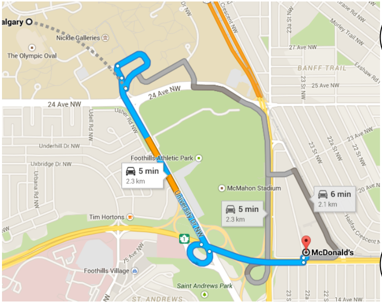
Dijkstra's Algorithm

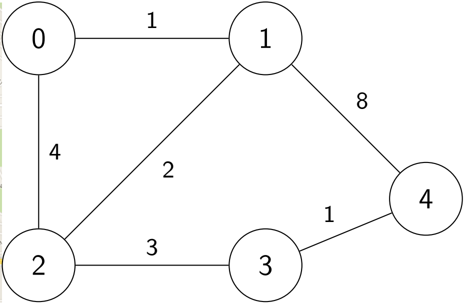
Problem Solving Club

February 1, 2017

Dijkstra's algorithm

- Dijkstra's is a **greedy** algorithm that finds **shortest paths** from a **single source** to **every other vertex** in the graph.
- As usually implemented:
  - Works for **weighted** graphs with **non-negative weights**.
  - Works for **directed** and **undirected** graphs.
  - Runs in $O((V + E) \log V)$.





Binary heap (priority queue) data structure

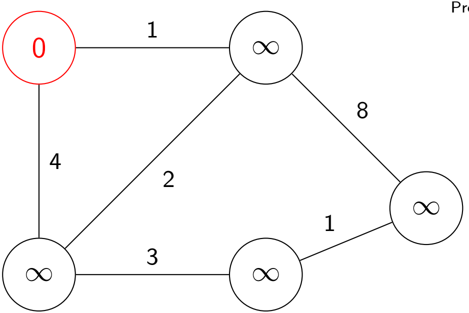
- A binary heap is a data structure with two operations:
  - **Insert:** Insert an element into the heap.
  - **Extract:** Remove the max (or min) element from the heap.
- Both operations take at worst $O(\log N)$.
- C++ std::priority_queue, Java PriorityQueue

Dijkstra's Algorithm

Procedure:

1. Assign all nodes a (tentative) distance of infinity.
2. Mark all nodes as unvisited.
3. Set the current node as start point and set its distance to zero.
4. For the current node, consider all neighbours. If [distance to current node + edge weight] is smaller than the current tentative distance of that node, update its tentative distance.
5. Mark the current node as visited.
6. Set the current node to the unvisited node with the smallest tentative distance, and go back to step 4 until there are no more unvisited nodes.

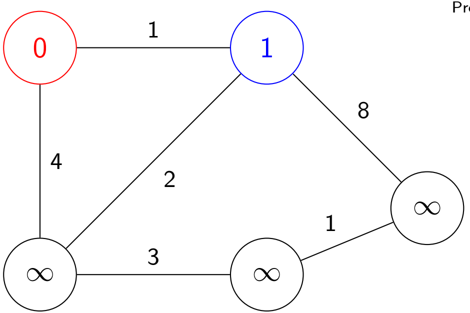
Dijkstra's Algorithm: An Example



Procedure:

1. Assign all nodes a (tentative) distance of infinity.
2. Mark all nodes as unvisited.
3. Set the current node as start point and set its distance to zero.
4. For the current node, consider all neighbours. If [distance to current node + edge weight] is smaller than the current tentative distance of that node, update its tentative distance.
5. Mark the current node as visited.
6. Set the current node to the unvisited node with the smallest tentative distance, and go back to step 4 until there are no more unvisited nodes.

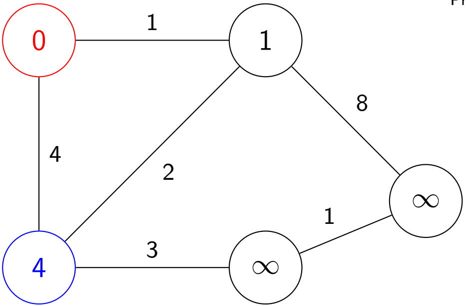
Dijkstra's Algorithm: An Example



Procedure:

1. Assign all nodes a (tentative) distance of infinity.
2. Mark all nodes as unvisited.
3. Set the current node as start point and set its distance to zero.
4. For the current node, consider all neighbours. If [distance to current node + edge weight] is smaller than the current tentative distance of that node, update its tentative distance.
5. Mark the current node as visited.
6. Set the current node to the unvisited node with the smallest tentative distance, and go back to step 4 until there are no more unvisited nodes.

