Data Structures and Algorithms

Kurt Mehlhorn
Peter Sanders

Organization

- Instructors: Kurt Mehlhorn and Peter Sanders
- want to know more about KM and his group: today, 1:30, MPI, room 024.
- Tutors: Guido Schäfer and Manual Bodirsky
- Media Support: XXX
- Classes: Monday and Wednesday at 11:00. Classes that fall on holidays are moved to Fridays.
- Additional Classes (Schmankerl) on Fridays, about once a month
- Exercises
  - handed out on Monday, to be handed in on the following Monday
  - Übungsgruppen meet on TODO
- Language: English

- The grade for the course is a combination of three grades:
  - exercises
  - midterm exam (will take place on Wednesday, December 16th, 11am)
  - final exam (will take place on Friday, March 2nd, 9am)
  - details on web-page
- WEB-page: see `www.mpi-sb.mpg.de/~mehlhorn` or sanders
- Prerequisites:
  - Einführung in Algorithmen und Datenstrukturen
  - Softwarepraktikum
- Course Notes and Books: see web-page
- LEDA and my books: CD-ROM
- Lectures will be recorded.

Challenges

- challenging tasks related to the course
- outside grading scheme, but champagne prizes
- Make LEDA look bad challenge (organized by Guido)
  - construct difficult instances or instance families for some of the LEDA algorithms
  - prize for the instance family with the largest asymptotic growth
- Programming Challenge: dynamic transitive closure
  - maintain a graph under edge insertions and deletions
  - answer reachability queries: is there a path from v to w?
  - there is a trivial solution (query = graph search from v), try to do better
- Master topics can be found on my WEB page

Contents

- Shortest Paths, Priority Queues, Amortization
- Network Flow and Bipartite Matchings
- Schmankerl: Min Cost Flow
- Generic Methods: Local Search, Simulated Annealing, linear programming and integer linear programming
- Hashing: Perfect Hashing, Universal Hashing,
- Computational Geometry: Convex Hulls, Delaunay Triangulations and Voronoi diagrams, augmented search trees
- Strings: Pattern Matching, Suffix Trees,
- . . .

Recent Developments I

- New Degree Programs
  - Angewandte Informatik (50% CS, 50% Business Administration)
  - Information und Kommunikation (50% CS, 50% EE)
  - Bioinformatik (50% CS, 50% Life Sciences)
  - PhD program (English language) Bachelor → Master → PhD
- New Scholarship Programs
  - Graduiertenkolleg (DFG)
  - Max-Planck-Research School (MPG)
  - Marie-Curie Training Site (EU)
- Center for Bioinformatics established (funded by DFG)

Recent Developments II

- Changes in AG1
  - Hans-Peter Lenhof: chair for bioinformatics (Uni des Saarlandes, Bielefeld, Uni München)
  - Job Sibeyn: Professor at Umea (Sweden)
  - Susanne Albers: offer for full professorship in Freiburg
  - Stefan Schirra: joined Think-and-Solve (SB)
  - many new faces
- Prizes
  - Hannah Bast: Otto Hahn Medaille
  - Petra Mutzel: SEL-Alcatel Prize
  - Wolfgang Wahlster: Beckurts Prize
  - Reinhard Wilhelm: ACM Fellow

The Shortest Path Problem

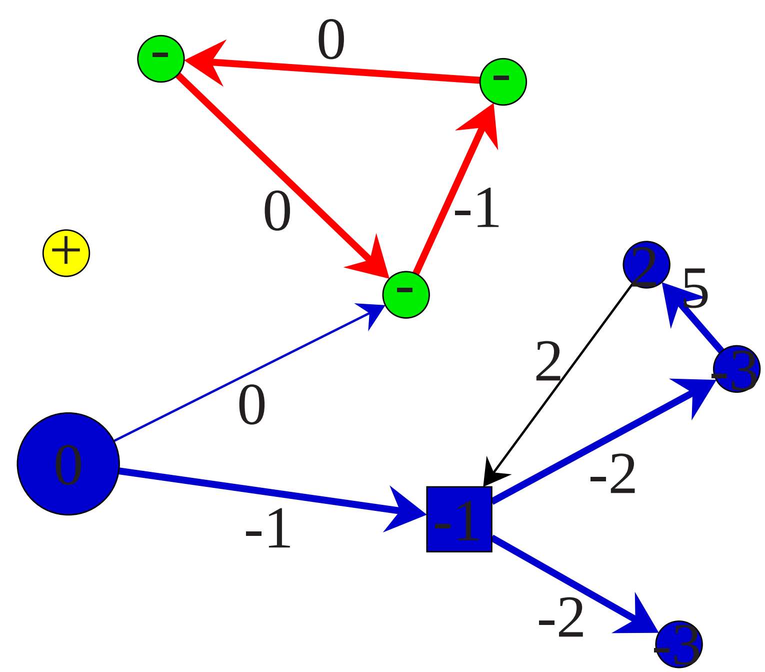
given a directed graph $G = (V, E)$, a cost function c on the edges, compute

