Bellman-Ford algorithm

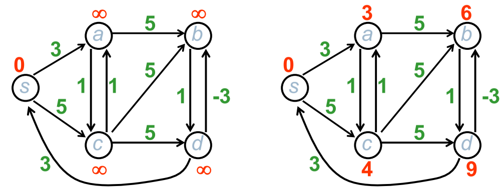
Bellman-Ford algorithm

No condition on weights : for all edges (p, q), $w(p, q) \in \mathbf{R}$

```
begin
    INIT;
    Q = V ;
    for i=1 to |V|-1 do
        for each (q, r) ∈ E do
            RELAX(q, r) ;
    for each (q, r) ∈ E do
        if d[q] + w(q, r) < d[r]  then
            return « negative cost cycle detected »
        else
            return « minimum costs computed »
end
```

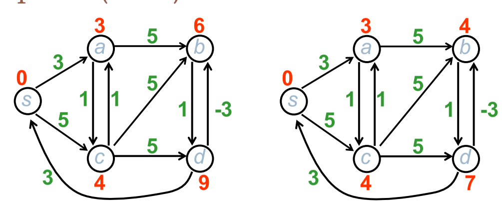
Time complexity : O(n·m)

Example 1



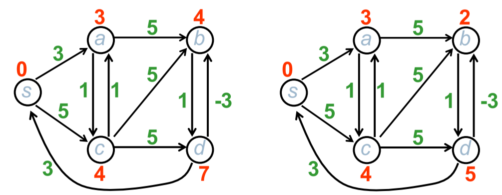
Step 1: relaxing all edges in the following order:
(*s*,*a*) (*s*,*c*) (*a*,*b*) (*a*,*c*) (*b*,*d*) (*c*,*a*) (*c*,*b*) (*c*,*d*) (*d*,*b*) (*d*,*s*)

Example 1 (cont)



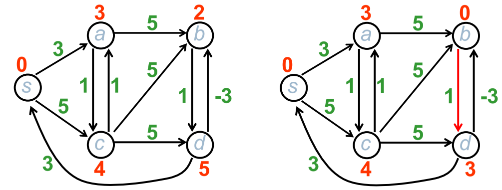
Step 2: relaxing all edges in the following order:
(*s*,*a*) (*s*,*c*) (*a*,*b*) (*a*,*c*) (*b*,*d*) (*c*,*a*) (*c*,*b*) (*c*,*d*) (*d*,*b*) (*d*,*s*)

Example 1 (cont)



Step 3: relaxing all edges in the following order:
(*s*,*a*) (*s*,*c*) (*a*,*b*) (*a*,*c*) (*b*,*d*) (*c*,*a*) (*c*,*b*) (*c*,*d*) (*d*,*b*) (*d*,*s*)

Example 1 (cont)



Step 4: relaxing all edges in the following order:
(*s*,*a*) (*s*,*c*) (*a*,*b*) (*a*,*c*) (*b*,*d*) (*c*,*a*) (*c*,*b*) (*c*,*d*) (*d*,*b*) (*d*,*s*)

relaxation still possible ⇒ cycle of negative cost

Why Bellman-Ford is correct?

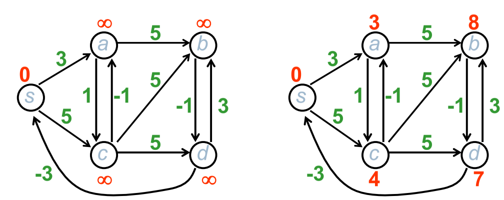
because, if there is no negative-cost cycle, every node has a cycle-free shortest path with at most $|V|$-1 edges

$((s_0,s_1), (s_1,s_2), \ldots, (s_{k-1},s_k))$ with $s_0=s$ and $s_k=t$, $k \le |V|-1$

At iteration *i*, we will relax (among other edges) (s_{i-1},s_i). This guarantees the shortest path value for all nodes. No relaxation will be possible anymore.

