



# CSE 332: Data Structures and Parallelism

Spring 2022

Richard Anderson

Lecture 24: Dijkstra's Algorithm

## Announcements

- Upcoming lectures
  - Intro to graphs
  - Topological Sort
  - Graph Algorithms
    - Graph Traversal
    - Shortest Paths
    - Minimum Spanning Tree
  - Union-Find Data Structure
  - Theory of NP-Completeness (2 lectures)

## Graph Theory

- $G = (V, E)$ 
  - $V$ : vertices,  $|V| = n$
  - $E$ : edges,  $|E| = m$
- Undirected graphs
  - Edges sets of two vertices  $\{u, v\}$
- Directed graphs
  - Edges ordered pairs  $(u, v)$
- Many other flavors
  - Edge / vertices weights
  - Parallel edges
  - Self loops
- Path:  $v_1, v_2, \dots, v_k$ , with  $(v_i, v_{i+1})$  in  $E$ 
  - Simple Path
  - Cycle
  - Simple Cycle
- Neighborhood
  - $N(v)$
- Distance
- Connectivity
  - Undirected
  - Directed (strong connectivity)
- Trees
  - Rooted
  - Unrooted

## Graph Representation

$V = \{a, b, c, d\}$   
 $E = \{(a, b), \{a, c\}, \{a, d\}, \{b, d\}\}$

Adjacency List

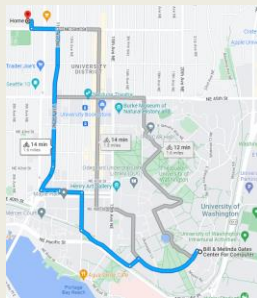
$O(n + m)$  space

	1	1	1
1		0	1
1	0		0
1	1	0	

Adjacency Matrix

$O(n^2)$  space

## Find the shortest path



## The Shortest Path Problem

Given a graph  $G$ , and vertices  $s$  and  $t$  in  $G$ , find the shortest path from  $s$  to  $t$ .

Two cases: weighted and unweighted.

For a path  $p = v_0 v_1 v_2 \dots v_k$

– unweighted length of path  $p = k$  (a.k.a. length)

– weighted length of path  $p = \sum_{i=0..k-1} c_{i,i+1}$  (a.k.a. cost)

We will assume the graph is directed

## Single Source Shortest Paths (SSSP)

Given a graph  $G$  and vertex  $s$ , find the shortest paths from  $s$  to all vertices in  $G$ .

- How much harder is this than finding single shortest path from  $s$  to  $t$ ?
  - Most algorithms will have to find the shortest path to every vertex in the graph in the worst case
    - Although may stop early in some cases

5/23/2022

CSE 332

7

## SSSP: Unweighted Version

- This is just Breadth First Search
  - Build a breadth first search tree starting from  $s$

5/23/2022

CSE 332

8

```
void BFS(Vertex s) {
    Queue q(NUM_VERTICES);
    Vertex v, w;
    for each w {
        w.dist = INFINITY;
        w.prev = -1;
    }
    s.dist = 0;
    q.enqueue(s);

    while (!q.isEmpty()) {
        v = q.dequeue();
        for each w adjacent to v
            if (w.dist == INFINITY) {
                w.dist = v.dist + 1;
                w.prev = v;
                q.enqueue(w);
            }
    }
}
```

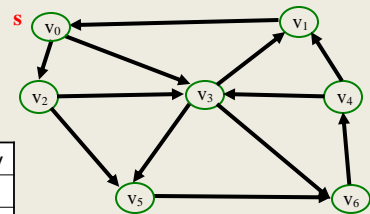
each edge examined at most once – if adjacency lists are used

each vertex enqueued at most once

5/23/2022

CSE 332

9



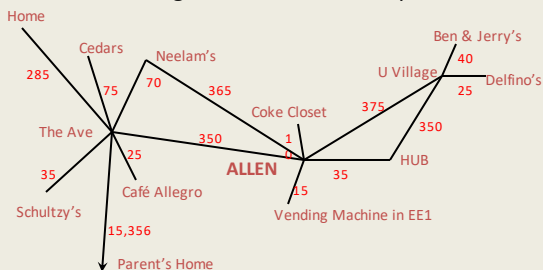
V	Dist	prev
V <sub>0</sub>		
V <sub>1</sub>		
V <sub>2</sub>		
V <sub>3</sub>		
V <sub>4</sub>		
V <sub>5</sub>		
V <sub>6</sub>		

CSE 332

10

## Weighted SSSP:

All edges are not created equal



Can we calculate shortest distance to all vertices from Allen Center?

