Dynamic Programming & Optimal Control

Lecture 8
Shortest Path Algorithms

Fall 2023

Prof. Raffaello D'Andrea

ETH Zurich

Learning Objectives

Topic: Shortest Path Algorithms

Objectives

- You understand and are able to apply the *Label Correcting Algorithm*.
- You know the *different methods* to remove nodes from the OPEN bin (e.g. depth first, breadth first, best first, ...) and their *differences*.
- You understand and are able to apply the *A* Algorithm*.

Outline

Label correcting methods (1/4)

Now, we look at more efficient approaches to solve shortest path (SP) problems.

In SP problems, we are interested in reaching a destination node from a starting node in the shortest way possible.

- The Dynamic Programming Algorithm (DPA) approach solves the shortest path problem *from every node* to the destination node.
- In the Label Correcting Algorithm (LCA), we progressively discover paths from S to other nodes $i \in \mathcal{V}$ and keep track of the associated lengths to get to i in a variable d_i, called the *label* of i.

  In LCA we also keep updating a set, called OPEN, which is the current list of nodes that could potentially be a part of the shortest path to T.

We assume[1] that $c_{i,j} \geq 0$ for all $(i, j, c_{i,j}) \in \mathcal{C}$, which ensures that Assumption 7.1 (no negative-cycles) is satisfied.

[1] We delve into this assumption in Problem Set 3.

Label correcting methods (2/4)

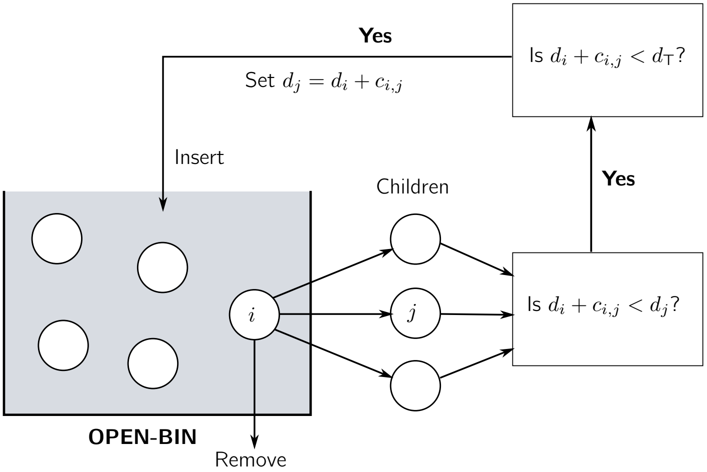

Yes
Is $d_i + c_{i,j} < d_\mathsf{T}$?
Set $d_j = d_i + c_{i,j}$
Insert
Yes
Children
i
j
Is $d_i + c_{i,j} < d_j$?
OPEN-BIN
Remove

Figure: Label correcting algorithm. A node j is called a child of node i if there is an edge (i, j) connecting i with j.

Label correcting methods (3/4)

- **STEP 0:** Place node S in OPEN, set $d_{\mathsf{S}} = 0$ and $d_j = \infty \ \forall j \in \mathcal{V} \backslash \{\mathsf{S}\}$.

- **STEP 1:** Remove a node i from OPEN and execute step 2 for all children j of i (that is, for all nodes $j \in \mathcal{V}$ with $c_{i,j} < \infty$).

- **STEP 2:** If $(d_i + c_{i,j}) < d_j$ *and* $(d_i + c_{i,j}) < d_{\mathsf{T}}$, set $d_j = d_i + c_{i,j}$ and set i to be the parent of j in the path from S to T. If $j \neq \mathsf{T}$, put j in OPEN (it may be already there).

- **STEP 3:** If OPEN is empty, we are done. Else go to **STEP 1**.

Label correcting methods (4/4)

Theorem 8.1: Label correcting algorithm

If at least one finite cost path from S to T exists, the Label Correcting Algorithm terminates with $d_{\mathsf{T}} = J_{Q^*}$. Otherwise, the LCA terminates with $d_{\mathsf{T}} = \infty$.

Proof of LCA (1/3)

The LCA always terminates: each time a node j enters the OPEN list, its label is decreased and becomes equal to the length of some path from s to j.