- the shortest path between two given nodes s and t (single source, single sink)
- the shortest paths from a given node s to all other nodes (single source)
- the shortest paths between any pair of nodes (all pairs problem)

The Shortest Path Problem

given a directed graph $G = (V, E)$, a cost function c on the edges, compute

- the shortest path between two given nodes s and t (single source, single sink)
- the shortest paths from a given node s to all other nodes (single source)
- the shortest paths between any pair of nodes (all pairs problem)



- source node s = fat blue node
- yellow node has distance $+\infty$ from s
- blue nodes have finite distance from s
- square blue node has distance -1 from s. There are paths of length $-1, 4, 9, \ldots$
- green nodes have distance $-\infty$ from s

Prequisites and Further Reading

please recapitulate graphs, DFS and BFS, shortest paths and heaps

main source for lectures on shortest paths: [MN99]

additional sources: Tarjan's book [Tar83], Cormen-Leiserson-Rivest [CLR90].

For the lectures on amortization, in addition [Tar85, Meh98].

[CLR90] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. Introduction to Algorithms. MIT Press/McGraw-Hill Book Company, 1990.

[Meh98] K. Mehlhorn. Amortisierte Analyse. In Th. Ottmann, editor, Prinzipien des Algorithmenentwurfs. Spektrum Lehrbuch, 1998. www.mpi-sb.mpg.de/~{}mehlhorn/ftp/Amortization.ps.

[MN99] K. Mehlhorn and S. Näher. The LEDA Platform for Combinatorial and Geometric Computing. Cambridge University Press, 1999. 1018 pages.

[Tar83] R.E. Tarjan. Data Structures and Network Algorithms. SIAM, 1983.

[Tar85] R.E. Tarjan. Amortized computational complexity. SIAM Journal on Algebraic and Discrete Methods, 6(2):306–318, 1985.

Notation

- path $p = [e_1, e_2, \ldots, e_k]$, sequence of edges with $\mathit{target}(e_i) = \mathit{source}(e_{i+1})$ for $1 \leq i < k$, p is a path from $\mathit{source}(e_1)$ to $\mathit{target}(e_k)$.

Notation

- path $p = [e_1, e_2, \ldots , e_k]$, sequence of edges with $\mathit{target}(e_i) = \mathit{source}(e_{i+1})$ for $1 \leq i < k$, p is a path from $\mathit{source}(e_1)$ to $\mathit{target}(e_k)$.
- for every node v, there is the empty path $[\ ]$ from v to itself.

Notation

- path $p = [e_1, e_2, \ldots, e_k]$, sequence of edges with $\mathit{target}(e_i) = \mathit{source}(e_{i+1})$ for $1 \leq i < k$, p is a path from $\mathit{source}(e_1)$ to $\mathit{target}(e_k)$.
- for every node v, there is the empty path [ ] from v to itself.
- cardinality of a path = number of edges also called length

Notation

- path $p = [e_1, e_2, \ldots, e_k]$, sequence of edges with $\mathit{target}(e_i) = \mathit{source}(e_{i+1})$ for $1 \leq i < k$, p is a path from $\mathit{source}(e_1)$ to $\mathit{target}(e_k)$.
- for every node v, there is the empty path [ ] from v to itself.
- cardinality of a path = number of edges also called length
- given $c : E \to \mathbb{R}$, a cost (or length) function on the edges
  - cost of a path is the sum of the cost of its edges, i.e., $c(p) = \sum_{1 \leq i \leq k} c(e_i)$
  - empty path has cost zero

Notation

- path $p = [e_1, e_2, \ldots, e_k]$, sequence of edges with $\mathit{target}(e_i) = \mathit{source}(e_{i+1})$ for $1 \le i < k$, p is a path from $\mathit{source}(e_1)$ to $\mathit{target}(e_k)$.
- for every node v, there is the empty path [ ] from v to itself.
- cardinality of a path = number of edges   also called length
- given $c : E \to \mathbb{R}$, a cost (or length) function on the edges
  - cost of a path is the sum of the cost of its edges, i.e., $c(p) = \sum_{1 \le i \le k} c(e_i)$
  - empty path has cost zero
- $\mu(v, w) = \inf \{c(p)\ ;\ p \text{ is a path from } v \text{ to } w\} \in \mathbb{R} \cup \{-\infty, +\infty\}$.
  - $+\infty$, if there is no path from v to w
  - $-\infty$, if there are paths of arbitrarily small cost   (min does not exist).
  - $\in \mathbb{R}$, otherwise

