Design and Analysis of Algorithms I

Graph Primitives

Dijkstra's Algorithm: The Basics

Single-Source Shortest Paths

Input: directed graph G=(V, E). (m=|E|, n=|V|)

- each edge has non negative length l_e
- source vertex s

Output: for each $v \in V$, compute

L(v) := length of a shortest s-v path in G

Length of path
= sum of edge lengths

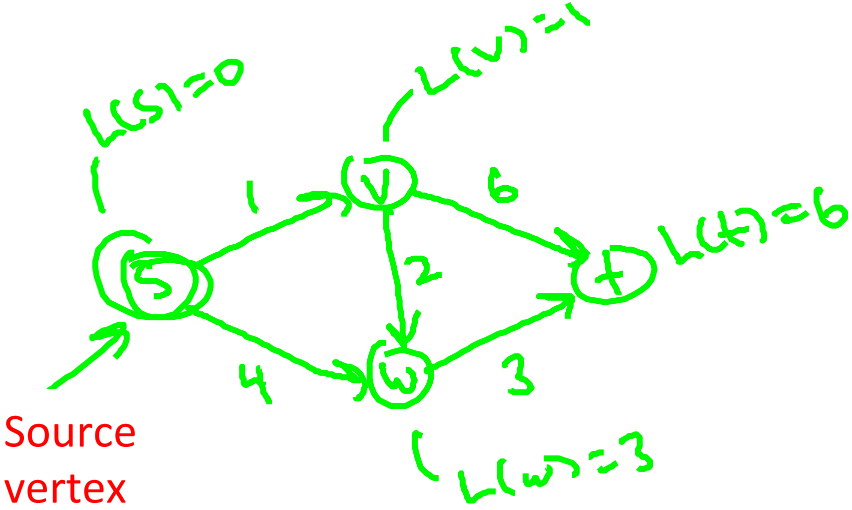
Path length = 6

Assumption:

1. [for convenience] $\forall v \in V, \exists s \Rightarrow v$ path
2. [important] $le \geq 0 \;\; \forall e \in E$

One of the following is the list of shortest-path distances for the nodes s,v,w,t, respectively. Which is it?

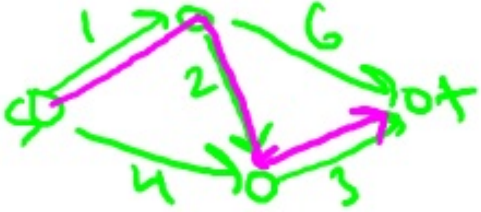
- 0,1,2,3
- 0,1,4,7
- 0,1,4,6
- 0,1,3,6



Why Another Shortest-Path Algorithm?

Question: doesn't BFS already compute shortest paths in linear time?

Answer: yes, IF $l_e = 1$ for every edge e.



Question: why not just replace each edge e by directed path of l_e unit length edges:

Answer: blows up graph too much

Solution: Dijkstra's shortest path algorithm.

Dijkstra's Algorithm

This array only to help explanation!

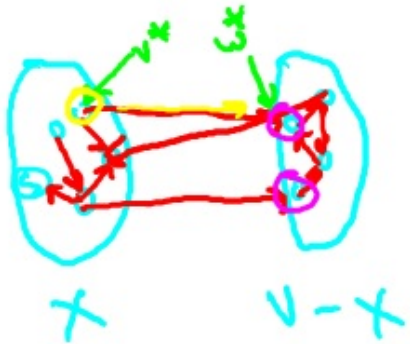
Initialize:

- X = [s] [vertices processed so far]
- A[s] = 0 [computed shortest path distances]
- B[s] = empty path [computed shortest paths]

Main Loop

- while X≠V:

-need to grow x by one node



Main Loop cont'd:

- among all edges $(v, w) \in E$ with $v \in X, w \notin X$, pick the one that minimizes $A[v] + l_{vw}$