The number of distinct lengths of paths from s to j that are smaller than any given number is finite (since the graph does not contain negative cycles).

Thus there can only be a finite number of label reductions for each node j and hence, since the LCA removes nodes in Step 1, the algorithm will eventually terminate.

Proof of LCA (2/3)

A node $i \in \mathcal{V}$ is in OPEN only if there is a finite cost path from S to i.

Suppose there is no finite cost path from S to T. Then, for any node i in OPEN, $c_{i,\mathsf{T}}$ cannot be finite, because if it were, then there would be a finite cost path from S to T.

By Step 2, d_T is never updated and remains at ∞.

Proof of LCA (3/3)

Let $Q^* = (\mathsf{S}, i_1, i_2, \ldots, i_{q-2}, \mathsf{T}) \in \mathbb{Q}_{\mathsf{S},\mathsf{T}}$ be a shortest path from S to T with length J_{Q^*}.

Note that by the principle of optimality, $Q^*_m = (\mathsf{S}, i_1, \ldots, i_m)$ is the shortest path from S to $i_m,\ m = 1, \ldots, q-2$, with length $J_{Q^*_m}$.

Assume for the sake of contradiction that upon termination, $d_\mathsf{T} > J_{Q^*}$. Then, $d_\mathsf{T} > J_{Q^*_m}, m = 1, \ldots, q-2$ throughout the algorithm (as d_T only decreases in the algorithm and every arc length is non-negative).

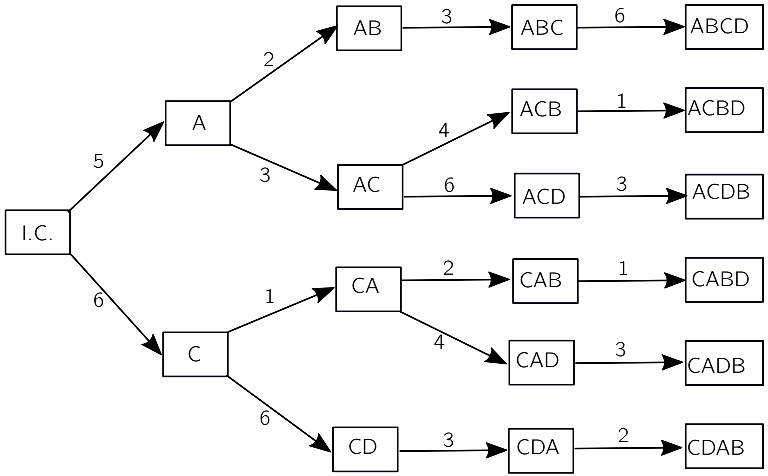
This implies that i_{q-2} does not enter OPEN with $d_{i_{q-2}} = J_{Q^*_{q-2}}$ since if it did, then the next time i_{q-2} is removed from OPEN, d_T would be updated to J_{Q^*}.

Similarly, i_{q-3} will not enter OPEN with $d_{i_{q-3}} = J_{Q^*_{q-3}}$.

Continuing this way, i_1 will not enter OPEN with $d_{i_1} = J_{Q^*_1} = c_{\mathsf{S},i_1}$. But this happens at the first iteration of the algorithm, which leads to a contradiction. Hence, $d_\mathsf{T} = J_{Q^*}$ upon completion of the algorithm.

Example: Deterministic Scheduling Problem (revisited)

Consider a deterministic scheduling problem, where 4 operations A, B, C, D are used to produce a certain product. Operation A must occur before B, and C before D. Between each two operations there is a transition cost.



Example: Deterministic Scheduling Problem

We can map the deterministic finite-state system depicted to a shortest path problem on which we can apply the label correcting algorithm. Before we do so, we can simplify the state transition diagram in order to reduce the number of nodes.

