Final Exam Review
Day 2

CSCI 3100

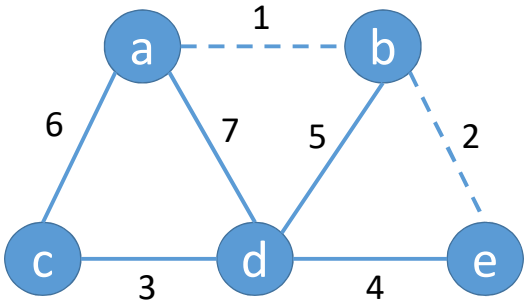
Prim's Algorithm

```
1. MST_PRIM(V, E)
2. V' = empty, E' = E
3. Select u from V, add u to V'
4. While |V'| < |V|
5.     select smallest edge (x,y) crossing the
cut between |V'| and |V-V'| (x is in V')
6.     add y to V'
7.     add (x, y) to E'
8. return E'
```

If we use binary heap to implement min priority queue for Prim's algorithm, what is the complexity of finding the next edge to include in the MST (without fixing up the heap)?

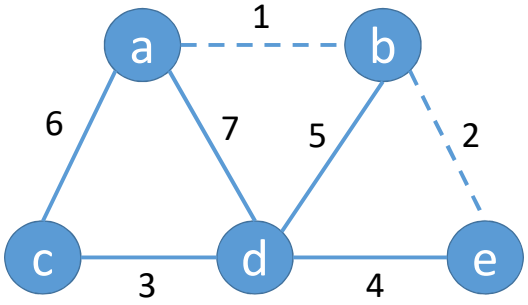
A. O(lg n)
B. O(n)
C. O(1)
D. O(n lg n)
E. $O(n^2)$

Consider the graph below. We are building a Minimum Spanning Tree of this graph. Dashed edges have already been selected by the Kruskal's algorithm to be included in the MST. What is the **next** edge the algorithm will select?



A. (e, d)
B. (a, c)
C. (a, d)
D. (b, d)
E. (c, d)

Consider the graph below. We are building a Minimum Spanning Tree of this graph. Dashed edges have already been selected by the Prim's algorithm to be included in the MST. What is the **next** edge the algorithm will select?



A. (e, d)
B. (a, c)
C. (a, d)
D. (b, d)
E. (c, d)

Shortest Path Algorithms

Variations of the problem

- Single Source shortest path
  - Shortest paths from a given source vertex to all other vertices
- Single Destination shortest path
  - Shortest paths from all vertices in the graph to a given destination
- All pairs shortest path
  - Shortest paths between all pairs of vertices

Single Source Shortest Path

- Bellman-Ford Algorithm
- Dijkstra's algorithm
- Relaxation approach:
  - Estimate the path length from the source to each vertex to be infinity
  - Update the estimate:
    - Given an edge (u, v) and an estimate d[v], we can update estimate d[u]
    - If d[u] > d[v] + w(v, u) => d[u] = d[v] + w(v, u). Predecessor of u becomes v

Relaxation Properties

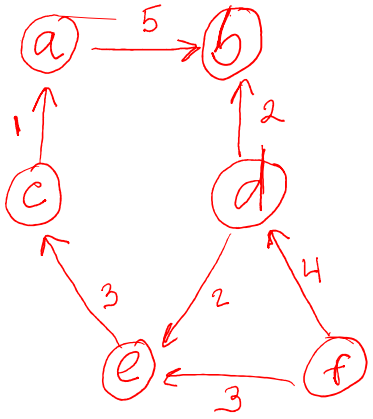
- Define δ(s, u) be the weight of a shortest path from s to u.
- **Triangle inequality**:
  For any edge (u, v), δ(s, v) ≤ δ(s, u) + w (u, v)
- **No-path property:**
  If there is no path from s to v, then δ(s, v) = ∞
- **Path relaxation property**
  - If p=<v_1, v_2, …, v_k> is a shortest path from v_1 to v_k, and we relax the edges of p in order (v_1, v_2), (v_2, v_3), …, (v_{k-1}, v_k), then v_k.d = δ(v_1, v_k)
  - This property holds even if other relaxations are intermixed with the relaxation of edges of p
- **Convergence property**
  - IF p is the shortest path from s to v using edge (u, v)
  - AND d[u] – shortest distance from s to u prior to relaxing edge (u, v)
  - THEN d[v] – shortest distance from s to v after relaxing edge (u, v)

