LRTA*

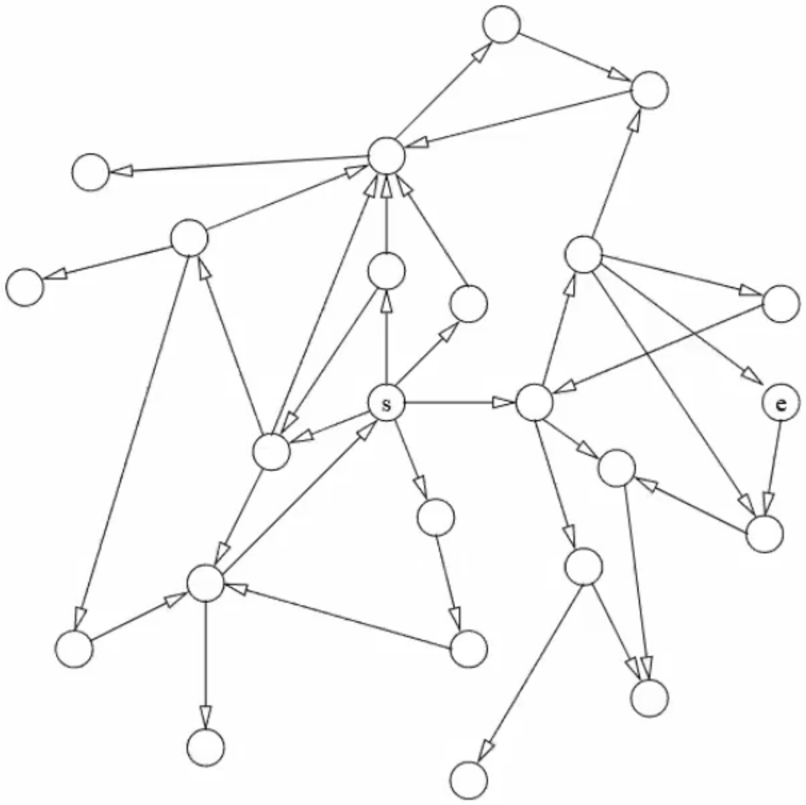
A*:

- keeps a set of reachable states (frontier)
- never changes the heuristics

LRTA*:

- immediately choose a single successor state
- changes the heuristics as it runs

hasty algorithm



in the example: goal `e` is on the right of start `s`

always follow the arrow heading furthest on the right

hasty algorithm: principle

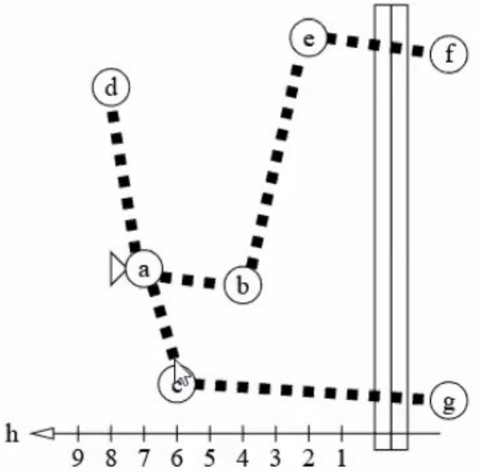
the heuristic function `h(s)` approximates the distance from the a state `s` to the goal

choose the action that leads to the closest successor

two drawbacks:

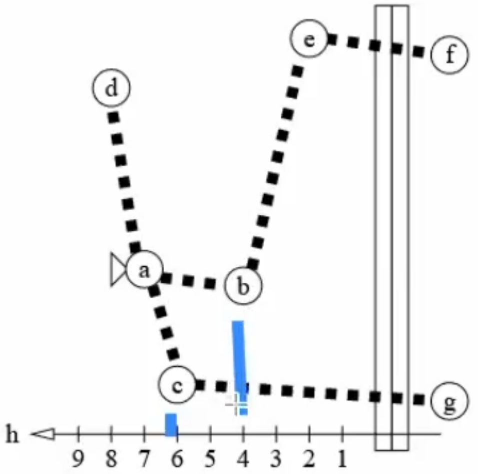
- may not find the optimal path
- may get trapped in dead ends

