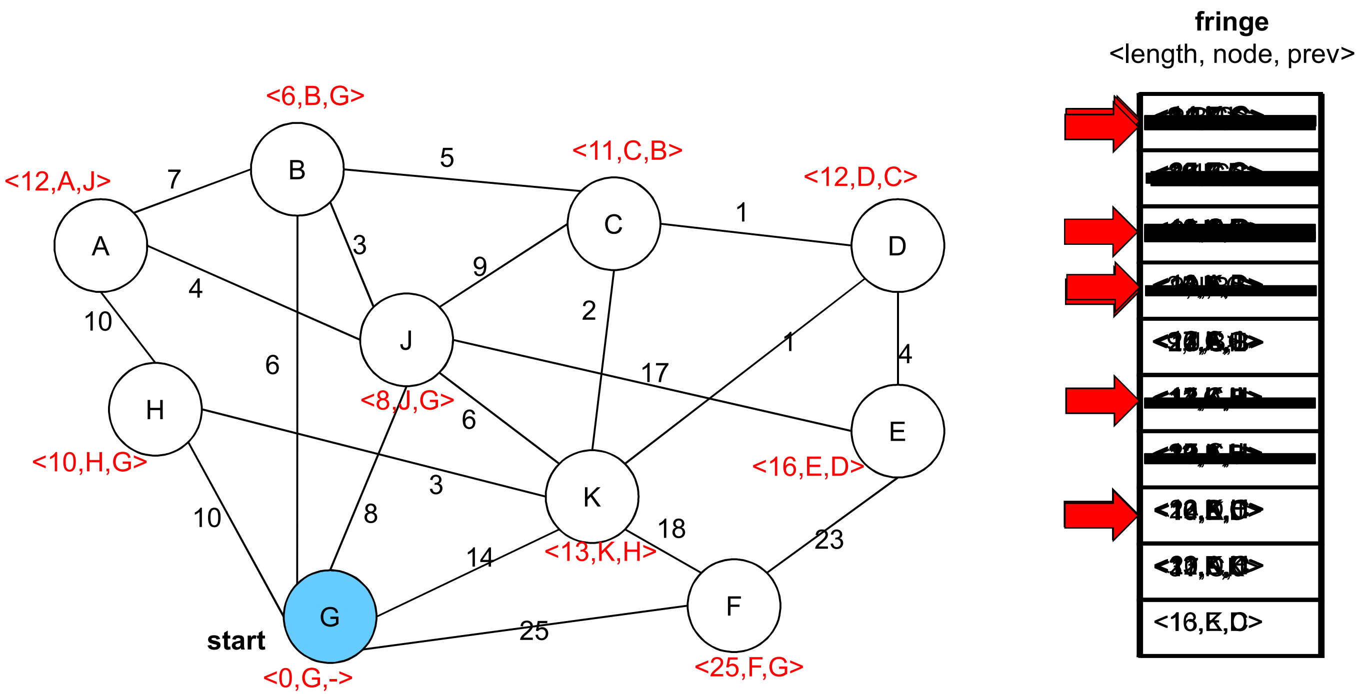
Example of Dijkstra's Algorithm



Visited:

Backpointers: <dist, node, prev>

What's the cost of Dijkstra's algorithm?

If a graph has N nodes and E edges:

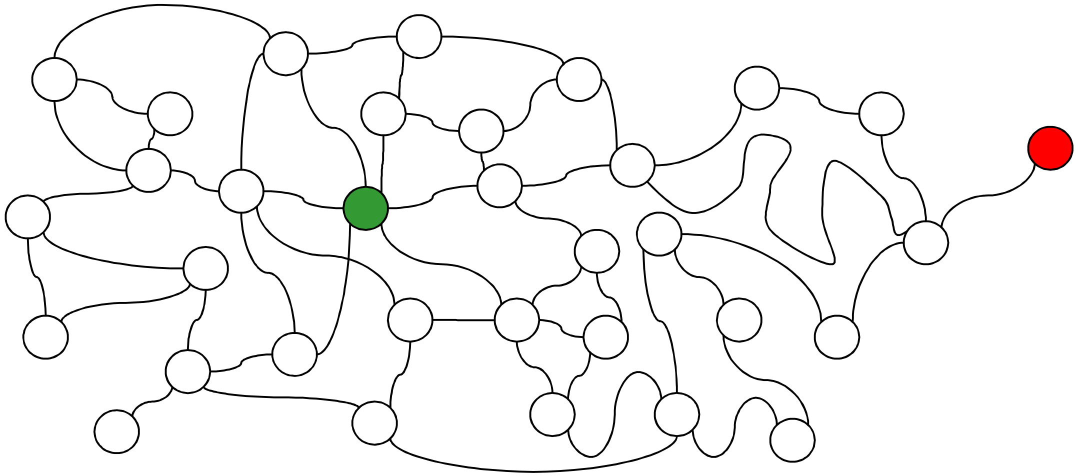
Identify the most expensive line:

```
while  fringe is not empty:
      :
   for each edge out of node to a neighbour:
      :
         add ⟨neighbour, edge,  length-to-neighbour⟩ to fringe
```

How many times might we do that line?
What is the cost of that line?

