Introduction to Algorithms Lecture 11

By: Darmen Kariboz

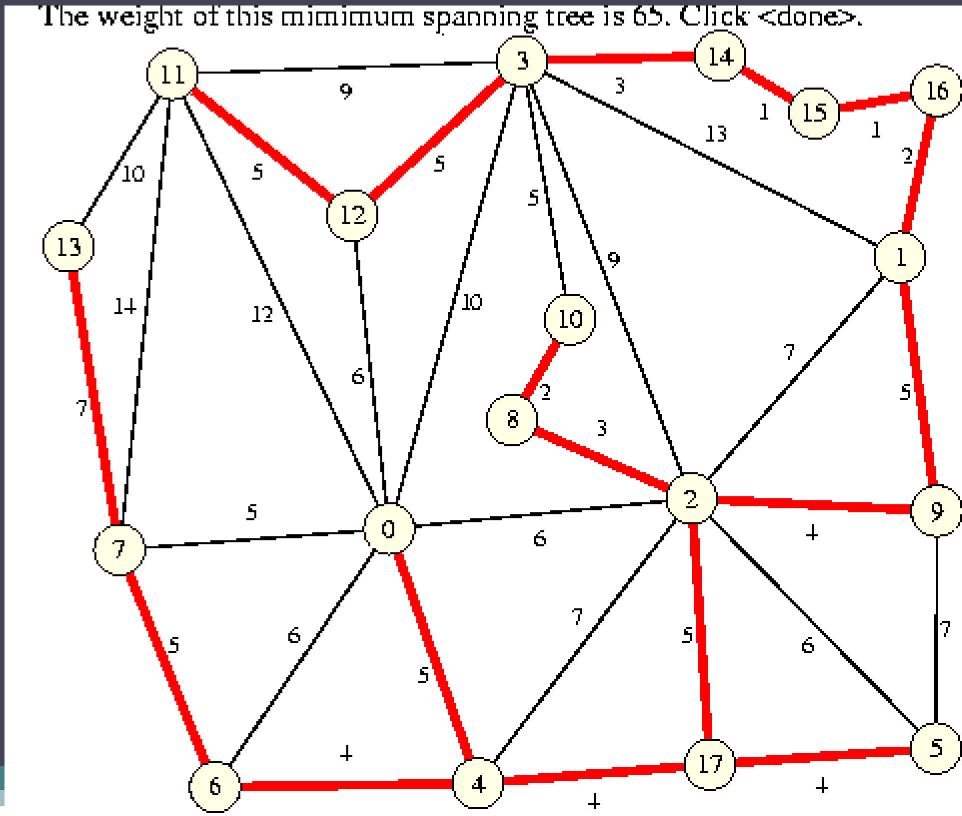
Outline

- Minimum spanning tree
- Prim’s Algorithm
- Kruskal’s Algorithm
- Dijkstra’s Algorithm
- Floyd Warshall’s Algorithm

Problem definition

- In a city, intersections (nodes) are connected by streets (edges).
- Now an electricity network is to be built that provides all intersections with electricity.
- Cables may be laid only under streets that already exist.
- The question is how to connect all intersections with a network that is as short as possible.
- This means that a *minimum* electricity network is to be *spanned* that provides all intersection points with electricity.

