16 Feb 2018 — Bellman-Ford, reductions, and RNA.

Recap of Bellman-Ford algorithm for DAGs...

Say G has n vertices, m edges.

Algorithm:

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

for $k = 1, \ldots, n$

n loop iterations

$O(1 + \text{indegree}(v_k))$ for iteration k.

  if $k = 1$

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

  if $k > 1$

    $T[k] = \min \{ T[j] + \text{length}(e) \mid e = (v_j, v_k) \in E \}$

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

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

  end if

end for

output $P[n]$.

Total time spent in loop:

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

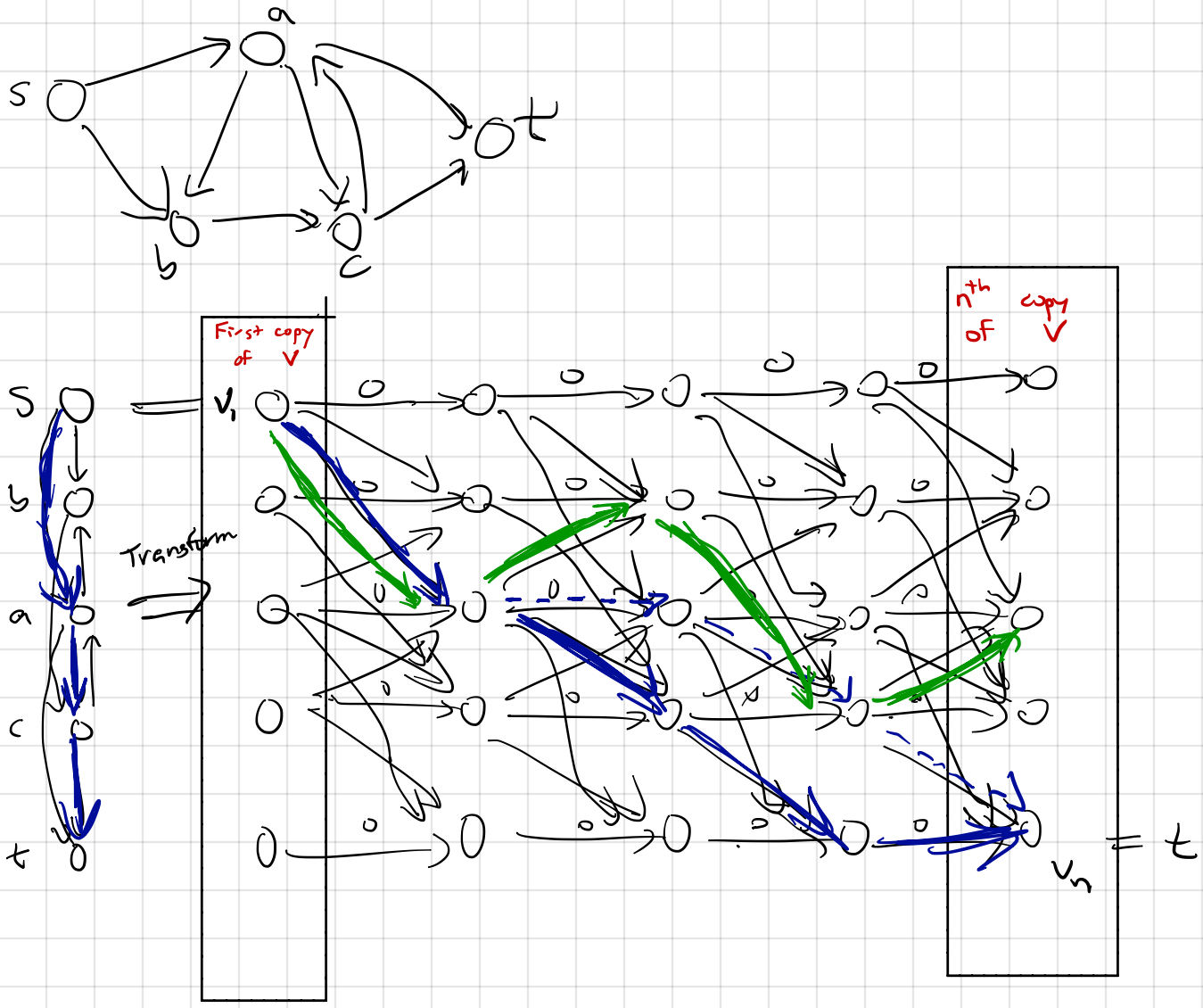
Total running time $O(n+m)$... faster than Dijkstra!

