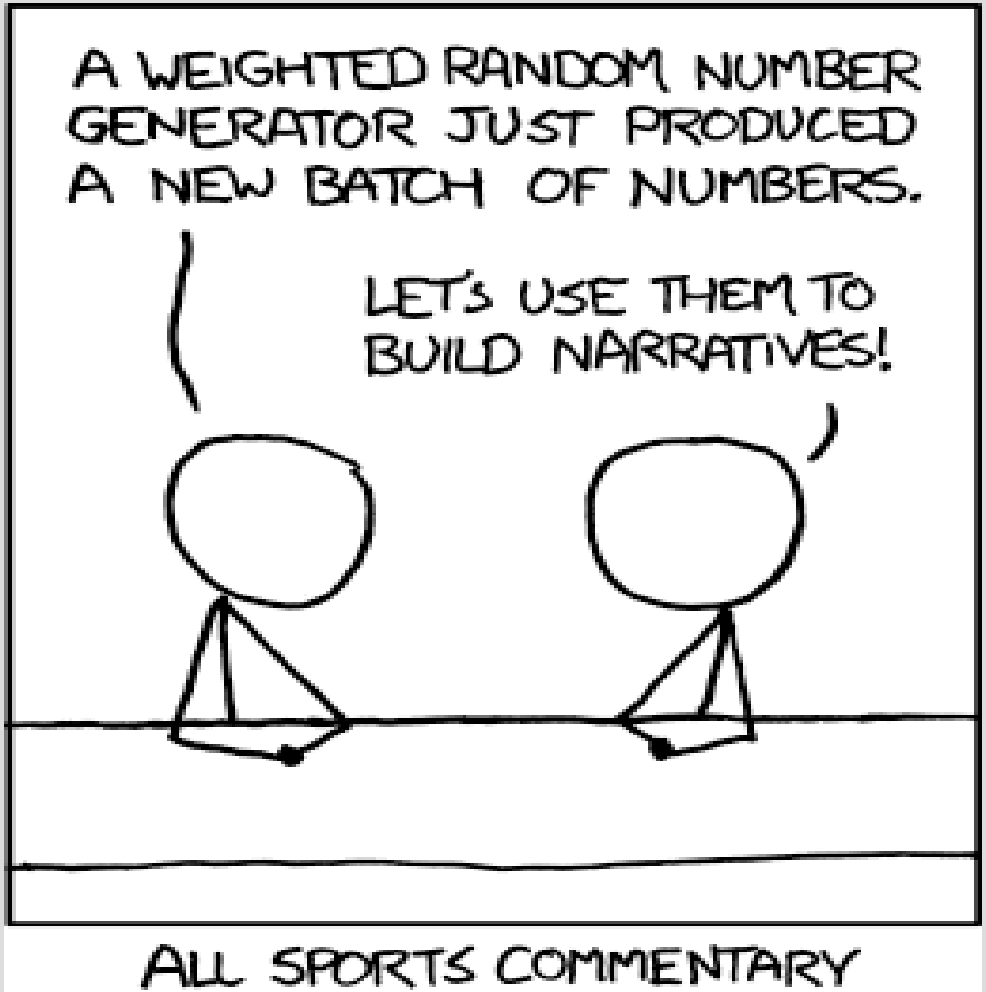
Weighted graphs



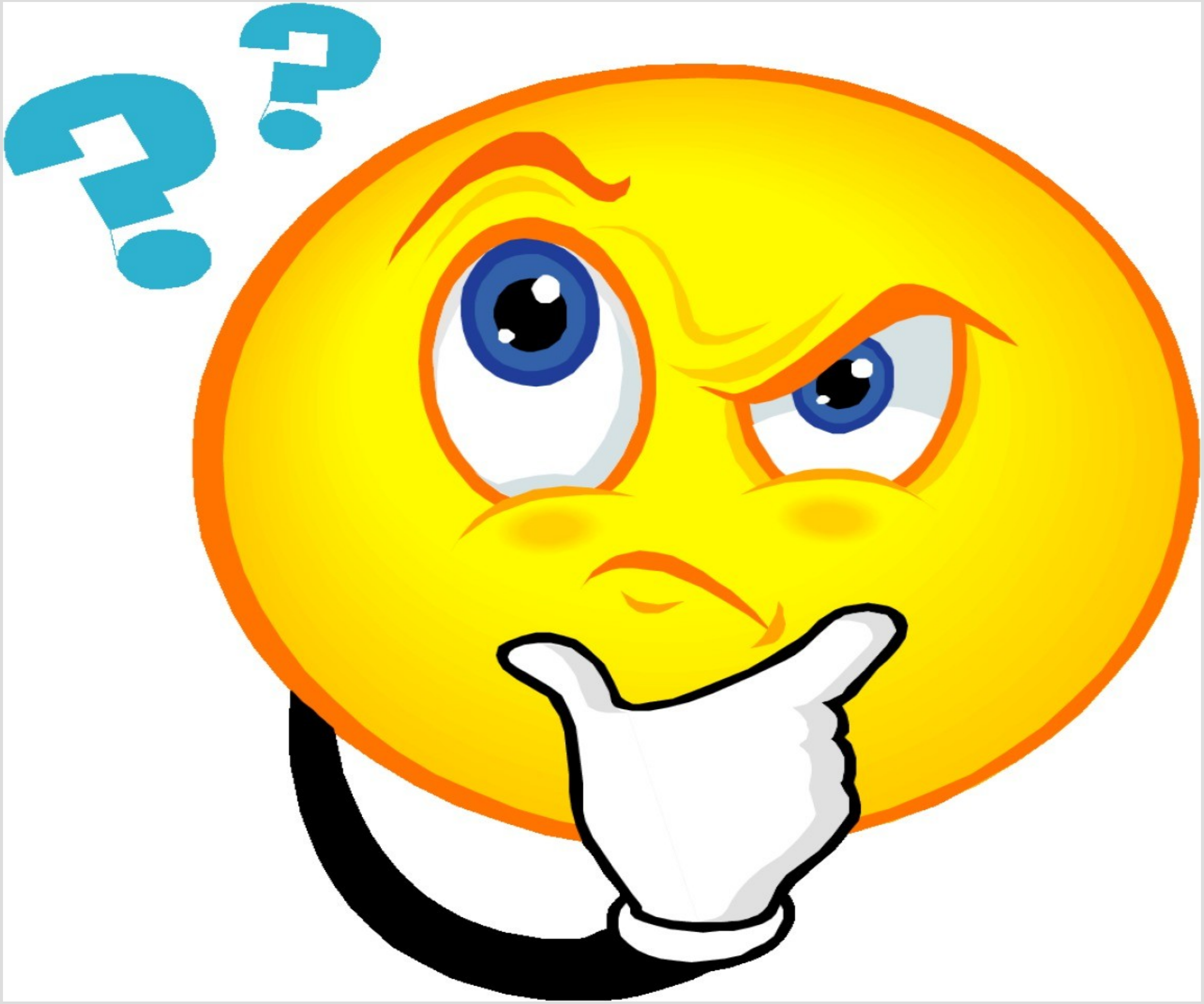
Weighted graphs

So far we have only considered weighted graphs with "weights $\geq$ 0" (Dijkstra is a super-star here)

Now we will consider graphs with any integer edge weight (i.e. negative too)

