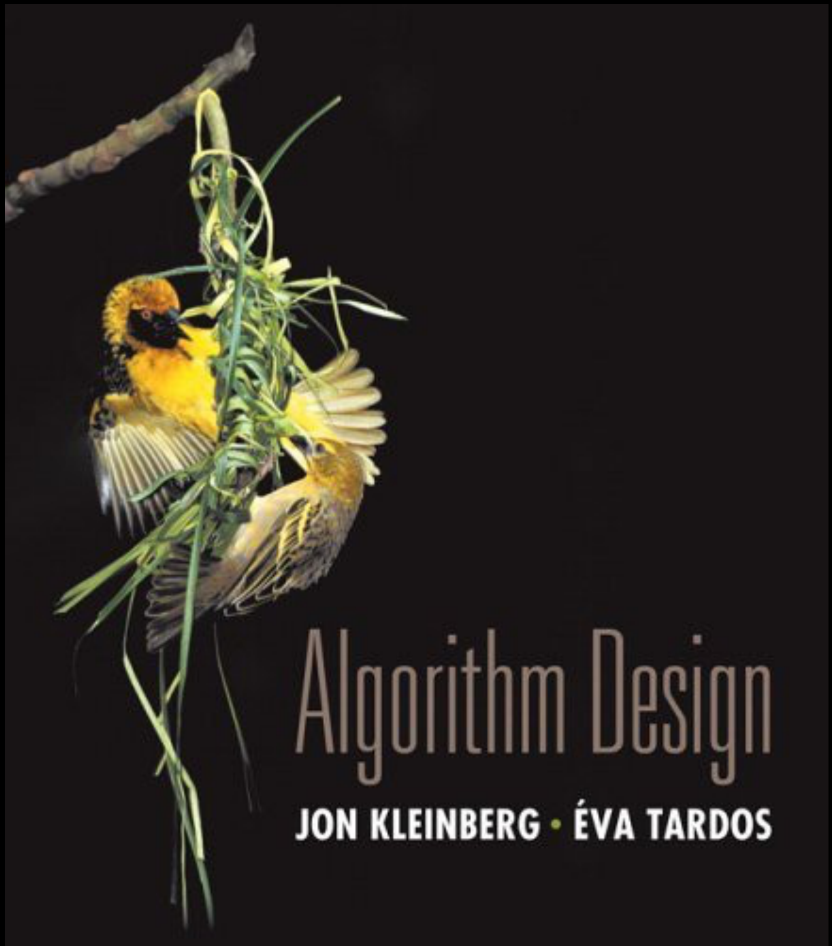
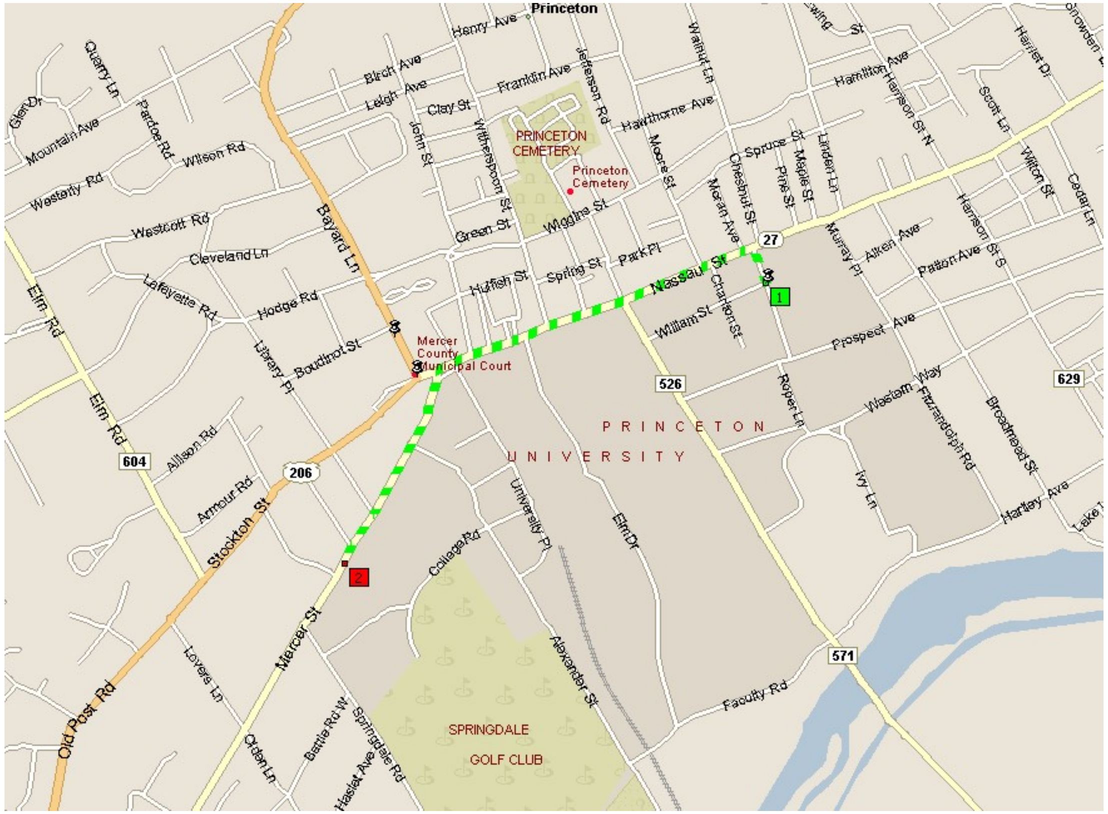


