EECS 3101 M: Design and Analysis of Algorithms

Suprakash Datta
Office: LAS 3043

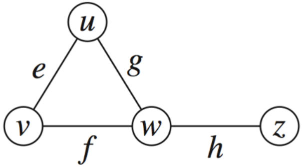
Course page: `http://www.eecs.yorku.ca/course/3101M`
Also on Moodle

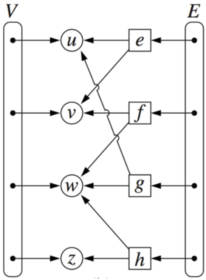
Graph Representations

- Edge list
- Adjacency list
- Adjacency matrix

Edge Lists

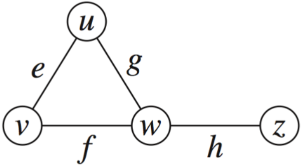
- Vertex object: reference to position in vertex sequence
- Edge object: origin vertex object, destination vertex object, reference to position in edge sequence
- Vertex sequence: sequence of vertex objects
- Edge sequence: sequence of edge objects

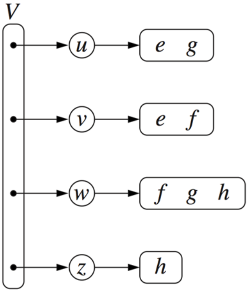




Adjacency Lists

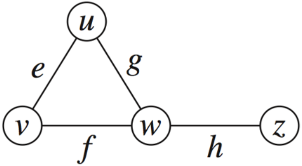
- Incidence sequence for each vertex: sequence of references to edge objects of incident edges
- Augmented edge objects: references to associated positions in incidence sequences of end vertices





Adjacency Matrix

- Edge list structure
- Augmented vertex objects: Integer key (index) associated with vertex
- 2D-array adjacency array: Reference to edge object for adjacent vertices, null for non nonadjacent vertices
- The “old fashioned” version just has 0 for no edge and 1 for edge



		0	1	2	3
$u \longrightarrow$	0		e	g	
$v \longrightarrow$	1	e		f	
$w \longrightarrow$	2	g	f		h
$z \longrightarrow$	3			h	

Performance

▪ n vertices, m edges ▪ no parallel edges ▪ no self-loops	Edge List	Adjacency List	Adjacency Matrix
Space	$n + m$	$n + m$	n^2
incidentEdges(v)	m	$\deg(v)$	n
areAdjacent (v, w)	m	$\min(\deg(v), \deg(w))$	1
insertVertex(o)	1	1	n^2
insertEdge(v, w, o)	1	1	1
removeVertex(v)	m	$\deg(v)$	n^2
removeEdge(e)	1	1	1

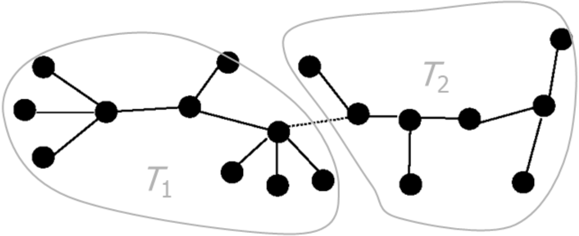
Minimum Spanning Trees (MST)

- Undirected, connected graph $G = (V, E)$
- Weight function $w : E \to \mathbb{R}$ (assigning cost or length or other values to edges)
- Spanning tree: tree that connects all vertices
- Minimum spanning tree: tree T that connects all the vertices and minimizes $w(T) = \sum_{(u,v) \in T} w(u, v)$

Minimum Spanning Trees: Questions

- Is DP applicable?
- Is a greedy strategy applicable?

MST: Optimal Substructure



- Removing the edge (u, v) partitions T into T_1 and T_2: $w(T) = w(T_1) + w(T_2) + w(u, v)$
- We claim that T_1 is the MST of $G_1 = (V_1, E_1)$, the subgraph of G induced by vertices in T_1.
- Similarly, T_2 is the MST of G_2

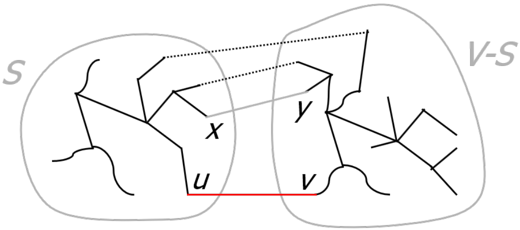
MST: Greedy Choice Property

Greedy choice property: locally optimal (greedy) choice yields a globally optimal solution

Theorem:

- Let $G = (V, E)$, and let $S \subseteq V$ and
- Let (u, v) be min-weight edge in G connecting S to $V - S$
- Then $(u, v) \in T$ for some MST T of G

MST: Proof of Greedy Choice Property



- Let (u, v) be min-weight edge in G connecting S to V–S; suppose $(u, v) \notin T$
- look at path from u to v in T
- swap (x, y), the first edge on path from u to v in T that crosses from S to $V - S$, with (u, v)
- this decreases the cost of T: contradiction (T supposed to be MST)