Cycles

Does a shortest path need to contain a cycle?



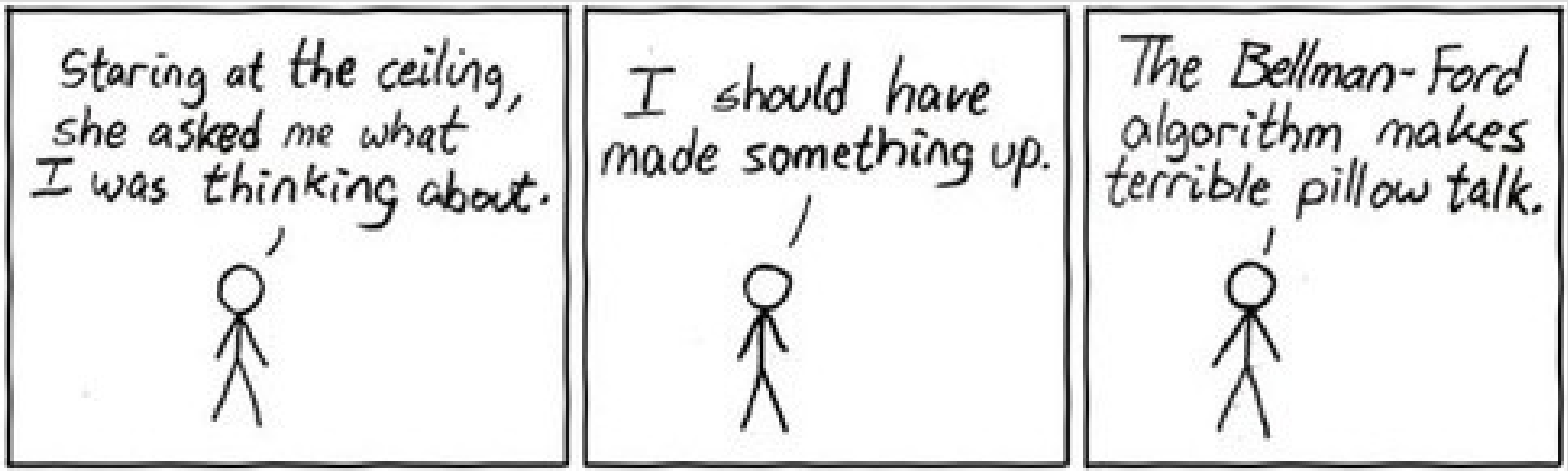
Cycles

Does a shortest path need to contain a cycle?

No, case by cycle weight:
positive: why take the cycle?!
zero: can delete cycle and find same
length path
negative: cannot ever leave cycle