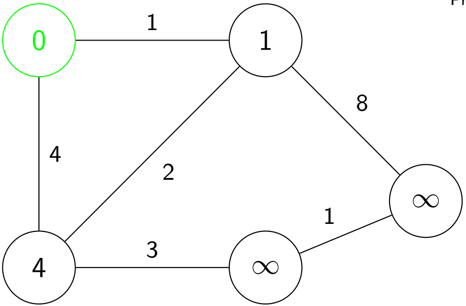
Dijkstra's Algorithm: An Example



Procedure:

1. Assign all nodes a (tentative) distance of infinity.
2. Mark all nodes as unvisited.
3. Set the current node as start point and set its distance to zero.
4. For the current node, consider all neighbours. If [distance to current node + edge weight] is smaller than the current tentative distance of that node, update its tentative distance.
5. Mark the current node as visited.
6. Set the current node to the unvisited node with the smallest tentative distance, and go back to step 4 until there are no more unvisited nodes.

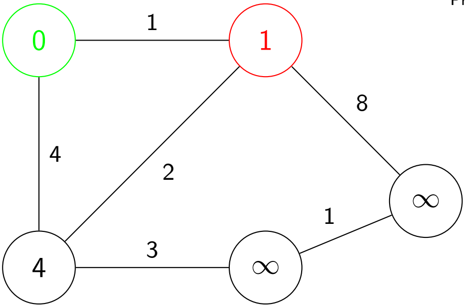
Dijkstra's Algorithm: An Example



Procedure:

1. Assign all nodes a (tentative) distance of infinity.
2. Mark all nodes as unvisited.
3. Set the current node as start point and set its distance to zero.
4. For the current node, consider all neighbours. If [distance to current node + edge weight] is smaller than the current tentative distance of that node, update its tentative distance.
5. Mark the current node as visited.
6. Set the current node to the unvisited node with the smallest tentative distance, and go back to step 4 until there are no more unvisited nodes.

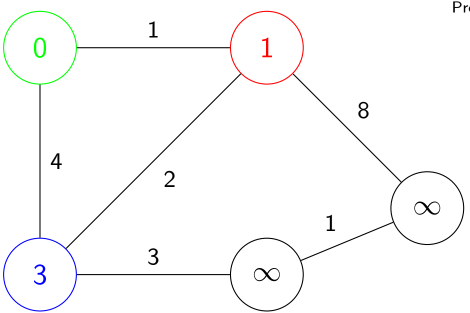
Dijkstra's Algorithm: An Example



Procedure:

1. Assign all nodes a (tentative) distance of infinity.
2. Mark all nodes as unvisited.
3. Set the current node as start point and set its distance to zero.
4. For the current node, consider all neighbours. If [distance to current node + edge weight] is smaller than the current tentative distance of that node, update its tentative distance.
5. Mark the current node as visited.
6. Set the current node to the unvisited node with the smallest tentative distance, and go back to step 4 until there are no more unvisited nodes.

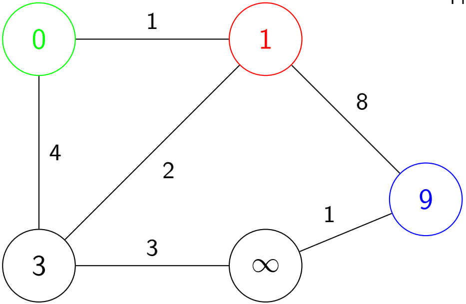
Dijkstra's Algorithm: An Example



Procedure:

1. Assign all nodes a (tentative) distance of infinity.
2. Mark all nodes as unvisited.
3. Set the current node as start point and set its distance to zero.
4. For the current node, consider all neighbours. If [distance to current node + edge weight] is smaller than the current tentative distance of that node, update its tentative distance.
5. Mark the current node as visited.
6. Set the current node to the unvisited node with the smallest tentative distance, and go back to step 4 until there are no more unvisited nodes.

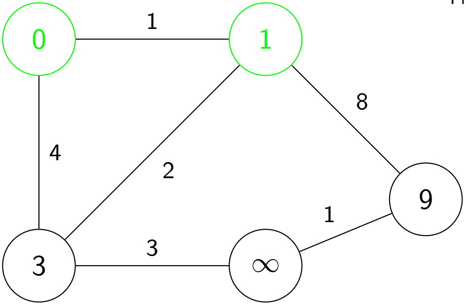
Dijkstra's Algorithm: An Example



Procedure:

1. Assign all nodes a (tentative) distance of infinity.
2. Mark all nodes as unvisited.
3. Set the current node as start point and set its distance to zero.
4. For the current node, consider all neighbours. If [distance to current node + edge weight] is smaller than the current tentative distance of that node, update its tentative distance.
5. Mark the current node as visited.
6. Set the current node to the unvisited node with the smallest tentative distance, and go back to step 4 until there are no more unvisited nodes.