hasty algorithm: example



initial state `a`
goal: go over the barrier
segments can be traversed in both directions
cost of traversing: number of boxes in the segment
heuristic is distance from barrier

best solution

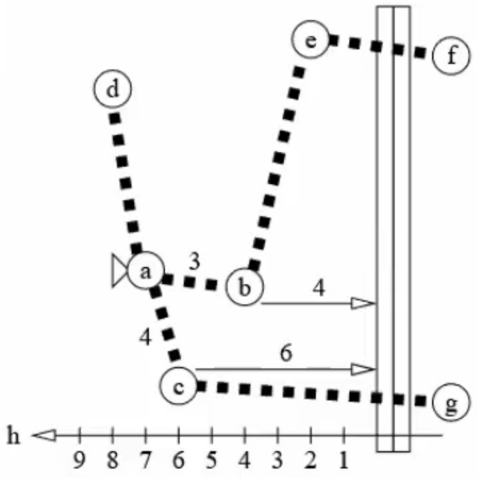


best solution is to go to `c`, then `g`
cost: first action 4, second action 11, total 15

second best: go to `b`, then `e`, then `f`
cost: 3+10+5=18

the hasty algorithm makes the wrong choice

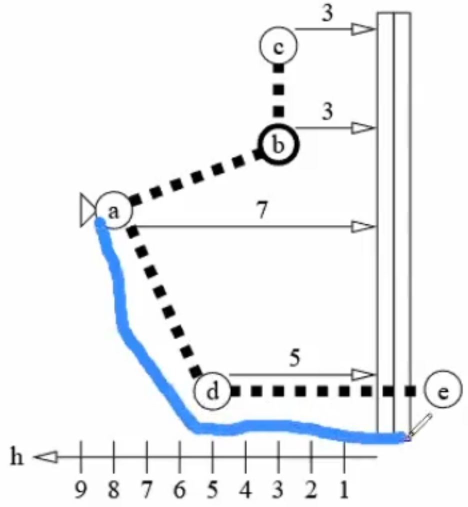
hasty algorithm drawback 1: non-optimality



start from a
cost of action + heuristics of successor is:
to b: action 3, distance 4, total 7
to c: action 4, distance 6, total 10

b looks better (7 vs. 10)
it is not (previous slide: 18 vs. 15)

hasty algorithm drawback 2: traps



a different example

start from a
best successor is b

```
in b:
to a: action 7, distance 7, total 14
to c: action 3, distance 3, total 6

algorithm goes to c
from c, only possible choice is go back to b
where it goes to c again
```

principle

heuristic function `h(s)`
estimate the cost for reaching the goal from `s`

hasty algorithm:

> in the state `s`, execute the action `a` of minimal `c+h(s')`
> where: `c` is the cost of `a` and `s'` the resulting state

short latency: fast to find the first action
try to solve drawbacks 1 & 2, above

about the heuristic distance

if the heuristics is admissible, `h(s)` is always lower than or equal to the cost of reaching the goal from `s`

but…

an higher `h(s)` is a better estimate of the cost
as long as it is admissible, higher is better

solution to drawbacks 1 & 2: increase `h(s)`

improving the heuristics by learning

current state s

cost of possible actions: $c_1, c_2 \ldots c_n$
resulting states: $s_1, s_2 \ldots s_n$

cost of action + estimate of resulting state:
$c_1+h(s_1), \quad c_1+h(s_1) \quad \ldots \quad c_1+h(s_1)$

these are estimate costs of reaching the goal depending on the first action
optimistic estimates, since the heuristic is admissible

minimal $c_i+h(s_i)$:
cannot do better, $h(s_i)$ is already optimistic
all others $c_j+h(s_j)$ are larger