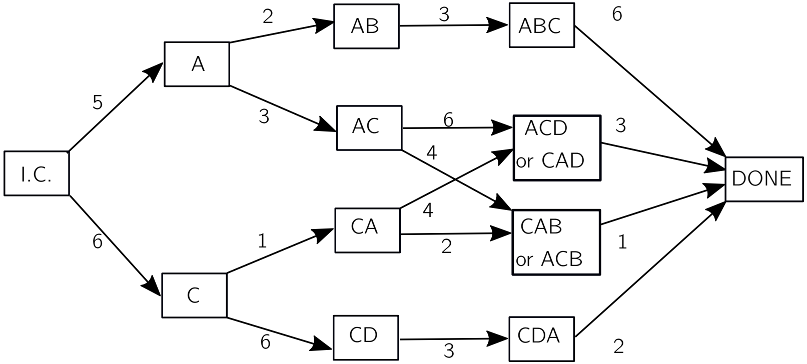


Figure: Simplified state transition diagram.

Example: Deterministic Scheduling Problem

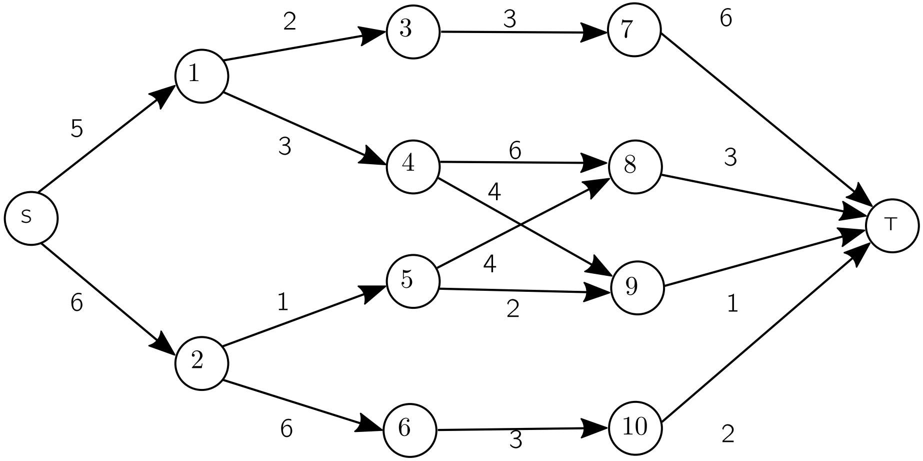
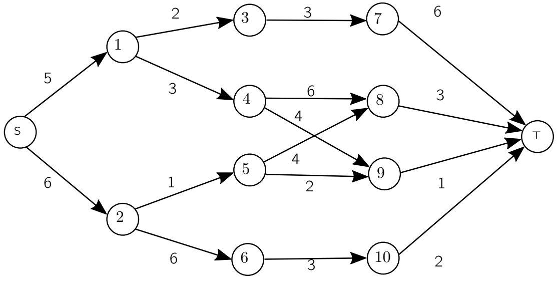


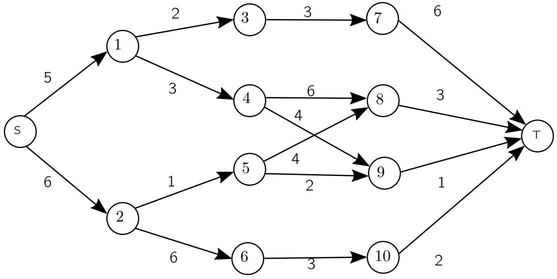
Figure: Deterministic scheduling problem with enumerated nodes.

Example: Deterministic Scheduling Problem



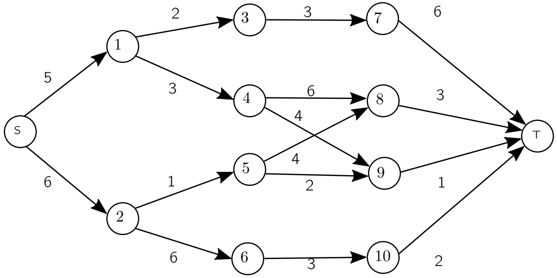
Iteration	Remove	OPEN	d_S	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_T
0	–	S	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	S	1,2	0	5	6	∞	∞	∞	∞	∞	∞	∞	∞	∞