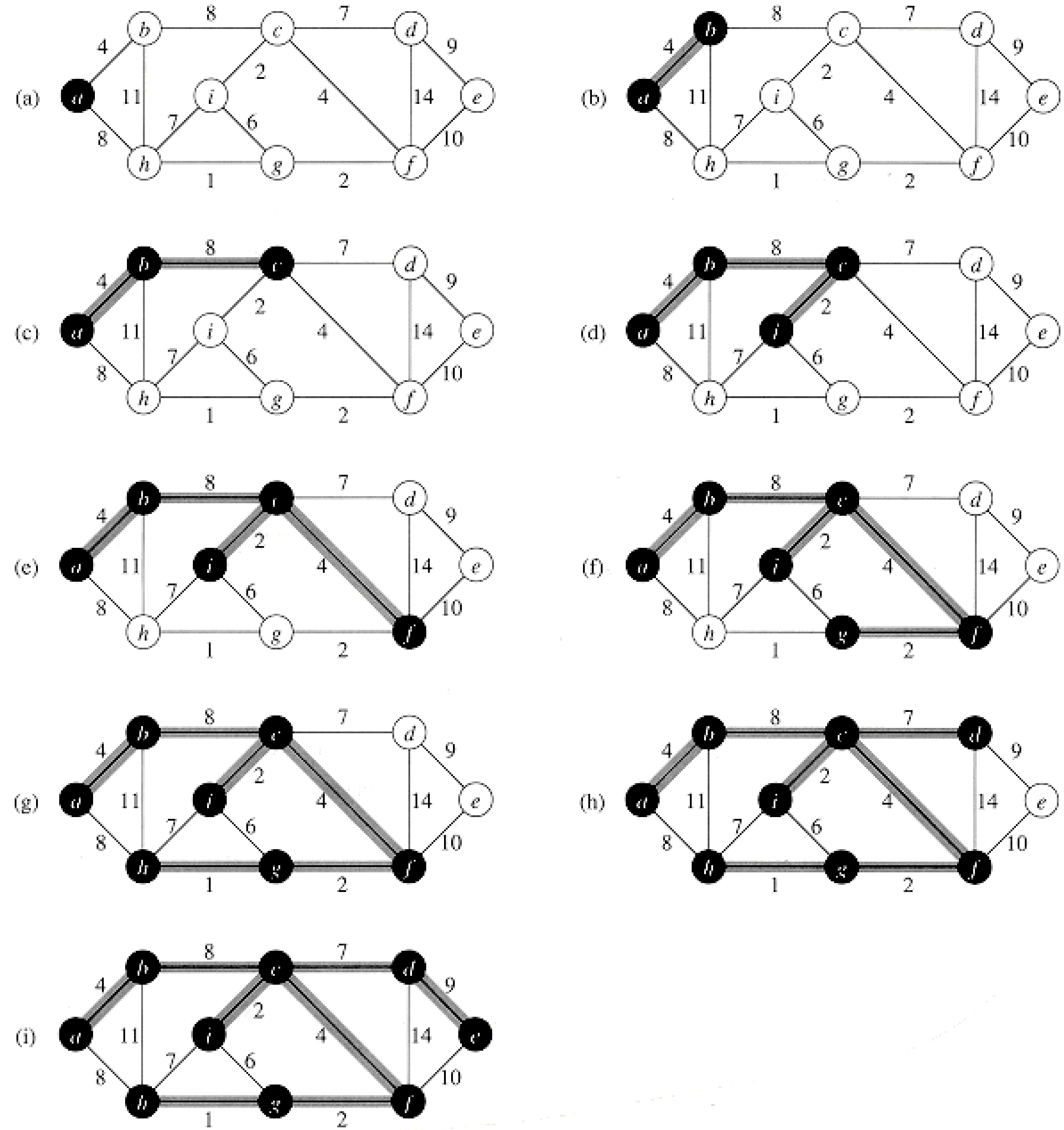
Minimum spanning tree



- is a subgraph that connects all nodes and has minimum weight.

Prim's

Algorithm
- every step connect smallest edge to current tree.

