An unusual way of solving the SSSP Problem

Ferienakademie im Sarntal — Course 2
Distance Problems: Theory and Praxis

Corelius Diekmann

Fakultät für Informatik
TU München

26. September 2010

Friedrich-Alexander-Universität Erlangen-Nürnberg

Technische Universität München

Universität Stuttgart

Outline

1. Preliminaries
   - The SSSP – Single Source Shortest Path Problem
   - Terminology
   - First conclusions
   - Terminology
2. Towards an alternative visiting algorithm
   - Idea
   - Component Hierarchy
   - Visiting a vertex
3. The first algorithm
   - SSSP-Algorithm
   - Visit-Algorithm
4. Conclusion

The SSSP – Single Source Shortest Path Problem

- many solutions exist
- all are based on Dijkstra
- no Algorithm has achieved linear time so far (1997)

The SSSP – Single Source Shortest Path Problem

- many solutions exist
- all are based on Dijkstra
- no Algorithm has achieved linear time so far (1997)
- Idea:
  - deviate from Dijkstra's visiting approach
  - construct a theoretical linear time Algorithm

Terminology

- $G = (V, E)$: undirected connected graph with positive integer edge weights
- $l : E \to \mathbb{N}$: weight function for G
  - l only works on Integers
  - $sizeof(Integer)$: length of an Integer in Bit

Terminology

- $G = (V, E)$: undirected connected graph with positive integer edge weights
- $l : E \to \mathbb{N}$: weight function for G
  - l only works on Integers
  - *sizeof*(*Integer*): length of an Integer in Bit
    - 32 for i386
    - 64 for most modern 64 bit systems
    - probably 128 for special IPv6 routers

Terminology

- $G = (V, E)$: undirected connected graph with positive integer edge weights
- $l : E \to \mathbb{N}$: weight function for G
  - l only works on Integers
  - $sizeof(Integer)$: length of an Integer in Bit
    - 32 for i386
    - 64 for most modern 64 bit systems
    - probably 128 for special IPv6 routers
  - $\Rightarrow 2^{sizeof(Integer)} - 1$ maximum value of l
  - if $(v, w) \notin E$, then $l(v, w) = \infty$.

Terminology

- $G = (V, E)$: undirected connected graph with positive integer edge weights
- $l : E \to \mathbb{N}$: weight function for G
  - l only works on Integers
  - $sizeof(Integer)$: length of an Integer in Bit
    - 32 for i386
    - 64 for most modern 64 bit systems
    - probably 128 for special IPv6 routers
  - $\Rightarrow 2^{sizeof(Integer)} - 1$ maximum value of l
  - if $(v, w) \notin E$, then $l(v, w) = \infty$.
- $s \in V$: source for the SSSP-problem

Terminology

- $G = (V, E)$: undirected connected graph with positive integer edge weights
- $l : E \to \mathbb{N}$: weight function for G
  - l only works on Integers
  - $sizeof(Integer)$: length of an Integer in Bit
    - 32 for i386
    - 64 for most modern 64 bit systems
    - probably 128 for special IPv6 routers
  - $\Rightarrow 2^{sizeof(Integer)} - 1$ maximum value of l
  - if $(v, w) \notin E$, then $l(v, w) = \infty$.
- $s \in V$: source for the SSSP-problem
- $d : V \to \mathbb{N}$: $d(v)$ is the shortest distance from s to v

Terminology

- $G = (V, E)$: undirected connected graph with positive integer edge weights
- $l : E \to \mathbb{N}$: weight function for G
  - l only works on Integers
  - $sizeof(Integer)$: length of an Integer in Bit
    - 32 for i386
    - 64 for most modern 64 bit systems
    - probably 128 for special IPv6 routers
  - $\Rightarrow 2^{sizeof(Integer)} - 1$ maximum value of l
  - if $(v, w) \notin E$, then $l(v, w) = \infty$.
- $s \in V$: source for the SSSP-problem
- $d : V \to \mathbb{N}$: $d(v)$ is the shortest distance from s to v
- $D : V \to \mathbb{N}$, super distance for d
- $D(v)$: shortest distance from s to v currently known in the ongoing step
- $\Rightarrow D(v)$: upper limit for the absolute shortest distance $d(v)$

Terminology cont.

- $S \subseteq V$: set of vertices already visited

initalize(G, l, s):

Require: $G = (V, E)$ is an undirected graph with positive integer edge weights defined by l.

Require: $s \in V$.