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.

If you want to see pseudocode for filling in this table, look at the textbook.

Instead here's a reduction from general directed graphs without negative length cycles to DAGs...



Form a larger DAG with

- vertices v_{xi} for all $x \in V$, $i \in \{1, \ldots, n\}$
- edges $(v_{xi}, v_{x,i+1})$ of length 0 $\forall x \in V$, $i < n$.
- edges $(v_{xi}, v_{y,i+1})$ of length $\ell(x,y)$ $\forall (x,y) \in E$, $i < n$.

This DAG has n^2 vertices and $O(mn)$ edges.
Run Bellman-Ford on the DAG to find a shortest path from v_{s1} to v_{tn}.
Say the path is

$$v_{s1}, v_{x_2,2}, v_{x_3,3}, \ldots, v_{x_{n-1},n-1}, v_{t,n}$$

Then $s, x_2, x_3, \ldots, x_{n-1}, t$ with repetitions eliminated is a shortest s-t path in G.
This corresponds to a path and not a walk (sequence in which vertices may appear more than once) because G has no negative-length cycles.

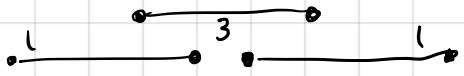
Running time of Bellman-Ford in general graphs. $O(mn)$.

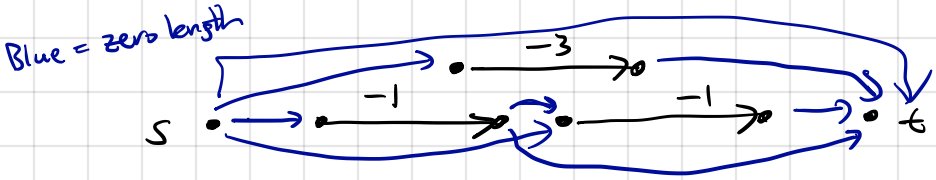
(# vertices, # edges)

Reasoning: We created a graph with n^2 vertices, mn edges, then ran an alg with $O(n^2 + nm)$ running time.
If $n^2 >> nm$ it means G has many isolated vertices.
Preprocess to eliminate those in $O(n)$ time.
So can be sure that big graph, after preproc, has only $O(\min\{n^2, mn\})$ vertices, $O(mn)$ edges.
This takes care of $O(n^2)$ term.
Running time is simply $O(mn)$.

Reducing other problems to shortest path.

Ex 1. Weighted Interval Scheduling





Graph has:
- source s, sink t. Edge (s,t)
- vertices s_i, f_i for each interval i
- edge (s_i, f_i) length $-w_i$.
- edge (s, s_i) $\forall i$
- edge (f_i, t) $\forall i$
- edge (f_i, s_j) $\forall i,j$ s.t. $f_i < s_j$.

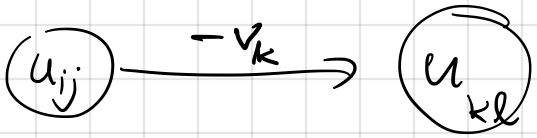
} zero cost

Feasible interval schedules are in 1:1 correspondence with s-t paths in this graph. This correspondence transforms combined weight to negative length.

Ex. 2 Knapsack:

Vertex u_{ij} corresponds to knapsack with weight j containing no items numbered higher than i.

edges correspond to putting one more item in. (numbered higher than the items already in the knapsack)



"put k in knapsack"

Edge exists when $i < k$ and $j + w_k = l \leq W$.

Graph has $O(nW)$ vertices, $O(n^2W)$ edges.