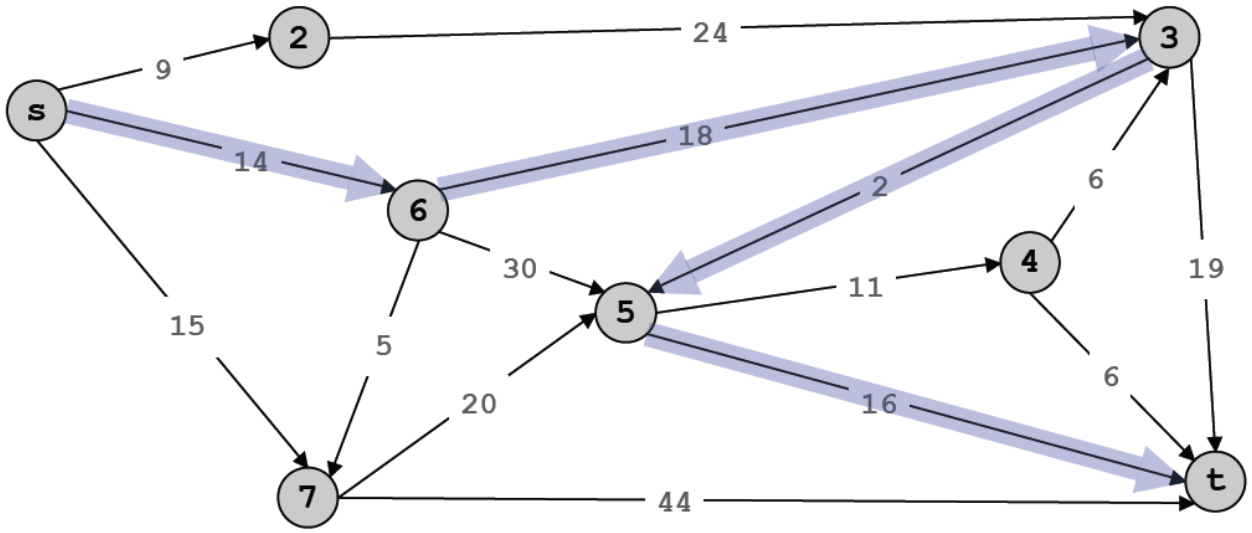
COS 397 – week 15

Shortest path problem. Given a weighted digraph, find the shortest directed path from s to t. (cost of path = sum of edge costs in path)



Path: s→6→3→5→t
Cost: 14 + 18 + 2 + 16 = 50

Versions:

- Point-to-point, single source, all pairs.
- Nonnegative edge weights, arbitrary weights, Euclidean weights.

Applications:

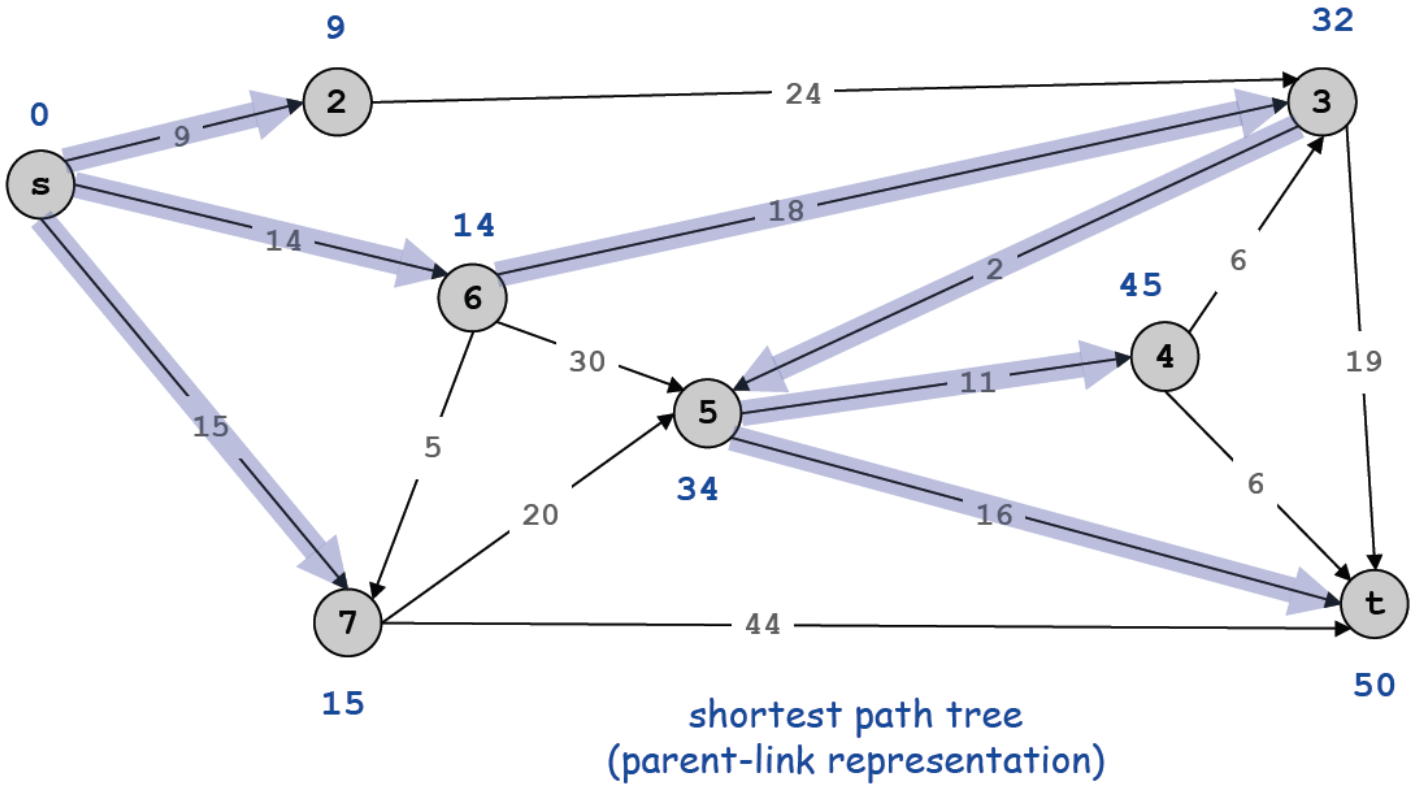
- Robot navigation.
- Urban traffic planning.
- Optimal pipelining of VLSI chip.
- Telemarketer operator scheduling.
- Subroutine in higher level algorithms.
- Routing of telecommunications messages.
- Network routing protocols (OSPF, BGP, RIP).
- Exploiting arbitrage opportunities in currency exchange.
- Optimal truck routing through given traffic congestion pattern.

