

Shortest Paths

CS 4231, Fall 2020

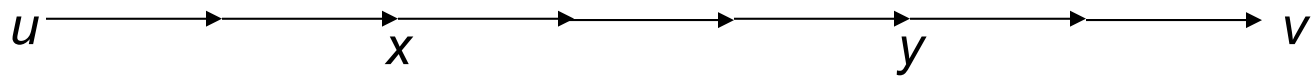
Mihalis Yannakakis

Shortest Paths

- Given graph (directed or undirected) $G=(N,E)$ with lengths (or weights or costs) on the edges $w: E \rightarrow \mathbb{R}$
- Length of a path = sum of lengths of the edges.
- shortest s-t path = path with minimum length from s to t
- $\text{distance}(s,t)$ = length of shortest s-t path
- If all lengths are ≥ 0 , then we only have to consider simple paths (no need to repeat a node) \Rightarrow distances between all pairs are well-defined
- If there are edges with negative length, there may be no shortest path

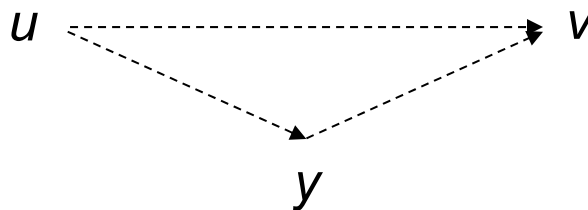
Properties of distances

- Every subpath of a shortest path is a shortest path between the endnodes of the subpath



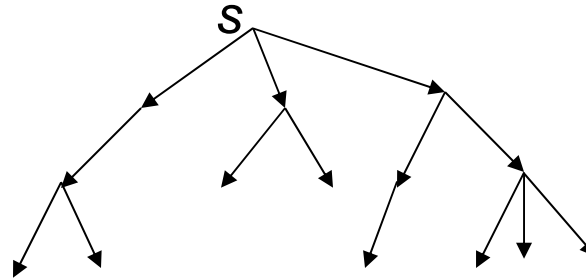
If shortest path from u to v , then x - y subpath is shortest path from x to y

- $\text{dist}(u,u)=0$, for all nodes u
- $\text{dist}(u,v) \leq w(u,v)$, for all edges (u,v)
- $\text{dist}(u,v) \leq \text{dist}(u,y)+\text{dist}(y,v)$, for all nodes u,y,v
(triangle inequality)



Shortest paths from a source node s

Want to compute distances from s to all the nodes and shortest path tree from s



Let $d(v) = \text{dist}(s, v)$

- $d(s) = 0$
- $d(v) \leq d(u) + w(u, v)$ for all edges (u, v) ,
with $=$ for some node $u \Rightarrow$
- $d(v) = \min \{ d(u) + w(u, v) \mid \text{all edges } (u, v) \}$

Distances from a source node s

If we have a path from s to each node v of length $\delta(v)$ and the lengths $\delta(v)$ satisfy

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

then $\delta(v)=\text{dist}(s,v)$ for all v , and the paths are shortest paths

Proof: Show by induction on #edges of any path from s to any node v that $\text{length}(\text{path}) \geq \delta(v)$

Basis: #edges=0 $\Rightarrow v=s$ and $\text{length}(\text{path})=0=\delta(s)$

Induction step: $s \longrightarrow \longrightarrow \longrightarrow \longrightarrow u \longrightarrow v$

$\text{length}(s-u \text{ path}) \geq \delta(u)$ (by i.h.) \Rightarrow

$\text{length}(s-v \text{ path}) \geq \delta(u)+w(u,v) \geq \delta(v)$

Shortest Paths in Graphs with Edge Lengths (Weights) ≥ 0

- Undirected graph \Leftrightarrow Directed graph with two opposite arcs for each edge (digraph with same Adj lists)
- **Dijkstra's algorithm**
- Wave out of source s traveling along edges of the graph at unit speed
- $d[v]$ = *tentative distance* of v from s : time it reaches v
- Done nodes v : $d[v]$ = true distance
- Other nodes v : **Min-Priority Queue Q** with priority $d[v]$

Dijkstra's algorithm

Dijkstra(G, w, s)

for each $v \in N - \{s\}$ do $\{d[v] = \infty; p[v] = \perp\}$

$d[s] = 0; p[s] = \perp$;

