Algorithm Design and Analysis

Victor Adamchik — CS 15-451 — Spring 2015

Lecture 10 — Feb 16, 2015 — Carnegie Mellon University

Graph Algorithms - 3

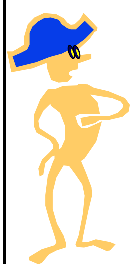
Plan:

Min-cost Spanning Tree Algorithms:

- Prim's (review)
- Arborescence problem

Kleinberg-Tardos, Ch. 4

The Minimum Spanning Tree for Undirected Graphs



Find a spanning tree of minimum total weight.

The weight of a spanning tree is the sum of the weights on all the edges which comprise the spanning tree.

The Minimum Spanning Tree

Joseph Kruskal (1929-2010)

Boruvka's Algorithm (1926)
Kruskal's Algorithm (1956)
Prim's Algorithm (1957)

Robert Prim (1921-)

Prim's Algorithm

Greedy algorithm that builds a tree
one VERTEX at a time.

First described by Jarnık in a 1929 letter to Boruvka.
Rediscovered by Kruskal in 1956, by Prim in 1957, by Loberman and Weinberger in 1957, and finally by Dijkstra in 1958.

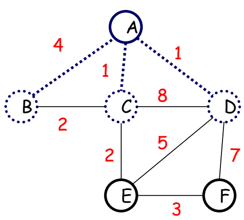
Prim's Algorithm

algorithm builds a tree one VERTEX at a time.

- Start with an arbitrary vertex as component *C*
- Expand *C* by adding a new vertex having the minimum weight edge with exactly one end point in *C*.
- Continue to grow the tree until *C* gets all vertices.

Prim's Algorithm

C={a}



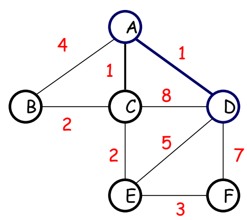
heap

d-1 c-1 b-4 e-oo f-oo

decleteMin

Prim's Algorithm

C={a,d}

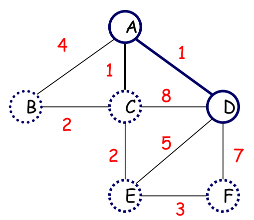


heap

c-1 b-4 e-oo f-oo

Prim's Algorithm

C={a,d}



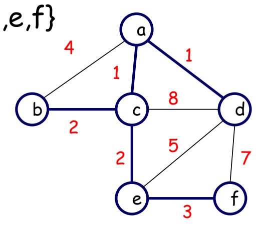
heap

c-1 b-4 e-5 f-7

decreaseKey

Prim's Algorithm

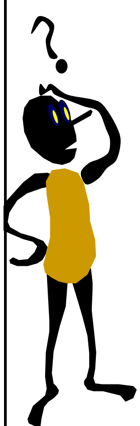
C={a,d,c,b,e,f}



Weight = 1+1+2+2+3 = 9

Property of the MST

Lemma: Let X be any subset of the vertices of G, and let edge e be the smallest edge connecting X to G-X. Then e is part of the minimum spanning tree.



What is the worst-case runtime complexity of Prim's Algorithm?

We run deleteMin V times
We update the queue E times

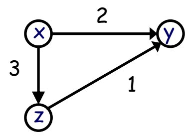
O(V*log V + E*log V)

deleteMin

decreaseKey

O(1) – Fibonacci heap

The Minimum Spanning Tree for Directed Graphs



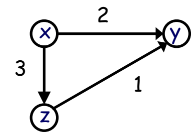
Start at X and follow the greedy approach

We will get a tree of size 5, though the min is 4, (x-z-y)

However there is even a smaller subset of edges – 3, (x-y, z-y)

The Minimum Spanning Tree for Directed Graphs

This example exhibits two problems



What is the meaning of MST for directed graphs?

Clearly, we want to have a rooted tree, in which we can reach any vertex staring at the root

How would you find it?

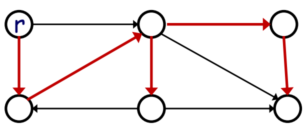
Clearly, the greedy approach of Prim's does not work

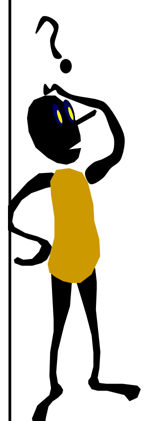
Arborescences

Def. Given a digraph G = (V, E) and a vertex r∈V, an arborescence (rooted at r) is a treeT s.t.

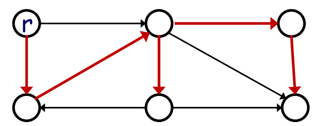
T is a spanning tree of G if we ignore the direction of edges.

There is a directed unique path in T from r to each other node v ∈ V.





Given a digraph G, find an arborescence rooted at r (if one exists)



Arborescences

Theorem 1. A subgraph T of digraph G is an arborescence rooted at r iff T has no directed cycles and each its node v ≠ r has exactly one entering edge.

