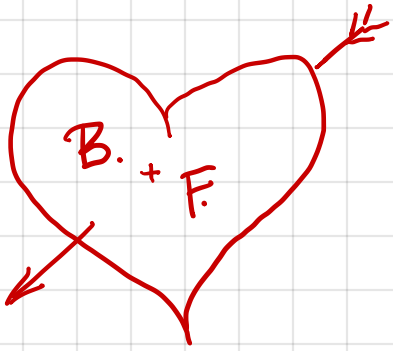
14 Feb 2018 The Bellman-Ford Shortest Path Algorithm.



Announcements

[1] Friday lunches with Prof. Kleinberg at Mattin's, 12-1. See sign-up sheet (yellow pad at front) if interested.

[2] Make-up date for Prelim 1. See Piazza post if you can't take Prelim 1 at 7:30pm-9:00pm on Tues, 2/27. The date of the make-up exam is not yet selected and we need your input!

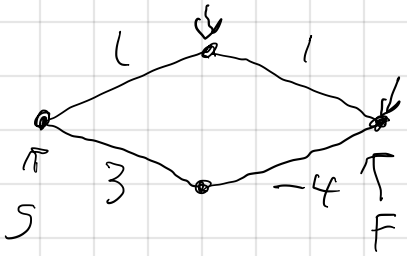
[3] Grades/comments for Homework 1 have been released on CMS. Regrade requests due a week from tomorrow.

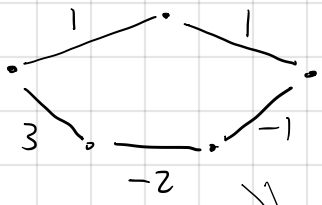
The Bellman-Ford Shortest Path Algorithm.

Finds the shortest path even when G has some edges of negative length, as long as no negative cycles.

(Q1) Why does Dijkstra's Alg fail in the presence of negative edge lengths?

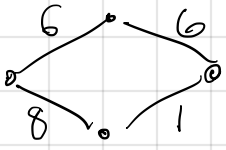
(Q2) WTF?!! Negative edge lengths!? Length is a positive number.

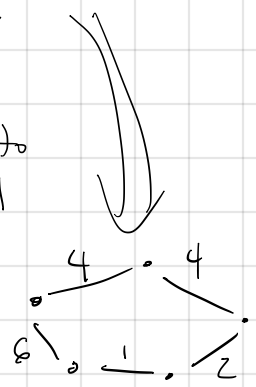




Potential solution: add a large enough constant to each edge length to make them all positive.

E.g.





Why negative edge lengths?

1. Sometimes one wants to minimize product of edge lengths rather than their sum, e.g. if the number denoting an edge's "length" represents an exchange rate.

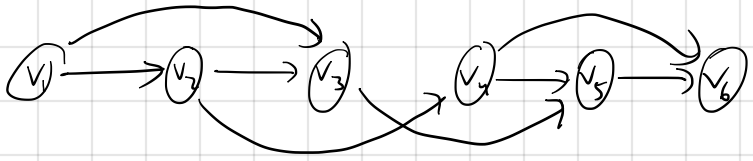
   No negative cycle assumption $\equiv$ No arbitrage

2. Sometimes one wants to find the longest path rather than the shortest. (Examples to be given later.)
   Longest path in graph with pos edge lengths
   $\equiv$ Shortest in graph with neg edge lengths.

Solving the shortest path problem in DAGs.

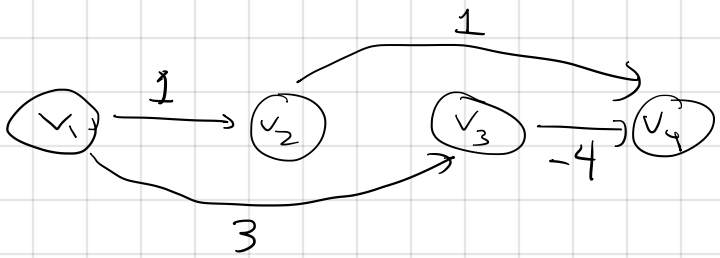
Step 1. Perform topological sort to number the vertices as $v_1, \ldots, v_n$ so that every edge (v_i, v_j) has $i < j$.

E.g.



Assume shortest path problem is to find path from v_1 to v_n. (If the actual start is v_i and actual finish is v_j then $v_1, \ldots, v_{i-1}$ can be deleted. $v_{j+1}, \ldots v_n$ can deleted.)

E.g.



Let $T[i]$ store shortest path length from v_1 to v_i.
Let $P[i]$ store shortest path itself. (as a list of vertices)

E.g. $T[4] = \min\{T[2] + 1,\ T[3] - 4\}$.

In general

$$T[k] = \min\left\{ T[j] + \text{length}(e) \;\middle|\; e = (v_j, v_k) \in E \right\}.$$

Say G has n vertices, m edges.

Algorithm:

Perform topological sort. $O(n+m)$

for $k = 1, \ldots, n$

n loop iterations

if $k = 1$

$T[k] = 0$; $P[k] = (v_1)$.

if $k > 1$

$O(\text{indegree}(v_k))$ for iteration k.

$$T[k] = \min\left\{ T[j] + \text{length}(e) \;\middle|\; e = (v_j, v_k) \in E \right\}$$

$$j^* = \arg\min\left\{ T[j] + \text{length}(e) \;\middle|\; e = (v_j, v_k) \in E \right\}.$$

$$P[k] = (P[j^*], v_k)$$

end if

end for

output $P[n]$.

Total time spent in loop:

$$O\left(n + \sum_{k=1}^{n} \text{indegree}(v_k)\right) = O(n+m)$$

Total running time $O(n+m)$ ... faster than Dijkstra!

IF your graph has cycles... instead of searching thru vertices in increasing order, search thru path lengths in increasing order.

$T[i, h]$ will attempt to store the length of the shortest path made up of $\leq h$ hops from v_1 to v_i.