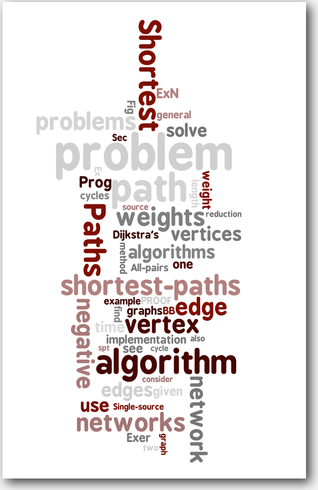
5.4 Shortest Paths

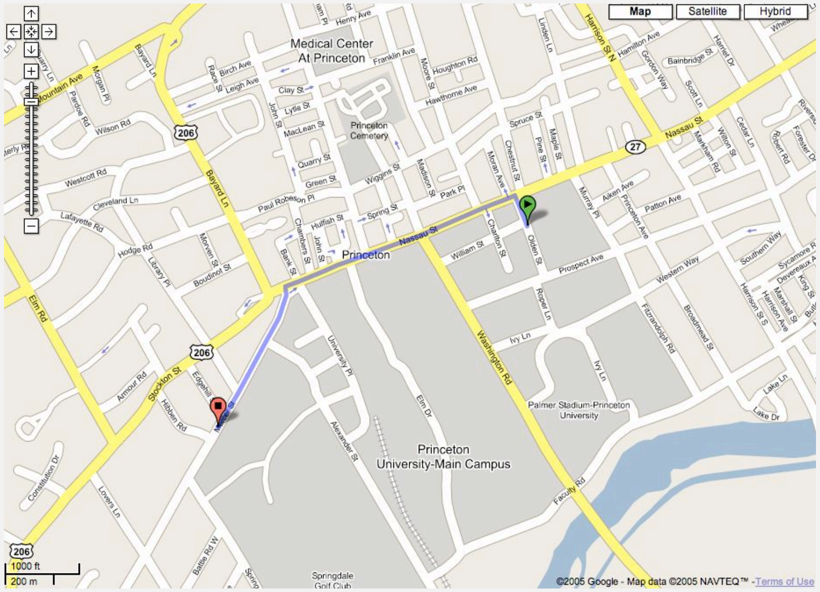


- Dijkstra's algorithm
- implementation
- negative weights

References: Algorithms in Java, 3rd edition, Chapter 21

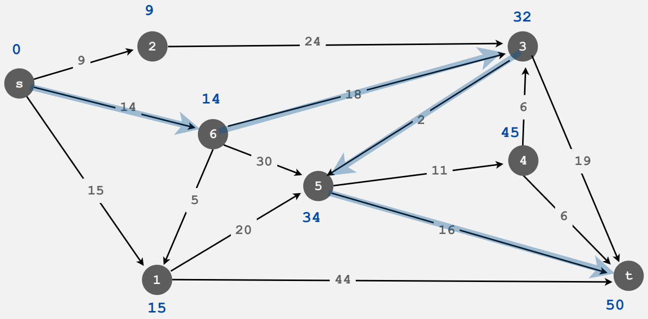
Algorithms in Java, 4th Edition · Robert Sedgewick and Kevin Wayne · Copyright © 2009 · November 11, 2009 7:04:36 AM

Google maps



Shortest paths in a weighted digraph

Given a weighted digraph G, find the shortest directed path from s to t.



shortest path: s→6→3→5→t

cost: 14 + 18 + 2 + 16 = 50

Shortest path versions

Which vertices?

- From one vertex to another.
- From one vertex to every other.
- Between all pairs of vertices.

Restrictions on edge weights?

- Nonnegative weights.
- Arbitrary weights.
- Euclidean weights.

Early history of shortest paths algorithms

Shimbel (1955). Information networks.

Ford (1956). RAND, economics of transportation.

Leyzorek, Gray, Johnson, Ladew, Meaker, Petry, Seitz (1957).
Combat Development Dept. of the Army Electronic Proving Ground.

