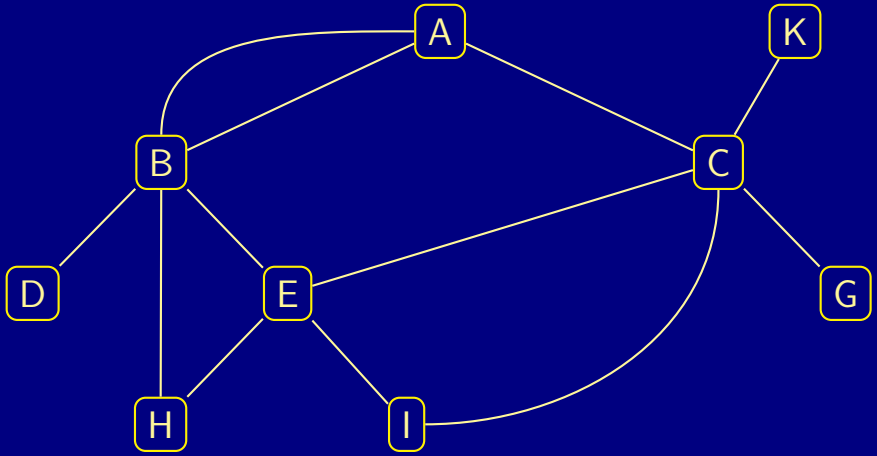
Dijkstra

Johan Montelius

KTH

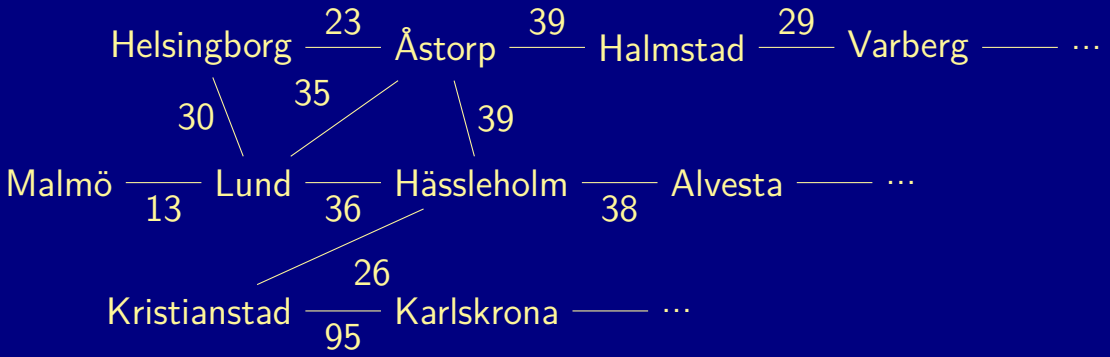
HT22

undirectional graph

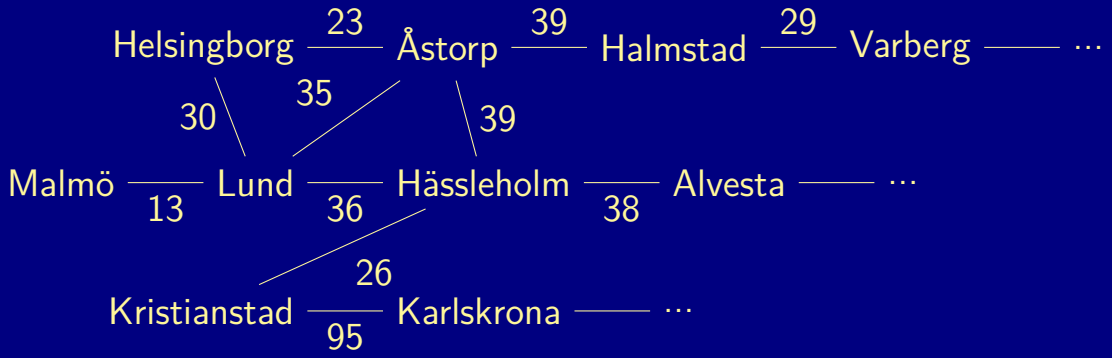


trains in Sweden

trains in Sweden

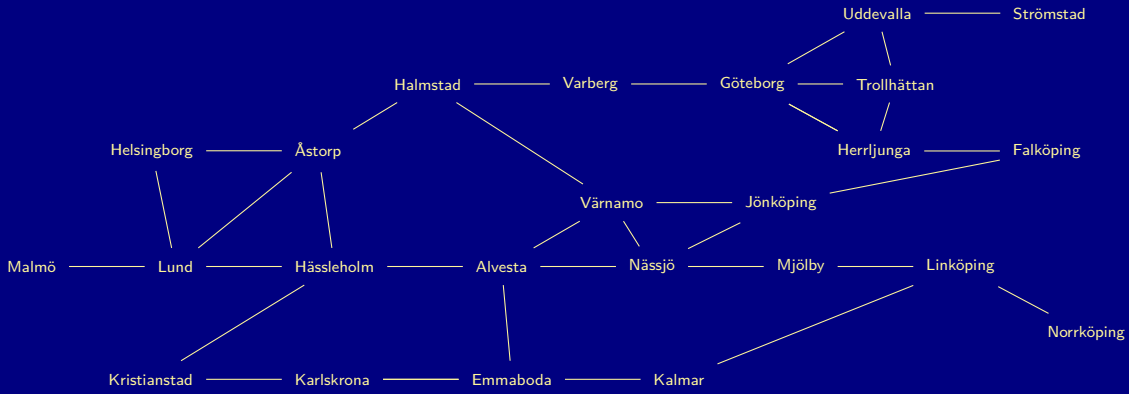


trains in Sweden



A weighted graph.

trains in Sweden

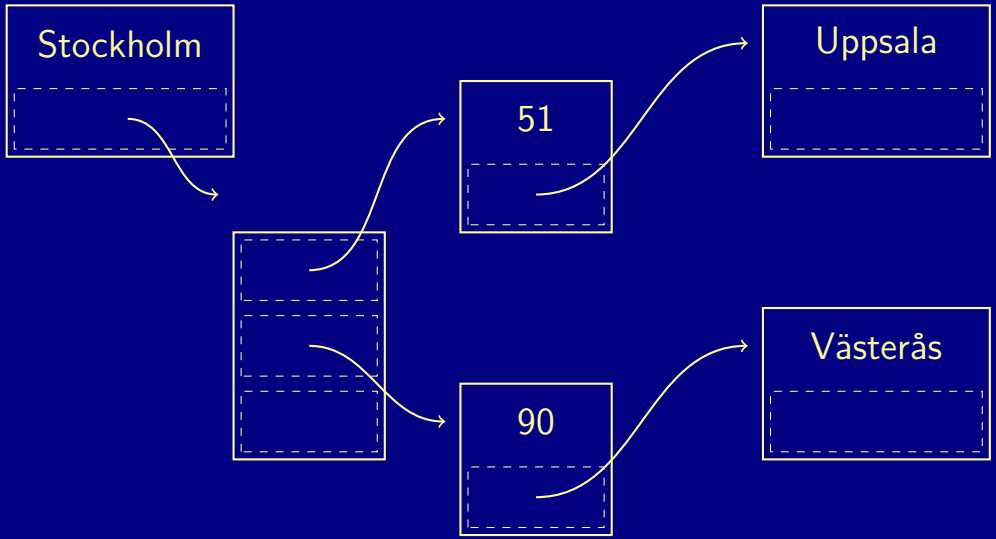


represent the graph

```
public class City {
  String name;
  Connection[] neigbours;
     :
}
```

```
public class Connection {
  City city;
  Integer distance;
     :
}
```

