

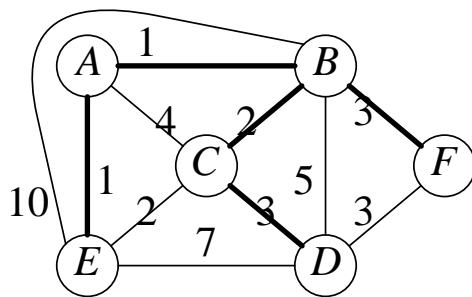
MINIMUM WEIGHT SPANNING TREE

Spanning Tree: Assume G is a weighted, connected graph.

- A tree $T = (V_T, E_T)$ contained in $G = (V, E)$, with $V_T = V$ and $E_T \subseteq E$.
- Weight $w(T) = \sum_{(x, y) \in T} w(x, y)$.

Min. Weight Spanning Tree (MST) T_0 :

- $w(T_0) = \min \{w(T) : T \text{ is a spanning tree of } G\}$.



An weighted graph G and an MST whose edges are shown in bold.

Questions:

- ? How many other MST 's are there in the above G ?

A Brute-Force Algorithm:

1. Find all spanning trees and their costs.
2. Choose one with the minimum weight.

Complexity: Can be exponential in $N = |V|$.

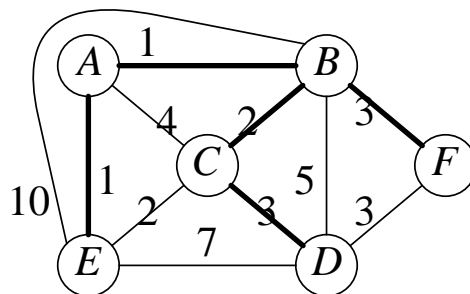
- The number of spanning trees in a complete graph with N nodes is N^{N-2} . (Finding them systematically without repetitions is non-trivial.)
- Computing $w(T)$ for each spanning tree T takes $O(N)$.

AN IMPORTANT PROPERTY OF MST

Structure of An *MST*.

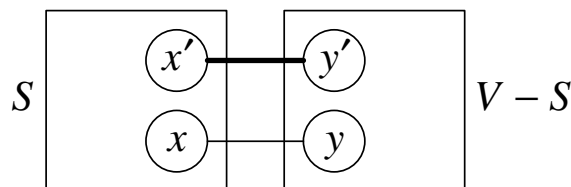
- Let (x, y) be a minimum-weight edge in $G = (V, E)$ which connects a node in $S \subset V$ and a node in $V - S$. Then, there is an *MST* which contains (x, y) .

Question: For $S = \{A, B, E\}$, what are some (x, y) below?



Can we get an *MST* containing (x, y) in each case?

Proof: Let T be an *MST* and $(x, y) \notin T$, and let C be the cycle in $T + (x, y)$ containing (x, y) . Clearly, C contains at least one other edge (x', y') connecting S and $V - S$.

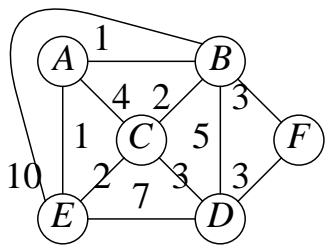


Since $w(x', y') \geq w(x, y)$ and $T' = T + (x, y) - (x', y')$ is a spanning tree and $w(T) \leq w(T') = w(T) + w(x, y) - w(x', y')$ it follows that $w(x', y') = w(x, y)$ and $w(T') = w(T)$ and T satisfies the theorem.

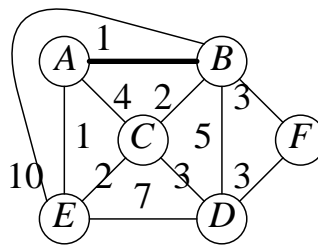
APPLICATION OF MST-STRUCTURE

A Method for Constructing an MST:

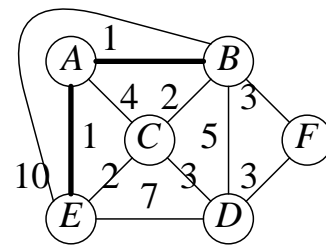
- Start with an arbitrary node x_0 and let $S = \{x_0\}$ and $T_E = \emptyset$.
- Successively add an edge to T based on the structure-theorem until T becomes a spanning tree.



