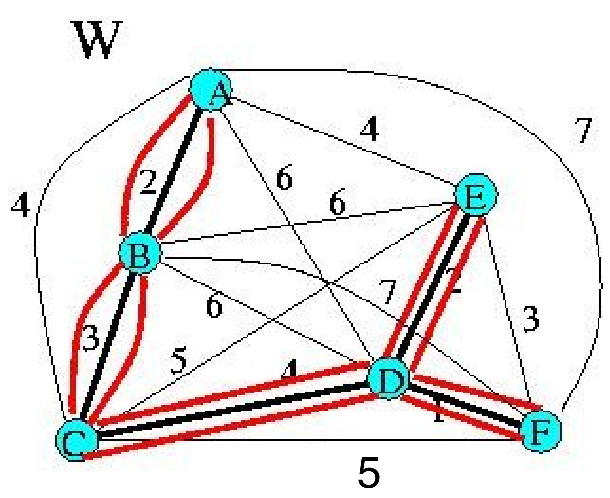
	A	B	C	D	E	F
A	0	2	4	6	4	7
B		0	3	6	6	7
C			0	4	5	5
D				0	2	1
E					0	3
F						0

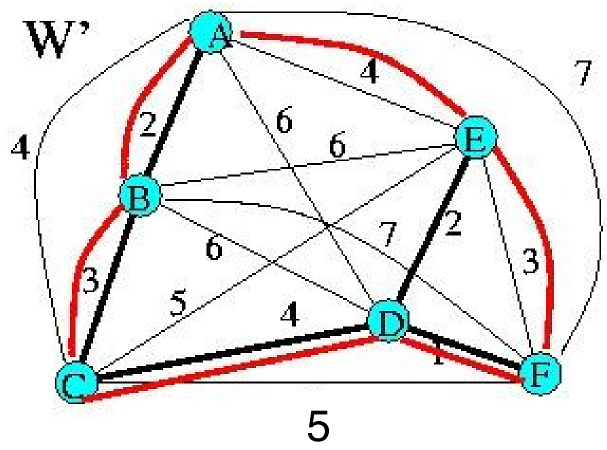




Suboptimal Algorithm







TSP – Suboptimal Algorithm

- Determine minimum spanning tree T.
- Walk around T (traversing each edge twice) getting W.
- $|W| = 2|T|$
- $|T^*| \geq |T^* - e| \geq |T|$
- $2|T^*| \geq 2|T^* - e| \geq 2|T| = |W|$

$$|W| / |T^*| \leq 2$$