$S := \{s\}$

$D(s) := 0$

$d(s) := 0$

for all $v \in V \setminus S$ **do**

$D(v) := l(s, v)$

end for

Visiting a vertex

- $v \in V \setminus S$
- setting $D(w)$ for all $(v, w) \in E, w \notin S$ to $\min\{D(w), D(v) + l(v, w)\}$
- moving v to S
- v can only be visited if $D(v) = d(v)$
  cf. Dijkstra

visit(v):

Require: $D(v) = d(v) \wedge v \in V \setminus S$
  for all $(v, w) \in E, w \notin S$ **do**
    $D(w) := \min\{D(w), D(v) + l(v, w)\}$
    $S := S \cup \{v\}$
  end for

First conclusions

- SSSP-Algorithm terminates with $S = V$
- returns: minimum distance $d(v)$ for all $v \in V$.
- in every step: $\forall v \in S : D(v) = d(v)$.

Lemma

If $v \in V \setminus S$ minimizes $D(v)$, then $D(v) = d(v)$.

Proof.

$u \in V \setminus S$: a vertex on a shortest path from s to v which can be visited.

$\Rightarrow D(u) = d(u)$

$$\Rightarrow D(v) \overset{\text{super distance}}{\geq} d(v) \overset{\text{on a shortest path}}{\geq} d(u) = D(u) \overset{\text{minimum}}{\geq} D(v). \qquad \square$$

First conclusions cont.

Lemma

$\min D(V \setminus S) = \min d(V \setminus S)$ *is nondecreasing.*

Proof.

cf. Dijkstra's Algorithm. □

First conclusions cont.

Lemma

$\min D(V \setminus S) = \min d(V \setminus S)$ *is nondecreasing.*

Proof.

cf. Dijkstra's Algorithm. □

By combining all previous lemmata, Dijkstra's Algorithm can be derived. However, our aim is not to use Dijkstra's Algorithm, but the previous lemmata will prove to be useful, when handled with care.

Terminology: shift operator

$$x \gg i = \lfloor \frac{x}{2^i} \rfloor$$

- $x \gg i$: shifting the binary representation of x i digits to the right
- corresponds to the fast and cheap x86 assembly operation *shr*

Terminology: shift operator

$$x \gg i = \lfloor \frac{x}{2^i} \rfloor$$

- $x \gg i$: shifting the binary representation of x i digits to the right
- corresponds to the fast and cheap x86 assembly operation *shr*

Lemma

$x \leq y \Rightarrow x \gg i \leq y \gg i$

Terminology: shift operator

$$x \gg i = \lfloor \frac{x}{2^i} \rfloor$$

- $x \gg i$: shifting the binary representation of x i digits to the right
- corresponds to the fast and cheap x86 assembly operation *shr*

Lemma

$x \leq y \Rightarrow x \gg i \leq y \gg i$

Lemma

$x \gg i < y \gg i \Rightarrow x < y$

Towards an alternative visiting algorithm

First Idea

- divide V into disjoint subsets $V_1, \ldots, V_k$
- let edges between the subsets have length at least δ
- find a way to visit vertices with $d(v) = D(v)$
- avoid any kind of sorting

Towards an alternative visiting algorithm

Lemma

Suppose the vertex set V divides into disjoint subsets $V_1, \ldots, V_k$ and that all edges between the subsets have length at least δ. Further suppose for some i, $v \in V_i \setminus S$ that $D(v) = \min D(V_i \setminus S) \leq \min D(V \setminus S) + \delta$. Then $d(v) = D(v)$.

