Week 12: Minimum Spanning trees and Shortest Paths

Agenda:

- Kruskal's Algorithm
- Single-source shortest paths
- Dijkstra's algorithm for non-negatively weighted case

Reading:

- Textbook : 561-574, 580-587, 595-601

Kruskal's algorithm for the MST problem:

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

- Output: an MST

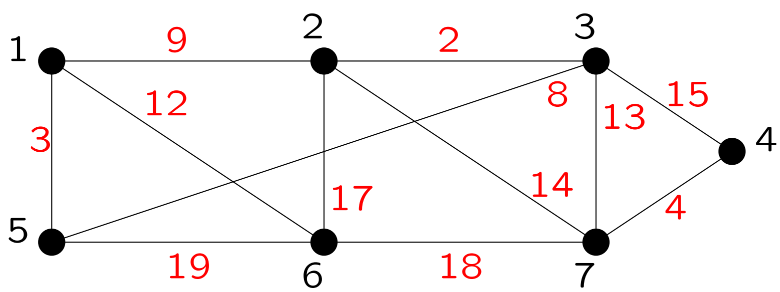
- Idea:
  - Start with a forest T on all the vertices and no edges
  - Grow the forest T to become a tree by adding one edge at a time
  - The edges are considered in non-decreasing order of their weight
  - an edge can be added if it joins two different connected components (i.e. two trees of T)
  - So an edge is added if it does not create a cycle, otherwise it is discarded
  - For each vertex we keep an index which tells the index of the "cluster" to which it belongs.
  - When we add an edge, we merge the clusters (i.e. the sub-trees) that it connects.

Kruskal's algorithm for the MST problem:

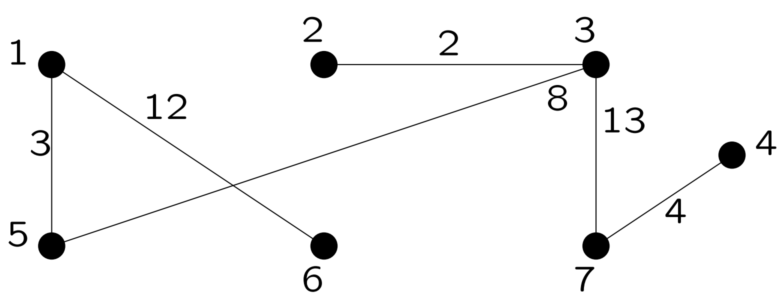
- <u>`procedure kruskal` (G)</u>

  $T \leftarrow \emptyset$
  `for each` $v \in V(G)$ `do`
      `Define cluster` $C(v) \leftarrow v$
  `sort edges in` $E(G)$ `into non-decreasing weight order`
  `for each edge` $e_i = (u, v) \in E(G)$ `do`
      `if` $C(u) \neq C(v)$ `then`
      $T \leftarrow T \cup \{e_i\}$
      `merge clusters` $C(u)$ `and` $C(v)$
  `return` T

- An example:



- `kruskalMST`(G, w) returns:



Kruskal's algorithm for the MST problem — analysis:

Correctness:

- We prove that after every step, where we have selected i edges and put into T, call it T_i, there is a MST T_{opt} which has all these i edges and has none of the edges we discarded.
- This is proved by induction on i, for all $0 \leq i \leq n-1$.
- Once we prove it for $i = n-1$ it implies that the solution is a MST.
- The critical point is in the induction step when we select an edge e to be added to T_{i-1} to obtain T_i but T_{opt} does not have it.
- In this case, $T' = T_{opt} + e$ has a cycle, C.
- This cycle contains at least one edge e' that is not in T_i (why?)
- Furthermore edge e' is among the edges we have not considered yet, because up until edge e, all the decisions made were consisten with T_{opt}.
- So $w(e') \geq w(e)$. So $T'' = T_{opt} + e - e'$ is also a MST that extends T_i.
- Running time analysis: how to implement "Merge clusters $C(u)$ and $C(v)$"?

Kruskal's algorithm for the MST problem — analysis:

Running time analysis:

- Each cluster will be an unordered linked list of vertices in that cluster
- Each vertex v also keeps the index of the cluster to which it belongs
- To find $C(v)$ it takes $O(1)$ time only (check the index)
- To merge $C(v)$ and $C(u)$: merge the smaller list into the larger one and update the index of the vertices whose list is merged.
- Thus, merging $C(v)$ and $C(u)$ takes $O(\min\{|C(u)|, |C(v)|\})$ time.
- Observation: each time we update the reference for a vertex the size of the cluster to which it belongs at least doubles; starts from 1 and goes up to n
- Thus: number of times we update a vertex's reference is $O(\log n)$.
- Total time for all merges and cluster updates: $O(n \log n)$.
- Time for sorting edges: $O(m \log m) = O(m \log n)$, time for the while loop: $O(m) + O(n \log n)$.
- Total time for Kruskal's algorithm $O((m + n) \log n)$, same as Prim's algorithm.