Single Source Shortest Path

Assumptions:

- Digraph G.
- Single source s.
- Edge weights c(v, w) are nonnegative.

Goal: Find shortest path from s to every other vertex.

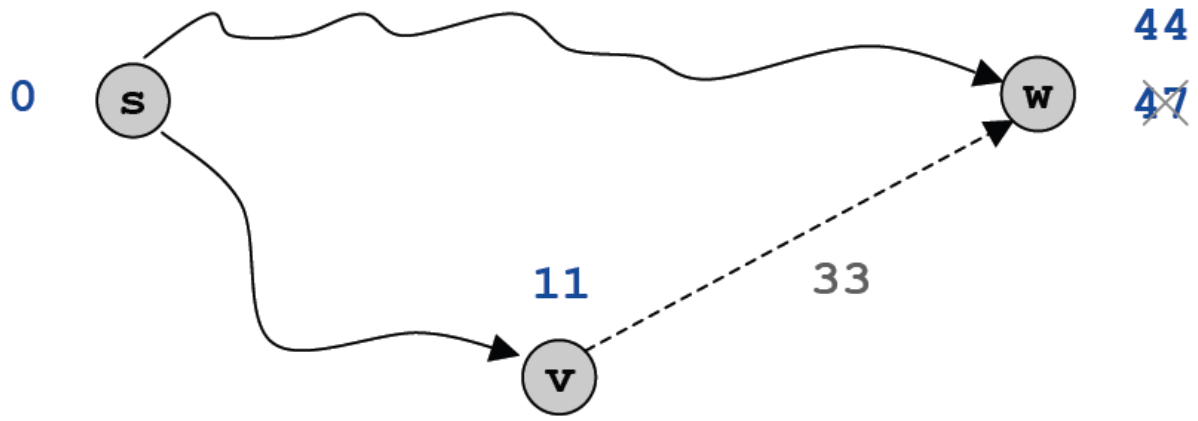


Edge Relaxation

For all vertices v, $\pi(v)$ is the length of some path from s to v.

Edge relaxation.

- Consider edge e = v → w.
- If current path from s to v plus edge v → w is shorter than current path to w, then update current path to w.



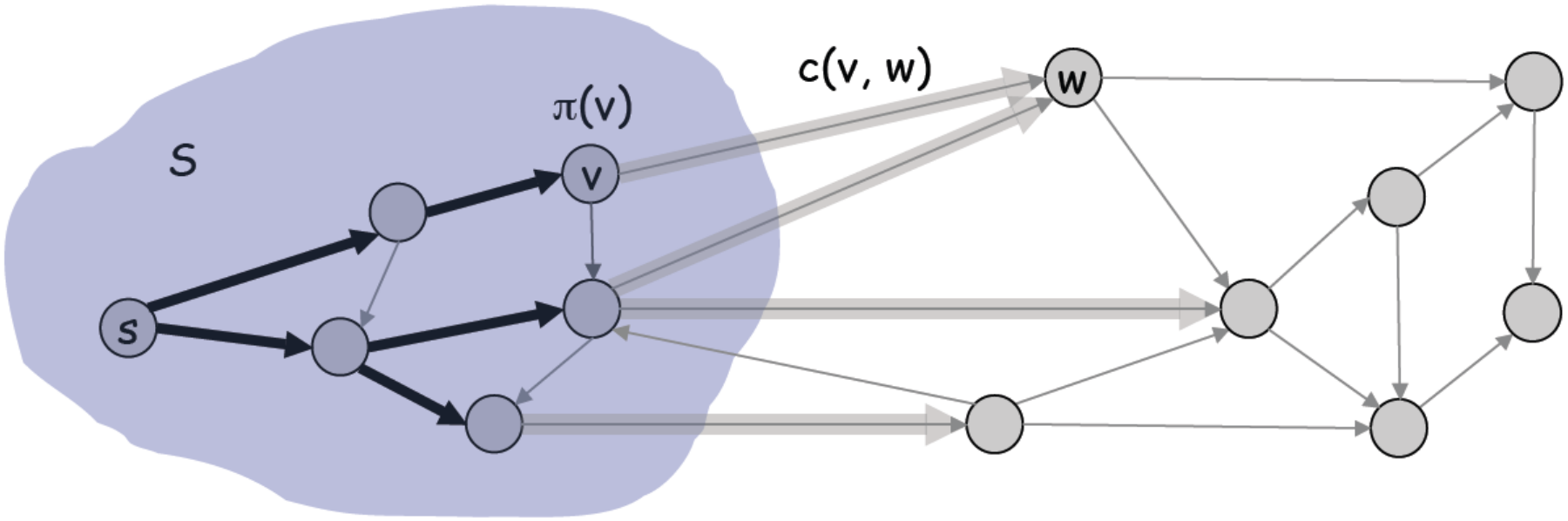
Dijkstra's algorithm. Maintain a set of weights π (v) and a set of explored vertices S for which π (v) is the length of the shortest s - v path.

