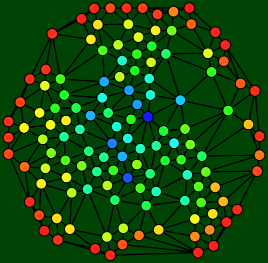
CSE 332

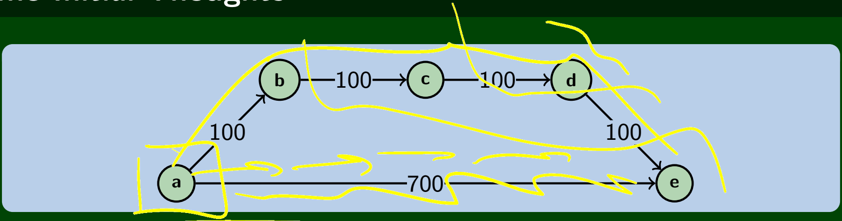
Data Abstractions

Graphs 3: Single-Source Shortest Paths

"ish"



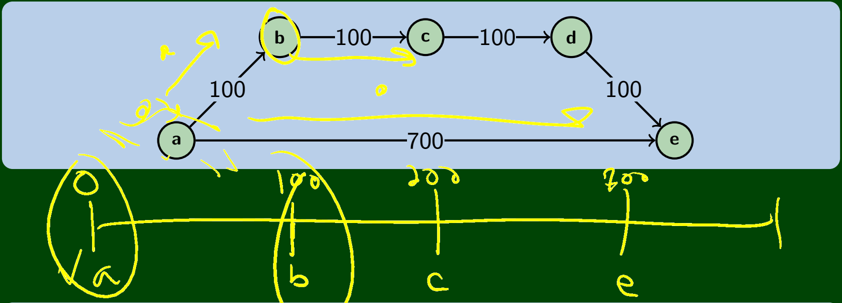
Some Initial Thoughts

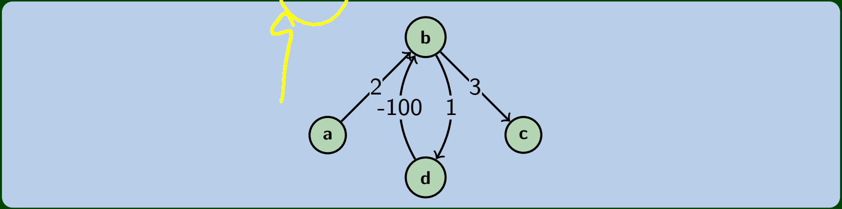


{a: 0, b: 1, c: 2, d: 3, e: 1}

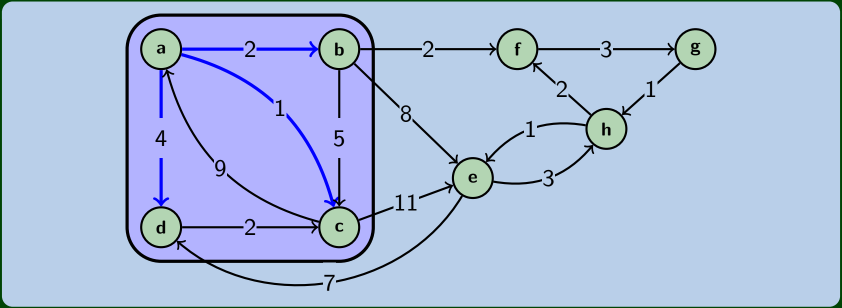
```
Shortest(source){
    len = new Dict();
    do a bfs from source
    keep track of lengths as we go
    return len;
}
```