Example: Deterministic Scheduling Problem



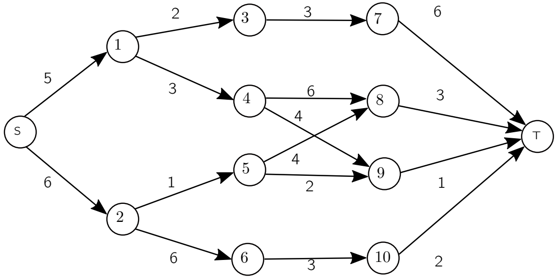
Iteration	Remove	OPEN	d_S	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_T
0	–	S	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	S	1,2	0	5	6	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	2	1,5,6	0	5	6	∞	∞	7	12	∞	∞	∞	∞	∞

Example: Deterministic Scheduling Problem



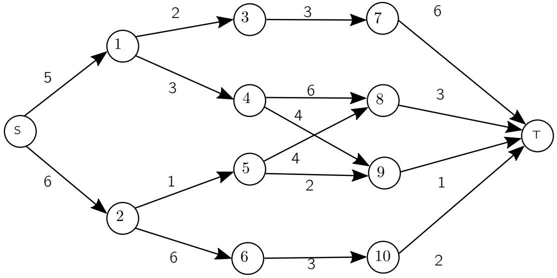
Iteration	Remove	OPEN	d_S	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_T
0	–	S	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	S	1,2	0	5	6	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	2	1,5,6	0	5	6	∞	∞	7	12	∞	∞	∞	∞	∞
3	6	1,5,10	0	5	6	∞	∞	7	12	∞	∞	∞	15	∞

Example: Deterministic Scheduling Problem



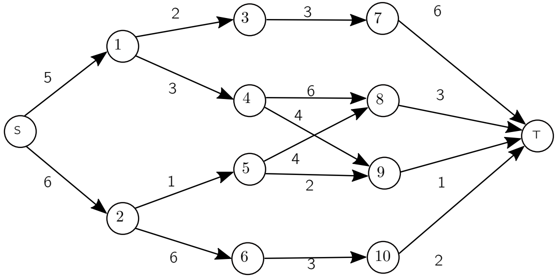
Iteration	Remove	OPEN	d_S	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_T
0	–	S	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	S	1,2	0	5	6	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	2	1,5,6	0	5	6	∞	∞	7	12	∞	∞	∞	∞	∞
3	6	1,5,10	0	5	6	∞	∞	7	12	∞	∞	∞	15	∞
4	10	1,5	0	5	6	∞	∞	7	12	∞	∞	∞	15	17

Example: Deterministic Scheduling Problem



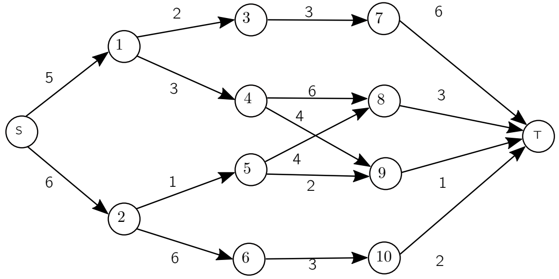
Iteration	Remove	OPEN	d_S	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_T
0	–	S	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	S	1,2	0	5	6	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	2	1,5,6	0	5	6	∞	∞	7	12	∞	∞	∞	∞	∞
3	6	1,5,10	0	5	6	∞	∞	7	12	∞	∞	∞	15	∞
4	10	1,5	0	5	6	∞	∞	7	12	∞	∞	∞	15	17
5	5	1,8,9	0	5	6	∞	∞	7	12	∞	11	9	15	17