Chapter 4

Greedy Algorithms

Slides by Kevin Wayne.

4.4 Shortest Paths in a Graph



shortest path from Princeton CS department to Einstein's house

Shortest Path Problem

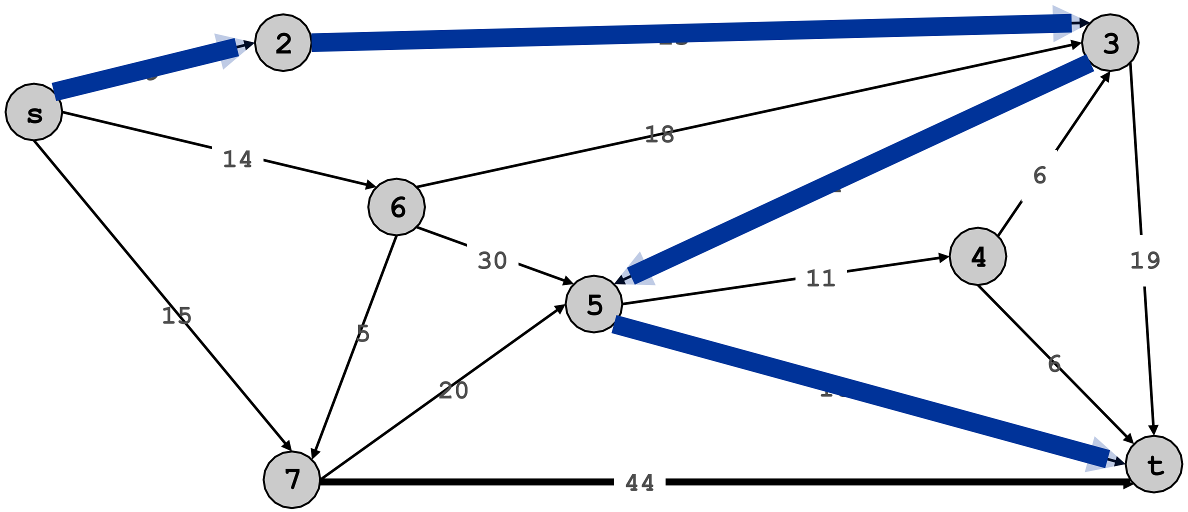
Input: Weighted connected graph <G = (V, E), ℓ:E→R+>; Source **s** in **V**

(Length ℓ_e = length of edge e)

Feasible Solution: Any set of **simple** paths from **s** to **t**, for all **t in V**.