Some Initial Thoughts





The Idea



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

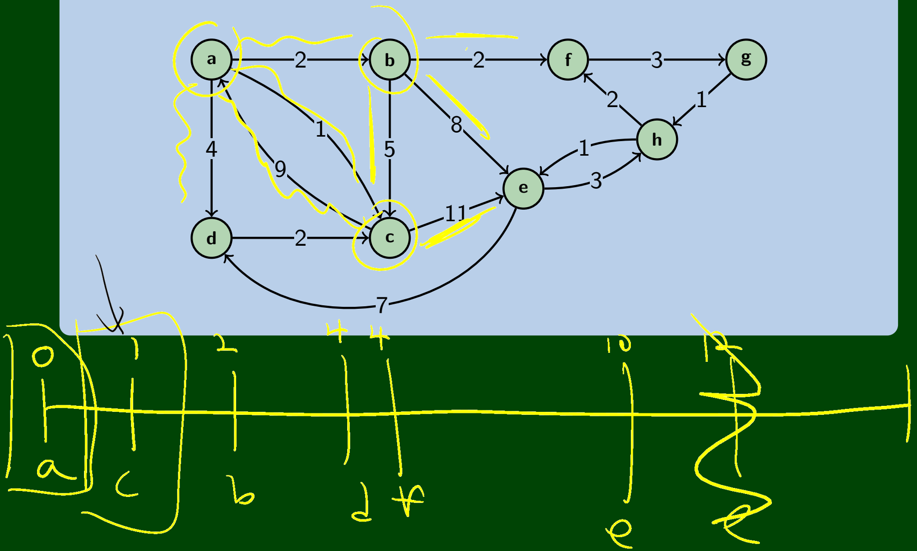
The ants all move at identical speeds.

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!