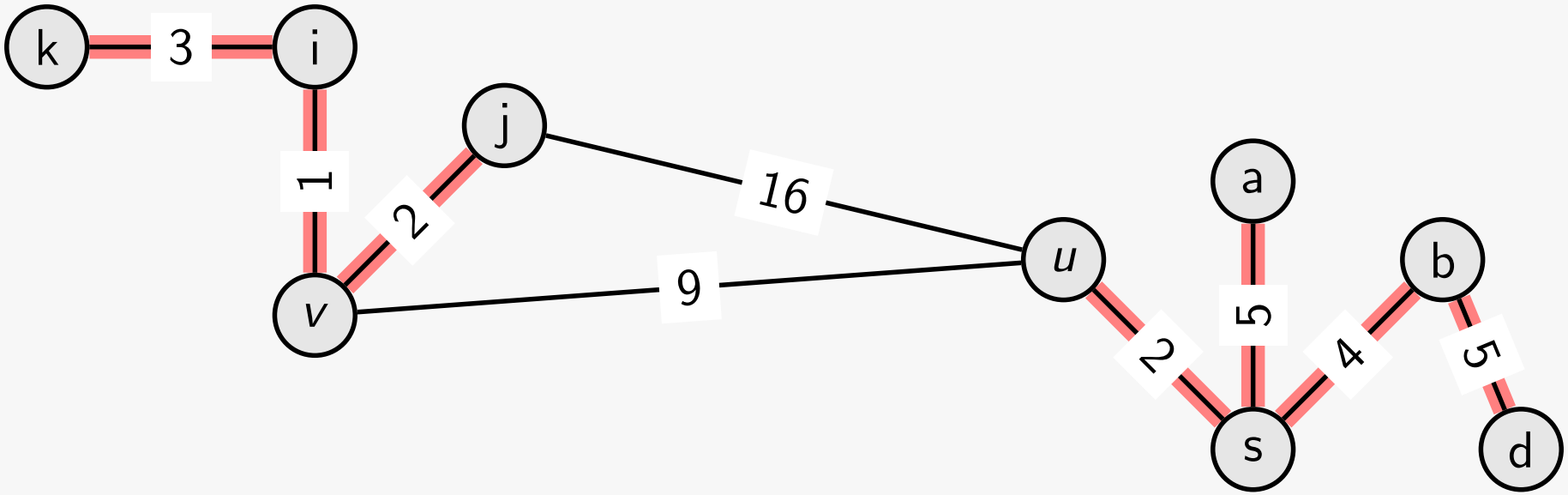

Figure: Example: $\delta = 8$, red highlighted edges describe the two disjoint subsets.

Proof Part 1: $u \in V_i \setminus S$

Proof.

To imagine the previous Lemma, suppose u to be the first vertex outside S on a shortest path from s to v. If $u \in V_i \setminus S$ then $D(v) \geq d(v) \geq d(u)$. The way we visit each vertex and the fact that u is one of the vertices directly outside S, concludes $d(u) = D(u)$. However, as $D(v)$ is the minimum in $V_i \setminus S$, $D(u) \geq D(v)$. Putting the inequation together $d(v) = D(v)$. □

Proof Part 2: $u \notin V_i \setminus S$

Proof.

If $u \notin V_i$, every edge between the component of u and V_i has at least a weight of δ. Again, $D(v) \geq d(v)$. A component border has to be passed to (possibly indirectly) connect u and v, thus $d(v) \geq d(u) + \delta = D(u) + \delta$. However, the minimal vertex concerning D in the whole graph might still be less than u, in combination with the assumption,

$D(u) + \delta \geq \min D(V \setminus S) + \delta \overset{\text{assumption}}{\geq} \min(V_i \setminus S) \overset{\text{assumption}}{=} D(v)$. Thus $d(v) = D(v)$. □

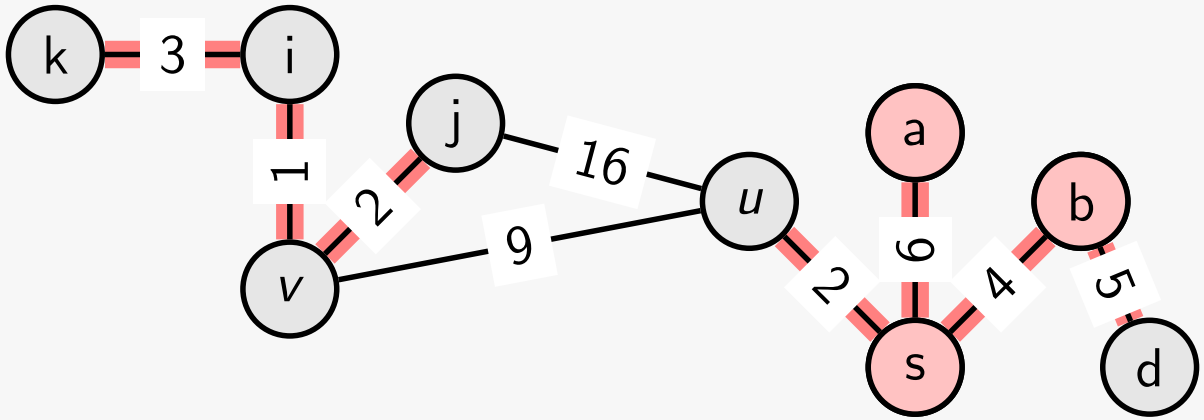

Figure: Example: $\delta = 8$, $S = \{s, a, b\}$, red highlighted edges describe the two components.

