Tom-Tom: smart routing

Alexandra Silva and Henk Barendregt

{alexandra,henk}@cs.ru.nl
http://www.cs.ru.nl/~{alexandra,henk}

Institute for Computing and Information Sciences
Radboud University Nijmegen

20 March 2012

Recap

Last weeks' message (or *leerdoelen*)

- Knowing some algorithms.
- Going from algorithm to its T.
- Estimating from above and below this T.
- Doing some math (recurrences and so on).
- Knowing difference between worse case and average case and best case.
- Knowing what is P/NP (more about this in the werkcollege this week).

Today

- graph algorithms
- T will be a function of two variables (V,E)

Today

- graph algorithms
- T will be a function of two variables (V,E)
- in practice, this class does not add much to your *leerdoelen* (it is about *consolidate*).

Let's go from A to B

How to find the shortest route between two points on a map.
Input:

- Directed graph $G = (V, E)$
- Weight function $W: E \to \mathbb{R}$

Let's go from A to B

How to find the shortest route between two points on a map.
Input:

- Directed graph $G = (V, E)$
- Weight function $W : E \to \mathbb{R}$

Weight of path $p =< v_0, v_1, \ldots, v_k >$

$$w(p) = \sum_{i=1}^{k} W(v_{i-1}, v_i) = \text{ sum of edge weights on path } p.$$

Let's go from A to B

How to find the shortest route between two points on a map.
Input:

- Directed graph $G = (V, E)$
- Weight function $W \colon E \to \mathbb{R}$

Weight of path $p = < v_0, v_1, \ldots, v_k >$

$$w(p) = \sum_{i=1}^{k} W(v_{i-1}, v_i) = \text{ sum of edge weights on path } p.$$

Can think of weights as representing any measure that

- accumulates linearly along a path,
- we want to minimize.
- Examples: time, cost, penalties, loss.

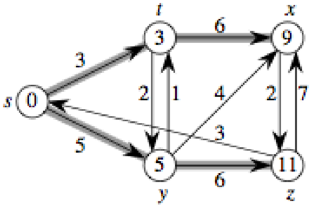
Let's go from A to B

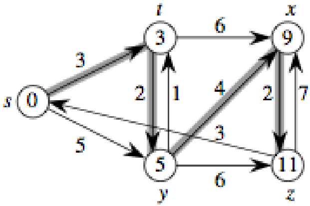
Shortest-path weight u to v

$$\delta(u,v) = \begin{cases} min\{w(p) \mid u \xrightarrow{p} v\} & \text{if there exists a path from } u \text{ to } v \\ \infty & \textit{otherwise}. \end{cases}$$

Shortest path u to v is any path p such that $w(p) = \delta(u,v)$.

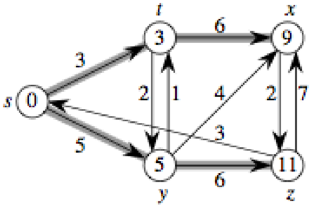
Example: shortest paths are not unique

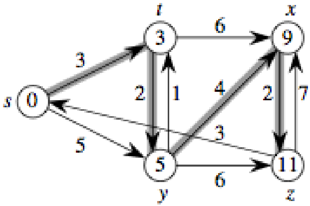




[labels in the nodes: distance from s]

Example: shortest paths are not unique





[labels in the nodes: distance from *s*]

- This example shows that the shortest path might not be unique.
- It also shows that when we look at shortest paths from one vertex to all other vertices, the shortest paths are organized as a tree.

Some properties of shortest paths

Lemma

Any subpath of a shortest path is a shortest path.

Some properties of shortest paths

Lemma

Any subpath of a shortest path is a shortest path.

Proof. Take a shortest path from u to v.

Then, $\delta(u, v) = w(p) = w(p_{ux}) + w(p_{xy}) + w(p_{yv})$.

Some properties of shortest paths

Lemma

Any subpath of a shortest path is a shortest path.

