Probabilistic Graphical Models

(Concluded)

Lecture 20

Ganesh Ramakrishnan

Recap: Inferencing in Graphical Models

- Exact inferencing
  - Message passing
  - Beam Search
  - A* search
  - Junction tree algorithms
  - Integer Linear Programming based inferencing

Let us revisit

(exact with large beam size)

Recap in context of HMM

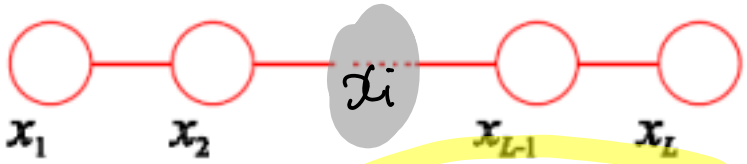
optional: see notes

Exact Inferencing in Graphical Models

- Why hard ? We saw
- Message passing algorithm
  - Applicable to tree-structured directed and undirected graphical models – illustration on HMM Part of Speech Tagging
- (Extra & Optional) Junction Tree algorithm
  - Applicable to arbitrary undirected graphs with cycles

Message Passing algorithm: Broad Idea

- Example



- Find marginal for a particular node x_i

$$p(x_i) = \sum_{x_1} \sum_{x_2} \cdots \cdots \sum_{x_{i-1}} \sum_{x_{i+1}} \cdots \cdots \sum_{x_L} p(x_1, x_2 \ldots x_L)$$

  - for M-state nodes, cost is $O(M^L)$
  - exponential in length of chain
  - but, we can exploit
    1. the graphical structure (conditional independences)
    2. Avoidance of redundant computations through dynamic programming

[Exploiting that x_i "blocks" info from $x_1 \ldots x_{i-1}$ to $x_{i+1} \ldots x_L$]

Message Passing: Broad idea

- Exchange sums and products

(Recap exchange of operations in DFT)

- Express as product of messages (Z is obtained by normalization)

$x_1, x_2 \ldots x_{L-1} \rightarrow m_\alpha(x_i)$

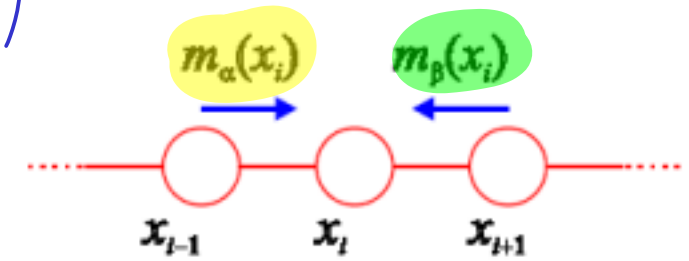
$$p(x_i) = \frac{1}{Z} \cdots \sum_{x_2} \psi(x_2, x_3) \left[\sum_{x_1} \psi(x_1, x_2) \right]$$

$$\cdots \sum_{x_{L-1}} \psi(x_{L-2}, x_{L-1}) \left[\sum_{x_L} \psi(x_{L-1}, x_L) \right]$$

$m_\beta(x_i) \leftarrow x_{i+1} \ldots x_L$

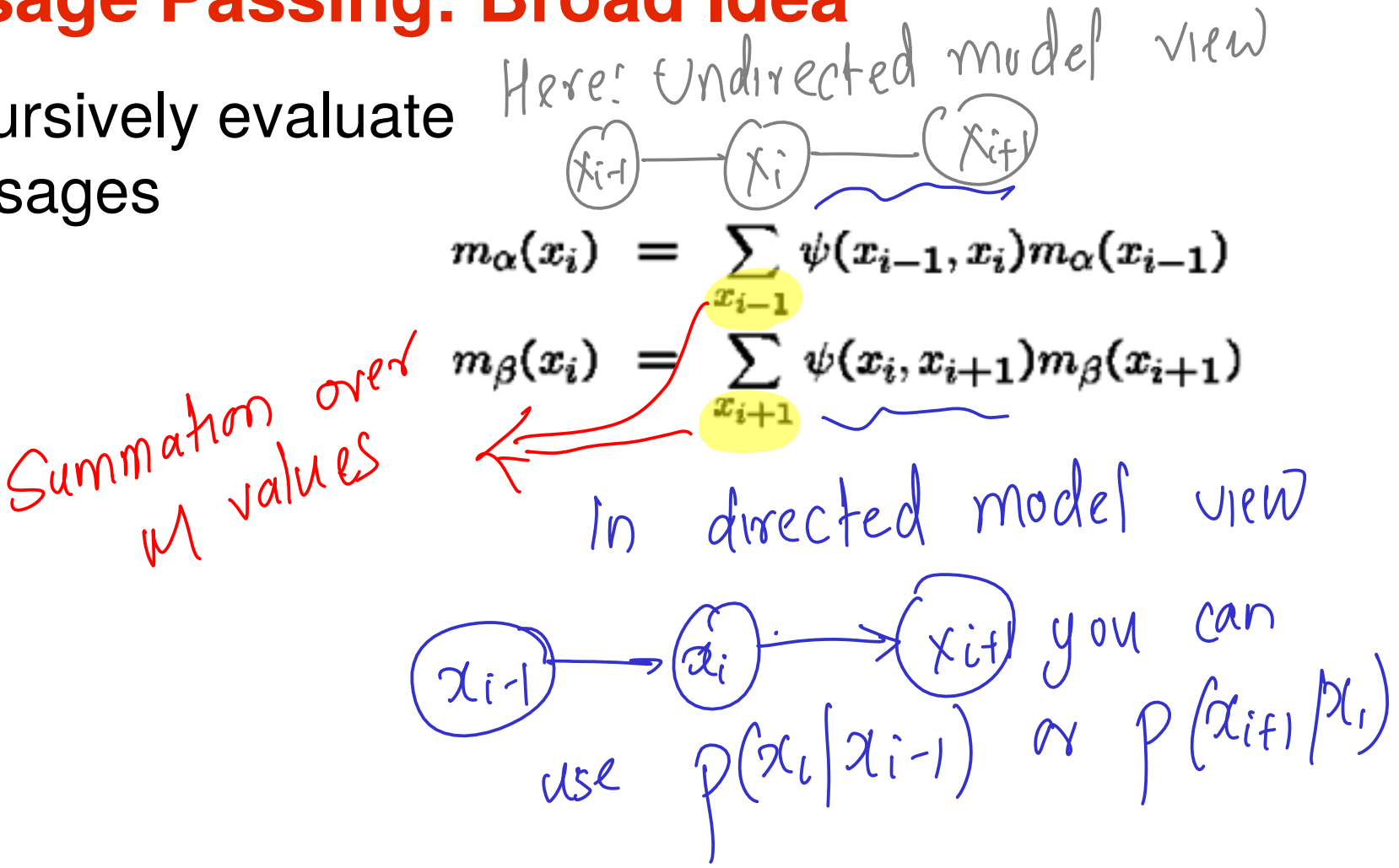
Further: $m_\alpha(x_i)$, $m_\alpha(x_{i-1})$, $m_\alpha(x_{i+1})$ etc are related!

$$p(x_i) = \frac{1}{Z} m_\alpha(x_i) m_\beta(x_i)$$



Message Passing: Broad Idea

- Recursively evaluate messages



Max Product Algorithm: Broad Idea

- Goal: find $\mathbf{x}^{\text{MAP}} = \arg\max_{\mathbf{x}} p(\mathbf{x})$

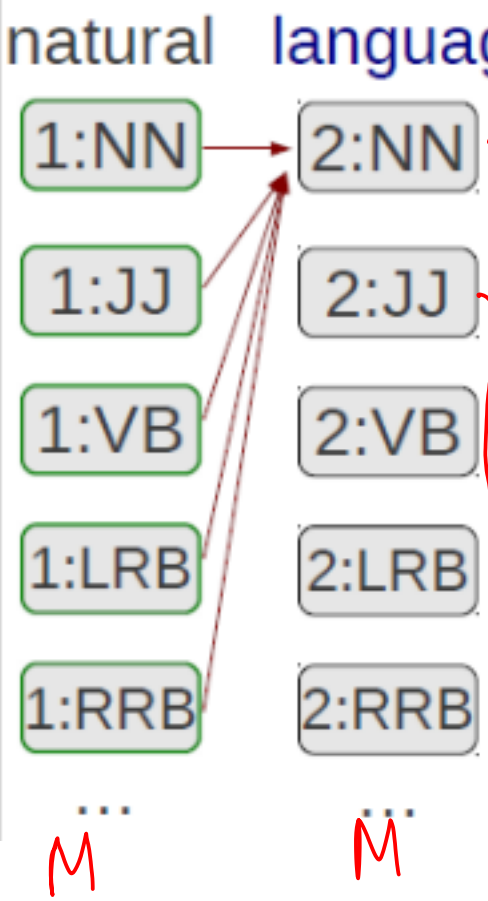
Recap Viterbi for POS

- Define $\phi(x_i) = \max_{x_1} \cdots \max_{x_{i-1}} \max_{x_{i+1}} \cdots \max_{x_L} p(x_1, \ldots, x_L)$
- Then $x_i^{\text{MAP}} = \arg\max_{x_i} \phi(x_i)$
- Message passing algorithm with "sum" replaced by "max"
- Generalization of Viterbi algorithm for HMMs