Bellman-Ford

One of the few “brute force” algorithms that got a name



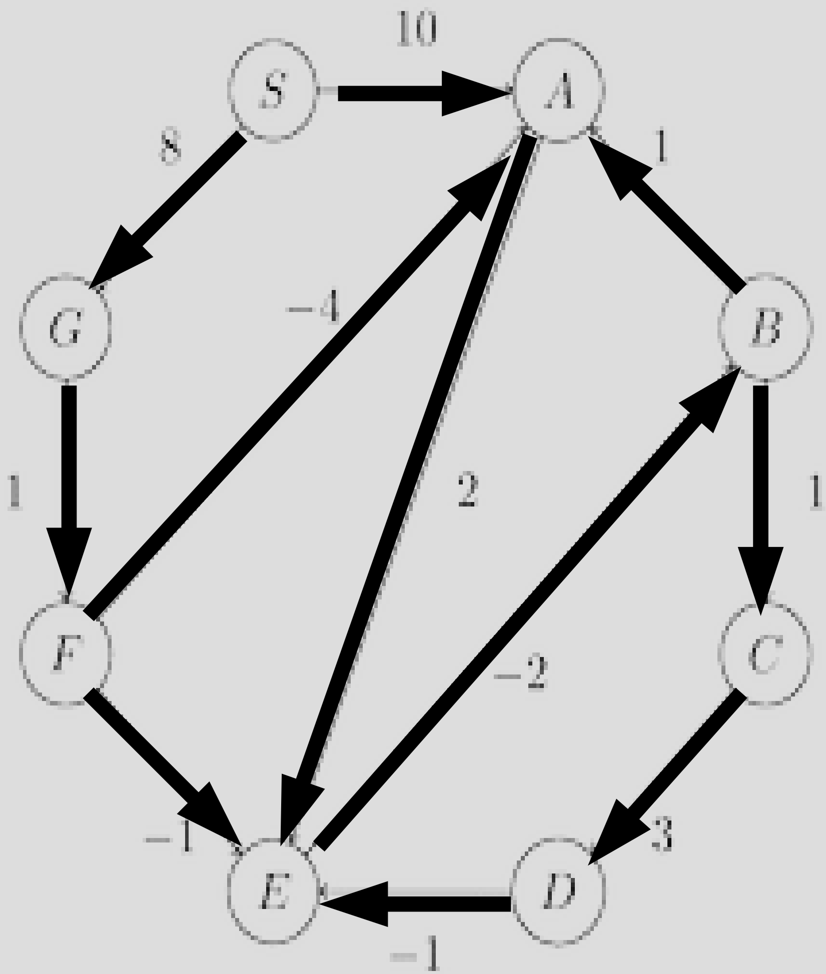
Idea:

1. Relax every edge (yes, all)
2. Repeat step 1 |V| times (or |V|-1)

Bellman-Ford

```
BF(G, w, s)
initialize graph
for i=1 to |V| - 1
   for each edge (u,v) in G.E
      relax(u,v,w)
for each edge (u,v) in G.E
   if v.d > u.d+w(u,v): return false
return true
```

Bellman-Ford



	Iteration							
Node	0	1	2	3	4	5	6	7
S	0	0	0	0	0	0	0	0
A	∞	10	10	5	5	5	5	5
B	∞	∞	∞	10	6	5	5	5
C	∞	∞	∞	∞	11	7	6	6
D	∞	∞	∞	∞	∞	14	10	9
E	∞	∞	12	8	7	7	7	7
F	∞	∞	9	9	9	9	9	9
G	∞	8	8	8	8	8	8	8

Bellman-Ford

Correctness: (you prove)

After BF finishes: if $\delta(s,u)$ exists, then $\delta(s,u) = u.d$

Bellman-Ford

Correctness: (you prove)

After BF finishes: if $\delta(s,u)$ exists, then $\delta(s,u) = u.d$

Relxation property 5, as every edge is relaxed |V|-1 times and there are no loops

Bellman-Ford

Correctness: returns false if neg cycle

Suppose neg cycle: $c = \langle v_0, v_1, \ldots v_k \rangle$

then $w(c) < 0$, suppose BF return true

Then $v_i.d \leq v_{i-1}.d + w(v_{i-1}, v_i)$

sum around cycle c:

$\sum^k_{i=1} v_i.d \leq \sum^k_{i=1} (v_{i-1}.d + w(v_{i-1}, v_i))$

$\sum^k_{i=1} v_i.d \leq \sum^k_{i=1} v_{i-1}.d$ as loop

Bellman-Ford

Correctness: returns false if neg cycle

$\sum^{k}_{i=1} v_i.d \leq \sum^{k}_{i=1} (v_{i-1}.d + w(v_{i-1},v_i))$

$\sum^{k}_{i=1} v_i.d = \sum^{k}_{i=1} v_{i-1}.d$ as loop

so $0 \leq \sum^{k}_{i=1} w(v_{i-1},v_i)$

but $\sum^{k}_{i=1} w(v_{i-1},v_i) = w(c) < 0$

Contradiction!

All-pairs shortest path

So far we have looked at:
Shortest path from a specific start to any other vertex

Next we will look at:
Shortest path from any starting vertex to any other vertex
(called "All-pairs shortest path")