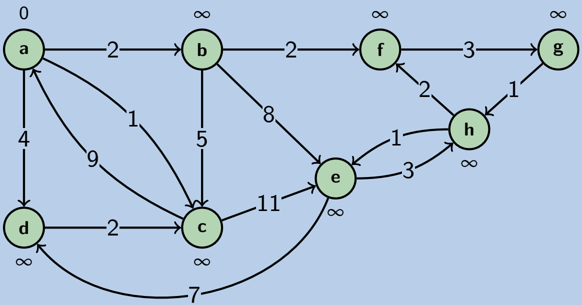
Example

worklist ←



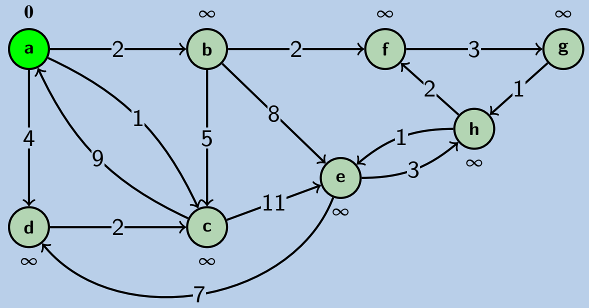
Example

worklist ←



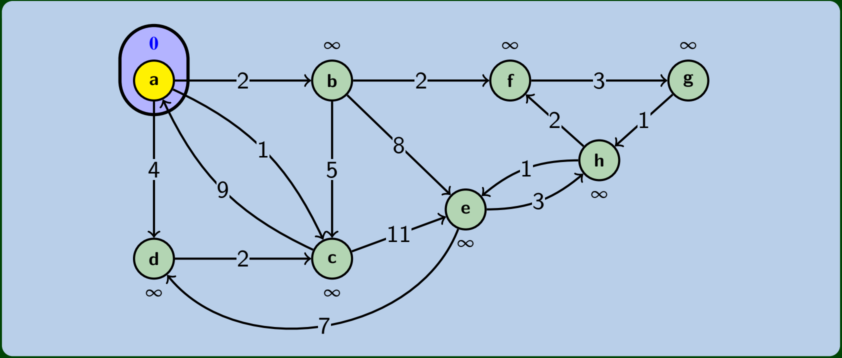
Example

worklist $\leftarrow \boxed{\text{a } \leq 0} \leftarrow$



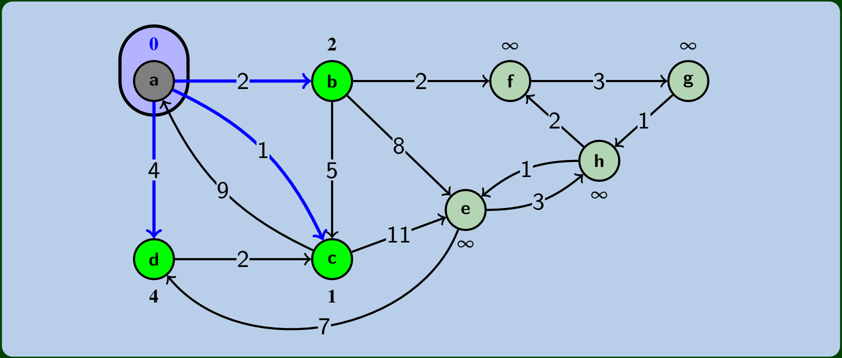
Example

worklist $\leftarrow$



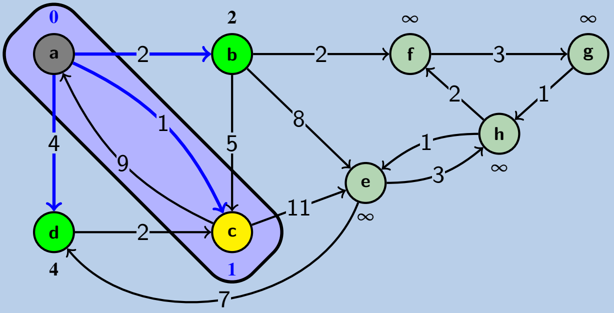
Example

worklist $\leftarrow$ | c ≤ 1 | b ≤ 2 | d ≤ 4 | $\leftarrow$



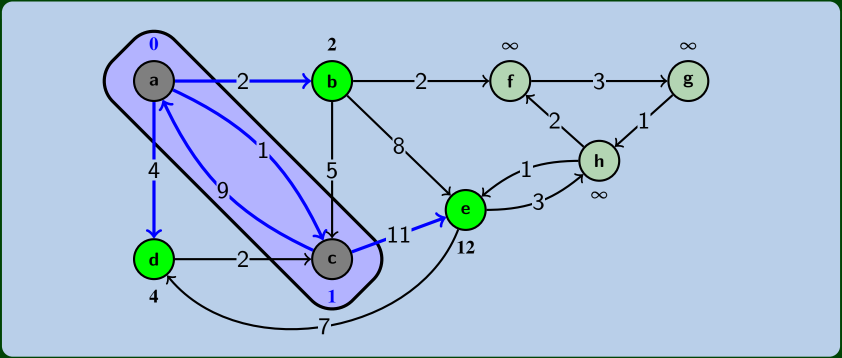
Example

worklist ← | b ≤ 2 | d ≤ 4 | ←



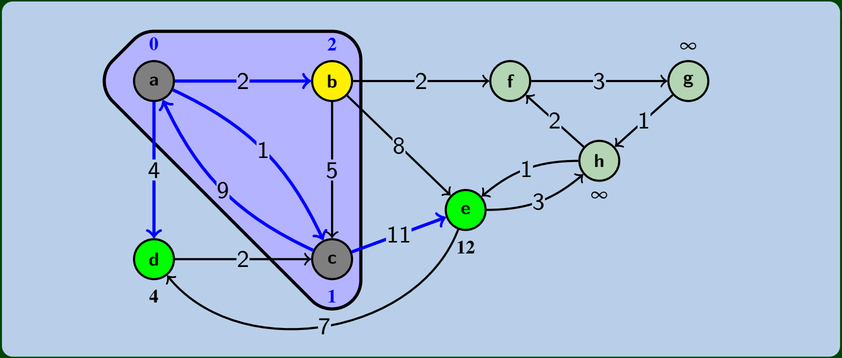
Example

worklist ← | b ≤ 2 | d ≤ 4 | e ≤ 12 | ←



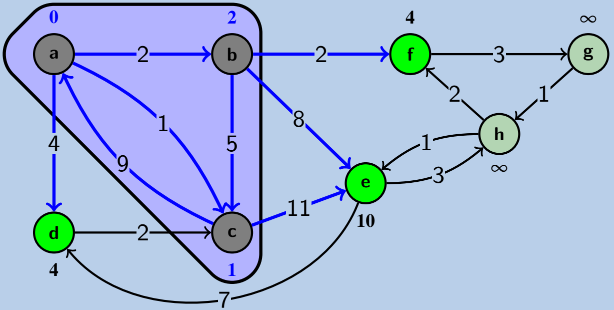
Example

worklist $\leftarrow$ | d ≤ 4 | e ≤ 12 | $\leftarrow$



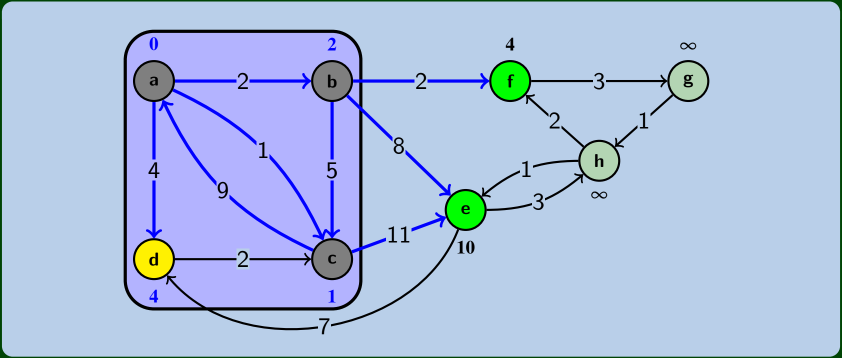
Example

worklist $\leftarrow$ | d ≤ 4 | f ≤ 4 | e ≤ 10 | $\leftarrow$



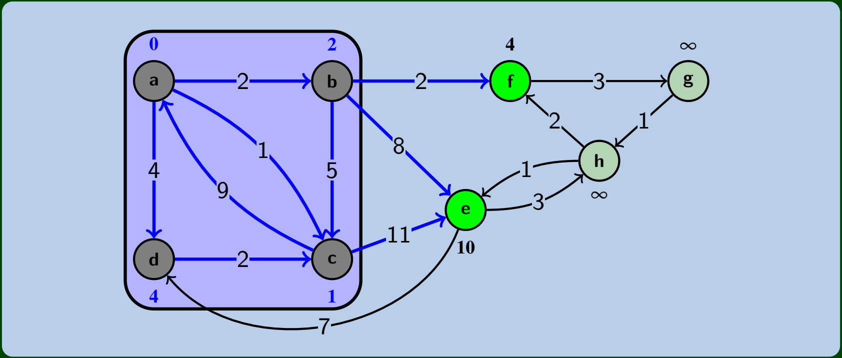
Example

worklist ← $\boxed{f \le 4}\boxed{e \le 10}$ ←