the graph



the naive solution

What is the shortest path from Malmö to Stockholm?

the naive solution

What is the shortest path from Malmö to Stockholm?
Set a maximum distance of the path.

the naive solution

What is the shortest path from Malmö to Stockholm?
Set a maximum distance of the path.

- Do we have more time left?

the naive solution

What is the shortest path from Malmö to Stockholm?
Set a maximum distance of the path.

- Do we have more time left?
- Are we at the destination?

the naive solution

What is the shortest path from Malmö to Stockholm?
Set a maximum distance of the path.

- Do we have more time left?
- Are we at the destination?
- For each of the direct connected cites:

the naive solution

What is the shortest path from Malmö to Stockholm?
Set a maximum distance of the path.

- Do we have more time left?
- Are we at the destination?
- For each of the direct connected cites:
  - set the maximum distance allowed and

the naive solution

What is the shortest path from Malmö to Stockholm?
Set a maximum distance of the path.

- Do we have more time left?
- Are we at the destination?
- For each of the direct connected cites:
  - set the maximum distance allowed and
  - find the shortest path from the city to the destination.

the naive solution

What is the shortest path from Malmö to Stockholm?
Set a maximum distance of the path.

- Do we have more time left?
- Are we at the destination?
- For each of the direct connected cites:
  - set the maximum distance allowed and
  - find the shortest path from the city to the destination.
- Return the shortest distance found (or null).

the naive solution

What is the shortest path from Malmö to Stockholm?
Set a maximum distance of the path.

- Do we have more time left?
- Are we at the destination?
- For each of the direct connected cites:
  - set the maximum distance allowed and
  - find the shortest path from the city to the destination.
- Return the shortest distance found (or null).

the path solution

the path solution

As before but keep a trail of cities that you have past.

the path solution

As before but keep a trail of cities that you have past.

- Do we have more time left?

the path solution

As before but keep a trail of cities that you have past.

- Do we have more time left?
- Are we at the destination?

the path solution

As before but keep a trail of cities that you have past.

- Do we have more time left?
- Are we at the destination?
- Is the city in the path?

the path solution

As before but keep a trail of cities that you have past.

- Do we have more time left?
- Are we at the destination?
- Is the city in the path?

  :

improvement

improvement

If you have found a path with a distance d,

improvement

If you have found a path with a distance d,
then any other path should be shorter than the found.

bounded depth first search

bounded depth first search

If we had a DAG - we would be able to use a depth first search directly (might not be efficient but it works),

bounded depth first search

If we had a DAG - we would be able to use a depth first search directly (might not be efficient but it works),

Since we have cirkular paths we need a trick to avoid looping.

bounded depth first search

If we had a DAG - we would be able to use a depth first search directly (might not be efficient but it works),

Since we have cirkular paths we need a trick to avoid looping.

Set a maximum "time traveled".

bounded depth first search

If we had a DAG - we would be able to use a depth first search directly (might not be efficient but it works),

Since we have cirkular paths we need a trick to avoid looping.

Set a maximum "time traveled".

Works for railroads - does it work for all weighted graphs?

iterative deepening

iterative deepening

Do a bounded depth first search - if no path is found, try increasing the bound.

iterative deepening

Do a bounded depth first search - if no path is found, try increasing the bound.

iterative deepening

Do a bounded depth first search - if no path is found, try increasing the bound.

Strategy - start with bound 30 min, if not found increase to 60 min, if not found increas to 90 min ....

iterative deepening

Do a bounded depth first search - if no path is found, try increasing the bound.

Strategy - start with bound 30 min, if not found increase to 60 min, if not found increas to 90 min ....

If the minimum path is found at 300 min, how much time have we wasted.

avoid circular paths

avoid circular paths

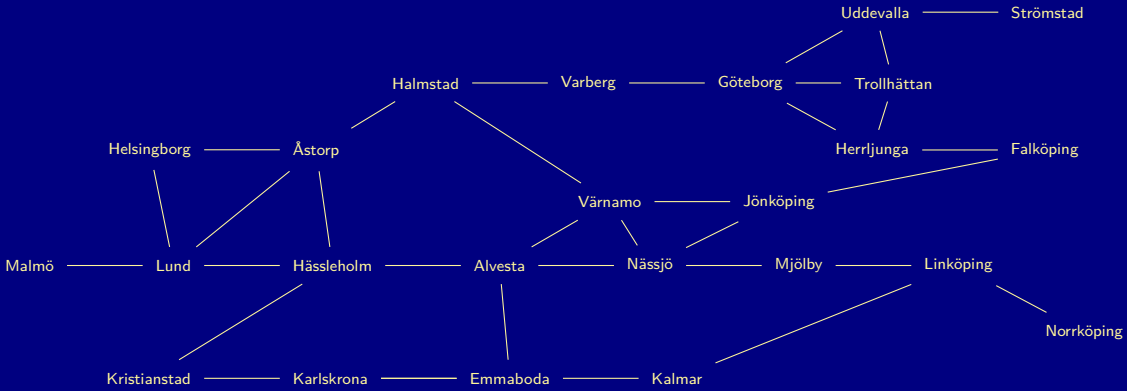
More bookkeeping - keep a trail of cities and don't go back, or