Dantzig (1958). Simplex method for linear programming.

Bellman (1958). Dynamic programming.

Moore (1959). Routing long-distance telephone calls for Bell Labs.

Dijkstra (1959). Simpler and faster version of Ford's algorithm.

Shortest path applications

- Maps.
- Robot navigation.
- Texture mapping.
- Typesetting in TeX.
- Urban traffic planning.
- Optimal pipelining of VLSI chip.
- Telemarketer operator scheduling.
- Subroutine in advanced algorithms.
- Routing of telecommunications messages.
- Approximating piecewise linear functions.
- Network routing protocols (OSPF, BGP, RIP).
- Exploiting arbitrage opportunities in currency exchange.
- Optimal truck routing through given traffic congestion pattern.

Reference: Network Flows: Theory, Algorithms, and Applications, R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, Prentice Hall, 1993.

- **Dijkstra's algorithm**
- implementation
- negative weights

Edsger W. Dijkstra: select quote

> *“ The question of whether computers can think is like the question of whether submarines can swim. ”*
>
> *“ Do only what only you can do. ”*
>
> *“ In their capacity as a tool, computers will be but a ripple on the surface of our culture. In their capacity as intellectual challenge, they are without precedent in the cultural history of mankind. ”*
>
> *“ The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offence. ”*
>
> *“ APL is a mistake, carried through to perfection. It is the language of the future for the programming techniques of the past: it creates a new generation of coding bums. ”*

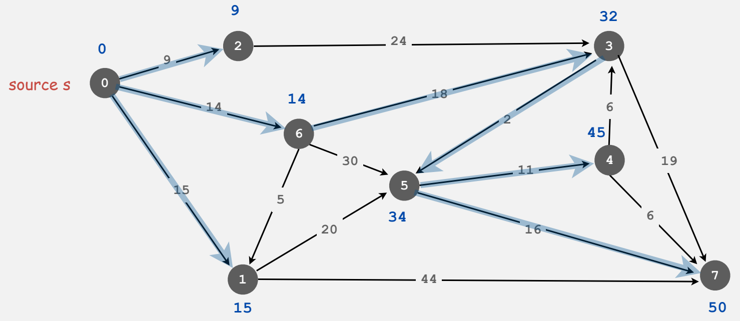
Edger Dijkstra
Turing award 1972

Single-source shortest-paths

Given. Weighted digraph `G`, source vertex `s`.

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

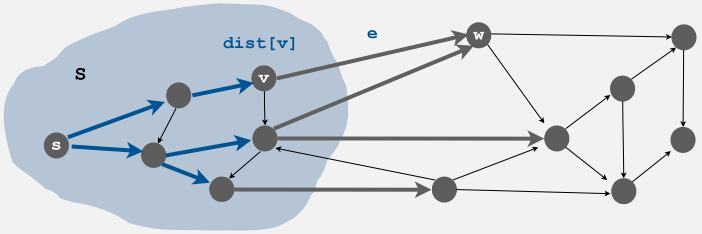
Observation. Use parent-link representation to store shortest path tree.



	0	1	2	3	4	5	6	7
`dist[v]`	0	15	9	32	45	34	14	50
`pred[v]`	-	0→1	0→2	6→3	5→4	3→5	0→6	5→7

Dijkstra's algorithm

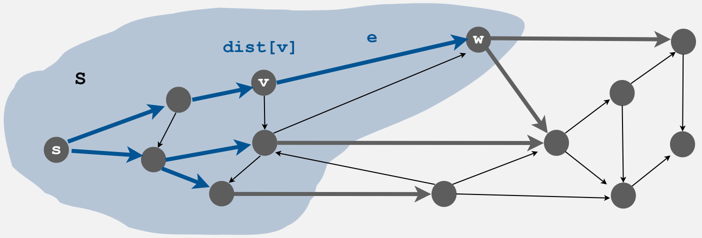
- Initialize S to `s`, `dist[s]` to `0`.
- Repeat until S contains all vertices connected to `s`:
  - find edge `e` with `v` in S and `w` not in S that minimizes `dist[v] + e.weight()`.