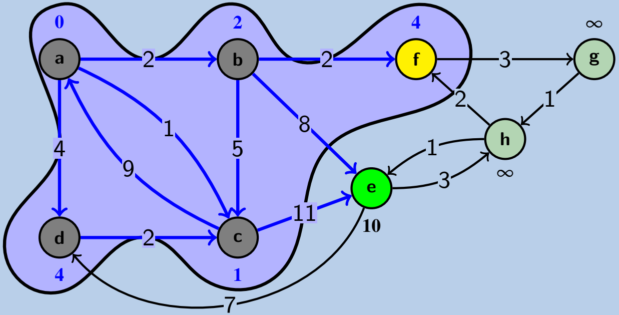
Example

worklist $\leftarrow \boxed{f \le 4} \boxed{e \le 10} \leftarrow$



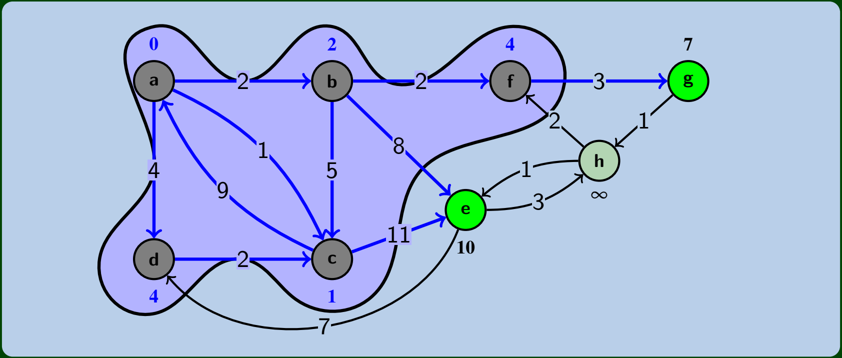
Example

worklist $\leftarrow \boxed{\text{e} \leq 10} \leftarrow$



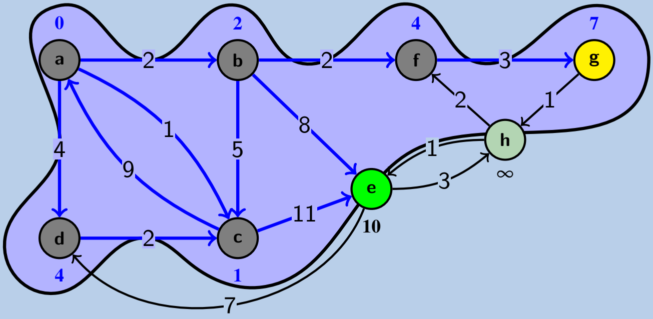
Example

worklist $\leftarrow$ | g ≤ 7 | e ≤ 10 | $\leftarrow$



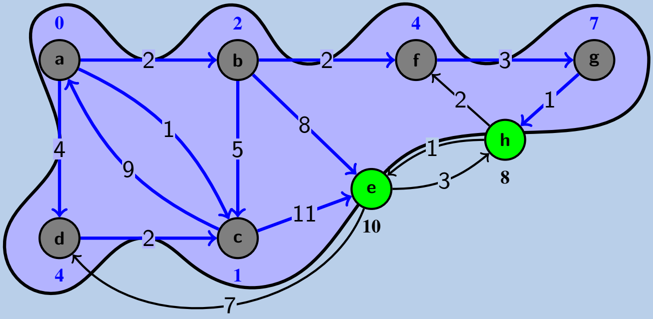
Example

worklist $\leftarrow \boxed{e \leq 10} \leftarrow$



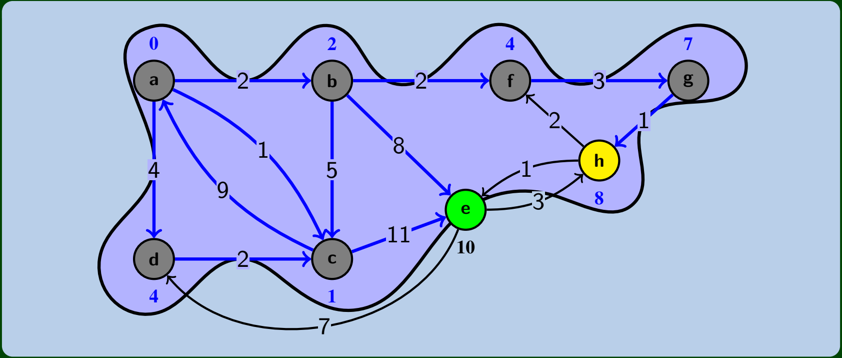
Example

worklist ← $\boxed{h \le 8}\boxed{e \le 10}$ ←



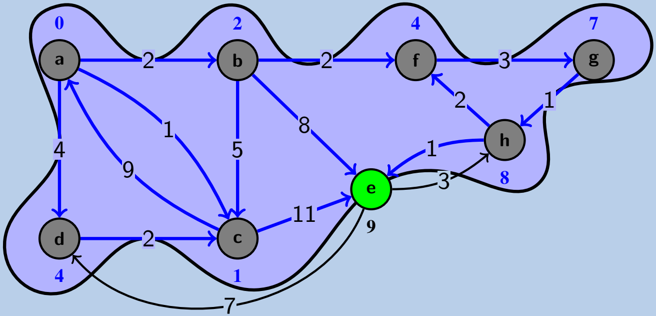
Example

worklist ← $\boxed{\text{e} \leq 10}$ ←



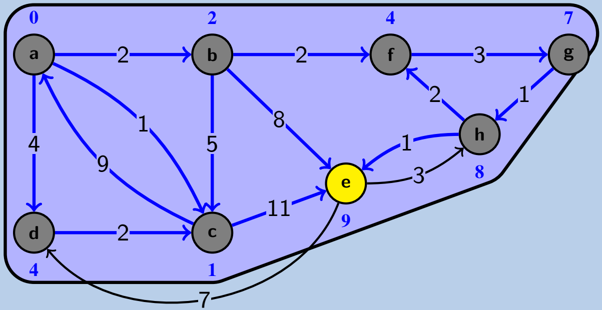
Example

worklist $\leftarrow \boxed{\text{e} \leq 9} \leftarrow$



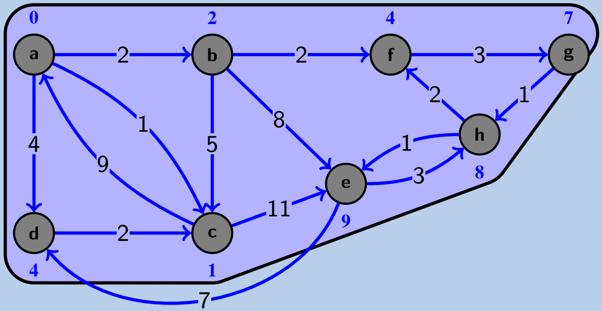
Example

worklist ←



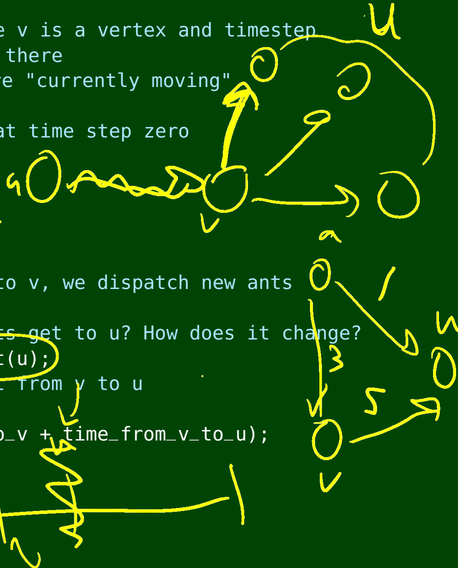
Example

worklist ←

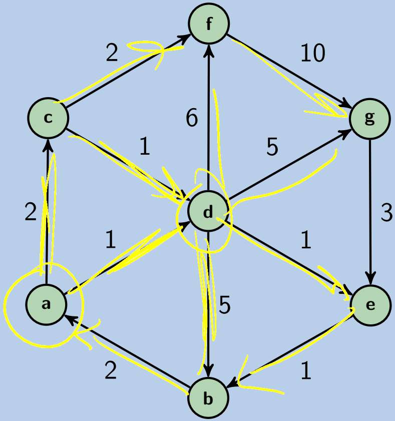


The Algorithm