$Q = N$ [alternatively, $Q = \{s\}$ and insert nodes when reached]

while $Q \neq \emptyset$ do

{ $u = \text{Extract-Min}(Q)$ [Extract-Min operation; first time, $u = s$]

for each $v \in \text{Adj}[u]$ do

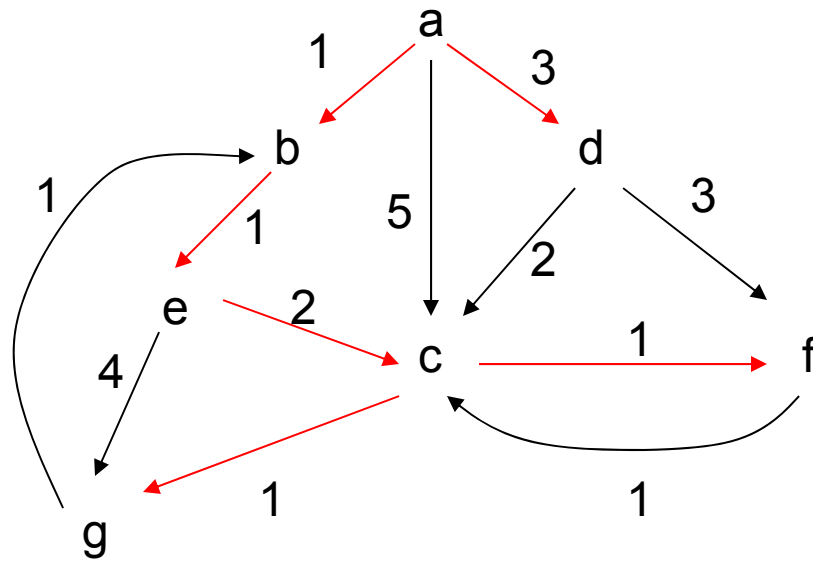
if $d[v] > d[u] + w(u, v)$ then

$\{d[v] = d[u] + w(u, v); p[v] = u\}$ [Decrease-Key(v)]

}

Shortest path tree: $p[v]$ gives the parent of each node v = previous node in a shortest path from s to v

Example



Correctness

- **Invariants:**

1. $\forall u, d[u] < \infty \Rightarrow \exists \text{ s-u path of length } d[u]$
2. Done (R-Q) nodes u have $d[u] \leq \text{Min}(Q)$
(\Rightarrow done nodes u will never change their $d[u]$)

Theorem: $\forall v$: Final $d[v]$ = length of shortest s-v path

Proof:

- \geq : length of s-v path in Dijkstra tree = $d[v]$
- \leq : By induction on length of shortest s-v path

Consider shortest path $s - - - u - v$

By i.h. $d[u] \leq \text{length of s- - -u path}$

Look at time u is processed:

$d[u]$ will not change thereafter and $d[v] \leq d[u] + w(u, v)$

Time Complexity

Operations:	Extract-Min	Decrease-Key
# of ops:	n	e
Time/Op.		
Array:	$O(n)$	$O(1)$
Heap:	$O(\log n)$	$O(\log n)$
Fibonacci Heap: (amortized)	$O(\log n)$	$O(1)$

Total (Worst-Case) Time Complexity:

Array: $O(n^2)$

Heap: $O((n+e)\log n)$

Fibonacci Heap: $O(e+n\log n)$

Shortest Paths in General Weighted Directed Graphs

- Graphs with + or – weights.
- If \nexists negative weight cycle, then distances are well-defined, shortest path between any two nodes is *simple*: does not repeat any node
- In particular for DAGs, distances always well-defined for both positive and negative weights
- If \exists negative weight cycle, then weights of some paths can be made arbitrarily low by going repeatedly around the negative cycle

Directed Acyclic Graphs

Distances $d(v) = \text{dist}(s, v)$ from a source node s satisfy

- $d(s) = 0$
- $d(v) = \min \{ d(u) + w(u, v) \mid \text{all edges } (u, v) \}$

If the graph is acyclic, then the recurrence is not circular.

- Can compute $d(v)$ for all v in topological order.

