The A* algorithm and the shortest path problem

1. **Background and example**
2. **A* algortihm**
3. **Remarks**

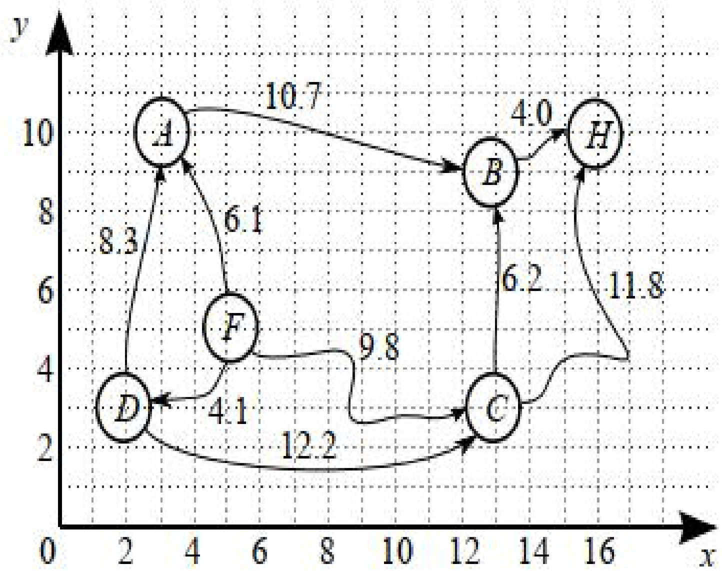
1. Background and example

Starting point: a directed weighted graph $G = (V, E)$

Goal: given a starting node s (source) and a target node t, find shortest path from s to t, if such a path exists

- for edge (x, y), weight $w((x, y))$ is interpreted as a distance
- sometimes called simply shortest path problem (or single pair shortest path problem)
- assume non-negative weights

Example



Shortest path from s = F to t = H?

□

A modified Dijkstra's algorithm can be used to solve the single-pair shortest path problem.

```
DIJKSTRAPAIR(s, t, G)
starting from source node s finds the shortest path to a
target node t given weighted graph G

forEach node x in G
  x.color = white, x.d = ∞, x.π =NIL
end

/* Give source node appropriate values. */
s.color = gray, s.d = 0
▷ initialize a priority queue Q
INSERT(Q,s,0)
while Q is not empty
  x = EXTRACT-MIN(Q)
  /* Check if target node has been found or not. */
  if x == t then
     return
  end

  forEach node y in x.Adj
     y.old = y.d, RELAX(x, y)
     if y.color == white then
        /* Node y is undiscovered. */
        y.color = gray, y.π = x
        INSERT(Q,y,y.d)
     else
        if y.d < y.old and y.color ≠ black then
          /* Take into account that y.old < y.d. */
          INSERT(Q,y,y.d)
        end
     end
  end
  x.color = black
end
```

```
RELAX(x, y)
When shorter path to y is found using edge (x,y), length y.d
and parent y.π are reset.

if y.d > x.d + w((x,y)) then
/* Shorter path than current path found via edge (x,y). */
    y.d = x.d + w((x,y)), y.π = x
end
```

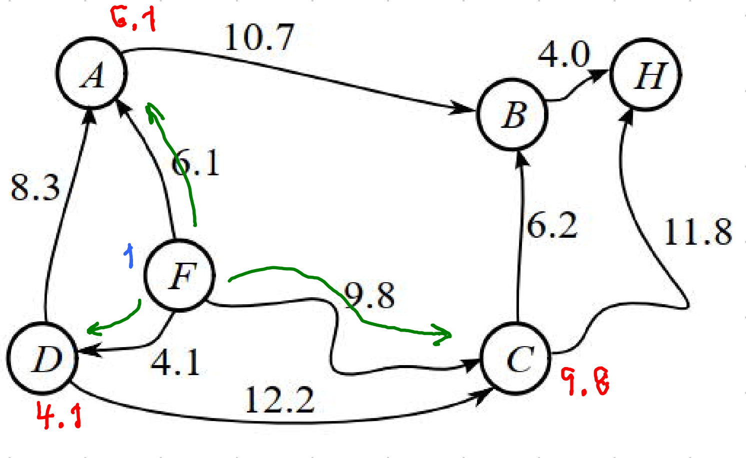
Example (cont'd)