Breaking up the Graph

The previous Lemma gives us a basic idea of how a Graph can be broken up:

Breaking up the Graph

The previous Lemma gives us a basic idea of how a Graph can be broken up:

Component Hierarchy

- Dividing the graph into components, forming a component hierarchy.
- $G_i = (V_i, E_i)$: subgraph of G with $\forall e \in E_i : l(e) < 2^i$.
- We consider the connected components of G_i
- $[v]_i$: component on level i containing v
- Component Hierarchy can be imagined like a tree: all $[w]_{i-1}$ with $[w]_i = [v]_i$ are called children of $[v]_i$.

Component Hierarchy

- G_0 only consists of singleton vertices.
- $G_{sizeof(Integer)}$ represents the whole graph.
- If $[v]_i \neq [w]_i \Rightarrow dist(v, w) \geq 2^i$.

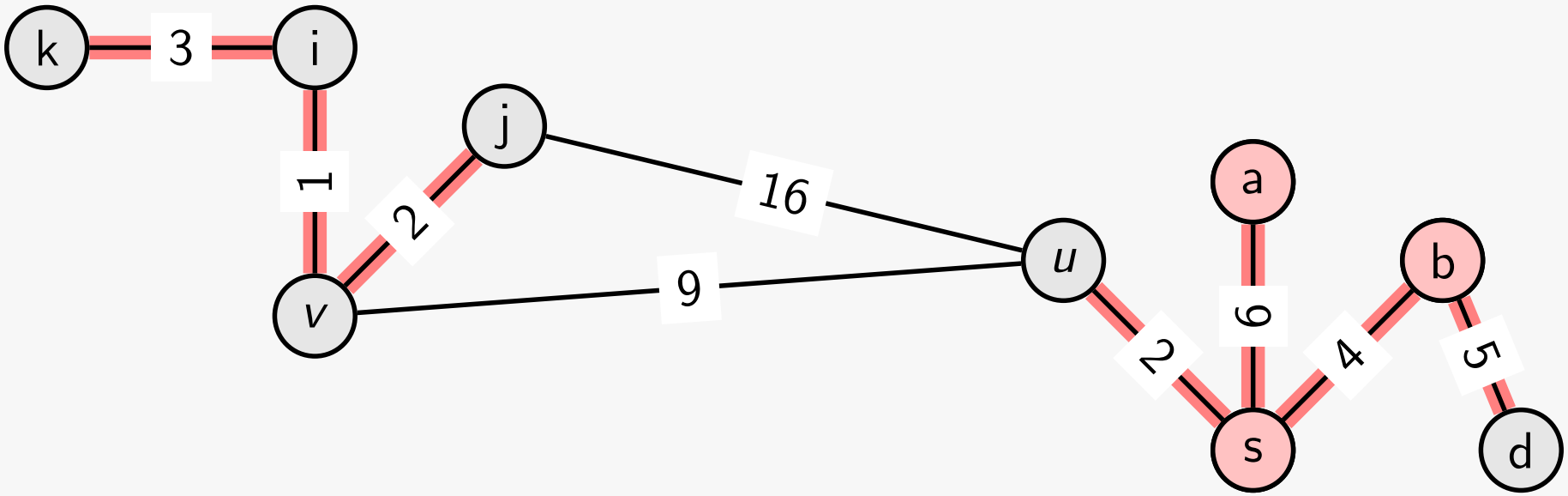

Figure: Example: $\delta = 8$, $S = \{s, a, b\}$, red highlighted edges describe the two components.

The red highlighted edges describe the components of G_3: All highlighted edges carry a weight less than 8.
$[k]_3 = [i]_3 = [v]_3 = [j]_3 \neq [u]_3 = [s]_3 = [a]_3 = [b]_3 = [d]_3$

- $[v]_i^-$: abbreviation for $[v]_i \setminus S$.
- $[v]_i^-$: all vertices on level i in the component $[v]_i$ which have not been visited.
- $[v]_i^-$ is not necessarily connected and depends on S!