Dijkstra's algorithm

- Initialize S to `s`, `dist[s]` to `0`.
- Repeat until S contains all vertices connected to `s`:
  - find edge `e` with `v` in S and `w` not in S that minimizes `dist[v] + e.weight()`.
  - set `dist[w] = dist[v] + e.weight()` and `pred[w] = e`
  - add `w` to S

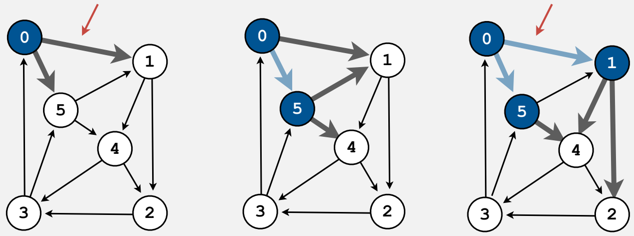
```
dist[w] = dist[v] + e.weight();
pred[w] = e;
```



Dijkstra's algorithm example

edge with v in S and w not in S

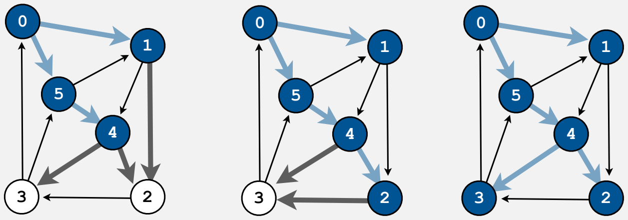
edge in shortest path tree



0→5 (.29)
0→1 (.41)

0→1 (.41)
5→4 (.50 = .29 + .21)
5→1 (.58 = .29 + .29)

5→4 (.50)
1→4 (.73 = .41 + .32)
1→2 (.92 = .41 + .51)



4→2 (.82 = .50 + .32)
4→3 (.86 = .50 + .36)
1→2 (.92)

4→3 (0.86)
2→3 (1.32 = .82 + .50)

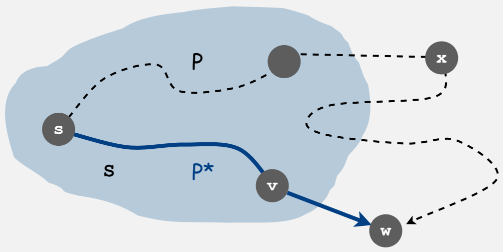
0→1	.41
0→5	.29
1→2	.51
1→4	.32
2→3	.50
3→0	.45
3→5	.38
4→2	.32
4→3	.36
5→1	.29
5→4	.21

Dijkstra's algorithm: correctness proof

Invariant. For `v` in S, `dist[v]` is the length of the shortest path from `s` to `v`.

Pf. (by induction on |S|)

- Let `w` be next vertex added to S.
- Let P* be the `s` → `w` path through `v`.
- Consider any other `s` → `w` path P, and let `x` be first node on path outside S.
- P is already as long as P* as soon as it reaches `x` by greedy choice.
- Thus, `dist[w]` is the length of the shortest path from `s` to `w`.



Shortest path trees

Remark. Dijkstra examines vertices in increasing distance from source.

