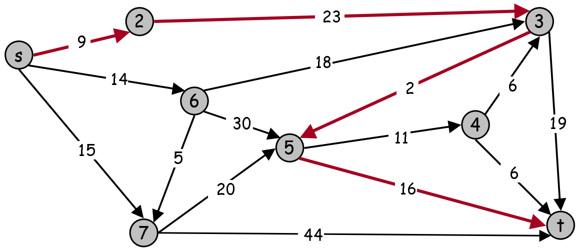
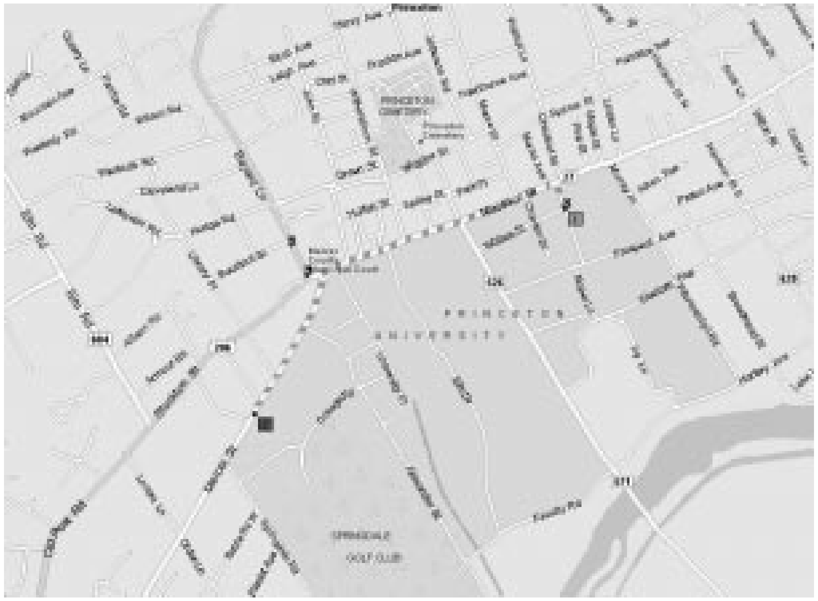
Shortest Paths



Fastest Route from CS Dept to Einstein's House



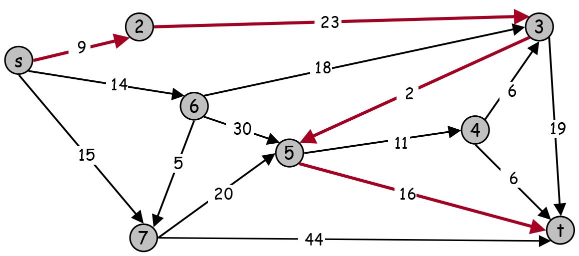
Shortest Path Problem

Shortest path network.

- Directed graph.
- Source s, destination t.
- Arc costs c(v, w).

Shortest path problem: find shortest directed path from s to t.

- Cost of path = sum of arc costs in path.



Cost of path s - 2 - 3 - 5 - t
= 9 + 23 + 2 + 16
= 48.

Graphs

Graph	Vertices	Edges
communication	telephones, computers	fiber optic cables
circuits	gates, registers, processors	wires
mechanical	joints	rods, beams, springs
hydraulic	reservoirs, pumping stations	pipelines
financial	stocks, currency	transactions
transportation	street intersections, airports	highways, airway routes
scheduling	tasks	precedence constraints
software systems	functions	function calls
internet	web pages	hyperlinks
games	board positions	legal moves
social relationship	people, actors	friendships, movie casts

Applications

More applications.

- Urban traffic planning.
- Routing of telecommunications messages.
- Approximating piecewise linear functions.
- Exploiting arbitrage opportunities in currency exchange.
- Typesetting in TeX.
- Tramp steamer problem.
- Telemarketer operator scheduling.
- Optimal pipelining of VLSI chip.
- Subroutine in higher level algorithms.

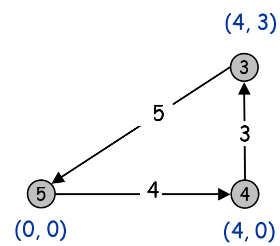
Reference: *Network Flows: Theory, Algorithms, and Applications*, R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, Prentice Hall, 1993.

