Shortest Paths

Section 4.4–4.5

Dr. Mayfield and Dr. Lam

Department of Computer Science
James Madison University

Nov 6, 2015

For SSSP on Weighted Graph but without Negative Weight Cycle

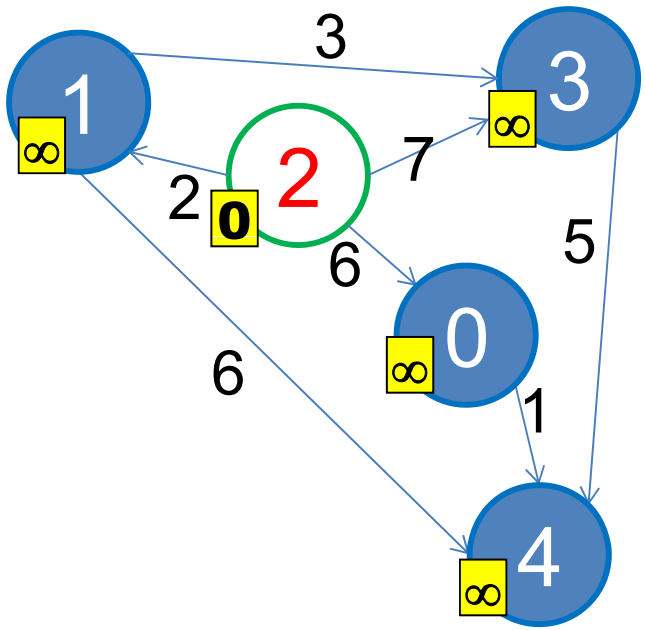
DIJKSTRA's

CS3233 - Competitive Programming,
Steven Halim, SoC, NUS

Single-Source Shortest Paths (1)

- If the graph is **un weighted**, we can use BFS
  - But what if the graph is **weighted**?
- UVa 341 (Non Stop Travel)
- Solution: Dijkstra O((V+E) log V)
  - A Greedy Algorithm
  - Use Priority Queue

Modified Dijkstra's – Example (1)

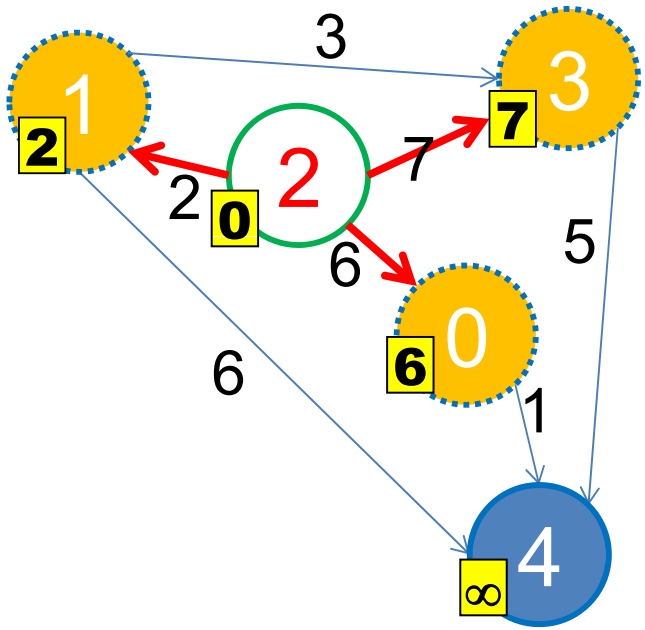


pq = {(0, 2)}

We store this pair of information to the priority queue: (D[vertex], vertex), sorted by increasing D[vertex], and then if ties, by vertex number

See that our priority queue is "clean" at the beginning of (modified) Dijsktra's algorithm, it only contains (0, the source s)

Modified Dijkstra's – Example (2)