$c_i+h(s_i)$ is an optimistic estimate of the cost of the best path from s to the goal
it is itself an *admissible heuristics* for s

if the minimal $c_i+h(s_i)$ is higher than $h(s)$, then it is a better heuristic than it

update $h(s)$ by setting it to the minimal $c_i+h(s_i)$

LRTA*

current state s

possible successors $s_1, s_2 \ldots s_n$
cost $c_1, c_2 \ldots c_n$

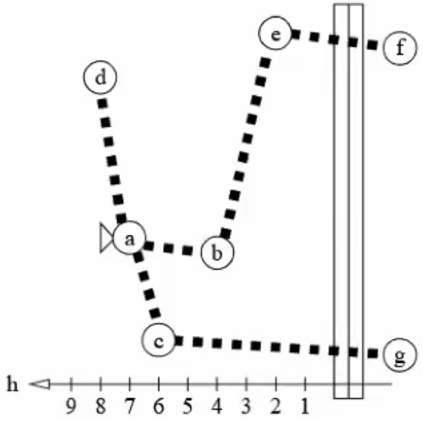
go to s_i such that $c_i+h(s_i)$ is minimal

if $h(s)$ is lower than $c_i+h(s_i)$
set $h(s)=c_i+h(s_i)$

LRTA* vs. hasty algorithm

- LRTA* does not get trapped in dead ends
- LRTA* obtains the best plan with iterated runs

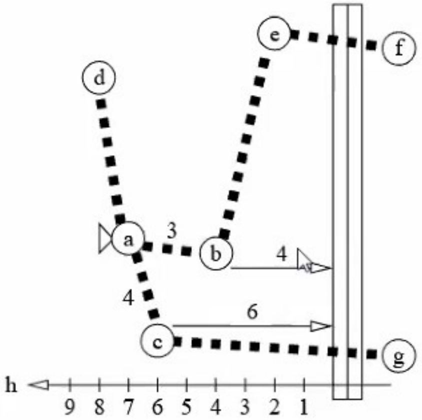
LRTA*: example



initial state s
goal: get over the barrier

cost of action: number of squares in segment
initially, h is the distance from the barrier
(axis at the bottom of the figure)

example: first action



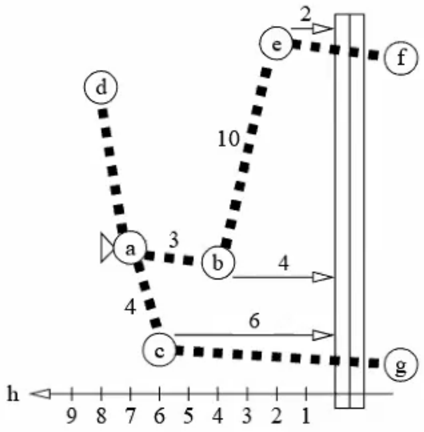
initial state a

successors: b and c
b: cost of action 3 squares, distance to the barrier h(b)=4, total 7
c: cost of action 4 squares, distance to the barrier h(c)=6, total 10

move to b
h(a) already equal to 7, do not change

note: this **is not A***
LRTA* takes the best choice
A* iteratively expands the set of all (direct and indirect) best successors

underestimate



h(b)=4 means:
estimate cost of reaching the goal from b is 4

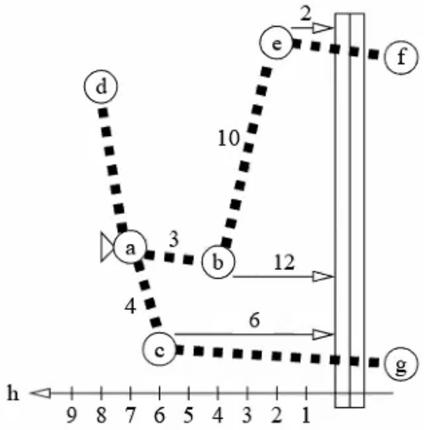
actually: 15
heuristics underestimates the cost

when in b, LRTA* realizes that h(b)=4 is too low
should have been at least:
10, cost to get to e, plus
2, estimate distance from e to the goal

update h(b)=12
closer to reality than previous value h(b)=4

too late?

