Correctness of Bellman-Ford:

let $len(i,v)$ denote the length of a shortest path from s to v that contains at most i edges. Since the sh. path from s to v contains $\leq n-1$ edges, it's sufficient to prove that after i iterations $len(v) \leq len(i,v)$

By induction on i

Base case: $i=0$

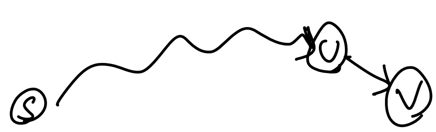
$$len(s) = 0 \leq len(0,s) = 0$$
$$len(v \neq s) = +\infty = len(0, v\neq s)$$

Inductive hypothesis: $len(v) \leq len(k,v) \;\; \forall\, 1 \leq k < i$

Take $i \geq 1$ and a shortest path from s to v with $\leq i$ edges. Let (u,v) be the last edge of this path. Then

$$len(i,v) = w(u,v) + \underbrace{len(i-1,u)}$$

why? by contradiction



By the ind. hyp. $len(u) \leq len(i-1, u)$
In the i-th iteration we update

$$len(v) = \min\{len(v), \underbrace{len(u) + w(u,v)}\}$$

$$\leq len(i-1,u) + w(u,v)$$
$$= len(i,v)$$

$$\leq len(i,v)$$ as desired

All-Pairs Shortest Paths (APSP)

input: a directed, weighted graph $G=(V,E)$
output: one of the following

a) $dist(u,v)$ for every ordered vertex pairs
b) a declaration that G contains a negative cycle

Obvious solution: invoke B-F once for every vertex

$\longrightarrow O(m \cdot n^2)$ very high

Can we do better?

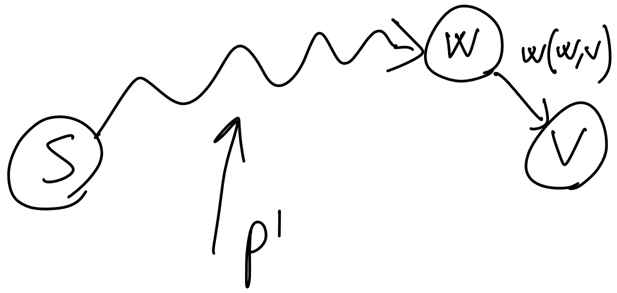
Yes, using dynamic programming

Outline:

1) the B-F algorithm has a dynamic programming formulation
2) (we won't see) one can adapt that formulation to APSP, obtaining an $O(mn^2)$ algorithm; an improved formulation can be made to run in $O(n^3 \log n)$
3) a different dyn. prog. strategy gives an $O(n^3)$ algorithm

Bellman-Ford via dynamic programming

what are the subproblems here?



← P = shortest s-v path

obs.: P' is a shortest path (to a ≠ destination) with fewer edges than P then P' can be interpreted as a solution to a smaller subproblem

$\Rightarrow$ idea: introduce a parameter i that restricts the maximum number of edges allowed in a path, with smaller subproblems having smaller edge budgets

$\hookrightarrow$ serves as a measure of subproblem size

Subproblems: compute $len(i, v)$, the length of a shortest path from s to v that contains at most i edges. (If no such path exists, define $len(i,v)$ as $+\infty$.) $\longrightarrow O(n^2)$ subproblems

Obs: every subproblem works with the full input; the idea is to control the allowable size of the output.

Bellman-Ford recurrence

$$len(i,v) = \begin{cases} 0 & i=0 \text{ and } v=s \\ +\infty & i=0 \text{ and } v\neq s \\ \min \begin{cases} len(i-1,v) \\ \min\limits_{(u,v)\in E} \{len(i-1,u)+w(u,v)\} \end{cases} & \text{otherwise} \end{cases}$$

(It's easy to transform a dyn.-prog. evaluation of this recurrence into our original formulation of Bellman-Ford)

This formulation can be adapted to APSP $\to O(n^3 \log n)$

The Floyd-Warshall algorithm

Idea: go one step further: instead of restricting the number of edges allowed in a solution, restrict the identities of the vertices that are allowed in a solution. (In other words, now path can pass through only certain vertices)

Let's define the subproblems:

- call the vertices $1, 2, \ldots, n$

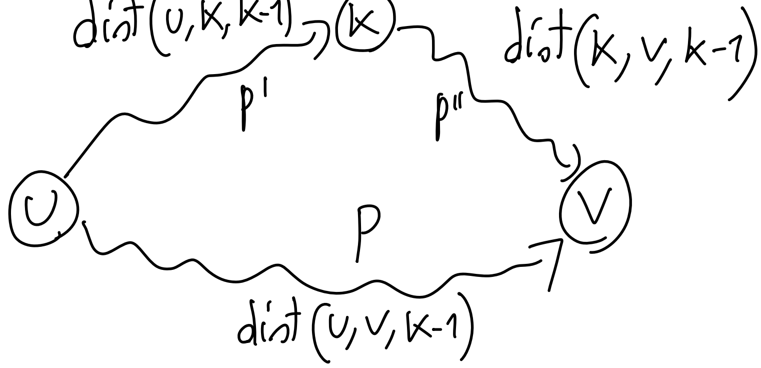
Compute $dist(u, v, k)$ = length of a shortest path from u to v that uses only vertices from $\{1, 2, \ldots, k\}$ as internal (i.e., not u or v) vertices, and that does not contain a directed cycles. (If no such path exists, define $dist(u,v,k)$ as $+\infty$.)

$k \rightarrow$ measures the subproblem size

$\rightarrow O(n^3)$ subproblems

Algorithm: expand the set of allowed internal vertices, one vertex at a time, until this set is V.

Payoff of defining subproblems in this way: only 2 candidates for the optimal solution to a subproblem, depending on whether it uses vertex k or not:



$\Rightarrow$ $O(1)$ work per subproblem

$\Rightarrow$ Complexity : $O(n^3)$

Floyd-Warshall (G)

label the vertices $V = \{1, 2, \dots, n\}$ arbitrarily

// subproblems (k indexed from 0)

$A = n \times n \times (n+1)$ array

// base cases ($k=0$)

```
for u = 1 to n do
    for v = 1 to n do
        if u = v then A[u,v,0] = 0
        else if (u,v) ∈ E then A[u,v,0] = w(u,v)
        else A[u,v,0] = +∞
```

// solve all subproblems

```
for k = 1 to n do
    for u = 1 to n do
        for v = 1 to n do
            A[u,v,k] = min{A[u,v,k-1],
                A[u,k,k-1] + A[k,v,k-1]}
```

\\ check for a negative cycle
for $v = 1$ to n do
  if $A[v, v, n] < 0$ then
    return "G contains a negative cycle"

Is there a truly-subcubic algorithm for APSP?

$O(n^{3-\varepsilon})$ for some constant $\varepsilon > 0$