Problem with Dijkstra's Algorithm

- If we want all shortest paths: Dijkstra is best.
  - Greedy: never backtracks and every iteration adds a path to the answer
- If we want the shortest path to a goal: Dijkstra is wasteful:
  - spends time building paths to useless nodes, not on the way to the goal:
- Need to combine:
  - length of path to here, AND
  - estimate of remaining distance
    - Biases the choice towards nodes that are on the way to the goal.



A* Search.

- A* search is the standard good algorithm for finding shortest paths to a goal.

- Nodes on the fringe are ordered by estimated total path length through this node:

$$= \underbrace{\text{length of path from start to this node}}_{\text{same as Djikstra's)}} + \underbrace{\text{estimate of remaining distance.}}_{\text{new: (same as first heuristic)}}$$

- Fringe items must have
  - node,
  - previous node or edge (for the backpointers)
  - distance to node from start (needed to compute the distance to the neighbours)
  - total estimated path length

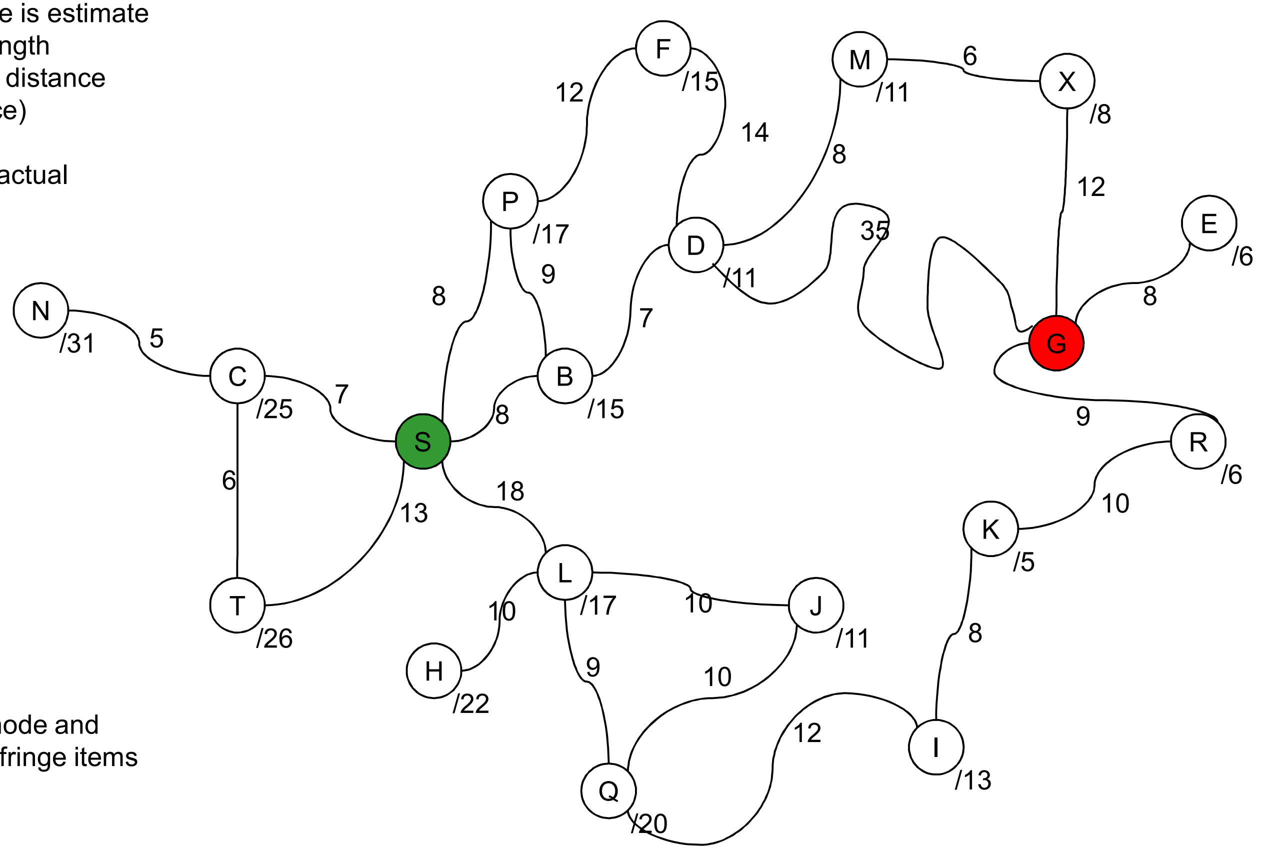
Finding the Shortest Path (A*)

```
FindShortestPath(start, goal):
   fringe ←  PriorityQueue of ⟨node, edge, length-to-node, estimate-total-path⟩  Ordered by estimate
   backpointers ← Map of nodes to edges
   put ⟨start, null, 0, est(start,goal)⟩ on the fringe.                          .
   while fringe is not empty:
      ⟨node, edge, length-to-node, estimate-total-path⟩ ← remove from fringe
      if node is not visited:
         visit node
         put ⟨node, edge⟩ into backpointers
         if node=goal:
            return ReconstructPath(start, goal, backpointers)              // see earlier slide
         for each neigh-edge out of node to a neighbour:
            if neighbour is not visited:
               length-to-neighbour ← length-to-node + neigh-edge.length
               estimate-total-path ← length-to-neighbour + est(neighbour, goal)
               add ⟨neighbour, neigh-edge, length-to-neighbour, estimate-total-path⟩ to fringe
```