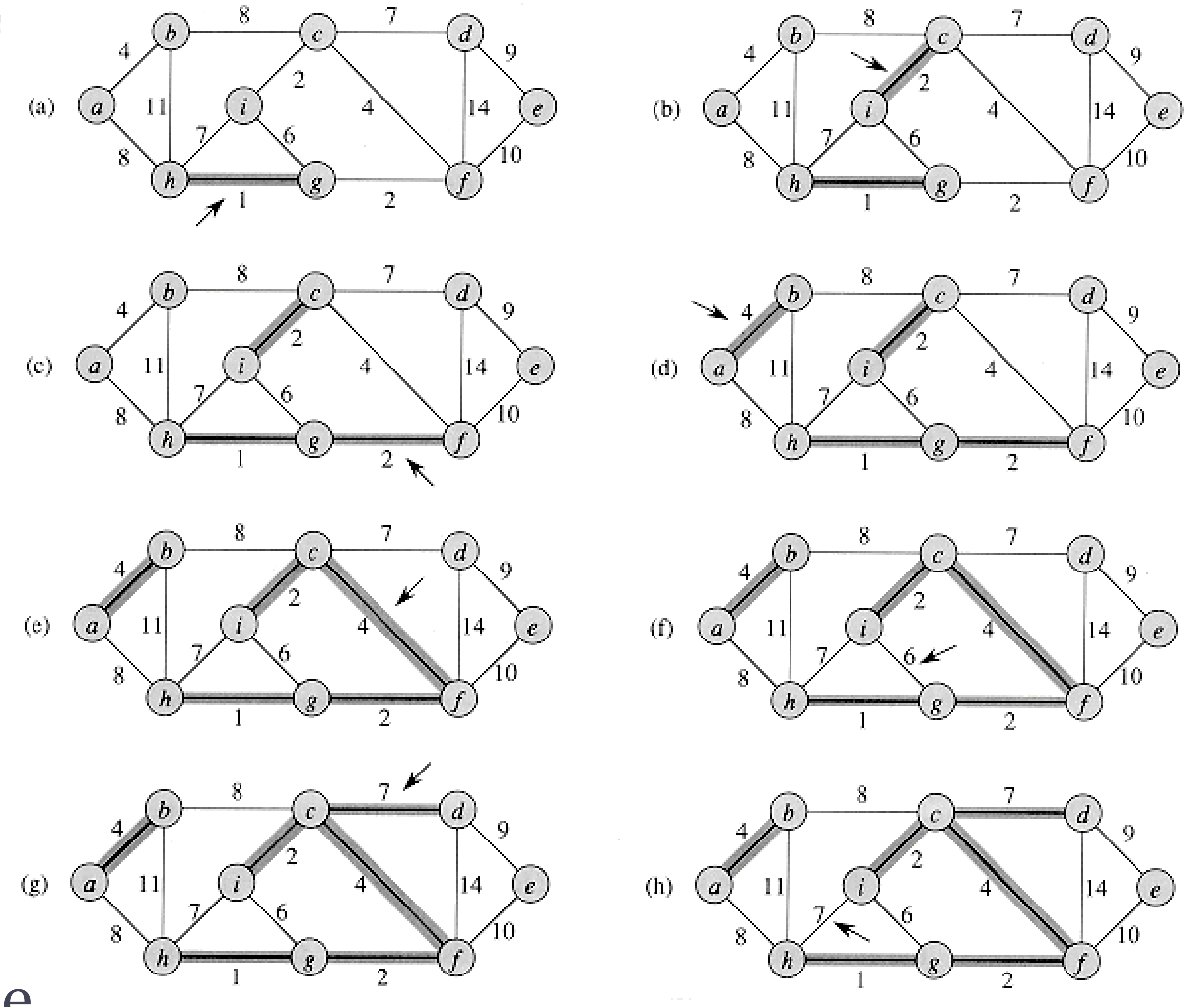


Prim's Algorithm

```
E = vector of edges
V = vector of nodes
TE = empty vector
TV = empty SET
TV <- add first node
While size(TV) < size(V) do
   min = infinity
   foreach edge(A,B) in E
      if (A in TV and not B in TV) or
         (B in TV and not A in TV) then
                  if min > length(edge) then
                        min = length(edge)
                        minEdge = edge;  minA = A;  minB = B;
   TE <- add minEdge
   TV <- add minA if not exist
   TV <- add minB if not exist
```

Kruskal’s

Algorithm
- Every step select minimum acceptable edge to forest, until one tree is formed.

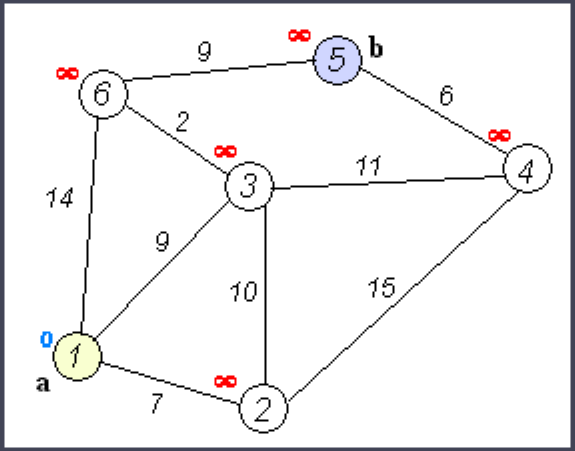


Kruskal's Algorithm

A <- Is empty set
for each vertex *v* from *V*[*G*]
        do MAKE-SET (*v*)

