CS 341: Algorithms

Lecture 14: Single-source shortest paths

Éric Schost

based on lecture notes by many other CS341 instructors

David R. Cheriton School of Computer Science, University of Waterloo

Fall 2024

Conventions

Input:

- a **directed** graph $G = (V, E)$
- with **weights** $w(e)$ on the edges
  $w(\gamma)$ = weight of a path γ = sum of the weights of its edges
- optional: no **isolated vertices**, with no incoming or outgoing edge $m \geq n/2$

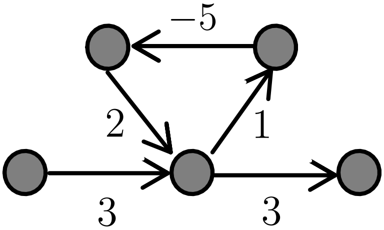
Output:

- today: the shortest (=minimal weight) paths between a **source** s and **all vertices**
- next: shortest paths between **all vertices**

Remark: nothing faster known (to me) for single-source, single-destination

Remarks

1. shortest walks may not exist if there are **negative length cycles**



- some algorithms can deal with negative edges or detect negative cycles
- Dijkstra's algorithm needs positive weights
- if negative cycles possible, **shortest path** (=simple walk) NP-complete
- if no negative cycle, **shotest walk=shortest path**

Remarks

2. if there exists a shortest path $s \rightsquigarrow t$, write $\boldsymbol{\delta(s,t)}$ for its weight

- called the **distance** from s to t (but we may not have $\delta(s,t) = \delta(t,s)$)
- if there is **no path** $s \rightsquigarrow t$, $\delta(s,t) = \infty$

3. easy special case: G is a DAG

- topological sort the vertices $v_1 < \cdots < v_n$
- DP algorithm to compute all distances $\boldsymbol{\delta(s,v_1),\ldots,\delta(s,v_i)}$
- linear runtime

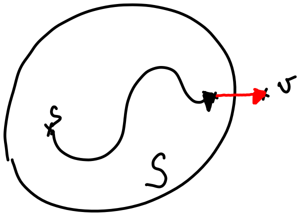
Dijkstra's algorithm

Assumption

All weights are non-negative

Idea of the algorithm:

- starting from s, grow a tree (S, T), together with the **distances** $\delta(s, v)$ for v in S
- at every step, add to S the remaining vertex v **closest to s**
- **no negative weight:** this vertex is on an edge (u, v), u in S, v in $V - S$
- if there is no such edge, we're done (all remaining vertices are unreachable)



both a greedy algorithm and a generalization of BFS

Key property

Claim

Let (S, T) be a tree rooted at s and take an edge (u, v) such that

- u is in S, v is in $V - S$
- $\delta(s, u) + w(u, v)$ **minimal** among these edges

Then $\boldsymbol{\delta(s, u) + w(u, v) = \delta(s, v)}$
(and it is the minimum of **all** $\delta(s, v)$ for v not in S, but we don't need this)

Key property

Claim

Let (S, T) be a tree rooted at s and take an edge (u, v) such that

- u is in S, v is in $V - S$
- $\delta(s, u) + w(u, v)$ **minimal** among these edges

Then $\boldsymbol{\delta(s, u) + w(u, v) = \delta(s, v)}$
(and it is the minimum of **all** $\delta(s, v)$ for v not in S, but we don't need this)

Proof:

- take a path $\gamma : s \rightsquigarrow v$ and let (x, y) be its first edge $\boldsymbol{S \to V - S}$
- $w(\gamma) = w(\boldsymbol{s \rightsquigarrow x}) + w(x, y) + w(\boldsymbol{y \rightsquigarrow v}) \geq \delta(s, x) + w(x, y) \mathbf{+0}$
- so $\boldsymbol{w(\gamma) \geq \delta(s, u) + w(u, v)}$ — choice of u, v
- but also $\boldsymbol{\delta(s, u) + w(u, v) \geq \delta(s, v)}$ — def of distance $s \to v$
- take **shortest** $\boldsymbol{\gamma}$: $w(\gamma) = \delta(s, v)$ so $\boldsymbol{\delta(s, v) \geq \delta(s, u) + w(u, v) \geq \delta(s, v)}$

High-level view of the algorithm

Dijkstra(G, s)
1. $S \leftarrow \{s\}$
2. **while** $S \neq V$ **do**
3. choose (u, v) with u in S, v not in S and $\delta(s, u) + w(u, v)$ minimal (the min value gives $\delta(s, v)$)
4. add v to S
5. **if** not such (u, v), **stop**

Correctness:

- we find $\delta(s, v)$ for all v in S
- if $S = V$ at the end, OK
- if not, when we stop, the remaining vertices are unreachable

Data structure:

- how to find (u, v) efficiently?
- use a priority queue of vertices

The min-priority queue

Building P

- contains all vertices in $\boldsymbol{V - S}$ (initially, all V)
- for $v \neq s$, we will maintain $\mathsf{priority}[v] = \mathbf{min}_{u \in S, (u,v) \in E} (\boldsymbol{\delta(s,u) + w(u,v)})$ (with $\min(\emptyset) = \infty$)
- also store the vertex u that gives the min, if applicable
- need to be able to update priorities

Initialization:

- $\mathsf{priority}[s] = 0$
- $\mathsf{priority}[v] = \infty$ for $v \neq s$

The min-priority queue

Updating P

- if v is the vertex with **minimal priority**, then

$$\begin{aligned} \mathsf{priority}[v] &= \min_{v' \in V - S} \mathsf{priority}[v'] \\ &= \min_{v' \in V - S} \min_{u \in S, (u,v') \in E} (\delta(s,u) + w(u,v')) \\ &= \delta(s,v) \qquad \text{(key property)} \end{aligned}$$

(once we get it out the min-queue, we store it in an array $d[v]$)

The min-priority queue

Updating P

- if v is the vertex with **minimal priority**, then

$$\begin{aligned}\mathsf{priority}[v] &= \min_{\boldsymbol{v'\in V-S}} \mathsf{priority}[v'] \\ &= \min_{\boldsymbol{v'\in V-S}} \min_{\boldsymbol{u\in S,(u,v')\in E}} (\delta(s,u)+w(u,v')) \\ &= \delta(s,v) \qquad \text{(key property)}\end{aligned}$$

  (once we get it out the min-queue, we store it in an array $d[v]$)

- then for all v' remaining in P, we must set

$$\mathsf{priority}[v'] = \boldsymbol{\min_{u\in S+v,(u,v')\in E} (\delta(s,u)+w(u,v'))}$$

  - if there is no edge (v,v'), $\mathsf{priority}[v']$ unchanged
  - else, the new priority is $\min(\mathsf{priority}[v'], d[v]+w(v,v'))$

Pseudo-code

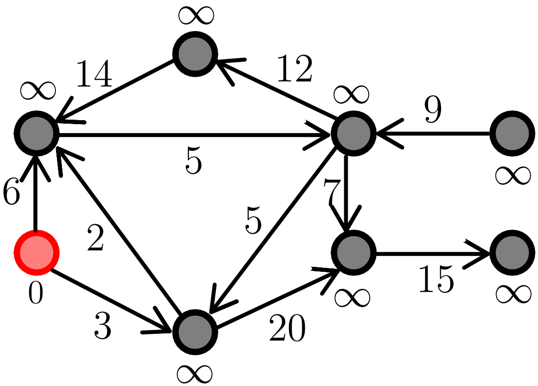
Dijkstra(G, s)

$P \leftarrow$ **heapify**$([s, 0, s], [v, \infty, \bullet]_{v \neq s})$
while P not empty **do**
    $[v, \ell, u] \leftarrow$ **remove_min**(P)
    $d[v] \leftarrow \ell$
    parent$[v] \leftarrow u$
    for all edges (v, v') **do**
        if $d[v] + w(v, v') <$ priority$[v']$ **then**
            replace $[v', \_, \_]$ by $[v', d[v] + w(v, v'), v]$ in P

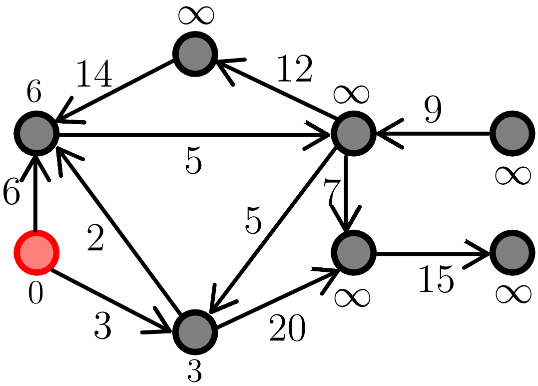
Missing details:

- implement P as a heap
- use an array index to know where v' is in P
- change priorities in P
- update index as needed