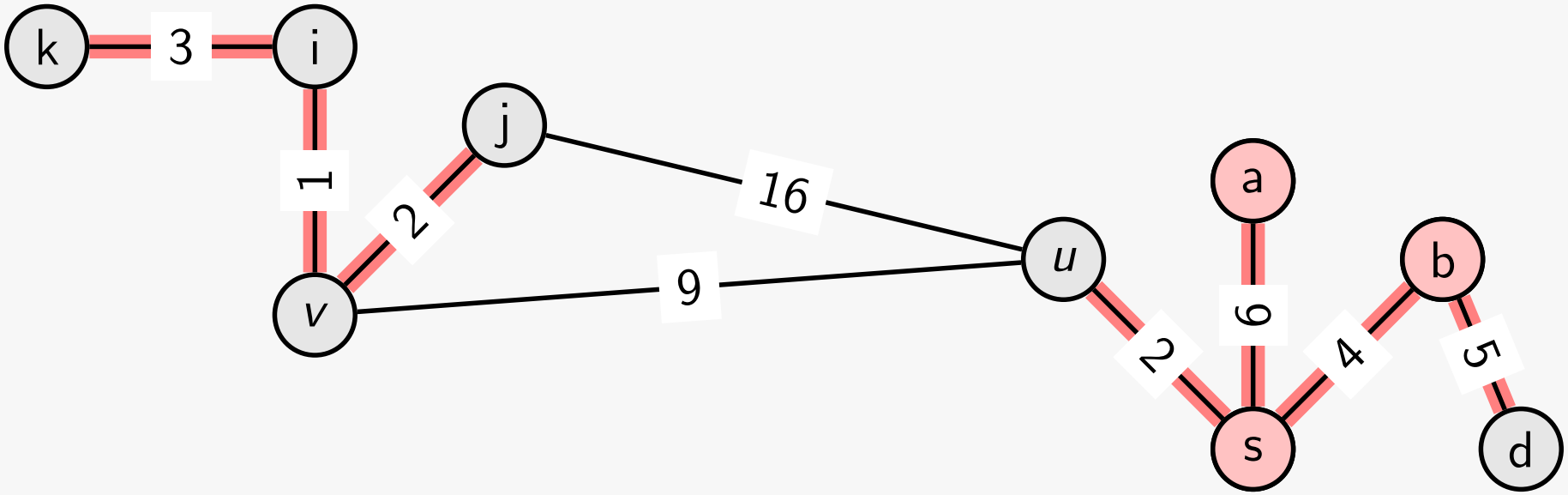

Figure: Example: $\delta = 8$, $S = \{s, a, b\}$, red highlighted edges describe the two components.

- The red highlighted vertices are in S, thus $[u]_3^- = \{u, d\}$ which is not connected.
- $[v]_3^- = \{k, i, v, j\}$

Definition

$[v]_i$ is a min-child of $[v]_{i+1}$ if $\min(D([v]_i^-)) \gg i \;=\; \min(D([v]_{i+1}^-)) \gg i$

Definition

$[v]_i$ is minimal if $[v]_i^- \neq \emptyset$ and $\forall j \geq i : [v]_j$ is a min-child of $[v]_{j+1}$

Definition

$[v]_i$ is a min-child of $[v]_{i+1}$ if $\min(D([v]_i^-)) \gg i \;=\; \min(D([v]_{i+1}^-)) \gg i$

Definition

$[v]_i$ is minimal if $[v]_i^- \neq \emptyset$ and $\forall j \geq i : [v]_j$ is a min-child of $[v]_{j+1}$

Lemma

If $v \notin S, [v]_i$ *is minimal, and* $i < j \leq sizeof(Integer)$,
$\min D([v]_i^-) \gg j-1 \;=\; \min D([v]_j^-) \gg j-1.$

Definition

$[v]_i$ is a min-child of $[v]_{i+1}$ if $\min(D([v]_i^-)) \gg i \;=\; \min(D([v]_{i+1}^-)) \gg i$

Definition

$[v]_i$ is minimal if $[v]_i^- \neq \emptyset$ and $\forall j \geq i : [v]_j$ is a min-child of $[v]_{j+1}$

Lemma

If $v \notin S, [v]_i$ is minimal, and $i < j \leq sizeof(Integer)$,
$\min D([v]_i^-) \gg j-1 \;=\; \min D([v]_j^-) \gg j-1.$

Lemma

Suppose $v \notin S$ and there is a shortest path to v where the first vertex u outside S is in $[v]_i$. Then $d(v) \geq \min D([v]_i^-)$.