Example: Deterministic Scheduling Problem



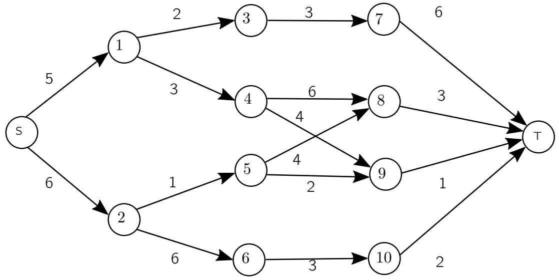
Iteration	Remove	OPEN	d_S	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_T
0	–	S	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	S	1,2	0	5	6	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	2	1,5,6	0	5	6	∞	∞	7	12	∞	∞	∞	∞	∞
3	6	1,5,10	0	5	6	∞	∞	7	12	∞	∞	∞	15	∞
4	10	1,5	0	5	6	∞	∞	7	12	∞	∞	∞	15	17
5	5	1,8,9	0	5	6	∞	∞	7	12	∞	11	9	15	17
6	9	1,8	0	5	6	∞	∞	7	12	∞	11	9	15	10

Example: Deterministic Scheduling Problem



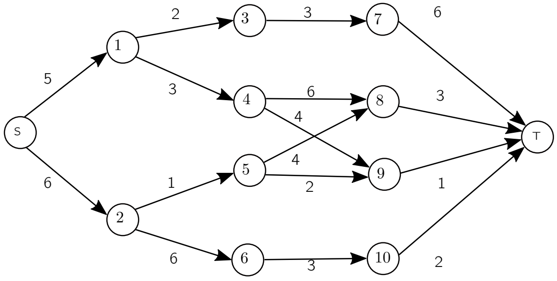
Iteration	Remove	OPEN	d_S	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_T
0	–	S	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	S	1,2	0	5	6	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	2	1,5,6	0	5	6	∞	∞	7	12	∞	∞	∞	∞	∞
3	6	1,5,10	0	5	6	∞	∞	7	12	∞	∞	∞	15	∞
4	10	1,5	0	5	6	∞	∞	7	12	∞	∞	∞	15	17
5	5	1,8,9	0	5	6	∞	∞	7	12	∞	11	9	15	17
6	9	1,8	0	5	6	∞	∞	7	12	∞	11	9	15	10
7	8	1	0	5	6	∞	∞	7	12	∞	11	9	15	10

Example: Deterministic Scheduling Problem



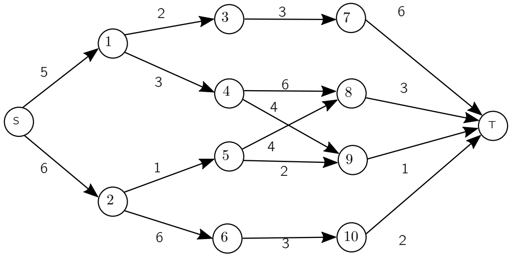
Iteration	Remove	OPEN	d_S	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_T
0	–	S	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	S	1,2	0	5	6	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	2	1,5,6	0	5	6	∞	∞	7	12	∞	∞	∞	∞	∞
3	6	1,5,10	0	5	6	∞	∞	7	12	∞	∞	∞	15	∞
4	10	1,5	0	5	6	∞	∞	7	12	∞	∞	∞	15	17
5	5	1,8,9	0	5	6	∞	∞	7	12	∞	11	9	15	17
6	9	1,8	0	5	6	∞	∞	7	12	∞	11	9	15	10
7	8	1	0	5	6	∞	∞	7	12	∞	11	9	15	10
8	1	3,4	0	5	6	7	8	7	12	∞	11	9	15	10

Example: Deterministic Scheduling Problem



Iteration	Remove	OPEN	d_S	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_T
0	–	S	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	S	1,2	0	5	6	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	2	1,5,6	0	5	6	∞	∞	7	12	∞	∞	∞	∞	∞
3	6	1,5,10	0	5	6	∞	∞	7	12	∞	∞	∞	15	∞
4	10	1,5	0	5	6	∞	∞	7	12	∞	∞	∞	15	17
5	5	1,8,9	0	5	6	∞	∞	7	12	∞	11	9	15	17
6	9	1,8	0	5	6	∞	∞	7	12	∞	11	9	15	10
7	8	1	0	5	6	∞	∞	7	12	∞	11	9	15	10
8	1	3,4	0	5	6	7	8	7	12	∞	11	9	15	10
9	4	3	0	5	6	7	8	7	12	∞	11	9	15	10

