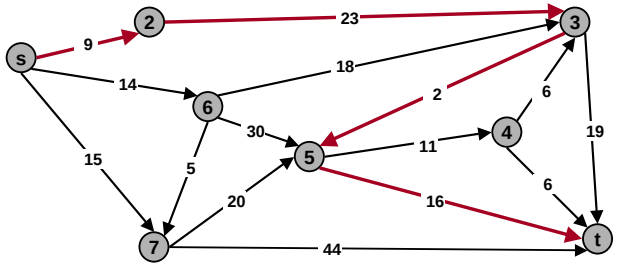
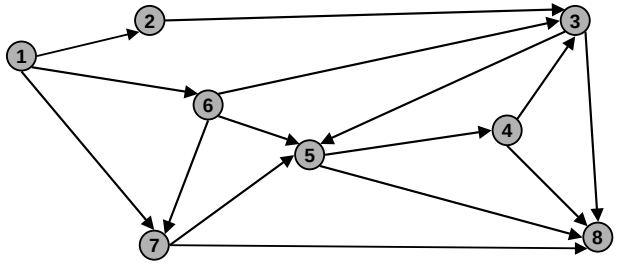
Greed: Shortest Path



Directed Graph

Directed graph: G = (V, E) .

- V = set of vertices or nodes.
- $E \subseteq V \times V$ = set of edges or arcs.
- n = |V|, m = |E|.
- Directed path: s - 2 - 3 - 5 - t.
  - simple
- Directed cycle: 5 - 4 - 3 - 5.



Networks

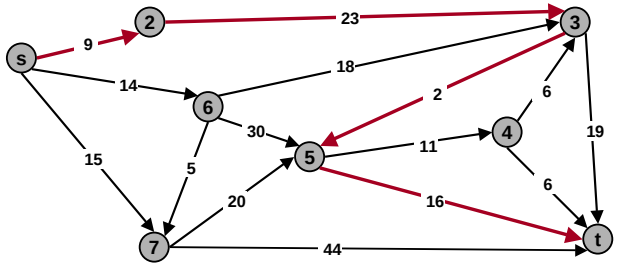
Network	Nodes	Arcs	Flow
communication	telephone exchanges, computers, satellites	cables, fiber optics, microwave relays	voice, video, packets
circuits	gates, registers, processors	wires	current
mechanical	joints	rods, beams, springs	heat, energy
hydraulic	reservoirs, pumping stations, lakes	pipelines	fluid, oil
financial	stocks, currency	transactions	money
transportation	airports, rail yards, street intersections	highways, railbeds, airway routes	freight, vehicles, passengers

Shortest Path Network

Shortest path network: (V, E, s, t, c) .

- Directed graph (V, E).
- Source $s \in V$, sink $t \in V$.
- Arc costs c(v, w).
- Cost of path = sum of arc costs in path.

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



Shortest Path

Shortest path problem. (CLR 25.1-25.2)

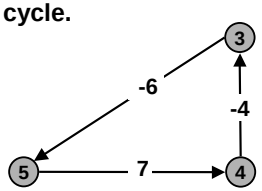
- Shortest path network (V, E, s, t, c).
- Find shortest directed path from s to t.

Assumptions.

- Network contains directed path from s to every other node.
- Network does not contain a negative cost cycle.

Application.

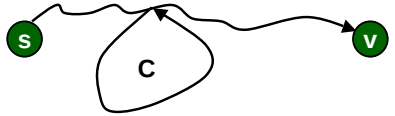
- Online directions.



Shortest Path: Existence

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

⇒ If negative cycle, can produce arbitrarily negative path by traversing cycle enough times.



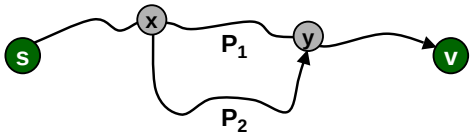
c(C) < 0

⇐ If no negative cycle, can remove cycles without increasing cost.

Shortest Path: Properties

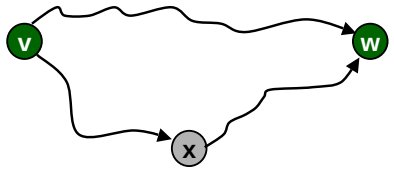
Optimal substructure property. All sub-paths of shortest paths are shortest paths.

- Let P_1 be x-y sub-path of shortest s-v path P.
- Let P_2 be any x-y path.
- $c(P_1) \leq c(P_2)$, otherwise P not shortest s-v path.



Triangle inequality.

- Let d*(v, w) be the length of the shortest path from v to w.
- Then, d*(v, w) ≤ d*(v, x) + d*(x, w)



Dijkstra's Algorithm

Upon termination.

- π(v) = distance of shortest s-v path.
- pred(v) gives shortest path.

Dijkstra's Algorithm

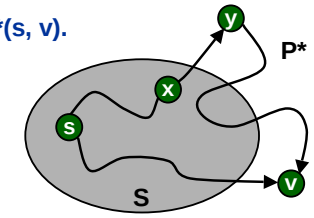
```
for each v ∈ V
   π(v) ← ∞
   pred(v) ← nil
π(s) ← 0
S ← φ
init(Q)
for each v ∈ V
   insert(v, Q)
while (Q ≠ φ)
   v = delete-min(Q)
   S ← S ∪ {v}
   for each w s.t (v,w) ∈ E
      if π(w) > π(v) + c(v,w)
         π(w) ← π(v) + c(v,w)
         pred(w) ← v
```

decrease-key

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 the 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 lengths
	$= \pi(x) + d(x, y)$	inductive hypothesis
	$\geq \pi(y)$	algorithm

so Dijkstra's algorithm would have selected y instead of v

Priority Queues and Heaps (CLR 20, 21)

		Heaps			
Operation	Linked List	Binary	Binomial	Fibonacci *	Relaxed
make-heap	1	1	1	1	1
insert	1	log N	log N	1	1
find-min	N	1	log N	1	1
delete-min	N	log N	log N	log N	log N
union	1	N	log N	1	1
decrease-key	1	log N	log N	1	1
delete	N	log N	log N	log N	log N
is-empty	1	1	1	1	1

Dijkstra
1 make-heap
n insert
n delete-min
m decrease-key

n (n) + m(1) = $O(n^2)$

n (log n) + m(log n) = O(m log n)

n (log n) + m(1) = O(m + n log n)

Shortest Path Extensions

Variants of shortest path:

- Undirected graph.
  - O(m + n) using Thorup's algorithm
- Negative weights but no negative cycles.
  - O(mn) using Bellman-Ford
- Unit weights.
  - O(m + n) using breadth first search
- Integer weights between 0 and constant C.
- DAGs.
  - O(m + n) using topological sort
- All-pairs.
  - $O(n^3)$ using Floyd-Warshall
  - O(mn + n log log n) using Pettie's algorithm