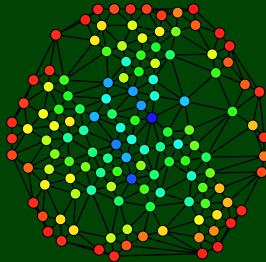
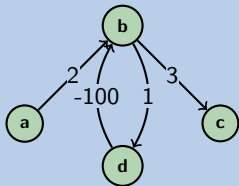
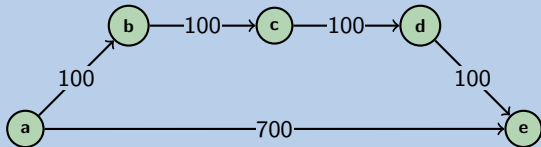


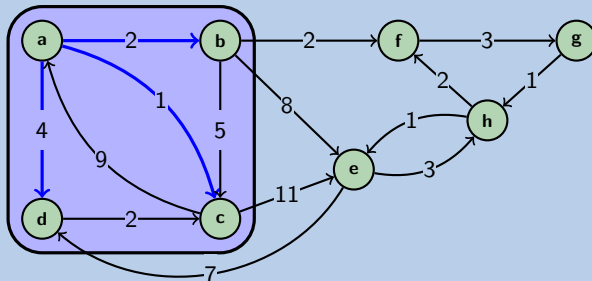
CSE 332

Data Structures and Parallelism

Graphs 3: Single-Source Shortest Paths







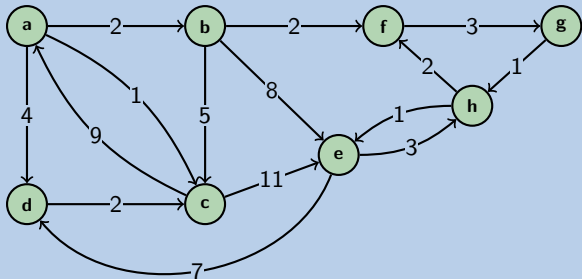
We will run a **simulation** of (infinitely many) ants exploring the graph.

The ants all move at identical speeds.

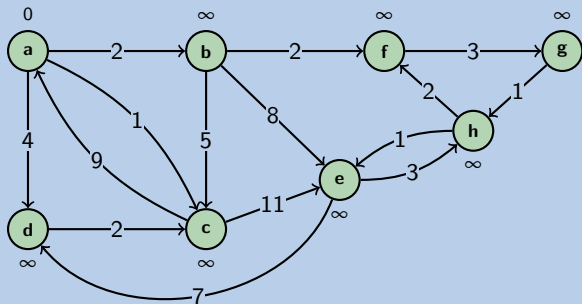
We're interested in the **time step** that some ant first reaches each vertex.

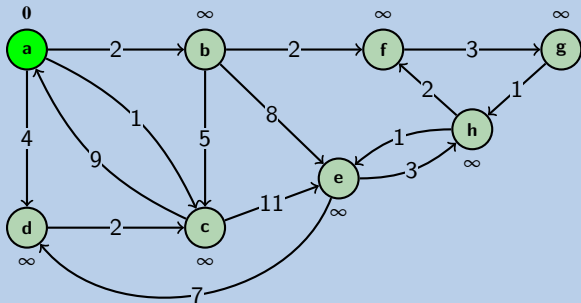
- At each step...
 - The ants try to move along some new edge
 - We “process” a vertex at the timestep that an ant arrives there
 - When an ant arrives, they dispatch new ants to every out-edge
- We're done!