Dijkstra's algorithm
implementation
negative weights

Weighted directed graph API

public class DirectedEdge implements Comparable<DirectedEdge>		
	`DirectedEdge(int v, int w, double weight)`	*create a weighted edge v→w*
`int`	`from()`	*vertex v*
`int`	`to()`	*vertex w*
`double`	`weight()`	*the weight*

public class WeightedDigraph		*weighted digraph data type*
	`WeightedDigraph(int V)`	*create an empty digraph with V vertices*
	`WeightedDigraph(In in)`	*create a digraph from input stream*
`void`	`addEdge(DirectedEdge e)`	*add a weighted edge from v to w*
`Iterable<DirectedEdge>`	`adj(int v)`	*return an iterator over edges leaving v*
`int`	`V()`	*return number of vertices*

Weighted digraph: adjacency-set implementation in Java

```
public class WeightedDigraph
{
   private final int V;
   private final SET<Edge>[] adj;

   public WeightedDigraph(int V)
   {
      this.V = V;
      adj = (SET<DirectedEdge>[]) new SET[V];
      for (int v = 0; v < V; v++)
         adj[v] = new SET<DirectedEdge>();
   }

   public void addEdge(DirectedEdge e)
   {
      int v = e.from();
      adj[v].add(e);
   }

   public Iterable<DirectedEdge> adj(int v)
   {  return adj[v];  }

   public int V()
   {  return V;  }
}
```

same as weighted undirected graph, but only add edge to v's adjacency set

---

Weighted directed edge: implementation in Java

```
public class DirectedEdge implements Comparable<DirectedEdge>
{
   private final int v, w;
   private final double weight;

   public DirectedEdge(int v, int w, double weight)
   {
      this.v = v;
      this.w = w;
      this.weight = weight;
   }

   public int from()    {  return v;       }
   public int to()      {  return w;       }
   public int weight()  {  return weight;  }

   public int compareTo(DirectedEdge that)
   {
      if (this.v < that.v) return -1;
      if (this.v > that.v) return +1;
      if (this.w < that.w) return -1;
      if (this.w > that.w) return +1;
      if (this.weight < that.weight) return -1;
      if (this.weight > that.weight) return +1;
      return 0;
    }
}
```

same as `Edge`, except `from()` and `to()` replace `either()` and `other()`

for use in a symbol table (allow parallel edges with different weights)

Shortest path data type

Design pattern.

- `Dijkstra` class is a `WeightedDigraph` client.
- Client query methods return distance and path iterator.

public class Dijkstra	
`Dijkstra(WeightedDigraph G, int s)`	*shortest path from s in graph G*
`double distanceTo(int v)`	*length of shortest path from s to v*
`Iterable <DirectedEdge> path(int v)`	*shortest path from s to v*

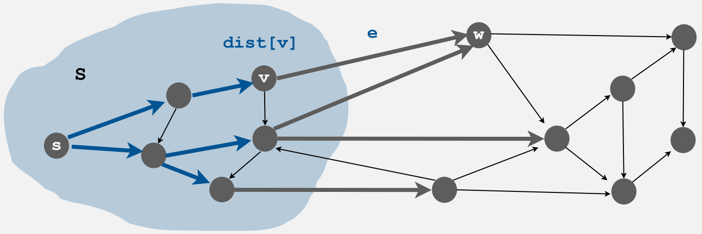
```
In in = new In("network.txt");
WeightedDigraph G = new WeightedDigraph(in);
int s = 0, t = G.V() - 1;
Dijktra dijkstra = new Dijkstra(G, s);
StdOut.println("distance = " + dijkstra.distanceTo(t));
for (DirectedEdge e : dijkstra.path(t))
    StdOut.println(e);
```

Dijkstra implementation challenge

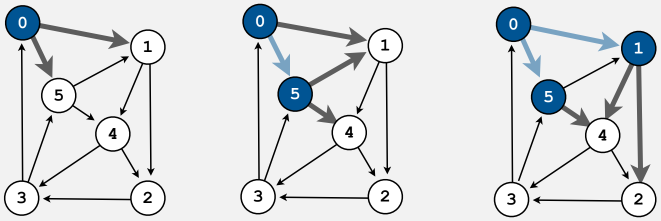
Find edge `e` with `v` in S and `w` not in S that minimizes `dist[v] + e.weight()`.

How difficult?

- Intractable.
- O(E) time. ← try all edges
- O(V) time.
- O(log E) time. ← Dijkstra with a binary heap
- O(log* E) time.
- Constant time.