avoid circular paths

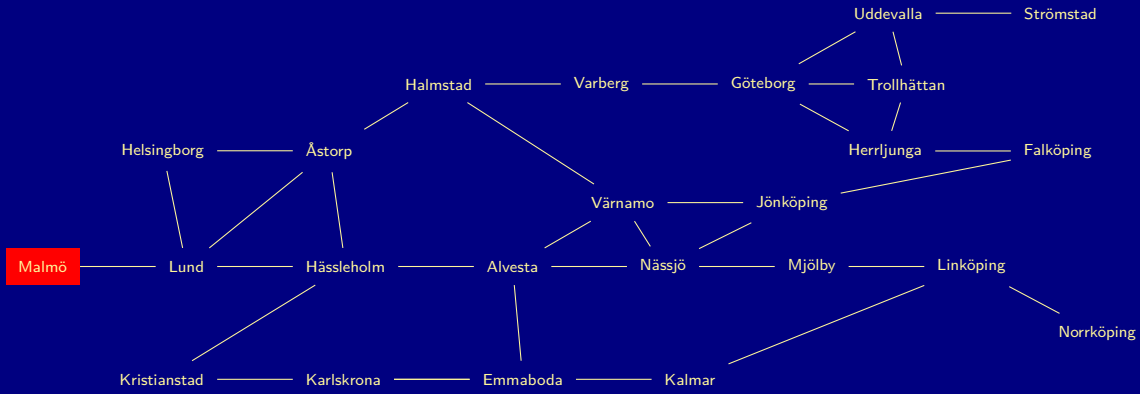
More bookkeeping - keep a trail of cities and don't go back, or

... hava a party!