- Initialize: S = { s }, π (s) = 0.
- Repeatedly choose unexplored node w which minimizes:

$$\pi(w) = \min_{(v,w) : v \in S} \pi(v) + c(v,w)$$

(the shortest path to some v in explored part, followed by a single edge e = (v, w))

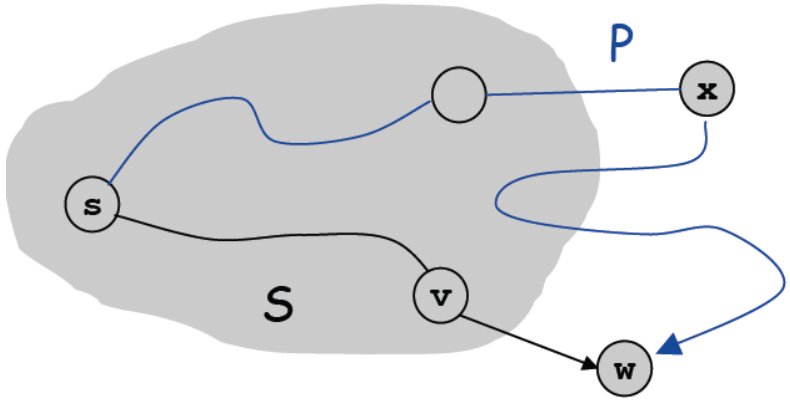
- set pred[w] = v
- add w to S, and set π (w) = π (v) + c(v, w)

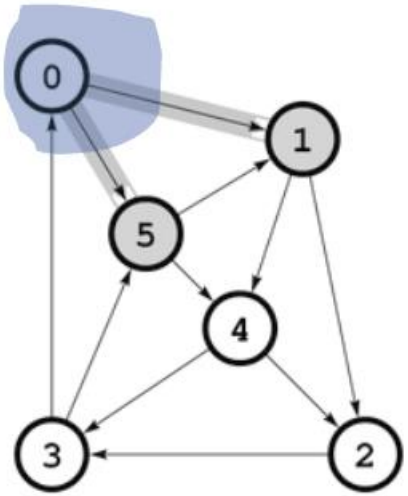


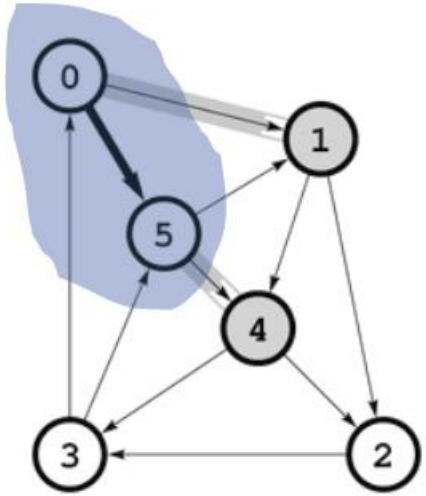
Dijkstra's Algorithm: Proof of Correctness

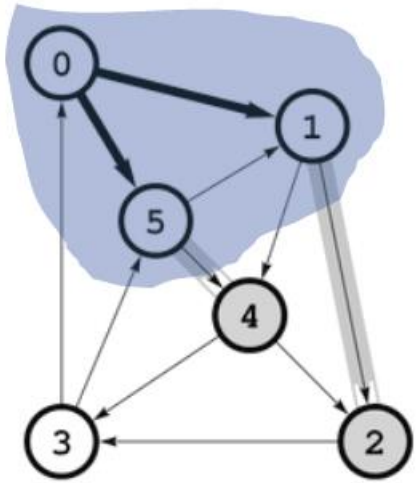
Invariant.
For each vertex v in S, π (v) is the length of shortest s-v path.
Proof by induction on |S|:

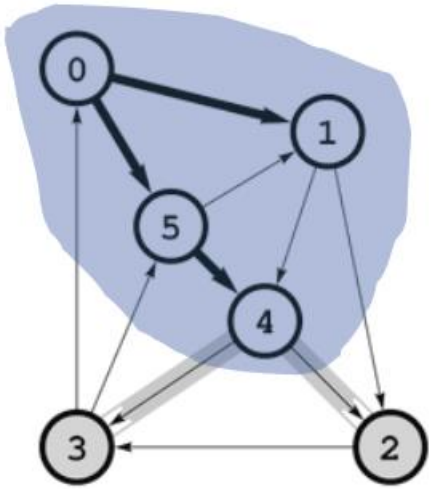
- Let w be the next vertex added to S.
- π (w) = π (v) + c(v, w) is a length of some s-w path.
- Consider any s-w path P, and let x be first node on a path outside S.
- P is already too long as soon as it reaches x by a greedy choice