Shortest Path

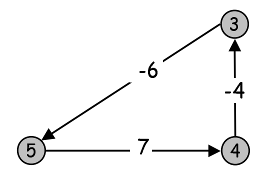
Versions of the problem that we consider.

- Single source.
- All-pairs.
- Arc costs are $\geq$ 0.
- Points and distances are Euclidean.
- Arc costs can be < 0, but no negative cycles.
- Arc costs can be arbitrary.

next programming assignment



Euclidean



Negative cycle

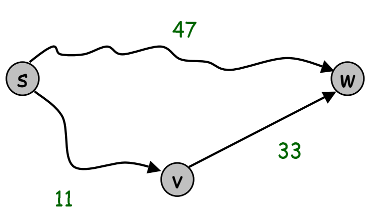
Shortest Path: Relaxation

Valid weights $\pi(v)$.

- For all v, $\pi(v)$ is length of some path from s to v.
- Provides lower bound on length of shortest path from s to v.

Relaxation.

- Consider edge v-w with weight c(v, w).
- If $\pi(w) > \pi(v) + c(v, w)$ then update $\pi(w) = \pi(v) + c(v, w)$.
- Found better route: path from s to v, then arc v-w.



Dijkstra's Algorithm: Implementation

Dijkstra's algorithm.

- Initialize $S = \phi$, $\pi[s] = 0$, pred[s] = s, $\pi[v] = \infty$, pred[v] = -1.
- Insert all nodes onto PQ.
- Repeatedly delete node v with min $\pi[v]$ from PQ.
  - add v to S
  - for each v-w, if $\pi[w] > \pi[v] + c(v, w)$ then update $\pi[w] = \pi[v] + c(v, w)$

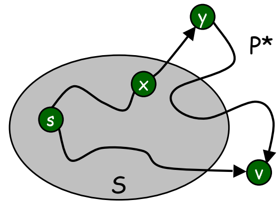
```
while (!PQisempty()) {
   v = PQdelmin();
   for (t = G->adj[v]; t != NULL; t = t->next) {
      w = t->w;
      if (pi[v] + t->wt < pi[w]) {
         pi[w] = pi[v] + t->wt;
         pred[w] = v;
      }
}
```

relax

decrease key

Main Loop

Dijkstra's Algorithm: Proof of Correctness



Invariant. For each vertex $v \in S$, $\pi[v] = d^*(s, v)$.

Proof: by induction on $|S|$.
Base case: $|S| = 0$ is trivial.
Induction step:
- Suppose Dijkstra's algorithm adds vertex v to S.
- $\pi[v]$ is the length of some path from s to v.
- If $\pi[v]$ is not the length of the shortest s-v path, then let P* be a shortest s-v path.
- P* must use an edge that leaves S, say (x, y)

then $\pi[v]$	$> d^*(s, v)$	assumption
	$= d^*(s, x) + d(x, y) + d^*(y, v)$	optimal substructure
	$\geq d^*(s, x) + d(x, y)$	nonnegative weights
	$= \pi[x] + d(x, y)$	inductive hypothesis
	$\geq \pi[y]$	algorithm

- So Dijkstra's algorithm would have selected y instead of v.

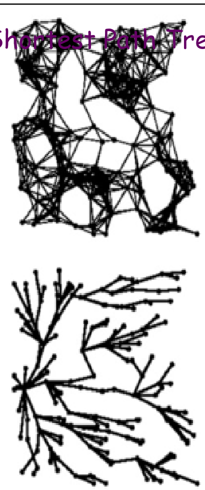
Dijkstra's Algorithm: Implementation Cost Summary

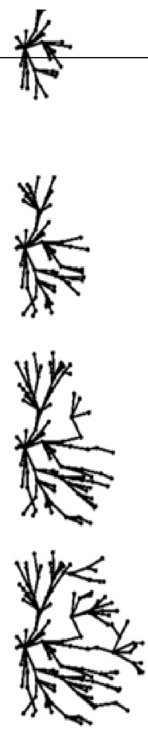
		Priority Queue			
Operation	Dijkstra	Array	Binary heap	d-way Heap	Fib heap †
insert	V	V	log V	$d \log_d V$	1
delete-min	V	V	log V	$d \log_d V$	log V
decrease-key	E	1	log V	$\log_d V$	1
is-empty	V	1	1	1	1
total		V^2	E log V	$E \log_{E/V} V$	E + V log V

† Individual ops are amortized bounds

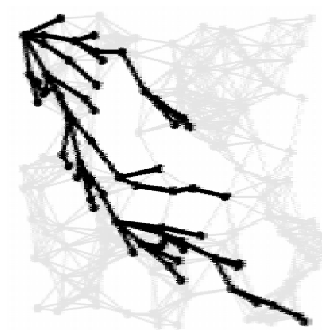
Exactly the same as Prim's MST algorithm!
- PFS: variations on a theme.