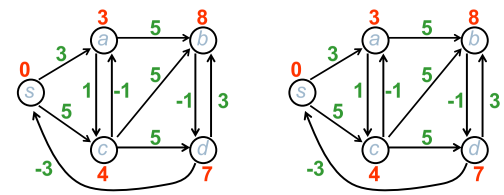
If there exists a negative-cost cycle, one of the edges along the cycle must be possible to relax (prove).

Example 2



Step 1: relaxing all edges in the following order:
(*s*,*a*) (*s*,*c*) (*a*,*b*) (*a*,*c*) (*b*,*d*) (*c*,*a*) (*c*,*b*) (*c*,*d*) (*d*,*b*) (*d*,*s*)

Example 2 (cont)



Step 2: relaxing all edges in the following order:
(*s*,*a*) (*s*,*c*) (*a*,*b*) (*a*,*c*) (*b*,*d*) (*c*,*a*) (*c*,*b*) (*c*,*d*) (*d*,*b*) (*d*,*s*)

no more possible relaxation ⇒ costs correctly computed

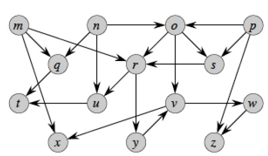
Shortest paths in Directed Acyclic Graphs

Shortest paths in Directed Acyclic Graphs

- $G = (V, E)$, $w : E \rightarrow \mathbf{R}$ (possibly negative)
- *Problem*: given a node $s \in V$, compute shortest paths from s to all other nodes reachable from s

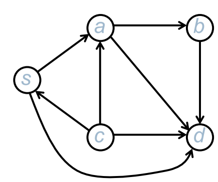
Directed Acyclic Graph (DAG)

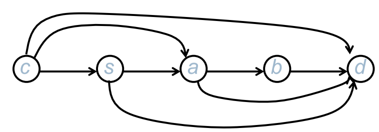
- Directed graph without cycles
- $\Rightarrow$ at least one node with indegree 0, and at least one with outdegree 0



Topological sort

- linearly order vertices such that all edges go from smaller to larger

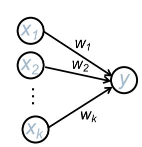




- Topological sort can be done in time $O(n + m)$ (iterative solution using a queue, solution based on DFS, …)

Shortest paths in Directed Acyclic Graphs

- $G = (V, E)$, $w : E \rightarrow \mathbf{R}$ (possibly negative)
- *Problem*: given a node $s \in V$, compute shortest paths from s to all other nodes reachable from s

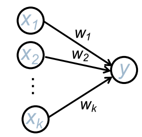


- *main idea*: $d(y)=\min\{d(x_1)+w_1, d(x_2)+w_2, \ldots, d(x_k)+w_k\}$

“Swipe-through” solution

- for all nodes t, assign $d(t)=\infty$
- $d(s) = 0$
- starting from s, for all y in topological order

$$d(y)=\min\{d(x_1)+w_1, d(x_2)+w_2, \ldots, d(x_k)+w_k\}$$

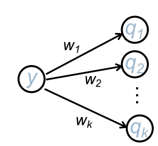


Time: $O(n + m)$

“Dijkstra-style” solution

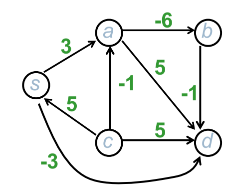
- for all nodes t, assign $d(t) = \infty$
- $d(s) = 0$
- starting from s, for all y in topological order

  for each edge (y,q), **RELAX**(y,q)



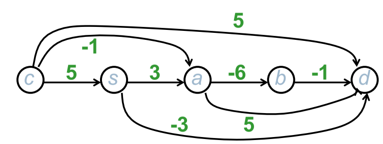
Time: $O(n + m)$

Example

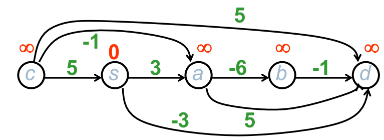


Topological order

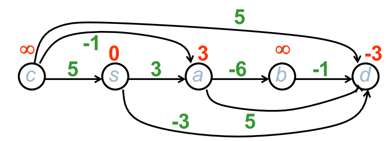
c, s, a, b, d



Computing shortest paths (Dijkstra-style)



Processing s

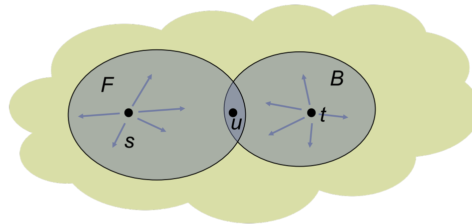


Source-to-destination search

- Assume all edges have non-negative weight. How to search for a shortest path from s to t with Dijkstra's algorithm?