Lazy Dijkstra's algorithm example

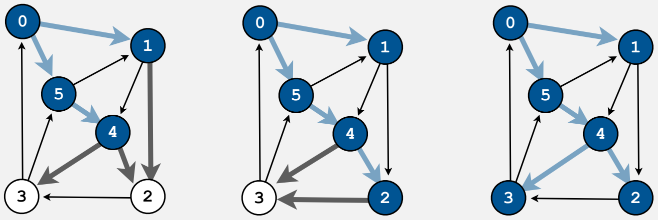


0→5 (.29)
0→1 (.41)

priority queue

0→1 (.41)
5→4 (.50 = .29 + .21)
5→1 (.58 = .29 + .29)

5→4 (.50)
1→4 (.73 = .41 + .32)
1→2 (.92 = .41 + .51)



1→4 (.73)
4→2 (.82 = .50 + .32)
4→3 (.86 = .50 + .36)
1→2 (.92)

4→3 (0.86)
1→2 (.92)
2→3 (1.32 = .82 + .50)

1→2 (.92)
2→3 (1.32)

0→1	.41
0→5	.29
1→2	.51
1→4	.32
2→3	.50
3→0	.45
3→5	.38
4→2	.32
4→3	.36
5→1	.29
5→4	.21

Lazy implementation of Dijkstra's algorithm

```
public class LazyDijkstra
{
   private boolean[] scanned;
   private double[] dist;
   private DirectedEdge[] pred;
   private MinPQ<DirectedEdge> pq;

   private class ByDistanceFromSource implements Comparator<DirectedEdge>
   {
      public int compare(DirectedEdge e, DirectedEdge f) {
         double dist1 = dist[e.from()] + e.weight();
         double dist2 = dist[f.from()] + f.weight();
         if      (dist1 < dist2) return -1;
         else if (dist1 > dist2) return +1;
         else                    return  0;
      }
   }

   public LazyDijkstra(WeightedDigraph G, int s) {
      scanned = new boolean[G.V()];
      pred = new DirectedEdge[G.V()];
      dist = new double[G.V()];
      pq = new MinPQ<DirectedEdge>(new ByDistanceFromSource());
      dijkstra(G, s);
   }
```

compare edges on pq by `dist[v] + e.weight()`

Lazy implementation of Dijkstra's algorithm

```
private void dijkstra(WeightedDigraph G, int s)
{
   scan(G, s);
   while (!pq.isEmpty()) {
      DirectedEdge e = pq.delMin();
      int v = e.from(), w = e.to();
      if (scanned[w]) continue;
         pred[w] = e;
         dist[w] = dist[v] + e.weight();
         scan(G, w);
   }
}

private void scan(WeightedDigraph G, int v) {
   scanned[v] = true;
   for (DirectedEdge e : G.adj(v))
      if (!scanned[e.to()]) pq.insert(e);
}
```

both endpoints in S

found shortest path to w

add all edges v->w to pq,
provided w not already in S

Dijkstra's algorithm running time

Proposition. Dijkstra's algorithm computes shortest paths in O(E log E) time.

Pf.

operation	frequency	time per op
delete min	E	log E
insert	E	log E

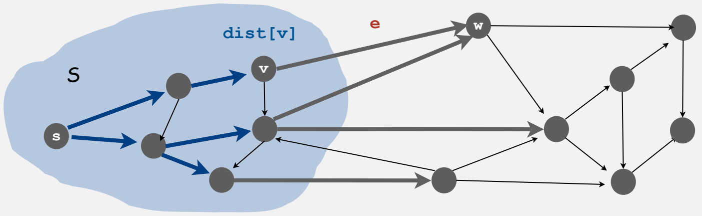
Improvements.

- Eagerly eliminate obsolete edges from PQ.
- Maintain on PQ at most one edge incident to each vertex v not in T
  ⇒ at most V edges on PQ.
- Use fancier priority queue: best in theory yields O(E + V log V).