Recap: Viterbi in HMM for POS

- For middle words, calculate the minimum score for all possible previous POS tags

(we saw application of "beam" on Viterbi)



best_score["2 NN"] = min(
best_score["1 NN"] + -log P_T(NN|NN) + -log P_E(language | NN),
best_score["1 JJ"] + -log P_T(NN|JJ) + -log P_E(language | NN),
best_score["1 VB"] + -log P_T(NN|VB) + -log P_E(language | NN),
best_score["1 LRB"] + -log P_T(NN|LRB) + -log P_E(language | NN),
best_score["1 RRB"] + -log P_T(NN|RRB) + -log P_E(language | NN),
...
)

best_score["2 JJ"] = min(
best_score["1 NN"] + -log P_T(JJ|NN) + -log P_E(language | JJ),
best_score["1 JJ"] + -log P_T(JJ|JJ) + -log P_E(language | JJ),
best_score["1 VB"] + -log P_T(JJ|VB) + -log P_E(language | JJ),

Summary of inference methods

	Chain (online)	Low treewidth	High treewidth
Discrete	BP = forwards (Viterbi) Boyen-Koller (ADF), beam search (BFS) A* (DFS)	VarElim, Jtree, recursive conditioning	Loopy BP, mean field, structured variational, EP, graphcuts Gibbs
Gaussian	BP = Kalman filter	Jtree = sparse linear algebra	Loopy BP Gibbs
Other	EKF, UKF, moment matching (ADF) Particle filter	EP, EM, VB, NBP, Gibbs	EP, variational EM, VB, NBP, Gibbs

A bit more

Exact Deterministic approximation Stochastic approximation

BP=belief propagation, EP = expectation propagation, ADF = assumed density filtering, EKF = extended Kalman filter, UKF = unscented Kalman filter, VarElim = variable elimination, Jtree= junction tree, EM = expectation maximization, VB = variational Bayes, NBP = non-parametric BP

Slide Credit: Kevin Murphy

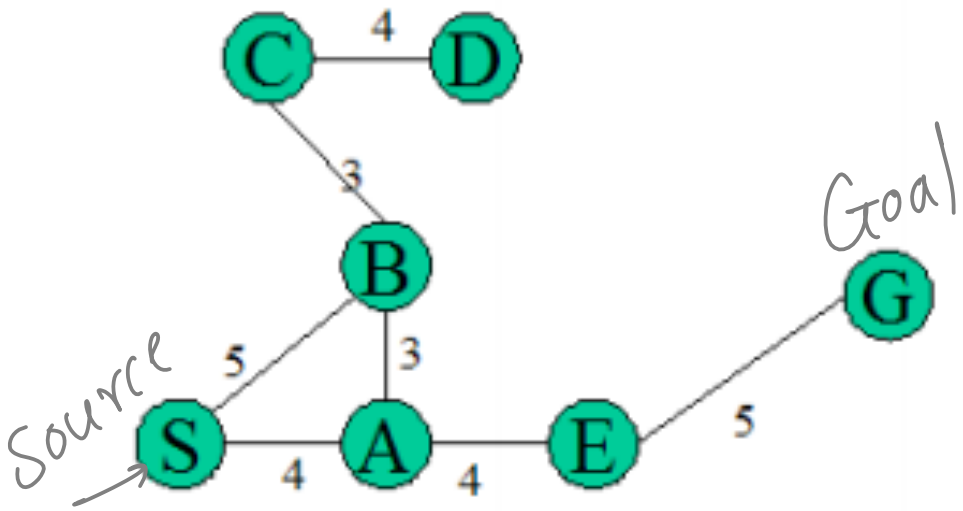
Search Algorithms: Further Illustrated

Lecture 20

Specifically culminating in A*

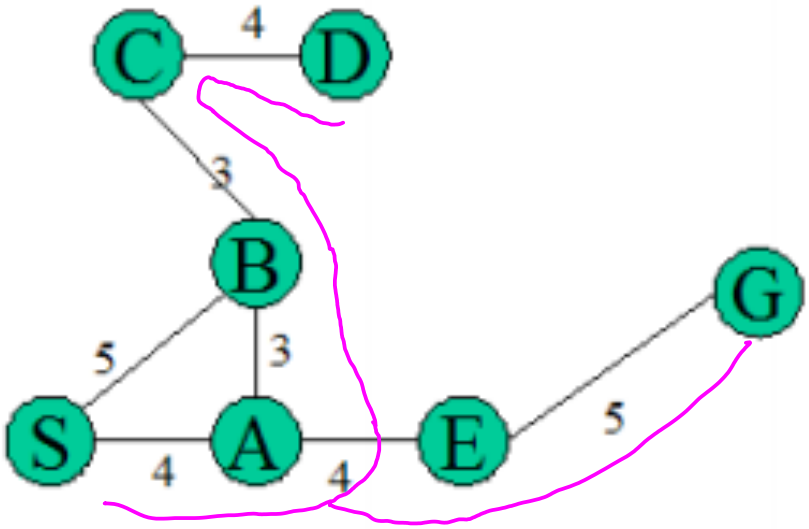
Ganesh Ramakrishnan

Search Algorithms Illustrated

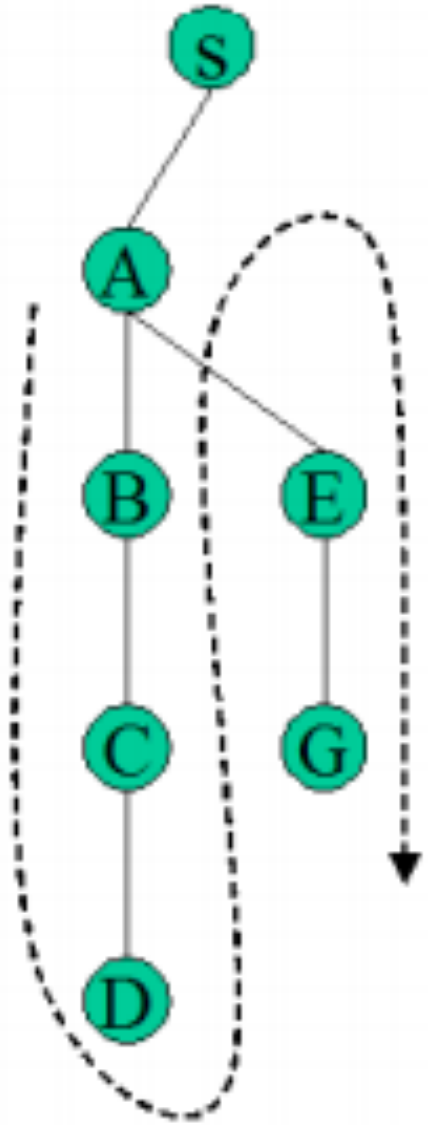


- Consider the problem of searching shortest path(s) from S (start) to G (goal).
- Refer: https://moodle.iitb.ac.in/pluginfile.php/270556/mod_resource/content/1/searchingOnGraphs.pdf