Example

- $[s]_3 = \{s, a, w, v\}$
- $[s]_3^- = \{a, w, v\}$
- $[s]_3$ is minimal. $[s]_3 = [v]_3 \Rightarrow [v]_3$ minimal.
- $[v]_2 = \{s, v\}$
- $[v]_2^- = \{v\}$
- $\min D([v]_2^-) = \min D(\{v\}) = D(v)$
- $\min D([v]_3^-) = \min D(\{v, a, w\}) = D(v)$
- $\min D([v]_2^-) \gg 2 = D(v) \gg 2 = 0$ and $\min D([v]_3^-) \gg 2 = D(v) \gg 2 = 0$:
  $[v]_2$ is a min-child of $[v]_3$, thus $[v]_2$ is minimal.
  $[v]_1 = [v]_0$, hence $[v]_0$ is minimal.

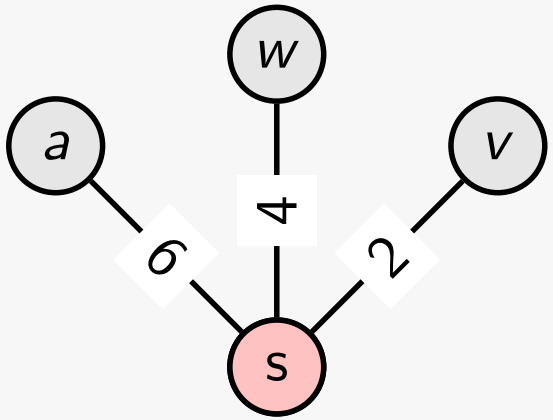

Figure: min-child example: $S = \{s\}$

$[v]_0$ minimal $\Rightarrow D(v) = d(v)$

Lemma

If $v \notin S$ and $[v]_i$ is minimal, $\min D([v]_i^-) = \min d([v]_i^-)$.

Proof.

$\forall w \in V : D(w) \geq d(w) \Rightarrow \min D([v]_i^-) \geq \min d([v]_i^-)$.
Let $x \in [v]_i^-$, when we imagine the graph, we see that
$d(x) \geq \min D([v]_i^-)$[1].
Resulting equation: $d(x) \geq \min D([v]_i^-) \geq \min d([v]_i^-)$ for all x, especially for the x which minimizes $d([v]_i^-)$, which delivers the result
$\min D([v]_i^-) = \min d([v]_i^-)$. □

[1] $d(x) \geq \min D([v]_i^-)$ will be shown on the next slide

$d(x) \geq \min D([v]_i^-)$

Lemma

If $v \notin S$ and $[v]_i$ is minimal, $\min D([v]_i^-) = \min d([v]_i^-)$.

Proof.

Let u be the first vertex outside S on a shortest path to x. On the one hand assume $u \in [v]_i$. $D(u)$ is a lower bound for the length of the path from s to x, thus $d(x) \geq D(u)$ and $D(u) \geq \min D([v]_i^-)$ since $u \in [v]_i^-$. On the other hand assume $u \notin [v]_i$. $[v]_i$ is minimal, thus $\min(D([v]_i^-)) \gg i = \min(D([v]_{i+j}^-)) \gg i$, for all $j \in \mathbb{N}_0$. When starting with $j = 0$ and incrementing j until $u \in [v]_{i+j}^-$, the path from s to x over u must at least contain one edge of weight $\geq 2^i$. Note that $2^i \gg i = 1$. However, $\min(D([v]_{i+j}^-)) \gg i$ remains constant when incrementing j and $u \notin [v]_i$ results in $d(x) \geq \min D([v]_i^-)$. □

Lemma

If $v \notin S$ and v minimizes $D(v)$, $[v]_0$ is minimal.

Proof.

$\forall i : \min D([v]_i^-) \gg i = \min D([v]_{i+1}^-) \gg i = D(v) \gg i$, so $[v]_0$ is minimal. □

SSSP(G, l, s):

initialize(G, l, s)
$w = sizeof(\text{Integer})$
for $i \in [0, w]$ **do**
  declare globally x_i {just define this variable, it will be used by the Visit Algorithm}
end for
Visit($[s]_w$)
return D

SSSP(G, l, s):

- Provides a solution to the SSSP-Problem.
- Not in linear time.
- Recall that $[s]_{sizeof(Integer)}$ represents the whole graph $\Rightarrow$ $[s]_{sizeof(Integer)}$ is minimal by definition.