Shortest path problems:

- BFS recall: outputs every s-to-v shortest path
  - s — start vertex
  - v — reachable vertex from s (residing in a same connected component)
  - shortest — # edges
  - running time $\Theta(n + m)$
- BFS solves *the single-source-shortest-path problem* on undirected unweighted graphs

  Single-Source-Shortest-Path (SSSP) problem: given a source s, find out for all vertices their shortest paths from s
- Variants:
  - single source *vs.* all pairs
  - graphs: undirected *vs.* directed
  - edges: unweighted *vs.* weighted
  - edge weights: non-negative *vs.* may have negative weights
  - digraphs: acyclic *vs.* may have di-cycles

  Note: if there is no path, the distance is set to ∞ ...
-

  SSSP problem on non-negatively weighted digraphs

  Dijkstra's algorithm (today)

Dijkstra's SSSP algorithm:

- $d[v]$ — weight of the shortest path from source s to v

  if no such path, set to ∞

- Idea in Dijkstra's algorithm:
  - greedily grows an SSSP tree
  - ensures that when adding a vertex, its shortest path in the current (induced) subgraph is determined
  - records for every non-tree vertex v its best parent tree vertex $p[v]$

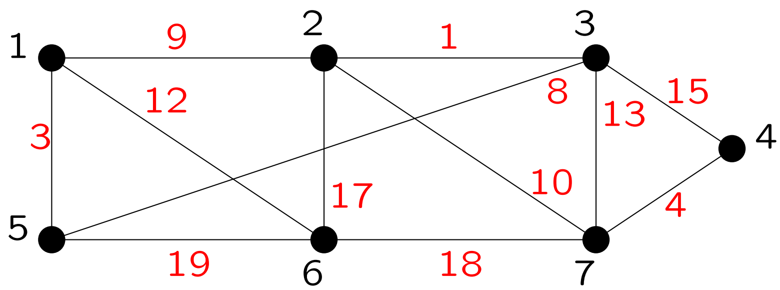
  Note: very similar to Prim's MST algorithm (the min-priority queue implementation)

- Pseudocode (use $d[v]$ as the key):

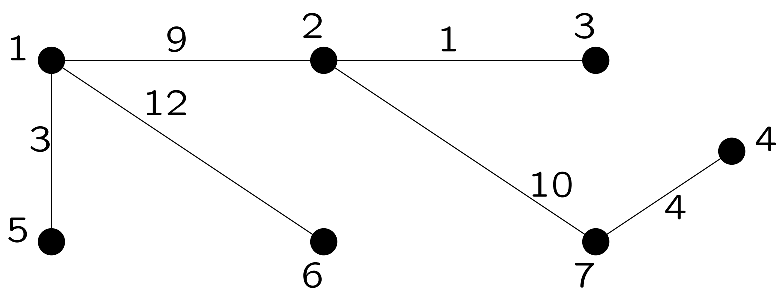
```
procedure dijkstra(G, w, s)          **G = (V, E)

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

Dijkstra's SSSP algorithm — an example:

- Input graph G:



- `dijkstra`$(G, 1)$:



- `dijkstra`$(G, 1)$ trace:

v / $d[v]/p[v]$	1 0/NIL	2 ∞/NIL	3 ∞/NIL	4 ∞/NIL	5 ∞/NIL	6 ∞/NIL	7 ∞/NIL
1 dequeued	0/NIL	9/1	∞/NIL	∞/NIL	3/1	12/1	∞/NIL
5 dequeued	0/NIL	9/1	11/5	∞/NIL	3/1	12/1	∞/NIL
2 dequeued	0/NIL	9/1	10/2	∞/NIL	3/1	12/1	19/2
3 dequeued	0/NIL	9/1	10/2	25/3	3/1	12/1	19/2
6 dequeued	0/NIL	9/1	10/2	25/3	3/1	12/1	19/2
7 dequeued	0/NIL	9/1	10/2	23/7	3/1	12/1	19/2

Dijkstra's SSSP algorithm — analysis:

- Applies to undirected graphs too

  See the last example :-)

- Running time:

  Same as the running time for Prim's MST algorithm

  — $\Theta(m \log n)$, assuming adjacency list graph representation and min-priority queue implemented by a heap

- Correctness:

  Let $S = V - Q$

  (`while`) Loop Invariant: for every $v \in S$, $d[v]$ records the weight of the shortest path from s to v in graph G

  Proof:

  – initialization (S is empty):

  – maintenance:
    Exercise: fill in the detail

  – termination: S becomes V, so LI implies that for every v, $d[v]$ records the weight of the shortest path from s to v in graph G