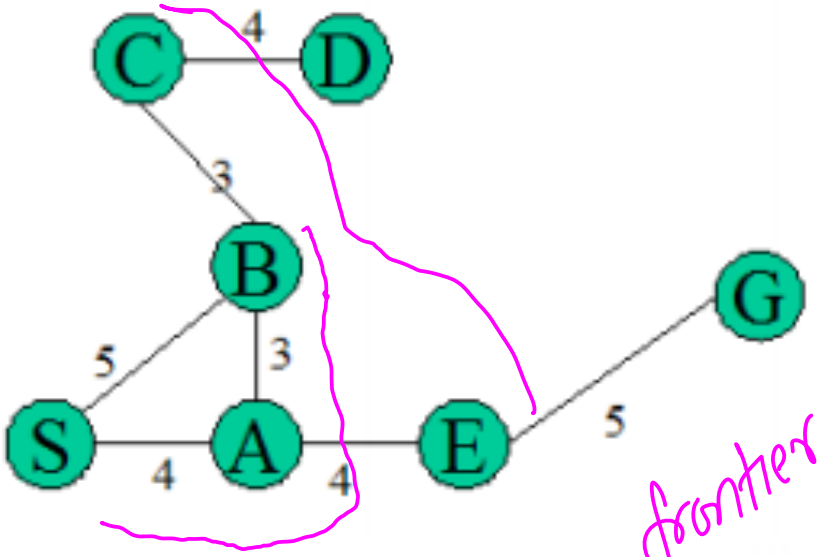
Depth First Search



Expand using stack



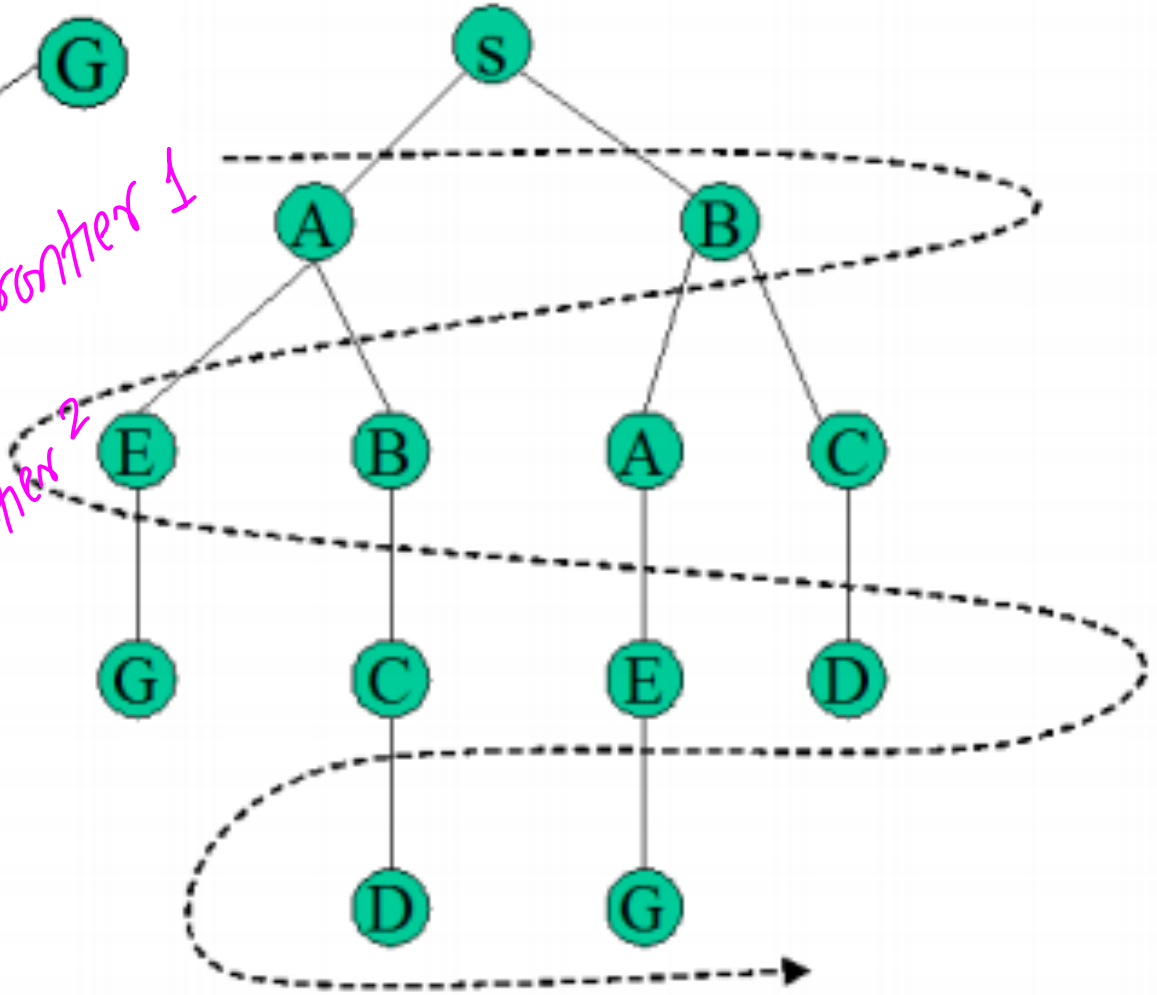
Breadth First Search



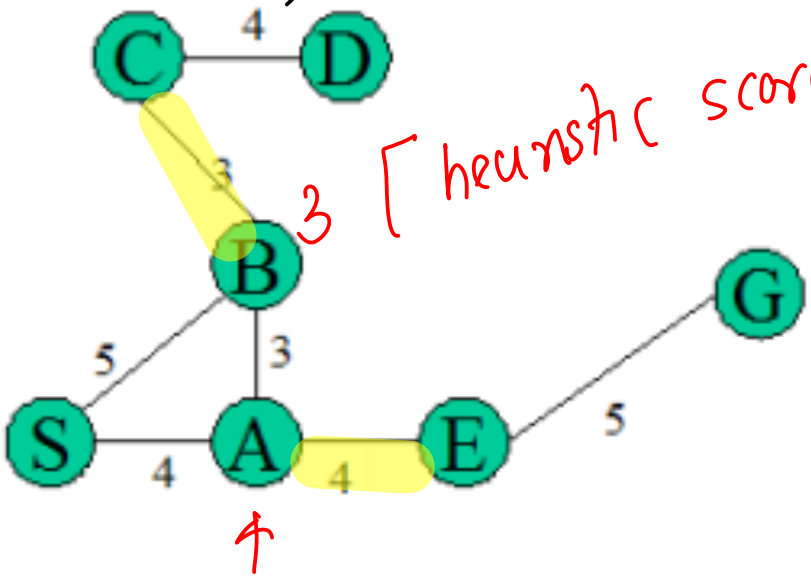
Queue

frontier 1

frontier 2



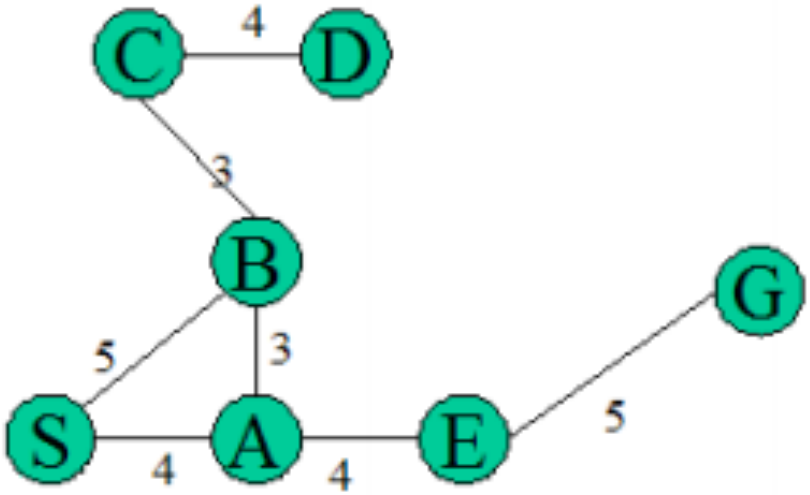
(Simple) # Hill Climbing Search

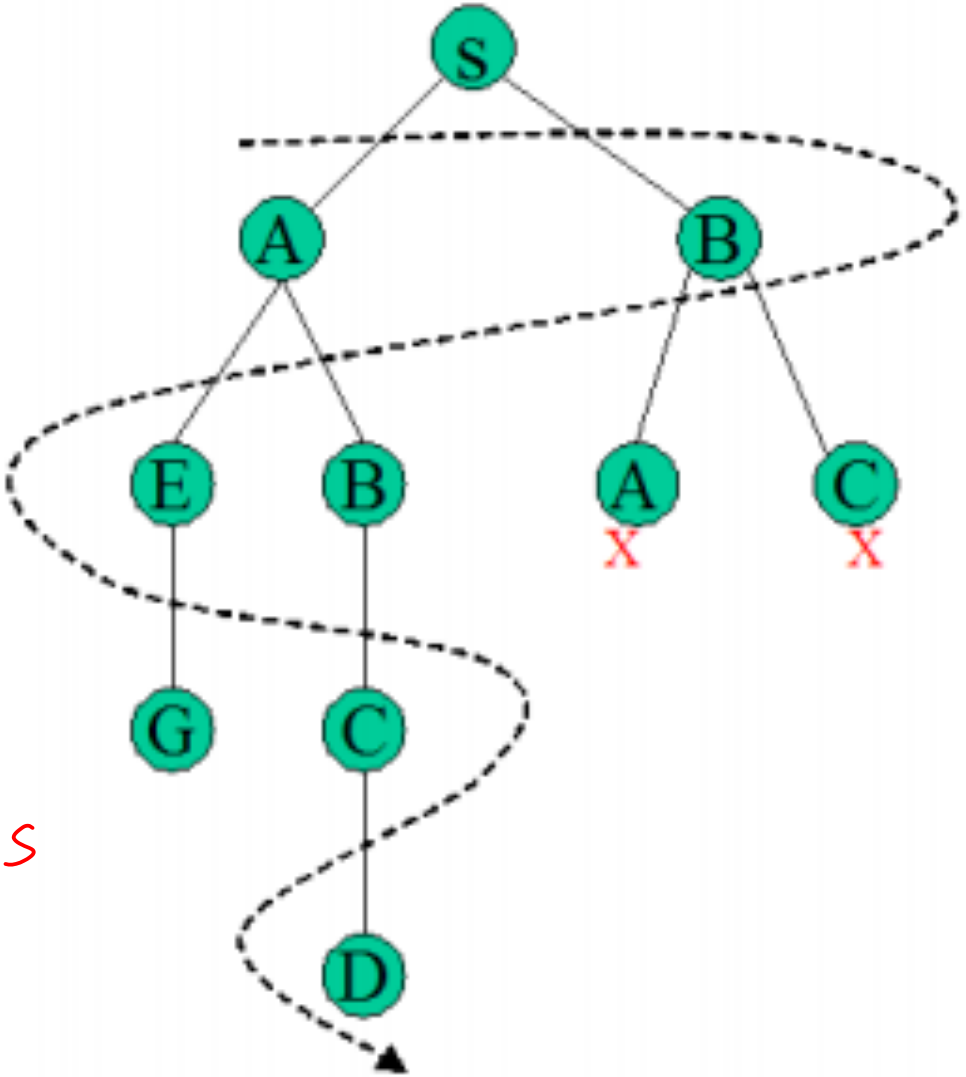


3 [heuristic scores]

Consider heuristic score as smallest wt on edge incident on a non-repeating neighbor

Beam Search





Beam search on BFS

Beam search could also be applied on others

B = 2

Recap Beam Search for HMM POS

on Viterbi More informed

- Calculate score, but ignore removed hypotheses

(B=3)

Heuristic Search

Suppose in addition to the standard search specification we also have a *heuristic*.

A heuristic function maps a state onto an estimate of the cost to the goal from that state.

Can you think of examples of heuristics?

- Eucledian distance?
- Smallest of all outgoing edge weights at a node?
- Recap HMM: The highest emission probability