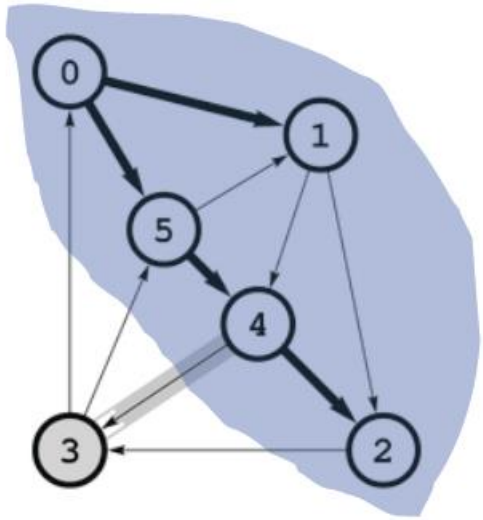


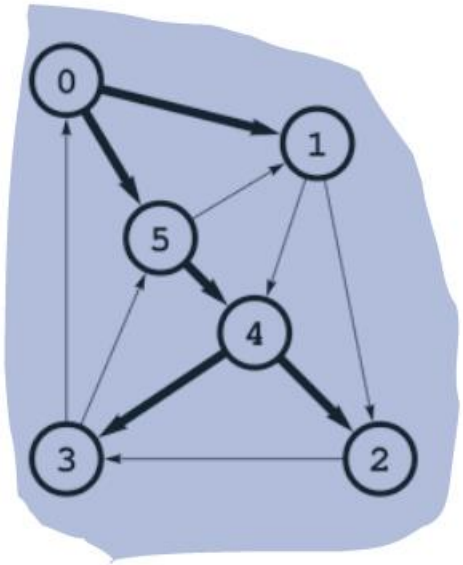

0
1
5
4
3
2


0
1
5
4
3
2


0
1
5
4
3
2


0
1
5
4
3
2


0
1
5
4
3
2


0
1
5
4
3
2

Dijkstra's Algorithm: Implementation

Critical step. Choose unexplored node w which minimizes:

$$\pi(w) \;=\; \min_{(v,w)\,:\,v \in S} \pi(v) + c(v,\ w)$$

Brute force implementation. Test all edges: O(EV) time.

Efficient implementation. Maintain a priority queue of unexplored vertices, prioritized by π (w).

When exploring v, for each edge v → w leaving v, update

$$\pi(w) = \min \{\ \pi(w),\ \pi(v) + \ c(v,\ w)\ \}$$

Indexed PQ.

- Assume items are named 0 to N-1.
- Insert, delete min, test if empty. [PQ ops]
- Decrease key, contains. [ST-like ops]

`class IndexMinPQ` (indexed priority queue)

`IndexMinPQ(int N)` create an empty PQ on N elements
`void insert(int i, Key key)` add element i with given key
`void decrease(int i, Key key)` decrease value of item i
`int delMin()` delete and return smallest item
`bool isEmpty()` is the PQ empty?
`bool contains(int i)` does the PQ contain item i?

Indexed PQ: array implementation.

- Maintain vertex indexed array keys[i].
- Insert key: change keys[i].
- Decrease key: change keys[i].
- Delete min: scan through keys[i] for each item i.
- Maintain a boolean array marked[i] to mark items in the PQ.

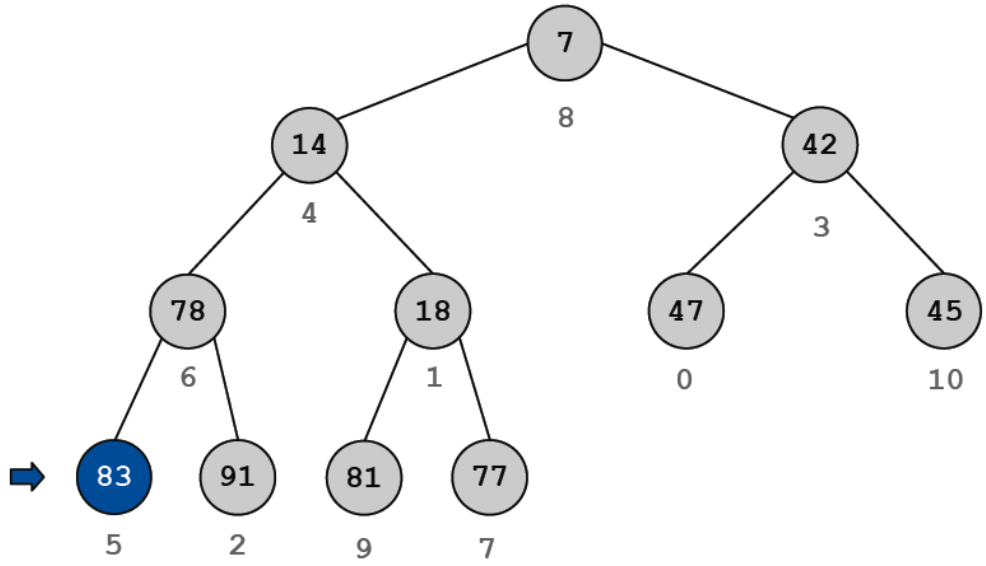
Operation	Array	Dijkstra
insert	1	× V
delete-min	V	× V
decrease-key	1	× E
is-empty	1	× V
contains	1	× V
total	V^2	

Indexed PQ: binary heap implementation.

- Assume items are named 0 to N-1.
- Store priorities in a binary heap.

