Graph Primitives

Dijkstra's Algorithm: Fast Implementation

Design and Analysis of Algorithms I

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!

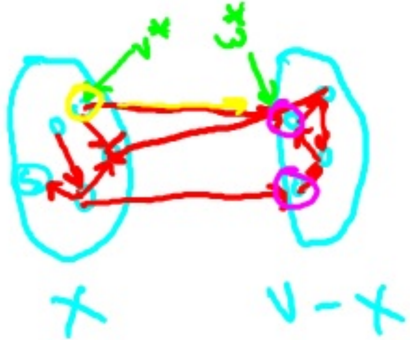
<u>Initialize</u>:

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

<u>Main Loop</u>

- while X≠V:

-need to grow x by one node



<u>Main Loop cont'd:</u>

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

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 scan edges to see if they are eligible

[$\theta(1)$ 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

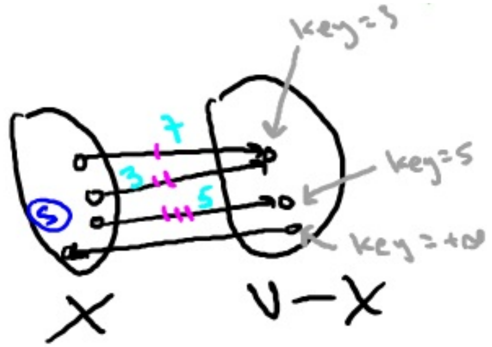
Dijkstra's greedy score of (v, w) : $A[v] + l_{vw}$

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)

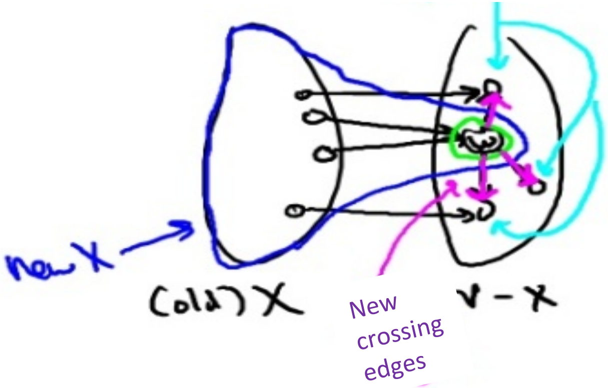


Point: by invariants, Extract-Min yields correct vertex w* to add to X next.

(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

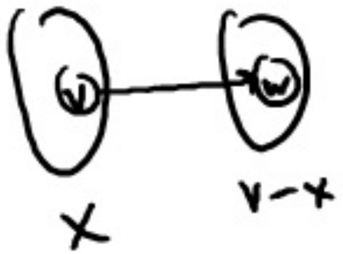
Greedy score of (w,v)

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