Single Source Shortest Paths in a DAG

Input: Weighted DAG $G=(N,E)$, weights $w: E \rightarrow \mathbb{R}$, source s

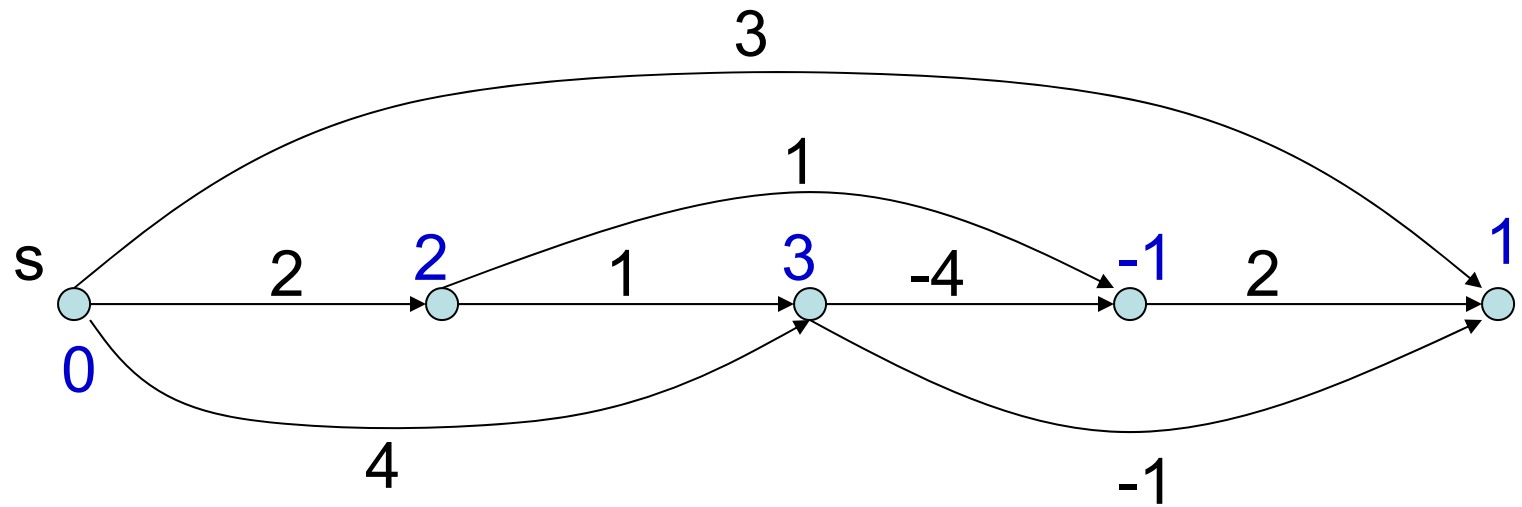
Output: Weight $d[v]$ of shortest path from s to each node v and shortest path tree

1. Initialization: for each $v \in N - \{s\}$ do $\{d[v] = \infty; p[v] = \perp\}$
 $d[s] = 0; p[s] = \perp$;
2. Sort topologically the nodes
3. For each node u in topological order do
 for each v in $\text{Adj}[u]$ do
 if $d[v] > d[u] + w(u,v)$ then $\{d[v] = d[u] + w(u,v); p[v] = u\}$

Time Complexity: $O(n+e)$

Similarly: Longest paths, other problems on DAGs....

Example



Longest Paths in a DAG

Input: Weighted DAG $G=(N,E)$, weights $w: E \rightarrow \mathbb{R}$

Output: Maximum weight $d[v]$ of a path starting from each node v

1. Sort topologically the nodes: v_1, \dots, v_n
2. For $i=n$ down to 1 do
 - if $\text{Adj}[v_i] = \emptyset$ then $d[v_i]=0$
 - else $d[v_i]=\max(0, \{ d[v_j]+w(v_i, v_j) \mid v_j \in \text{Adj}[v_i] \})$

Time Complexity: $O(n+e)$

Dynamic Programming, DAGs

Shortest path in DAG: prototypical Dynamic Programming

$$d(v) = \min \{ d(u) + w(u, v) \mid u \in \text{Pred}(v) \} : \text{“recursive calls } d(u)\text{”}$$

Example: LCS, edit distance can be expressed as a longest / shortest path problem in a DAG

- Many Problems on DAGs can be solved by DP:
 - Compute a function value at u from values at $\text{Pred}(u)$
 - evaluate in topological order of nodes
 - or from values at $\text{Adj}(u)$: use reverse topological order

HW Exercises:

1. Count the # of paths that start at each node of a DAG
2. Count the # paths from s to t

Note: # paths is generally exponential, so cannot list them all

Dynamic Programming, Trees, Graphs

- Rooted trees = special case of DAGs
 - Many problems on trees can be solved this way:
process tree bottom-up (or recursively top-down)
 - Sometimes, have to extend problem so that DP will go through (from value at children, compute value at parent)
- Problems on directed graphs:
Often decompose into SCCs, process SCCs in topological order.