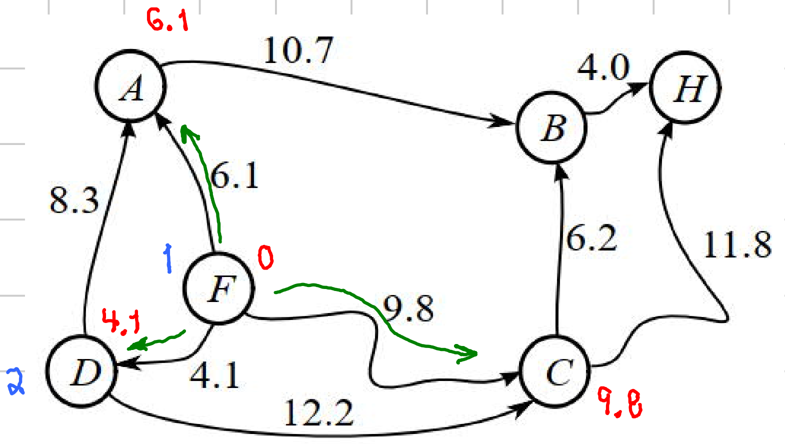
Shortest path from *s* = *F* to *t* = *H* using DIJKSTAPAIR.

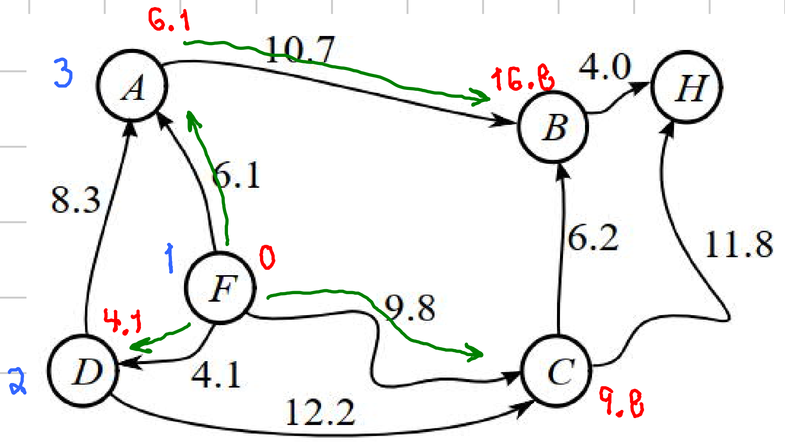
i = order in which node handled

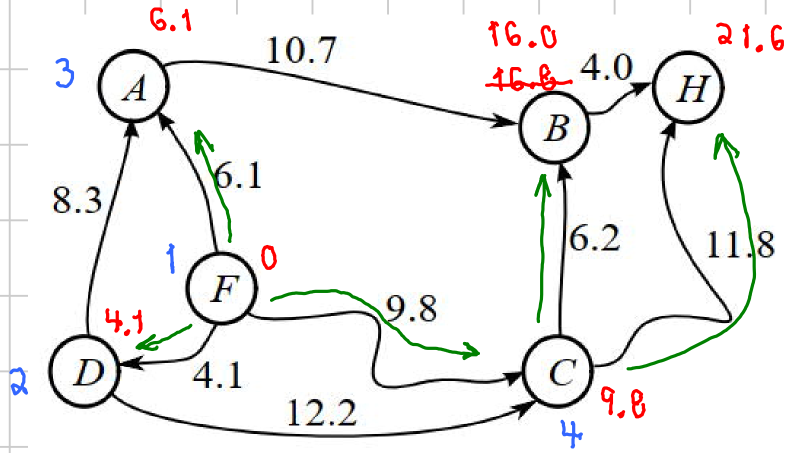
x.d = length of shortest path from s = F to x