worklist ←

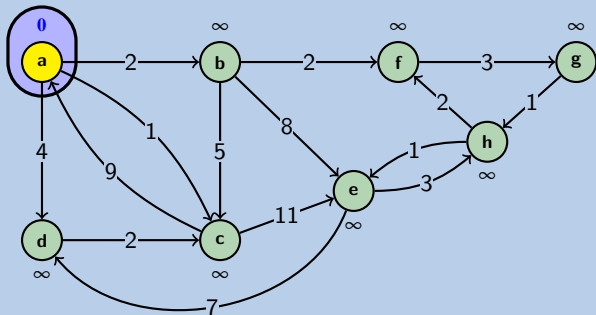


worklist ←



worklist ← $a \leq 0$ ←

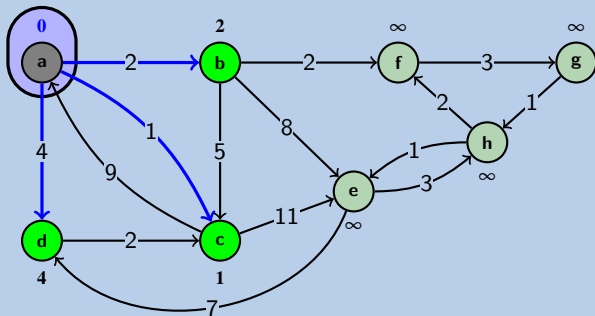
worklist ←



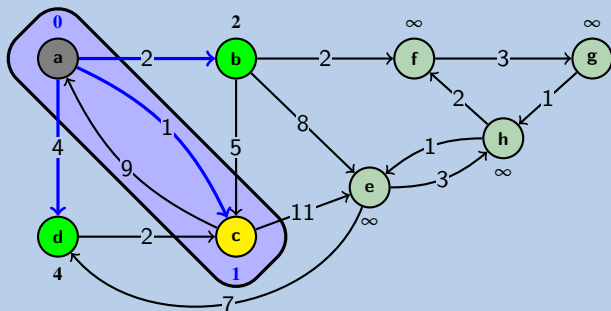
worklist ←

$c \leq 1$	$b \leq 2$	$d \leq 4$
------------	------------	------------

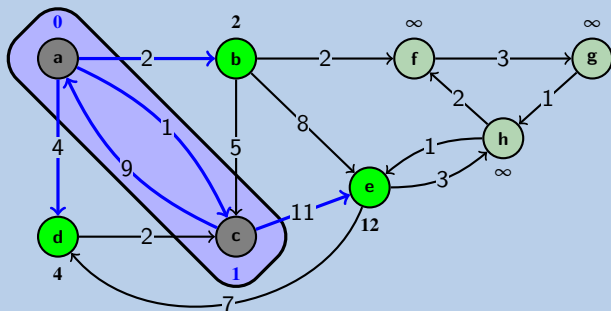
 ←



worklist ← $b \leq 2$ $d \leq 4$ ←

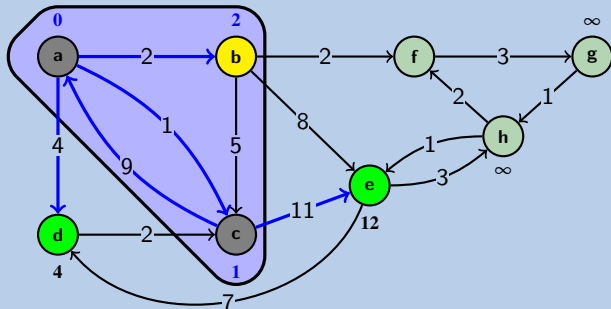


worklist ← $b \leq 2$ | $d \leq 4$ | $e \leq 12$ ←



worklist ←

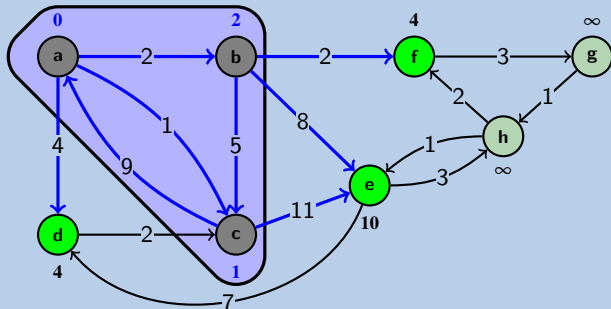
$d \leq 4$	$e \leq 12$
------------	-------------

