COMP20003
Algorithms and Data Structures
Greedy Algorithms and the MST

Nir Lipovetzky
Department of Computing and Information Systems
University of Melbourne
Semester 2

Greedy Algorithms

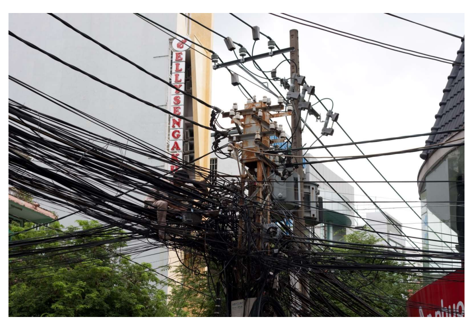
Greedy algorithms are used in optimization problems

Greedy algorithms keep taking the next best step repeatedly, until the best solution is reached

- Dijkstra's algorithm is greedy: takes the next best edge to add to the path tree

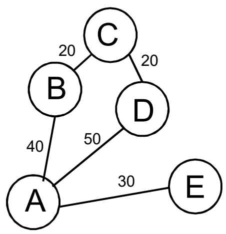
Minimum Spanning Tree

- Undirected weighted graphs
- Minimum spanning tree = subgraph that is:
  - A tree (no cycles)
  - Contains every vertex (spans)
  - Minimum sum of edge weights
- Also called:
  - Minimum weight spanning tree (sum of weights)
  - Minimal spanning tree (might be more than one)

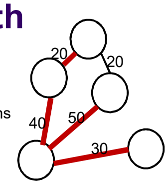


Saigon X, Health and Safety? What's That? Reena Mahtani
Creative Commons License

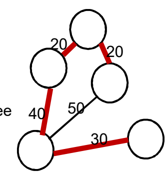
MST *vs.* shortest path



Shortest paths



Min Span Tree



MST and Graph characteristics

- Graph must be connected
- MST *must* have exactly *V-1* edges
- No cycles in MST

Building a MST: General approach

- **Start** with isolated vertices (all), **no edges**
- **Begin** with **any vertex** (Prim's) **or** the **least cost edge** (Kruskal's)
  - This is a MST subtree

Keep adding vertices/edges to **extend** this MST **subtree**

- Shortest connections
- No cycles

Famous MST algorithms

- Prim's
  - **Shortest connection networks and some generalizations**. R.C. Prim, *Bell System Technical Journal* **36**(6), 1389-1401, 1957.
- Kruskal's
  - **On the shortest spanning subtree of a graph and the traveling salesman problem**. J.B. Kruskal, *Proceedings of the American Mathematical Society* **7**, 48-50, 1956.
- Borůvka's (1926, published in Czech)
  - **Otakar Borůvka on minimum spanning tree problem: translation of both the 1926 papers, comments, history**. Nešetřil, Jaroslav; Milková, Eva; Nešetřilová, Helena (2001). *Discrete Mathematics* **233** (1–3): 3–36

Prim's MST algorithm

- Preferred method for dense graphs
- Easiest with matrix representation
- Prim's algorithm relies on picking the next best edge that joins two set of vertices:
  - Vertices already in the tree (S)
  - Vertices not yet in the tree (V-S)

These two sets form a "cut"

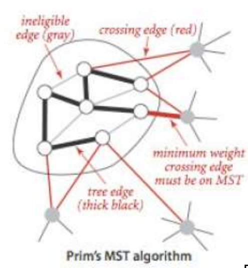
Definitions

- A Cut *(V,V-S)* of *G* is a partition of *V*
- Cross: an edge *(u,v)* in *E* with one endpoint in *S* and the other in *V-S*
- Light edge: the minimum weight edge crossing the cut
- Respect: a cut respects a set *A* of edges if no edge in *A* crosses the cut

Cut during MST construction

Cut:

- *S*: set of vertices already **in** the MST
- *V-S*: **not** yet in the MST
  - Fringe: part of *V-S* **one step away** from the MST
  - **Vertices in *V-S*** have a cost (**distance**) from the MST subtree so far constructed
  - **Distances** between non-MST vertices and MST vertices **are updated** as vertices are added to MST



From R. Sedgewick, Algorithms 4th edition

Prim's MST construction

Start:

- *S* = {any vertex}
- *S-V* = {all the others}
- The **cut** *S/V-S* **respects** edges **in** the **MST** as it is being constructed
- The cut itself changes

Prim's MST construction

Respect:

- The cut S/V-S respects edges in the MST being constructed
  - Fringe: vertices in V-S one step away from the MST
  - Vertices in V-S have a cost(distance) from the MST subtree so far constructed (some may be ∞)

Prim's MST construction

- Pick **lightest** edge **crossing** the cut:
  - Crossing edge *(u,v)* has *u* in *S* and *v* in *V-S*
    - Add v to S
    - Keep track of path (pred[ ])
    - Update distances between non-MST vertices and MST vertices (could be closer now) (w[ ])
- Repeat until V-S = `{0}`
- Reconstruct connections and distances from `pred`[ ] and `wt`[ ]

Prim's: Pseudocode

```
void prim(G,wt,root)
{
   for every u in V { dist[u] = ∞;  inmst[u] = FALSE;}
   dist[root] = 0; pred[root] = NULL;
   PQ = makePQ(V); /* all vertices in PQ */
   while(!empty PQ){
     u = deletemin(PQ);
     for every (v adjacent to u){
         if ((inmst[v]==FALSE)&& (w[u][v]< dist[v])){
            dist[v] = w[u][v]; /* update distance */
            decreasewt(PQ,v,dist[v]);/* update PQ dist*/
            pred[v]=u; /* update path information */
         }
     }
     inmst[u] = TRUE;
   }
 } /* at end: MST = {{v,pred[v]}: v in V - {root}} */
```