Priority-first search

Insight. All of our graph-search methods are the same algorithm!
- Maintain a set of explored vertices S.
- Grow S by exploring edges with exactly one endpoint leaving S.

DFS. Take edge from vertex which was discovered most recently.
BFS. Take edge from vertex which was discovered least recently.
Prim. Take edge of minimum weight.
Dijkstra. Take edge to vertex that is closest to s.



Challenge. Express this insight in reusable Java code.

- Dijkstra's algorithm
- implementation
- negative weights

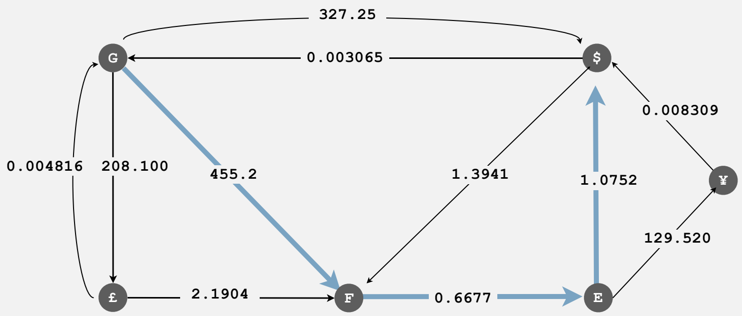
Currency conversion

Problem. Given currencies and exchange rates, what is 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	1.45999	1.0000	0.007721	0.6677	0.9303	304.028
Japanese Yen	189.50	129.520	1.0000	85.4694	120.400	39346.7
Swiss Franc	2.1904	1.4978	0.01574	1.0000	1.3941	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

Currency conversion

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

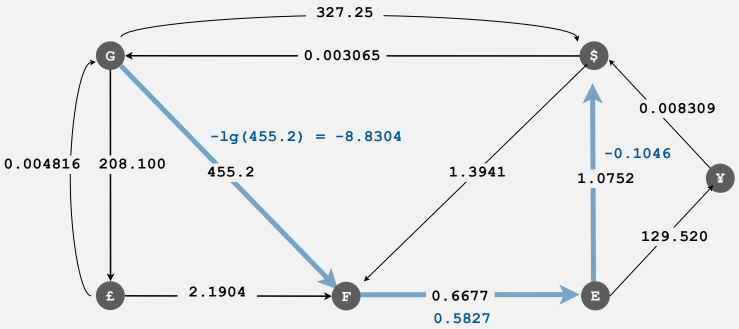


Challenge. Express as a shortest path problem.

Currency conversion

Reduce to shortest path problem by taking logs.

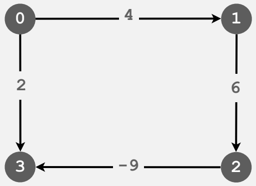
- Let weight of edge `v→w` be - lg (exchange rate from currency `v` to `w`).
- Multiplication turns to addition.
- Shortest path with given weights corresponds to best exchange sequence.



Challenge. Solve shortest path problem with negative weights.

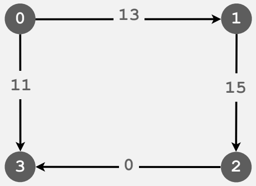
Shortest paths with negative weights: failed attempts

Dijkstra. Doesn't work with negative edge weights.



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

Re-weighting. Add a constant to every edge weight also doesn't work.

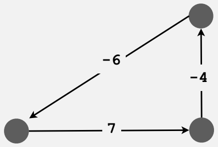


Adding 9 to each edge changes the shortest path
because it adds 9 to each edge;
wrong thing to do for paths with many edges.

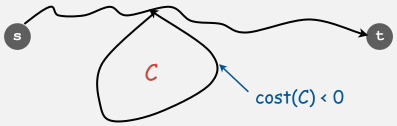
Bad news. Need a different algorithm.

