Lecture 7 - Greedy Graph Algorithms: Shortest Paths

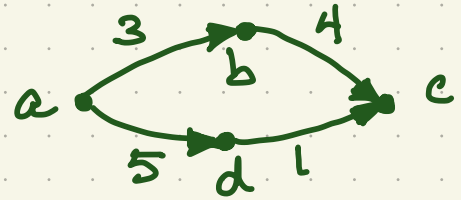
CMPT 307 D-1 Fall 2024

Instructor: David Mitchell

Shortest Paths in Graphs.

- In "unweighted graphs":
  - path length = # edges
  - $d(s,t)$ = min. length of any s-t path
  - BFS(s) finds $d(s,v)$ from a "source" vertex s to every other vertex v in $\Theta(n+m)$ time.
  - BFS is a simple greedy alg.

- Edge weighted graphs:



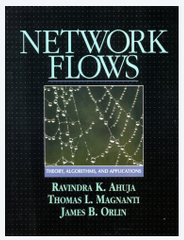
$w(a,b) = 3$
$w(b,c) = 4$

  - path length = Σ of edge weights: $l(a,b) = 3$, $l(a,b,c) = 7$
  - $d(s,t)$ = min. s-t path length: $d(a,c) = 6$.
  - Q: Is there a greedy algorithm for shortest paths in weighted graphs?

(With positive edge weights).

Shortest path applications

- PERT/CPM.
- Map routing.
- Seam carving.
- Robot navigation.
- Texture mapping.
- Typesetting in LaTeX.
- Urban traffic planning.
- Telemarketer operator scheduling.
- Routing of telecommunications messages.
- Network routing protocols (OSPF, BGP, RIP).
- Optimal truck routing through given traffic congestion pattern.



Network Flows: Theory, Algorithms, and Applications, by Ahuja, Magnanti, and Orlin, Prentice Hall, 1993.

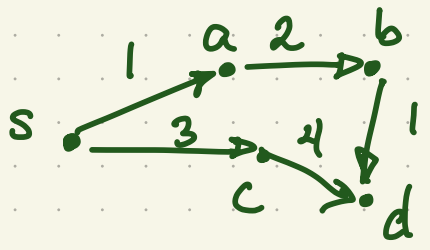
Single Source Shortest Paths Problem

Given: . digraph $G=(V,E)$ with non-negative edge weights $w: E \to \mathbb{R}^{\geq 0}$
. start node $s \in V$

Output: . for every $v \in V$: . $d(s,v)$
. the shortest $s \to v$ path in G.

We can efficiently represent all $n=|V|$ shortest paths in size $O(n)$ with an array $P: V \to V$ s.t. $P[v]$ is the predecessor of v on the shortest $s \to v$ path.

Eg.



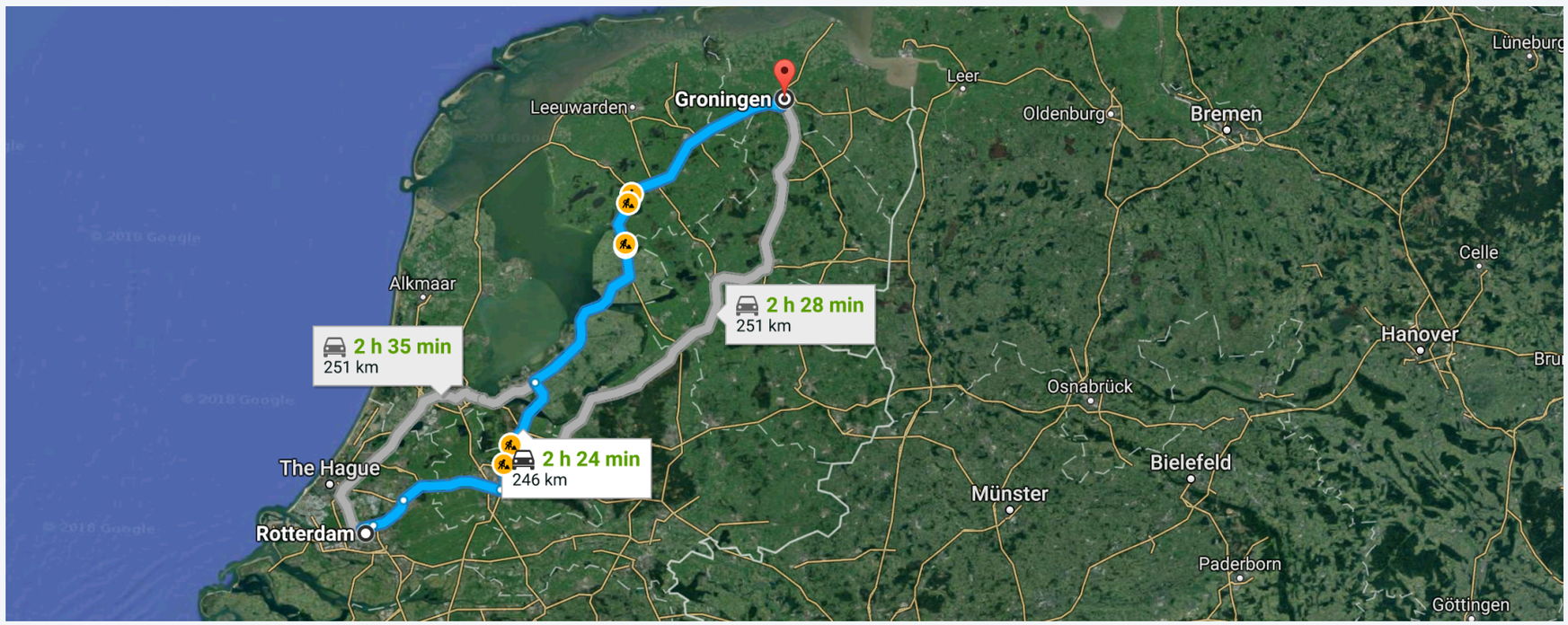
$d(s,a) = 1$
$d(s,b) = 3$
$d(s,c) = 3$
$d(s,d) = 4$