Proof. Take a shortest path from u to v.

Then, $\delta(u, v) = w(p) = w(p_{ux}) + w(p_{xy}) + w(p_{yv})$.
Now, suppose there exists a shorter path from x to y, say p'_{xy}.
Then $w(p'_{xy}) < w(p_{xy})$. Construct p':

and observe:

$$w(p') = w(p_{ux})+w(p'_{xy})+w(p_{yv}) < w(p_{ux})+w(p_{xy})+w(p_{yv}) = w(p).$$

So p was not a shortest path after all!

Some properties of shortest paths

Lemma (Triangle inequality)

For all $(u, v) \in E$*, we have* $\delta(s, v) \leq \delta(s, u) + w(u, v)$*.*

Proof.

Weight of shortest path s to v is $\leq$ than the weight of any path s to v. Path s to v via u is a path s to v, and if we use a shortest path s to u, its weight is $\delta(s, u) + w(u, v)$. $\square$

Calculating shortest path: Dijkstra's algorithm

Goal: Compute shortest path between u and v.

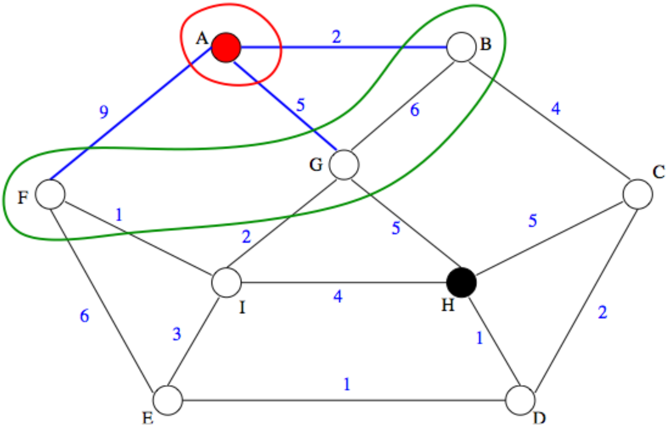
An obvious strategy: brute force. Build all paths from u to v and then select the shortest.

We will now see a more efficient algorithm: Dijkstra's algorithm.

Calculating shortest path: Dijkstra's algorithm

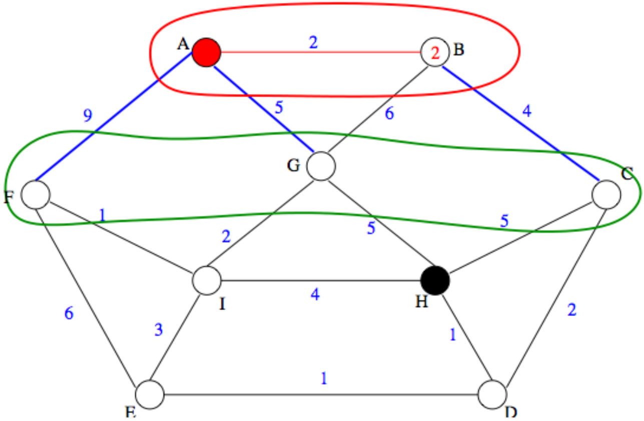
- Very similar to Prim's algorithm for minimal search trees.
- In each step it selects a node from the immediate neighborhood to add to the tree that is building.
- The algorithm constructs longer and longer paths, organized in a tree, from u until it finds v.
- Have two sets of vertices:
  $S =$ vertices whose final shortest-path weights are determined, $Q =$ priority queue $= V \setminus S$.