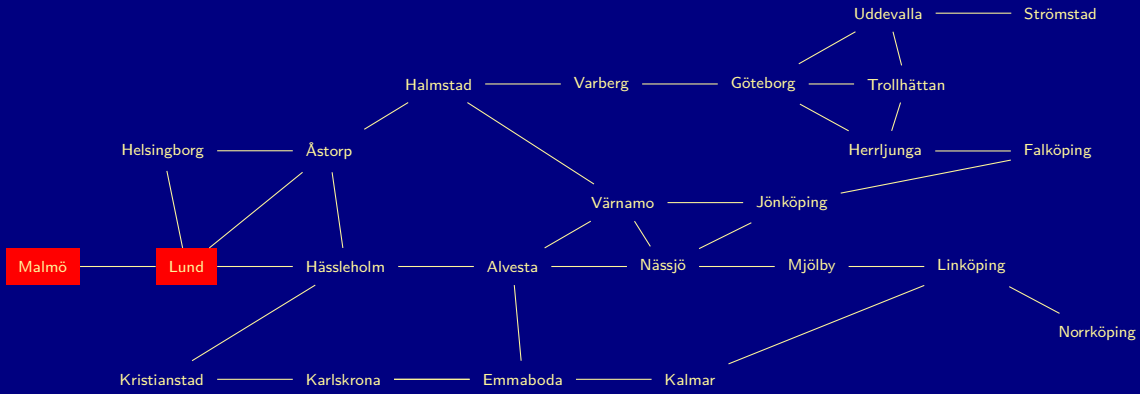
paint the town red



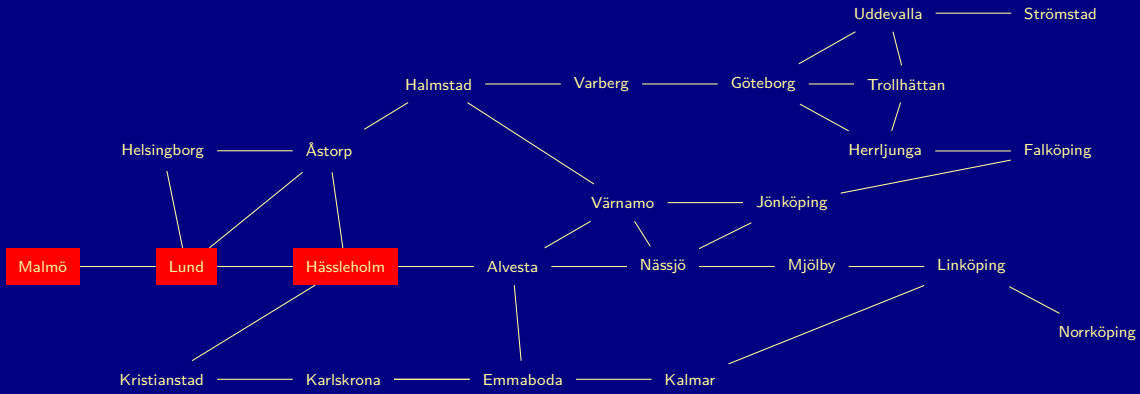
paint the town red



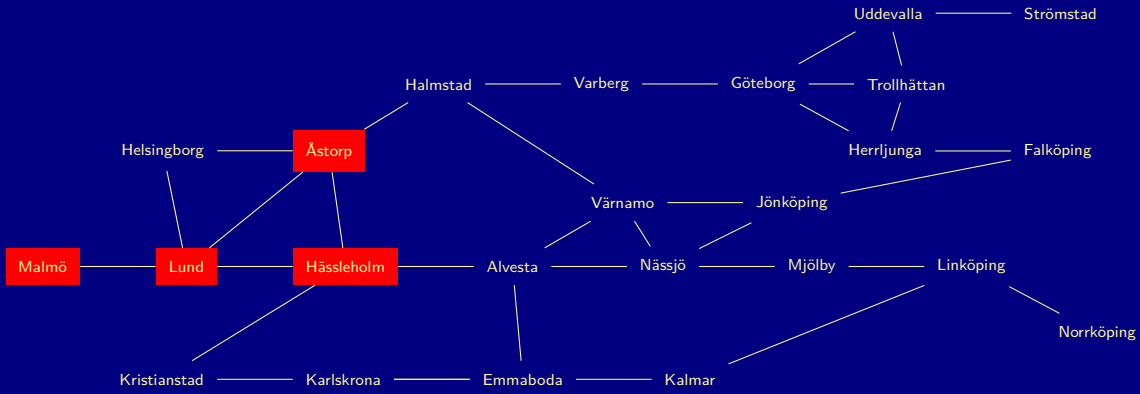
paint the town red



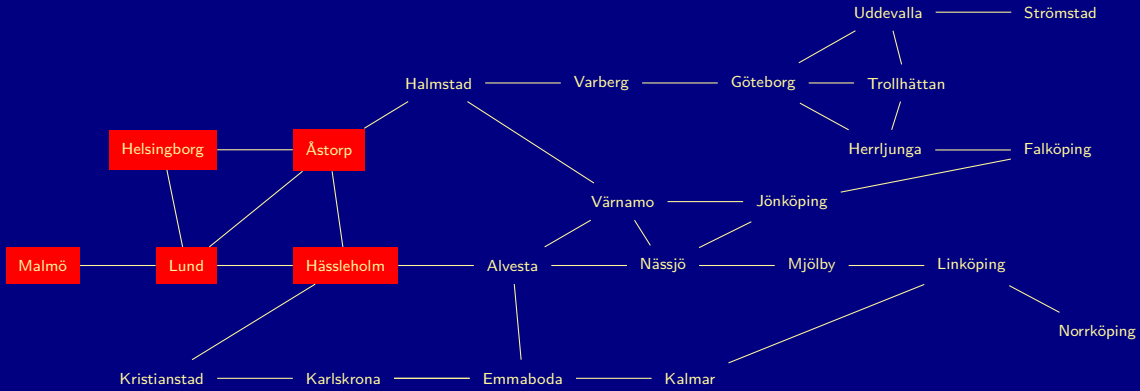
paint the town red



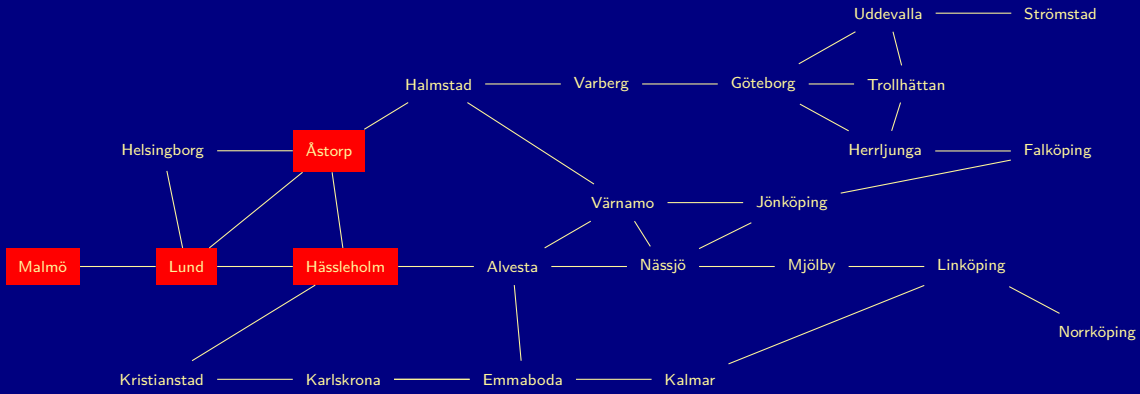
paint the town red



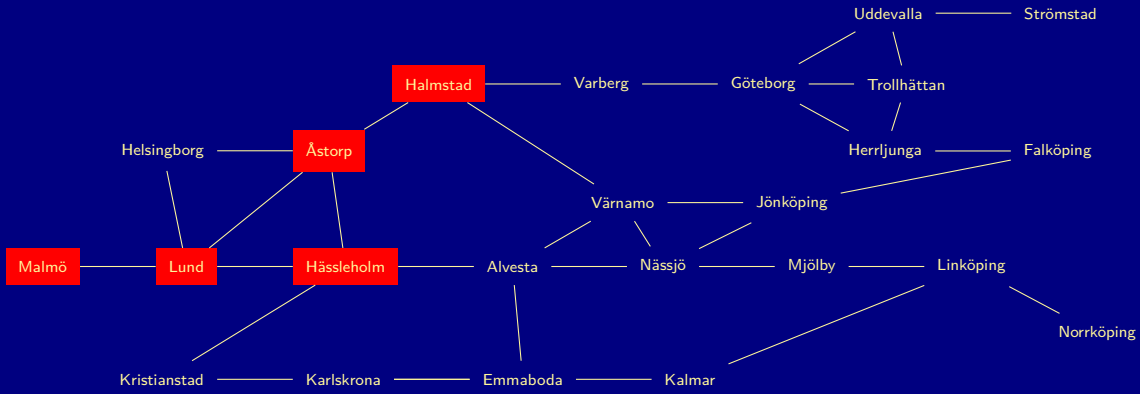
paint the town red



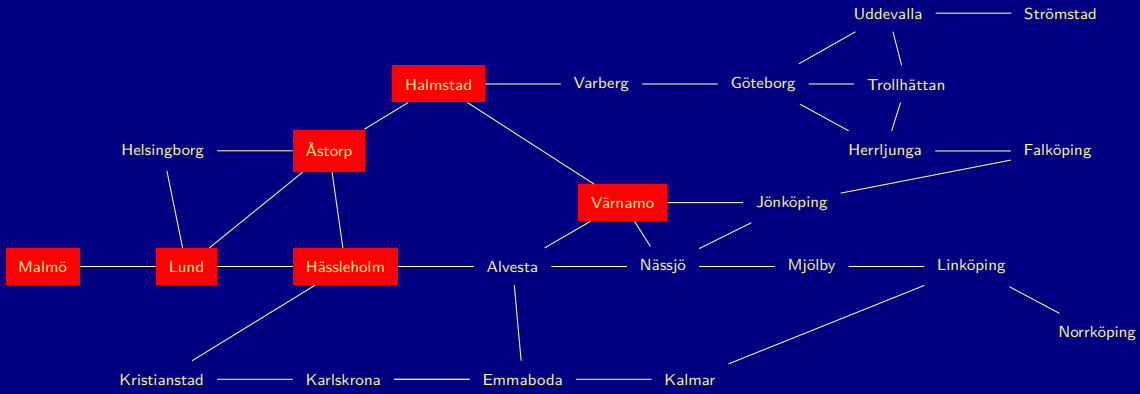
paint the town red



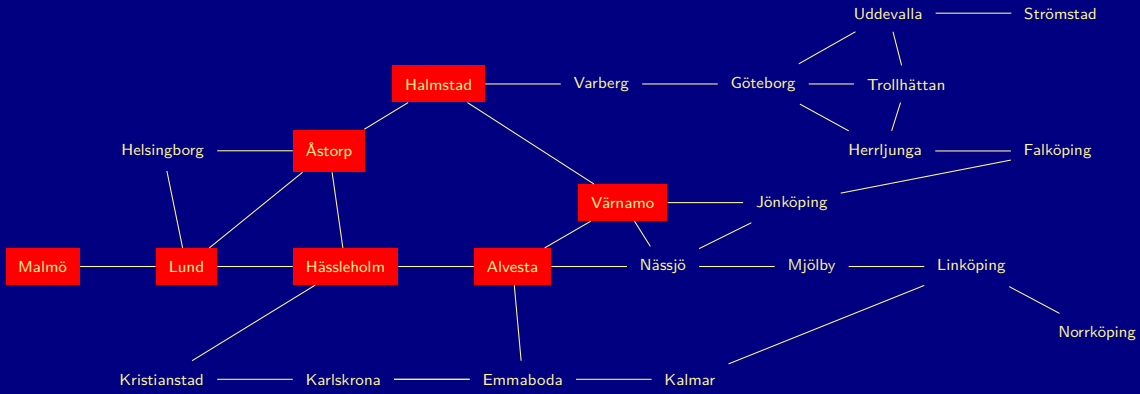
paint the town red



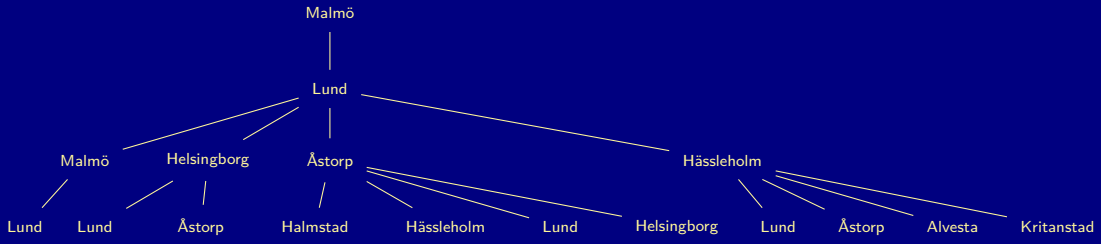
paint the town red



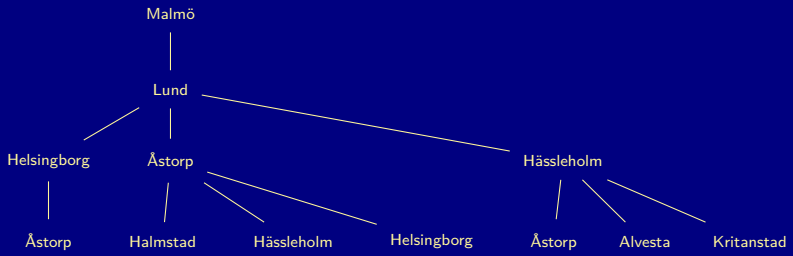
paint the town red



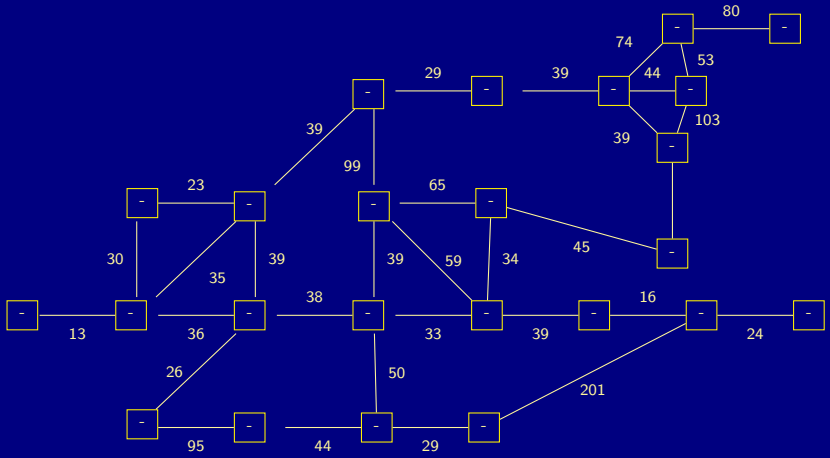
a search tree



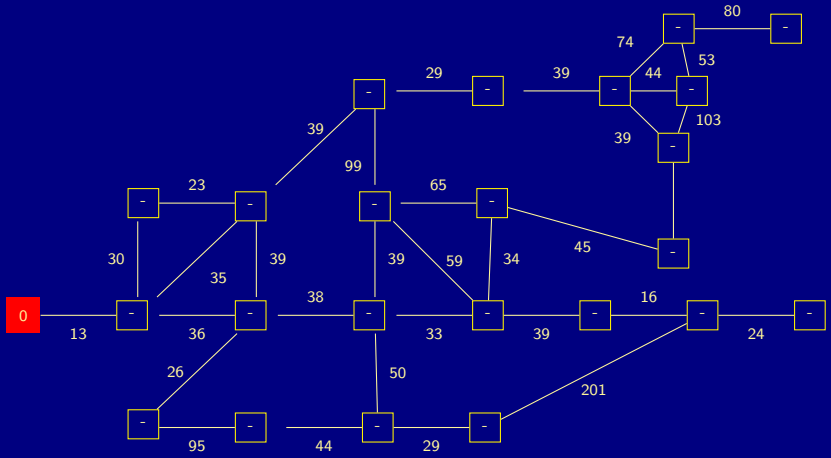
avoiding circular paths



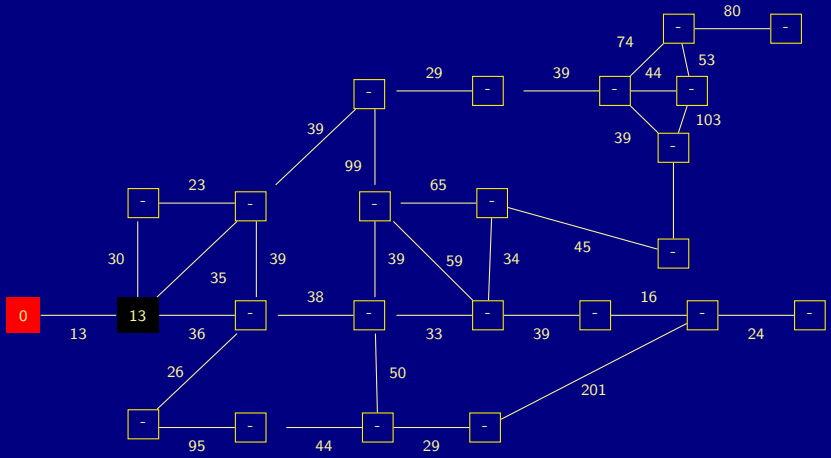
move slowly forward



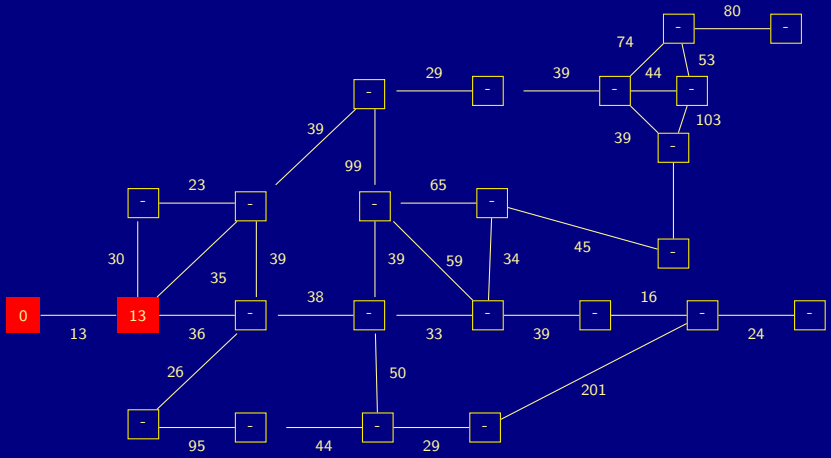
move slowly forward



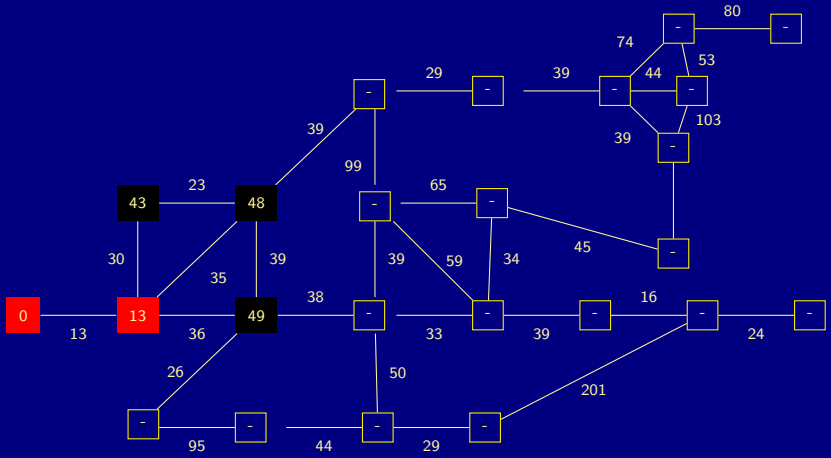
move slowly forward



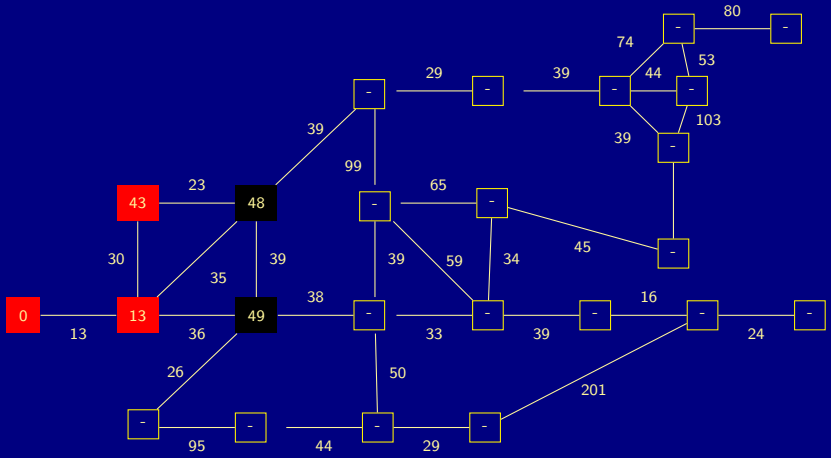