- How to decrease key of item i? Bubble it up.
- How to know which heap node to bubble up? Maintains an extra array qp[i] that stores the heap index of item i.

decrease key of element 5 from 83 to 31



i	pq	qp	key
0	-	6	47
1	8	5	18
2	4	9	91
3	3	3	42
4	6	2	14
5	1	8	83
6	0	4	78
7	10	11	77
8	5	1	7
9	2	10	81
10	9	7	45
11	7	-	-

The choice of priority queue matters in Dijkstra's implementation.

- Array: $O(V^2)$.
- Binary heap: $O(E \log V)$.

Priority first search. Maintain a set of explored vertices S, and grow S by exploring edges with exactly one endpoint leaving S.

DFS. Edge from vertex which was discovered most recently.

BFS. Edge from vertex which was discovered least recently.

Prim. Edge of minimum weight.

Dijkstra. Edge to vertex which is closest to s.

Application: Currency Conversion

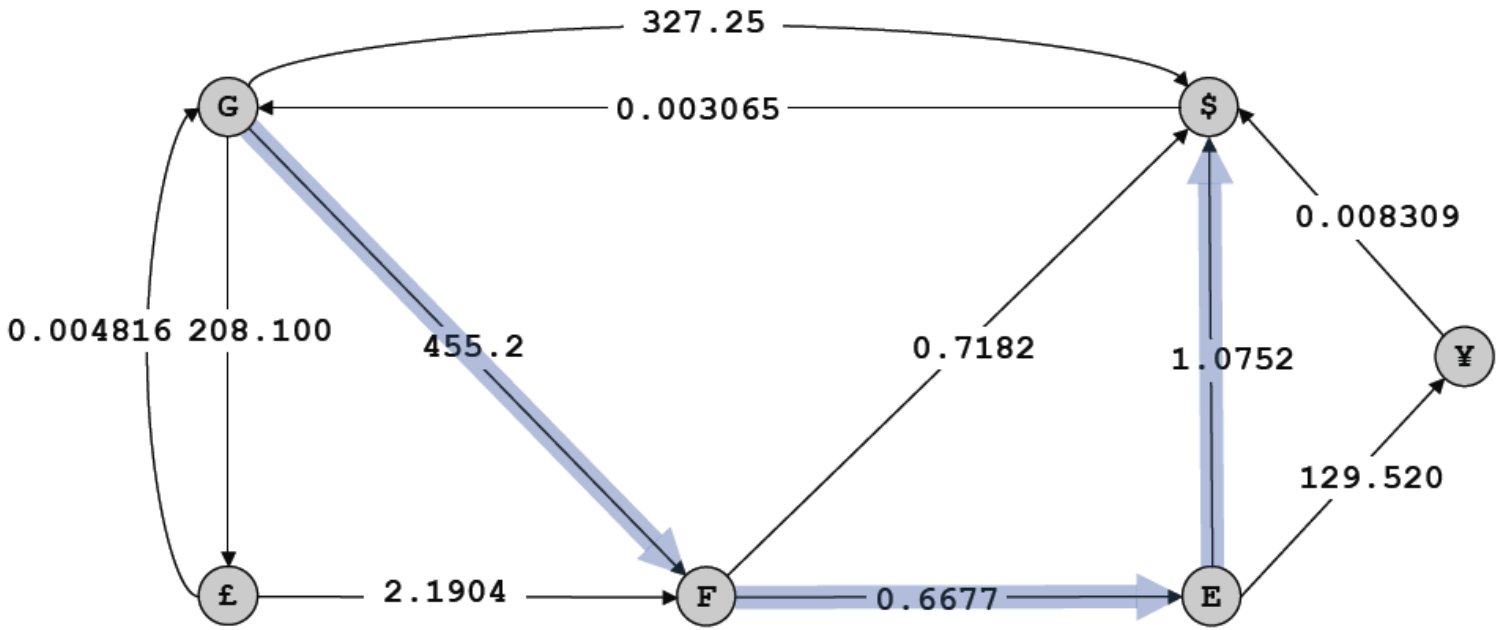
Given currencies and exchange rates, what is the best way to convert one ounce of gold to US dollars?

- 1 oz. gold ⇒ $327.25.
- 1 oz. gold ⇒ £208.10 ⇒ ⇒ $327.00. [208.10 × 1.5714]
- 1 oz. gold ⇒ 455.2 Francs ⇒ 304.39 Euros ⇒ $327.28. [455.2 × .6677 × 1.0752]

Currency	£	Euro	¥	Franc	$	Gold
UK Pound	1.0000	0.6853	0.005290	0.4569	0.6368	208.100
Euro	1.4599	1.0000	0.007721	0.6677	0.9303	304.028
Japanese Yen	189.050	129.520	1.0000	85.4694	120.400	39346.7
Swiss Franc	2.1904	1.4978	0.011574	1.0000	1.3929	455.200
US Dollar	1.5714	1.0752	0.008309	0.7182	1.0000	327.250
Gold (oz.)	0.004816	0.003295	0.0000255	0.002201	0.003065	1.0000

