SINGLE-SOURCE SHORTEST PATHS (SSSP)

SSSP I

Given a road map with distances, find shortest route from city A (say, Lund) to city B (say, Stockholm)

Model as graph problem

- intersections $\longleftrightarrow$ vertices
- road segments $\longleftrightarrow$ edges (DIRECTED)
- distances $\longleftrightarrow$ edge weights

$\boxed{\text{Weight function}}$ $w : E \to \mathbb{R}$

For P a path from u to v, use notation $P : u \rightsquigarrow v$

For $P = v_0, v_1, \ldots, v_n$ define $\boxed{\text{WEIGHT of path}}$ as

$$w(P) = \sum_{i=1}^{n} w((v_{i-1}, v_i))$$

From now on, write sometimes $w(u,v)$ instead of $w((u,v))$ just to avoid unnecessary typing

The $\boxed{\text{DISTANCE}}$ from u to v is

$$\boxed{\delta(u,v)} = \begin{cases} \min\{w(P) \mid P : u \rightsquigarrow v\} & \text{if exists paths from } u \text{ to } v \\ \infty & \text{otherwise} \end{cases}$$

Refer to $G = (V, E)$ with weight function $w: E \to \mathbb{R}$ as **WEIGHTED GRAPH**

A path $P: u \rightsquigarrow v$ for which $w(P) = \delta(u,v)$ is a **SHORTEST PATH**

<u>NOTE</u> that shortest paths need not be unique — there could be many

① **SINGLE-PAIR SHORTEST PATH PROBLEM**

Given $G = (V, E)$, $w: E \to \mathbb{R}$, and pair (u, v), find a shortest path from u to v

VARIANTS OF THIS PROBLEM

② **SINGLE-SOURCE SHORTEST PATH PROBLEM**

Find shortest paths from specified "source vertex" s to all other vertices

③ **SINGLE-DESTINATION SHORTEST PATH PROBLEM**

Find shortest paths from all vertices to specified "destination vertex" t

④ **ALL-PAIRS SHORTEST PATH PROBLEM**

Find shortest paths from u to v for all $u, v \in V$.

If we can solve (2), then clearly solve (1). And best known algorithms for (1) in fact solve (2) in same worst-case time!

So from now on focus on single-source shortest path (SSSP)

(3): Reverse directions of all edges and use solution of SSSP in (2)

(4) For all-pairs shortest paths (APSP), can run SSSP algorithm $|V|$ times, but it is possible to be smarter

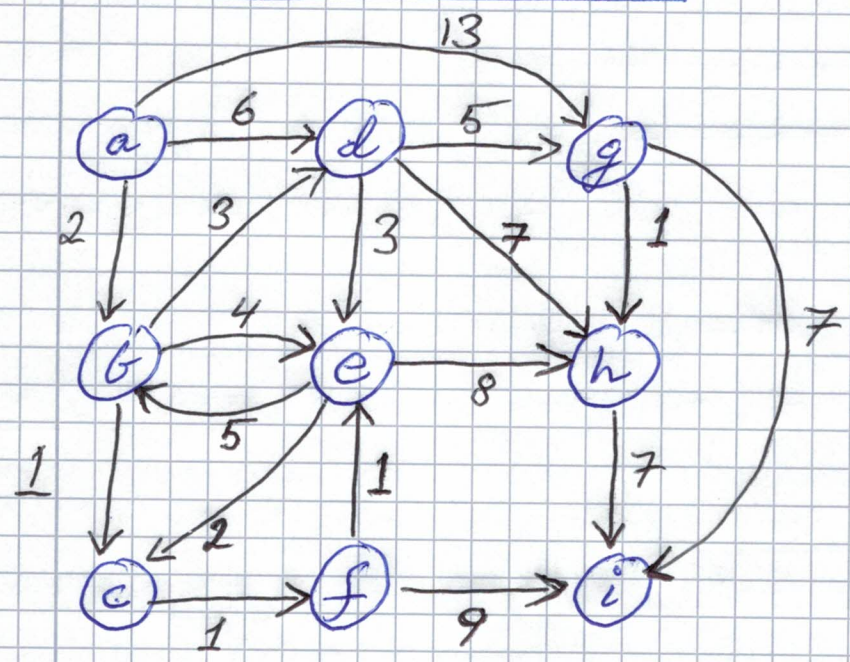
We won't talk about this — see Chapter 25 in CLRS for APSP.

For SSSP, how do we output solutions as quickly and concisely as possible?

Turns out solutions can be represented as directed tree rooted in source vertex s. Unique path to v in this tree will be shortest path from s to v.