move slowly forward



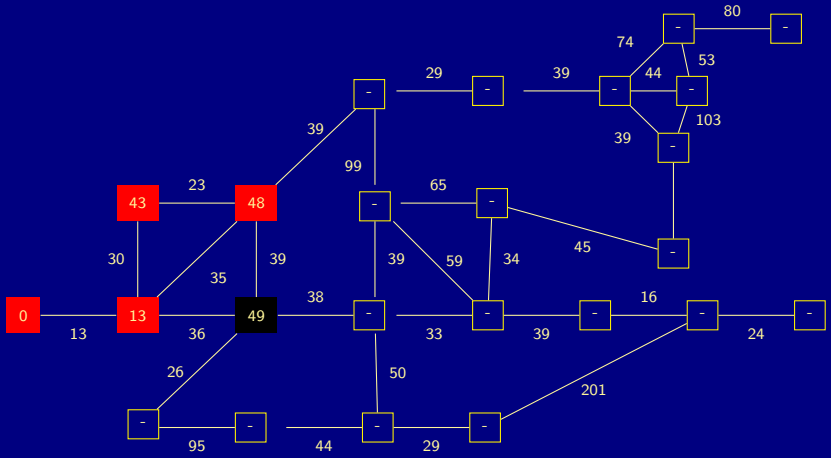
move slowly forward



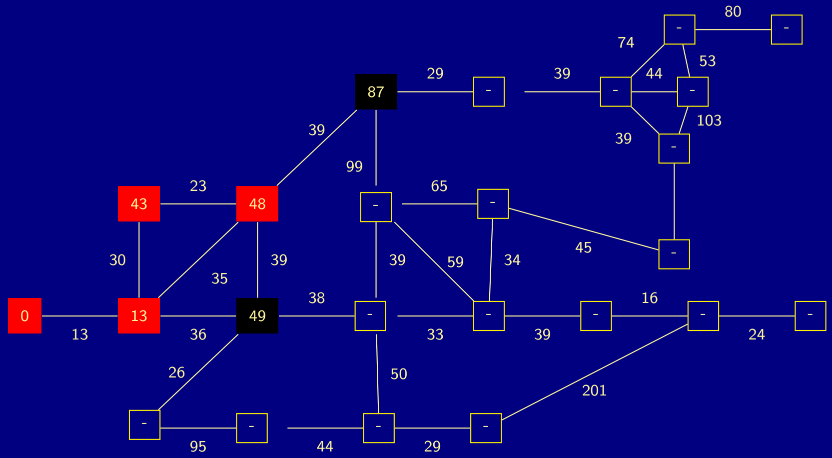
move slowly forward



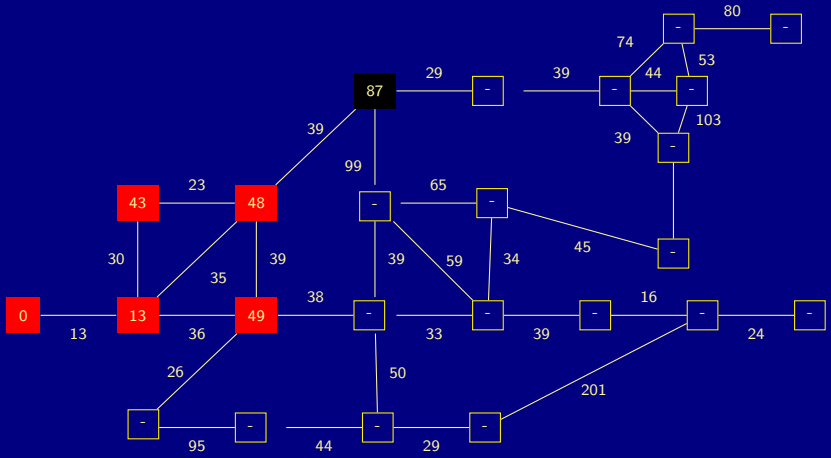
move slowly forward



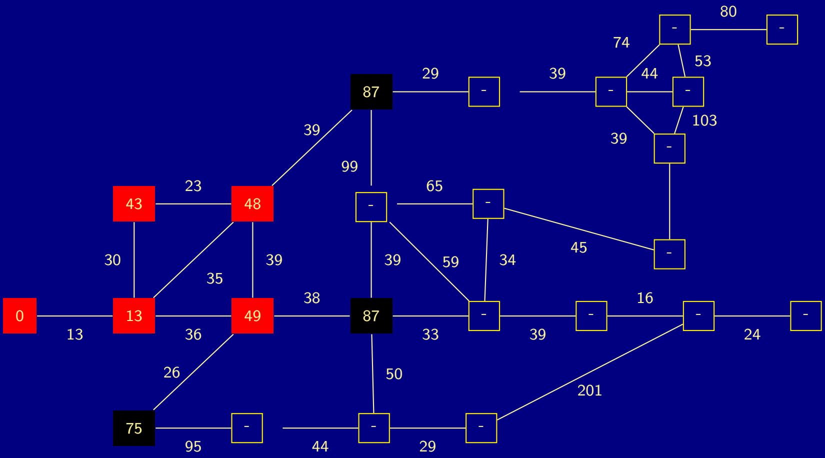
move slowly forward



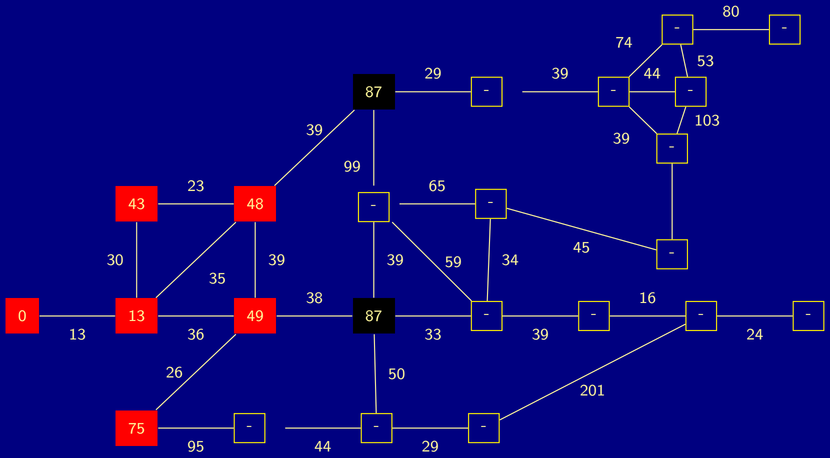
move slowly forward



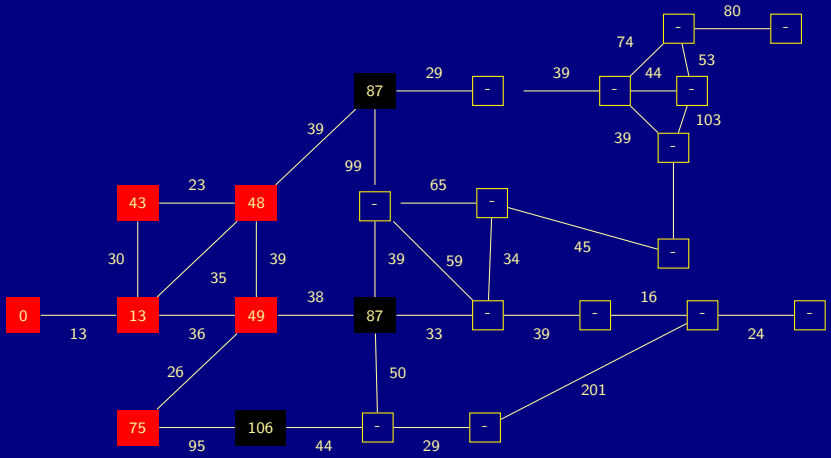
move slowly forward



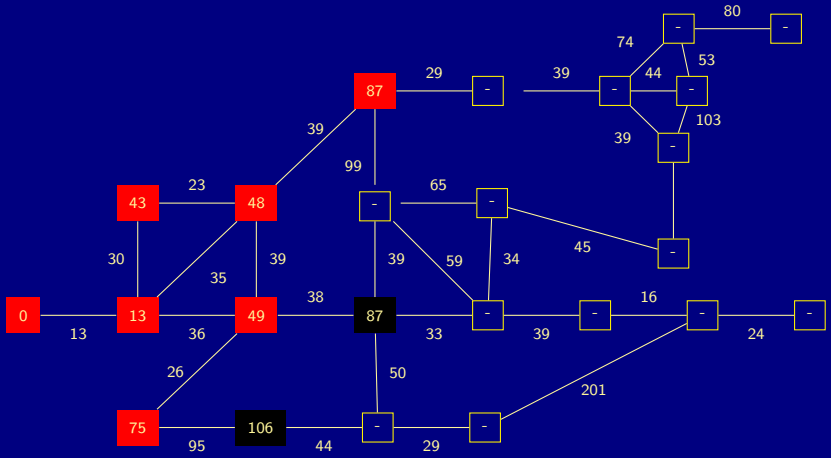
move slowly forward



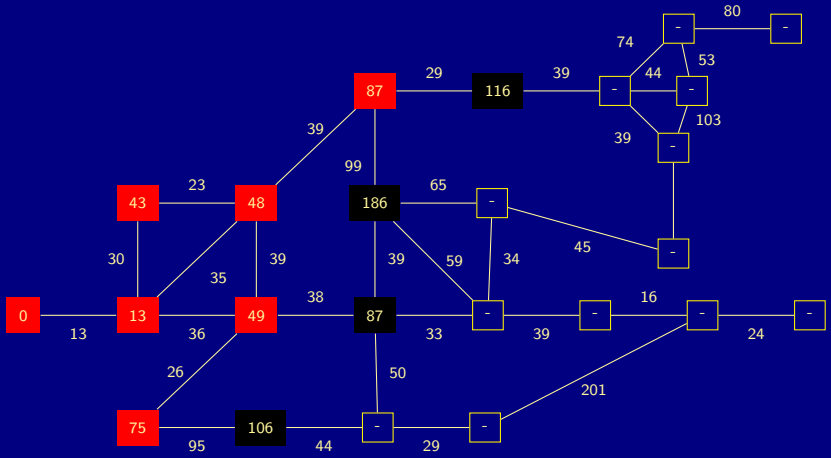
move slowly forward



move slowly forward

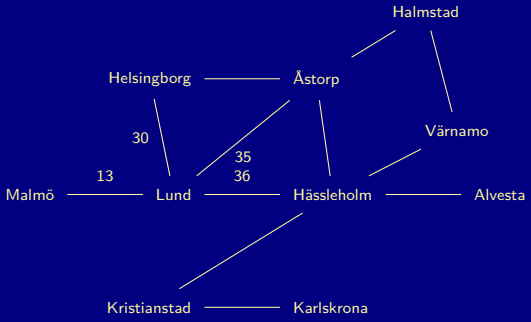


move slowly forward



Dijkstra's shortest path

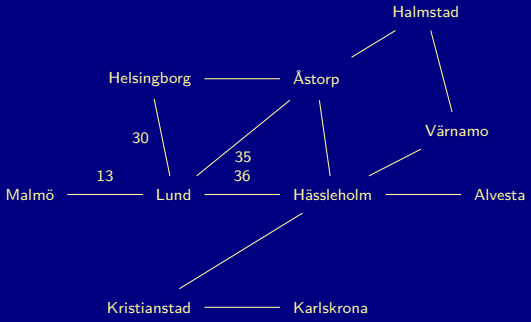
Shortest:



Queue

Dijkstra's shortest path

Shortest:

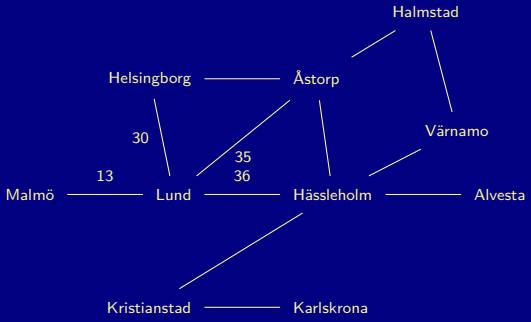


Queue

Malmö-0

Dijkstra's shortest path

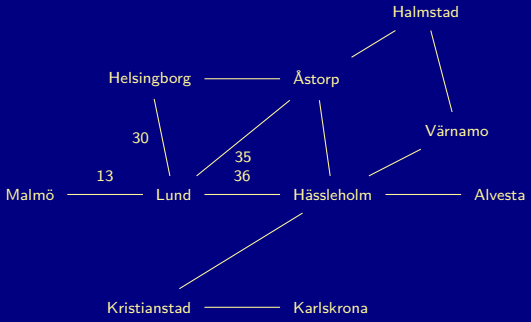
Shortest: Malmö-0



Queue

Dijkstra's shortest path

Shortest: Malmö-0

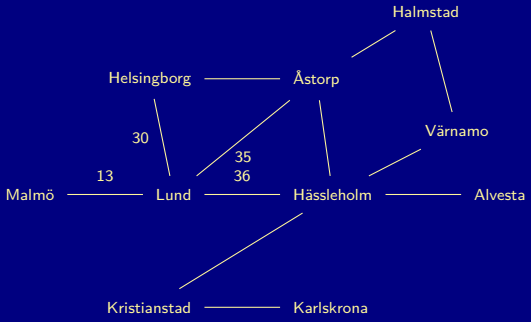