we looked at max prob path $P(\text{language} \mid NN) \geq P(\text{language} \mid NN)\, P(\text{processing} \mid N)$

Denote the heuristic by a function $h(s)$ from states to a cost value.

Recap: A* Search: Optimistic Heuristic for HMM POS Taggiong

- Consider the words remaining
  - Use Optimistic Heuristic:
    - Upper bound on score (highest possible score)
  - Optimistic heuristic for tagging:
    - Best Emission Prob

natural	language	processing
log(P(natural|NN)) = -2.4	log(P(lang.|NN)) = -2.4	log(P(proc.|NN)) = -2.5
log(P(natural|JJ)) = -2.0	log(P(lang.|JJ)) = -3.0	log(P(proc.|JJ)) = -3.4
log(P(natural|VB)) = -3.1	log(P(lang.|VB)) = -3.2	log(P(proc.|VB)) = -1.5
log(P(natural|LRB)) = -7.0	log(P(lang.|LRB)) = -7.9	log(P(proc.|LRB)) = -6.9
log(P(natural|RRB)) = -7.0	log(P(lang.|RRB)) = -7.9	log(P(proc.|RRB)) = -6.9

H(1+) = -5.9 H(2+) = -3.9 H(3+) = -1.5 H(4+) = 0.0

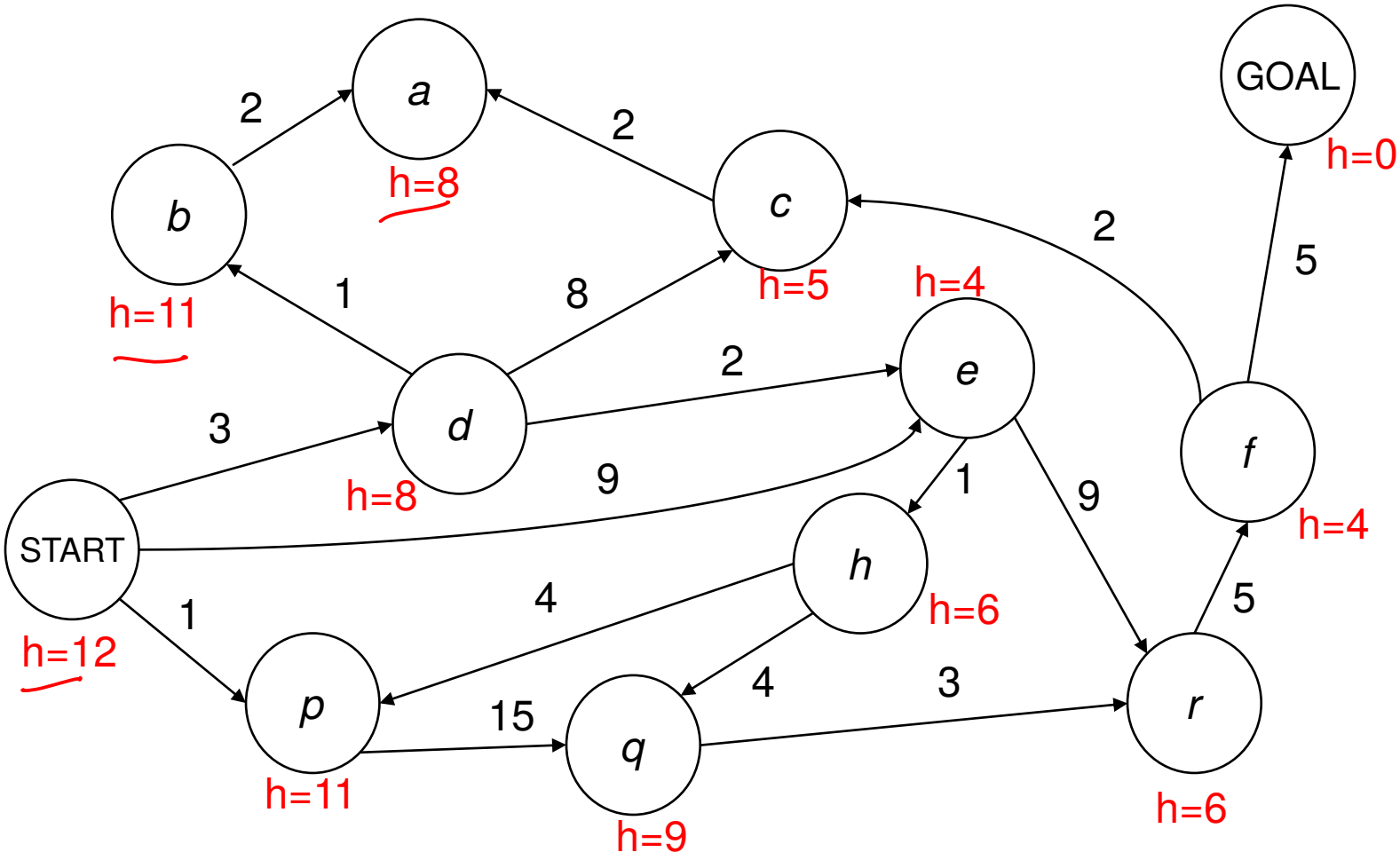
Recap: A* Search: Optimistic Heuristic for HMM POS Taggiong