Generic MST Algorithm

```
Generic-MST(G, w)
A←∅  // Contains edges that belong to a MST
while A does not form a spanning tree do
    Find an edge (u,v) that is safe for A
    A←A∪{(u,v)}
return A
```

- Loop invariant: before each iteration, A is a subset of some MST
- Safe edge – edge that preserves the loop invariant

Generic MST Algorithm - 2

```
MoreSpecific-MST(G, w)
  A←∅  // Contains edges that belong to a MST
  while A does not form a spanning tree do
    Make a cut (S, V-S) of G that respects A
    Take the min-weight edge (u,v) connecting S to V-S
    A←A∪{(u,v)}
  return A
```

- A cut respects A if no edge of A crosses the cut
- Same LI: before each iteration, A is a subset of an MST
- Correctness proof in Theorem 23.1 in the text
- Many ways to choose cuts

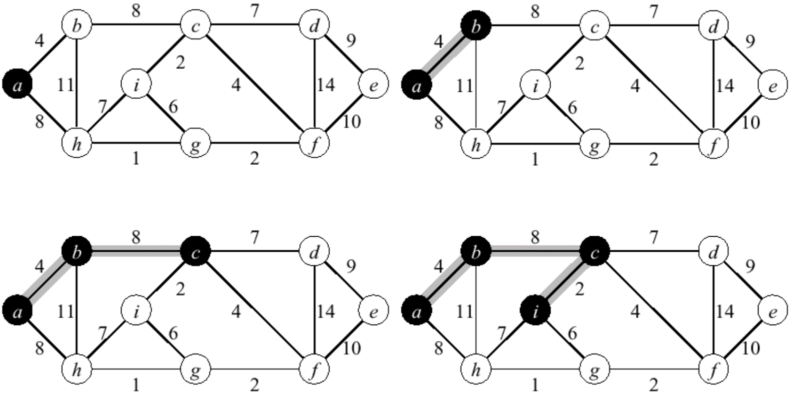
Prim's Algorithm

- Vertex based algorithm
- Grows one tree T, one vertex at a time
- Imagine a "blob" covering the portion of T already computed
- Label the vertices v outside the blob with $key[v] =$ the minimum weight of an edge connecting v to a vertex in the blob, $key[v] = \infty$, if no such edge exists
- At each iteration, add the minimum weight vertex to T

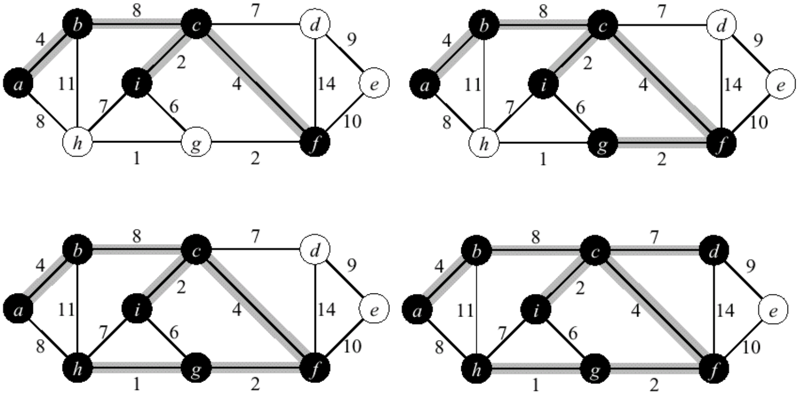
Prim's Algorithm: Steps

- Pseudocode on pg 634
- Put all vertices in a priority queue Q with labels ∞
- Remove the start vertex and set its label to 0
- While Q is not empty, remove the vertex u with the minimum label and add it to the tree;
  For each neighbour v of u in Q, if $w(u, v) < label[v]$, set $label[v] = w(u, v)$

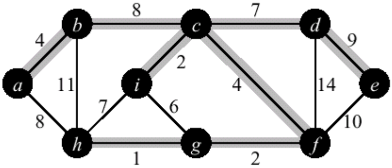
Prim's Algorithm: Illustration



Prim's Algorithm: Illustration



Prim's Algorithm: Illustration



Prim's Algorithm: Analysis

- Proof of correctness on page 636
- Time $= O(|V| T(\mathit{ExtractMin})) + O(|E| T(\mathit{ModifyKey}))$
- Times depend on PQ implementation
- Heap based PQ:
  $\mathit{BuildPQ}$: $O(n)$, $\mathit{ExtractMin}$ and $\mathit{ModifyKey}$: $O(\lg n)$
  So running time:
  $O(|V| \log |V| + |E| \log |V|) = O(|E| \log |V|)$
- With Fibonacci heaps: $O(|V| \log |V| + |E|)$