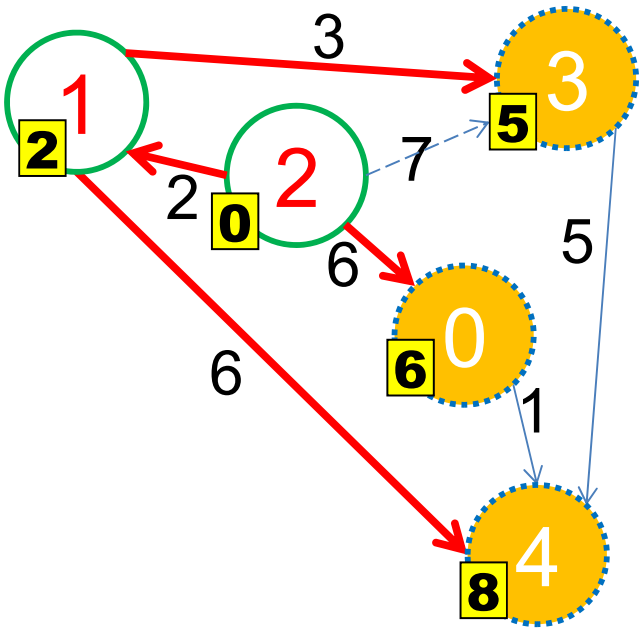


pq = {~~(0, 2)~~}
pq = {(2, 1), (6, 0), (7, 3)}

We greedily take the vertex in the front of the queue (here, it is vertex 2, the source), and then successfully relax all its neighbors (vertex 0, 1, 3).

Priority Queue will order these 3 vertices as 1, 0, 3, with shortest path estimate of 2, 6, 7, respectively.

Modified Dijkstra's – Example (3)



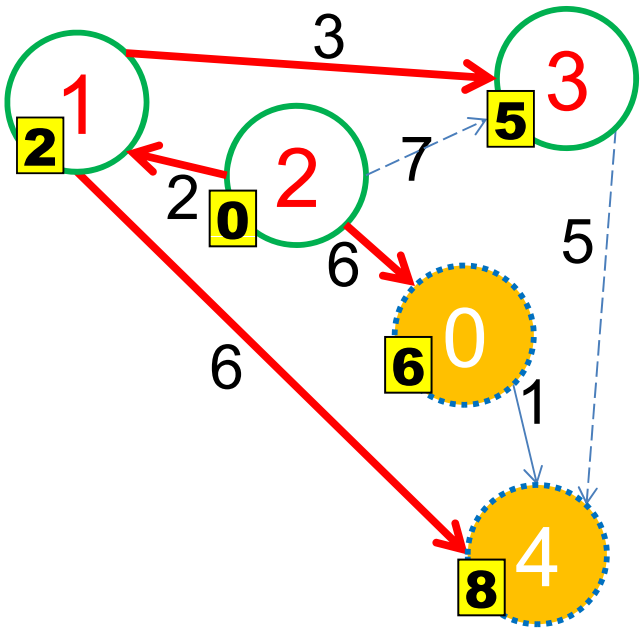
Vertex 3 appears twice in the priority queue, but this does not matter, as we will take only the first (smaller) one

pq = {~~(0, 2)~~}
pq = {~~(2, 1)~~, (6, 0), (7, 3)}
pq = {(5, 3), (6, 0), (7, 3), (8, 4)}

We greedily take the vertex in the front of the queue (now, it is vertex 1), then successfully relax all its neighbors (vertex 3 and 4).

Priority Queue will order the items as 3, 0, 3, 4 with shortest path estimate of 5, 6, 7, 8, respectively.