Example: Dijkstra's algorithm



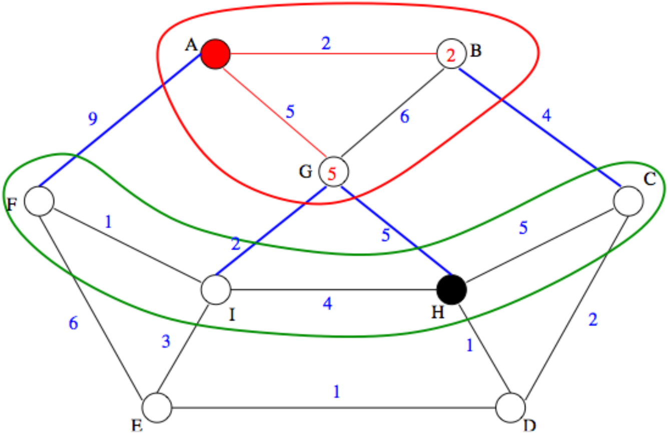
$$\delta(A,A)+w(A,B)=2; \delta(A,A)+w(A,F)=9; \delta(A,A)+w(A,G)=5.$$

Example: Dijkstra's algorithm



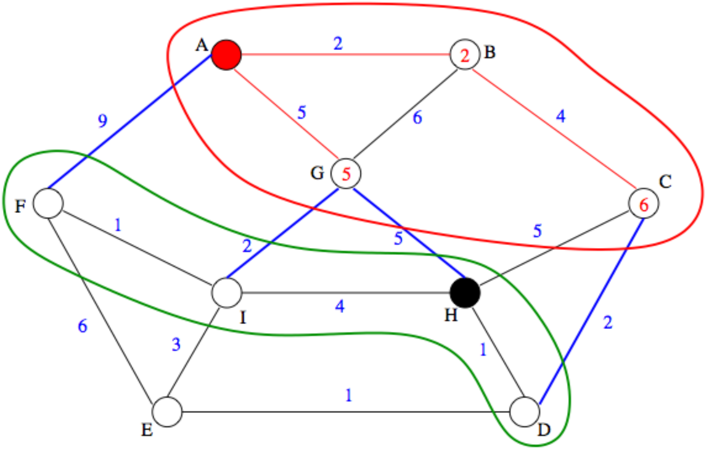
$\delta(A,B)+w(B,C) = 6; \delta(A,A)+w(A,F) = 9; \delta(A,A)+w(A,G) = 5.$

Example: Dijkstra's algorithm



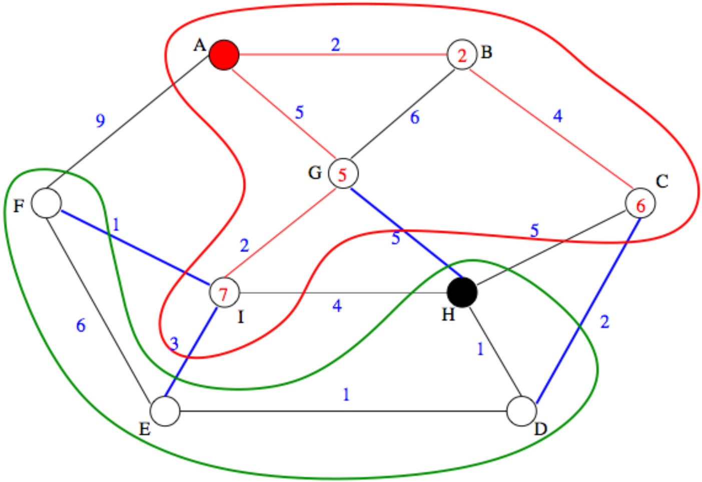
$$\delta(A,B) + w(B,C) = 6; \delta(A,A) + w(A,F) = 9;$$
$$\delta(A,G) + w(G,H) = 10; \delta(A,G) + w(G,I) = 7.$$

