Jacqueline
Ben
Caroline

Teal
Tenzin
Pierce

Alderik
Angel
Will

Bruce
Peter
Dylan

Scott
Eliza
Lillie

Christian
Henry
Michael Cal.

Zeke
Chris
Tommaso

Zach
Graham
Annika

Bryan
Kai
Nadani

Michael
Cze.
Jackson
Joonwoo

S. KIMMEL

Shortest Paths

E = set of edges
V = set of vertices

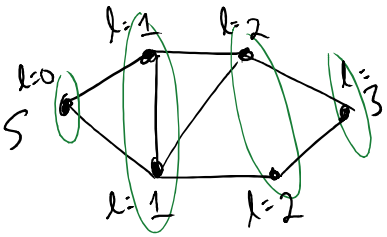
Input: Unweighted, undirected Graph $G=(V,E)$, starting node $s \in V$

Output: Array l, $l[v]$ = length of shortest path from s to v.

($l[v] = \infty$ if no path from s to v)

Applications? Maps, social connections, finacial transactions

Idea: explore each distance layer in turn:



Similar? Breadth-First Search!

```
l[v] = ∞  ∀ v ∈ V
vis[v] = false  ∀ v ∈ V      //make true after visiting
A = {}                       // create queue
A.add(s)                     // initialize with vertex s
l[s] = 0
vis[s] = true
while (A is not empty)
    - v = A.pop
    - for w: {v,w} ∈ E :
        if (vis[w] = false):
            • A.add(w)
            • Vis[w] = true
            • l[w] = l[v] + 1.
```

If remove lines involving l, have BFS

S. KIMMEL

Q: Will the algorithm be correct under following conditions?
(If not always, provide counter example.)

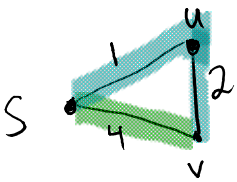
- directed
- positive weighted edges
- negative weights (uniform)

- What is time complexity of graph search if you store G in adjacency list data structure?

S. KIMMEL

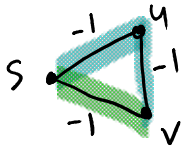
Q: Will the algorithm be correct under following conditions?
(If not always, provide counter example.)

- directed — OK!
- positive weighted edges — No



BFS Solution: $\ell[v]=4$
Correct: $\ell[v]=3$

- negative weights (uniform) — No



BFS Solution $\ell[v]=-1$
Correct: $\ell[v]=-2$

- What is time complexity of graph search if you store G in adjacency list data structure?

$O(n+m)$ $\quad n=|V| \quad m=|E|$

S. KIMMEL

Explain: Why is runtime $O(n+m)$

- Looks at edges (in for loop)
- Each edge can only be examined when its adjoining vertex is popped. Each vertex can only show up in QUEUE one time. $\Rightarrow$ Each edge is examined twice

- Looking at each edge takes constant time b/c use adjacency list.

$O(m)$ to do while loop

+

Initialization: $O(n)$

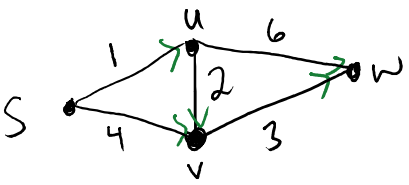
$O(n+m)$ $= O(n+m)$

Very fast!

(Optimal! Why?)

S. KIMMEL

Shortest Paths



Input: • Directed graph $G = (E, V)$ with positive weights l_{uv} $\forall$ $(u,v) \in E$

• vertex $s \in V$

Output: $\forall$ $v \in V$, find

$L(v)$ = length of shortest path from s to v

- path following direction of edges
- add weights of edges on path

Q: What is $L(s), L(u), L(v), L(w)$ for graph above?

A) $0, 1, 4, 6$ B) $0, 1, 3, 7$ C) $0, 1, 1, 2$

D) $0, 1, 3, 6$

Applications: navigation
financial transactions

Assumptions:
- s is connected to all other vertices (not important, just makes life easier)
- No negative edge weights (Important!)

S. KIMMEL

Dijkstra's Algorithm : Intuition, BFS

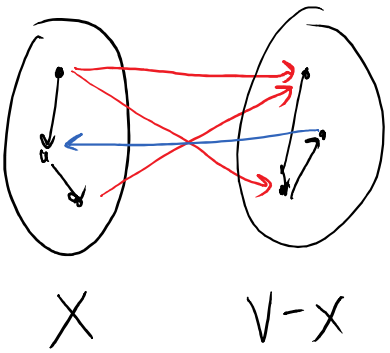
Initialization:

$X = \{s\}$ (vertices processed)

$A[s] = 0$ (array storing shortest path distance to v from s)

$B[s] = \phi$ (array storing shortest path to v from s)

B not necessary for implementation, just helpful for understanding



We care about edges from X to V-X

While $X \neq V$:

- among edges $(v,w) \in E$ with $v \in X$, $w \in V-X$, pick edge that minimizes

$$A[v] + \ell_{vw}$$

(ℓ_{vw}: weight of edge (v,w))

Dijkstra's greedy criterion

Let (v^*, w^*) be minimizing edge

- $X = X + w^*$
- $A[w^*] = A[v^*] + \ell_{v^*w^*}$ ($A[v^*]$: already computed, since $v^* \in X$)
- $B[w^*] = B[v^*] + (v^*, w^*)$

S.KIMMEL

Q: What kind of algorithm is Dijkstra's Algorithm?

A) Divide & Conquer

B) Dynamic Programming

No recurrence relation

C) Greedy

- No smaller subsystems
- Choose best thing right now
- Easy to describe
- Hard to prove

D) Other

S. KIMMEL

Runtime with Heap + Adjacency List

Run Time

$X = \{s\}$
$A[s] = 0$
$B[s] = \phi$

→ $O(n)$ to initialize length n arrays

Each $v \in V - s$ calculate $k(v) = \ell_{sv}$ store in heap

Initialize heap

→ $O(n \log n)$

While $X \neq V$

- among edges $(v, w) \in E$ with $v \in X$, $w \in V - X$, pick edge that minimizes

  $A[v] + \ell_{vw}$

→ $O(1)$ ⇒ total $O(n)$

Let (v^*, w^*) be minimizing edge

- $X = X + w^*$
- $A[w^*] = A[v^*] + \ell_{v^* w^*}$
- $B[w^*] = B[v^*] + (v^*, w^*)$
- Update heap
  - Remove w^* → $O(\log n)$ ⇒ Over algorithm: $O(n \log n)$
  - Update keys → Tricky - see next page