Proof.

⇒) Trivial.

⇐) Start at vertex v and follow edges in backward direction.

Since no cycles you eventually reach r.

Arborescences

Theorem 2. A digraph G contains an arborescence if and only if each vertex in G is reachable from r.

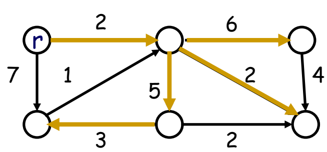
Proof.

⇒) Trivial.

⇐) Each vertex is reachable from r, the BFS will find an arborescence.

Min-cost Arborescences

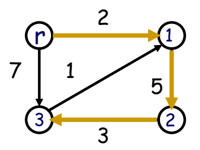
Given a digraph G with a root node r and with a nonnegative cost on each edge, compute an arborescence rooted at r of minimum cost.

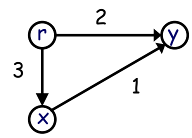


We assume that all vertices are reachable from r.

Min-cost Arborescences

Observation 1. This is not a min-cost spanning tree. It does not necessarily include the cheapest edge.



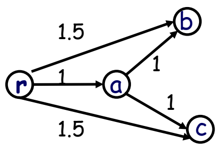


Running Prim's on undirected graph won't help.

Running an analogue of Prim's for directed graph won't help either

Min-cost Arborescences

Observation 2. This is not a shortest-path tree



Edges rb and rc won't be in the min-cost arborescence tree

Edge reweighting

For each $v \neq r$, let $\delta(v)$ denote the min cost of all edges entering v.

In the picture, $\delta(x)$ is 1.

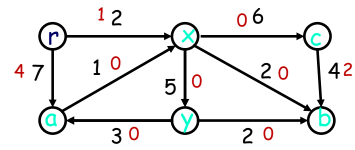
The reduced cost $w^*(u, v) = w(u, v) - \delta(v) \geq 0$

$\delta(y)$ is 5.

$\delta(a)$ is 3.

$\delta(b)$ is 3.

$\delta(c)$ is 6.



$w^*(u, v) = w(u, v) - \delta(v)$

Lemma. An arborescence in a digraph has the min-cost with respect to w iff it has the min-cost with respect to w*.

Proof. Let T be an arborescence in G(V,E).

Compute w(T) - w*(T)

δ(v) - min cost of any edge entering v

$$w(T) - w^*(T) = \sum_{e \in T} w(e) - w^*(e) = \sum_{v \in V \setminus r} \delta(v)$$

The last term does not depend on T. QED

Algorithm: intuition

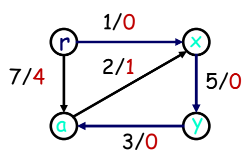
Let G* denote a new graph after reweighting.

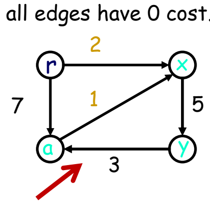
For every v≠r in G* pick 0-weight edge entering v.

Let B denote the set of such edges.

If B is an arborescence, we are done.

Note B is the min-cost since all edges have 0 cost.

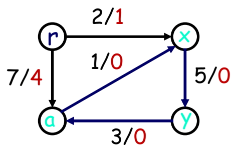




If B is NOT an arborescence…

Algorithm: intuition

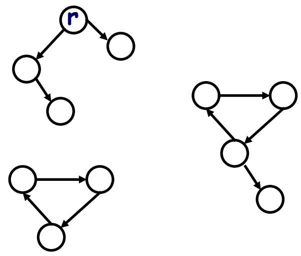
When B is not an arborescence?



…when B contains a cycle

How can it happen B is not an arborescence?

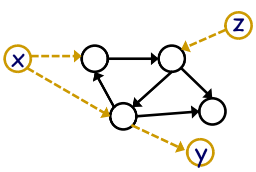
Note, only a single edge can enter a vertex

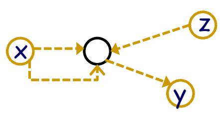


when it has a <u>directed</u> cycle or several cycles…

Vertex contraction

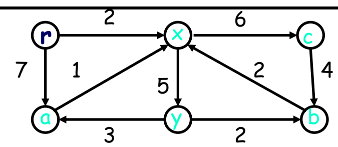
The main idea is to contract every cycle into a supernode.
Dashed edges and nodes are from the original graph G.