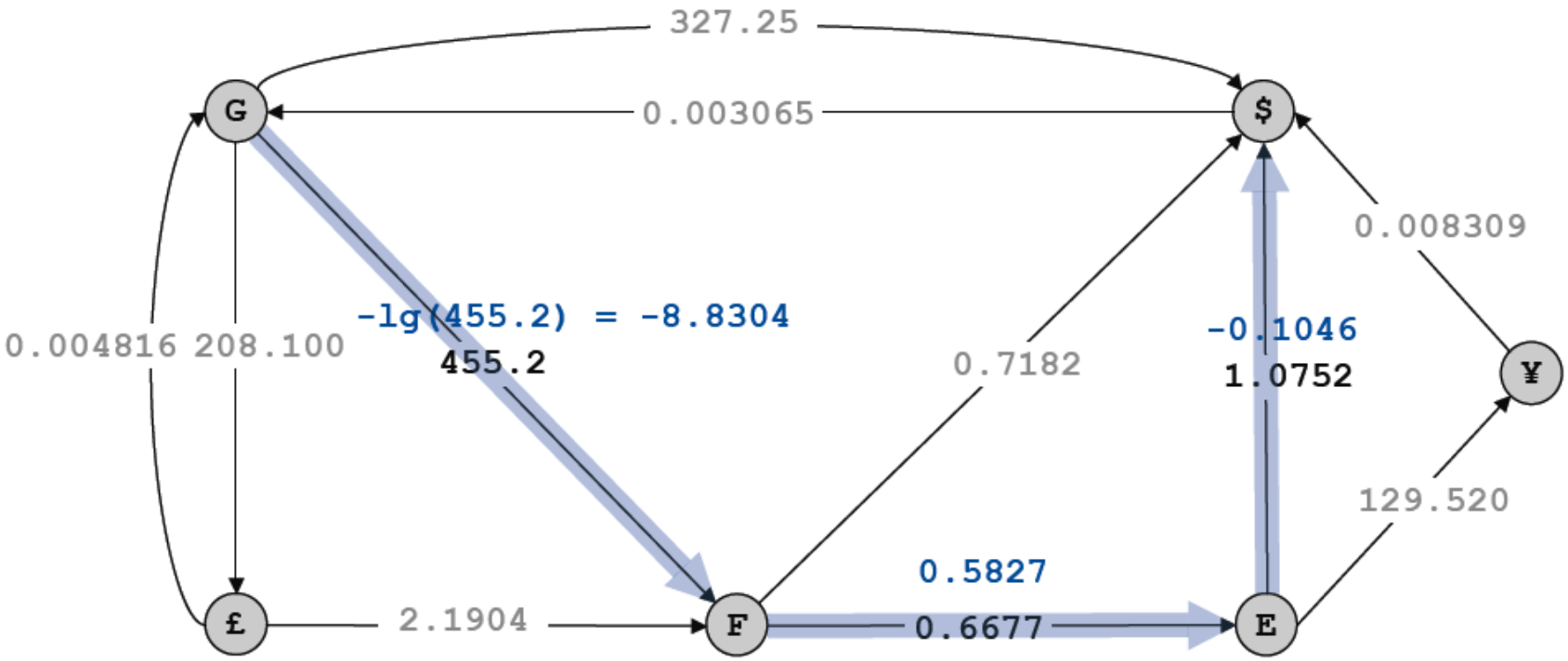
Graph formulation.

- Vertex = currency.
- Edge = transaction, with weight equal to exchange rate.
- Find path that maximizes product of weights.



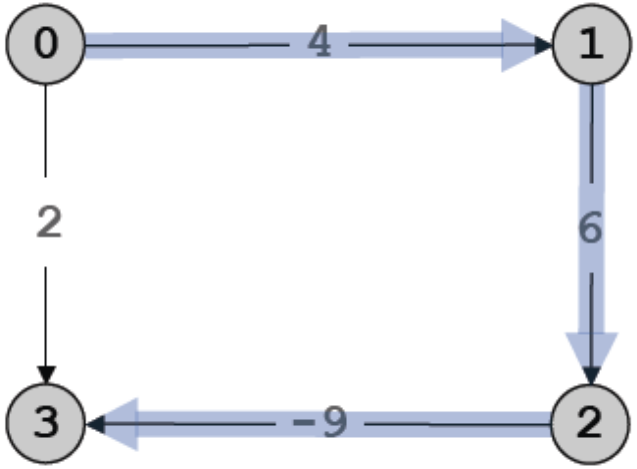
Reduction to shortest path problem.

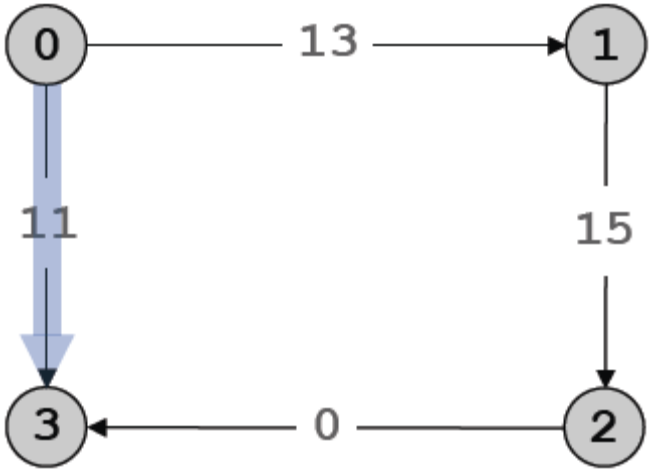
- Let r(v, w) be exchange rate from currency v to w.
- Let c(v, w) = –lg r(v, w).
- Shortest path with costs c corresponds to best exchange sequence.



Challenge. Solve shortest path problem with *negative* weights.