A* example.

Number next to node is estimate
of remaining path length
based on Euclidean distance
(straight-line distance)

Number on edge is actual
length of the edge.



Only show path-to-node and
est-total-path in the fringe items

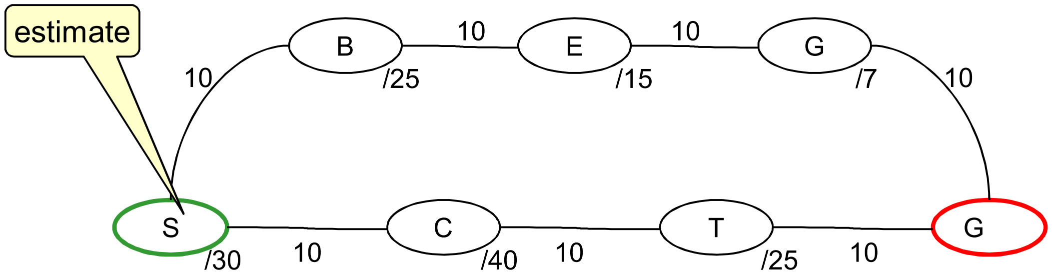
Does A* Search always work?

- The path found for by A* Search is the shortest path from the start node to the goal node if the following conditions are satisfied.
  1. The estimated cost to goal is an underestimate - never greater than the true cost (the heuristic estimate must be "admissible")
  2. When we take a node off the fringe, this must be the shortest path to that node from the start. (the heuristic estimate must be "consistent" or "monotonic")

- If the estimate doesn't satisfy these conditions, the A* algorithm may break.

Admissable heuristics for A*

- A heuristic estimate of the remaining path is admissible if it always underestimates the remaining cost
  - overestimating will cause A* to avoid the path, even though it is actually the best
- If it is not admissible, it may not find the shortest path:



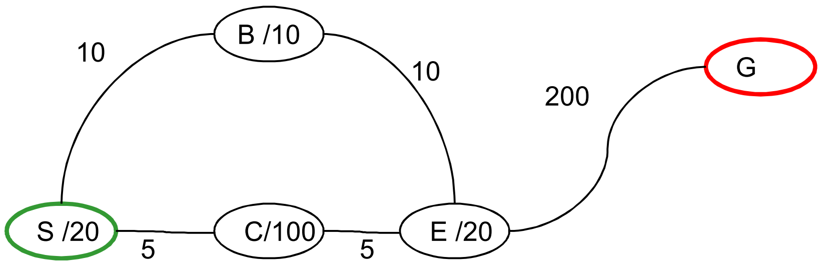
Fringe:

Monotonic/Consistent heuristic for A*

- Admissible is not enough for A*:

  When we visit a node, must be the best path to a node

  Fringe:

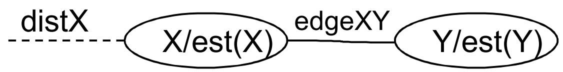


There is a slower, more complicated version of A* that doesn't require a consistent heuristic

change est(E) to 120

- To be able to commit to visited nodes:
  - estimated path length must not get less accurate as you get closer to the goal

dist-to-X + est(X) ≤ dist-to-X + edge-X-Y + est(Y)



- Consistent heuristic:

  est(X) - est(Y) ≤ edge-X-Y

A* heuristic

- Consistent heuristics can be hard to find
  (Euclidean distance to goal *is* consistent)

- If the estimate is admissible, but is not consistent, then:
  - ⇒ cannot commit to a node when we take it off the queue
    - ⇒ may need to revisit nodes
    - ⇒ no point in the visited set

Summary

- A* Search is more effective than Dijkstra's algorithm for 1-to-1 pathfinding
- Many real-world applications
  - not just paths: e.g. search for optimal loading of a truck
  - any optimisation problem where build up a solution as a series of steps, and the cost of the solution is the sum of the costs of the steps.

- Conditions for success
  - Admissible heuristic: never overestimate
  - Consistent/Monotonic heuristic: f=g+h is monotonically non-decreasing
  - The key is to design heuristic function to meet the conditions