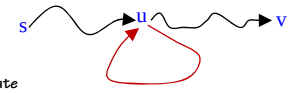
Termination

- We can terminate at phase k if, for all v, $d^k(v) = d^{k-1}(v)$, since no more changes will happen in $d^k(v)$ for larger values of k.
- Terminates in n-1 phases if no negative cycles (might terminate earlier)
  - » Proof in the textbook.
  - » Main idea:
    - $d^k(v)$ –distance to reach v in up to k hops.
    - $d^k(v)$ does not increase with k, decreased with each update
    - If for some k>n-1 it is updated, then $d^k(v) < d^{n-1}(v)$
    - If more than n-1 hops are used, then path is not simple
    - Implies a negative-cost cycle along this path



- Total running time: $O(nm)$
- Total "extra space" (beyond the graph structure): $O(n)$
- Basis for RIP protocol

Bellman-Ford

- If negative cycle exists then no termination, even in n-1 phases:
  Proof:
  - Consider edge $v_i v_{i+1}$ along the cycle at termination.
  - If terminated, then for all edges $v_i v_{i+1}$ on the cycle: $d(v_{i+1}) \le d(v_i) + w(v_i v_{i+1})$
  - Sum up over all $v_i v_{i+1}$ edges:

$$\sum_i d(v_{i+1}) \le \sum_i d(v_i) + \text{(weight of the cycle)}$$
$$\Rightarrow \text{weight of the cycle} \ge 0$$

- What if cycle is unreachable from s ?
  - » Add a supernode s'
  - » Connect s' to all nodes, use 0-cost on these extra edges.
  - » No new cycles created
  - » Run Bellman-Ford, check termination in n steps.

All-pairs shortest paths

- First approach:
  - Use Bellman-Ford for each source node
  - n sources, $O(nm)$ per run, Total = $O(n^2m)$
  - For dense graph, $m=\Theta(n^2)$, so total is $O(n^4)$, slow !
- Second try:
  - Use w_{ij} to denote the weights matrix, zeroize diagonal for convenience
  - Infinity if no ij edge
  - Define L_{ij}^k =distance from node i to node j in k or less hops

$$L_{ij}^{k+1} = \min(L_{ij}^k, \min_{1\le v\le n}\{L_{iv}^k + w_{vj}\})$$
$$= \min_{1\le v\le n}\{L_{iv}^k + w_{vj}\} \qquad \text{since } w_{jj} = 0 \text{ for all } j$$

- Still $O(n^4)$... (Why ??)
- Can we do better ?

More all-pairs shortest paths

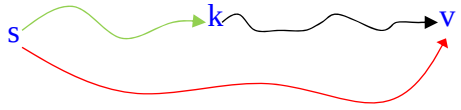
- Lets look again at the formula:

$$L_{ij}^{k+1} = \min_{1\le q\le n}\{L_{iq}^k + w_{qj}\} \qquad \left[L\times W\right]_{i,j} = \sum_{k=1}^{n} L_{ik}W_{kj}, \quad \text{time} = O(n^3)$$

- Looks like matrix multiplication with
  - "min" instead of "+"
  - "+" instead of multiplication
  - "semiring" – ring without "additive inverse" (e.g. natural numbers under add and multiply)
- Let k be smallest such that $p=2^k > n$
  - But then W^p gives us all pairs shortest paths
  - Again $O(n^3p)=O(n^4)$
  - Can compute by repeated squaring !
  - $WW=W^2$, $W^2W^2=W^4$, $W^4W^4=W^8$, …
  - $O(\log n)$ multiplications
  - $O(n^3 \log n)$ time

Floyd-Warshall

- Change subproblem:
  - » L_{ij}^k - Shortest path distance from i to j using only intermediate nodes 1 through k.
  - » Adding another intermediate node: either it helps (i.e. path uses it) or it does not.



  - » Formally:

$$L_{ij}^k = \begin{cases} w_{ij} & \text{if } k = 0 \\ \min(L_{ij}^{k-1}, L_{ik}^{k-1} + L_{kj}^{k-1}) & \text{if } k > 0 \end{cases}$$

- Time $\Theta(n^3)$

Back to shortest paths: Dijkstra's Algorithm

- We can do better than Bellman-Ford if no negative-weight edges !
- Algorithm:

$d(s) = 0; \quad \forall v \neq s : d(v) = \infty;$
Construct heap, $\forall v \in V : \text{key}(v) = d(v);$
While heap not empty :
  $u = \text{extract\_min}(\text{heap});$
  $\forall v : uv \in E \quad \text{do} :$
    $d(v) = \min(d(v), d(u) + w(uv))$

MST:

$key(s) = 0; \quad \forall v \neq s : key(v) = \infty;$
Construct heap with $key(v)$ as key
$key(s) = 0$
While heap not empty:
  $x = \text{extract\_min}(heap)$
  $\forall v : vx \in E \quad$ do:
    $key(v) = \min(key(v), w(vx))$
  add x to A.

- Main idea: add node with shortest perceived distance.
- Time: n extract_min, m decrease_key

  binary heap: O(m log n)
  Fib. Heap: O(m+n log n)

Compare to O(nm) for Bellman Ford

Update step:

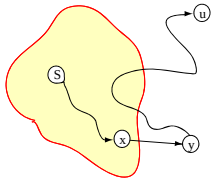
$\forall v : uv \in E$ do:

$d(v) = \min(d(v), d(u) + w(uv))$

Correctness of Dijkstra's algorithm

- **Correctness Proof**:
  - Let u be the first extracted node with d(u) not equal to distance. (note that once any v is extracted, its d(v) is not adjusted)
  - All distances are non-negative and d(v) is at least dist(s,v) for all v [i.e. upper bound]
  - But d(u) incorrect, so d(u) > dist (s,u) (strictly larger)
  - Consider shortest path s to u, focus on edge (xy) where x was extracted already (its d(x) is correct distance) and y was not yet extracted. (Why does such edge exist ?)
  - Observe that d(y) is at most d(x)+w(xy), since x was already processed.

$$\begin{aligned} d(u) &> dist(s,u) \\ &= dist(s,x) + w(xy) + dist(y,u) \\ &= d(x) + w(xy) + dist(y,u) \\ &\geq d(y) + dist(y,u) \\ &\geq d(y) \end{aligned}$$



  - Thus d(u) is currently not minimum and u will not be extracted !

What did we study ?

- Math background
  - Asymptotic analysis
  - A bit of probability
  - Recurrences
- Data Structures
  - Heap
  - Hash function
  - Bloom filter
  - Binary search tree
  - Union-find
- Techniques
  - Binary Search
  - Divide-and-Conquer
  - Randomization
  - Greedy
  - Dynamic Programming
- Proof techniques
  - Optimum substructure
  - Cut-and-paste
  - Amortized analysis
- Specific algorithms
  - Stable matching
  - Closest pair of points
  - Global min-cut
  - Min-cost spanning tree
  - Shortest paths
  - Knapsack
  - Subset sum
  - Huffman encoding
  - Longest common subsequence