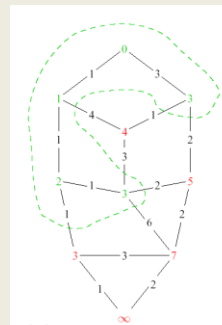
5/23/2022

CSE 332

11

Assume all edges have non-negative cost

## Dijkstra's Algorithm: Idea



Adapt BFS to handle weighted graphs

Two kinds of vertices:

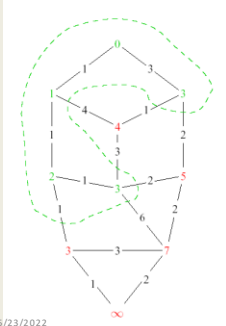
- **Known**
  - shortest distance is already known
- **Unknown**
  - Have tentative distance

5/23/2022

CSE 332

12

## Dijkstra's Algorithm: Idea



At each step:

- 1) Pick closest **unknown** vertex
- 2) Add it to **known** vertices
- 3) Update distances

5/23/2022

CSE 332

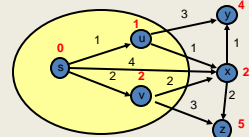
13

Assume all edges have non-negative cost

## Dijkstra's Algorithm

```

S = {}; d[s] = 0; d[v] = infinity for v != s
while S != V
  Choose v in V-S with minimum d[v]
  Add v to S
  for each w in the neighborhood of v
    newCost = d[v] + c(v, w)
    if (newCost < d[w])
      d[w] = newCost
      prev[w] = v
    
```



5/23/2022

CSE 332

14

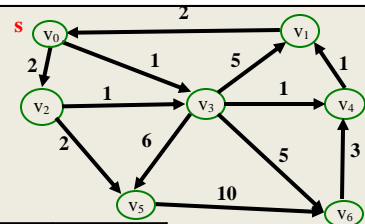
## Important Features

- Once a vertex is **known** (in S), the cost of the shortest path to that vertex is correct
- While a vertex is still **unknown**, another shorter path to it might still be found
- The shortest path can be found by following the previous pointers stored at each vertex

5/23/2022

CSE 332

15



V	Known?	Cost	Previous
v0			
v1			
v2			
v3			
v4			
v5			
v6			

5/23/2022

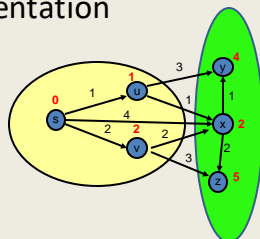
CSE 332

16

## Implementation

```

S = {}; d[s] = 0; d[v] = infinity for v != s
while S != V
  Choose v in V-S with minimum d[v]
  Add v to S
  for each w in the neighborhood of v
    newCost = d[v] + c(v, w)
    if (newCost < d[w])
      d[w] = newCost
      prev[w] = v
    
```



What are the heap operations?

How many heap operations?

5/23/2022

CSE 332

17

## Dijkstra Algorithm

```

int dist[N], prev[N];
for (int i = 0; i < N; i++){
  dist[i] = INFINITY;
  prev[i] = -1;
}
dist[s] = 0;
Heap h = new Heap(dist);

while (!h.isEmpty()){
  v = h.DeleteMin();
  for each w adjacent to v {
    int newCost = dist[v] + cost(v,w);
    if (newCost < dist[w]){
      dist[w] = newCost;
      h.DecreaseKey(w, newCost);
      prev[w] = v;
    }
  }
}
    
```

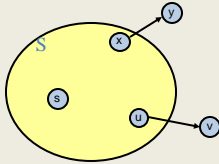
5/23/2022

CSE 332

18

## Correctness Proof

- Elements in  $S$  have the correct label
- Induction: when  $v$  is added to  $S$ , it has the correct distance label
  - $\text{Dist}(s, v) = d[v]$  when  $v$  added to  $S$



5/23/2022

CSE 332

19

## D-Heaps (again)

- Heaps with branching factor  $D$
- DeleteMin runtime  $O(D \log_D N)$
- DecreaseKey runtime  $O(\log_D N)$

5/23/2022

CSE 332

20

## Dijkstra's Algorithm with D heaps

- $n$  DeleteMin operations
- $m$  DecreaseKey operations
- Runtime  $O(n D \log_D n + m \log_D n)$
- What value for  $D$ ?

5/23/2022

CSE 332

21

## Why do we worry about negative cost edges?

5/23/2022

CSE 332

22