Lemma 2 ***(a)*** *$\mu(v, w) = +\infty$ iff w is not reachable from v.*

(b) *$\mu(v, w) = -\infty$ iff there is a path from v to w containing a negative cycle.*

(c) *$-\infty < \mu(v, w) < +\infty$ otherwise (w is reachable from v and there is no path from v to w passing through a negative cycle). In this case, $\mu(v, w)$ is the length of a simple path from v to w.*

Proof:

$\blacksquare$

Lemma 3 ***(a)*** *$\mu(v, w) = +\infty$ iff w is not reachable from v.*

(b) *$\mu(v, w) = -\infty$ iff there is a path from v to w containing a negative cycle.*

(c) *$-\infty < \mu(v, w) < +\infty$ otherwise (w is reachable from v and there is no path from v to w passing through a negative cycle). In this case, $\mu(v, w)$ is the length of a simple path from v to w.*

Proof: Part (a) is true by definition.

$\blacksquare$

Lemma 4 ***(a)*** *$\mu(v, w) = +\infty$ iff w is not reachable from v.*

(b) *$\mu(v, w) = -\infty$ iff there is a path from v to w containing a negative cycle.*

(c) *$-\infty < \mu(v, w) < +\infty$ otherwise (w is reachable from v and there is no path from v to w passing through a negative cycle). In this case, $\mu(v, w)$ is the length of a simple path from v to w.*

Proof: Part (a) is true by definition.

(b,$\leftarrow$): going around the cycle one more time yields a path of smaller cost. Thus $\mu(v, w) = -\infty$.

∎

Lemma 5 ***(a)*** *$\mu(v,w)=+\infty$ iff w is not reachable from v.*

(b) *$\mu(v,w)=-\infty$ iff there is a path from v to w containing a negative cycle.*

(c) *$-\infty<\mu(v,w)<+\infty$ otherwise (w is reachable from v and there is no path from v to w passing through a negative cycle). In this case, $\mu(v,w)$ is the length of a simple path from v to w.*

Proof: Part (a) is true by definition.

(b, $\leftarrow$): going around the cycle one more time yields a path of smaller cost. Thus $\mu(v,w)=-\infty$.

(c, $\leftarrow$): Consider any path p from v to w. As long as p contains a cycle, remove it. Since p contains no negative cycle, the cost cannot go up. We obtain a simple path whose cost is at most the cost of p. Thus

$$\mu(v,w)=\inf\{c(p)\ ;\ p \text{ is a simple path from } v \text{ to } w\}.$$

The number of simple paths is finite and hence $\mu(v,w)=c(p)$ for some simple path p.

∎

Lemma 6 ***(a)*** *$\mu(v, w) = +\infty$ iff w is not reachable from v.*

(b) *$\mu(v, w) = -\infty$ iff there is a path from v to w containing a negative cycle.*

(c) *$-\infty < \mu(v, w) < +\infty$ otherwise (w is reachable from v and there is no path from v to w passing through a negative cycle). In this case, $\mu(v, w)$ is the length of a simple path from v to w.*

Proof: Part (a) is true by definition.
(b, ←): going around the cycle one more time yields a path of smaller cost. Thus $\mu(v, w) = -\infty$.
(c, ←): Consider any path p from v to w. As long as p contains a cycle, remove it. Since p contains no negative cycle, the cost cannot go up. We obtain a simple path whose cost is at most the cost of p. Thus

$$\mu(v, w) = \inf \{c(p) \,;\, p \text{ is a simple path from } v \text{ to } w\} .$$

The number of simple paths is finite and hence $\mu(v, w) = c(p)$ for some simple path p.
(b, →) and (c, →) since (a), (b), and (c) are exhaustive. ∎

From now on: single source problem with source s

$\mu(v) = \mu(s, v)$, distance from s to v

Arithmetic and order on $\mathbb{R} \cup \{-\infty, +\infty\}$: $-\infty < x < +\infty$, $+\infty + x = +\infty$, and $-\infty + x = -\infty$ for all $x \in \mathbb{R}$.

Lemma 8 (Characterization of μ) *μ satisfies the following equations:*

$$\begin{aligned}
\mu(s) &= \min(0, \min\{\mu(u) + c(e) \; ; e = (u, s) \in E\}) \\
\mu(v) &= \min\{\mu(u) + c(e) \; ; e = (u, v) \in E\} \textit{ for } v \neq s
\end{aligned}$$

From now on: single source problem with source s

$\mu(v) = \mu(s, v)$, distance from s to v

Arithmetic and order on $\mathbb{R} \cup \{-\infty, +\infty\}$: $-\infty < x < +\infty$, $+\infty + x = +\infty$, and $-\infty + x = -\infty$ for all $x \in \mathbb{R}$.