General Digraphs, General Weights

- Suppose there are edges with negative weight
- Shortest paths, distances well defined for all pairs of nodes if there is no negative cycle.
- Easier question: Is there a cycle that contains a negative edge?
- Can be answered in $O(n+e)$ time:
 - Compute the SCCs,
 - Check if any negative edge (u,v) is contained inside a SCC (i.e. u,v in same SCC)
- Determining if there is a negative cycle harder: no $O(n+e)$ algorithm known

Single-source shortest paths – general weights: Bellman-Ford Algorithm

- **Input:** Digraph $G(N,E)$, weights $w: E \rightarrow \mathbb{R}$, source s
- **Output:** indicates if \exists reachable negative cycle, and if \nexists , distances from s to all other nodes

for each $v \in N - \{s\}$ do $\{d[v] = \infty; p[v] = \perp\}$

$d[s] = 0; p[s] = \perp$;

repeat $n-1$ times

 for each edge $(u,v) \in E$ do

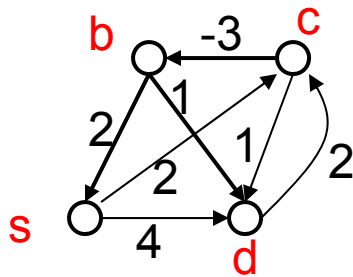
 if $d[v] > d[u] + w(u,v)$ then $\{d[v] = d[u] + w(u,v); p[v] = u\}$

for each edge $(u,v) \in E$ do

 if $d[v] > d[u] + w(u,v)$ then return “negative cycle”

return “no reachable negative cycle”

Example



s	b	c	d	iteration
0	∞	∞	∞	0
0	∞	2	4	1
0	-1	2	3	2
0	-1	2	0	3
0	-1	2	0	4

If edge $d \rightarrow c$ had weight 1 instead of 2, then last iteration would give value 1 for c, indication of a negative cycle.

Negative cycle: $c \rightarrow b \rightarrow d \rightarrow c$

Correctness

1. If there is no reachable negative cycle, then
 $d[v] = \text{distance}(s,v)$ for all reachable nodes v , and
algorithm responds correctly “no reachable negative cycle”

Proof: By induction on # k of iterations of the loop show

Invariant: $d[v] \leq \text{min weight of any } s\text{-}v \text{ path with } \leq k \text{ edges}$

Basis: $k=0$. By initialization

Induction step: Shortest s - v path with $\leq k+1$ edges



In $(k+1)$ th iteration: $d[v] \leq d[u] + w(u,v) \leq \text{weight of } s\text{-}v \text{ path}$

At the end, $d[v] \leq \text{weight of shortest simple } s\text{-}v \text{ path} = \text{dist}(s,v)$

Correctness ctd.

2. If the algorithm returns “no reachable negative cycle”, then \nexists reachable negative cycle

- Proof: Reachable Cycle $C: v_1 \rightarrow v_2 \rightarrow \dots v_m \rightarrow v_1$
reachable $\Rightarrow d[v_i] < \infty, \forall i$
 $d[v_i] \leq d[v_{i-1}] + w(v_{i-1}, v_i), \forall i$
 $\Rightarrow \sum_i d[v_i] \leq \sum_i d[v_{i-1}] + \sum_i w(v_{i-1}, v_i)$
 $\Rightarrow 0 \leq w(C)$

Time Complexity : $O(ne)$

Detection of Negative Cycles

- **Input:** Digraph $G(N,E)$, weights $w: E \rightarrow \mathbb{R}$
- **Output:** \exists negative weight cycle ?
- **One method:**
 1. Compute the strongly connected components
 2. For each scc C , pick arbitrary “source” node s in C and run Bellman-Ford from s restricted to C
- \exists negative cycle iff \exists negative cycle inside a scc
- Time Complexity: $O(ne)$