Bellman-Ford Algorithm

- Line 1
- Lines 2 – 4
- Lines 5 - 7

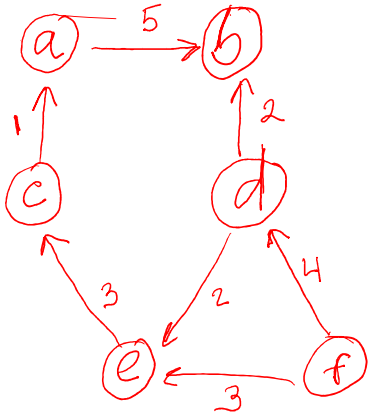
```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i = 1 to |G.V| − 1
3      for each edge (u, v) ∈ G.E
4          RELAX(u, v, w)
5  for each edge (u, v) ∈ G.E
6      if v.d > u.d + w(u, v)
7          return FALSE
8  return TRUE
```

Bellman-Ford example



Start: f

$d[f] = 0$
$d[a] = \infty$
$d[b] = \infty$
$d[c] = \infty$
$d[d] = \infty$
$d[e] = \infty$



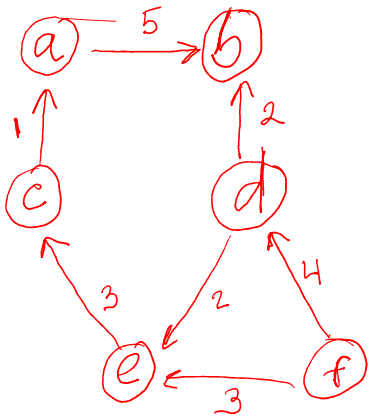
Start: f

$d[f] = 0$
$d[a] = \infty$
$d[b] = \infty$
$d[c] = \infty$
$d[d] = \infty$
$d[e] = \infty$

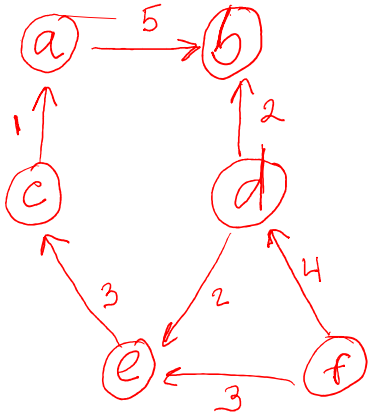
Iteration 1

Relax

(a, b): $d[b] = \infty$
(c, a): $d[a] = \infty$
(d, b): $d[b] = \infty$
(e, c): $d[c] = \infty$
(d, e): $d[e] = \infty$
(f, d): $d[d] = 4$
(f, e): $d[e] = 3$



$d[a] = \infty$
$d[b] = \infty$
$d[c] = \infty$
$d[d] = 4$
$d[e] = 3$
$d[f] = 0$

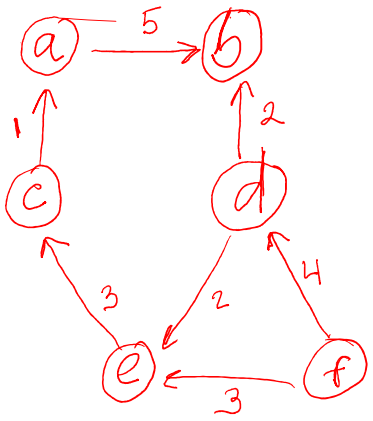


$d[a] = \infty$
$d[b] = \infty$
$d[c] = \infty$
$d[d] = 4$
$d[e] = 3$
$d[f] = 0$

Iteration 2

Relax

(a, b): $d[b] = \infty$
(c, a): $d[a] = \infty$
(d, b): $d[b] = 6$
(e, c): $d[c] = 6$
(d, e): $d[e] = 3$
(f, d): $d[d] = 4$
(f, e): $d[e] = 3$