Lemma 9 (Characterization of μ) *μ satisfies the following equations:*

$$\begin{aligned}\mu(s) &= \min(0, \min\{\mu(u) + c(e)\ ;\ e = (u, s) \in E\})\\ \mu(v) &= \min\{\mu(u) + c(e)\ ;\ e = (u, v) \in E\} \text{ for } v \neq s\end{aligned}$$

Proof: We only consider the case $v \neq s$ and leave the case $v = s$ to the reader. Any path p from s to v consists of a path from s to some node u plus an edge from u to v. Thus

$$\begin{aligned}\mu(v) &= \inf\{c(p)\ ;\ p \text{ is a path from } s \text{ to } v\}\\ &= \min_u \inf\left\{c(p') + c(e)\ ;\ p' \text{ is a path from } s \text{ to } u \text{ and } e = (u, v) \in E\right\}\\ &= \min\{\mu(u) + c(e)\ ;\ e = (u, v) \in E\}.\end{aligned}$$

∎

Lemma 10 (sufficient conditions for a function being equal to μ)

If d is a function from V to $\mathbb{R} \cup \{-\infty, +\infty\}$ with

- $d(v) \geq \mu(v)$ *for all* $v \in V$,
- $d(s) \leq 0$, *and*
- $d(v) \leq d(u) + c(u, v)$ *for all* $e = (u, v) \in E$

then $d(v) = \mu(v)$ *for all* $v \in V$.

Lemma 10 (sufficient conditions for a function being equal to μ)
If d is a function from V to $\mathbb{R} \cup \{-\infty, +\infty\}$ with

- $d(v) \geq \mu(v)$ *for all* $v \in V$,
- $d(s) \leq 0$, *and*
- $d(v) \leq d(u) + c(u, v)$ *for all* $e = (u, v) \in E$

then $d(v) = \mu(v)$ *for all* $v \in V$.

Proof: Assume otherwise and let v be such that $d(v) > \mu(v)$. Then $\mu(v) < +\infty$. We distinguish two cases: $\mu(v) > -\infty$ and $= -\infty$.

Lemma 10 (sufficient conditions for a function being equal to μ)
If d is a function from V to $\mathbb{R} \cup \{-\infty, +\infty\}$ with

- $d(v) \geq \mu(v)$ *for all* $v \in V$,
- $d(s) \leq 0$, *and*
- $d(v) \leq d(u) + c(u, v)$ *for all* $e = (u, v) \in E$

then $d(v) = \mu(v)$ *for all* $v \in V$.

Proof: Assume otherwise and let v be such that $d(v) > \mu(v)$. Then $\mu(v) < +\infty$. We distinguish two cases: $\mu(v) > -\infty$ and $= -\infty$.

If $\mu(v) > -\infty$, let $[s = v_0, v_1, \ldots, v_k = v]$ be a shortest path from s to v. We have $\mu(s) = 0 = d(s)$, $\mu(v_i) = \mu(v_{i-1}) + c(v_{i-1}, v_i)$ for $i > 0$, and $\mu(v) < d(v)$. Thus, there is a least $i > 0$ with $\mu(v_i) < d(v_i)$ and hence

$$d(v_i) > \mu(v_i) = \mu(v_{i-1}) + c(v_i, v_{i-1}) = d(v_{i-1}) + c(v_i, v_{i-1}),$$

a contradiction.

If $\mu(v) = -\infty$, let $[s = v_0, v_1, \ldots, v_i, \ldots, v_j, \ldots, v_k = v]$ be a path from s to v containing a negative cycle. Such a path exists by Lemma 6. Assume that the sub-path from v_i to v_j is a negative cycle. If $d(v) > \mu(v)$ then $d(v) > -\infty$ and hence $d(v_l) > -\infty$ for all l, $0 \leq l \leq k$.

If $\mu(v) = -\infty$, let $[s = v_0, v_1, \ldots, v_i, \ldots, v_j, \ldots, v_k = v]$ be a path from s to v containing a negative cycle. Such a path exists by Lemma 6. Assume that the sub-path from v_i to v_j is a negative cycle. If $d(v) > \mu(v)$ then $d(v) > -\infty$ and hence $d(v_l) > -\infty$ for all l, $0 \le l \le k$.

Thus,

$$
\begin{aligned}
d(v_i) &= d(v_j) && \text{since } v_i = v_j \\
&\le d(v_{j-1}) + c(v_{j-1}, v_j) \\
&\le d(v_{j-2}) + c(v_{j-2}, v_{j-1}) + c(v_{j-1}, v_j) \\
&\vdots \\
&\le d(v_i) + \textstyle\sum_{l=i}^{j-1} c(v_l, v_{l+1}),
\end{aligned}
$$

and hence $\sum_{l=i}^{j-1} c(v_l, v_{l+1}) \ge 0$, a contradiction to the fact that the sub-path from v_i to v_j is a negative cycle. ∎

Call an edge $e = (u, v)$ red if $d(u) + c(e) < d(v)$ and call it black otherwise.