Queue

Lund-13

Dijkstra's shortest path

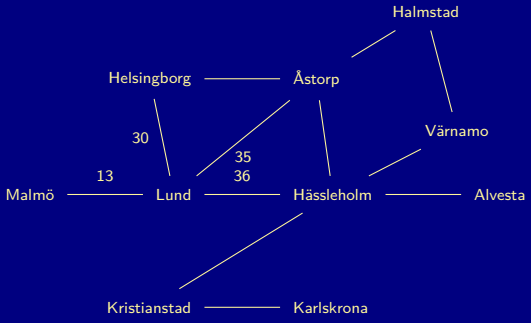
Shortest: Malmö-0 Lund-13



Queue

Dijkstra's shortest path

Shortest: Malmö-0 Lund-13



Queue

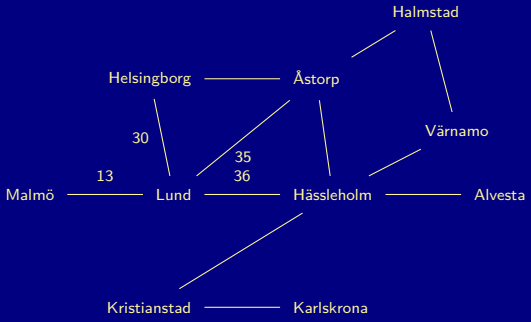
Helsingborg-43

Åstorp-48

Hässleholm-49

Dijkstra's shortest path

Shortest: Malmö-0 Lund-13 Helsingborg-43



Queue

Åstorp-48

Hässleholm-49

the algorithm - invariants

the algorithm - invariants

- Cities in the "shortest found" list are done, there is no shorter path.

the algorithm - invariants

- Cities in the "shortest found" list are done, there is no shorter path.
- If a city is in the front of the queue we have found the shortes distance to the city.

the algorithm - the operation

the algorithm - the operation

Place the source city in the queue with distance 0.

the algorithm - the operation

Place the source city in the queue with distance 0.

- Remove the first city from the queue and

the algorithm - the operation