Dijkstra: Can fail if negative edge weights

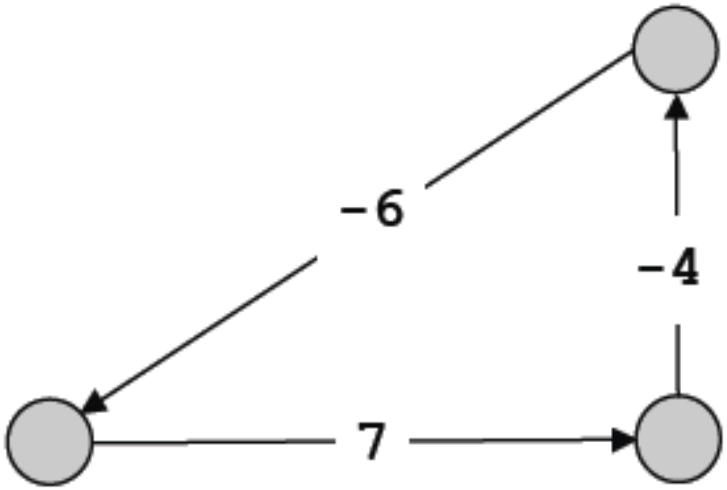




Dijkstra selects vertex 3 immediately after 0.
But shortest path from 0 to 3 is $0 \rightarrow 1 \rightarrow 2 \rightarrow 3$.

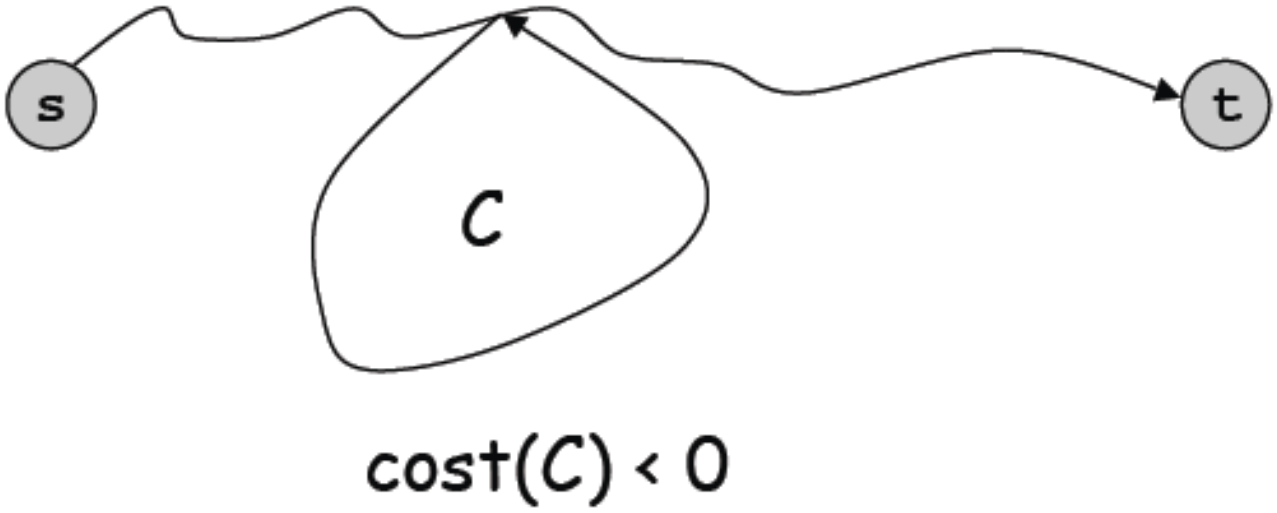
Re-weighting: Adding a constant to every edge weight can fail.

Adding 9 to each edge changes the shortest path.



Shortest Paths: Negative Cost Cycles

Negative cycle. Directed cycle whose sum of edge weights is negative.



Observation: If there is a negative cycle C on path from s to t, then shortest path can be made arbitrarily negative by spinning around cycle; otherwise, there exists a shortest s-t path that is simple.

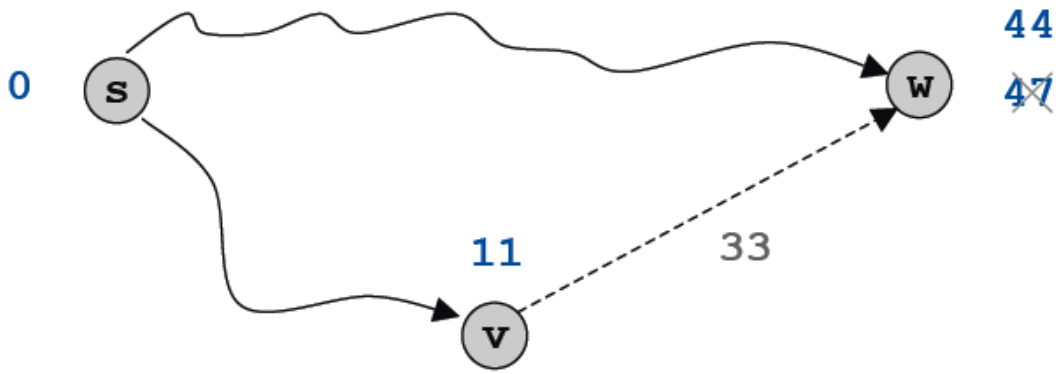
Review: Edge Relaxation

For all vertices v, π (v) is the length of some path from s to v.

Edge relaxation.

Consider edge e = v → w.

If current path from s to v plus edge v → w is better than the current path to w, then update current path to w.



```
if (pi[w] > pi[v] + e.weight)
{  pi[w] = pi[v] + e.weight);
  pred[w] = v;
}
```

Dynamic programming algorithm.