Goal: for any **t in V, minimize the cost** of the **s-t path**

cost of path = sum of edge costs in path



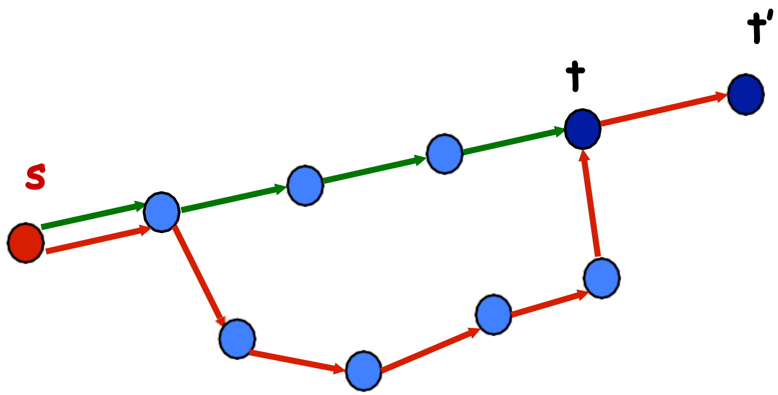
Cost of path s-2-3-5-t
= 9 + 23 + 2 + 16
= 50.

Shortest Path Trees

Theorem. For any Input **[<G = (V, E), ℓ:E✐R+>; s in V]**, there always exists an **optimal** solution that forms a *Spanning Tree* for ***G***.

Proof. Easy consequence of the **Principle** of **Sub-Optimality** of Shortest Paths in a graph with positive weights:

"***Any sub-path of a shortest path is a shortest path***."



If path is an s-t' shortest path
Then sub-path s-t must be a shortest path as well ✐ the s-t path can be removed from the optimal solution

Dijkstra's Algorithm

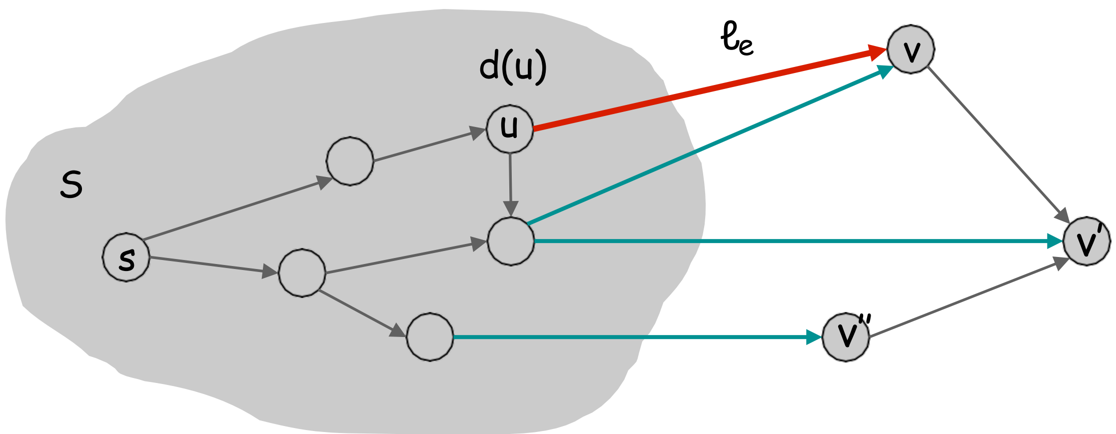
Dijkstra's algorithm.

- Maintain a set of explored nodes **S** for which we have determined the shortest path distance **d(u)** from s to **u**.
- Initialize **S** = { **s**}, d(**s**) = 0.
- Repeatedly choose unexplored node v which minimizes

$$\pi(v) = \min_{e = (u, v)\,:\, u \in S} d(u) + \ell_e ,$$

shortest path to some u in explored part, followed by a single edge (u, v)

add v to **S**, set d(v) = π(v), and store the father of v (i.e u)



Dijkstra's Algorithm

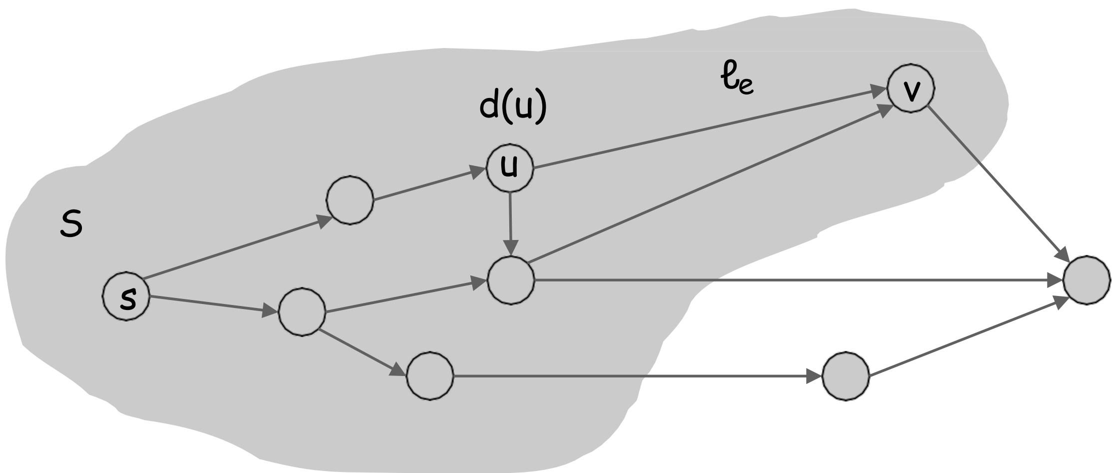
Dijkstra's algorithm (Overall Scheme).