- Use Forward Score + Optimistic Heuristic

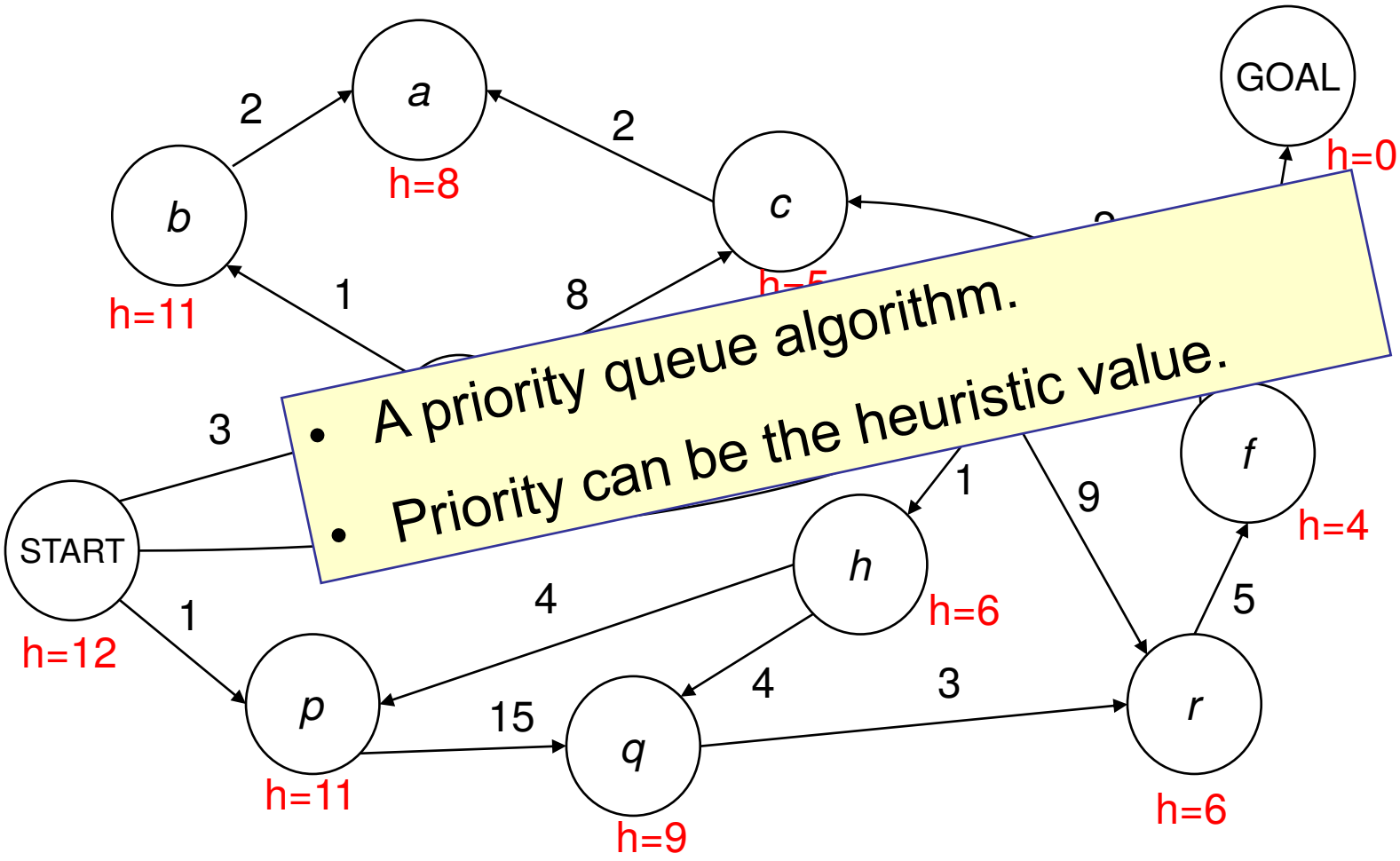
A* heap included heuristic for

Regular Heap			A* Heap	
2:VB	F(2:VB)=-5.7	H(3+)=-1.5	3:NN	-7.2
2:JJ	F(2:JJ)=-5.9	H(3+)=-1.5	2:VB	-7.2
2:JJ	F(2:JJ)=-6.7	H(3+)=-1.5	3:VB	-7.3
3:NN	F(3:NN)=-7.2	H(4+)=-0.0	2:JJ	-7.4
3:VB	F(3:VB)=-7.3	H(4+)=-0.0	2:JJ	-8.2
1:RRB	F(1:RRB)=-8.1	H(2+)=-3.9	3:JJ	-9.8
1:LRB	F(1:LRB)=-8.2	H(2+)=-3.9	1:RRB	-12.0
3:JJ	F(3:JJ)=-9.8	H(4+)=-0.0	1:LRB	-12.1
2:LRB	F(2:LRB)=-11.2	H(3+)=-1.5	2:LRB	-12.7

Euclidian Heuristic



Euclidian Heuristic



Steepest Ascent Hill Climbing

```
Init-PriQueue(PQ)
Insert-PriQueue(PQ,START,h(START))
while (PQ is not empty and PQ does not contain a goal state)
    (s , h ) := Pop-least(PQ)
    foreach s' in succs(s)
    if s' is not already in PQ and s' never previously been visited
            Insert-PriQueue(PQ,s',h(s'))
```

HMM: we used pop-max

// least : for shortest path

But can be suboptima

ensures we will reach goal

- *Steepest ascent hill climbing* considers all possible extensions of the current path
- In contrast, the *simple hill climbing* algorithm expands only on that s' in succs(s) that has least value of h(s')

for each s' in succ(s): Push only s' into PQ where s' has least h(s')

Steepest Ascent Hill Climbing

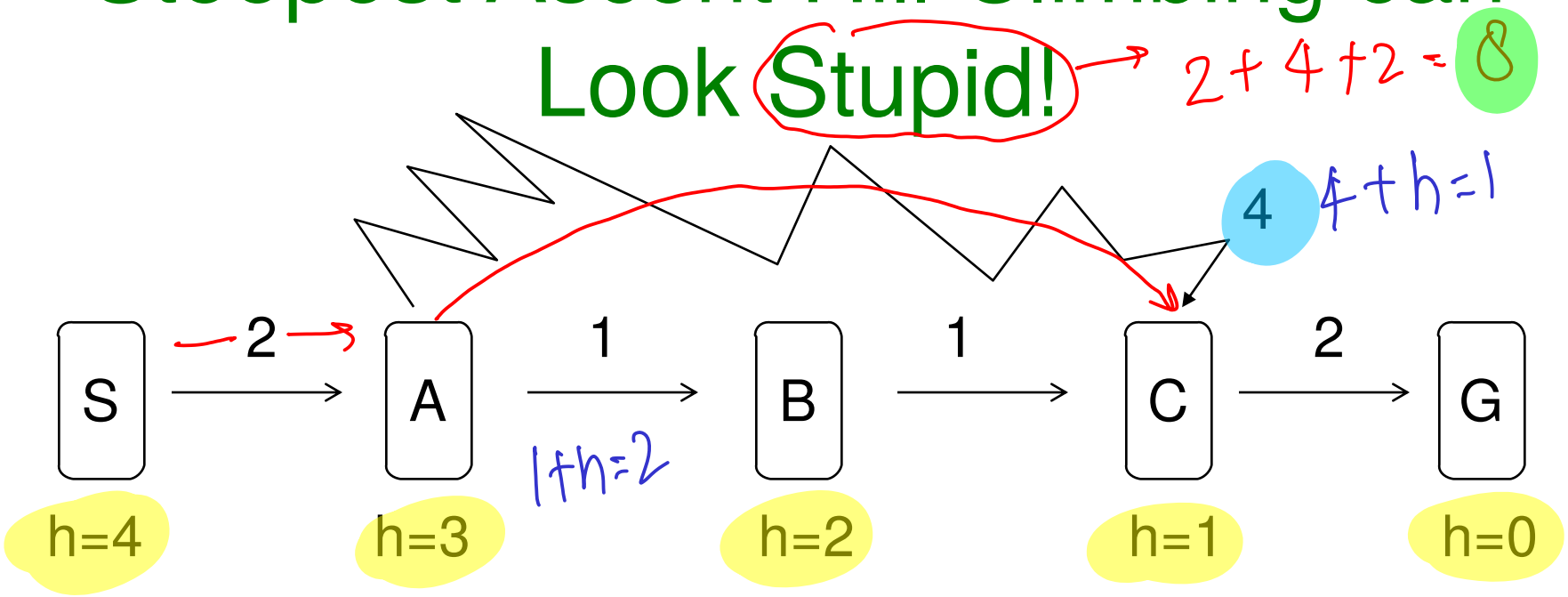
```
Init-PriQueue(PQ)
Insert-PriQueue(PQ,START,h(START))
while (PQ is not empty and PQ does not contain a goal state)
        (s , h ) := Pop-least(PQ)
        foreach s' in succs(s)
        if s' is not already in PQ and s' never previously been visited
                Insert-PriQueue(PQ,s',h(s'))
```

N	number of states in the problem
B	the average branching factor (the average number of successors) ($B>1$)
L	the length of the path from start to goal with the shortest number of steps
LMAX	Length of longest **cycle-free** path from start to anywhere
Q	the average size of the priority queue

Algorithm		Complete	Optimal	Time	Space
BestFS	Best First Search	Y	**N**	$O(min(N,B^{LMAX}))$	$O(min(N,B^{LMAX}))$

A few improvements to this algorithm can make things better

Steepest Ascent Hill Climbing can Look Stupid!