Johnson's algorithm

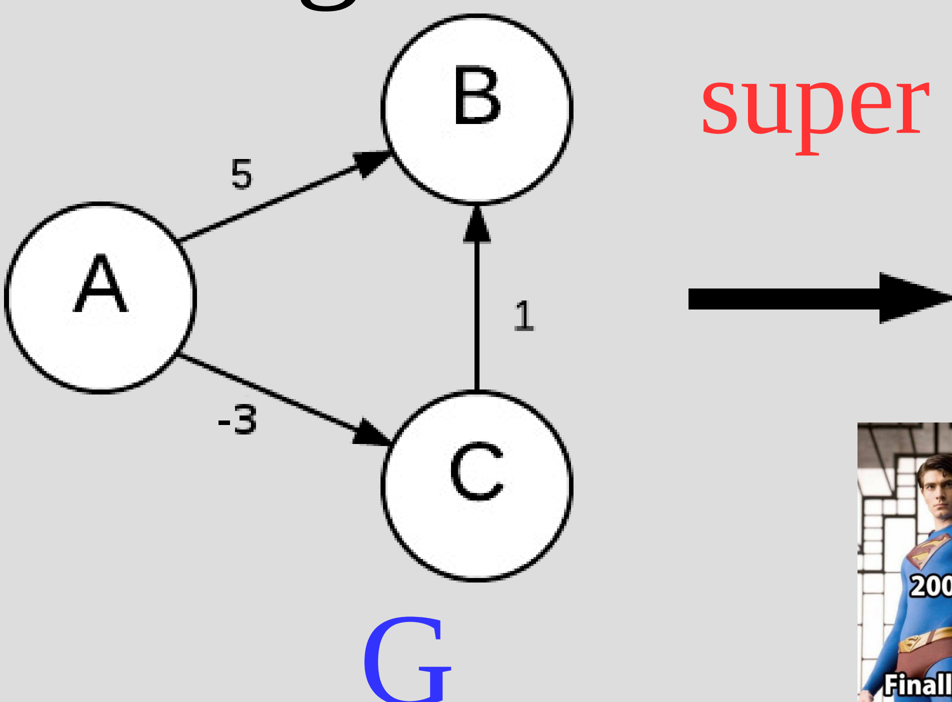
We will start by doing something a little funny

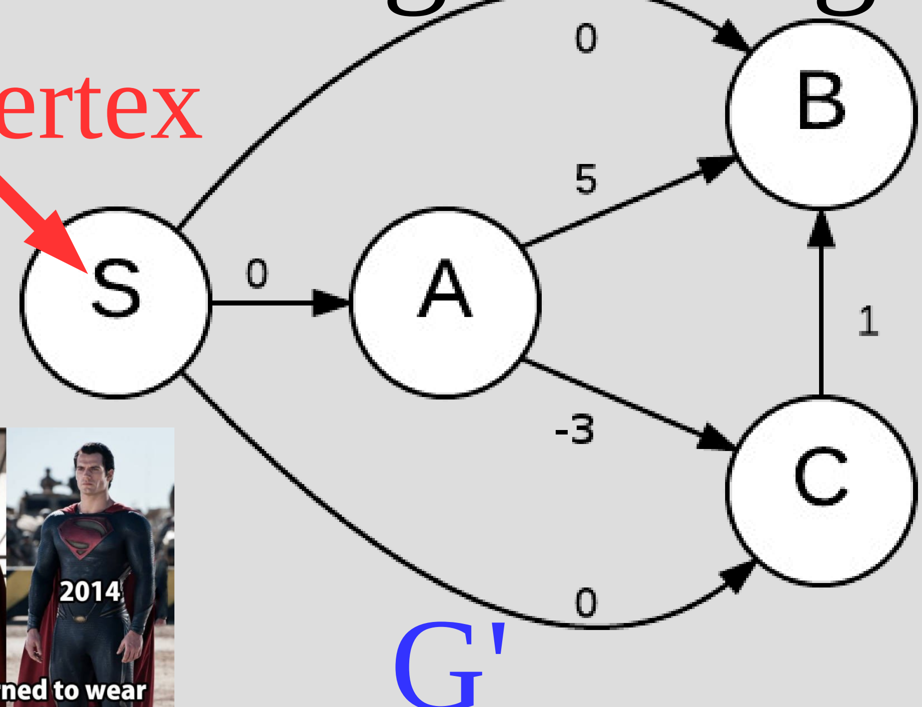
(This will be the most efficient for graphs without too many edges)

To compute all-pairs shortest path on G, we will modify G to make G'

Johnson's algorithm

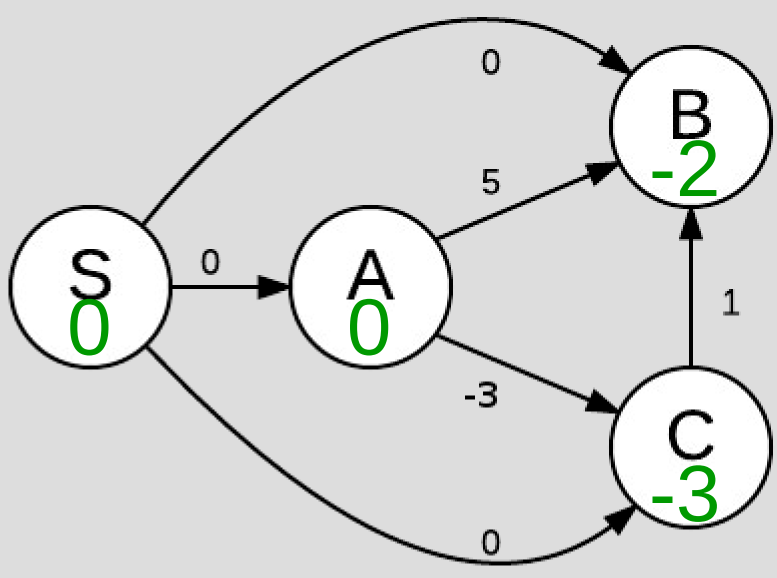
To make G', we simply add one "super vertex" that connects to all the original nodes with weight 0 edge





Johnson's algorithm

Next, we use Bellman-Ford (last alg.) to find the shortest path from the "super vertex" in G' to all others (shortest path distance, i.e. d-value)



Johnson's algorithm

Then we will “reweight” the graph:

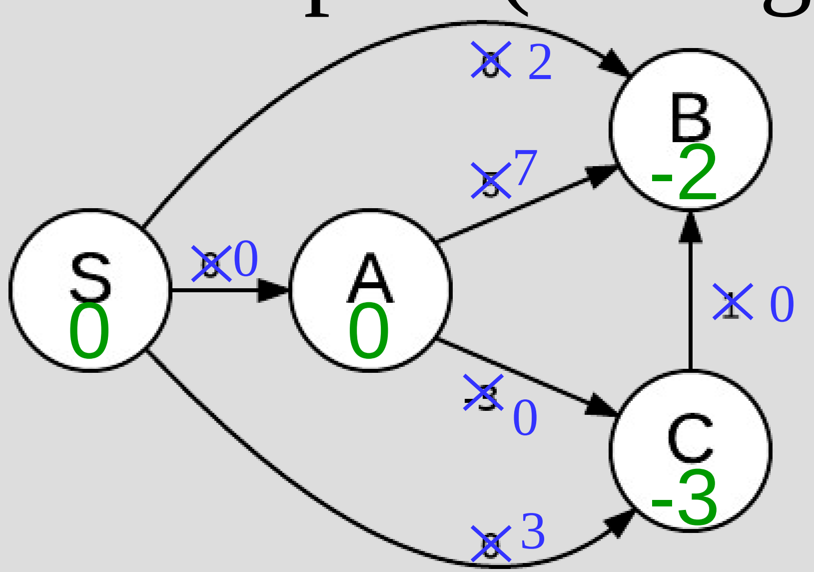
$$\hat{w}(u,v) = w(u,v) + h(u) - h(v)$$

old weight

d-value in vertex

(u,v) is a vertex pair (an edge from u to v)

new weight



Johnson's algorithm

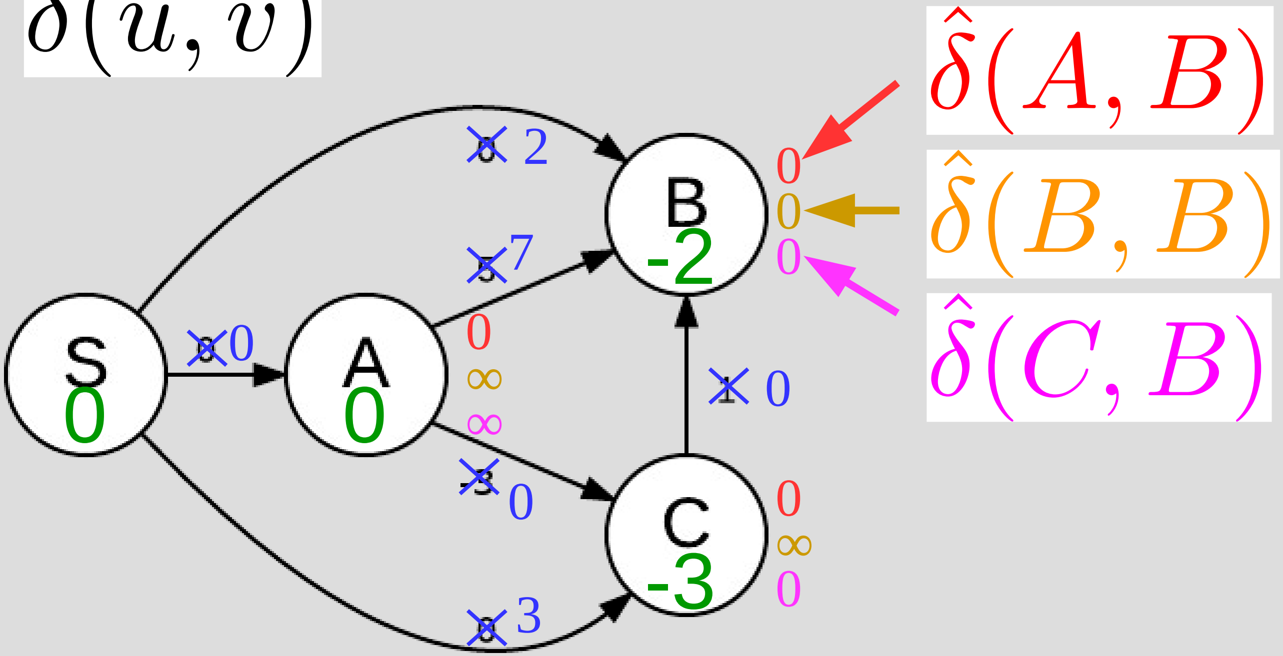
Next, we just run Dijkstra's starting at each vertex in G (starting at A, at B, and at C for this graph)

Call these $\hat{\delta}(u,v)$

start A

start B

start C



Johnson's algorithm

Finally, we “un-weight” the edges:

$$\delta(u,v) = \hat{\delta}(u,v) - h(u) + h(v)$$

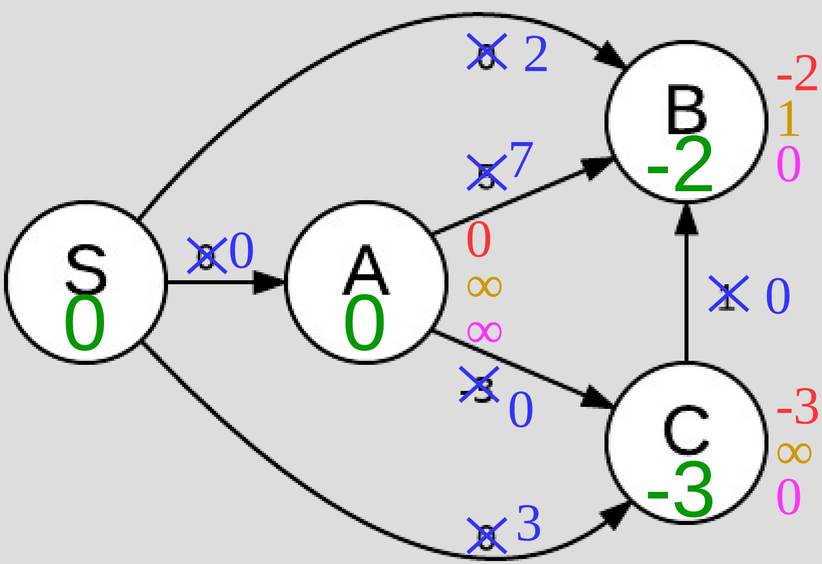
last time +

last time -

start A

start B

start C



Johnson's algorithm

```
Johnson(G)
    Make G'
    Use Bellman-Ford on G' to get h
    (and ensure no negative cycle)
    Reweight all edges (using h)
    for each vertex v in G
        Run Dijkstra's starting at v
    Un-weight all Dijkstra paths
    return all un-weighted Dijkstra paths(matrix)
```

Johnson's algorithm

Runtime?

Johnson's algorithm

Runtime:

Bellman-Ford = $O(|V|\,|E|)$

Dijkstra = $O(|V| \lg |V| + E)$

Making G' takes $O(|V|)$ to add edges

Bellman-Ford run once

weight edges = $O(|E|)$

unweighting paths = $O(|V|^2)$

Dijkstra run |V| times ← most costly

Johnson's algorithm

Runtime:

Bellman-Ford = $O(|V|\ |E|)$

Dijkstra = $O(|V| \lg |V| + E)$

$O(|V|) + O(|V|\ |E|) + O(|E|) + O(|V|^2)$
$+ |V|\ O(|V| \lg |V| + E)$

$= O(\ |V|^2 \lg |V| + |V|\ |E|\ )$

Correctness

The proof is easy, as we can rely on Dijkstra's correctness

We need to simply show:

(1) Re-weighting in this fashion does not change shortest path

(2) Re-weighting makes only positive edges (for Dijkstra to work)

Correctness

(1) Re-weighting keeps shortest paths

Here we can use the optimal sub-structure of paths:

If $\delta(u,x) =< v_0, v_1, ...v_k >$ with $v_0 = u$ and $v_k = x$

then $\delta(u,x) = \delta(v_0,v_1) + \delta(v_1,v_2) + ... + \delta(v_{k-1},v_k)$

But as (v_i, v_{i+1}) is the edge taken:

$\delta(v_i, v_{i+1}) = w(v_i, v_{i+1})$

Correctness

(1) Re-weighting keeps shortest paths

Then by definition of $\hat{\delta}(u, x)$

$$\begin{aligned}\hat{\delta}(u,x) &= \hat{w}(path)\\ &= \sum_{i=1}^{k} \hat{w}(v_{i-1}, v_i)\\ &= \sum_{i=1}^{k} w(v_{i-1}, v_i) + h(v_{i-1} - h(v_i)\\ &= h(v_0) - h(v_k) + \sum_{i=1}^{k} w(v_{i-1}, v_i)\\ &= h(v_0) - h(v_k) + \delta(u,x)\end{aligned}$$

Correctness

(1) Re-weighting keeps shortest paths

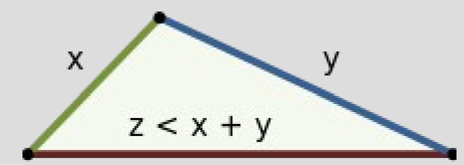
Thus, the shortest path is just offset by "$h(v_0) - h(v_k)$" (also any path)

As v_0 is the start vertex and v_k is the end, so vertices along the path have no influence on $\hat{\delta}(u, x)$ (same path)

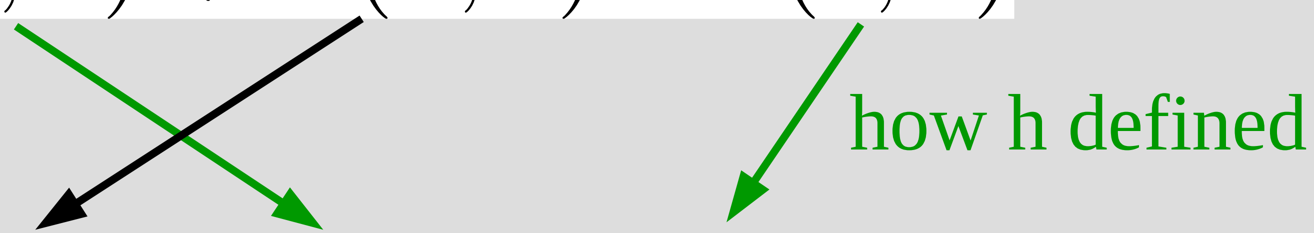
Correctness

(2) Re-weighting makes edges > 0

One of our "relaxation properties" is the "triangle inequality"