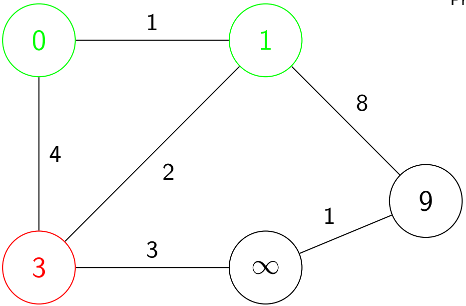
Dijkstra's Algorithm: An Example



Procedure:

1. Assign all nodes a (tentative) distance of infinity.
2. Mark all nodes as unvisited.
3. Set the current node as start point and set its distance to zero.
4. For the current node, consider all neighbours. If [distance to current node + edge weight] is smaller than the current tentative distance of that node, update its tentative distance.
5. Mark the current node as visited.
6. Set the current node to the unvisited node with the smallest tentative distance, and go back to step 4 until there are no more unvisited nodes.

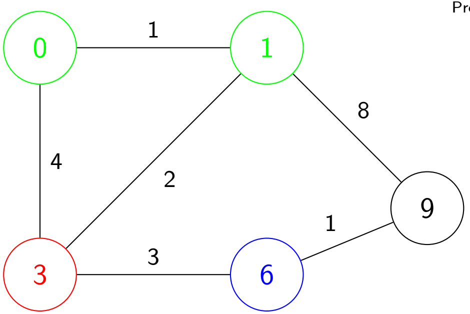
Dijkstra's Algorithm: An Example



Procedure:

1. Assign all nodes a (tentative) distance of infinity.
2. Mark all nodes as unvisited.
3. Set the current node as start point and set its distance to zero.
4. For the current node, consider all neighbours. If [distance to current node + edge weight] is smaller than the current tentative distance of that node, update its tentative distance.
5. Mark the current node as visited.
6. Set the current node to the unvisited node with the smallest tentative distance, and go back to step 4 until there are no more unvisited nodes.

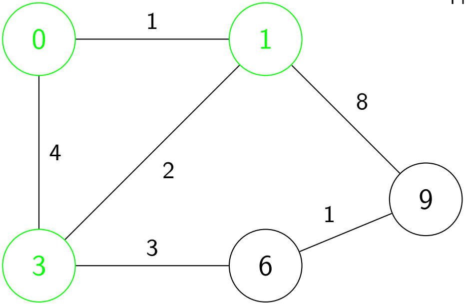
Dijkstra's Algorithm: An Example



Procedure:

1. Assign all nodes a (tentative) distance of infinity.
2. Mark all nodes as unvisited.
3. Set the current node as start point and set its distance to zero.
4. For the current node, consider all neighbours. If [distance to current node + edge weight] is smaller than the current tentative distance of that node, update its tentative distance.
5. Mark the current node as visited.
6. Set the current node to the unvisited node with the smallest tentative distance, and go back to step 4 until there are no more unvisited nodes.

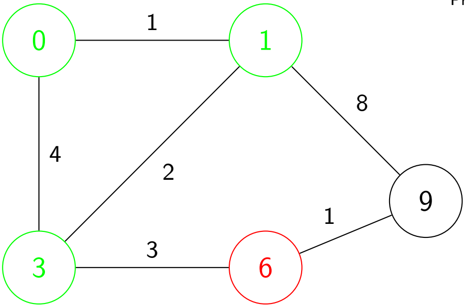
Dijkstra's Algorithm: An Example



Procedure:

1. Assign all nodes a (tentative) distance of infinity.
2. Mark all nodes as unvisited.
3. Set the current node as start point and set its distance to zero.
4. For the current node, consider all neighbours. If [distance to current node + edge weight] is smaller than the current tentative distance of that node, update its tentative distance.
5. Mark the current node as visited.
6. Set the current node to the unvisited node with the smallest tentative distance, and go back to step 4 until there are no more unvisited nodes.

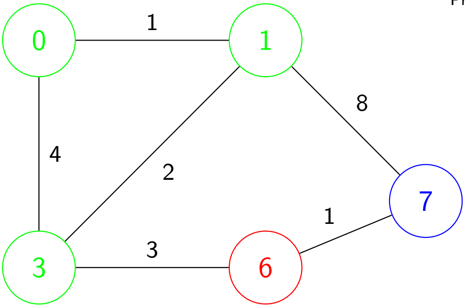
Dijkstra's Algorithm: An Example



Procedure:

1. Assign all nodes a (tentative) distance of infinity.
2. Mark all nodes as unvisited.
3. Set the current node as start point and set its distance to zero.
4. For the current node, consider all neighbours. If [distance to current node + edge weight] is smaller than the current tentative distance of that node, update its tentative distance.
5. Mark the current node as visited.
6. Set the current node to the unvisited node with the smallest tentative distance, and go back to step 4 until there are no more unvisited nodes.