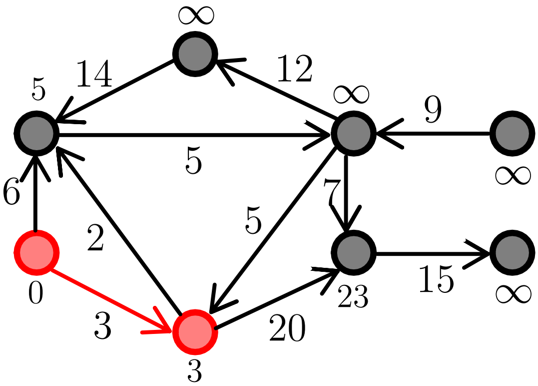
Example



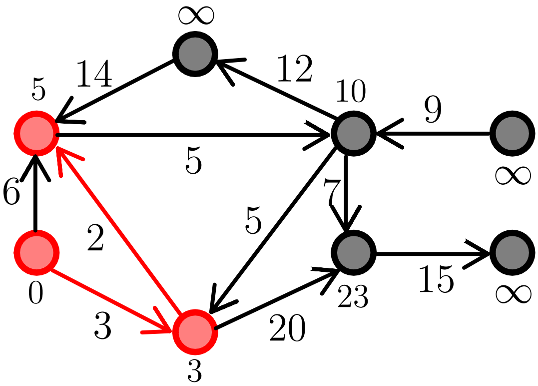
Example



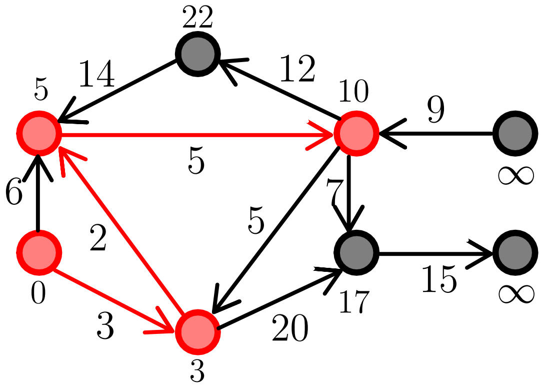
Example



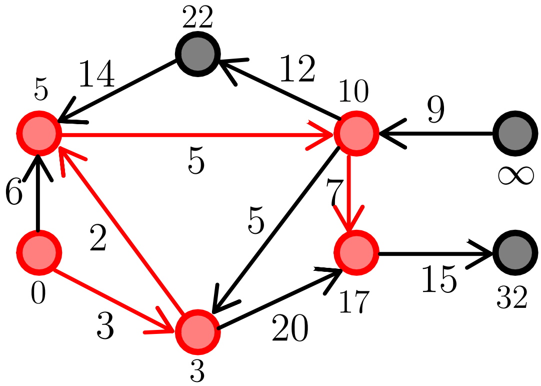
Example



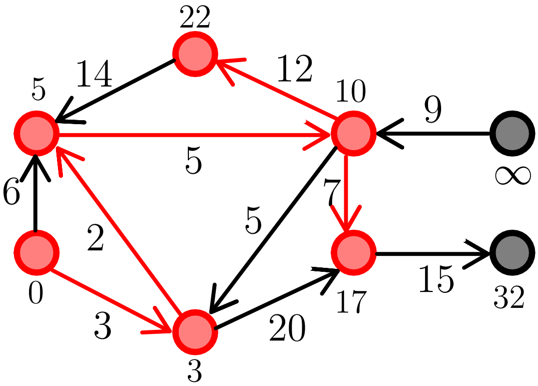
Example



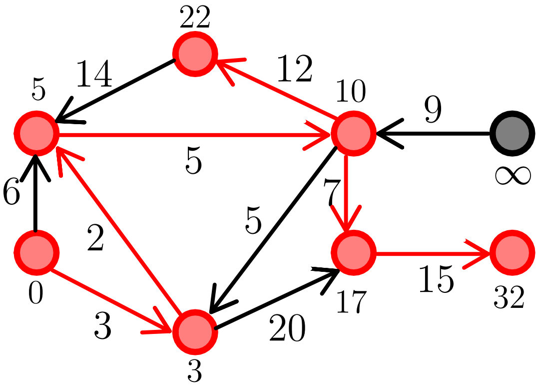
Example



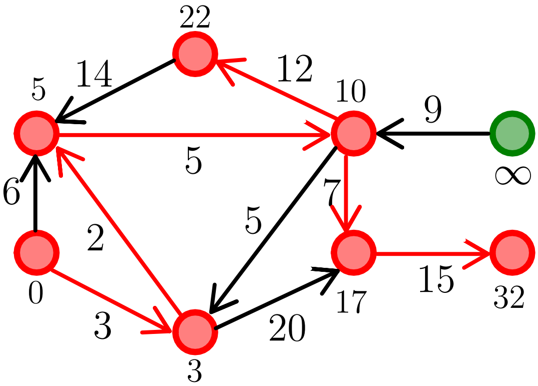
Example



Example



Example



Runtime

Priority queue

- binary heap implementation: $\boldsymbol{O(\log(n))}$ for remove-min and change priority

Total

- n remove min, m change priority, so total $\boldsymbol{O((m+n)\log(n))}$
- if no isolated vertex, $\boldsymbol{n/2 \leq m}$, so total $\boldsymbol{O(m\log(n))}$

Remark

- **Fibonacci heaps:**
  - $O(1)$ insert
  - $O(\log(n))$ **amortized** remove min
  - $O(1)$ **amortized** decrease priority
- total becomes $\boldsymbol{O(m + n\log(n))}$