Greedy Algorithms

Prim and Kruskal

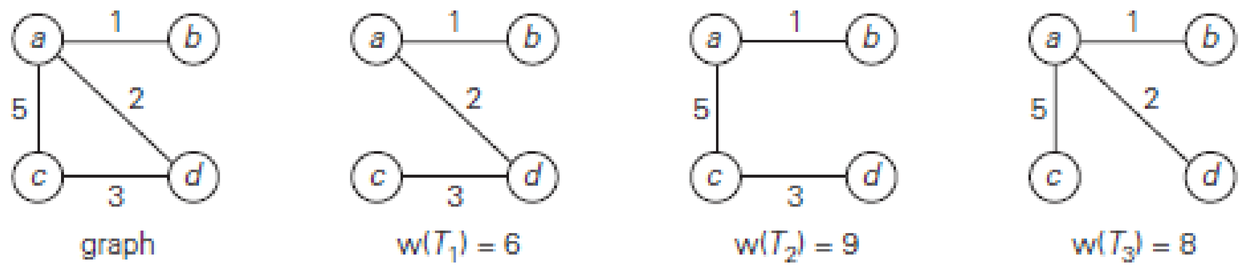
Problemset

- given n points, connect them in the cheapest possible way so that there will be a path between every pair of points
- represent the points given by vertices of a graph, possible connections by the graph's edges, and the connection costs by the edge weights
- minimum spanning tree problem

Minimum spanning tree

- A spanning tree of an undirected connected graph is its connected acyclic subgraph (i.e., a tree) that contains all the vertices of the graph. If such a graph has weights assigned to its edges, a minimum spanning tree is its spanning tree of the smallest weight, where the weight of a tree is defined as the sum of the weights on all its edges. The minimum spanning tree problem is the problem of finding a minimum spanning tree for a given weighted connected graph.

Building a spanning tree



- If we were to try constructing a minimum spanning tree by exhaustive search, we would face two serious obstacles.
- First, the number of spanning trees grows exponentially with the graph size (at least for dense graphs).
- Second, generating all spanning trees for a given graph is not easy; in fact, it is more difficult than finding a *minimum* spanning tree for a weighted graph by using one of several efficient algorithms available for this problem.

Prims Algorithm

- through a sequence of expanding subtrees.
- initial subtree: sequence consists of a single vertex selected arbitrarily from the set V of the graph's vertices
- On each iteration: expands the current tree in the greedy manner by simply attaching to it the nearest vertex not in that tree.
- Stop: after all the graph's vertices have been included in the tree being constructed
- Since the algorithm expands a tree by exactly one vertex on each of its iterations, the total number of such iterations is $n - 1$, where n is the number of vertices in the graph.
- The tree generated by the algorithm is obtained as the set of edges used for the tree expansions.

Prims Pseudocode

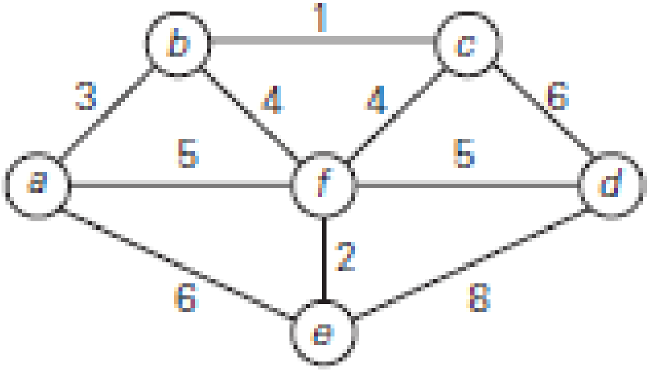
ALGORITHM *Prim(G)*

//Prim's algorithm for constructing a minimum spanning tree
//Input: A weighted connected graph $G = \langle V, E \rangle$
//Output: E_T, the set of edges composing a minimum spanning tree of G
$V_T \leftarrow \{v_0\}$ //the set of tree vertices can be initialized with any vertex
$E_T \leftarrow \varnothing$
for $i \leftarrow 1$ **to** $|V| - 1$ **do**
    find a minimum-weight edge $e^* = (v^*, u^*)$ among all the edges (v, u)
    such that v is in V_T and u is in $V - V_T$
    $V_T \leftarrow V_T \cup \{u^*\}$
    $E_T \leftarrow E_T \cup \{e^*\}$
return E_T

- provide each vertex not in the current tree with the information about the shortest edge connecting the vertex to a tree vertex
- We can provide such information by attaching two labels to a vertex: the name of the nearest tree vertex and the length (the weight) of the corresponding edge
- With such labels, finding the next vertex to be added to the current tree $T = \{V\_T, E\_T\}$ becomes a simple task of finding a vertex with the smallest distance label in the set $V - V\_T$