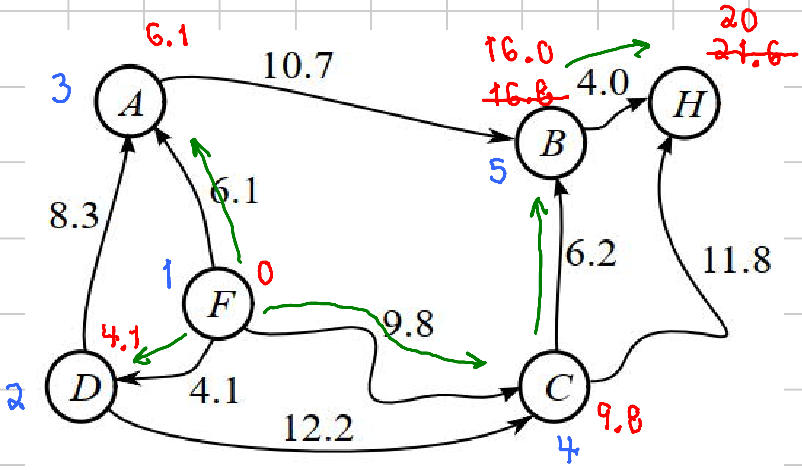
→ = edge of current shortest path tree




A
B
H
F
D
C
10.7
4.0
8.3
6.1
6.2
11.8
9.8
4.1
12.2
6.1
0
1
4.1
2
9.8


A
B
H
F
D
C
10.7
4.0
8.3
6.1
6.2
11.8
9.8
4.1
12.2
6.1
3
16.8
0
1
4.1
2
9.8


A
B
H
F
D
C
10.7
4.0
8.3
6.1
6.2
11.8
9.8
4.1
12.2
6.1
3
16.0
16.8
21.6
0
1
4.1
2
9.8
4



□

Remarks

- Dijkstra often must process nodes that are never included in shortest path
- optimal processing would be that we only ever process nodes that are in shortest path

2. A* algortihm

Dijkstra selects next node x from priority queue based on the length of the shortest path from starting node s to x.

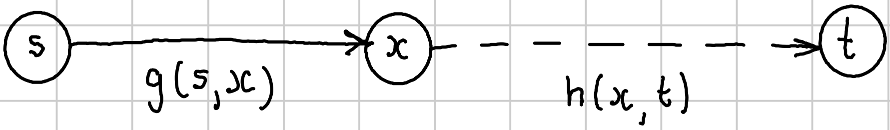
A^* selects next node x based on an estimate $e(x)$ of the length of the path from s to t that goes through x.

$$e(x) = g(s,x) + h(x,t)$$

$$g(s,x) = \text{length of shortest path from source } s \text{ to node } x$$

$$h(x,t) = \text{estimate of length of path from node } x \text{ to target node } t$$

Note: *g(s,x)* is the same as *x.d* in Dijkstra



What do we want from $h(x,t)$?

- Let the actual length of the shortest path from node x to node t be $g((x,t))$. We require $h(x,t) \leq g((x,t))$.
- The function $h(x,t)$ must be easy (fast) to compute.

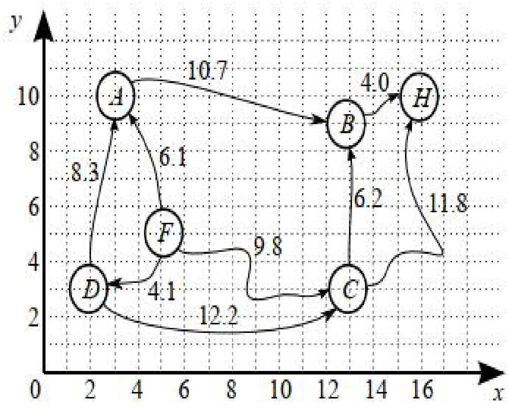
Example (contd)

When x and t are points on an (x,y)-plane with coordinates $x = (\alpha, \beta)$ and $t = (\gamma, \delta)$:

$$h(x,t) = \left((\alpha - \gamma)^2 + (\beta - \delta)^2\right)^{(1/2)}$$

For graph shown, using the coordinates and above Euclidean distance:

(x,H)	(A,H)	(B,H)	(C,H)	(D,H)	(F,H)	(H,H)
$h(x,t)$	13.0	3.2	7.6	15.7	12.1	0.0



□

To perform A^* algorithm, for each node x we need the following.