update value



already in b, updating h(b) looks pointless

but:

a new plan from d may later be needed
use the new value of h(b), make better choices

when in b the choices will be:
a: cost of action 3, estimate distance to goal h(b)=12, total 15
c: cost of action 4, estimate distance to goal h(b)=6, total 10

this time, go to c
it was the best choice

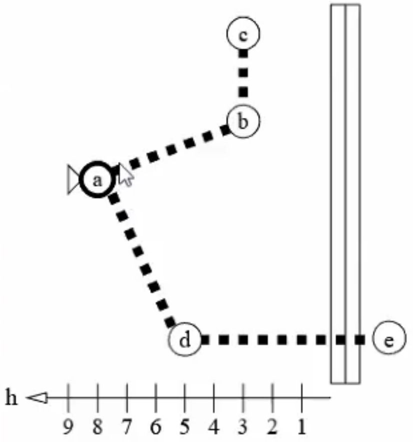
with repeated runs, LRTA* finds the optimal solution

escape a dead end

when trapped, `h(s)` keeps increasing until coming out

works on *safely explorable domains*
goal is reachable from every state

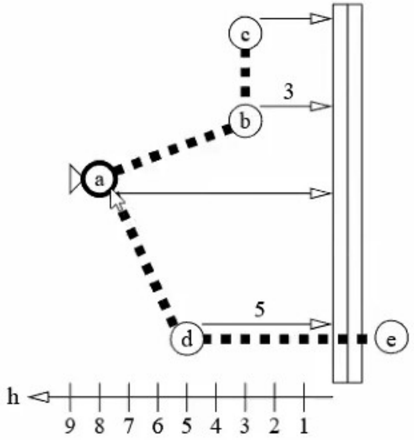
a dead end



cost of following a segment: number of squares
heuristic: distance from barrier
(all as before)

starting state `a`
`b` and `c` were a dead end for the hasty algorithm

first action

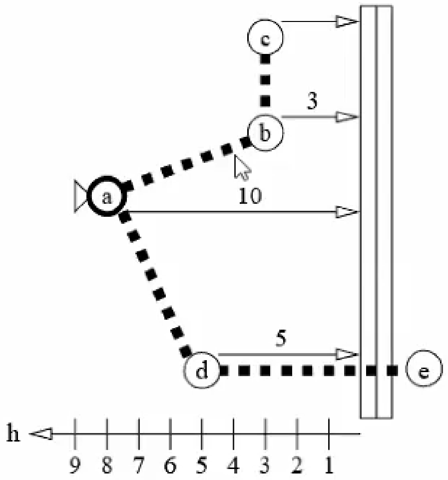


two actions:
go to `b`: cost of action `7`, estimate distance from there to goal `3`, total `10`
go to `d`: cost of action `8`, estimate distance from there to goal `5`, total `13`

go to `b`, like the hasty algorithm

but, before that…

update the heuristics



`h(a)=8` means that the estimate cost of reaching the goal is `8`

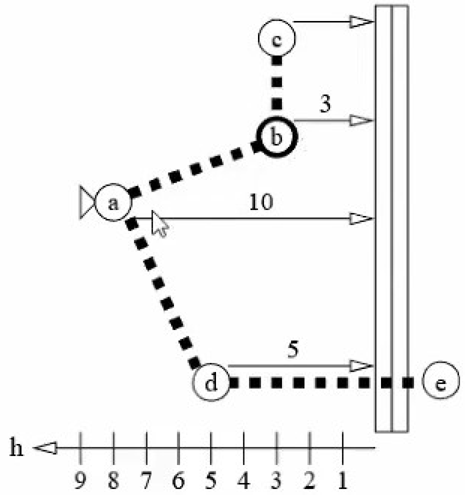
would be possible going directly, without following the paths

following the paths, cannot be lower than `7+3=10` (going to `b` first) or `8+5=13` (going to `c` first)