Kruskal's Algorithm

- Edge based algorithm
- Add the edges one at a time, in increasing weight order
- The algorithm maintains A – a **forest** of trees. An edge is accepted it if connects vertices of distinct trees

Kruskal's Algorithm: Requirements

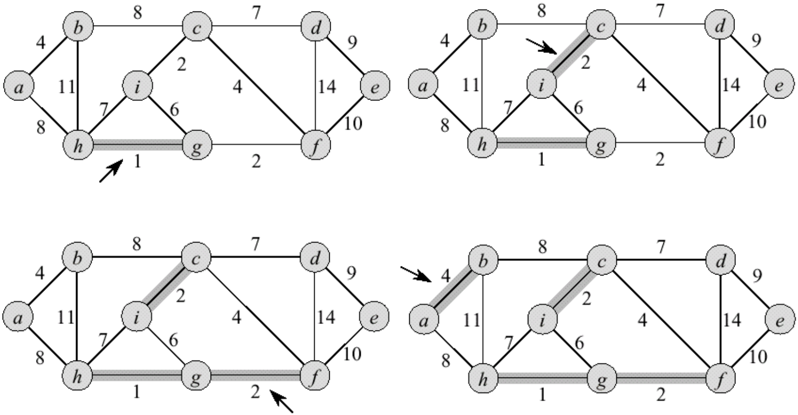
We need an ADT that maintains a partition, i.e.,a collection of disjoint sets

Operations:

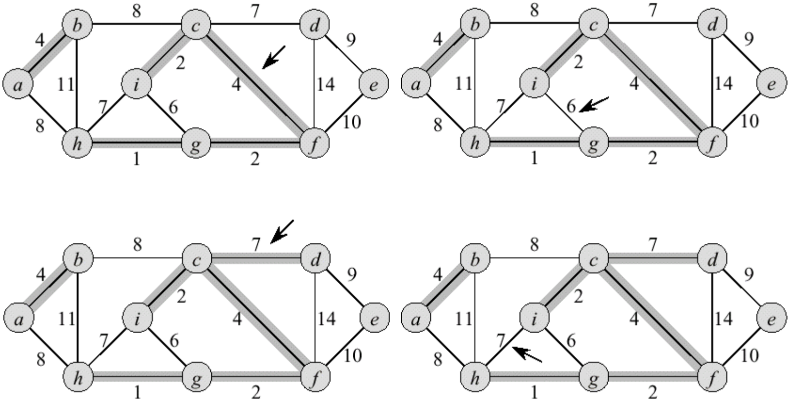
- $MakeSet(S, x)$: $S \leftarrow S \cup x$
- $Union(S_i, S_j)$: $S \leftarrow S - S_i, S_j \cup S_i \cup S_j$
- $FindSet(S, x)$: returns unique $S_i \in S$, where $x \in S_i$

Good ADT's for maintaining collections of disjoint sets are covered in EECS 4101

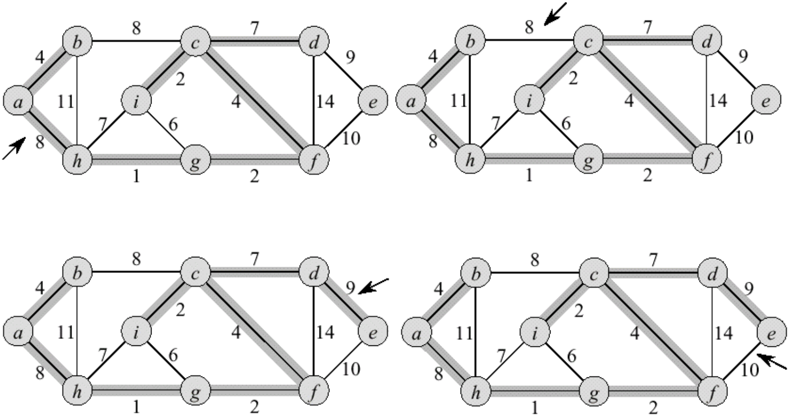
Kruskal's Algorithm: Illustration



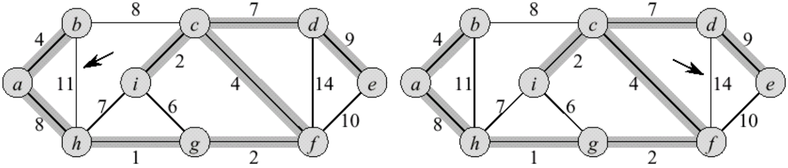
Kruskal's Algorithm: Illustration



Kruskal's Algorithm: Illustration



Kruskal's Algorithm: Illustration



Kruskal's Algorithm: Analysis

- Proof of correctness: easy since minimum weight edge has to be a safe edge
- Sorting the edges $O(|E| \lg |E|) = O(|E| \lg |V|)$
- $O(|E|)$ calls to FindSet, Union
- With advanced data structures, the running time is $O(|E| \lg |V|)$