Example: Deterministic Scheduling Problem



Iteration	Remove	OPEN	d_S	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_T
0	–	S	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	S	1,2	0	5	6	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	2	1,5,6	0	5	6	∞	∞	7	12	∞	∞	∞	∞	∞
3	6	1,5,10	0	5	6	∞	∞	7	12	∞	∞	∞	15	∞
4	10	1,5	0	5	6	∞	∞	7	12	∞	∞	∞	15	17
5	5	1,8,9	0	5	6	∞	∞	7	12	∞	11	9	15	17
6	9	1,8	0	5	6	∞	∞	7	12	∞	11	9	15	10
7	8	1	0	5	6	∞	∞	7	12	∞	11	9	15	10
8	1	3,4	0	5	6	7	8	7	12	∞	11	9	15	10
9	4	3	0	5	6	7	8	7	12	∞	11	9	15	10
10	3	–	0	5	6	7	8	7	12	∞	11	9	15	10

Example: Deterministic Scheduling Problem

Iteration	Remove	OPEN	d_S	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_T
0	–	S	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	**S**	1,2	0	5	**6**	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	**2**	1,5,6	0	5	6	∞	∞	**7**	12	∞	∞	∞	∞	∞
3	6	1,5,10	0	5	6	∞	∞	7	12	∞	∞	∞	15	∞
4	10	1,5	0	5	6	∞	∞	7	12	∞	∞	∞	15	17
5	**5**	1,8,9	0	5	6	∞	∞	7	12	∞	11	**9**	15	17
6	**9**	1,8	0	5	6	∞	∞	7	12	∞	11	9	15	**10**
7	8	1	0	5	6	∞	∞	7	12	∞	11	9	15	10
8	1	3,4	0	5	6	7	8	7	12	∞	11	9	15	10
9	4	3	0	5	6	7	8	7	12	∞	11	9	15	10
10	3	–	0	5	6	7	8	7	12	∞	11	9	15	10

Keeping track of the parent nodes when a child node is added to open, it can be determined that the optimal path is $(S, 2, 5, 9, T)$, which corresponds to the state evolution of (C,CA,CAB,CABD), and has a total cost of 10.

How to remove nodes from OPEN?

The proof of Theorem 8.1 does not depend on the order in which nodes exit OPEN. In principle, any method works but some are more efficient.

- **Depth-First Search** or "last in, first out" strategy, that is, a node is always removed from the top of the OPEN bin and each node entering OPEN is placed at the top. It is often implemented as a stack.
- **Breadth-First Search** or "first in, first out" strategy; that is, a node is always removed from the top of the OPEN bin and each node entering OPEN is placed at the bottom. It is often implemented as a queue.
- **Best-First Search (Dijkstra's Algorithm):** At each iteration, the node that is removed from OPEN is the node i^* where

  $$d_{i^*} = \min_{i \in \text{OPEN}} d_i,$$

  that is, remove the node that currently has the best label. It is often implemented as a priority queue (min-heap).

A*-algorithm

This algorithm is a modification to the LCA: in Step 2, we strengthen the requirement of a node j being admitted to OPEN from

$$d_i + c_{i,j} < d_{\mathsf{T}}$$

to

$$d_i + c_{i,j} + h_j < d_{\mathsf{T}}$$

where h_j is some positive lower bound on the cost to get from node j to T.

This lower bound can be constructed depending on special knowledge about the problem. This more stringent criterion can reduce the number of iterations required by the LCA.

Some examples:

- Straight-line distance for planning minimum distance routes.
- Straight-line distance divided by maximum speed for planning minimum time routes.

Outline

Shortest Path Algorithms

- Label Correcting Methods
- Additional reading material

Additional reading material

We strongly encourage you to implement all of these algorithms from scratch. Although there are plenty of implementations in the Internet, getting your hands dirty helps with:

1. understanding the details, strengths, and pitfalls of these algorithms and deciding which one to use based on the problem at hand; and
2. mastering the key ideas (of algorithms and implementations), which increases your problem solving abilities.

Some pointers:

1. Some useful data structures: Stack, Queue, Priority Queue, Fibonacci Heap.
2. How do you assess the complexity of a computer algorithm? Keywords: Asymptotic notation and analysis.