Example: Dijkstra's algorithm

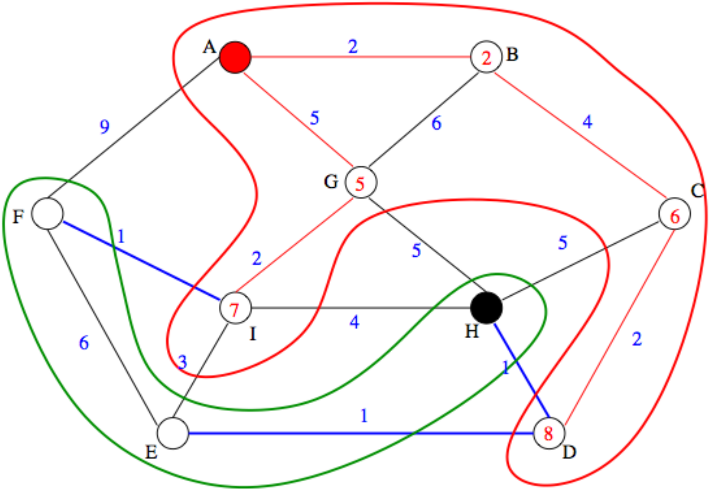


$$d(A, C) + w(C, D) = 8; d(A, A) + w(A, F) = 9;$$
$$d(A, G) + w(G, H) = 10; d(A, G) + w(G, I) = 7.$$

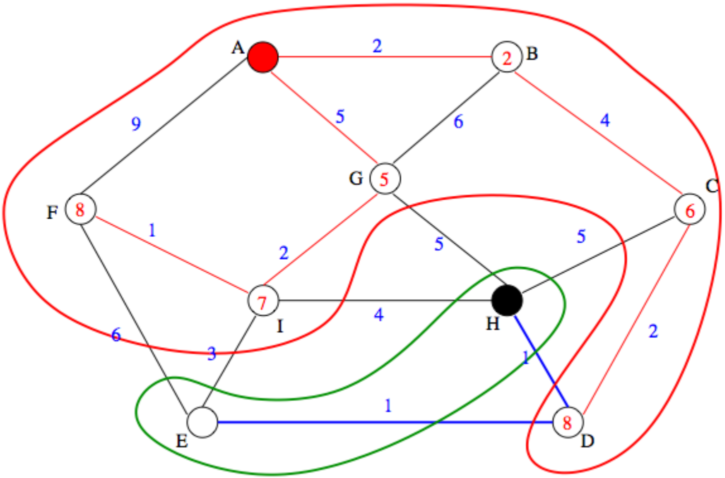
Example: Dijkstra's algorithm



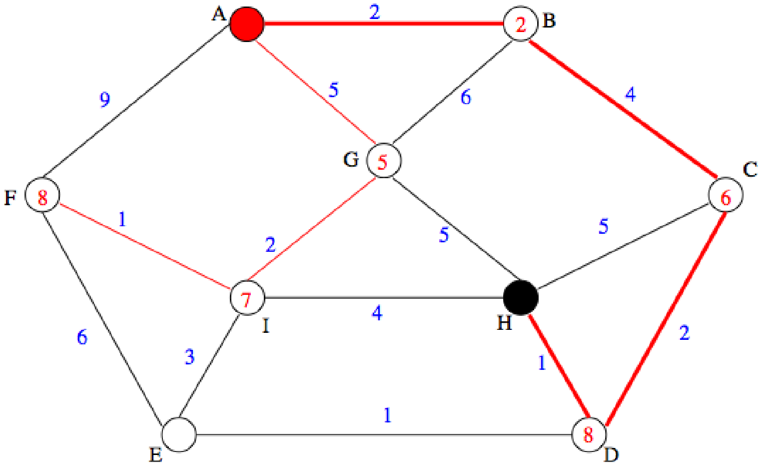
Example: Dijkstra's algorithm



Example: Dijkstra's algorithm



Example: Dijkstra's algorithm



Break


Samen houden we het studentenleven draaiende, word bestuurslid!
19 T/M 23 MAART: WEEK VAN HET STUDENTBESTUUR
Woensdag 21 maart: Open Kamerdag
Donderdag 22 maart: Besturen iets voor jou?!
Voorlichtingsbijeenkomst
CC5 > 12.30 - 13.30 uur