Visit($[v]_i$):

Require: $[v]_i$ is minimal
  if $i = 0$ **then**
    visit v {This visit refers to the basic visitng Algorithm}
    return
  end if
  if $[v]_i$ has not been visited previously **then**
    $x_i([v]_i) := \min D([v]_i^-) \gg i - 1$
  end if
  repeat
    while $\exists$ child $[w]_{i-1}$ of $[v]_i$ such that $\min D([w]_{i-1}^-) \gg i - 1 = x_i([v]_i)$ **do**
      assert($\min D([v]_i^-) \gg i = \min d([v]_i^-) \gg i$)
      assert($[w]_{i-1}$ is minimal)
      Visit($[w]_{i-1}$)
    end while
    $x_i([v]_i) := x_i([v]_i) + 1$
  until $[v]_i^- = \emptyset \vee x_i([v]_i) \gg 1$ is increased

- Termination condition $i = 0$ must be reasonable!
- The two assertions in the inner while loop describe the intuition of that loop.
  - $\min D([v]_i^-) \gg i = \min d([v]_i^-) \gg i$ is preserved constant throughout the whole while loop.
    c.f. Lemma 13 in [Tho99]
  - while loop only visits all $w \in [v]_i^-$ with $d(w) \gg i = \min D([v]_i^-) \gg i$.
- Termination conditions of the repeat until loop only holds if $[v]_i$ is no longer minimal.
  - If $[v]_i^- = \emptyset$, $[v]_i$ is not minimal by definition.
  - $x_i([v]_i) \gg 1$ is equal to $\min D([v]_i^-) \gg i$. The following lemma explains the repeat until loop condition.

Lemma

If $[v]_i$ *is minimal, it remains minimal until* $\min D([v]_i^-) \gg i$ *is increased, in which case* $\min d([v]_i^-) \gg i$ *is also increased.*

Proof Part 1: $|[v]_i^-|$ was 1

Lemma

If $[v]_i$ *is minimal, it remains minimal until* $\min D([v]_i^-) \gg i$ *is increased, in which case* $\min d([v]_i^-) \gg i$ *is also increased.*

Proof.

Assuming $[v]_i$ is minimal and visiting $w \in [v]_i^-$ stops $[v]_i$ from being minimal. If $[v]_i^-$ was $\{w = v\}$, after visiting w,
$\min D([v]_i^-) = \min d([v]_i^-) = \infty$ and after the visit $[v]_i^- = \emptyset$. □

Proof Part 2: $|[v]_i^-|$ was greater 1

Before the visit to w, $\min d([v]_i^-)$ was equal to $\min D([v]_i^-)$ according to the Lemma on slide 33. Now we pick the smallest possible $j \geq i \in \mathbb{N}$ such that $[v]_j$ is minimal after the visit of w. If $[v]_i \neq \emptyset$, this j must exist, in the worst case it could be $sizeof(Integer)$. Before the visit $\min D([v]_i^-) \gg j \leq \min D([v]_j^-) \gg j$. The way we have chosen j, $\underbrace{\min D([v]_j^-) \gg j}_{before} \leq \underbrace{\min D([v]_j^-) \gg j}_{after}$. However as j was chosen at the minimality border $\underbrace{\min D([v]_j^-) \gg j}_{after} < \underbrace{\min d([v]_{j-1}^-) \gg j}_{after}$. After the visit, it also holds that $\min D([v]_i^-) \gg j \geq \min d([v]_i^-) \gg j \geq \min d([v]_{j-1}^-) \gg j$. Putting it all together, we conclude

Proof cont.

$$\underbrace{\min D([v]_i^-) \gg j}_{after} \geq \underbrace{\min d([v]_i^-) \gg j}_{after} \geq$$

$$\underbrace{\min d([v]_{j-1}^-) \gg j}_{after} > \underbrace{\min D([v]_j^-) \gg j}_{after} \geq \underbrace{\min D([v]_j^-) \gg j}_{before} =$$

$$\underbrace{\min D([v]_i^-) \gg j}_{before} = \underbrace{\min d([v]_i^-) \gg j}_{before}$$

□

Require: $[v]_i$ is minimal
  if $i = 0$ **then**
    visit v {This visit refers to the basic visitng Algorithm}
    return
  end if
  if $[v]_i$ has not been visited previously **then**
    $x_i([v]_i) := \min D([v]_i^-) \gg i - 1$
  end if
  repeat
    while $\exists$ child $[w]_{i-1}$ of $[v]_i$ such that $\min D([w]_{i-1}^-) \gg i - 1 = x_i([v]_i)$ **do**
      assert($\min D([v]_i^-) \gg i = \min d([v]_i^-) \gg i$)
      assert($[w]_{i-1}$ is minimal)
      Visit($[w]_{i-1}$)
    end while
    $x_i([v]_i) := x_i([v]_i) + 1$
  until $[v]_i^- = \emptyset \vee x_i([v]_i) \gg 1$ is increased

Observing the line in which $x_i([v]_i)$ is increased by exactly one, the next Lemma will explain this line and the benefit of the shift operator.