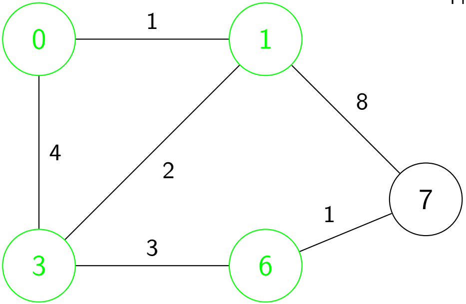
Dijkstra's Algorithm: An Example



Procedure:

1. Assign all nodes a (tentative) distance of infinity.
2. Mark all nodes as unvisited.
3. Set the current node as start point and set its distance to zero.
4. For the current node, consider all neighbours. If [distance to current node + edge weight] is smaller than the current tentative distance of that node, update its tentative distance.
5. Mark the current node as visited.
6. Set the current node to the unvisited node with the smallest tentative distance, and go back to step 4 until there are no more unvisited nodes.

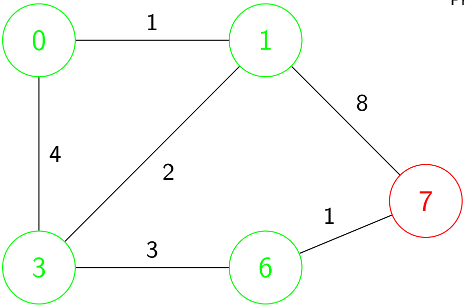
Dijkstra's Algorithm: An Example



Procedure:

1. Assign all nodes a (tentative) distance of infinity.
2. Mark all nodes as unvisited.
3. Set the current node as start point and set its distance to zero.
4. For the current node, consider all neighbours. If [distance to current node + edge weight] is smaller than the current tentative distance of that node, update its tentative distance.
5. Mark the current node as visited.
6. Set the current node to the unvisited node with the smallest tentative distance, and go back to step 4 until there are no more unvisited nodes.

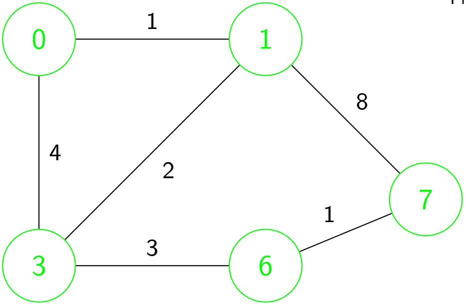
Dijkstra's Algorithm: An Example



Procedure:

1. Assign all nodes a (tentative) distance of infinity.
2. Mark all nodes as unvisited.
3. Set the current node as start point and set its distance to zero.
4. For the current node, consider all neighbours. If [distance to current node + edge weight] is smaller than the current tentative distance of that node, update its tentative distance.
5. Mark the current node as visited.
6. Set the current node to the unvisited node with the smallest tentative distance, and go back to step 4 until there are no more unvisited nodes.

Dijkstra's Algorithm: An Example



Procedure:

1. Assign all nodes a (tentative) distance of infinity.
2. Mark all nodes as unvisited.
3. Set the current node as start point and set its distance to zero.
4. For the current node, consider all neighbours. If [distance to current node + edge weight] is smaller than the current tentative distance of that node, update its tentative distance.
5. Mark the current node as visited.
6. Set the current node to the unvisited node with the smallest tentative distance, and go back to step 4 until there are no more unvisited nodes.

Example code

```
vector<edge> adj[100];
vector<int> dist(100, INF);

void dijkstra(int start) {
    dist[start] = 0;
    priority_queue<pair<int, int>,
                   vector<pair<int, int> >,
                   greater<pair<int, int> > > pq;
    pq.push(make_pair(dist[start], start));

    while (!pq.empty()) {
        int u = pq.top().second,
            d = pq.top().first;
        pq.pop();
        if (d > dist[u]) continue;
        for (int i = 0; i < adj[u].size(); i++) {
            int v = adj[u][i].v,
                w = adj[u][i].weight;
            if (w + dist[u] < dist[v]) {
                dist[v] = w + dist[u];
                pq.push(make_pair(dist[v], v));
            }
        }
    }
}
```

Frequently asked questions

- How do I find the actual shortest paths?
  - Keep track of each vertex's parent using a separate array.
- Can I use std::set or TreeSet instead of a binary heap?
  - Yes. Same asymptotic performance, but worse in practice.
- Can Dijkstra's find the longest path in a graph?
  - No. Longest path problem for general graph is NP-hard.
- What if my graph has negative weights?
  - Bellman-Ford / Shortest Path Faster Algorithm: $O(VE)$.
    - Same as Dijkstra's, but works with negative weights/cycles.
  - Floyd-Warshall: $O\left(V^3\right)$.
    - Finds the shortest path between every pair of vertices.
  - These have much worse running time than $O((V+E)\log V)$.