Negative cycles

Def. A negative cycle is a directed cycle whose sum of edge weights is negative.



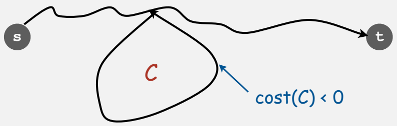
Observations. If negative cycle *C* is on a path from `s` to `t`, then shortest path can be made arbitrarily negative by spinning around cycle.



Worse news. Need a different problem.

Shortest paths with negative weights

Problem 1. Does a given digraph contain a negative cycle?
Problem 2. Find the shortest simple path from `s` to `t`.



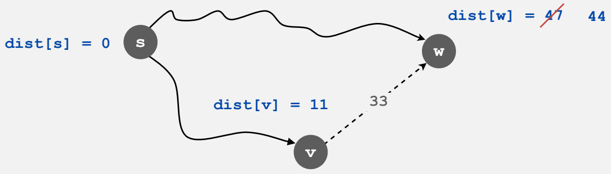
Bad news. Problem 2 is intractable.

Good news. Can solve problem 1 in O(VE) steps;
if no negative cycles, can solve problem 2 with same algorithm!

Edge relaxation

Relax edge `e` from `v` to `w`.

- `dist[v]` is length of some path from `s` to `v`.
- `dist[w]` is length of some path from `s` to `w`.
- If `v→w` gives a shorter path to `w` through `v`, update `dist[w]` and `pred[w]`.



```
int v = e.from(), w = e.to();
if (dist[w] > dist[v] + e.weight())
{
    dist[w] = dist[v] + e.weight();
    pred[w] = e;
}
```

Shortest paths with negative weights: dynamic programming algorithm

A simple solution that works!

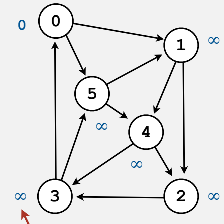
- Initialize `dist[v] = ∞`, `dist[s]= 0`.
- Repeat `V` times: relax each edge `e`.

```
for (int i = 1; i <= G.V(); i++)
   for (int v = 0; v < G.V(); v++)
      for (DirectedEdge e : G.adj(v))
      {
         int w = e.to();
         if (dist[w] > dist[v] + e.weight())
         {
            dist[w] = dist[v] + e.weight()
            pred[w] = e;
         }
      }
```

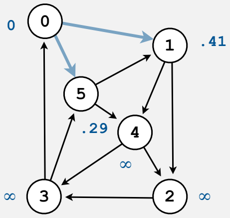
phase i

relax edge v-w

Dynamic programming algorithm trace

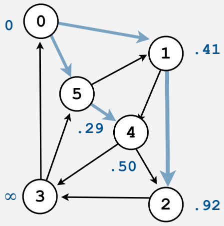


dist[v]



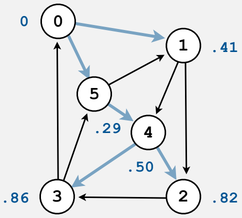
```
0→1  (.41 = 0 + .41)
0→5  (.50 = 0 + .50)
```

relaxed edges that update `dist[]`

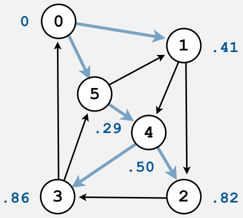


```
1→2  (.92 = .41 + .51)
1→4  (.73 = .41 + .32)
5→4  (.50 = .29 + .21)
```

0→1	.41
0→5	.29
1→2	.51
1→4	.32
2→3	.50
3→0	.45
3→5	.38
4→2	.32
4→3	.36
5→1	.29
5→4	.21



```
2→3  (1.33 = .83 + .50)
4→3  (.86 = .50 + .36)
4→2  (.82 = .50 + .32)
```



can stop early since
no entries in `dist[]` updated

Dynamic programming algorithm: analysis

Running time. Proportional to E V.