(That this is so requires an argument, though — we will get there soon)

EXAMPLE GRAPH



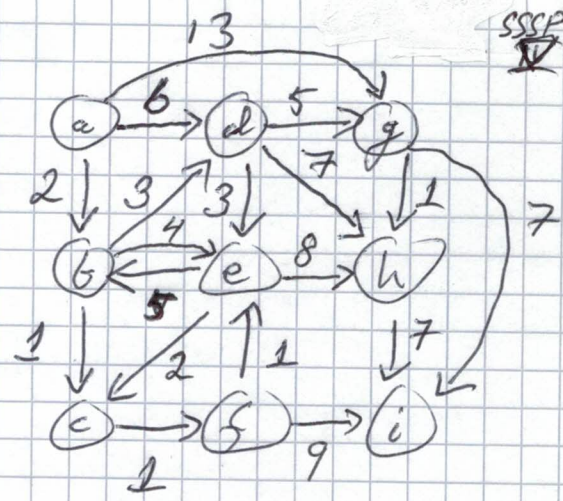
How can we compute shortest paths from a to all other vertices?

What is a shortest path from a to e, just to be concrete?

PLAN FOR TODAY

- Design algorithm
- Illustrate it on the above graph
- Argue correctness (a bit tricky)
- Analyze time complexity

Example graph



Shortest path from a to e is

$\boxed{a, b, c, f, e}$

weight: 5

For simplicity, let us assume that there is a path from s to every vertex v (not important)

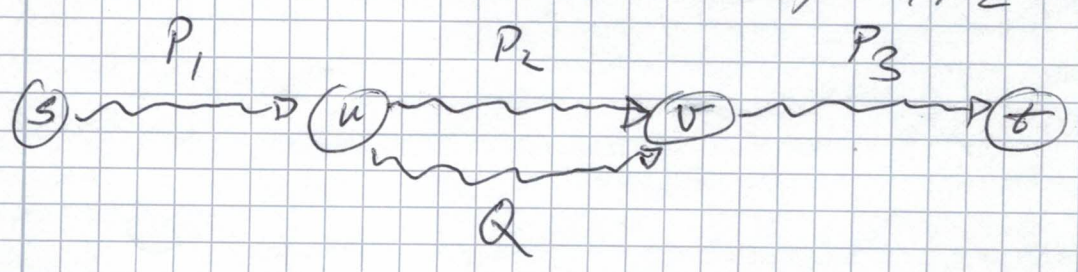
Let us also assume that all weights are non-negative (important assumption)

A basic and important property is as follows

LEMMA 1 Every subpath of a shortest path is a shortest path

Proof By contradiction. Let $P: s \rightsquigarrow t$ be a shortest path that goes through u and v, and suppose the subpath $P_2: u \rightsquigarrow v$ is not shortest but that $\exists$ shorter path $Q: u \rightsquigarrow v$

Write P as the concatenation of P_1, P_2 & P_3



We have $w(Q) < w(P_2)$

But then consider the path

$P' = P_1 \circ Q \circ P_3 \quad : s \rightsquigarrow t$

We have

$$\begin{aligned} w(P') &= w(P_1) + w(Q) + w(P_3) \\ &< w(P_1) + w(P_2) + w(P_3) \\ &= w(P) \end{aligned}$$

But this contradicts that P was the shortest path. Hence every subpath of a shortest path is a shortest path $\boxtimes$

DIJKSTRA'S ALGORITHM

Maintain distance estimates $v.d \quad \forall v \in V$

At the beginning

$s.d = 0$

$v.d = \infty$ for $v \neq s$

Build a directed tree T rooted in s

In each step, find $u \in V \setminus V(T)$ with smallest distance estimate $u.d$

Add u to T

Argue that $u.d = \delta(s,u)$, in fact

For all neighbours $w \in N(u)$

Update $w.d$ to

$u.d + w((u,w))$ if

this is smaller

$\boxed{\text{"RELAX"}}$ operation

Requires a $\boxed{\text{PRIORITY QUEUE}}$

Elements e with key

Sort on keys in increasing order

(smallest key — first in queue)

Operations

- Q. INSERT(e, keyval) — insert e with key keyval
- Q. DECREASE-KEY (e, newval) — update key of e to newval
- Q. EXTRACT-MIN — dequeue first element
- Q. EMPTY — queue empty or not ?