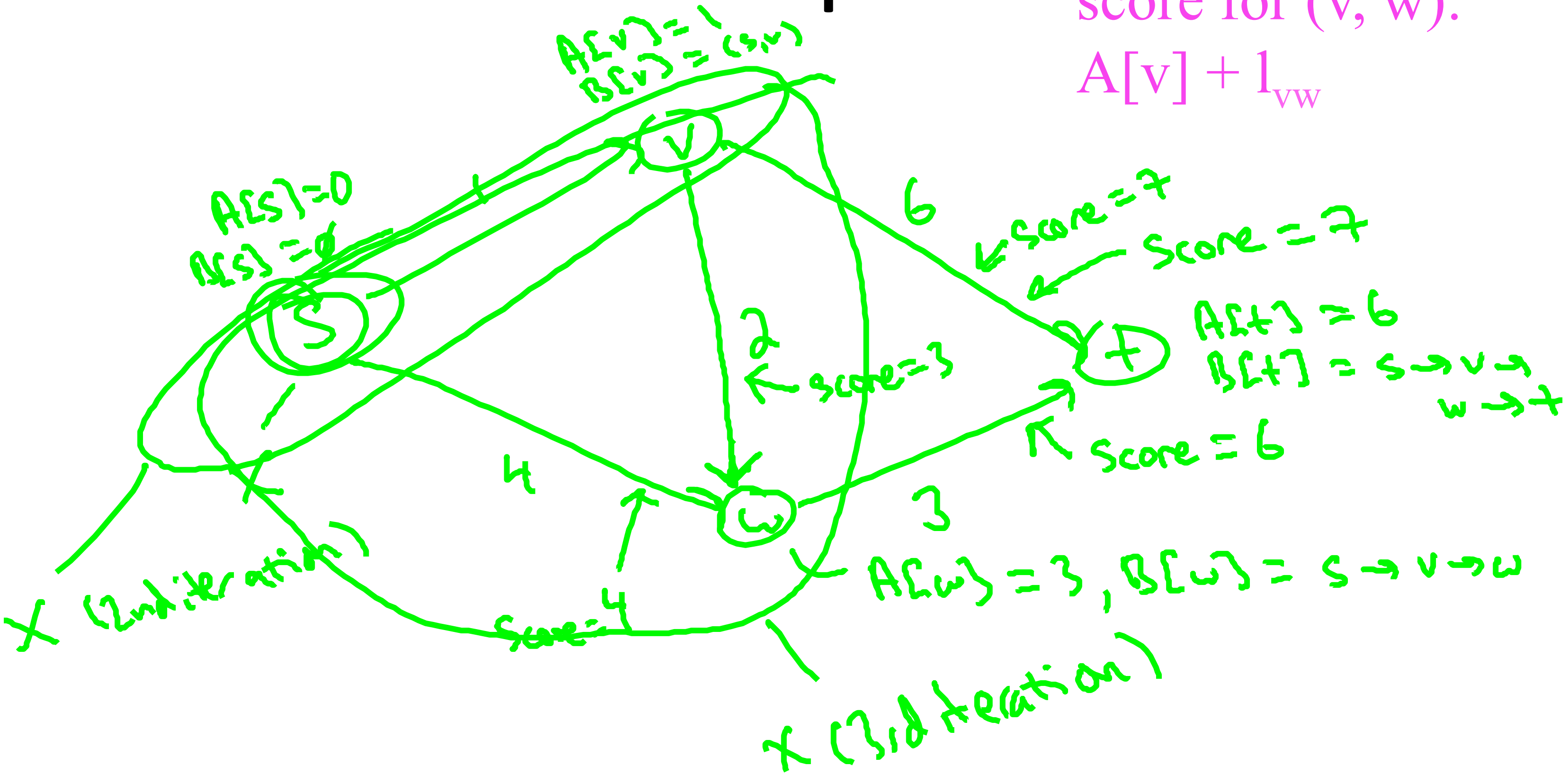
[call it (v*, w*)]

Already computed in earlier iteration

- add w* to X
- set $A[w^*] := A[v^*] + l_{v^*w^*}$
- set $B[w^*] := B[v^*]u(v^*, w^*)$