set `h(a)=10`

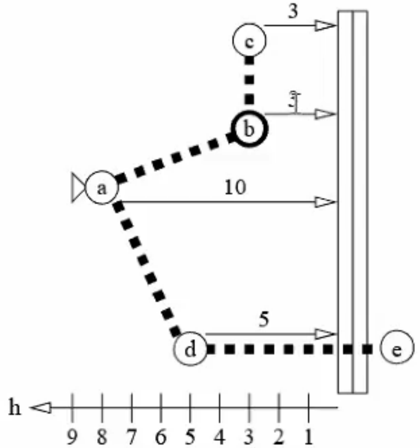
now go to `b`

possible second actions



current state `b`
from `b`: either go to `a` or to `c`

decide the second action

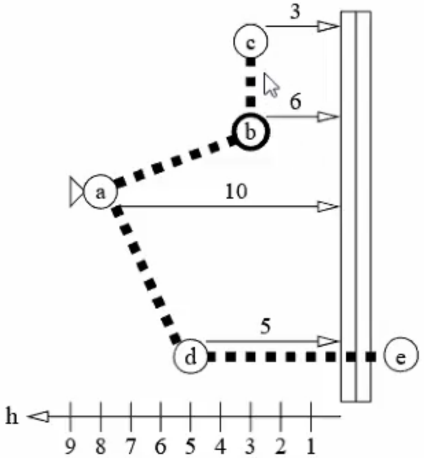


cost:
go to a: 7 squares + heuristics 10
go to c: 3 squares + heuristics 3

go to c

but, before moving…

another heuristic update

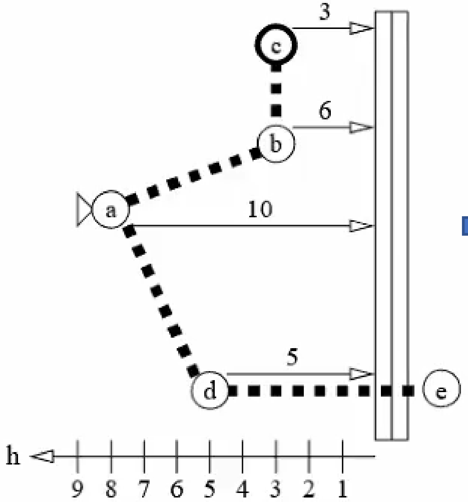


before going to c
(3 squares + heuristics 3)