Subroutine for updating distance estimates

```
RELAX (u, v)
    if (v.d > u.d + w(u,v))
        v.d := u.d + w(u,v)
        Q. DECREASE-KEY (v, v.d)
        v.π := u
```

$O(1)$

1 decrease key

```
DIJKSTRA (G=(V,E), s)
    for all v ∈ V
        v.d := ∞
        v.π := NULL
        Q. INSERT (v, v.d)
    Q. DECREASE-KEY (s, 0)
    while NOT( Q. EMPTY )
        u := Q. EXTRACT-MIN
        for v ∈ N(u)
            RELAX (u,v)
```

|V| times

$O(1)$

$O(|V|)$

|V| insert

1 decrease-key

|V| times

|V| empty checks

|V| extract-min

Total of |E| calls

Want to do:

- correctness proof
- time complexity analysis

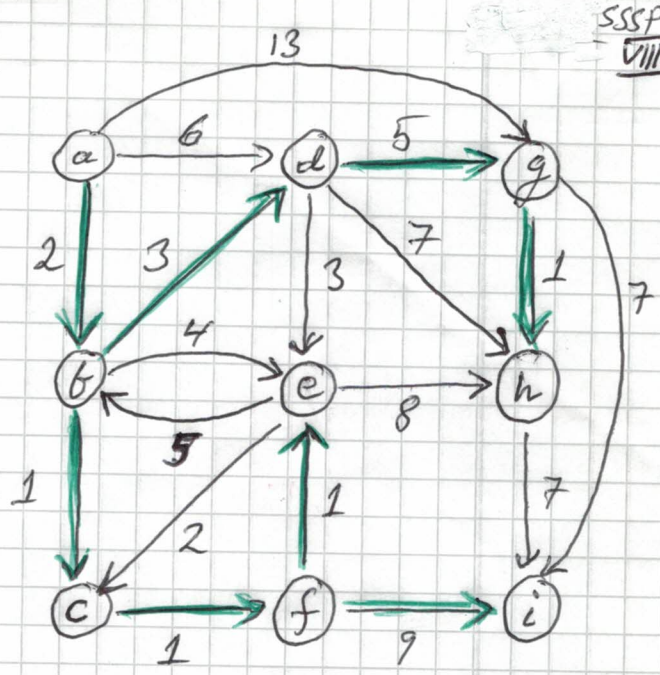
But first:

Dry-run on graph we saw before, starting in vertex a

In the table below, we have distance updates at every step

RED min queue element; taken case of next

BLUE relaxed edges



GREEN shortest paths edges

Vertex	Distance updates									
a	0	0	0	0	0	0	0	0	0	0
b	∞	2	2	2	2	2	2	2	2	2
c	∞	∞	3	3	3	3	3	3	3	3
d	∞	6	5	5	5	5	5	5	5	5
e	∞	∞	6	6	5	5	5	5	5	5
f	∞	∞	∞	4	4	4	4	4	4	4
g	∞	13	13	13	13	10	10	10	10	10
h	∞	∞	∞	∞	∞	12	12	11	11	11
i	∞	∞	∞	∞	13	13	13	13	13	13

LEMMA 2 At all times during Dijkstra's algorithm it holds that $v.d \geq \delta(s,v)$

(SHOULD BE KIND OF OBVIOUS... But needs to be proven.)

<u>Proof</u> By induction over # calls to RELAX

<u>Base case</u> Before first RELAX call we have

$$s.d = 0 = \delta(s,s)$$
$$v.d = \infty \geq \delta(s,v) \quad \text{for } v \neq s$$

<u>Induction step</u> Suppose $v.d \geq \delta(s,v)$ holds for all $v \in V$ after n calls to RELAX.
Suppose $(n+1)$st call is RELAX(u,v)
After this call we have

$$\begin{aligned} v.d &= u.d + w(u,v) && \text{[by the algorithm]} \\ &\geq \delta(s,u) + w(u,v) && \text{[Induction hypothesis]} \\ &\geq \delta(s,v) && \text{[by def. of } \delta(s,v)\text{]} \end{aligned}$$

For all $w \neq v$ $w.d$ didn't change.
Hence for all $v \in V$ $v.d \geq \delta(s,v)$ also after $(n+1)$st call.
The lemma follows by the induction principle.

THEOREM When run on a directed graph G with non-negative weights, DIJKSTRA(G,s) computes shortest paths from s to all $v \in V(G)$

<u>Proof</u> Let S_t be set of extracted vertices after t calls to Q.EXTRACT-MIN. Prove by induction that $\forall v \in S_t$ $d.v = \delta(s,v)$.

Base cases: $\boxed{t=0}$ Then $S=\emptyset$, and claim is vacuously true.

$\boxed{t=1}$ then $S=\{s\}$, and $d.s=\delta(s,s)=0$.

Induction step Want to prove $u.d=\delta(s,u)$ for dequeued element u at time $t+1$.

Induction hypothesis: $\forall v\in S_t \quad v.d=\delta(s,v)$

Argue by contradiction! $u.d\neq\delta(s,u)$.

