Analysis of Algorithms

V. Adamchik — CSCI 570 — Fall 2016

Discussion 5 — University of Southern California

Dijkstra's Algorithm

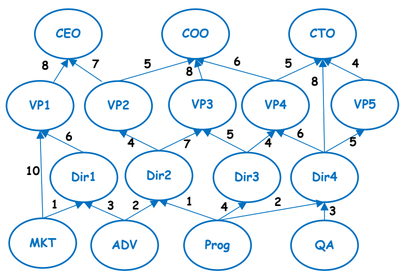
MST

Solving Recurrences

Problem 1

You are given a graph representing the several career paths available in industry. Each node represents a position and there is an edge from node v to node u if and only if v is a pre-requisite for u. Top positions are the ones which are not pre-requisites for any positions. Start positions are the ones which have no pre-requisites. The cost of an edge (v,u) is the effort required to go from one position v to position u. Ivan wants to start a career and achieve a top position with minimum effort. Using the given graph can you provide an algorithm with the same run time complexity as Dijkstra's? Assume the graph is a DAG.

Problem 1



Solution

We could run Dijkstra's algorithm from all of the start nodes, then compare the cost of the paths to all of the top positions in each execution.

Dijkstra's - O((V+E) log V)

Dijkstra's from all start vertices - O(V (V+E) log V)

We could use a topological sorting (Viterbi's algorithm)

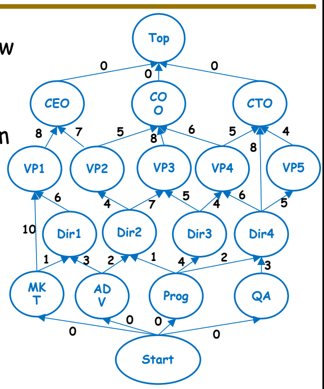
Viterbi's - O(V+E)

Viterbi's from all start vertices - O(V (V+E))

Solution

What if we create a new start node and end node?

Run Dijkstra's algorithm or Viterbi's algorithm from the new start node.

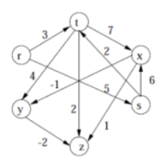


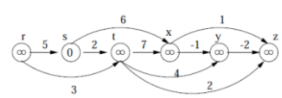
Viterbi's Algorithm

Find a topological order.

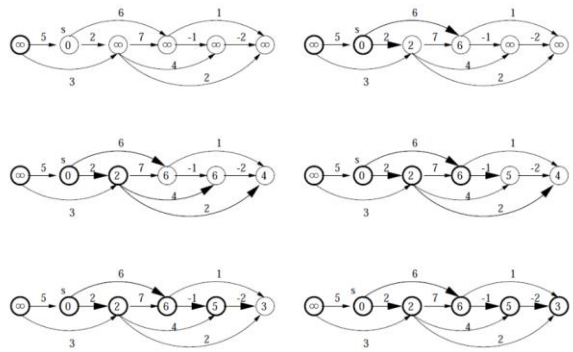
Process vertices in that order.

During processing, we update distances from to it to the adjacent vertices.





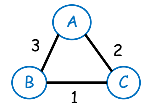
Viterbi’s Algorithm



Problem 2

Given a graph whose edge weights are integers in the range [0, W], where W is a relatively small integer number. We could run Dijkstra's algorithm to find the shortest distances from the start vertex to all other vertices.

Design a new algorithm that will run in linear time O(V +E) and therefore outperform Dijkstra's algorithm.



Solution

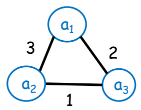
Create a new graph

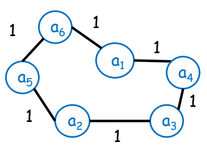
V_n= V - E + Σweights E_n= Σweights

Run BFS

Complexity V_n+E_n= V – E + 2 Σweights

≤ V – E + 2 E W





Problem 3

Suppose we have two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, along with T_1 which is a MST of G_1 and T_2 which is a MST of G_2. Now consider a new graph G = (V, E) such that $V = V_1 \cup V_2$ and $E = E_1 \cup E_2 \cup E_3$ where E_3 is a new set of edges that all cross the cut (V_1, V_2).

Consider the following algorithm, which is intended to find a MST of G

Maybe-MST(T_1, T_2, E_3)

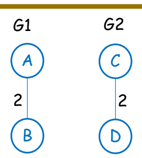
e_{min} = a minimum weight edge in E_3

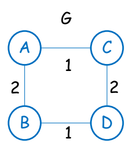
$T = T_1 \cup T_2 \cup \{e_{min}\}$

return T

Does this algorithm correctly find a MST of G? Either prove it does or prove it does not.