update h(b)=3+3

now go to c

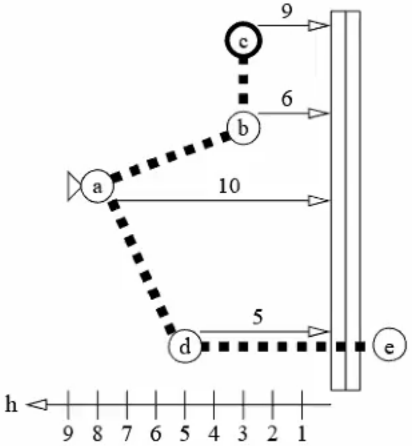
dead end



no other choice than go back to b

but, before moving…

before coming back



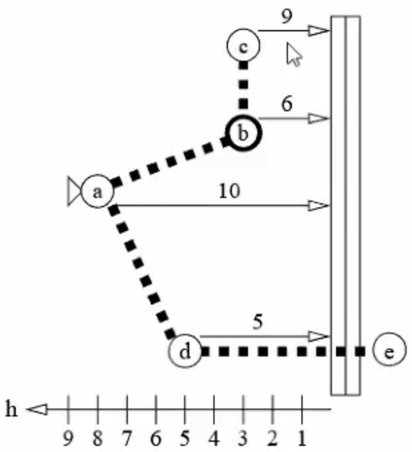
before going from c to b

action from c to b costs 3
estimate distance from b to goal is now 6

update h(c)=3+6=9

now move to b

same mistake again

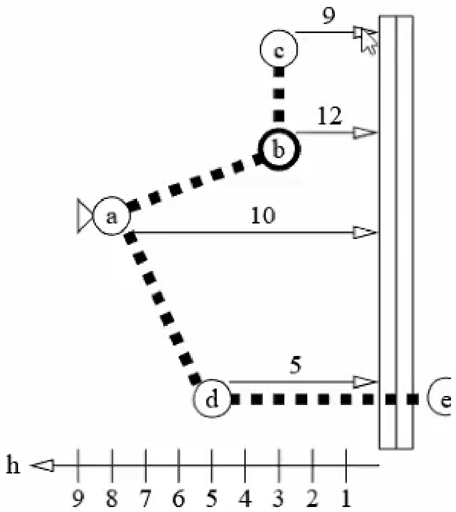


go to `c`: action `3`, heuristics `9`
go to `a`: action `7`, heuristics `10`

still go to `c`, as before
but now the difference is lower

and, increase `h(b)` again before going to `c`

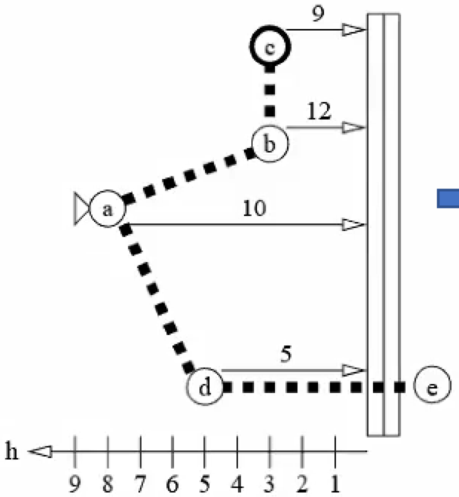
second increase



go to `c`: action `3`, heuristics `9`

update `h(b)=3+9=12`

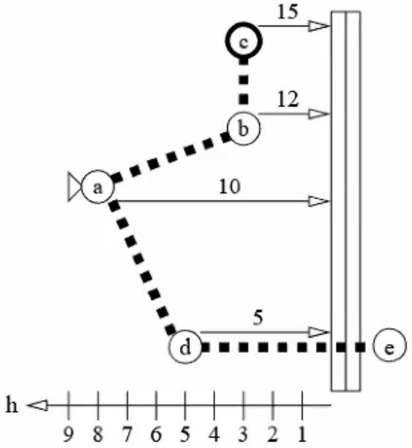
dead end, again



only choice is go to `b`

update `h(c)` before moving

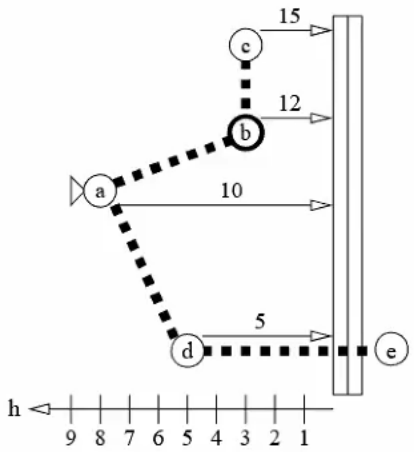
further increase in dead end



go to `b`: `3` squares, heuristic `12`

update `h(c)=15`

something completely different

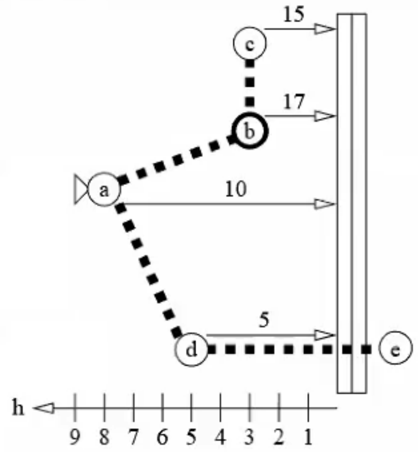


current state `b`

go to `c`: `3` squares + heuristics `15`
go to `a`: `7` squares + heuristics `10`

this time, go to `a` instead of `c`

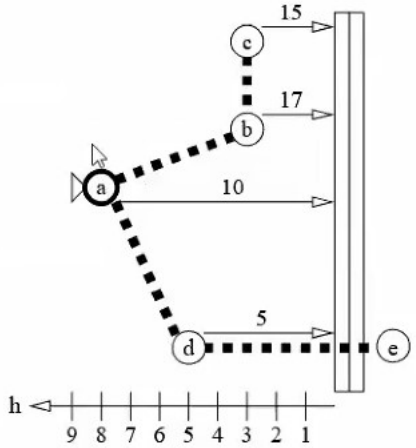
update `h(b)` before moving



update `h(b)`

how the dead end was escaped:
`h(c)` and `h(b)` kept increasing each other at every turn