Argument above shows that negative cycles contain at least one red edge.

Generic Shortest Path Algorithm

Recall: If d satisfies (1) $d(v) \geq \mu(v)$ for all $v \in V$, (2) $d(s) \leq 0$, and (3) $d(v) \leq d(u) + c(u, v)$ for all $e = (u, v) \in E$, then $d(v) = \mu(v)$ for all $v \in V$.

The generic algorithm maintains a function d satisfying (1) and (2) and aims at establishing (3). We call $d(v)$ the tentative distance label of v.

Generic Shortest Path Algorithm

Recall: If d satisfies (1) $d(v) \geq \mu(v)$ for all $v \in V$, (2) $d(s) \leq 0$, and (3) $d(v) \leq d(u) + c(u,v)$ for all $e = (u,v) \in E$, then $d(v) = \mu(v)$ for all $v \in V$.

The generic algorithm maintains a function d satisfying (1) and (2) and aims at establishing (3). We call $d(v)$ the tentative distance label of v.

$d(s) = 0$; $d(v) = \infty$ for $v \neq s$;

while there is an edge $e = (u,v) \in E$ with $d(v) > d(u) + c(e)$ $\quad$ e is red

{ // relax e (view e as a rubber band which wants to keep $d(v)$ below or at $d(u) + c(e)$.

$\quad d(v) = d(u) + c(e)$; $\quad$ relax it to make it black

}

Generic Shortest Path Algorithm

Recall: If d satisfies (1) $d(v) \geq \mu(v)$ for all $v \in V$, (2) $d(s) \leq 0$, and (3) $d(v) \leq d(u) + c(u, v)$ for all $e = (u, v) \in E$, then $d(v) = \mu(v)$ for all $v \in V$.

The generic algorithm maintains a function d satisfying (1) and (2) and aims at establishing (3). We call $d(v)$ the tentative distance label of v.

$d(s) = 0$; $d(v) = \infty$ for $v \neq s$;
while there is an edge $e = (u, v) \in E$ with $d(v) > d(u) + c(e)$ — e is red
{ // relax e (view e as a rubber band which wants to keep $d(v)$ below or at $d(u) + c(e)$.
  $d(v) = d(u) + c(e)$; — relax it to make it black
}

(1) and (2) are invariants of the algorithm:

$d(s)$ never increases and hence $d(s) \leq 0$ always and

If $d(v) < +\infty$, $d(v)$ is the length of some path from s to v and hence $d(v) \geq \mu(v)$ always.

Generic Shortest Path Algorithm

Recall: If d satisfies (1) $d(v) \geq \mu(v)$ for all $v \in V$, (2) $d(s) \leq 0$, and (3) $d(v) \leq d(u) + c(u,v)$ for all $e = (u,v) \in E$, then $d(v) = \mu(v)$ for all $v \in V$.

The generic algorithm maintains a function d satisfying (1) and (2) and aims at establishing (3). We call $d(v)$ the tentative distance label of v.

$d(s) = 0$; $d(v) = \infty$ for $v \neq s$;
while there is an edge $e = (u,v) \in E$ with $d(v) > d(u) + c(e)$ $\quad$ e is red
{ // relax e (view e as a rubber band which wants to keep $d(v)$ below or at $d(u) + c(e)$.
$\quad d(v) = d(u) + c(e)$; $\quad$ relax it to make it black
}

(1) and (2) are invariants of the algorithm:

$d(s)$ never increases and hence $d(s) \leq 0$ always and

If $d(v) < +\infty$, $d(v)$ is the length of some path from s to v and hence $d(v) \geq \mu(v)$ always.

When the algorithm terminates, we also have (3). **GREAT**

Problems:

1. GA does not determinate in the presence of negative cycles
2. GA may have exponential running time even without negative cycles

Problems:

1. GA does not determinate in the presence of negative cycles
2. GA may have exponential running time even without negative cycles

Observation (addresses second item): When $d(v)$ is decreased, the edges out of v may turn red.