Invariant. At end of phase `i`, `dist[v]` ≤ length of any path from `s` to `v` using at most `i` edges.

Proposition. If there are no negative cycles, upon termination `dist[v]` is the length of the shortest path from from `s` to `v`.

and `pred[]` gives the shortest paths

Bellman-Ford-Moore algorithm

Observation. If `dist[v]` doesn't change during phase `i`, no need to relax any edge leaving `v` in phase `i+1`.

FIFO implementation. Maintain queue of vertices whose distance changed.

↑ be careful to keep at most one copy of each vertex on queue

Running time.

- Proportional to EV in worst case.
- Much faster than that in practice.

Single source shortest paths implementation: cost summary

	algorithm	worst case	typical case
nonnegative costs	Dijkstra (binary heap)	*E log E*	*E*
no negative cycles	dynamic programming	*E V*	*E V*
	Bellman-Ford	*E V*	*E*

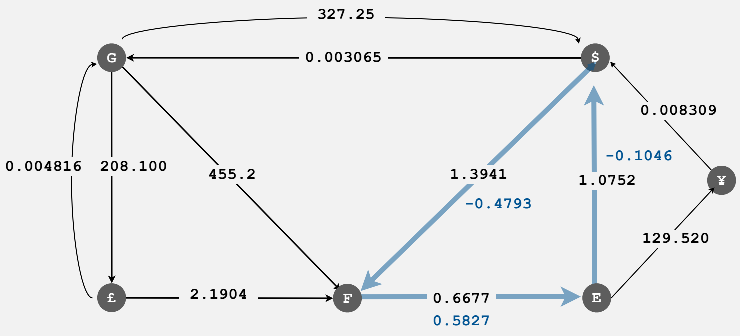
Remark 1. Negative weights makes the problem harder.

Remark 2. Negative cycles makes the problem intractable.

Shortest paths application: 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?



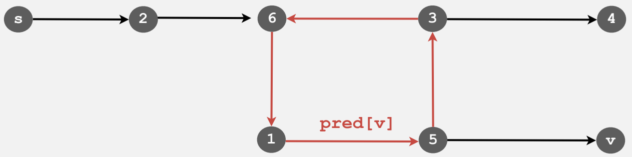
0.5827 - 0.1046 - 0.4793 < 0

Remark. Fastest algorithm is valuable!

Negative cycle detection

If there is a negative cycle reachable from s.

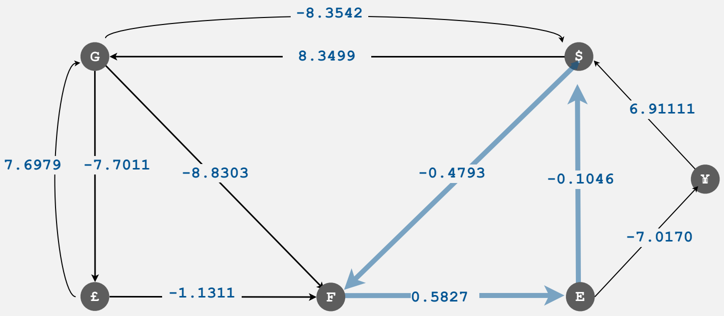
Bellman-Ford-Moore gets stuck in loop, updating vertices in cycle.



Proposition. 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

Goal. Identify a negative cycle (reachable from any vertex).



Solution. Initialize Bellman-Ford by setting `dist[v] = 0` for all vertices `v`.

Shortest paths summary

Dijkstra's algorithm.

- Nearly linear-time when weights are nonnegative.

Priority-first search.

- Generalization of Dijkstra's algorithm.
- Encompasses DFS, BFS, and Prim.
- Enables easy solution to many graph-processing problems.

Negative weights.

- Arise in applications.
- If negative cycles, problem is intractable (!)
- If no negative cycles, solvable via classic algorithms.

Shortest-paths is a broadly useful problem-solving model.