Example

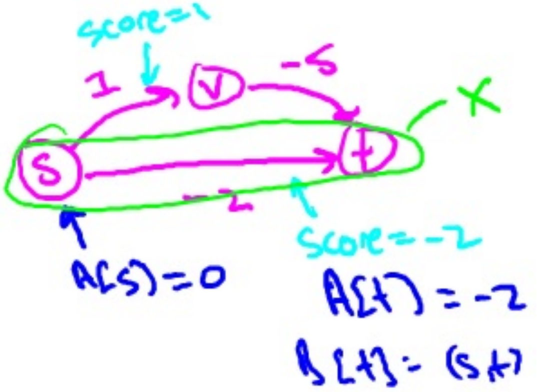
Dijkstra's greedy score for (v, w): $A[v] + l_{vw}$



Tim Roughgarden

Non-Example

Question: why not reduce computing shortest paths with negative edge lengths to the same problem with non negative lengths? (by adding large constant to edge lengths)



Problem: doesn't preserve shortest paths !

Also: Dijkstra's algorithm incorrect on this graph !
(computes shortest s-t distance to be -2 rather than -4)

Graph Primitives

Dijkstra's Algorithm: Why It Works

Design and Analysis of Algorithms I

Dijkstra's Algorithm

This array only to help explanation!

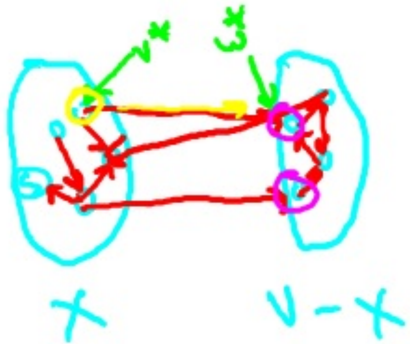
Initialize:

- X = [s] [vertices processed so far]
- A[s] = 0 [computed shortest path distances]
- B[s] = empty path [computed shortest paths]

Main Loop

- while X≠V:

-need to grow x by one node



Main Loop cont'd:

- among all edges $(v, w) \in E$ with $v \in X, w \notin X$, pick the one that minimizes $A[v] + l_{vw}$

[call it (v*, w*)]

Already computed in earlier iteration

- add w* to X
- set $A[w^*] := A[v^*] + l_{v^*w^*}$
- set $B[w^*] := B[v^*]u(v^*, w^*)$

Correctness Claim

Theorem [Dijkstra] For every directed graph with nonnegative edge lengths, Dijkstra's algorithm correctly computes all shortest-path distances.

$$[i.e., \; A[v] = L(v) \; \forall v \in V]$$

what algorithm computes

True shortest distance from s to v

Proof: by induction on the number of iterations.

Base Case: $A[s] = L[s] = 0$ (correct)

Proof

Inductive Step:

Inductive Hypothesis: all previous iterations correct (i.e., $A[v] = L(v)$ and $B[v]$ is a true shortest s-v path in G, for all v already in X).

In current iteration:

in X   not in X

We pick an edge (v^*, w^*) and we add w^* to X.

We set $B[w^*] = B[v^*] \cup (v^*, w^*)$

has length $L(v^*) + l_{v^*w^*}$ has length $L(v^*)$

L(v*) by I.H

Also: $A[w^*] = A[v^*] + l_{v^*w^*} = L(v^*) + l_{v^*w^*}$

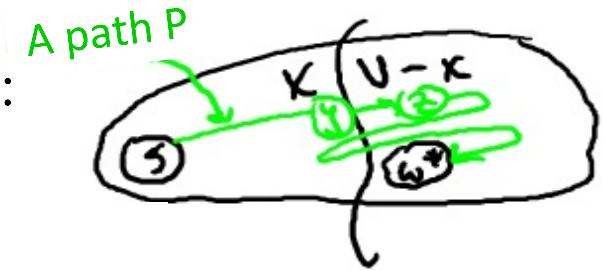
Proof (con'd)

Upshot: in current iteration, we set:

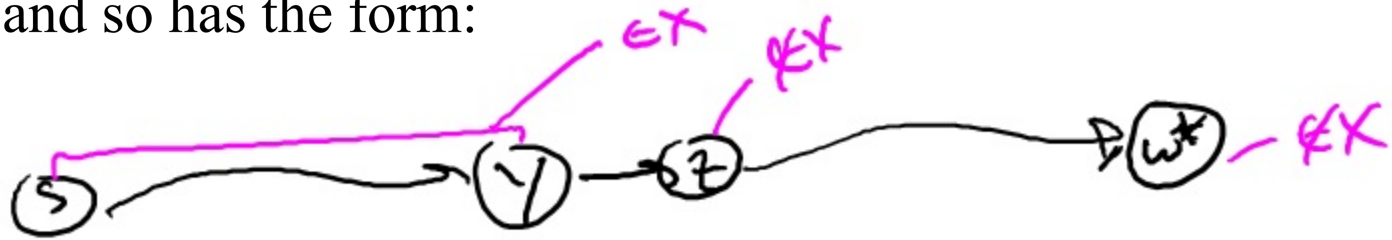
1. $A[w^*] = L(v^*) + l_{v^*w^*}$
2. $B[w^*]$ = an s -> w* path with length $(L(v^*) + l_{v^*w^*})$

To finish proof: need to show that *every* s-w* path has length >= $L(v^*) + l_{v^*w^*}$ (if so, our path is the shortest!)

So: Let P= any s->w* path. Must "cross the frontier":

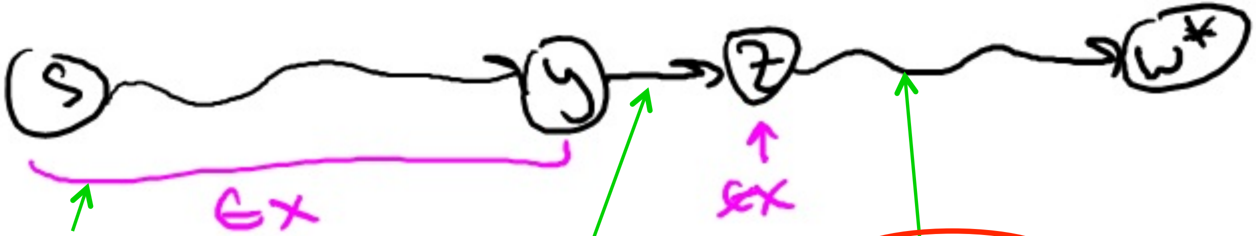


and so has the form:



Proof (con'd)

So: every s->w* path P has to have the form



(Since no negative edges !)

>= length of shortest s-y path = L(y) = A[y]

Length = l_{yz}

Length >=0

By inductive hypothesis (since $y \in X$)

($y \in X$
$z \notin X$)

Total length of path P: at least A[y] + C_{yz}

length of our path !

-> by Dijkstra's greedy criterion, $A[v^*] + l_{v^*w^*} \leq A[y] + l_{yz} \leq$ length of P

Q.E.D.

Design and Analysis of Algorithms I

Graph Primitives

Dijkstra's Algorithm: Fast Implementation

Single-Source Shortest Paths

Input: directed graph G=(V, E). (m=|E|, n=|V|)

- each edge has non negative length l_e
- source vertex s

Length of path = sum of edge lengths

Path length = 6

Output: for each $v \in V$,compute

L(v) := length of a shortest s-v path in G

Assumption:

1. [for convenience] $\forall v \in V, \exists s \Rightarrow v$ path
2. [important] $le \geq 0 \ \forall e \in E$

Dijkstra's Algorithm

This array only to help explanation!

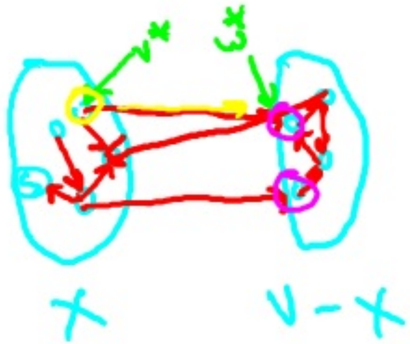
Initialize:

- X = [s] [vertices processed so far]
- A[s] = 0 [computed shortest path distances]
- ~~B[s] = empty path [computed shortest paths]~~

Main Loop

- while X≠V:

-need to grow x by one node



Main Loop cont'd:

- among all edges $(v,w) \in E$ with $v \in X, w \notin X$, pick the one that minimizes $A[v] + l_{vw}$

Already computed in earlier iteration

[call it (v*, w*)]

- add w* to X
- set $A[w^*] := A[v^*] + l_{v^*w^*}$
- ~~set $B[w^*] := B[v^*]u(v^*, w^*)$~~

Which of the following running times seems to best describe a "naïve" implementation of Dijkstra's algorithm?