	s	a	b	c	d
$P =$	$\perp$	s	a	s	b

If P is a s-u path and the shortest $s \to v$ path is P, v then the shortest s-u path is P.

Edsger Dijkstra

> *“ What’s the shortest way to travel from Rotterdam to Groningen? It is the algorithm for the shortest path, which I designed in about 20 minutes. One morning I was shopping in Amsterdam with my young fiancée, and tired, we sat down on the café terrace to drink a cup of coffee and I was just thinking about whether I could do this, and I then designed the algorithm for the shortest path.”* — Edsger Dijsktra



Priority Queue ADT

- Stores a set of objects/values with priorities.
- Priorities: values from some ordered set

  Typically: Priorities $\{0, 1, 2, \ldots\}$

  0 is "highest priority".

- Operations on priority queue Q:
  - insert(x, y) // add object x with priority y
  - removeMin() // remove object x with min. priority
  - updatePriority(x, y) // change priority of x to y

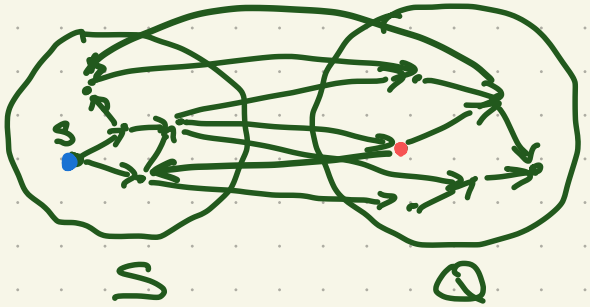
  Time $O(\log n)$ for $|Q| = n$ if implemented with a heap.

  - insert($\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$) // insert n elements.

    Time $O(n)$ if Q is empty.

Dijkstra's Algorithm for the Single Source Shortest Paths Problem

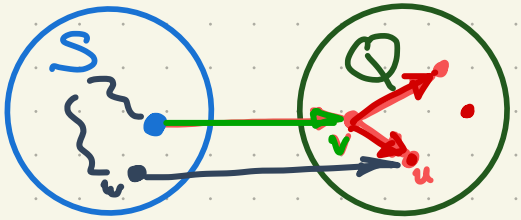
- **Maintain**
  1) Approximation $D(v)$ of $d(s,v)$ for each $v \in V$; $d(s,v) \leq D(v)$
  2) Partition $S \cup Q = V$ of the vertices,
  3) Q is maintained as a priority queue with priorities $D(v)$



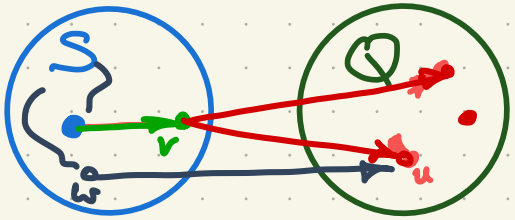
- **Such that**
  4) for each $v \in S$, $D(v) = d(s,v)$ ✳
  5) for each $v \in Q$, $D(v)$ = length of shortest $s \to v$ path using only vertices in S. // "Special paths"

- **Greedy step:**
  - move $v \in Q$ with min. $D(v)$ from Q to S
  - update $D(u)$ for neighbours of v in Q.
    - ↳ $D(u)$ is an approximation of $d(s,u)$ and the priority of u in Q.