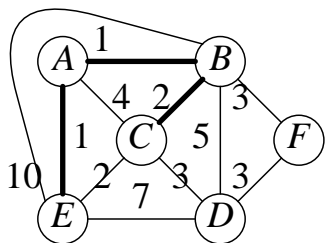
(i) $S = \{A\}$



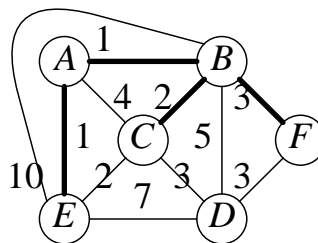
(ii) $S = \{A, B\}$;
edge added (A, B)



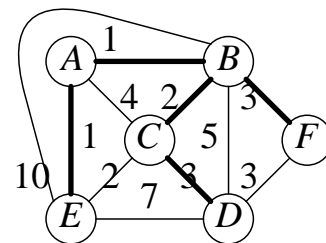
(iii) $S = \{A, B, E\}$;
edge added (A, E)



(iv) $S = \{A, B, E, C\}$;
edged added (B, C).



(v) $S = \{A, B, E, C, F\}$;
edged added (B, F).



(vi) Edged
added (C, D).

S (and edge added to T)	Edges between S and $V - S$ (new edges are underlined)	Reduced edge-set (one least weight edge at nodes in $V - S$)
$\{A\}$ (AB)	<u>AB</u> , AC, AE	<u>AB</u> , AC, AE (none at D, F)
$\{A, B\}$ (AE)	AC, AE, <u>BC</u> , <u>BD</u> , <u>BE</u> , <u>BF</u>	AE, <u>BC</u> , <u>BD</u> , <u>BF</u>
$\{A, B, E\}$ (BC)	AC, BC, <u>BD</u> , <u>BF</u> , <u>EC</u> , <u>ED</u>	<u>BC</u> , <u>BD</u> , <u>BF</u>
$\{A, B, E, C\}$ (BF)	<u>BD</u> , <u>BF</u> , <u>ED</u> , <u>CD</u>	<u>CD</u> , <u>BF</u>
$\{A, B, E, C, F\}$ (CD)	<u>BD</u> , <u>ED</u> , <u>CD</u> , <u>FD</u>	<u>BF</u>

Question:

- ? For $|S| = k$, how large can be the set of edges between S and $V - S$?

PRIM'S ALGORITHM

Notations:

- $\text{bestWeight}(y) = \min \{w(x, y): x \in S\}$, for $y \notin S$.
- $\text{best}(y) = x \in S$, where $w(x, y) = \text{bestWeight}(y)$.

Algorithm PRIM:

1. [Initialization.]

- (a) Choose a node x , let $S = \{x\}$ = the set of current nodes in T , and $E_T = \emptyset$, the set of edges in T .
- (b) For each $y \notin S$, let $\text{best}(y) = x$ and $\text{bestWeight}(y) = w(x, y)$ if $y \in \text{adjList}(x)$ and otherwise let $\text{best}(y) = y$ and $\text{bestWeight}(y) = \infty$ (a large number).

2. [Add a node and edge to T .] Repeat (a)-(b) $N - 1$ times:

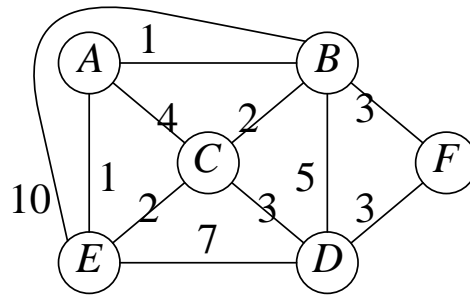
- (a) Choose $x \notin S$ such that $\text{bestWeight}(x) = \min\{\text{bestWeight}(y): y \notin S\}$; add x to S and $(\text{best}(x), x)$ to E_T .
- (b) For each $y \notin S$, do the following:
if $w(x, y) < \text{bestWeight}(y)$, then let $\text{best}(y) = x$ and $\text{bestWeight}(y) = w(x, y)$.

Note: For efficient execution of step 2(c), which processes each edge of G once, represent G by an adjacency matrix, with the entries $w(x, y)$.