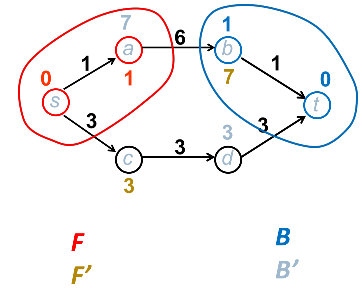
Better idea: bidirectional search

- Bidirectional search (idea): perform Dijkstra on G starting from s and on the reverse graph G^R starting from t. Stop when these searches "meet" (*to be defined*)



- *Catch*: if u is the first occurred node from $F \cap B$, the shortest path from s to t does may not pass through u !

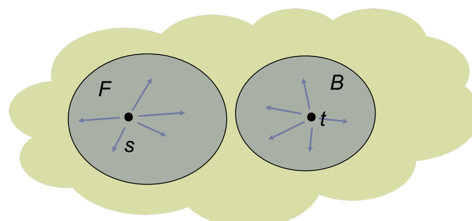
Counter-example



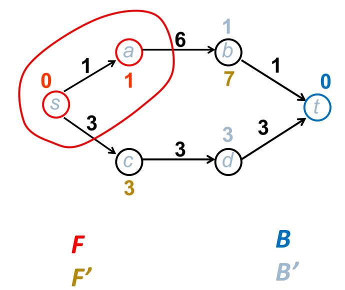
Source-to-destination search

Better idea: bidirectional search

- Bidirectional search (idea): perform Dijkstra on G starting from s and on the reverse graph G^R starting from t. Stop when these searches "meet" (*to be defined*)

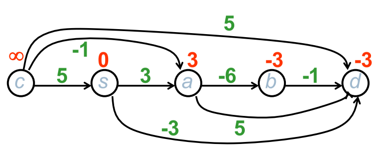


Counter-example

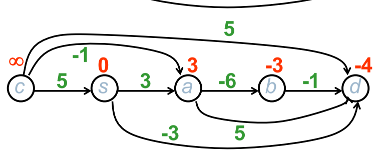


Computing shortest paths (Dijkstra-style)

Processing *a*



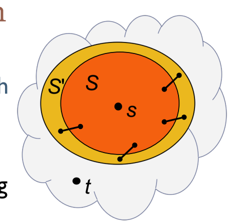
Processing *b*



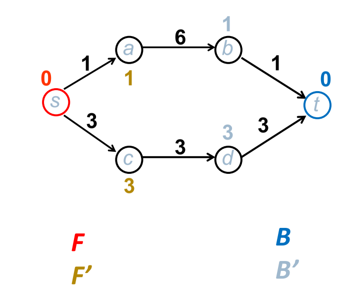
Source-to-destination search

- Assume all edges have non-negative weight. How to search for a shortest path from s to t with Dijkstra's algorithm?

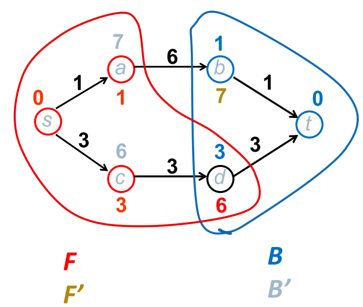
Early exit: Run Dijkstra's algorithm starting from s. Once t is extracted from Q, stop.



