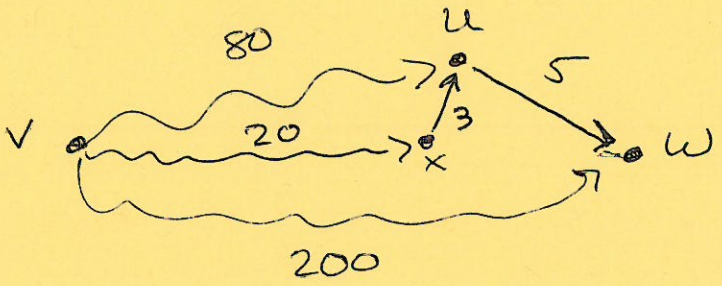
25 Sept 2019

Bellman-Ford Example:



$d(v,u) = 80$

$d(v,w) = 85$

End an iteration

(x,u) then (u,w)

$d(v,w) = 23$

$d(v,w) = 28$

if this happens

$\Rightarrow$ ~~must~~ $i < n-1$

$\Rightarrow$ must go through the loop at least one more time!

But

(u,w) then (x,u)

$d(v,u) = 23$

$d(v,w) = 85$

note:
- Link length = # edges
- length of path $p = \sum_{e \in p} w(e)$

Bellman-Ford Algorithm

Algorithm 1 BELLMAN-FORD(G, ω, v)

Input: graph $G = (V, E)$, weight function $\omega: E \to \mathbb{R}$, and initial vertex $v \in V$
Output: detects negative cycle if one exists. Otherwise, returns shortest distance to every vertex in G from v.

```
1: dist ← real-valued array of length |V|, and indexed by V.
2: dist[v] ← 0
3: i ← 1
4: while i < n do
5:     for (u, w) ∈ E do
6:         dist(w) ← min{dist(w), dist(u) + ω(u, w)}
7:     end for
8:     i++
9: end while
10: for (u, w) ∈ E do
11:     if dist(w) > dist(u) + ω(u, w) then
12:         return "Negative Cycle Detected"
13:     end if
14: end for
15: return dist
```

n = |V|

consider uninitialized values = ∞

repeats n-1 times

adding that Edge helps!

path from v to w already "known" + shortest

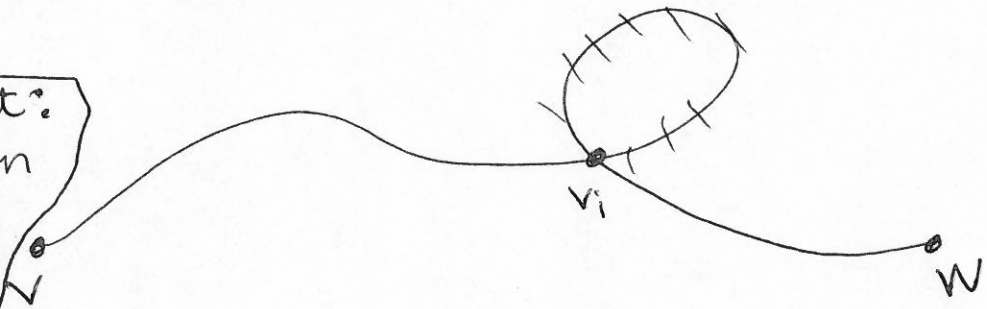
neg cycles bad b/c can keep going around to get "shorter" + shorter" path.

1. Explain to each other, in words, (1) what is the problem; (2) how this algorithm works. (No need to write down answer for this one).
2. Work through a small example.
3. What is the runtime?
4. What is the loop invariant of each of the loops?

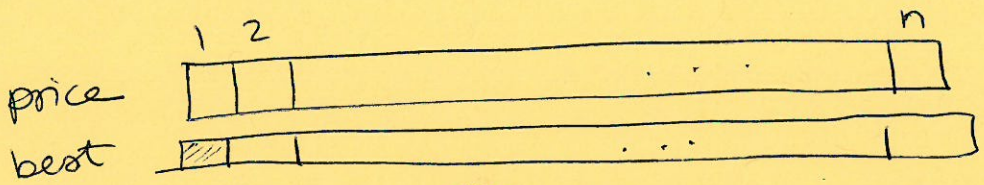
First time through while loop (lines 4-9), we know that
$dist(u) \leq w(v, u)$
in other words, we've considered all paths of link-length 1 so far (+ maybe a couple more)

Second time through... I've considered all paths of link-length ≤ 2 (ie, comprised of exactly 2 or less edges)

Loop invariant:
dist(w) ≤ length of SP considering paths of link length i or less, ∀w ∈ V



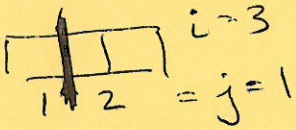
Rod cutting



RodCut (n, price) array of length n.

best ← array of length n, init to $-\infty$

for i = 1 ... n
  best(i) ← price(i)
  for j = 0 ... i
    best(i) = max (best(i), best(i−j) + best(j))
  end for
end for
return best(n)



constant time lookup

Runtime: $\Theta(n^2)$

Naive / Recursive way: $\Theta(2^n)$

note: there are exactly 2^{n-1} ways to cut the rod. why?