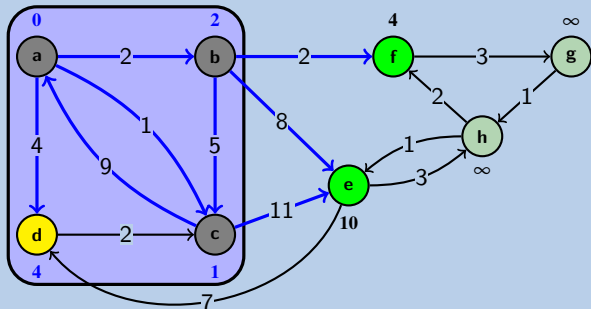
 ←

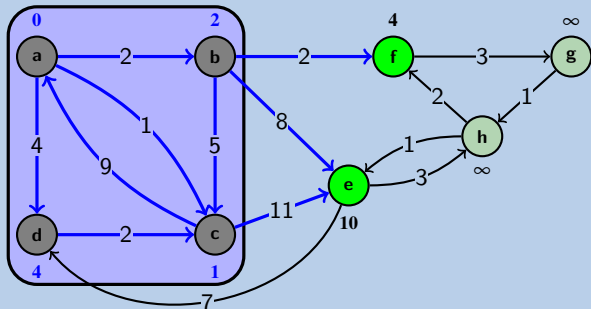
worklist ←

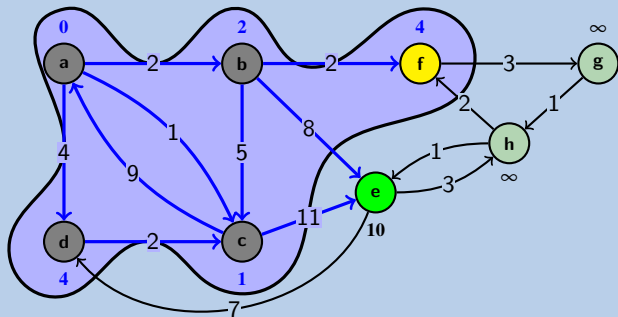
$d \leq 4$	$f \leq 4$	$e \leq 10$
------------	------------	-------------