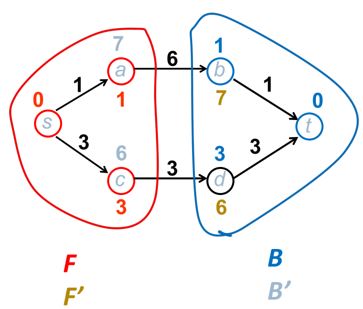
Counter-example



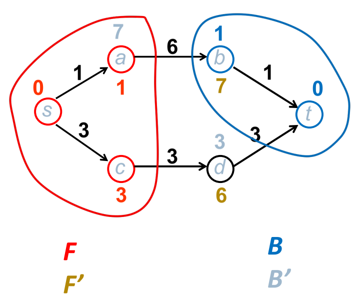
Counter-example



Counter-example



Counter-example



Correct stopping strategy

1. initially set $D_{min} = \infty$
2. when relaxing an edge (v, u), $v \in F, u \in B$, set $D_{min} = \min\{D_{min}, d_f[v] + w(v, u) + d_b[u]\}$ (similar for backward search)
3. let $top_f,\ top_b$ be the minimum d-values of forward and backward priority queues respectively. Then if $top_f + top_b \geq D_{min}$, then stop

Proof: by contradiction

To sum up

- **Breadth-first search** explores *the whole graph* and finds shortest paths *to all nodes* under assumption that all moves have equal cost. It uses a *queue*.
- **Dijkstra's algorithm** explores *the whole graph* and finds shortest paths *to all nodes* taking into account different move costs. It uses a *priority queue*
- **Bidirectional search** solves point-to-point shortest path problem by running two Dijkstra's

Heuristics for point-to-point search

- **(Greedy) Best-first** search finds a path to *a target node* by exploring the frontier nodes that are estimated to be closer to the target ($h(v)$: lower bound of min distance from v to target) https://www.youtube.com/watch?v=TdHbO3w68fY

Example: 15 puzzle

https://medium.com/@prestonbjensen/solving-the-15-puzzle-e7e60a3d9782

- $\sim 10^{13}$ distinct states, exploring the tree of possible moves leads to $\sim 10^{38}$ states
- possible functions h for best-first search:
  1. number of tiles in incorrect positions
  2. sum of Manhattan distances (absolute horizontal distance + absolute vertical distance) of every tile to its correct location

Example: 15 puzzle

https://medium.com/@prestonbjensen/solving-the-15-puzzle-e7e60a3d9782

- $\sim 10^{13}$ distinct states, exploring the tree of possible moves leads to $\sim 10^{38}$ states
- possible functions h for best-first search:
  1. number of tiles in incorrect positions
  2. sum of Manhattan distances (absolute horizontal distance + absolute vertical distance) of every tile to its correct location
- second is better than first

Solution Length		
	Manhattan	Number Wrong
mean	10.58	18.22
10th percentile	10	10
50th percentile	10	10
90th percentile	10	36

Explored States		
	Manhattan	Number Wrong
mean	27.71	580.1
10th percentile	11	11
50th percentile	11	14
90th percentile	28	1076

Heuristics for point-to-point search

- **(Greedy) Best-first** search finds a path to *a target node* by exploring the frontier nodes that are estimated to be closer to the target ($h(v)$: lower bound of min distance from v to target) https://www.youtube.com/watch?v=TdHbO3w68fY
- **A*** search finds a path to *a target node* by exploring the frontier nodes that have the minimum sum of distance *from* the source ($f(v)$) and estimated distance *to* the target ($h(v)$)

http://www.redblobgames.com/pathfinding/a-star/introduction.html

- more on heuristic search: *Pearl, J. Heuristics: Intelligent Search Strategies for Computer Problem Solving. Addison-Wesley, 1984*

Example: 15 puzzle (cont)

https://medium.com/@prestonbjensen/solving-the-15-puzzle-e7e60a3d9782

9	2	8	11
	5	13	7
15	1	4	10
3	14	6	12

- A*: $g(v)+h(v)$ where
  - $g(x)$: number of moves to state x
  - sum of Manhattan distances (as before)
- best-first: $h(v)$ only

- A* is better than best-first

Solution Lengths		
	A*	Pure Heuristic
mean	22	59.66
10th percentile	17	23
50th percentile	23	52
90th percentile	25	111

Explored States		
	A*	Pure Heuristic
mean	755.87	1240.35
10th percentile	71.1	45.8
50th percentile	350.5	664.5
90th percentile	1738.2	3498.1