$d[a] = \infty$
$d[b] = 6$
$d[c] = 6$
$d[d] = 4$
$d[e] = 3$
$d[f] = 0$

What will be the value of d[a] after the next iteration of relaxing all edges?

A. 7
B. 5
C. 1
D. 6
E. ∞

Dijkstra's Algorithm

- No negative weights on edges
- Keep a set of "Final" vertices
- Once a vertex is in the "Final" set, its estimate is the value of the shortest path
- The rest of the vertices are in a min-queue, based on the current estimate
- Continue until all vertices are in the "Final" set
  - Move the vertex at the front of the min-queue to the "Final" set
  - Relax all edges adjacent to the vertex you just moved

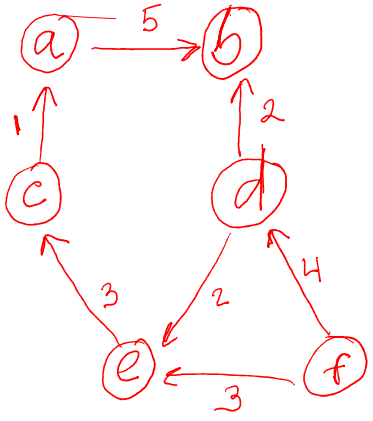
Dijkstra's Algorithm: Shortest path in G, where G has non-negative edge weights

```
Dijkstra(G, s)
   for each v ∈ V
      d[v] = ∞;
      v.π = NULL
   d[s] = 0; S = ∅; Q = V;
   while (Q ≠ ∅)
      u = ExtractMin(Q);
      S = S U {u};
      for each v ∈ u->Adj[]
         if (d[v] > d[u]+w(u,v))
            d[v] = d[u]+w(u,v);
            v.π = u;
```

Relaxation Step

Complexity:

Dijkstra's example



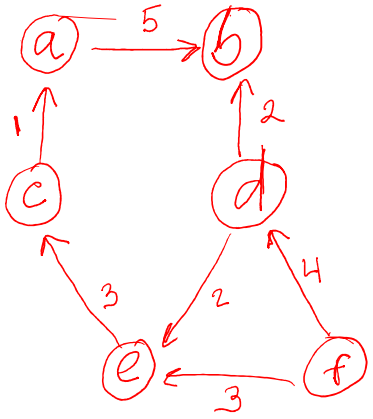
Start: f

d[f] = 0
d[a] = ∞
d[b] = ∞
d[c] = ∞
d[d] = ∞
d[e] = ∞

Move f to "Final" set

Relax (f,d), (f,e)
d[d] = 4
d[e] = 3

Dijkstra's example



d[e] = 3
d[d] = 4
d[a] = ∞
d[b] = ∞
d[c] = ∞

Final set
{ f }

Move e to Final set

Relax (e, c)

What vertex will be moved to the "Final" set next?

A. a
B. b
C. c
D. d

Shortest Path in a DAG

- If DAG contains a path from vertex *v* to vertex *u* then
  - *v* precedes *u* in a topological sort
- If we make one path over the vertices in the topological order
- What is the running time of this algorithm?

```
DAG-SHORTEST-PATHS(G, w, s)
1  topologically sort the vertices of G
2  INITIALIZE-SINGLE-SOURCE(G, s)
3  for each vertex u, taken in topologically sorted order
4      for each vertex v ∈ G.Adj[u]
5          RELAX(u, v, w)
```

A. O(|V| + |E|)
B. O($|V|^2$)
C. O($|E|^2$)
D. O(|V| * |E|)