back to the start

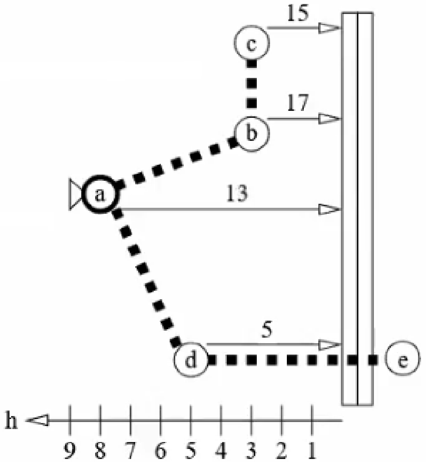


but not quite: updated heuristics in `b`

go to `b`: `7` squares, heuristics `17`
go to `d`: `8` squares, heuristics `5`

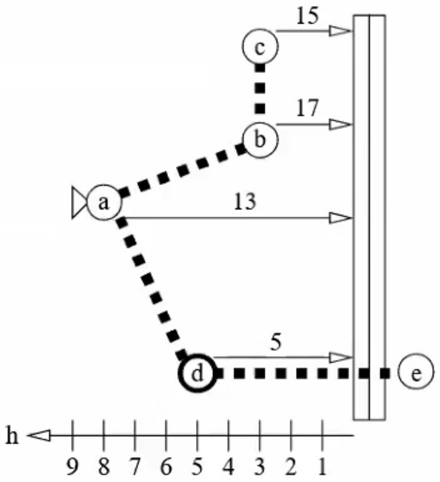
go to `d`

before moving



before going to `d`, update `h(a)`
to `d`: `8` squares, heuristics `5`
set `h(a)=8+5=13`

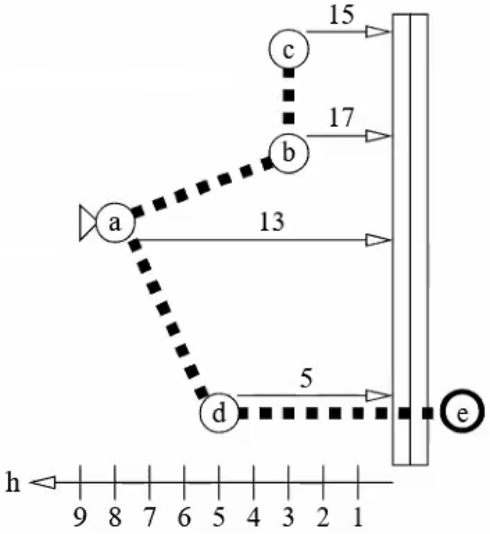
almost done



current state `d`

goal reached going to `e`

goal!



LRTA*: summary

when in a state `s`, update `h(s)=c+h(s')` before moving to `s'` with an action of cost `c`

updated value is a more realistic estimate of the distance to the goal

- good for replanning
  (a new plan is needed for the same domain, possibly from a different initial state)
- good for avoding dead ends
  (when trapped, the heuristic estimate keep increasing)

LRTA*: lights and shades

- little latency
  first action is decided in a very short time
- allows performing actions immediately
  no need to compute a whole plan before starting acting
  useful if the cost of actions is low when compared to that of planning
- always reaches the goal on safely explorable domains
  (= the goal is reachable from every state)
- always converges to the optimal solution
  but only iterating the whole search for a goal
  restarting from the initial state each time
- can be easily extended to nondeterministic domains

may not work on true dead ends (states where goal is unreachable)

takes time to obtain an optimal plan
requires restarting from the initial state until the heuristics converges

the name

LRTA* = Learning Real-Time A*

Learning = the heuristics is updated
Real-Time = decide an action that can be executed immediately
A* = an extension to A*

but really:

- "real-time" = able to satisfy external time constraints
  a better term is "online": able to plan concurrently with execution
- not a variant of A*
  just uses an heuristics as A* does

LRTA*: variants

deeper lookhaead
    instead of just the states after an action,
    check the states after two or more
weighted heuristics
    multiply the initial heuristics by a factor $(1+\varepsilon)$
backtracking
    each time the heuristics for a state is updated,
    go back to the previous state