Shortest Path Trees





Shortest Path in Euclidean Graphs



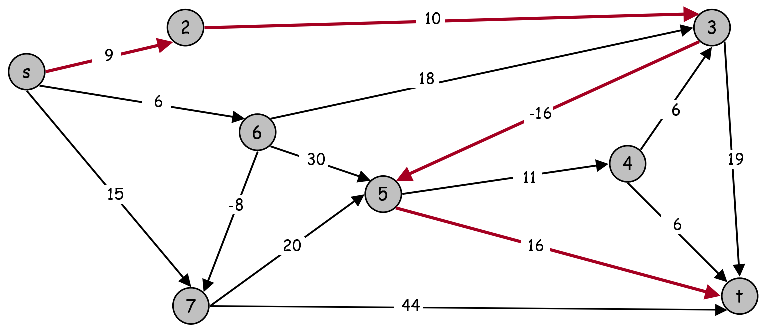
Euclidean graph (map).
- Vertices are points in the plane.
- Edges weights are Euclidean distances.

Sublinear algorithm.
- Assume graph is already in memory.
- Start Dijkstra at s.
- Stop as soon as you reach t.

Exploit geometry.
- Use $\pi[v]$ = length of some s-v path + Euclidean distance from v to t.
- $\pi[v]$ is a lower bound on length of shortest s-t path.
- Dijkstra proof of correctness still works.
- Typically only O(sqrt(V)) nodes examined for sparse graphs.
- A* algorithm.

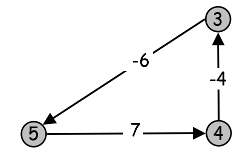
Shortest Path With Negative Weights

What if we allow negative cost arcs?

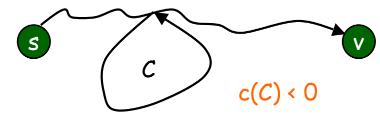


Shortest Path With No Negative Cycles

Obstacle: negative cost cycle.



If some path from s to v contains a negative cost cycle, shortest s-v path does not exist. Otherwise, there exists one that is simple.

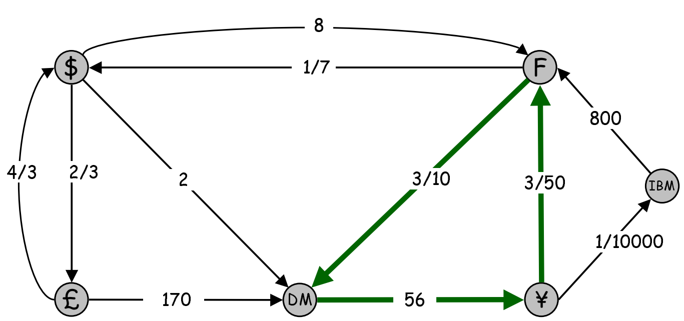


Algorithmic goal: find shortest path or output a negative costs cycle.

Currency Conversion Application

Currency conversion.

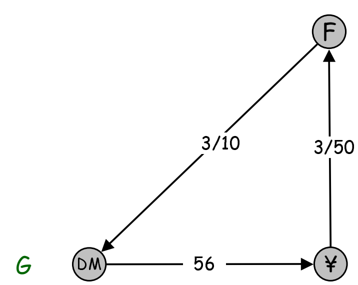
- Given V currencies (financial instruments) and exchange rates between pairs of currencies, is there an arbitrage opportunity?
- Fastest algorithm very valuable!



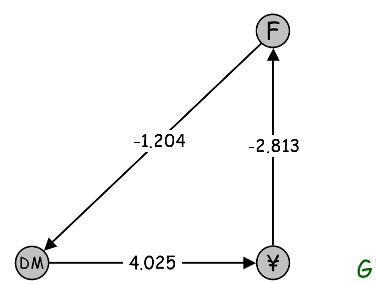
Currency Conversion Application

Reduction.