Lemma

Suppose $\min D([v]_i^-) \gg i = \min d([v]_i^-) \gg i$ *and that visiting a vertex* $w \in V \setminus S$ *changes* $\min D([v]_i^-) \gg i$. *Then* $w \in [v]_i$ *and if* $[v]_i^-$ *is not emptied, the change in* $\min D([v]_i^-) \gg i$ *is an increase by one.*

Proof.

When visiting w, $\min D([v]_i^-) \gg i$ is changed and due to the fact that $\min D([v]_i^-) \gg i$ was equal to $\min d([v]_i^-) \gg i$ and d is nondecreasing and a lower bound for D

$$\underbrace{\min D([v]_i^-) \gg i}_{after} > \underbrace{\min D([v]_i^-) \gg i}_{before}$$

However, when recalling the way a vertex is visited, we conclude that the D values can never increase, thus $[v]_i^-$ must have been decreased and hence w must have been in $[v]_i$. □

Proof cont.

Considering $[v]_i^-$ after the visit, according to the assumption, $[v]_i^-$ is not empty, and since $[v]_i$ is connected, there must be an edge (u, x) in $[v]_i$ with $u \notin [v]_i^-$ and $x \in [v]_i^-$. This edge must exist, we could for example choose $u = w$ and x one arbitrary vertex of $[v]_i^-$. Assuming $u = w \Rightarrow [u]_0 = [w]_0$ and with Lemma on slide 35 $[u]_0$ minimal, $d(u) \gg i = D(u) \gg i$ which equals $\min D([v]_i^-) \gg i$ before the visit. If $u \neq w$, $u \notin [v]_i^-$ before the visit to w, we are free to choose u from S. Since S contains the visited vertices, $d(u) \gg i \leq \min D([v]_i^-) \gg i$ just before the visit [2].

[2] c.f. [Tho99] Lemma 10 for an in depth proof of this well conceivable fact. This condition even holds, if we cannot pick u from S

Proof cont.

In any case $d(u) \gg i \leq \min D([v]_i^-) \gg i$ before the visit. Since the edge (u, x) is in $[v]_i$, $l(u, x) < 2^i$ and $D(x) \gg i \leq (d(u) + l(u, x)) \gg i \leq (\min D([v]_i^-) + l(u, x)) \gg i$. Thanks to the shifting, $l(u, x)$ can at least produce an overflow of one, thus $(\min D([v]_i^-) + l(u, x))) \gg i \leq (\min D([v]_i^-) \gg i) + 1$. Putting it together

$$\underbrace{\min D([v]_i^-) \gg i}_{after} \leq \underbrace{D(x) \gg i}_{after} \leq \underbrace{(\min D([v]_i^-) \gg i) + 1}_{before}$$

□

Visit($[v]_i$):

Require: $[v]_i$ is minimal
  if $i = 0$ **then**
    visit v {This visit refers to the basic visitng Algorithm}
    return
  end if
  if $[v]_i$ has not been visited previously **then**
    $x_i([v]_i) := \min D([v]_i^-) \gg i - 1$
  end if
  repeat
    while $\exists$ child $[w]_{i-1}$ of $[v]_i$ such that $\min D([w]_{i-1}^-) \gg i - 1 = x_i([v]_i)$ **do**
      assert($\min D([v]_i^-) \gg i = \min d([v]_i^-) \gg i$)
      assert($[w]_{i-1}$ is minimal)
      Visit($[w]_{i-1}$)
    end while
    $x_i([v]_i) := x_i([v]_i) + 1$
  until $[v]_i^- = \emptyset \vee x_i([v]_i) \gg 1$ is increased

- An unusual algorithm for visiting vertices has been presented.
- With this algorithmic structure and some additional data structures, a linear time and linear space algorithm can be created.

- An unusual algorithm for visiting vertices has been presented.
- With this algorithmic structure and some additional data structures, a linear time and linear space algorithm can be created.
  - Uses (among others) atomic heaps

- An unusual algorithm for visiting vertices has been presented.
- With this algorithmic structure and some additional data structures, a linear time and linear space algorithm can be created.
  - Uses (among others) atomic heaps
- Linear time characteristics only achievable for more than $2^{12^{20}}$ vertices.


Mikkel Thorup.
Undirected Single-Source Shortest Paths with Positive Integer Weights in Linear Time.
J. ACM, 46(3):362–394, 1999.

Thank you!