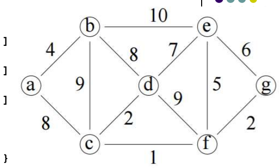
Fringe vertices are in a priority queue

- This is a Priority-First Search

Prim's example

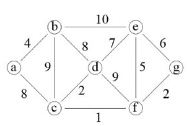
	a	b	c	d	e	f	g	
Dist = [								]
pred = [								]
inMST =[								]

PQ = { }

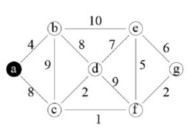



```
void prim(G,wt,root)
{
   for every u in V { dist[u] = ∞;  inmst[u] = FALSE;}
   dist[root] = 0; pred[root] = NULL;
   PQ = makePQ(V); /* all vertices in PQ */
   while(!empty PQ){
     u = deletemin(PQ);
     for every (v adjacent to u){
           if ((inmst[v]==FALSE)&& (w[u][v]< dist[v])){
              dist[v] = w[u][v]; /* update distance */
              decreasewt(PQ,v,dist[v]);/* update PQ dist*/
           pred[v]=u; /* update path information */
           }
     }
     inmst[u] = TRUE;
   }
 } /* at end: MST = {{v,pred[v]}: v in V - {root}} */
```


Prim's example

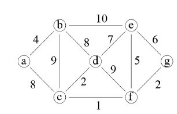


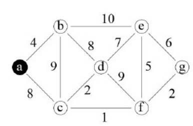
Connected graph



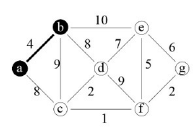
Step 0
S={a}
V \ S = {b,c,d,e,f,g}
lightest edge = {a,b}

Example from Mordechai Golin, Hong Kong University of Science and Technology
http://www.cse.ust.hk/faculty/golin/COMP271Sp03/Notes/MyL10.pdf

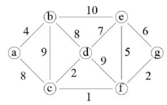


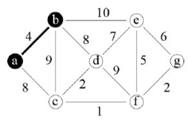


Step 1.1 before
S={a}
V \ S = {b,c,d,e,f,g}
A={}
lightest edge = {a,b}

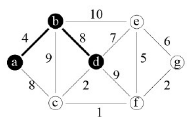


Step 1.1 after
S={a,b}
V \ S = {c,d,e,f,g}
A={{a,b}}
lightest edge = {b,d}, {a,c}

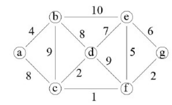


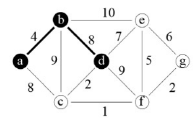


Step 1.2 before
S={a,b}
V \ S = {c,d,e,f,g}
A={{a,b}}
lightest edge = {b,d}, {a,c}

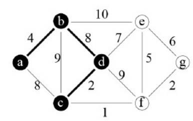


Step 1.2 after
S={a,b,d}
V \ S = {c,e,f,g}
A={{a,b},{b,d}}
lightest edge = {d,c}

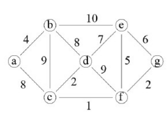


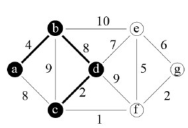


Step 1.3 before
S={a,b,d}
V \ S = {c,e,f,g}
A={{a,b},{b,d}}
lightest edge = {d,c}

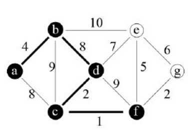


Step 1.3 after
S={a,b,c,d}
V \ S = {e,f,g}
A={{a,b},{b,d},{c,d}}
lightest edge = {c,f}

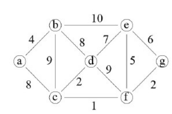


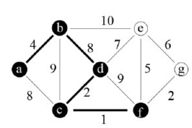


Step 1.4 before
S={a,b,c,d}
V \ S = {e,f,g}
A={{a,b},{b,d},{c,d}}
lightest edge = {c,f}

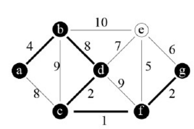


Step 1.4 after
S={a,b,c,d,f}
V \ S = {e,g}
A={{a,b},{b,d},{c,d},{c,f}}
lightest edge = {f,g}

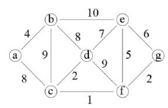


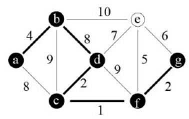


Step 1.5 before
S={a,b,c,d,f}
V \ S = {e,g}
A={{a,b},{b,d},{c,d},{c,f}}
lightest edge = {f,g}

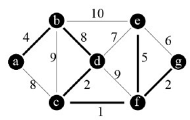


Step 1.5 after
S={a,b,c,d,f,g}
V \ S = {e}
A={{a,b},{b,d},{c,d},{c,f},
{f,g}}
lightest edge = {f,e}





Step 1.6 before
S={a,b,c,d,f,g}
V \ S = {e}
A={{a,b},{b,d},{c,d},{c,f},
{f,g}}
lightest edge = {f,e}



Step 1.6 after
S={a,b,c,d,e,f,g}
V \ S = {}
A={{a,b},{b,d},{c,d},{c,f},
{f,g},{f,e}}
MST completed

Prim's: Analysis

Initialize arrays pred[] dist[]: O( )
Make PQ of vertices: if heap O( )
Loop while PQ not empty *
  Deletemin (if heap) O( )
  Update adjacent weights,
    adjust wt in PQ O( )

= O( )
Noting that V deg(u) = E,
= O( )
= O( ) for dense graphs with heap PQ

Prim's *vs.* Dijkstra's