Solution





The spanning tree using the Maybe-MST algorithm has cost=5

The MST of G has a cost=4

Therefore the algorithm does not correctly find an MST

Problem 4

Suppose we are given a graph G with all distinct edge costs. Let T be a minimum spanning tree for G. Now suppose that we replace each edge cost c_e by its square, c_e^2, thereby creating a new graph but with the different distinct costs. Prove or disprove whether T is still an MST for this new graph.

Solution

If the graph only contains non-negative edge weights, squaring the cost of each edge would keep the relative comparisons the same. Therefore, an MST on G would also be an MST on G^2

Problem 5

Consider an undirected graph $G = (V, E)$ with distinct edge weights w_e. Suppose T an MST of G. Now suppose each edge weight is increased by 1: the new weights are $w'_e = w_e + 1$. Does the minimum spanning tree change? Give an example where it changes or prove it cannot change.

Solution

If $w_G(u, v) > w_G(u', v')$ then $w_{G+1}(u, v) > w_{G+1}(u', v')$.

This proves that the sorted order of the edges will not change, so we will add edges into our MST in the same order.

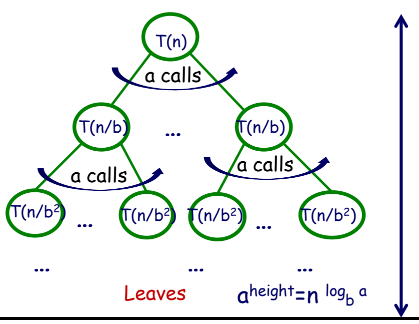
Therefore an MST on G would also be an MST on G+1.

Divide and Conquer
Master Theorem

Divide-and-Conquer $T(n) = a \cdot T(n/b) + f(n)$

$T(1) = O(1)$

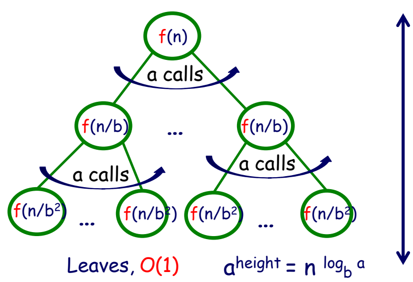
Tree of recursive calls:



Divide-and-Conquer $T(n) = a \cdot T(n/b) + f(n)$

$T(1) = O(1)$

This tree represents the total work:



The Master Theorem

$$T(n) = a\,T(n/b) + f(n)$$

Case 1:

if $f(n)=O(n^c)$ where $c < \log_b a$, then $T(n) = O(n^{\log_b a})$

Case 2:

if $f(n)=\Theta(n^c \log^k n)$ where $c = \log_b a$,

then $T(n) = \Theta(n^c \log^{k+1} n)$

Case 3:

if $f(n)=\Omega(n^c)$ where $c > \log_b a$, then $T(n) = \Theta(f(n))$

Problem 6

Solve the following recurrences using the Master Theorem:

$B(n) = 4\,B(n/2) + n^3$

$C(n) = 4\,C(n/2) + n^2$

$D(n) = 4\,D(n/2) + n$

Solution

$B(n) = 4B(n/2) + n^3$

Case 3: $a=4$, $b=2$, $f(n)=n^3$

$\log_b a = \log_2 4 = 2$

$f(n) = n^3 = \Omega(n^2)$

so $T(n) = \Theta(f(n)) = \Theta(n^3)$

$C(n) = 4C(n/2) + n^2$

Case 2: $a=4$, $b=2$, $f(n)=n^2$

$\log_b a = \log_2 4 = 2$ and $k=0$

$f(n) = n^2 = \Theta(n^2)$

so $T(n) = \Theta(n^c \log^{k+1} n) = \Theta(n^2 \log^{0+1} n) = \Theta(n^2 \log n)$

Solution

$D(n) = 4D(n/2) + n$

Case 1: $a=4$, $b=2$, $f(n)=n$

$\log_b a = \log_2 4 = 2$

$f(n) = n = O(n^2)$

so $T(n) = O(n^{\log_b a}) = O(n^2)$

Solution

$A(n) = 3A(n/3) + 15$

Case 1: $a=3$, $b=3$, $f(n)=15$

$\log_b a = \log_3 3 = 1$

$f(n) = 15 = O(n)$

so $T(n) = \Theta(n^{\log_b a}) = \Theta(n)$