Place the source city in the queue with distance 0.

- Remove the first city from the queue and
- place it in the "shortest found" list.

the algorithm - the operation

Place the source city in the queue with distance 0.

- Remove the first city from the queue and
- place it in the "shortest found" list.
- For each of the connections of the city:
- update, or add, the queue entry of that city.

the algorithm - the operation

Place the source city in the queue with distance 0.

- Remove the first city from the queue and
- place it in the "shortest found" list.
- For each of the connections of the city:
- update, or add, the queue entry of that city.

Could it be that we vilolate the invariants?

shortest path

shortest path

So you found the shortest distanc, where is the path?

runtime complexity

- In each iteration, one city moves from the queue to the "shortest found" list.

runtime complexity

- In each iteration, one city moves from the queue to the "shortest found" list.
- When all cities are in the found list, we are done.

runtime complexity

- In each iteration, one city moves from the queue to the "shortest found" list.
- When all cities are in the found list, we are done.
- When a city is moved, then:

runtime complexity

- In each iteration, one city moves from the queue to the "shortest found" list.
- When all cities are in the found list, we are done.
- When a city is moved, then:
  - find connecting cities and

runtime complexity

- In each iteration, one city moves from the queue to the "shortest found" list.
- When all cities are in the found list, we are done.
- When a city is moved, then:
  - find connecting cities and
  - add cities to queue if not in the found list.

runtime complexity

- In each iteration, one city moves from the queue to the "shortest found" list.
- When all cities are in the found list, we are done.
- When a city is moved, then:
  - find connecting cities and
  - add cities to queue if not in the found list.