```
dijkstra(G, source) {
  // We will use a "sorted list" as our worklist, because the items
  // in the work list are "events" which are processed in order

  // (v, timestep) in worklist, where v is a vertex and timestep
  // is the "time" the first ant got there
  worklist = [];    // These ants are "currently moving"

  // All the ants begin at vertex v at time step zero
  worklist.add((source, 0));

  while (worklist.hasWork()) {
    (v, time_to_v) = next();

    // Since a cluster of ants got to v, we dispatch new ants
    for (u : v.neighbors()) {
      // When does a cluster of ants get to u? How does it change?
      (u, time_to_u) = worklist.get(u);
      // w(v, u) is the edge weight from v to u
      time_from_v_to_u = w(v, u);
      to_u = min(time_to_u, time_to_v + time_from_v_to_u);
      worklist.add((u, to_u));
    }
  }
  ~~return~~ dist;
}
```



Example 2



a	b	c	d	e	f	g
2	3	2	1	2	4	6

Okay, and to implement this?

- Our sorted list is slow; so, replace it with a **priority queue**.

Okay, and to implement this?

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

Okay, and to implement this?

- 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.

Final Dijkstra's Algorithm

```
dijkstra(G, source) {
   dist = new Dictionary();
   worklist = [];
   for (v : V) {
      if (v == source) { dist[v] = 0; }
      else             { dist[v] = ∞; }
      worklist.add((v, dist[v]));
   }

   while (worklist.hasWork()) {
      v = next();
      for (u : v.neighbors()) {
         dist[u] = min(dist[u], dist[v] + w(v, u));
         worklist.decreaseKey(u, dist[u]);
      }
   }

   return dist;
}
```

$\mathcal{O}(|V|)$

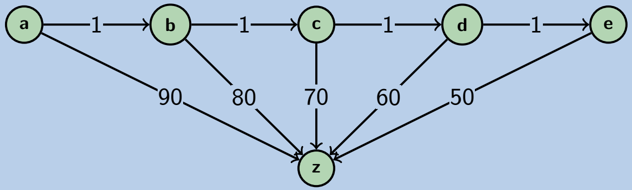
$\mathcal{O}(|V| \lg |V|)$

$\lg |V|$

$\mathcal{O}(|E| \lg |V|)$

$\mathcal{O}(\lg |V|)$

Example 3


a
1
b
1
c
1
d
1
e
90
80
70
60
50
z