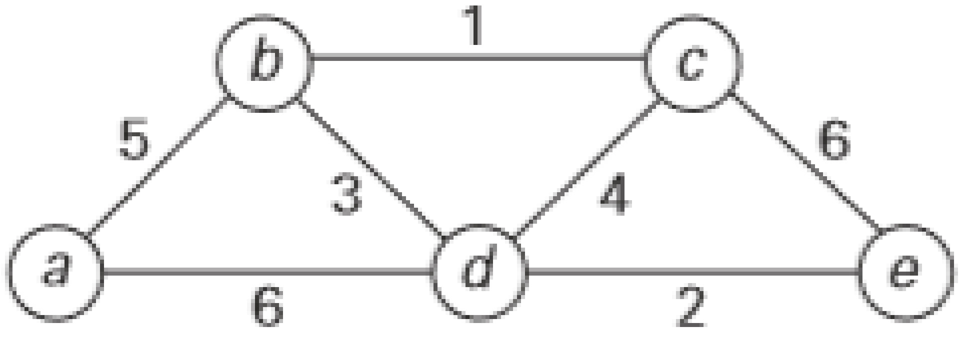
After we have identified a vertex u∗ to be added to the tree, we need to perform two operations:

- Move u∗ from the set V − V_T to the set of tree vertices V_T .
- For each remaining vertex u in V − VT that is connected to u∗ by a shorter edge than the u's current distance label, update its labels by u and the weight of the edge between u∗ and u, respectively



Example

Your turn



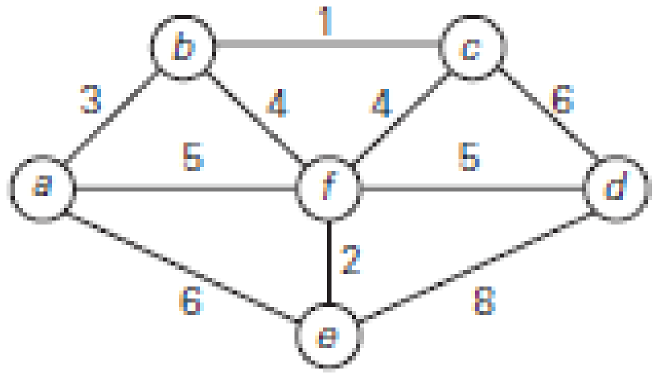
Kruskals Algorithm

- Kruskal's algorithm looks at a minimum spanning tree of a weighted connected graph G= V,Eas an acyclic subgraph with $|V|-1$ edges for which the sum of the edge weights is the smallest
- Consequently, the algorithm constructs a minimum spanning tree as an expanding sequence of subgraphs that are always acyclic but are not necessarily connected on the intermediate stages of the algorithm.
- The algorithm begins by sorting the graph's edges in nondecreasing order of their weights.
- starting with the empty subgraph: it scans this sorted list, adding the next edge on the list to the current subgraph if such an inclusion does not create a cycle and simply skipping the edge otherwise.

Kruskal Pseudocode

ALGORITHM *Kruskal(G)*

//Kruskal's algorithm for constructing a minimum spanning tree
//Input: A weighted connected graph $G = \langle V, E \rangle$
//Output: E_T, the set of edges composing a minimum spanning tree of G
sort E in nondecreasing order of the edge weights $w(e_{i_1}) \le \cdots \le w(e_{i_{|E|}})$
$E_T \leftarrow \varnothing$; $\quad ecounter \leftarrow 0$ //initialize the set of tree edges and its size
$k \leftarrow 0$ //initialize the number of processed edges
while $ecounter < |V| - 1$ **do**
$\quad k \leftarrow k + 1$
$\quad$**if** $E_T \cup \{e_{i_k}\}$ is acyclic
$\quad\quad E_T \leftarrow E_T \cup \{e_{i_k}\}$; $\quad ecounter \leftarrow ecounter + 1$
return E_T



Example

Disjoint Subsets and Union-Find Algorithms

- Dealing with an abstract datatype of a collection of disjoint subsets of a finite set with the following operations:
  - makeset(x): creates a one- element set{x}. It is assumed that this operation can be applied to each of the elements of set S only once.
  - find(x) returns a subset containing x.
  - union(x, y) constructs the union of the disjoint subsets Sx and Sy containing x and y, respectively, and adds it to the collection to replace Sx and Sy, which are deleted from it.