Dijkstra's Algorithm



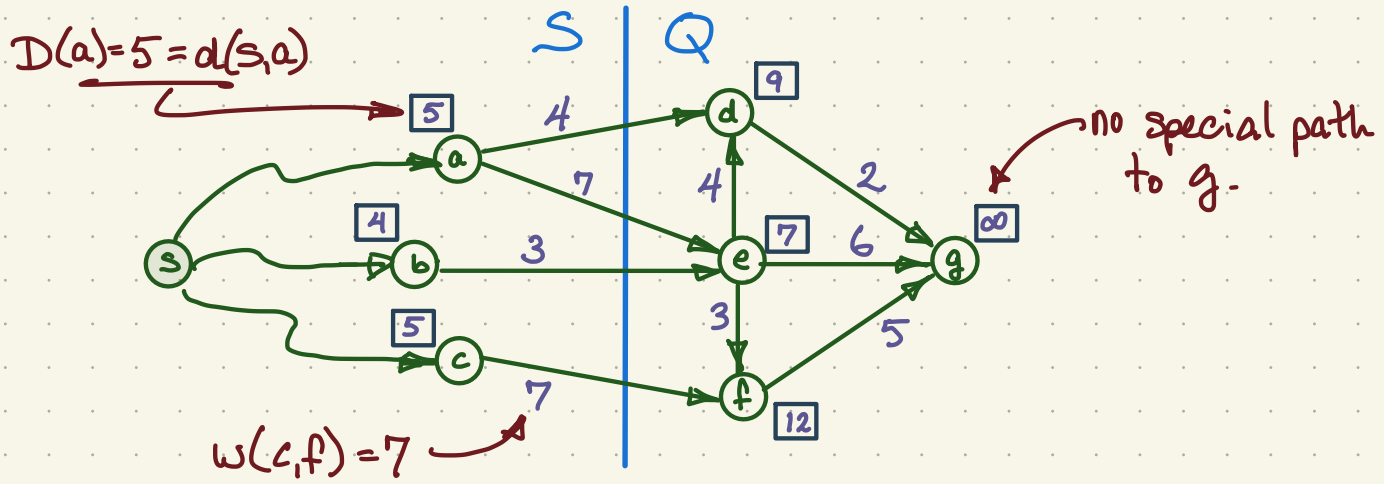
Input: $G=(V,E)$, $w: E \to \mathbb{R}^{\geq 0}$, $s \in V$.
Output: $D: V \to \mathbb{R}^{\geq 0}$ s.t. $D(v) = d(s,v)$, $P: V \to V$



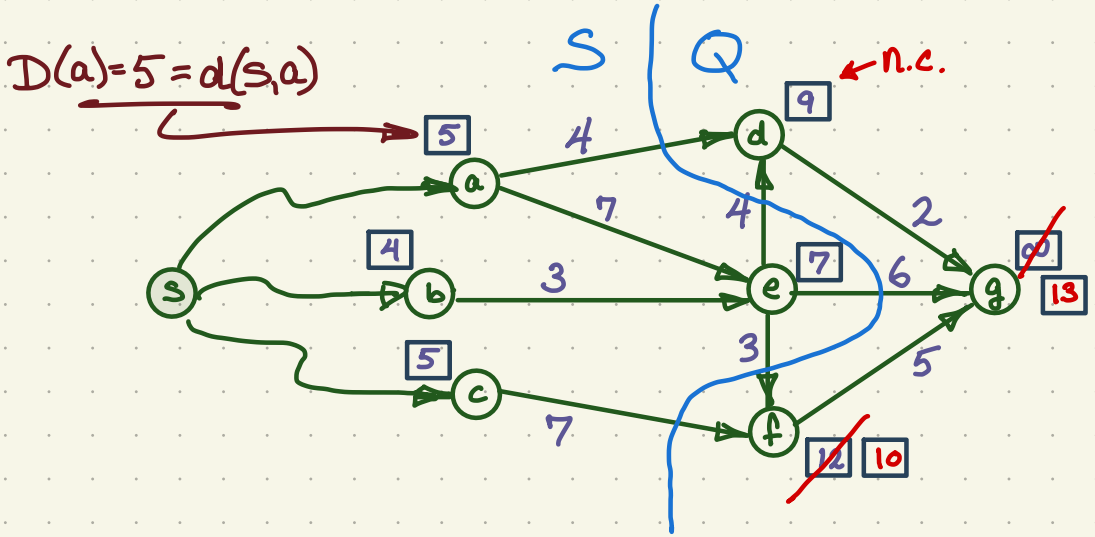

```
D(s) = 0,
D(v) = +∞ for v ≠ s
Q = { (v, D(v)) : v ∈ V }   // S = V - Q is implicit.
while Q ≠ ∅ {
    v ← Q.deleteMin   // Move v to S
    for each edge (v,u) with u ∈ Q {
        if ( D(v) + w(v,u) < D(u) ){
            D(u) = D(v) + w(v,u)
            Q.updatePriority(u, D(u))
            P[u] = v
        }
    }
}
Output D, P
```