By Lemma 2 $u.d>\delta(s,u)$

That is, we found a path that was not a shortest path!

Fix a shortest path $P: s \rightsquigarrow u$

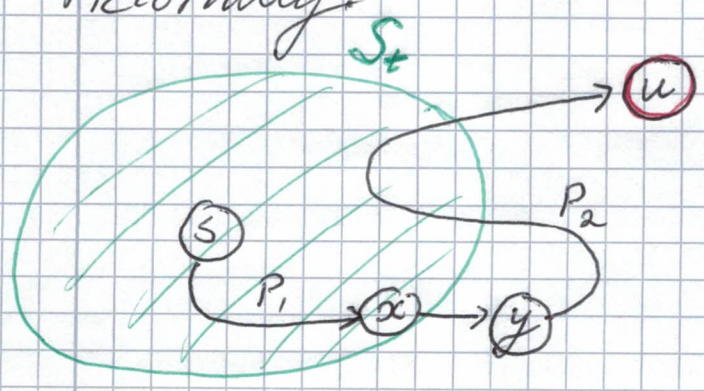
Let $\boxed{y}$ be the first vertex on the path P that is outside of S_t (there is at least one such vertex, since $u\notin S_t$). Let $\boxed{x}$ be the predecessor of y on P — there must exist such a predecessor, since $y\neq s$.

We have $x\in S_t$.

We can decompose the path P as

$$s \overset{P_1}{\rightsquigarrow} x \to y \overset{P_2}{\rightsquigarrow} u$$

Pictorially:



Since $x \in S_t$, by induction hypothesis

$$\boxed{x.d = \delta(s,x)}. \quad (1)$$

When x was dequeued y was relaxed. And any subpath of a shortest path is also a shortest path. Hence

$$y.d \leq \delta(s,x) + w(x,y) = \delta(s,y) \quad (2)$$

and appealing to Lemma 2 we have

$$\boxed{y.d = \delta(s,y)}. \quad (3)$$

Since all edge weights are non-negative, the length/weight of the subpath P_2 is non-negative, so

$$\boxed{\delta(s,y) \leq \delta(s,u)}. \quad (4)$$

Using Lemma 2 again, we have

$$\boxed{\delta(s,u) \leq u.d} \quad (5)$$

Since u was dequeued before y, we must have

$$\boxed{u.d \leq y.d}. \quad (6)$$

Putting together (6), (3), (4), and (5) we get

$$\boxed{u.d} \leq y.d = \delta(s,y) \leq \boxed{\delta(s,u) \leq u.d} \quad (7)$$

or $u.d = \delta(s,u)$, but this contradicts $u.d \neq \delta(s,u)$ which was our starting assumption. So it cannot be that $u.d \neq \delta(s,u)$. Hence, $u.d = \delta(s,u)$ and the induction step in the proof goes through.

i.e., induction hypothesis holds for $S_t \cup \{u\} = S_{t+1}$.
This proves that for all vertices v in G it holds that when v is dequeued, we have

$$v.d = \delta(s, v)$$

and the theorem follows ▨

What about time complexity?
(See also page G XCIV)

	Time complexity
INITIALIZATION	
$O(\lvert V\rvert)$	$O(\lvert V\rvert)$
$\lvert V\rvert$ insert operations	$O(\lvert V\rvert \log \lvert V\rvert)$
DEQUEUEING WHILE LOOP	
$O(\lvert V\rvert)$ +	$O(\lvert V\rvert)$
$\lvert V\rvert$ extract-min operations +	$O(\lvert V\rvert \log \lvert V\rvert)$
$\lvert V\rvert$ empty checks	$O(\lvert V\rvert)$
RELAX CALLS	
$O(\lvert E\rvert)$ +	$O(\lvert E\rvert)$
$\leq \lvert E\rvert$ decrease-key operations	$O(\lvert E\rvert \log \lvert V\rvert)$
	$O(\lvert E\rvert \log \lvert V\rvert)$

How fast is the priority queue?
Implementation with min-heap supports all operations in time $\boxed{O(\log n)}$ for n elements

Assuming that graph is connected (so that $\lvert E\rvert = \Omega(\lvert V\rvert)$) get time complexity $O(\lvert E\rvert \log \lvert V\rvert)$

with min-heap priority queue

By using more efficient data structures (Fibonacci heaps), can get running time down to

$$\boxed{O(|V| \log |V| + |E|)}$$

If edge weights are somewhat small non-negative integers, then can get faster algorithms still.

For undirected graphs with integer weights Mikkel Thorup, head of the Algorithms & Complexity Section and the BARC centre at DIKU, published an $O(|V|+|E|)$-time algorithm in 1999.