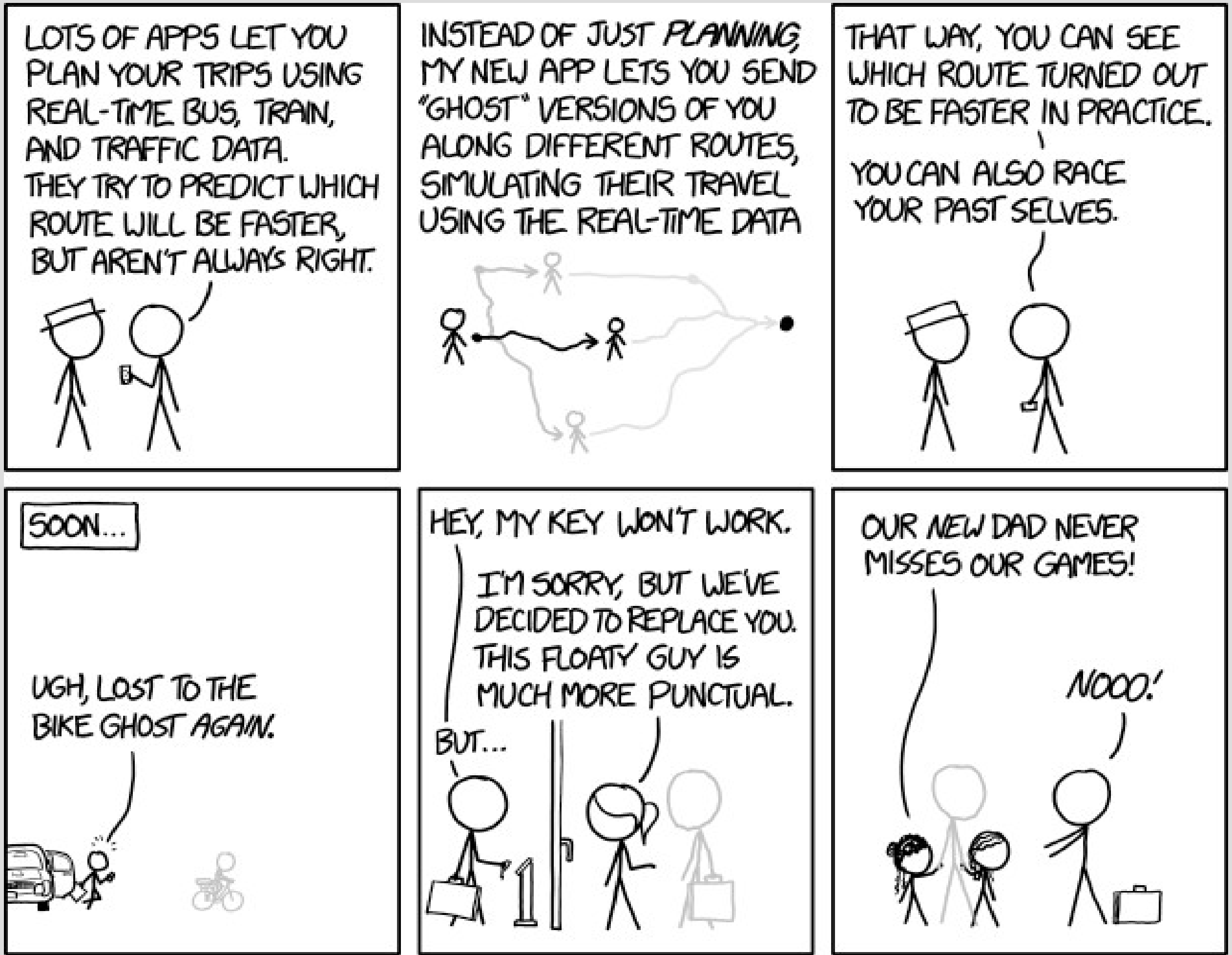


$$\delta(s,v) \leq \delta(s,u) + w(u,v)$$
$$= 0 \leq \delta(s,u) + w(u,v) - \delta(s,v)$$



$$0 \leq w(u,v) + h(u) - h(v) = \hat{w}(u,v)$$

All-Pairs Shortest Paths



TL;DR dynamic programming

What are two ways you can compute the Fibonacci numbers?

$F_n = F_{n-1} + F_{n-2}$
with $F_0=0$, $F_1=1$

Which way is better?

TL;DR dynamic programming

One way, simply use the definition

Recursive:

```
F(n):
    if(n==1 or n==0)
        return n
    else return F(n-1)+F(n-2)
```

TL;DR dynamic programming

Another way, compute F(2), then F(3)
... until you get to F(n)

Bottom up:

```
A[0] = 0
A[1] = 1
for i = 2 to n
    A[i] = A[i-1] + A[i-2]
```

TL;DR dynamic programming

This second way is *<u>much</u>* faster

It turns out you can take pretty much any recursion and solve it this way (called “dynamic programming”)

It can use a bit more memory,
but much faster

TL;DR dynamic programming

How many multiplication operations does it take to compute:

x^4?

x^{10}?

TL;DR dynamic programming

How many multiplication operations does it take to compute:

x^4? Answer: 2

x^{10}? Answer: 4

TL;DR dynamic programming

Can compute x^4 with 2 operations:

$x^2 = x * x$ (store this value)

$x^4 = x^2 * x^2$

Save CPU by using more memory!