Complexity: $O(N^2)$.

- All iterations of step 2(a): $O((N - 1) + \dots + 2 + 1) = O(N^2)$.
- All iterations of step 2(b): $O((N - 2) + \dots + 2 + 1) = O(N^2)$.

ILLUSTRATION OF PRIM'S ALGORITHM



("-" means not relevant.)

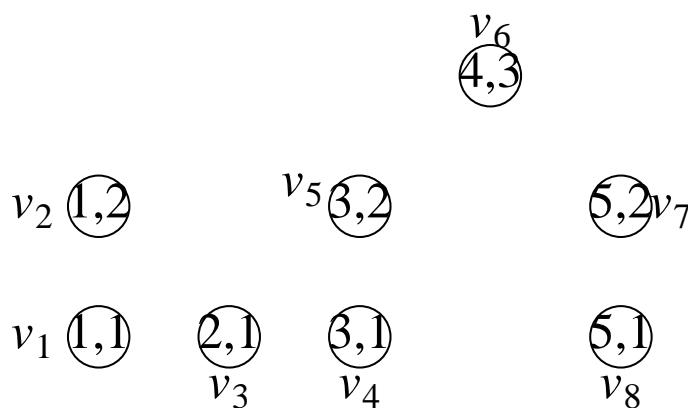
Node x	best(y) and bestWeight(y)						Edge added to T
	A	B	C	D	E	F	
A	-	A, 1	A, 4	D, ∞	A, 1	F, ∞	
B	-	-	B, 2	B, 5	A, 1	B, 3	(A, B)
E	-	-	B, 2	B, 5	-	B, 3	(A, E)
C	-	-	-	C, 3	-	B, 3	(B, C)
D	-	-	-	-	-	B, 3	(C, D)
F	-	-	-	-	-	-	(B, F)

Question:

- ? Give the complexity of PRIM's algorithm if we use a brute force implementation of step (2) as indicated in the second column of the table on page 6.3, without using best(y) and bestWeight(y).
- ? Let T_1 and T_2 be two *MST*'s for a graph G and $n(T_i, w) = \#(\text{edges in } T_i \text{ with weight } w)$. Prove that $n(T_1, w) = n(T_2, w)$ for all w . (You can verify the equality for G on page 6.1.)

EXERCISE

- Let $T = (V, E)$ be a tree with its edge-weights $w(v_i, v_j) > 0$. For each node-pair v_i and v_j , let $c(v_i, v_j) = |\pi(v_i, v_j)|$, where $\pi(v_i, v_j)$ is the unique $v_i v_j$ -path in T . Consider the complete digraph G on the vertices V with $c(v_i, v_j) = c(v_j, v_i)$. How can you identify the edges of T within G ?
- Consider a set of points $V = \{v_i(x_i, y_i), 1 \leq i \leq N\}$ in the plane, where (x_i, y_i) are the coordinates of the point v_i . Shown below is an example for $N = 8$. For each $r > 0$, define the set of edges $E_r = \{(v_i, v_j): d(v_i, v_j) \leq r\}$ among the nodes v_i . We want to find the smallest $r = r_c$ such that the edges E_r form a connected graph on the nodes V .



- How are the sets E_r and $E_{r'}$ related (for an arbitrary V) for $r < r'$? Explain your answer for the above nodes.
- Find $r = r_c$ for the example nodes V above; show E_r on the points V . Find the largest $r' < r_c$ such that $|E_{r'}| < |E_r|$ and show $E_{r'}$.
- Give an algorithm for finding r_c for an arbitrary V . Explain the algorithm using the example set of nodes.

USE OF INPUT/OUTPUT STRUCTURES IN PRIM'S ALGORITHM

Use Of Input Structure:

- A change in the weights of edges may change the reduced edge-set considered in each iteration of Step 2 and the successive edges (x, y) selected in Step 2(a), but it still looks at every edge once in updating $best(z)$ and $c(z)$ for $z \in V - V_T$ in the course of the algorithm and no short-cuts is made.

Thus, it *does not* make use of any input-structure. A change in the edges of the graph can be thought of as a change in the weights (why?).

Use Of Output Structure:

- Cycles are avoided by adding each time an edge connecting a node in V_T to a node not in V_T .

