Week 11: Minimum Spanning trees

Agenda:

- Minimum Spanning Trees
- Prim's Algorithm

Reading:

- Textbook : 561-574

Minimum spanning tree (MST) problem:

- Input: edge-weighted (simple, undirected) connected graphs (positive weights)

- Notions:

  - subgraph, acyclic, tree

  - spanning subgraph: subgraph including all the vertices

  - spanning tree: spanning subgraph which is a tree — acyclic connected subgraph $T = (V, E')$, where $E' \subset E$

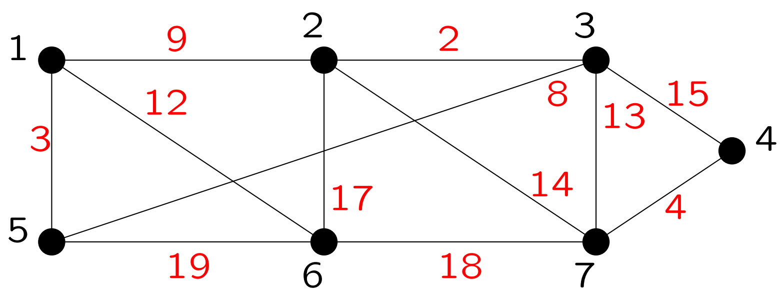
    e.g., BFS/DFS (on a connected input graph) tree is a spanning tree of the graph

  - minimum spanning tree: minimum weight

- The MST Problem:

  Find a minimum spanning tree for the input graph.

  For example:



- The minimum spanning forest problem:

  The given graph is not necessarily connected.

  Find an MST for each connected component.

Greedy algorithms and MST problem:

- Greedy algorithms:
  - greedy — each step makes the best choice (locally maximum)
  - iterative algorithms
  - optimal substructure

    an optimal solution to the original problem contains within it optimal solutions to subproblems

- Greedy solution may NOT be globally optimum

  e.g., matrix-chain multiplication: $A_{6\times 5} \times A_{5\times 2} \times A_{2\times 5} \times A_{5\times 6}$

  Greedy: $50 + 150 + 180 = 380$ scalar multiplications

  Dynamic programming: $60+60+72 = 192$ scalar multiplications

- The MST problem:

  Two greedy solutions are globally optimum
  - Prim's (Prim + Dijkstra + Boruvka's)

    growing the tree to include more vertices
  - Kruskal's (Kruskal + Boruvka's)

    growing the forest to become a tree

Prim's algorithm for the MST problem:

- Input: an edge-weighted (simple, undirected, connected) graph (positive weights)
- Output: an MST
- Idea:
  - suppose we have already an MST T' spanning subset V' of vertices (T' is initialized empty and V' is initialized to contain any one vertex)
  - grow T' to span one more vertex $v \in V - V'$
  - v is selected such that there is a vertex $u \in V'$, edge (u, v) is the minimum weighted over all edges of form (u', v') where $u' \in V'$ and $v' \in V - V'$
  - when V' becomes V, terminate
- One simplest implementation:

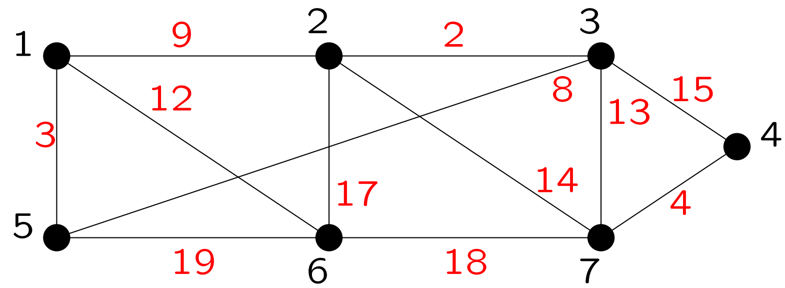
```
procedure primMST(G)                **G = (V, E)

S = {1}
T = ∅
while |S| < |V| do
    find a minimum weight edge e = uv:   u ∈ S and v ∈ V − S
    S ← S + v
    T ← T + {uv}
return T
```

Running time analysis:

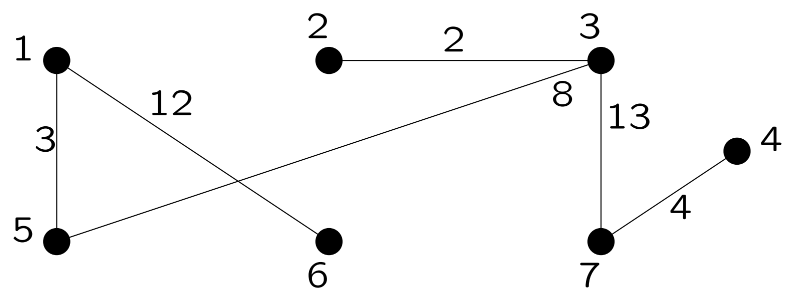
1. finding such an edge in $O(n^2)$ (or $O(m)$) time
2. there are $n - 1$ edges in the output MST
3. therefore, in total $O(n^3)$ (or $O(nm)$) time

Prim's algorithm for the MST problem — an example:

- Input graph G:



- $\texttt{primMST}(G, w, 1)$ returns:



- Correctness of Prim's algorithm (to follow)

- Improvement over the simplest implementation

  Observation: <u>every iteration it looks for minimum weight edge</u>

  — heap might help

Prim's algorithm for the MST problem — correctness:

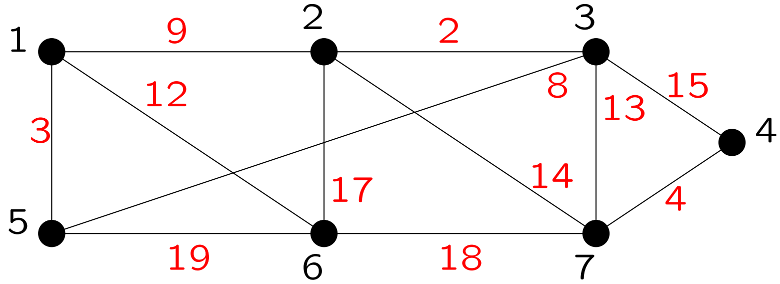
- Input graph $G = (V, E)$: $E = \{e_1, e_2, \ldots, e_m\}$
- Suppose edges in the output tree T are $e_{i_1}, e_{i_2}, \ldots, e_{i_{n-1}}$ (in the order picked by Prim's algorithm)
- Want to prove: T is an MST
- Suppose T' is an MST and it contains edges $e_{j_1}, e_{j_2}, \ldots, e_{j_{n-1}}$ (sorted in the way that it maps the edge order in T as much as possible). If $T \neq T'$ (otherwise we are done), then
  - there is a minimum index k, such that $e_{j_k} \neq e_{i_k}$
  - let T_0 denote the tree formed by $\{e_{i_1}, e_{i_2}, \ldots, e_{i_{k-1}}\}$
  - let $V_0 = V(T_0)$ and $V_1 = V - V_0$
  - adding e_{i_k} to T' creates a cycle which contains some edge, say e_{j_p}, that has one ending vertex in V_0 and the other in V_1
  - $T'' = T' + e_{i_k} - e_{j_p}$ is another spanning tree
  - T'' is another MST (why ?) sharing one more edge with T
  - repeat this argument to claim that T is also an MST
- Note: this is a proof using 'contradiction' + 'graph theory'.
- Proof can also be done by

  (`while`) Loop Invariant: <u>T is a MST on S.</u>

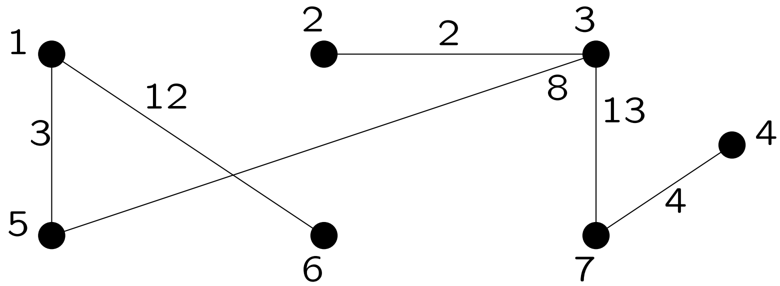
  Exercise !

Prim's algorithm for the MST problem — improvement:

- Where to improve: finding the minimum weight edge (u, v)
- Initially we need to scan all the edges, $\Theta(m)$ (worst case)
- Example: input graph G:

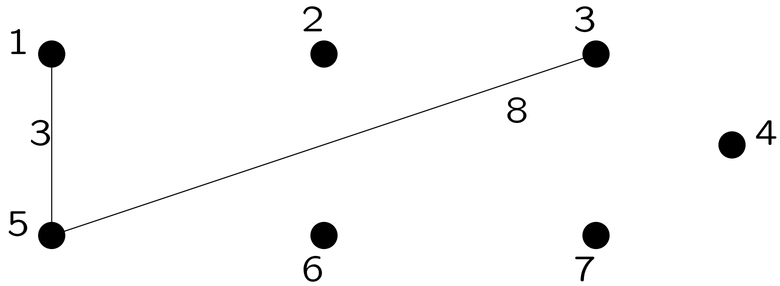


- `primMST`$(G, w, 1)$ returns:



- `primMST`$(G, w, 1)$: an intermediate tree

  <u>What are the candidate edges ?</u>



Prim's algorithm for the MST problem — improvement:

- Ideas:
  1. for each non-tree vertex v, store its minimum-weight tree neighbor $p[v]$
  2. store edges of type $(p[v], v])$ in a min-priority queue Q
  3. therefore, every time the target edge can be extracted `ExtractMin`(Q)

     note: need to update the neighbor information for non-tree vertices after the extraction

- Pseudocode:

```
procedure primMST(G)                                **G = (V, E)

for each v ∈ V(G) do
    key[v] ← ∞
    p[v] ← NIL
key[r] ← 0
Q ← V(G)
while Q ≠ ∅ do
    u ← ExtractMin(Q)                               **r dequeued first
    for each v ∈ Adj[u] do
        if v ∈ Q && w(u,v) < key[v] then            **update v
            p[v] ← u
            decrease-key(v, w(u,v))                 **key[v] ← w(u,v)
```

Prim's algorithm for the MST problem — improvement:

- Analysis of the improved algorithm:
  - correctness (almost done — need to prove that ExtractMin(Q) does extract the minimum weight edge)
  - running time: $\Theta\Big(n \log n + \sum_{u \in V} \Big(\texttt{degree}(u) \times \log n\Big)\Big)$

    so: $\Theta(m \log n)$ — adjacency list graph representation

- Remark: there may be several optimum spanning trees, Prim's algorithm only finds one.

- But if all the edge weights are distinct then the MST is unique.

- Next we will see another algorithm for computing MST.