- $\theta(m+n)$
- $\theta(m \log n)$
- $\theta(n \uparrow 2)$
- $\theta(mn)$

- (n-1) iterations of while loop
- $\theta(m)$ work per iteration

[$\theta(1)$ work per edge]

CAN WE DO BETTER?

Heap Operations

Recall: raison d'être of heap = perform Insert, Extract-Min in O(log n) time.
[rest of video assumes familiarity with heaps]

Height ~ $\log_2 n$

- conceptually, a perfectly balanced binary tree
- Heap property: at every node, key <= children's keys
- extract-min by swapping up last leaf, bubbling down
- insert via bubbling up

Also: will need ability to delete from middle of heap. (bubble up or down as needed)

Two Invariants

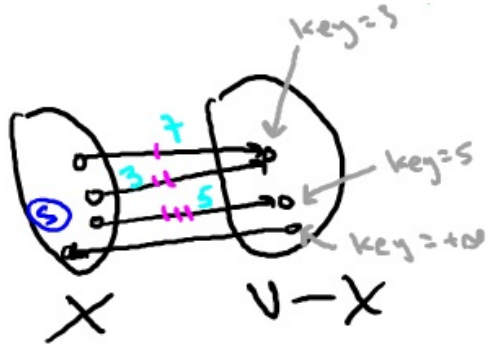
Dijkstra's greedy score of (v, w) : $A[v] + l_{vw}$

Invariant # 1: elements in heap = vertices of V-X.

Invariant #2: for $v \notin X$

Key[v] = smallest Dijstra greedy score of an edge (u, v) in E with v in X

(of $+\infty$ if no such edges exist)

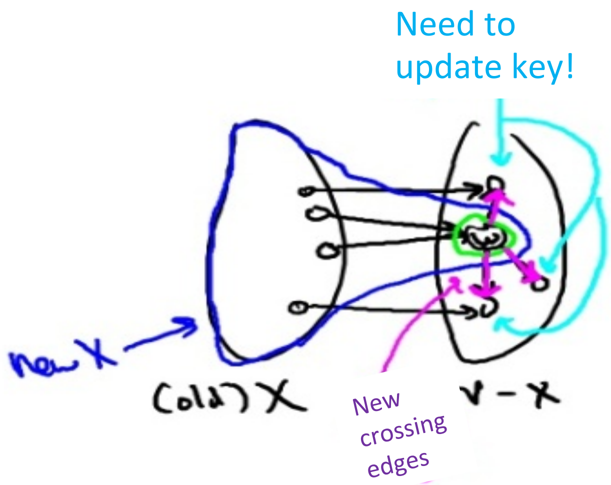


Point: by invariants, Extract-Min yields correct vertex w* to add to X next.

(and we set A[w*] to key[w*])

Maintaining the Invariants

To maintain Invariant #2: [i.e., that $\forall v \notin X$
Key[v] = smallest Dijkstra greedy score of edge (u,v) with u in X]



When w extracted from heap (i.e., added to X)

- for each edge (w,v) in E:
  - if v in V-X (i.e., in heap)
    - delete v from heap
    - recompute key[v] = min{key[v] , A[w] + l_{wv} }
    - re-Insert v into heap

Key update

Greedy score of (w,v)

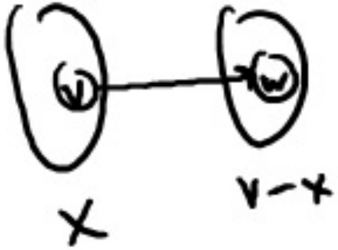
Tim Roughgarden

Running Time Analysis

You check: dominated by heap operations. (O(log(n)) each)

- (n-1) Extract mins
- each edge (v,w) triggers at most one Delete/Insert combo

(if v added to X first)



So: # of heap operations in O(n+m) = O(m)

So: running time = O(m log(n)) (like sorting)

Since graph is weakly connected