Idea: Maintain a set U with $U \supseteq \{u\ ;\text{ there is a red edge out of } u\}$ and rewrite the generic algorithm as:

```
d(s) = 0; d(v) = ∞ for v ≠ s; U = {s};
while U ≠ Ø
{ select u ∈ U and remove it;
  forall edges e = (u, v)
  { if d(u) + c(e) < d(v)
    { add v to U;
      d(v) = d(u) + c(e);
    }
  }
}
```

Idea: Maintain a set U with $U \supseteq \{u \,;\, \exists (u, v) \in E \text{ with } d(u) + c(u, v) < d(v)\}$ and rewrite the generic algorithm as:

$d(s) = 0$; $d(v) = \infty$ for $v \neq s$; $U = \{s\}$;
while $U \neq \emptyset$
{ select $u \in U$ and remove it;
  forall edges $e = (u, v)$
  { **if** $d(u) + c(e) < d(v)$
    { add v to U;
      $d(v) = d(u) + c(e)$;
    }
  }
}

Question: Which u do we select from U?

Idea: Maintain a set U with $U \supseteq \{u\ ;\ \exists(u, v) \in E \text{ with } d(u) + c(u, v) < d(v)\}$ and rewrite the generic algorithm as:

$d(s) = 0$; $d(v) = \infty$ for $v \neq s$; $U = \{s\}$;
while $U \neq \emptyset$
{ select $u \in U$ and remove it;
  forall edges $e = (u, v)$
  { **if** $d(u) + c(e) < d(v)$
    { add v to U;
      $d(v) = d(u) + c(e)$;
    }
  }
}

Question: Which u do we select from U?

Answer:
- There is always an optimal choice
- In some situations, the optimal choice can be made efficiently.

Let $V_f = \{v \in V \,;\, -\infty < \mu(v) < \infty\}$ nodes in V_f have shortest paths

Lemma 12 (Existence of Optimal Choice)

(a) *When a node u is removed from U with $d(u) = \mu(u)$, it is never added to U again.*

Let $V_f = \{v \in V \; ; \; -\infty < \mu(v) < \infty\}$ nodes in V_f have shortest paths

Lemma 13 (Existence of Optimal Choice)

(a) *When a node u is removed from U with $d(u) = \mu(u)$, it is never added to U again.* (it is an optimal choice)

Let $V_f = \{v \in V \; ; \; -\infty < \mu(v) < \infty\}$ nodes in V_f have shortest paths

Lemma 14 (Existence of Optimal Choice)

(a) *When a node u is removed from U with $d(u) = \mu(u)$, it is never added to U again.* (it is an optimal choice)

(b) *As long as $d(v) > \mu(v)$ for some $v \in V_f$: for any $v \in V_f$ with $d(v) > \mu(v)$ there is a $u \in U$ with $d(u) = \mu(u)$ and lying on a shortest path from s to v.*

Let $V_f = \{v \in V \; ; \; -\infty < \mu(v) < \infty\}$ nodes in V_f have shortest paths

Lemma 15 (Existence of Optimal Choice)

(a) *When a node u is removed from U with $d(u) = \mu(u)$, it is never added to U again.* (it is an optimal choice)

(b) *As long as $d(v) > \mu(v)$ for some $v \in V_f$: for any $v \in V_f$ with $d(v) > \mu(v)$ there is a $u \in U$ with $d(u) = \mu(u)$ and lying on a shortest path from s to v.*

Proof: (a) We have $d(u) \geq \mu(u)$ always. Also, when u is added to U, its tentative distance value $d(u)$ has just been decreased. Thus, if a node u is removed from U with $d(u) = \mu(u)$, it will never be added to U at a later time.

(b) Let $[s = v_0, v_1, \ldots, v_k = v]$ be a shortest path from s to v. Then $\mu(s) = 0 = d(s)$ and $d(v_k) > \mu(v_k)$. Let i be minimal such that $d(v_i) > \mu(v_i)$. Then $i > 0$, $d(v_{i-1}) = \mu(v_{i-1})$ and

$$d(v_i) > \mu(v_i) = \mu(v_{i-1}) + c(v_{i-1}, v_i) = d(v_{i-1}) + c(v_{i-1}, v_i).$$

Thus, $v_{i-1} \in U$. ∎

Lemma 16 (Algorithmic optimal choice)

non-negative costs: *If* $c(e) \geq 0$ *for all* $e \in E$ *then* $d(u) = \mu(u)$ *for the node* $u \in U$ *with minimal* $d(u)$*.*

acyclic graphs: *If* G *is acyclic and* $u_0, u_1, \ldots, u_{n-1}$ *is a topological order of the nodes of* G*, i.e., if* $(u_i, u_j) \in E$ *then* $i < j$*, then* $d(u) = \mu(u)$ *for the node* $u = u_i \in U$ *with* i *minimal.*

Lemma 16 (Algorithmic optimal choice)

non-negative costs: *If $c(e) \geq 0$ for all $e \in E$ then $d(u) = \mu(u)$ for the node $u \in U$ with minimal $d(u)$.*

acyclic graphs: *If G is acyclic and $u_0, u_1, \ldots, u_{n-1}$ is a topological order of the nodes of G, i.e., if $(u_i, u_j) \in E$ then $i < j$, then $d(u) = \mu(u)$ for the node $u = u_i \in U$ with i minimal.*

Proof: Assume otherwise, i.e., $d(u) > \mu(u)$ for the node u specified. By the preceding lemma there is a node $z \in U$ lying on a shortest path from s to u with $d(z) = \mu(z)$. We now distinguish cases.