Can compute x^n using $O(\lg n)$ ops

Also true if x is a matrix

Shortest paths using matrices

Any sub-path ($p_{x,y}$) of a shortest path ($p_{u,v}$) is also a shortest path

Thus we can recursively define a shortest path $p_{0,k} = <v_0, ..., v_k>$, as:

$w(p_{0,k})=\min_{“k-1”}(w(p_{0,“k-1”})+w(“k-1”,k))$

Shortest paths using matrices

Thus a shortest path (using less than m edges) can be defined as:

$L^m = l^m_{i,j} = \min_k(l^{m-1}_{i,k} + l^1_{k,j})$,
where L^1 is the edge weights matrix

Can use dynamic programming to find an efficient solution

Shortest paths using matrices

L^m is not the m^{th} power of L, but the operations are very similar:

$L^m = l^m_{i,j} = \min_k(l^{m-1}_{i,k} + l^1_{k,j})$ // ours

$L^m = l^m_{i,j} = \sum_k(l^{m-1}_{i,k} * l^1_{k,j})$ //real times

Thus we can use our multiplication saving technique here too!

(see: MatrixAPSPmult.java)

Shortest paths using matrices

```
All-pairs-shortest-paths(W)
L(1) = W, n = W.rows, m  = 1
while m < n
   L(2m) = ESP(L(m), L(m))
   m = 2m
return L(m)
```

(ESP is L min op on previous slide)

Shortest paths using matrices

Runtime:
$|V|^3 \lg |V|$

Correctness:
By definition (brute force with some computation savers)

Floyd-Warshall

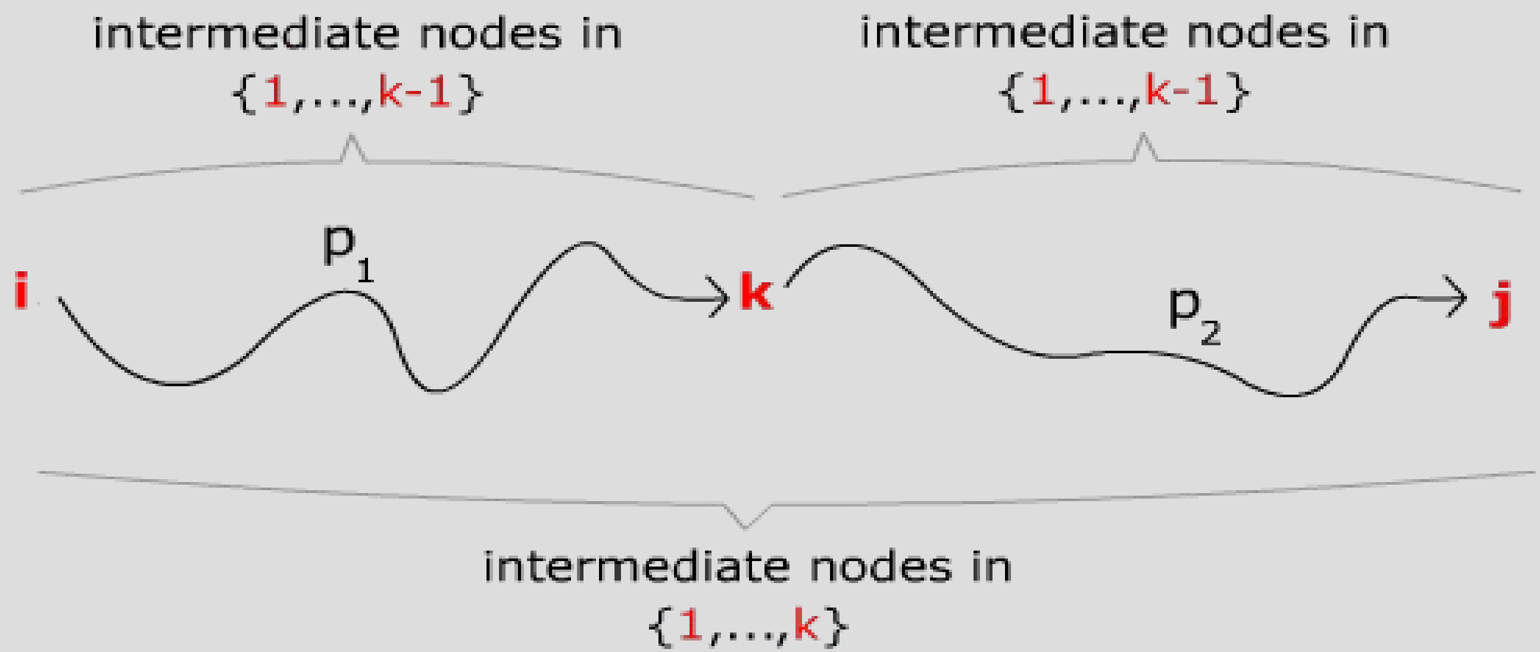
The Floyd-Warshall is similar but uses another shortest path property

Suppose we have a graph G, if we add a single vertex k to get G'

We now need to recompute all shortest paths

Floyd-Warshall

Either the path goes through k, or remains unchanged



$$d^{k}_{i,j} = \min\left(d^{k-1}_{i,j},\ d^{k-1}_{i,k} + d^{k-1}_{k,j}\right)$$

Floyd-Warshall

Floyd-Warshall(W) // dynamic prog

$d^0_{i,j} = W_{i,j}$, $n = W.rows$

for k = 1 to n

  for i = 1 to n

    for j = 1 to n

      $d^k_{i,j} = \min(d^{k-1}_{i,j}, d^{k-1}_{i,k} + d^{k-1}_{k,j})$

Floyd-Warshall

Runtime:
$O(|V|^3)$

Correctness:
Again, by definition of shortest path