Example

- S ={1, 2, 3, 4, 5, 6}.
- Makeset(i)
- union(1, 4) and union(5, 2)
- union(4,5) and then by union (3,6)

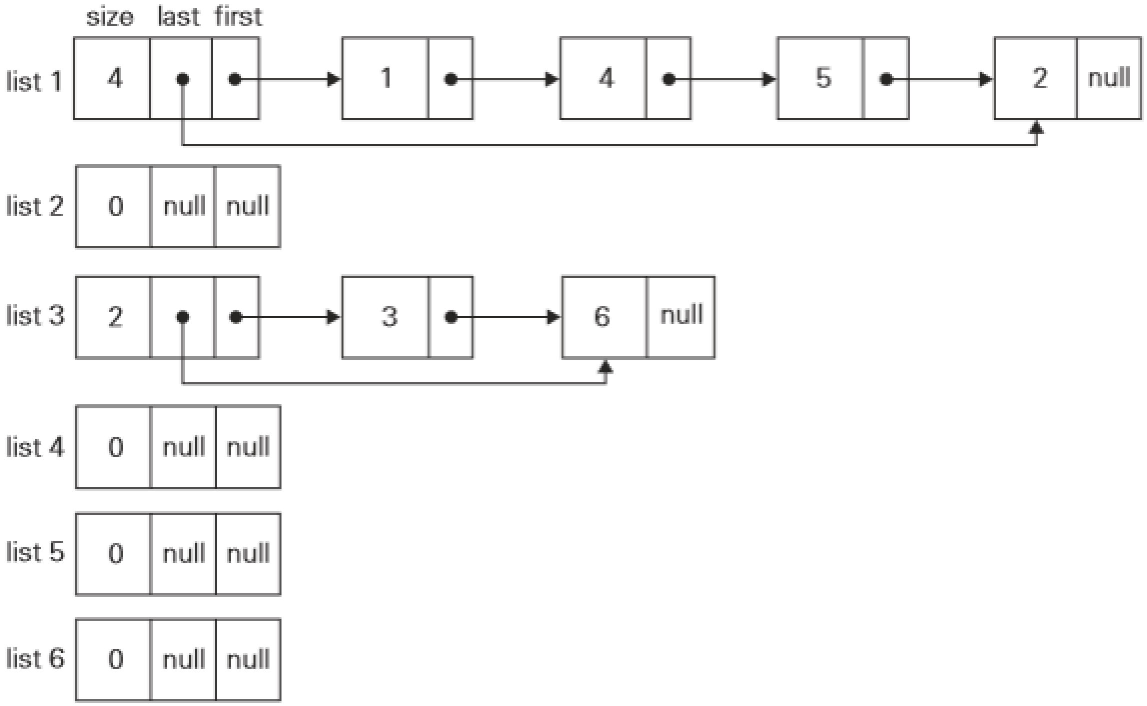
Representative

- Most implementations of this abstract data type use one element from each of the disjoint subsets in a collection as that subset's representative
- Some implementations do not impose any specific constraints on such a representative; others do so by requiring, say, the smallest element of each subset to be used as the subset's representative.
- Also, it is usually assumed that set elements are (or can be mapped into) integers.

Quick Find

- The quick find uses an array indexed by the elements of the underlying set S; the array's values indicate the representatives of the subsets containing those elements. Each subset is implemented as a linked list whose header contains the pointers to the first and last elements of the list along with the number of elements in the list

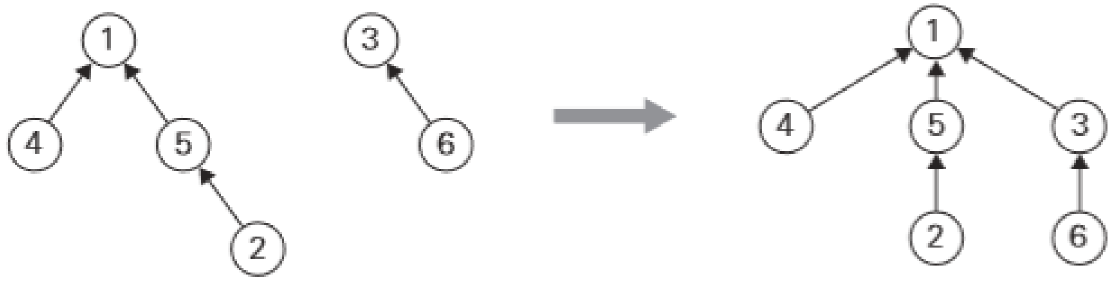
subset representatives	
element index	representative
1	1
2	1
3	3
4	1
5	1
6	3



Quick Union

- represents each subset by a rooted tree. The nodes of the tree contain the subset's elements (one per node), with the root's element considered the subset's representative;
- The tree's edges are directed from children to their parents.
- In addition, a mapping of the set elements to their tree nodes— implemented, say, as an array of pointers—is maintained.

Quick Union



(a) (b)

Your turn