Update D and P for each $u \in Q \cap N(v)$

Dijkstra's Algorithm – Greedy Step



- Vertex in Q with min D is e ⇒ move e from Q to S:



Dijkstra's Algorithm – Proof of *

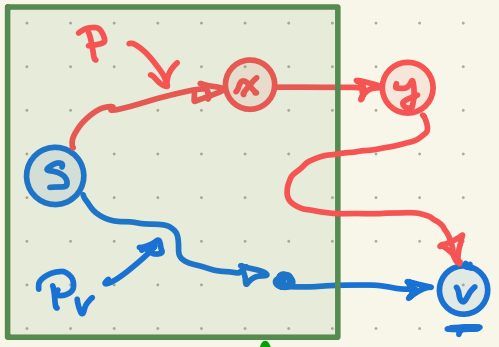
Lemma: At all times during execution of Dijkstra's Alg: $v \in S \Rightarrow D(v) = d(s,v)$. (*)

Pf: (Induction on $|S|$)

Basis: $|S|=1 \Rightarrow S=\{s\}$. $D(s)=0=d(s,s)$.

IH: Assume (*) holds when $|S|=k$, for some $k \geq 1$.

IS: Show (*) holds after adding $k+1^{st}$ vertex v to S.



$D(v)$ is the length of a "special" $s \to v$ path; Call it P_v. (*)

Suppose $D(v) > d(s,v)$. // P_v is not a shortest $s \to v$ path.

Then there is another $s \to v$ path P with $l(P) < l(P_v)$.

Let (x,y) be the first edge of P that leaves S.

Let P_x be the prefix of P up to x, and sim. for P_y.

Then: $l(P_y) \geq l(P_x) + w(x,y)$

$\geq d(s,x) + w(x,y)$ // P_x is a path $s \to x$

$= D(x) + w(x,y)$ // I.H.

$\geq D(y)$ // When moving x to S, $D(y)$ updated to $D(x)+w(x,y)$ or less.

$\geq D(v)$ // Because v has min. $D(\cdot)$ by construction.

Thus $l(P) \geq l(P_y) \geq D(v) \underset{(*)}{=} l(P_v)$ – a contradiction. ☒

Dijkstra's Algorithm – Correctness & Complexity

- **Thm:** At termination, $D(v) = d(s,v)$ for every $v \in V$.

  Pf: Follows from Lemma.

- Dijkstra's Alg. runs in time $O(m \log n)$, assuming G connected.

```
D(s) = 0,                                   } O(1)
D(v) = +∞ for v ≠ s                         }
Q = { (v, D(v)) : v ∈ V }                   } O(n)
while Q ≠ ∅ {
    v ← Q.deleteMin

    for each edge (v,u) with u ∈ Q {
        if ( D(v) + w(v,u) < D(u) ){
            D(u) = D(v) + w(v,u)
            Q.updatePriority(u, D(u))
        }
    }
}
```

$O(n \log n + m \log n)$

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

If G is connected, $m \geq n-1$, so this is $O(m \log n)$

```
for each v ∈ V {
    do Y.
    for each edge (v,u) {
        do X
    }
}
```

$O(n)$

$+ O(m \cdot T(x))$.

$= O(n + m \cdot T(x))$.

End