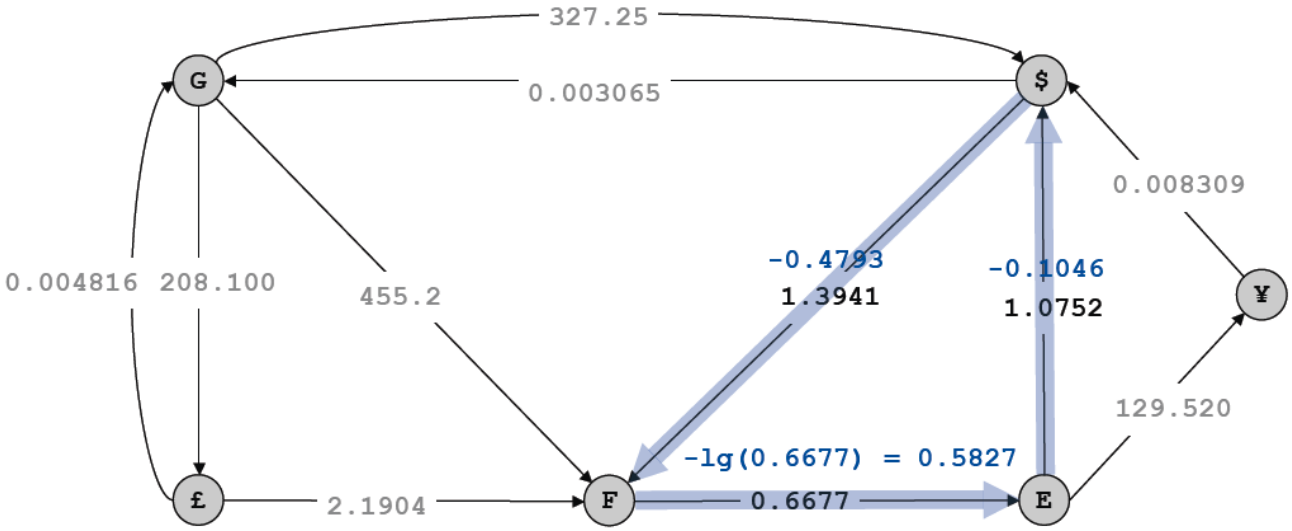
- Initialize pi[v] = ∞, pi[s]= 0.
- Repeat V times: relax each edge e.

```
for (int i = 1; i <= V; i++)
 for (int v = 0; v < G.V(); v++)
  for (all Edge e in G.adj(v))
  {
    int w = e.target;
    if (pi[w] > pi[v] + e.weight)
     {pi[w] = pi[v] + e.weight); pred[w] = v;}
   }
```

Bellman-Ford-Moore: Running time: O(E V).

Arbitrage. Is there an arbitrage opportunity in currency graph?

- Ex: $1 → 1.3941 Francs → 0.9308 Euros → $1.00084.
- Is there a negative cost cycle?
- Fastest algorithm very valuable!

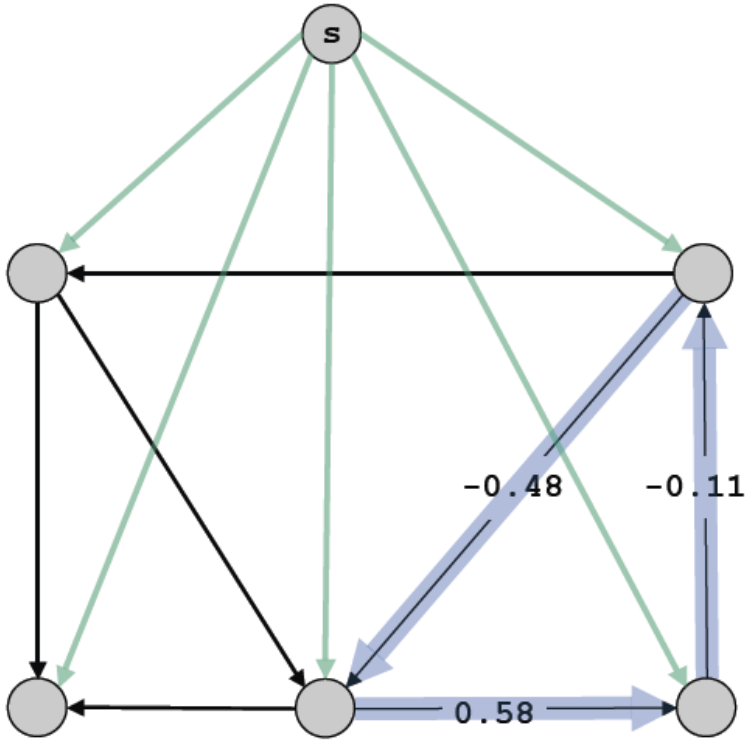


-0.4793 + 0.5827 - 0.1046 < 0

If negative cycle reachable from s. Bellman-Ford-Moore gets stuck in infinite loop, updating vertices in a cycle.
Finding a negative cycle. If any vertex v is updated in phase V, there exists a negative cycle, and we can trace back pred[v] to find it.

Negative Cycle Detection. Identify a negative cycle (reachable from any vertex).

Solution. Add 0-weight edge from artificial source s to each vertex v. Run Bellman-Ford from vertex s.



Shortest/Longest Path in DAG

Shortest path in DAG algorithm.

- Consider vertices v in topological order:relax each edge v → w

Theorem. Algorithm computes shortest path in linear time (even if negative edge weights).