- Steepest Ascent Hill Climbing is clearly not guaranteed to find optimal [S→A→B→C→G: 6]
- Obvious question: What can we do to avoid the stupid mistake? : h(S) + c(S) cost so far

A* - The Basic Idea

- Steepest Ascent Hill Climbing: When you expand a node *n*, take each successor *n'* and place it on PriQueue with priority $h(n')$
- A*: When you expand a node *n*, take each successor *n'* and place it on PriQueue with priority

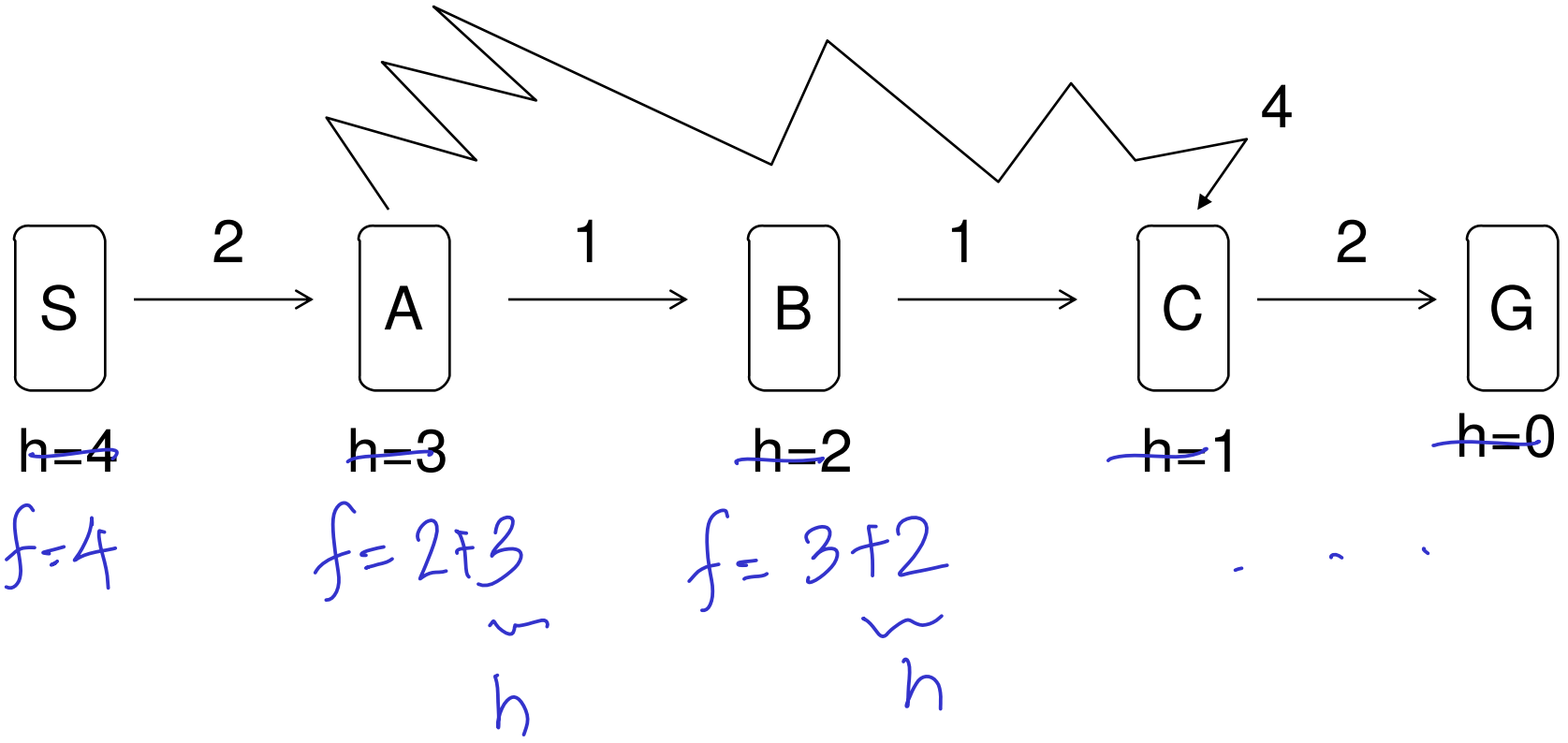
$$(\text{Cost of getting to } n') + h(n') \qquad (1)$$

$$\text{Let } g(n) = \text{Cost of getting to } n \qquad (2)$$

and then define…

$$f(n) = g(n) + h(n) \qquad (3)$$

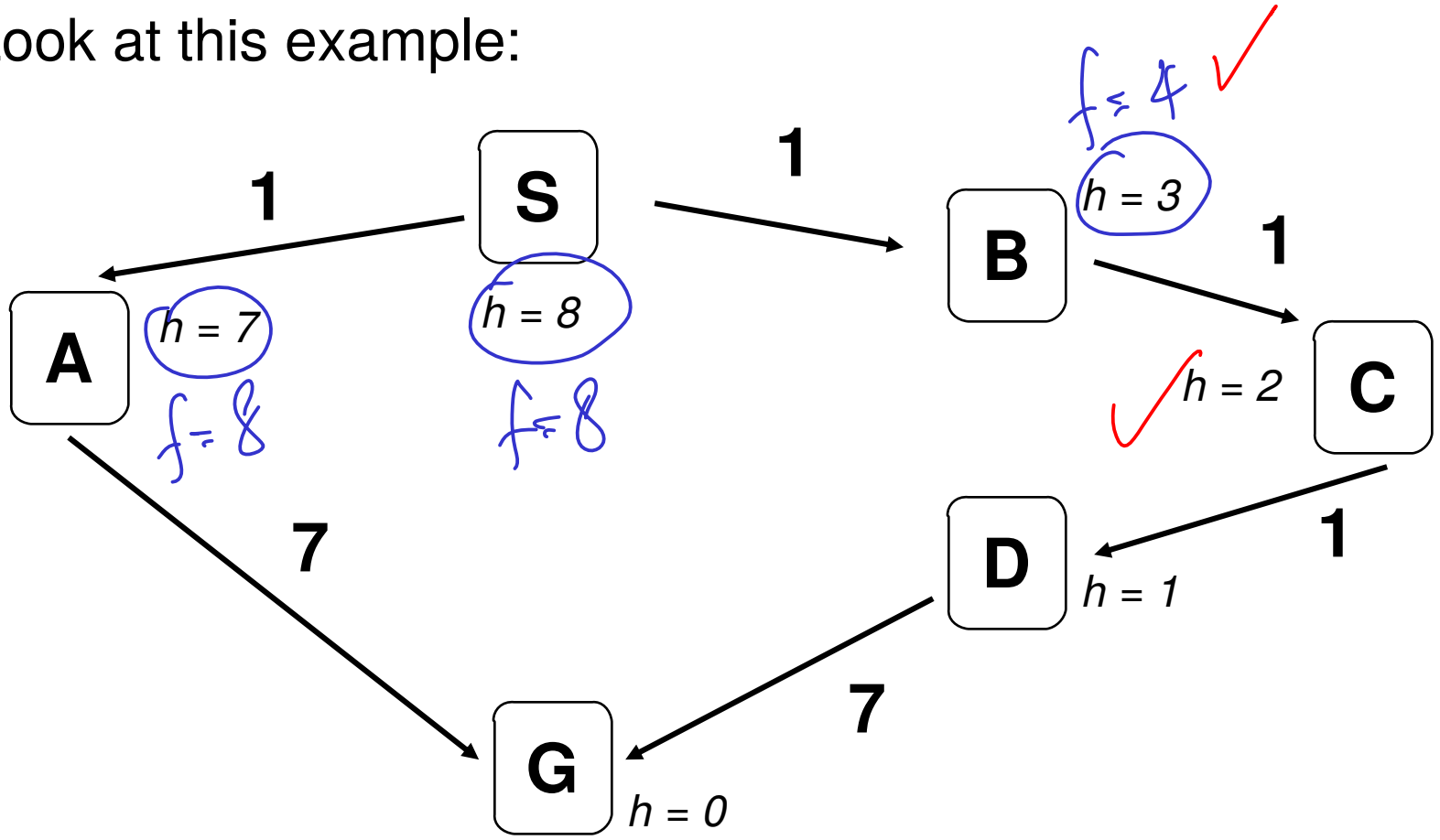
A* Looking Non-Stupid



When should A* terminate?