Modified Dijkstra's – Example (4)

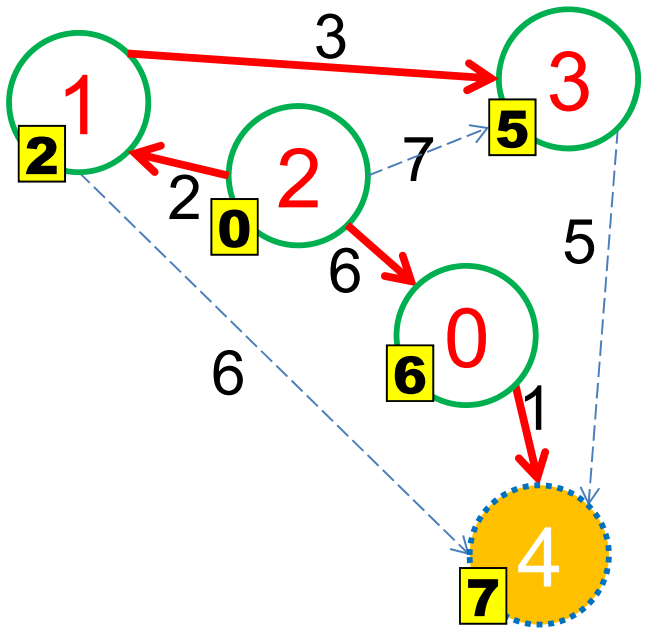


pq = {~~(0, 2)~~}
pq = {~~(2, 1)~~, (6, 0), (7, 3)}
pq = {~~(5, 3)~~, (6, 0), (7, 3), (8, 4)}
pq = {(6, 0), (7, 3), (8, 4)}

We greedily take the vertex in the front of the queue (now, it is vertex 3), then try to relax all its neighbors (only vertex 4). However D[4] is already 8. Since D[3] + w(3, 4) = 5 + 5 is worse than 8, we do not do anything.

Priority Queue will now have these items 0, 3, 4 with shortest path estimate of 6, 7, 8, respectively.

Modified Dijkstra's – Example (5)

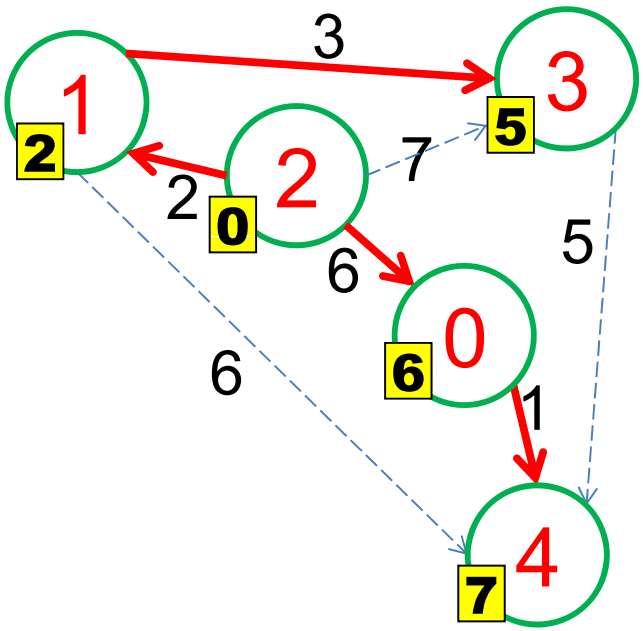


pq = {~~(0, 2)~~}
pq = {~~(2, 1)~~, (6, 0), (7, 3)}
pq = {~~(5, 3)~~, (6, 0), (7, 3), (8, 4)}
pq = {~~(6, 0)~~, (7, 3), (8, 4)}
pq = {(7, 3), (7, 4), (8, 4)}

We greedily take the vertex in the front of the queue (now, it is vertex 5), then successfully relax all its neighbors (only vertex 4).

Priority Queue will now have these items 3, 4, 4 with shortest path estimate of 7, 7, 8, respectively.