- Maintain a set of explored nodes **S** for which we have determined the shortest path distance **d(u)** from node *s* to node *u*.
- Initialize **S** = { s }, **d(s)** = 0.
- Repeatedly choose unexplored node v which minimizes

$$\pi(v) = \min_{e=(u,v)\,:\,u\in S} d(u) + \ell_e ,$$

add v to **S**, set d(v) = π(v), and
store the father of v (i.e u)

shortest path to some u in explored part, followed by a single edge (u, v)
How to do it?



Dijkstra's Algorithm: Proof of Correctness

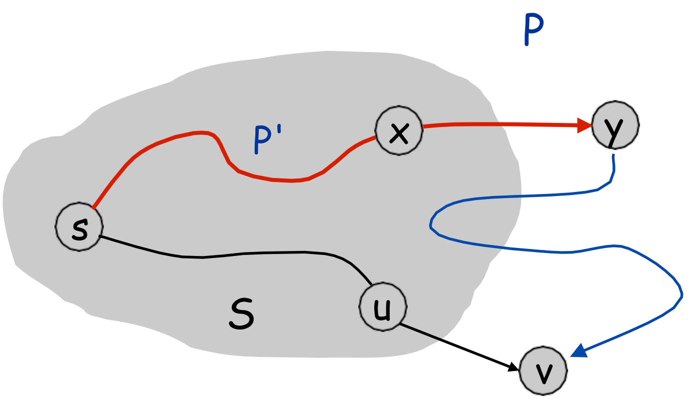
THM 1. For each node **u** ∈ **S**, **d(u)** is the **length** of the **shortest** **s-u** path.

Pf. (by induction on |S|)

Base case: |S| = 1 is trivial.

Inductive hypothesis: Assume true for |S| = k ≥ 1.

- Let v be next node added to **S**, and let u-v be the chosen edge.
- The shortest s-u path plus (u, v) is an **s-v** path of length $\pi(v)$.
- Consider any s-v path **P**. We'll see that it's no shorter than $\pi(v)$.
- Let **x-y** be the first edge in **P** that leaves **S**, and let **P'** be the subpath to **x**.
- P is already too long as soon as it leaves S.



$$\ell(P) \geq \ell(P') + \ell(x,y) \geq d(x) + \ell(x, y) \geq \pi(y) \geq \pi(v)$$

↑ nonnegative weights ↑ inductive hypothesis ↑ defn of $\pi(y)$ ↑ Dijkstra chose v instead of y

Dijkstra's Algorithm: Property of its execution

Corollary. For any **t=0,…,n**, let **v(t)** be the **t-th** node selected by D.'s Algorithm. Then, **v(t) is the t-th closest node to the source node s.**

Proof.

By induction on t (similar to proof of THM 1).

Do as excercise.

Dijkstra's Algorithm: Implementation

For each unexplored node, explicitly maintain $\pi(v) = \min_{e = (u,v) \,:\, u \in S} d(u) + \ell_e$.

- Next node to explore = node with minimum $\pi(v)$.
- When exploring v, **for each** incident edge $e = (v, w)$, update

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

Efficient implementation. Maintain a priority queue of unexplored nodes, prioritized by $\pi(v)$.

PQ Operation	Dijkstra	Array	Binary heap
Insert	n	n	log n
ExtractMin	n	n	log n
ChangeKey	m	1	log n
IsEmpty	n	1	1
Total		n^2	m log n

† Individual ops are amortized bounds

Edsger W. Dijkstra

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.