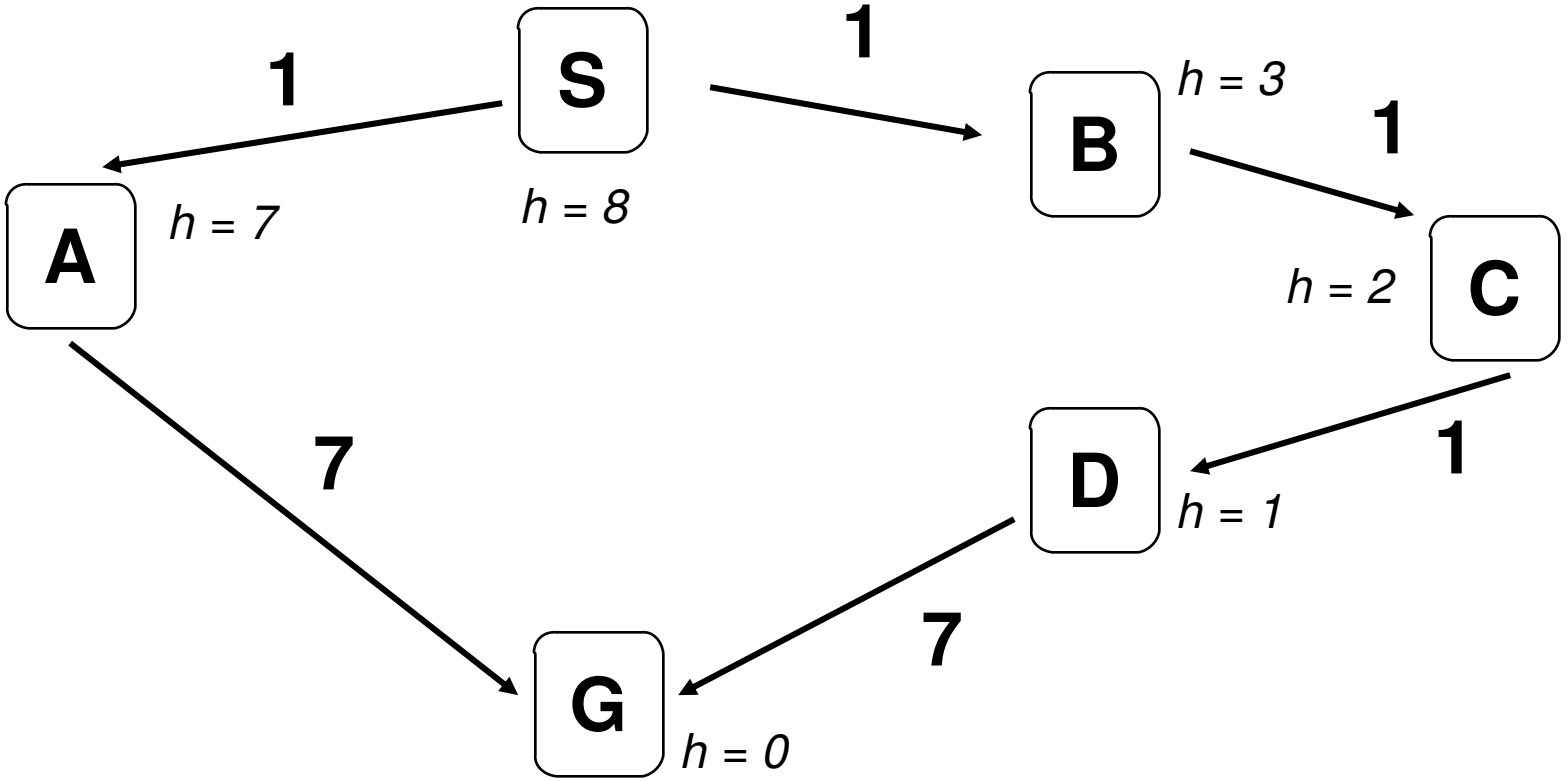
Idea: As soon as it generates a goal state?

Look at this example:



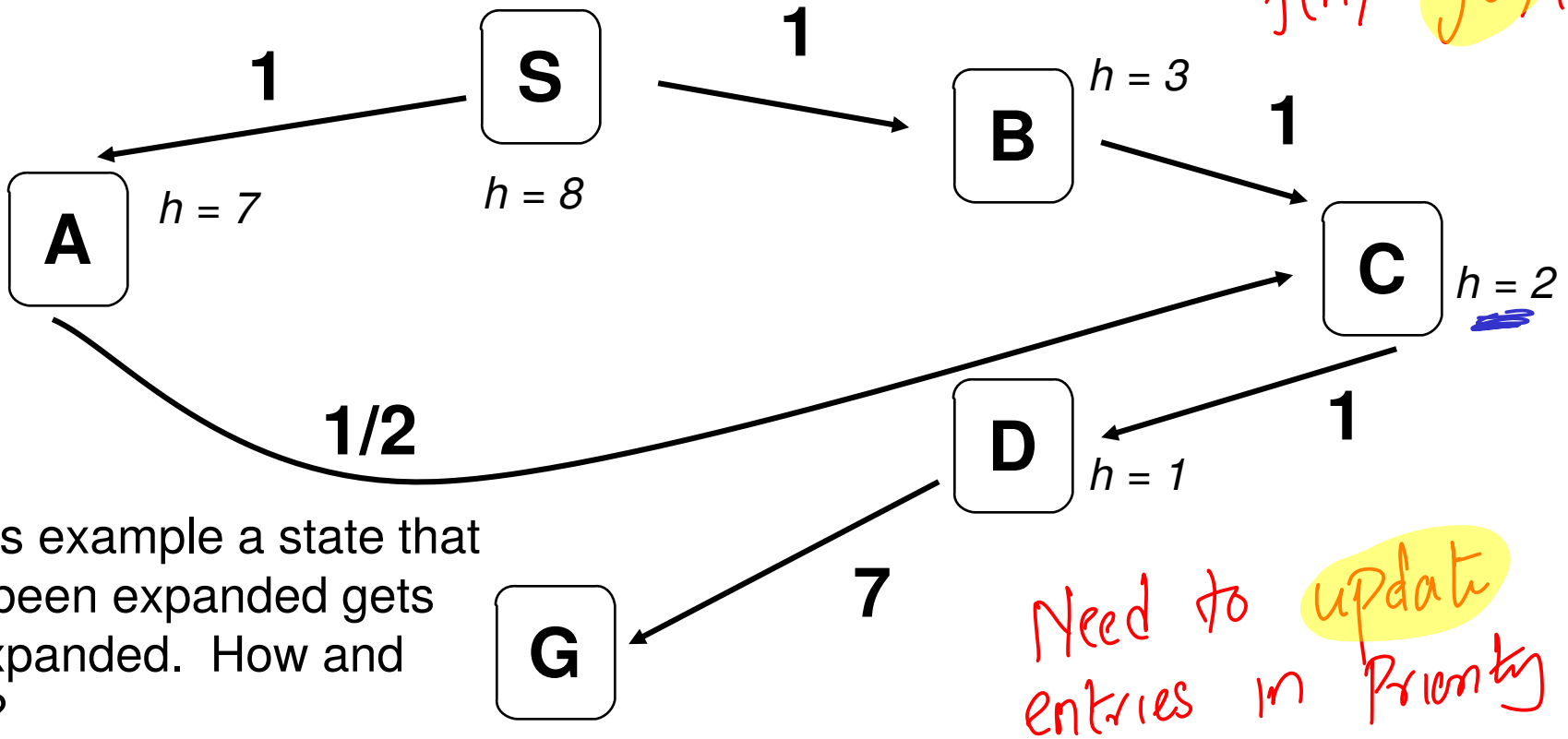
Correct A* termination rule:

A* Terminates Only When a Goal State Is Popped from the Priority Queue



A* revisiting states

Another question: What if A* revisits a state that was already expanded, and discovers a shorter path?