- $x.d$ length of shortest path x from source s to node x (hence $g(s, x)$ thus far)
- $x.e$ lower bound estimate of length of shortest path from s to target t assuming shortest path goes through x (hence $e(x)$ thus far)
- $x.colour$ = color of node
- $x.\pi$ = parent of node x in shortest path tree
- $x.Adj$ set containing nodes that are adjacent to x

A^* maintains the $x.e$ values of all gray nodes in a priority queue.

```
RELAX-ASTAR(x, y, t)
When shorter path to y is found using edge (x,y), length y.d
and parent y.π are reset. Also update estimate of length of
path to target via node y.

if y.d > x.d + w((x,y)) then
/* Shorter path than current path found via edge (x,y). */
   y.d = x.d + w((x,y)),  y.π = x,  y.e = y.d + h(y,t)
end
```

```
ASTAR(s, t, G)
starting from source node s finds the shortest path to a
target node t given weighted graph G

forEach node x in G
   x.color = white, x.d = ∞, x.π =NIL, x.e = ∞
end

/* Give source node appropriate values. */
s.color = gray, s.d = 0
▷ initialize a priority queue Q
INSERT(Q,s,0)
while Q is not empty
   x = EXTRACT–MIN(Q)

   /* Check if target node has been found or not. */
   if x == t then
      return
   end

   forEach node y in x.Adj
      y.old = y.e, RELAX-ASTAR(x, y, t)
      if y.color == white then
         /* Node y is undiscovered. */
         y.color = gray, y.π = x
         INSERT(Q,y,y.e)
      else
         if y.e < y.old and y.color ≠ black then
            /* Take into account that y.old < y.e. */
            INSERT(Q,y,y.e)
         end
      end
   end
   x.color = black
end
```

Example (contd)

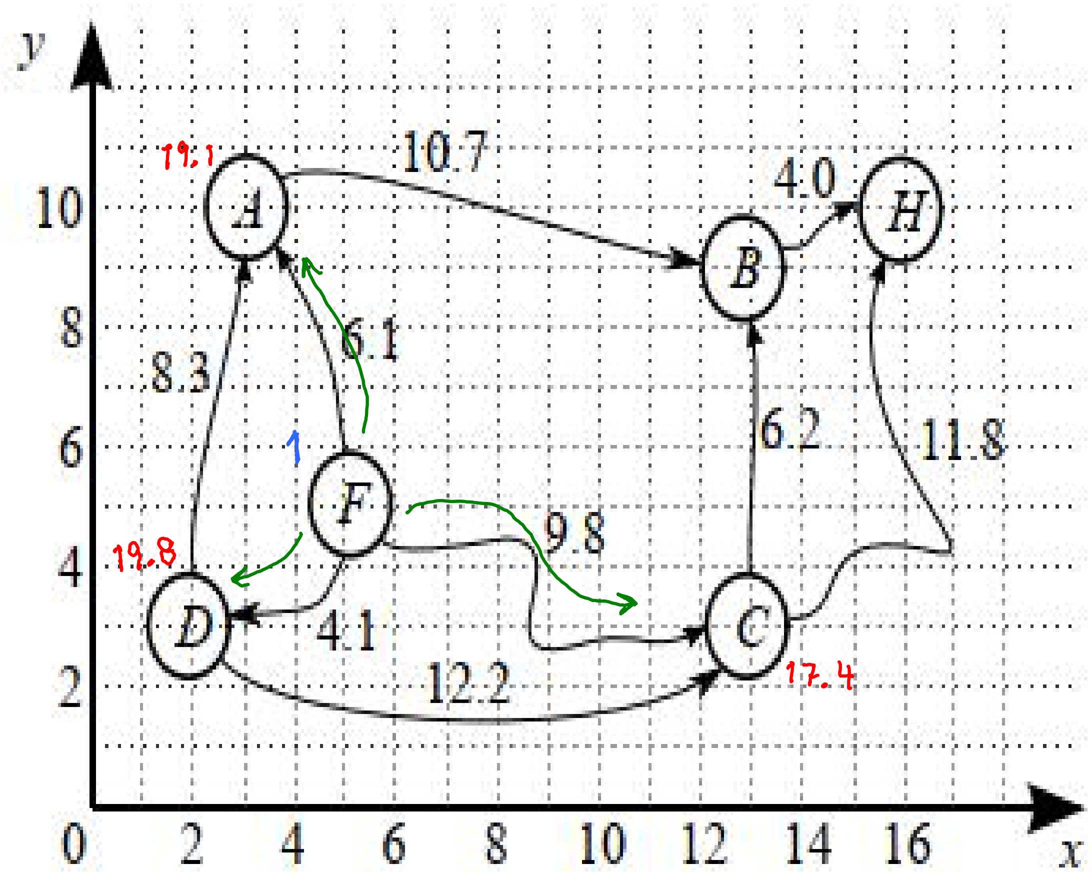
Shortest path from *s* = *F* to *t* = *H* using ASTAR