 ←

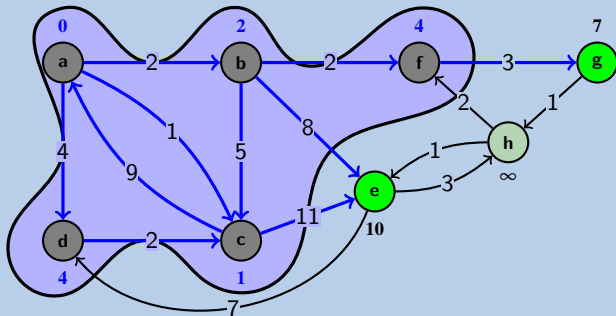


worklist ← $f \leq 4$ $e \leq 10$ ←

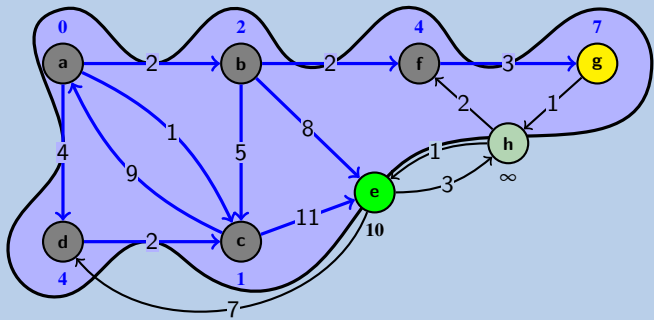
worklist ← $f \leq 4$ $e \leq 10$ ←

worklist ← $e \leq 10$ ←

worklist ← g ≤ 7 e ≤ 10 ←



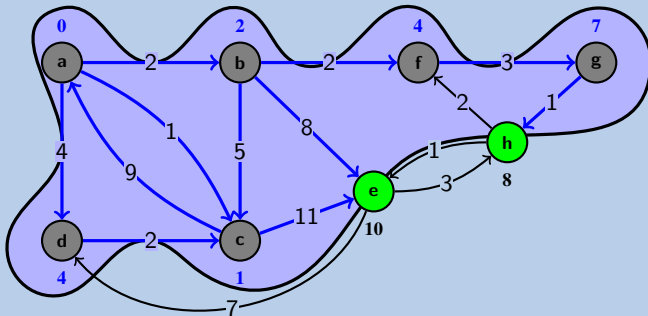
worklist ← e ≤ 10 ←



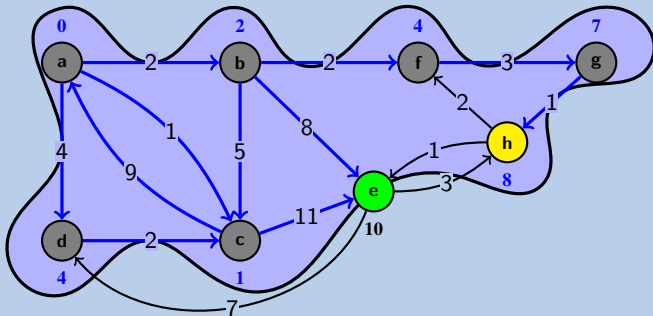
worklist ←

$h \leq 8$	$e \leq 10$
------------	-------------

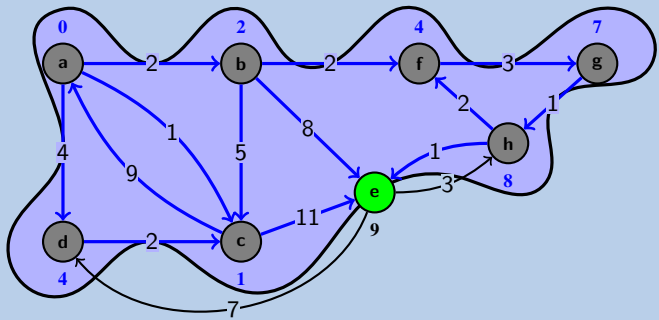
 ←



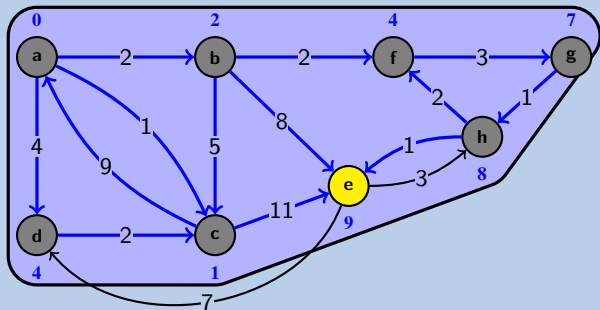
worklist ← e ≤ 10 ←



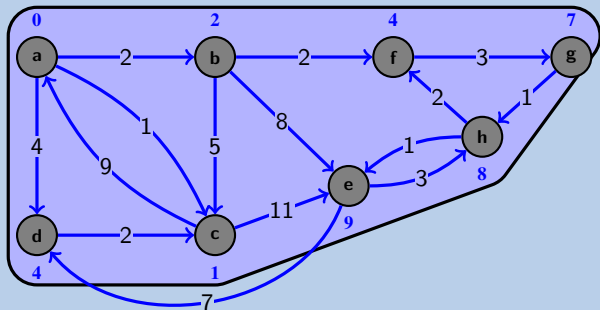
worklist ← e ≤ 9 ←



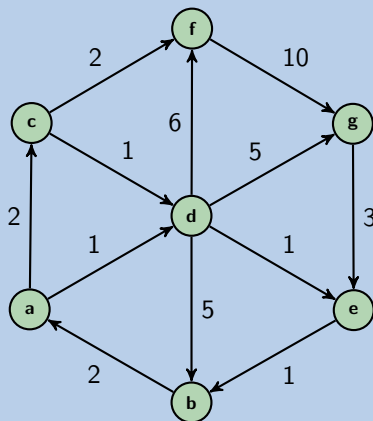
worklist ←



worklist ←



```
1  dijkstra(G, source) {
2    // We will use a "sorted list" as our worklist, because the items
3    // in the work list are "events" which are processed in order
4
5    // (v, timestep) in worklist, where v is a vertex and timestep
6    // is the "time" the first ant got there
7    worklist = [];    // These ants are "currently moving"
8
9    // All the ants begin at vertex v at time step zero
10   worklist.add((source, 0));
11
12   while (worklist.hasWork()) {
13     (v, time_to_v) = next();
14
15     // Since a cluster of ants got to v, we dispatch new ants
16     for (u : v.neighbors()) {
17       // When does a cluster of ants get to u? How does it change?
18       (u, time_to_u) = worklist.get(u);
19       // w(v, u) is the edge weight from v to u
20       time_from_v_to_u = w(v, u);
21       to_u = min(time_to_u, time_to_v + time_from_v_to_u);
22       worklist.add((u, to_u));
23     }
24   }
25   return dist;
26 }
```

- Our sorted list is slow; so, replace it with a **priority queue**.
- We need a way of “changing the priority of an element”

Remember, decreaseKey? That’s exactly what it does!

To make that work, we need to store a reference to the index/vertex in some dictionary.

```
1  dijkstra(G, source) {
2      dist = new Dictionary();
3      worklist = [];
4      for (v : V) {
5          if (v == source) { dist[v] = 0; }
6          else                { dist[v] = ∞; }
7          worklist.add((v, dist[v]));
8      }
9
10     while (worklist.hasWork()) {
11         v = next();
12         for (u : v.neighbors()) {
13             dist[u] = min(dist[u], dist[v] + w(v, u));
14             worklist.decreaseKey(u, dist[u]);
15         }
16     }
17
18     return dist;
19 }
```