Modified Dijkstra's – Example (6)



Remember that vertex 3 appeared twice in the priority queue, but this Dijkstra's algorithm will only consider the first (shorter) one

pq = {~~(0, 2)~~}
pq = {~~(2, 1)~~, (6, 0), (7, 3)}
pq = {~~(5, 3)~~, (6, 0), (7, 3), (8, 4)}
pq = {~~(6, 0)~~, (7, 3), (8, 4)}
pq = {~~(7, 3)~~, (7, 4), (8, 4)}
pq = {~~(7, 4)~~, (8, 4)}
pq = {~~(8, 4)~~}
pq = {}

Similarly for vertex 4. The one with shortest path estimate 7 will be processed first and the one with shortest path estimate 8 will be ignored, although nothing is changed anymore

Dijkstra's Algorithm (using STL)

```
vi dist(V, INF); dist[s] = 0; // INF = 2B
priority_queue< ii, vector<ii>, greater<ii> > pq;
pq.push(ii(0, s)); // sort based on increasing distance
while (!pq.empty()) { // main loop
  ii top = pq.top(); pq.pop(); // greedy
  int d = top.first, u = top.second;
  if (d == dist[u]) {
    for (int j = 0; j < (int)AdjList[u].size(); j++) {
      ii  v = AdjList[u][j]; // all outgoing edges from u
      if (dist[u] + v.second < dist[v.first]) {
        dist[v.first] = dist[u] + v.second; // relax
        pq.push(ii(dist[v.first], v.first));
      } // enqueue this neighbor regardless it is
    }   // already in pq or not
  }
}
```

```
int[] dist = new int[adjList.length];
for (int i = 0; i < dist.length; i++) { dist[i] = 99999999; }
dist[src] = 0;
PriorityQueue<Edge> pq = new PriorityQueue<Edge>();
pq.add(new Edge(0, src));
while (pq.size() > 0) {
    Edge top = pq.poll();
    int u = top.dest;
    if (top.weight == dist[u]) {
        for (int j = 0; j < adjList[u].length; j++) {
            Edge v = adjList[u][j];
            if (dist[u] + v.weight < dist[v.dest]) {
                dist[v.dest] = dist[u] + v.weight;
                pq.add(new Edge(dist[v.dest], v.dest));
            }
        }
    }
}
```

For All-Pairs Shortest Paths

FLOYD WARSHALL's

CS3233 - Competitive Programming,
Steven Halim, SoC, NUS

UVa 11463 – Commandos (1)

Al-Khawarizmi, Malaysia National Contest 2008

- Given:
  - A table that stores the amount of minutes to travel between buildings (there are **at most 100** buildings)
  - 2 special buildings: startB and endB
  - K soldiers to bomb all the K buildings in this mission
  - Each of them start at the same time from startB, choose one building B that has not been bombed by other soldier (bombing time negligible), and then gather in (destroyed) building endB.
- What is the minimum time to complete the mission?

UVa 11463 – Commandos (2)

Al-Khawarizmi, Malaysia National Contest 2008

- How long do you need to solve this problem?
- Solution:
  - The answer is determined by sp from starting building, detonate **furthest building**, and sp from that furthest building to end building
    - max(dist[start][i] + dist[i][end]) for all i $\in$ V
- How to compute **sp** for **many** pairs of vertices?

UVa 11463 – Commandos (3)

Al-Khawarizmi, Malaysia National Contest 2008

- This problem is called: All-Pairs Shortest Paths
- Two options to solve this:
  - Call SSSP algorithms multiple times
    - Dijkstra O(V * (V+E) * log V), if E = V^2 → O(V^3 log V)
    - Bellman Ford O(V * V * E), if E = V^2 → O(V^4)
    - Slow to code
  - Use Floyd Warshall, a clever **DP** algorithm
    - O(V^3) algorithm
    - Very easy to code!
    - In this problem, V is <= 100, so Floyd Warshall is DOABLE!!

Floyd Warshall – Template

- $O(V^3)$ since we have three nested loops!
- Use adjacency matrix: `G[MAX_V][MAX_V];`
  - So that weight of edge(i, j) can be accessed in O(1)

```
for (int k = 0; k < V; k++)
  for (int i = 0; i < V; i++)
    for (int j = 0; j < V; j++)
      G[i][j] = min(G[i][j], G[i][k] + G[k][j]);
```

  - See more explanation of this three-liner DP algorithm in CP