i = order in which node handled

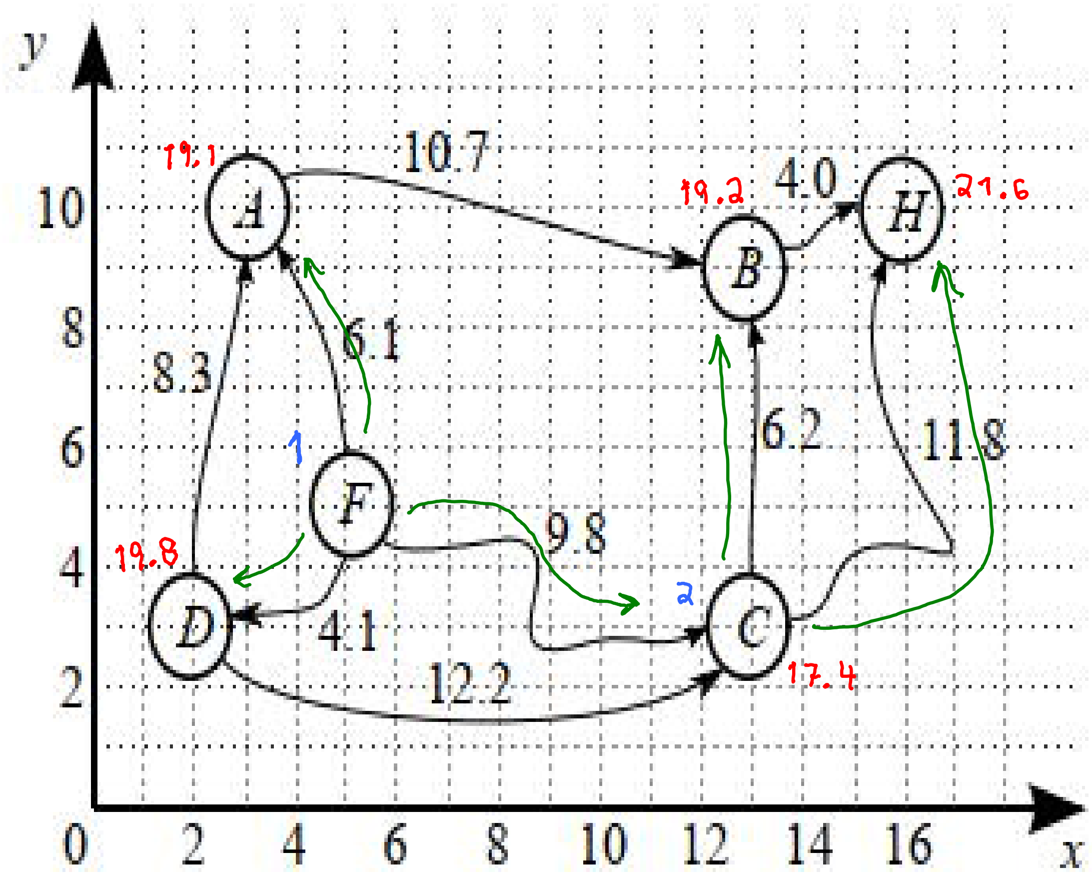
x.e = lower bound estimate of length of shortest path from *s* = *F* to *t* = *H* passing through *x*

→ = edge of current shortest path tree

(x,H)	(A,H)	(B,H)	(C,H)	(D,H)	(F,H)	(H,H)
$h(x,t)$	13.0	3.2	7.6	15.7	12.1	0.0



(x,H)	(A,H)	(B,H)	(C,H)	(D,H)	(F,H)	(H,H)
$h(x,t)$	13.0	3.2	7.6	15.7	12.1	0.0



continue ...

3. Remarks

- If $h(x,t) = 0$, then A^* is same as Dijkstra.
- Two important properties of $h(x,t)$:

  admissible If $h(x,t)$ is admissible, then $h(x,t) \leq g((x,t))$.

  consistent If $h(x,t)$ is consistent, then $h(x,t) \leq h(y,t) + w((x,y))$.

  Why important?

  - If $h(x,t)$ is admissible, then A^* is guaranteed to return a shortest path solution.
  - If $h(x,t)$ is consistent, then A^* is guaranteed to return a shortest path without ever removing the node more than once from the priority queue.