Thus, the output-structure is *used*; no attempt is made to determine which part of the final tree is to be created first.

Use Of Input-Output Relationship:

- The essential part of the algorithm (choosing a smallest cost edge between V_T and $V - V_T$) is based on the input-output relationship $T \subseteq G$.

Thus (as almost in all algorithms), the input-output relationship is used here.

Greedy nature of Prim's algorithm:

- At each iteration, a *locally* optimal ("greedy") choice is made (involving a search) from the edges between V_T and $V - V_T$.

- Once a choice is made, it is never revised and is a part of the final solution (i.e., no backtracking).

Question:

- (1) In what way, Dijkstra's algorithm for shortest x - y path is not "greedy"? (What local decisions are made at any stage and which of them may be revised subsequently?)

In what sense, Dijkstra's algorithm for shortest-path from x to all nodes (reachable from x) is greedy?

- (2) In what way, Floyd's algorithm is not "greedy"?

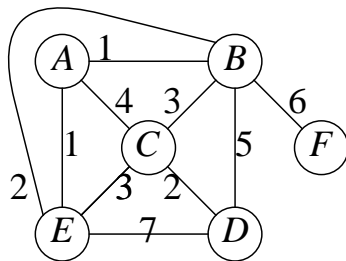
KRUSKAL'S ALGORITHM FOR MINIMUM-WEIGHT SPANNING TREE

- Idea:**
- (1) Successively add edges of *smallest* weights, without creating a cycle.
 - (2) Each intermediate stage consist of a *forest* on the nodes of G . (This is a slight disadvantage compared to Prim's algorithm.)

Advantage Over Prim's algorithm:

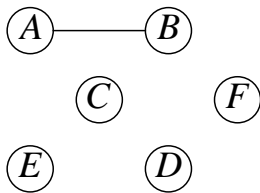
Gives a more efficient implementation, using the *disjoint set-union* algorithm, if G has $O(N^\alpha)$, $\alpha < 2$, edges.

Example.

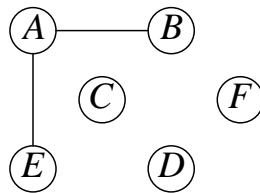


Order edges in non-decreasing weights:

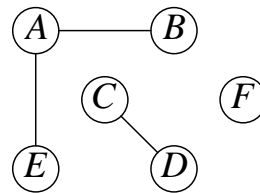
$(A, B) = 1$, $(A, E) = 1$, $(B, E) = 2$, $(C, D) = 2$, $(B, C) = 3$,
 $(C, E) = 3$, $(A, C) = 4$, $(B, D) = 5$, $(B, F) = 6$, $(D, E) = 7$.



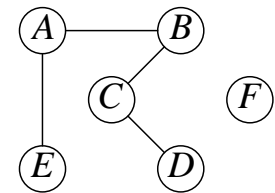
(A, B) added.



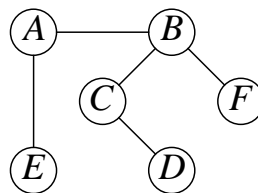
(A, E) added.



(B, E) not added;
 (C, D) added.



(B, C) added.



(C, E) , (A, C) , (B, D)
 not added; (B, F) added.

KRUSKAL'S ALGORITHM

Algorithm KRUSKAL:

Input: A weighted connected graph $G = (V, E)$.

Output: The edges E_T of a min-weight spanning tree of G .

1. [Initialize.] Sort the edges in the non-decreasing order of weights. Initialize a set $\{x\}$ for each node x in G , and initialize $E_T = \emptyset$.
2. For (each successive edge (x, y)) do the following until $|E_T| = |V| - 1$:
If (x and y belong to different sets), then add (x, y) to E_T and merge those two sets into a single set (throwing away the old ones).

EXERCISE

1. If we use an ordinary list representation of the various sets and each of the first $|V| - 1$ edges are added to E_T , then what is the complexity of processing the i th edge?
2. Give an example graph to illustrate the following extreme scenario: we add the edges e_1, e_2 , and do not add e_3 , then we add e_4 and not add the edges $\{e_5, e_6, e_7\}$, then add e_8 and not add $\{e_9, e_{10}, e_{11}, e_{12}\}$, etc.
3. Compare the use of input/output structure in Kruskal's algorithm with those in Prim's algorithm.