In the case of non-negative edge costs, we have $\mu(z) \leq \mu(u)$. Thus, $d(z) < d(u)$, contradicting the choice of u.

In the case of acyclic graphs, we have $z = u_j$ for some $j < i$, contradicting the choice of u. ∎

Lemma 16 (Algorithmic optimal choice)

non-negative costs: *If* $c(e) \geq 0$ *for all* $e \in E$ *then* $d(u) = \mu(u)$ *for the node* $u \in U$ *with minimal* $d(u)$.

acyclic graphs: *If* G *is acyclic and* $u_0, u_1, \ldots, u_{n-1}$ *is a topological order of the nodes of* G*, i.e., if* $(u_i, u_j) \in E$ *then* $i < j$*, then* $d(u) = \mu(u)$ *for the node* $u = u_i \in U$ *with* i *minimal.*

Proof: Assume otherwise, i.e., $d(u) > \mu(u)$ for the node u specified. By the preceding lemma there is a node $z \in U$ lying on a shortest path from s to u with $d(z) = \mu(z)$. We now distinguish cases.

In the case of non-negative edge costs, we have $\mu(z) \leq \mu(u)$. Thus, $d(z) < d(u)$, contradicting the choice of u.

In the case of acyclic graphs, we have $z = u_j$ for some $j < i$, contradicting the choice of u. ∎

Lemma is basis for Dijkstra's algorithm for graphs with non-negative edge costs and for a linear time algorithm for acyclic graphs.

Topologically Sorting Acyclic Graphs (Review)

DFS computes a topological order in linear time $O(n + m)$.

Topologically Sorting Acyclic Graphs (Review)

DFS computes a topological order in linear time $O(n+m)$.

- G is acyclic iff DFS produces no back edges
- tree, forward and cross edges go from larger to smaller completion number
- back edges go from smaller to larger completion number
- (negative of) completion numbers are a topological order

Topologically Sorting Acyclic Graphs (Review)

DFS computes a topological order in linear time $O(n+m)$.

- G is acyclic iff DFS produces no back edges
- tree, forward and cross edges go from larger to smaller completion number
- back edges go from smaller to larger completion number
- (negative of) completion numbers are a topological order

calls to DFS are either nested or disjoint, consider calls $DFS(v)$ and $DFS(w)$.

$[_v \ldots]_v \ldots [_w \ldots]_w$ or $[_w \ldots]_w \ldots [_v \ldots]_v$ or $[_w \ldots [_v \ldots]_v \ldots]_w$ or $[_v \ldots [_w \ldots]_w \ldots]_v$

Topologically Sorting Acyclic Graphs (Review)

DFS computes a topological order in linear time $O(n+m)$.

- G is acyclic iff DFS produces no back edges
- tree, forward and cross edges go from larger to smaller completion number
- back edges go from smaller to larger completion number
- (negative of) completion numbers are a topological order

calls to DFS are either nested or disjoint, consider calls $DFS(v)$ and $DFS(w)$.

$[_v \ldots]_v \ldots [_w \ldots]_w$ or $[_w \ldots]_w \ldots [_v \ldots]_v$ or $[_w \ldots [_v \ldots]_v \ldots]_w$ or $[_v \ldots [_w \ldots]_w \ldots]_v$

- If $(v, w) \in E$, $DFS(w)$ must start before $DFS(v)$ ends; exludes first poss.
- If $(v, w) \in E$, there is no path from w to v and hence $DFS(v)$ cannot be nested in $DFS(w)$; excludes third possibility.
- second and the fourth poss. remain. Thus $compnum[w] < compnum[v]$.

Acyclic Graphs

Let G be an acyclic graph, $v_1, v_2, \ldots, v_n$ be an ordering of the nodes such that $(v_i, v_j) \in E$ implies $i \leq j$.

The Algorithm:

Compute topological ordering;

Let $s = v_k$; (nodes v_j with $j < k$ are not reachable from s)

forall $(i, k \leq i \leq n$, in increasing order)

$\{$ **if** $(d(v_i) < \infty)$

$\{$ propagate $d(v_i)$ over all edges out of v_i; $\}$

$\}$

Acyclic Graphs

Let G be an acyclic graph, $v_1, v_2, \ldots, v_n$ be an ordering of the nodes such that $(v_i, v_j) \in E$ implies $i \leq j$.