Detection of Negative Cycles: Method 2

1. Add new source node s , and 0 weight edges from s to all the nodes of $G \rightarrow$ graph G'
 2. Apply Bellman-Ford to (G', w, s)
- \exists a reachable negative cycle in (G', w, s) iff (G, w) has a negative cycle
 - If no negative cycle, then final d values satisfy
$$d[v] \leq d[u] + w(u, v), \quad \forall (u, v) \in E$$
i.e. $d[v] - d[u] \leq w(u, v), \quad \forall (u, v) \in E$

One application: Difference Constraints

- Variables x_1, \dots, x_n
- Set of linear inequalities of the form
 $x_j - x_i \leq b_k$ (a constant)
- Consistent if there is a solution in Reals
- Constraint graph G with one node v_i for each variable x_i
- Edges (x_i, x_j) with weight b_k
- Set is consistent iff no negative cycle in G
- Distances from new source s satisfy the inequalities.

All Pairs Shortest Paths

All-Pairs Shortest Paths

- **Input:** Digraph $G(N,E)$, weights $w: E \rightarrow \mathbb{R}$
- **Find:** Shortest paths between all pairs of nodes or determine \exists negative weight cycle
- **To test if a weighted graph (G,w) has a negative cycle:**
 1. Add new source node s , and 0 weight edges from s to all the nodes of $G \rightarrow$ graph G'
 2. Apply Bellman-Ford to (G',w,s) :
 \exists a reachable negative cycle in (G',w,s) iff (G,w) has a negative cycle

Weight Transformation

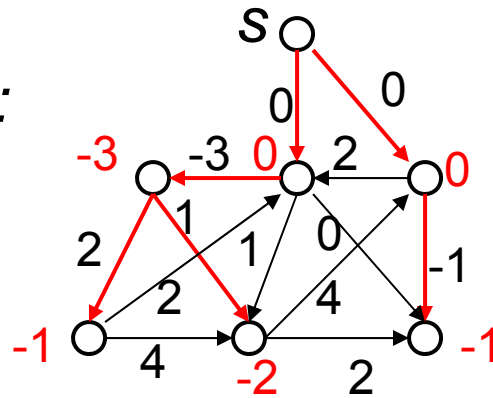
- If there are no negative cycles, we can use the computed distances from s in G', w', s as 'node potentials' to make all edge weights ≥ 0 , without affecting the shortest paths for any pair of nodes:
- $w'(u,v) = w(u,v) + d[u] - d[v], \quad \forall (u,v) \in E$

Properties:

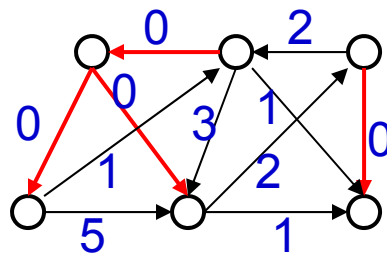
- $w'(u,v) \geq 0, \quad \forall (u,v) \in E$ (since $d[v] \leq d[u] + w(u,v)$)
- \forall pair of nodes x,y and path p from x to y ,
 $w'(p) = w(p) + d[x] - d[y]$
- p is shortest x - y path in $(G,w) \iff$ shortest in (G,w')

Example

Distances from s:



Modified weights:



Johnson's Algorithm for the All-pairs Shortest Path Problem

1. Use Bellman-Ford to determine if \exists negative weight cycle and transform the weights to ≥ 0
2. Apply Dijkstra from every node

Time Complexity: $O(n^2 \log n + en)$

Floyd-Warshall Algorithm

- Alternative DP algorithm for all-pair shortest paths
- Simple, good for dense graphs
- Nodes $N = \{ 1, \dots, n \}$, weights $w_{ij} = w(i, j)$
 $w_{ij} = \infty$ if no edge (i, j) , $w_{ij} = 0$ if $i = j$
- **Dynamic Programming:** Compute the length $d_{ij}^{(k)}$ of the shortest path from i to j that does not use any intermediate nodes beyond k , for $k=0, 1, 2, \dots$

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

Floyd-Warshall Algorithm

$$D^{(0)} = W$$

for $k = 1$ to n do

 for $i = 1$ to n do

 for $j = 1$ to n do

$$d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$$

return $D^{(n)}$

Time Complexity $O(n^3)$

Can drop the superscripts \rightarrow Space $O(n^2)$

To recover the shortest paths: record best k for each i, j

Graph has a negative cycle $\Leftrightarrow \exists$ negative entry on the diagonal of the final matrix