EFFICIENT IMPLEMENTATION OF KRUSKAL'S ALGORITHM

Set-operations used in Kruskal's algorithm:

- (1) *Find* the set containing a given element z ($z = x$ and $z = y$ for an edge (x, y)).
- (2) *Merge* the sets containing x and y if they are different (and hence disjoint).

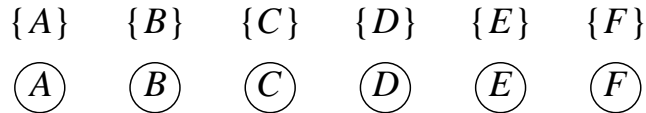
Data-structure: Each set is represented by a tree (a node may have more than 2 children).

- (1) If $r(x) =$ root of the tree containing x , then x and y are in the same set if and only if $r(x) = r(y)$.
- (2) To merge the sets containing x and y , make the tree with the root $r(x)$ a child of $r(y)$ if the tree with root $r(x)$ has smaller size than the tree with root $r(y)$. (This keeps the height of the trees small and hence makes the determination of $r(x)$ efficient.)

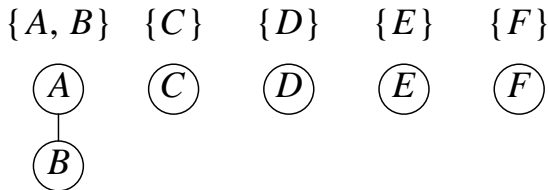
Complexity: $O(|E|\log N) < (N^2)$ if $|E| = O(N^\alpha)$ and $\alpha < 2$.

- (1) Sorting edges take $O(|E|\log |E|) = O(|E|\log N)$ time.
- (2) Height of a tree with k nodes can be shown to be $O(\log k)$.
- (3) For each edge (x, y) , finding $r(x)$ and $r(y)$ takes $O(\log N)$ time by following parent-links. Testing " $r(x) = r(y)$ " and then merging the trees, if necessary, take $O(1)$ time.
- (4) We update the size of the tree when two trees are merged.

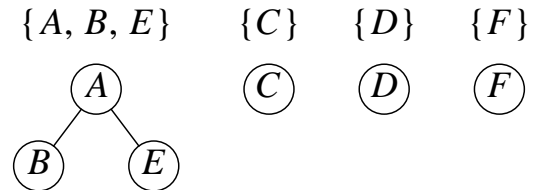
Example: For the graph shown earlier:



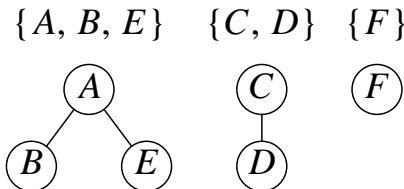
(i) Initial sets having one node each and their associated trees.



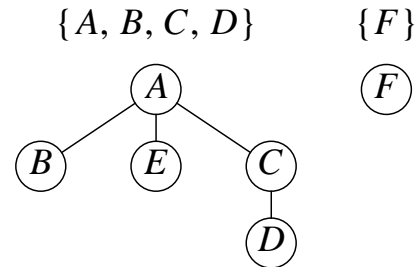
(ii) After processing edge (A, B) .



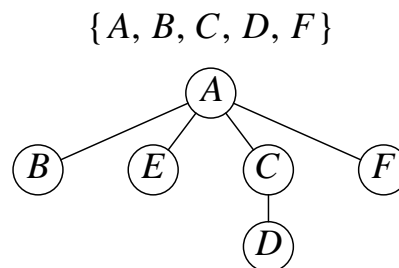
(iii) After processing edge (A, E) .



(iv) After processing edge (C, D) .



(v) After processing edge (B, C) .



(vi) After processing edge (B, F) .

EXERCISE

- Show all possible rooted trees with $N = 3, 4, 5,$ and 6 nodes. (Keep the subtrees of a node with larger size on the left. This means there are only 3 such trees for $N = 4$.) For each tree, show all possible sequence of merge-operations, if any, that give rise to that tree. (Label the edges as $1, 2, \dots, N - 1$ to indicate the order in which the subtrees were merged.)