Dijkstra's algorithm: pseudocode

```
DIJKSTRA(V, E, w, s)
INIT-SINGLE-SOURCE(V, s)
S ← Ø
Q ← V          ▷ i.e., insert all vertices into Q
while Q ≠ Ø
    do u ← EXTRACT-MIN(Q)
       S ← S ∪ {u}
       for each vertex v ∈ Adj[u]
         if d[v] > d[u] + w(u, v)
           then d[v] ← d[u] + w(u, v)
                π[v] ← u
```

- Init creates 2 arrays: d and π. $d[v]$ contains (in the end) $\delta(s, v)$ and π is the parent relation.
- Note that the **while** loop iterates $|V|$ times.
- for each vertex in the adjacency of u we check if we can improve the shortest path we have so far to v by going via u.

Dijkstra's algorithm: analysis

First observation: depends on the complexity of EXTRACT-MIN, which in turn depends on the implementation of Q, the priority queue.

EXTRACT-MIN is called once per vertex.
If binary heap, this operation takes $O(\lg |V|)$ time.

Dijkstra's algorithm: analysis

First observation: depends on the complexity of EXTRACT-MIN, which in turn depends on the implementation of Q, the priority queue.

EXTRACT-MIN is called once per vertex.
If binary heap, this operation takes $O(\lg |V|)$ time.

The for loop iterates a total number of $|E|$ times (for each vertex we look once at all its edges).

Dijkstra's algorithm: analysis

First observation: depends on the complexity of EXTRACT-MIN, which in turn depends on the implementation of Q, the priority queue.

EXTRACT-MIN is called once per vertex.
If binary heap, this operation takes $O(\lg|V|)$ time.

The for loop iterates a total number of $|E|$ times (for each vertex we look once at all its edges).
So, the total time will be $O((|V|+|E|)\lg|V|)$.

Dijkstra's algorithm: analysis

First observation: depends on the complexity of Extract-Min, which in turn depends on the implementation of Q, the priority queue.

Extract-Min is called once per vertex.
If binary heap, this operation takes $O(\lg |V|)$ time.

The for loop iterates a total number of $|E|$ times (for each vertex we look once at all its edges).
So, the total time will be $O((|V| + |E|) \lg |V|)$.

We could use a more efficient data structure. For instance, a heap in which computing the min takes $O(1)$ time. And then redo the analysis above!

Shortest vs. longest simple paths

- We saw that we can find shortest paths from a single source in a directed graph $G = (V, E)$ in $O(|E| \lg |V|)$ time.
- Finding a longest simple path between two vertices is difficult, however. Merely determining whether a graph contains a simple path with at least a given number of edges is NP-complete.

Variants of shortest path

- **Single-source**: Find shortest paths from a given source vertex $s \in V$ to every vertex $v \in V$. This can be solved with a variation of Dijkstra's algorithm.
- **Single-destination:** Find shortest paths to a given destination vertex. This can be solved with the same algorithm as **Single-source** (just flip the edges of the graph).
- **All-pairs:** Find shortest path from u to v for all $u, v \in V$.

Single-source and **Single-destination** are in the same O class as Dijkstra's algorithm. **All pairs** can be solved more efficiently (but that's for another time!).

Final message

Evaluation of this course

- We are aware of the things that could have gone better (suggestions for improvement are acknowledged and further welcome);
- We hope to have given you a taste of what complexity of algorithms is and why it is important!
- Next year, you can continue with **Analyse van Algoritmen** (very nice course by Hans Zantema & Josef Urban).

Final message

Evaluation of this course

- We are aware of the things that could have gone better (suggestions for improvement are acknowledged and further welcome);
- We hope to have given you a taste of what complexity of algorithms is and why it is important!
- Next year, you can continue with **Analyse van Algoritmen** (very nice course by Hans Zantema & Josef Urban).

- Next week this will be a *questions' class*
- I will show what the *toetsmatrix* looks like for the exam

The end