The Algorithm:

Compute topological ordering; takes time $O(n+m)$

Let $s = v_k$; (nodes v_j with $j < k$ are not reachable from s)

forall (i, $k \leq i \leq n$, in increasing order)

{ **if** ($d(v_i) < \infty$)

{ propagate $d(v_i)$ over all edges out of v_i; }

}

Acyclic Graphs

Let G be an acyclic graph, $v_1, v_2, \ldots, v_n$ be an ordering of the nodes such that $(v_i, v_j) \in E$ implies $i \leq j$.

The Algorithm:

Compute topological ordering; takes time $O(n+m)$

Let $s = v_k$; (nodes v_j with $j < k$ are not reachable from s)

forall $(i, k \leq i \leq n$, in increasing order)

$\{$ **if** $(d(v_i) < \infty)$

$\{$ propagate $d(v_i)$ over all edges out of v_i; takes time $O(outdeg(v_i))$ $\}$

$\}$

Acyclic Graphs

Let G be an acyclic graph, $v_1, v_2, \ldots, v_n$ be an ordering of the nodes such that $(v_i, v_j) \in E$ implies $i \leq j$.

The Algorithm:

Compute topological ordering; takes time $O(n+m)$

Let $s = v_k$; (nodes v_j with $j < k$ are not reachable from s)

forall $(i, k \leq i \leq n$, in increasing order)

{ **if** $(d(v_i) < \infty)$

{ propagate $d(v_i)$ over all edges out of v_i; takes time $O(outdeg(v_i))$ }

}

total time = $O(n+m) + O(n) + \sum_{1 \leq i \leq n} outdeg(v_i) = O(n+m)$

Acyclic Graphs

Let G be an acyclic graph, $v_1, v_2, \ldots, v_n$ be an ordering of the nodes such that $(v_i, v_j) \in E$ implies $i \leq j$.

The Algorithm:

Compute topological ordering; takes time $O(n+m)$

Let $s = v_k$; (nodes v_j with $j < k$ are not reachable from s)

forall $(i, k \leq i \leq n$, in increasing order)

{ **if** $(d(v_i) < \infty)$

{ propagate $d(v_i)$ over all edges out of v_i; takes time $O(outdeg(v_i))$ }

}

total time = $O(n+m) + O(n) + \sum_{1 \leq i \leq n} outdeg(v_i) = O(n+m)$

Theorem 2 *Shortest paths in acyclic graphs can be computed in time* $O(n+m)$.

An Implementation Issue

How can we represent $+\infty$?

Some number types have a represention for $+\infty$:

double does and *int* does not.

Warning: Do not use *MAXINT* for $+\infty$, since $MAXINT + 1 \neq MAXINT$.

An Implementation Issue

How can we represent $+\infty$?

Some number types have a represention for $+\infty$:
double does and *int* does not.

Warning: Do not use *MAXINT* for $+\infty$, since $MAXINT + 1 \neq MAXINT$.

Solution: (used in LEDA)

Maintain for each node v, the edge $pred[v]$ into v which defines $d[v]$.

$pred[v]$ is initialized to *nil* and updated whenever $d[v]$ is changed.

An Implementation Issue

How can we represent $+\infty$?

Some number types have a represention for $+\infty$:

double does and *int* does not.

Warning: Do not use *MAXINT* for $+\infty$, since $MAXINT + 1 \neq MAXINT$.

Solution: (used in LEDA)

Maintain for each node v, the edge $pred[v]$ into v which defines $d[v]$.

$pred[v]$ is initialized to *nil* and updated whenever $d[v]$ is changed.

we have the invariant: $d(v) = +\infty$ iff $v \neq s$ and $pred(v) = nil$

```
template <class NT>
void ACYCLIC_SHORTEST_PATH_T(const graph& G, node s, const edge_array
                             node_array<NT>& dist, node_array<edge>& 
{ node_array<int> top_ord(G); node w; edge e;
  TOPSORT(G,top_ord);  // top_ord is now a topological ordering of G
  array<node> v(1,G.number_of_nodes());
  forall_nodes(w,G) v[ top_ord[w] ] = w;   // top_ord[ v[i] ] == i fo
  dist[s] = 0;
  forall_nodes(w,G) pred[w] = nil;
  for (int i = top_ord[s]; i <= G.number_of_nodes(); i++)
  { node u = v[i];
    if ( pred[u] == nil && u != s ) continue;  // dist[u] is plus inf
    forall_out_edges(e,u)
    { node w = G.target(e);
      if ( pred[w] == nil || dist[u] + c[e] < dist[w])
      { pred[w] = e; dist[w] = dist[u] + c[e]; }
    }
  }
}
```