# Dijkstra's Algorithm for Single Source Shortest Path

Given a weighted graph, find the shortest path from a given vertex to all other vertices.

Path (from $v_0$ to $v_k$):
$v_0 v_1 \ldots v_k$, such that $(v_i, v_{i+1})$ is an edge in the graph, for $i < k$.

Simple Path: In the path, if $i \neq j$ then $v_i \neq v_j$.

Weight of the path: sum of weights of the edges in the path.

Shortest path: path with minimal weight.

Intuition behind algorithm:

Assuming positive integral distances between nodes.

Walk in "all possible directions" for 1 step. If we encounter a node, then the shortest distance to it must be 1.
Then walk further for 1 step in all possible directions. If we encounter a node, not yet encountered, then the shortest distance to it must be 2.

Continue, similarly until all nodes are encountered.

An example run:
http://www.unf.edu/~wkloster/foundations/DijkstraApplet/DijkstraApplet.htm

Graph $G = (V, E)$, with $wt(u, v)$ giving the weight of edge $(u, v)$
Graph edges are given as adjacency list.

$s$ is the source vertex.

$Dist(u)$ denotes the currently best known distance from $s$ to $u$.

Rem denotes the nodes for which the shortest distance is not yet found.
$V - Rem$ will thus denote the vertices for which shortest distance is already found.

$prev(v)$ denotes the predecessor node of $v$ in the currently known shortest path from $s$ to $v$.

# Dijkstra's Algorithm

Set $Dist[u] = \infty$, and $Prev[u] = null$ for all $u \in V$.

Set $Dist[s] = 0$.

Set $Rem = V$.

While $Rem \neq \emptyset$ {

1. Find node $u \in Rem$ with minimal $Dist[u]$.
2. $Rem = Rem - \{u\}$.
3. For each edge $(u, v)$ such that $v \in Rem - \{u\}$ {
   3.1 Let $Z = min(Dist[v], Dist[u] + wt(u, v))$.
   3.2 If $Dist[v] > Z$, then let $Dist[v] = Z$ and $Prev(v) = u$.
   }

}

Correctness:
Let $Short(v)$ denote the shortest distance from $s$ to $v$.
By induction we will show that in every iteration, when a node $u$ is removed from Rem, then

(a) $Dist(u) = Short(u)$, and $Dist(u)$ is never updated again (note that this implies: for all $w \in V - Rem$, $Dist(w) = Short(w)$).
(b) $Dist(v)$, at the beginning of any iteration, is the shortest distance from $s$ to $v$, using only the nodes in $V - Rem$ in the path (except for the "last" node $v$ itself).
(c) For all the nodes $w$ in $Rem$, and $w'$ in $V - Rem$, $Short(w) \geq Short(w')$.

At the beginning the invariants clearly hold.

Now consider any iteration of the loop.
Suppose $u$ is deleted from Rem in this iteration.

Then, we first claim that for all $w \in Rem - \{u\}$, at this iteration, $Short(w) \geq Dist(u)$.
Suppose otherwise, that is $Short(w) < Dist(u)$ for some $w$ in Rem.
Choose such a $w$ which minimizes $Short(w)$.
Then, all the nodes in the shortest path from $s$ to $w$ (except for $w$) are already in $V - Rem$ (as $w$ was the closest to $s$ which is in Rem).
Thus, by inductive hypothesis (b), we have $Dist(w) = Short(w) < Dist(u)$, a contradiction to the choice of $u$.
This implies that invariants (a) and (c) hold.
For (b) note that, for $v \in Rem$, either the shortest path from $s$ to $v$ has last node (just before $v$) $u$ or not. In either case, steps 3.1 and 3.2 of the algorithm ensure that invariant (b) is maintained.

Complexity:
While Loop is executed $n$ times.

Simple implementation:
Rem as a set: $Rem[u]$ =true, if $u \in Rem$.

Overall time for
Step 1: $n * O(n)$
Step 2: $n * O(1)$
Step 3: $O(m)$
Total: $O(n^2 + m)$.

If one uses binary minheap for weights:
Step 1: $O(n)$
Step 2: $O(n \log n)$
Step 3: $O(m \log n)$
Total: $O(m \log n + n \log n)$

Can be reduced further to $O(n \log n + m)$ using Fibonacci heaps.

# Prim's Algorithm for Minimal Spanning Tree

Spanning tree of a graph $G = (V, E)$ is a tree $T = (V, E')$ such that $E' \subseteq E$.
(Note: $V$ needs to be a connected graph)

Minimal spanning tree of a weighted graph is a spanning tree of the graph with minimal weight.

Note that there may be several minimal spanning trees.

Intuition:
1. Start with one vertex (say start) to be in the spanning tree.
2. Add an edge with minimum weight that has one endpoint in the tree already constructed, and the other endpoint outside the tree constructed. (Greedy!)
3. Repeat ...

1. Suppose $Rem$ denotes the set of vertices not yet in the spanning tree constructed.
2. We will maintain an array $D(v)$ and $parent(v)$:
3. For $v$ already in the tree, $parent(v)$ gives the node to which it is connected in the tree (at the time it was added to the tree). $parent(start) = null$.
4. For $v$ in Rem, $D(v)$ gives the shortest distance from $v$ to the tree already constructed; $parent(v)$ gives the node in tree to which this shortest distance applies. Note that $D(v)$ may be $\infty$ in case there is no edge from $v$ to the nodes already in constructed tree. In this case $parent(v) = null$
5. In each iteration, we will choose the vertex $u$ in $Rem$ which minimizes $D(u)$. We will add $u$ to the tree. Furthermore, we will update $D(v)$, for $v$ in $Rem - \{u\}$, in case, $wt(u, v) < D(v)$.

Prim's algorithm

1. Pick any node in $V$ as $start$.
2. Let $D(v) = \infty$ for all $v \in V - \{start\}$
3. Let $D(start) = 0$.
2. Let $parent(v) = null$ for all $v \in V$.
4. Let $Rem = V$.
5. While $Rem$ is not empty {
   Choose a node $u$ in $Rem$ which minimizes $D(u)$.
   Delete $u$ from Rem.
   For each edge $(u, v)$ such that $v \in Rem - \{u\}$,
   { if $D(v) > wt(u, v)$,
   then update $D(v) = wt(u, v)$ and $parent(v) = u$ }
   }

Time complexity: same as Dijkstra's Algorithm

Correctness:
By Induction:
At each iteration, the tree constructed is part of some minimal spanning tree.

Initially, this clearly holds, as starting vertex must be part of every minimal spanning tree.

Induction: Suppose $T$ is a minimal spanning tree which contains $T'$, the part of the tree already constructed before the iteration.

Suppose in the iteration, $v$ is added to the spanning tree, and its parent is $parent(v)$.
If $(v, parent(v))$ is an edge in $T$ then we are done.

Otherwise, consider $G' = T \cup \{(v, parent(v))\}$. This graph contains a loop, and $n$ edges, where $|V| = n$.
Furthermore, the loop must contain the edge $(v, parent(v))$, and some other edge which is not already in the tree $T'$, say $(w, w')$.

Now, if we delete $(w, w')$ from $G'$, we get a spanning tree $T''$.

Weight of this spanning tree is no more than weight of $T$ (as we added $(v, parent(v))$ and deleted $(w, w')$, the weight of former being no more than the weight of latter).
$T''$ is thus a minimal spanning tree, which contains $T'$ and $(v, parent(v))$.