S. KIMMEL

Dijkstra's Algorithm : Intuition, BFS

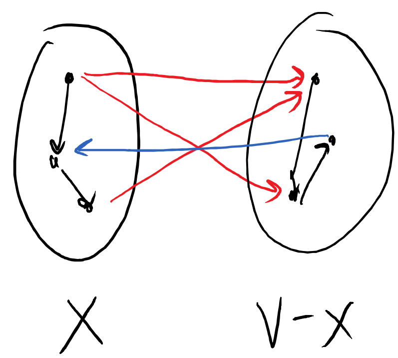
Initialization:

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

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

$B[s] = \emptyset$ (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

weight of edge (v,w)

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

Dijkstra's greedy criterion

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

- $X = X + w^*$
- $A[w^*] = A[v^*] + \ell_{v^* w^*}$

already computed, since $v^* \in X$

- $B[w^*] = B[v^*] + (v^*, w^*)$

S. KIMMEL

Analyzing Runtime

What do we repeatedly do ... what data structure would help us?

Find Minimum $\Rightarrow$ Min Heap!

What should we put in heap ... edges or vertices?

- Edges are more natural because we find edge with min Dijkstra criterion

- Vertices give faster runtime :)

Objects in Heap: $v \in X - V$

$\boxed{v}$ • key $= \min\limits_{\substack{u:\, u \in X \\ (u,v) \in E}} A[u] + l_{uv}$

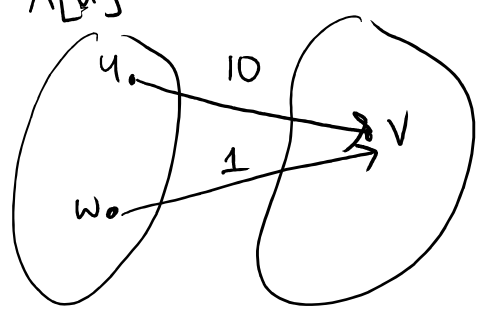
• prior $= u^*$ that minimizes key

attributes: • key

• prior

ex:

$A[u] = 5$



$A[w] = 7$

$\boxed{v}$ • key = 8

• prior = w

Idea:

- Each object in heap already does mini-competition to find best edge among all edges going to that vertex
- Then top element of heap gives edge with smallest Dijkstra Criterion overall.

S. KIMMEL

Runtime with Heap + Adjacency List

Run Time

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

$O(n)$ to initialize length n arrays

Initialize heap

Each $v \in V - s$ calculate

$\left.\begin{array}{l} Key = \ell_{sv} \\ Prior = s \end{array}\right\}$ if $\exists\ (s,v) \in E$

$\left.\begin{array}{l} Key = \infty \\ Prior = \phi \end{array}\right\}$ otherwise

$O(n \log n)$

While $X \neq V$

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

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

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

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

$O(1)$

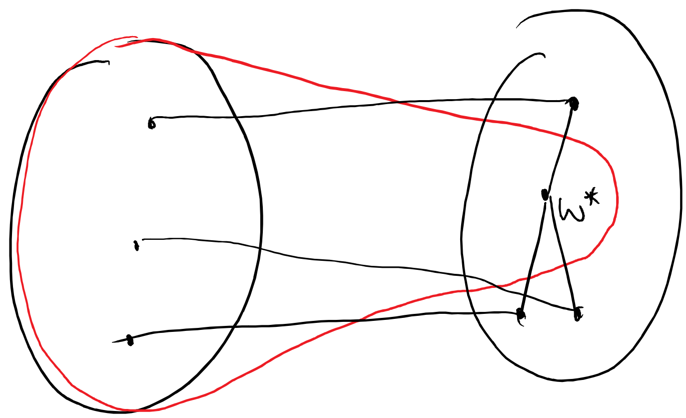
$\Rightarrow$ total $O(n)$

- Update heap
  - Remove w^*
  - Update keys

$O(\log n)$ $\Rightarrow$ Over algorithm: $O(n \log n)$

Tricky - see next page

S.KIMMEL



X before w^* added

X after w^* added

- For each $v \in V - X$ s.t. $(w^*, v) \in E$, need to check if key should be updated.