- Let γ(v,w) be exchange rate from currency v to w.
- Let c(v,w) = - log γ(v,w).
- Arbitrage opportunities in G correspond to negative cycles in G'.



γ = exchange rate



c = - log γ

Bellman-Ford-Moore Algorithm

Bellman-Ford-Moore.

- Initialize $\pi[v] = \infty$, pred[v] = -1, $\pi[s] = 0$, pred[s] = s.
- Repeat V times: relax each edge v-w

```
for (i = 1; i <= V; i++)                    ⟸ phase i
   for (v = 0; v < V; v++)
      for (t = G->adj[v]; t != NULL; t = t->next) {
         w = t->w;
         if (pi[v] + t->wt < pi[w]) {
            pi[w] = pi[v] + t->wt;           ⟸ relax
            pred[w] = v;
         }
      }
```

Invariant. At end of phase i, $\pi[v] \leq$ length of shortest path from s to v using at most i edges.

Running time. Θ(E V).

Bellman-Ford-Moore Algorithm

Practical improvement.

- If $\pi[v]$ doesn't change during phase i, don't relax any edges of the form v-w in phase i + 1.
- Programming solution: maintain queue of nodes that have changed.

```
wt[s] = 0;
QUEUEput(s);
while(!QUEUEisempty()) {
    v = QUEUEget();
    for (t = G->adj[v]; t != NULL; t = t->next) {
        w = t->w;
        if (pi[v] + t->wt < pi[w]) {
            pi[w] = pi[v] + t->wt;        ⟸ relax
            pred[w] = v;
            QUEUEput(w);
        }                  ↖ no duplicates
    }
}
```

Running time. Still O(E V) worst-case, but now O(E) in practice.

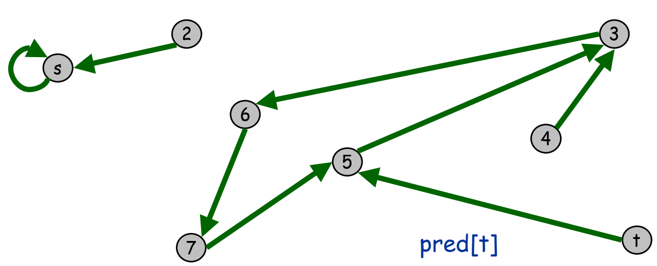
Bellman-Ford-Moore Algorithm

Finding the shortest path itself.

- Trace back pred[v] as in Dijkstra's algorithm.

Detecting a negative cycle.

- If any node v is enqueued V times, there must be a negative cycle.
- Fact: can trace back pred[v] to find cycle.



All Pairs Shortest Path

All pairs shortest path: Find the shortest path from v to w for all v, w.

Nonnegative weights.

- Run Dijkstra's algorithm V times.
- O(EV log V) time.

Negative weights, no negative cycles.

- Run Bellman-Ford once to preprocess graph.
- Run Dijkstra V times.
- O(EV log V) time.

Floyd-Warshall.

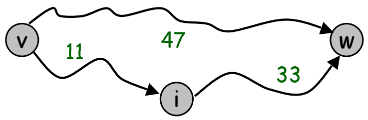
- Solve all-pairs problem directly in $\Theta(V^3)$ time.
- Only worthwhile on dense graphs.

Best in theory for sparse graphs: $O(EV + V^2 \log \log V)$.

Floyd's Algorithm

Floyd's algorithm.

- Initialize d[v][w] = c(v, w) if v-w exists, d[v][w] = ∞ otherwise.
- Want shorter path from v to w?
- Take path from v to i and then from i to w if shorter.




```
for (i = 0; i < G->V; i++)
   for (v = 0; v < G->V; v++)
      for (w = 0; w < G->V; w++)
         if (d[v][w] > d[v][i] + d[i][w])
            d[v][w] = d[v][i] + d[i][w];
```

Invariant. After ith iteration d[v][w] is shortest path from v to w whose intermediate nodes are 0, 1, …, i.

Shortest Path Variants

Variants of directed shortest path:

- Unit weights: O(E + V) using BFS.
- DAGs: O(E + V) using topological sort.
- Arc costs between –C and C: $O(EV^{1/2} \log C)$ by reducing to assignment problem.

Undirected shortest path.

- Nonnegative weights: O(E + V) by Thorup.
- No negative cycles: $O(EV + V^2 \log V)$ by reducing to weighted non-bipartite matching.