Design and Analysis of Algorithms I

Graph Primitives

Dijkstra's Algorithm: Fast Implementation

Single-Source Shortest Paths

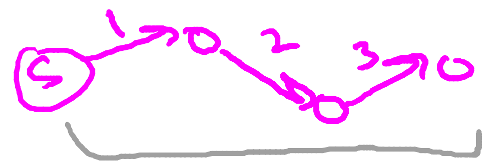
Input: directed graph $G=(V,E)$. $(m=|E|, n=|V|)$

- each edge has <u>nonnegative</u> 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

Assumptions:

① [for convenience] $\forall v \in V$, $\exists$ an $s \rightsquigarrow v$ path

② [important] $\ell_e \geq 0$ $\forall e \in E$

Tim Roughgarden

Dijkstra's Algorithm

Initialize:

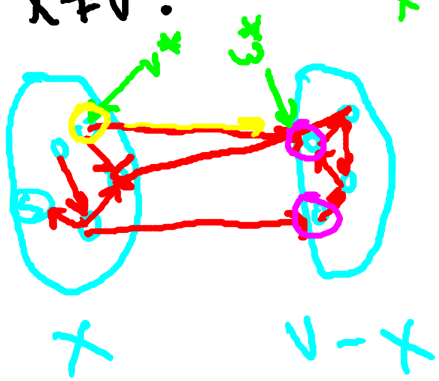
- $X = \{s\}$ [vertices processed so far]
- $A[s] = 0$ [computed shortest path distances]
- ~~$B[s]$ = empty path [computed shortest paths]~~

this array only to help explanation!

Main Loop

- while $X \neq V$:

- need to grow X by one node



Main Loop con'd

- among all edges $(v,w) \in E$ with $v \in X$, $w \notin X$, pick the one that minimizes

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

(Dijkstra's greedy criterion)

already computed in earlier iteration

[call it (v^*, w^*)]

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

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

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

- (n-1) iterations of while loop
- $\Theta(m)$ work per iteration
[$\Theta(1)$ work per edge]

CAN WE DO BETTER?

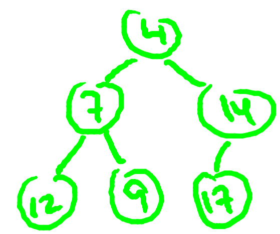
Heap Operations

Recall: raison d'être of heap = perform Insert, Extract-Min in $O(\log n)$ time.

[rest of video assumes familiarity with heaps]

height $\approx \log_2 n$

- 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)

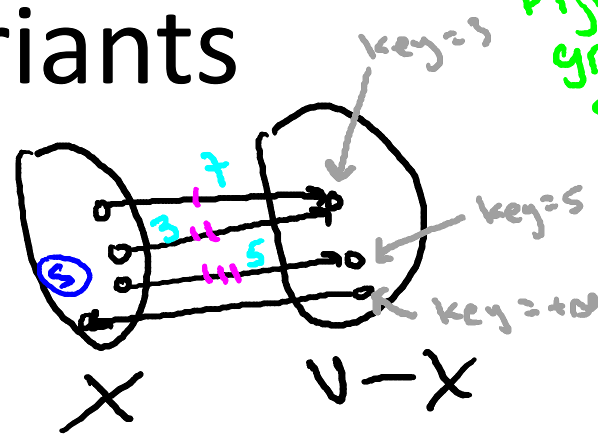
Two Invariants

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

Invariant #2: for $v \notin X$,

$key[v]$ = smallest Dijkstra greedy score of an edge $(u,v) \in E$ with $u \in X$

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



Dijkstra greedy score of (v,w): $A[v] + \ell_{vw}$

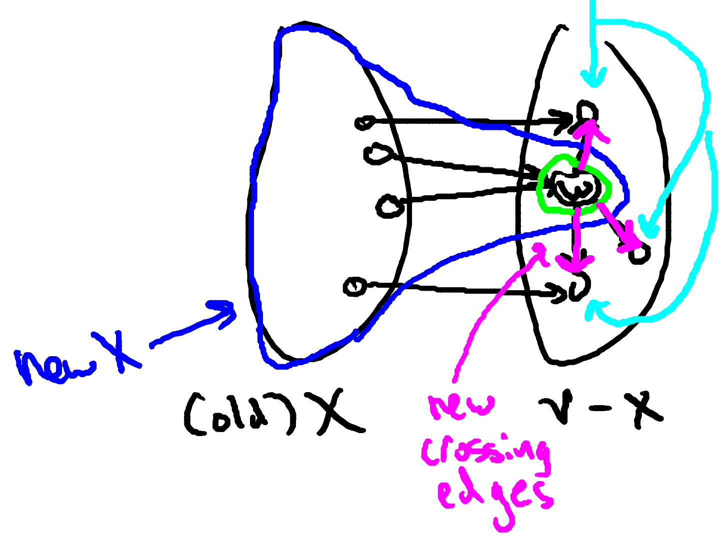
Point: by invariants, Extract-Min yields correct vertex w^* to add to X next.

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

Tim Roughgarden

Maintaining the Invariants

To maintain Invariant #2: [that, $\forall v \notin X$, $\text{Key}[v]$ = 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)

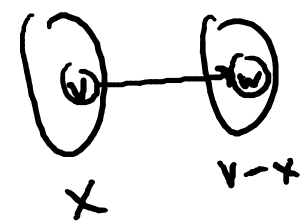
key update

- delete v from heap
- recompute $\text{key}[v] = \min\{\text{key}[v], \overbrace{A[w] + \ell_{wv}}^{\text{greedy Dijkstra score of } (w,v)}\}$
- re-Insert v into heap

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 is $O(n+m) = O(m)$ ← since graph weakly connected

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

QED!