- Using adjacency list data structure, can find neighbors of w^* efficiently

For each neighbor on the list

- remove from heap
- update key if needed
- reinsert

$\rightarrow O(\log n)$

(maintain list of pointers to heap objects to find easily)

* Each edge in graph triggers at most once over course of algorithm. m edges

Total from this step $\Longrightarrow O(m \log n)$

S. KIMMEL

Total Runtime

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

Only $\log n$ worse than BFS!

S.KIMMEL

Proof of Correctness of Dijkstra

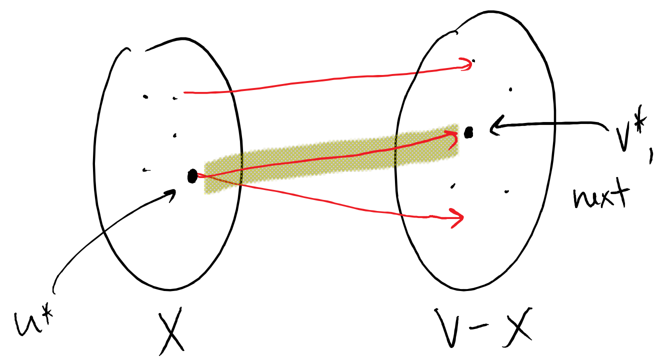
Loop Invariant: $\forall v \in X$, $A[v]$ = shortest distance from s to v
$B[v]$ shortest path from s to v

Initialization.

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

The shortest path from s to s has weight 0, and is empty ✓

Maintenance



Assume: $\forall$ $v \in X$

- $A[v]$ is shortest distance
- $B[v]$ is shortest path

• Let (u^*, v^*) be the edge with minimal Dijkstra's criterion

• Let $P = B[u^*] \cup (u^*, v^*)$ ← Dijkstra's Alg sets $B[v^*] = P$

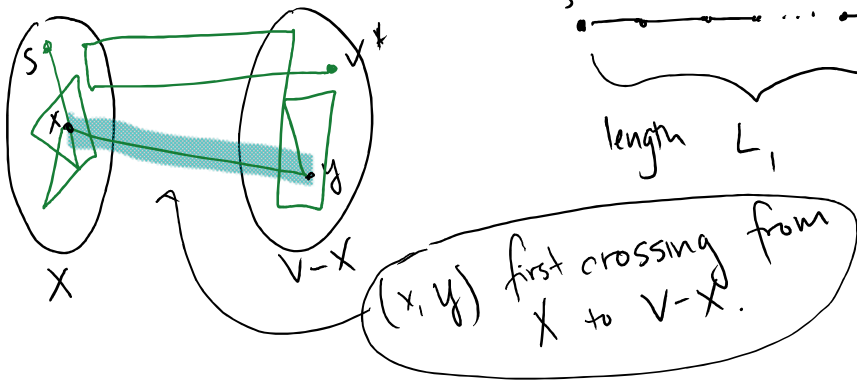
↑ actual shortest path path from s to u^* by our invariant

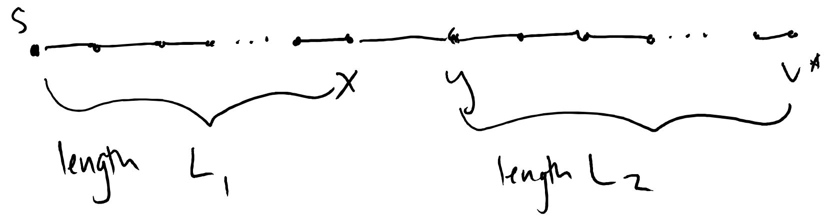
$\boxed{\text{Need to show: } P \text{ is shortest path from } s \text{ to } v^*}$

(This implies loop invariant is maintained.)

S. KIMMEL

Suppose for contradiction that there is a shorter path P^* from s to v^*





Q: Prove P^* is longer than or equal to P.

A: $L_1 \geq A[x]$ by inductive assumption

$L_2 \geq 0$ by non-negativity of edges

Path length is

$$L_1 + L_2 + l_{xy} \geq A[x] + l_{xy} \geq A[u^*] + l_{u^*, v^*} = \text{length of } P$$

Contradiction!

$\Rightarrow$ P is correct shortest path

<u>Termination</u> − A, B contain all the correct info!