sort the edges of *E* by nondecreasing weight *w*

for each edge (*u*, *v*) from *E*,
                        in order by nondecreasing weight
        do if FIND-SET(*u*) not equals FIND-SET(*v*)
                then *A* <- {(*u*, *v*)} // add new edge to A
                        UNION (*u*, *v*)

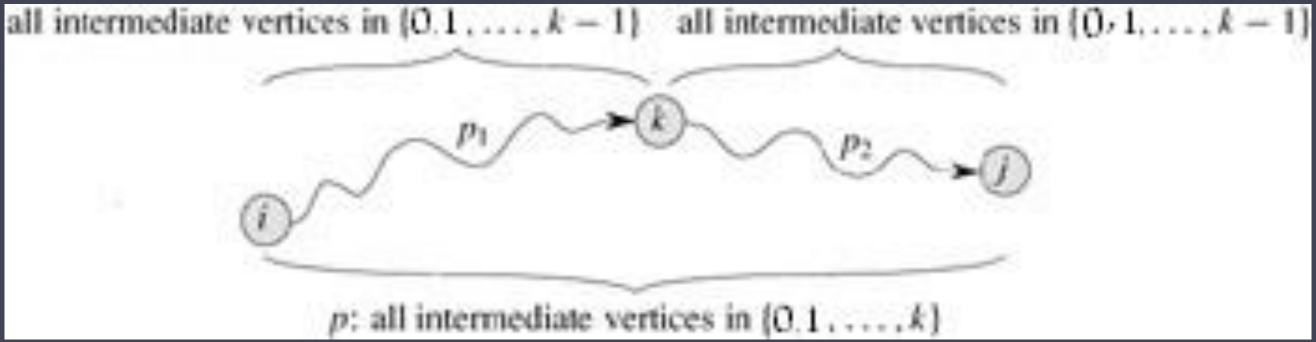


Dijkstra's

Algorithm
- Finds shortest distance between source and all other nodes in arbitrary directed graphs with nonnegative weights.

pseudocode

```
for each vertex v in Graph:                      // Initializations
    dist[v] := infinity ;                        // Unknown distance function from source to v
    previous[v] := undefined ;                   // Previous node in optimal path from source
end for ;
dist[source] := 0 ;                              // Distance from source to source
Q := the set of all nodes in Graph ;             // All nodes in the graph are in Q
while Q is not empty:                            // The main loop
    u := vertex in Q with smallest distance in dist[] ;
    if dist[u] = infinity:
        break ;                      // all remaining vertices are inaccessible from source
    end if ;
    remove u from Q ;
    for each neighbor v of u:                    // where v has not yet been removed from Q.
        alt := dist[u] + dist_between(u, v) ;
        if alt < dist[v]:                        // Relax (u,v,a)
            dist[v] := alt ;
            previous[v] := u ;
            decrease-key v in Q;                 // Reorder v in the Queue
        end if ;
    end for ;
end while ;
```



Floyd Warshall

Algorithm

- Finds shortest distances between every pair of vertices in a given edge weighted directed Graph.

Floyd Warshall's Algorithm

```
// init solution matrix same as input matrix
dist  <- graph

for k=0 to V   // intermediate
  for i=0 to V    // source
      for j=0 to V    // destination
           if dist[i][k] + dist dist[k][j] < dist[i][j]
                then dist[i][j] = dist[i][k] +  dist[k][j]
```

HomeWork Assignments

- **Realize Kruskal's algorithm for given Edge List.**
- Output Minimum spanning tree weight, and edges used in that tree.

- **Realize Dijkstra's Algorithm for given Distance Matrix.**
- Output distance from first node to all other.

- **Realize Floyd Warshall's Algorithm for given Distance Matrix.**
- Output distance matrix for all nodes.