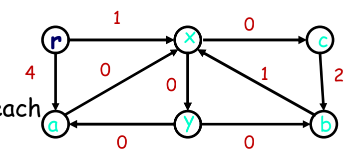


Recursively solve the problem in contracted graph

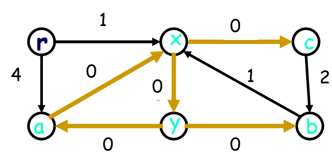
reweight



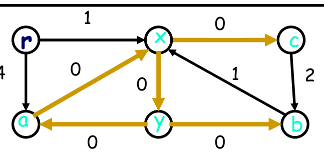
Take 0-weight edge for each vertex



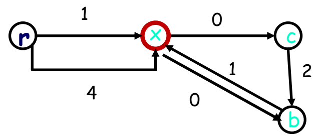
Cycle
axy



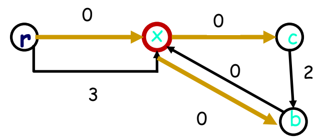
Contract
axy



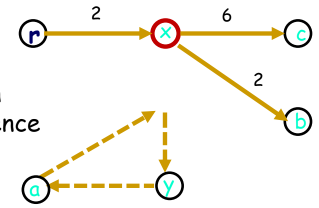
reweight



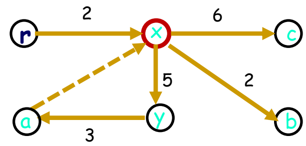
Take
0-weight
edges.
break ties
arbitrarily



Extend an
arborescence

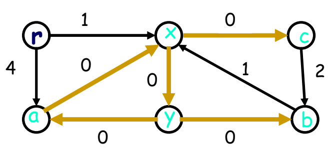


By removing
an edge from
a cycle



Correctness

Lemma. Let C be a cycle in G consisting of 0-cost edges. There exists a mincost arborescence rooted at r that has exactly one edge entering C.



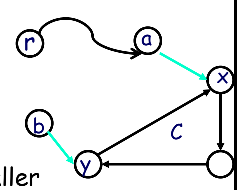
Correctness

Lemma. Let C be a cycle in G consisting of 0-cost edges. There exists a mincost arborescence rooted at r that has exactly one edge entering C.

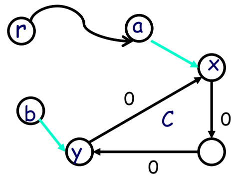
Proof. Let T be a min-cost arborescence that has more than one edge enters C.

Let a-x and b-y are edges entering C. Let b does not lie on the (shortest) path r-a.

We will show that we can find another arborescence with a smaller number of edges entering C.



Correctness



Construct a digraph B s.t.

1) delete all edges in T that enters C except (a,x)

2) add all edges in C except the one that enters x.

$B = T \setminus \{b,y\} \cup C$

By Th.2 B has an arborescence T_1, and its cost is

$cost(T_1) \leq cost(B) = cost(T) - w(b,y) \leq cost(T)$

The Algorithm

For each v≠r compute δ(v) – the mincost of edges entering *v.*

For each v≠r compute $w^*(u, v) = w(u, v) - \delta(v)$.

For each v≠r choose 0-cost edge entering v.

Let us call this subset of edges T.

If T forms an arborescence, we are done.

else

Contract every cycle C to a supernode

Repeat the algorithm

Extend an arborescence by adding all but one edge of C.

Return

Complexity

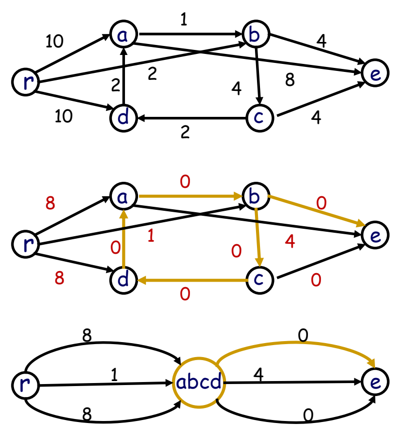
At most V contractions (since each one reduces the number of nodes).

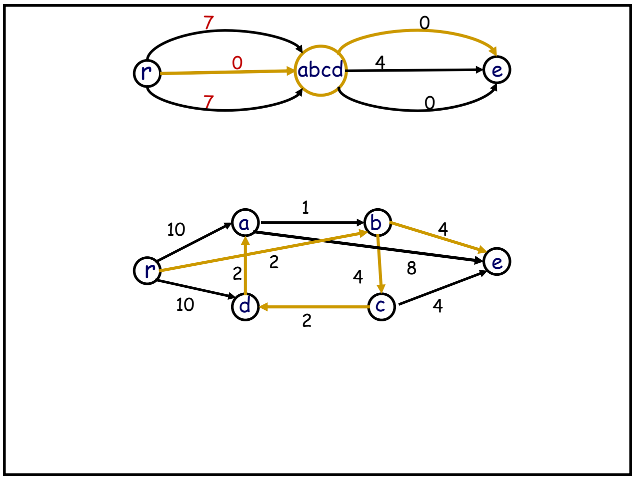
Finding and contracting the cycle C takes O(E).

Transforming T' into T takes O(E) time.

Total - O(V E).

Faster for Fibonacci heaps.




7
0
7
abcd
r
e
0
4
0
r
a
b
c
d
e
10
10
1
2
2
4
4
8
4
2