In this example a state that had been expanded gets re-expanded. How and why?

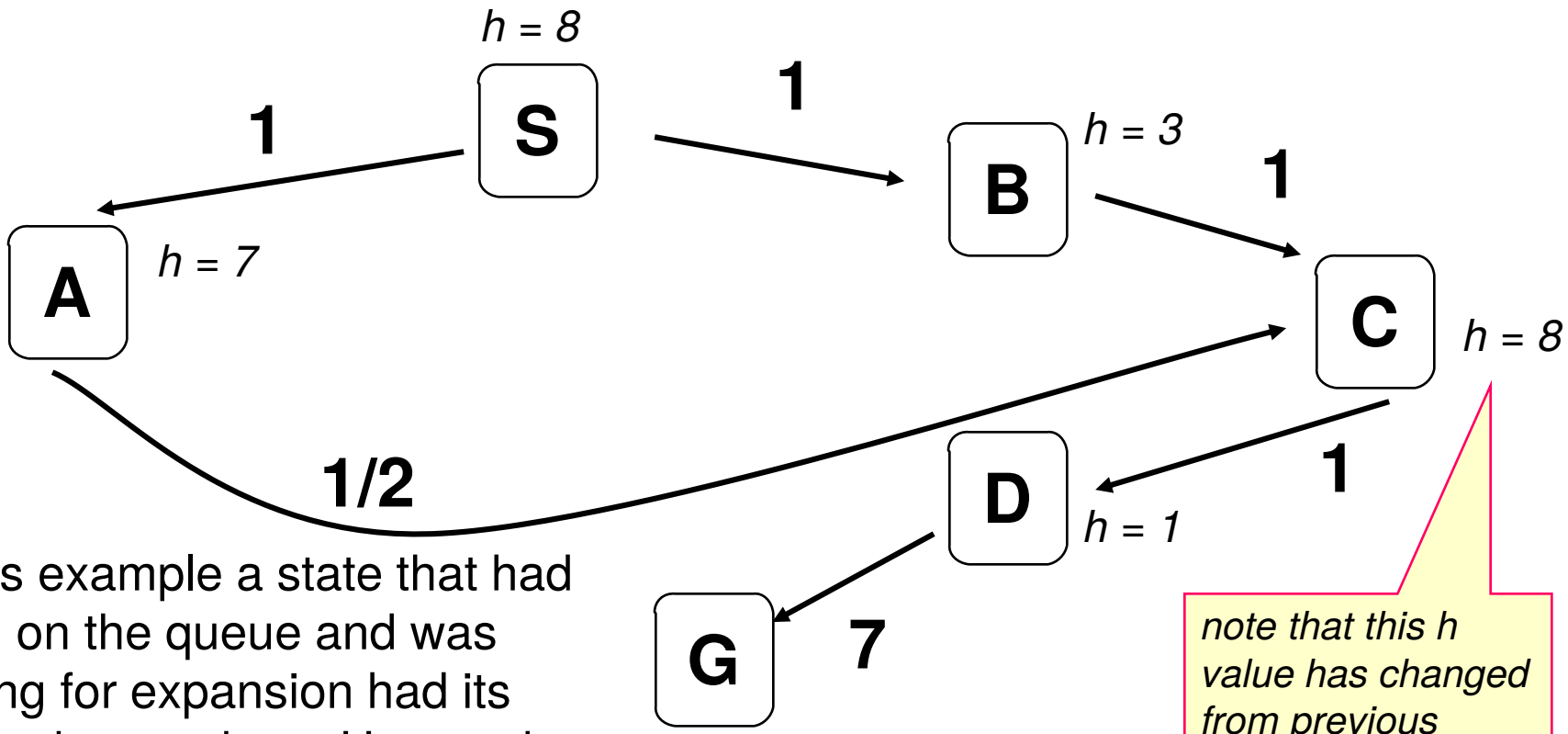
nodes with longer paths are ignorable visited

f(n) = g(n) + h(n)

Need to update entries in Priority Queue

A* revisiting states

What if A* visits a state that is already on the queue?



In this example a state that had been on the queue and was waiting for expansion had its priority bumped up. How and why?

The A* Algorithm

Reminder: $g(n)$ is cost of shortest known path to n

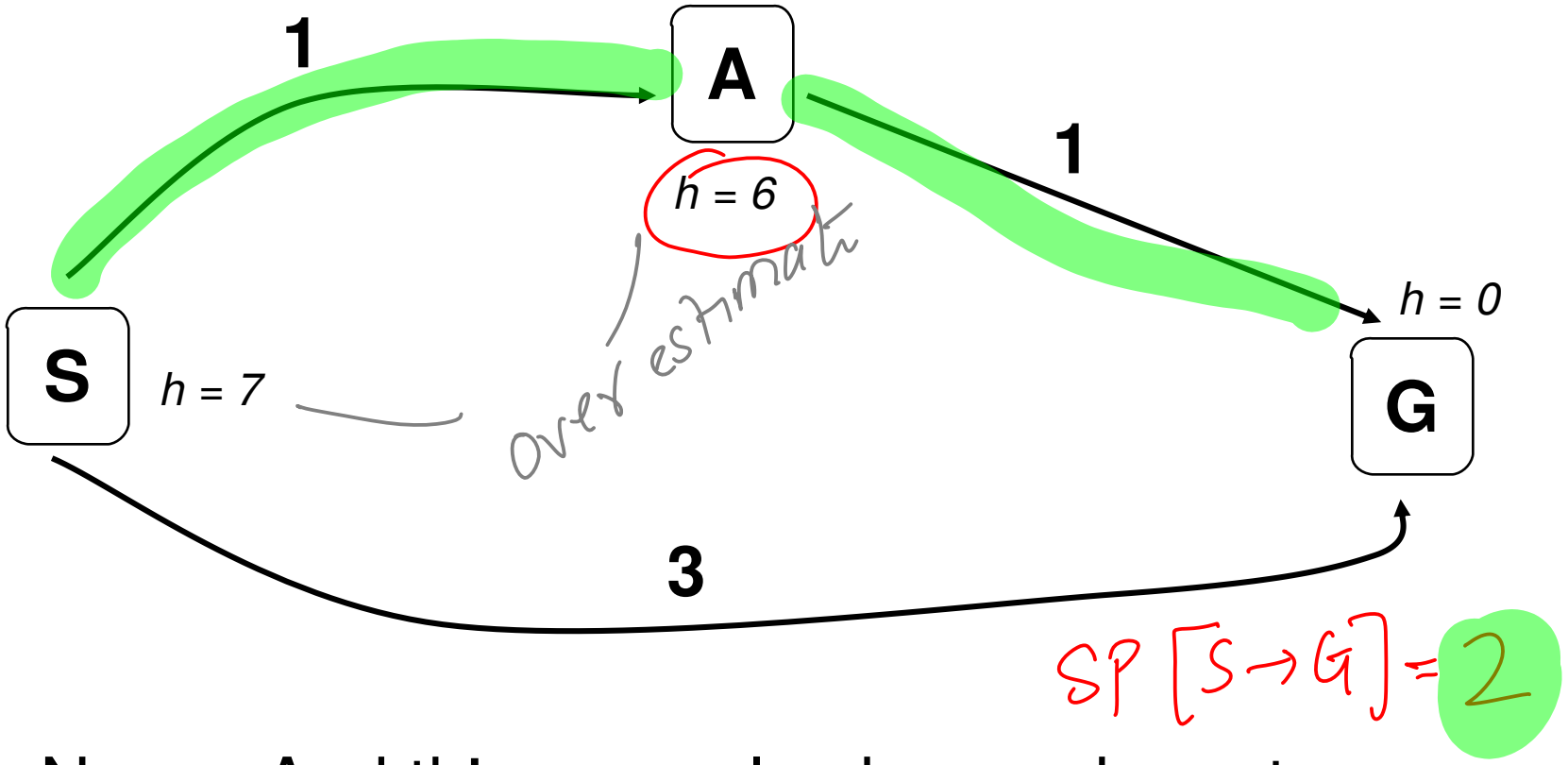
Reminder: $h(n)$ is a heuristic estimate of cost to a goal from n

- Priority queue *PQ* begins empty.
- *V* (= set of previously visited (*state*,*f*,*backpointer*)-triples) begins empty.
- Put *S* into *PQ* and *V* with priority $f(s) = g(s) + h(s)$
- Is *PQ* empty?
  - Yes? Sadly admit there's no solution
  - No? Remove node with lowest $f(n)$ from queue. Call it *n*.
  - If *n* is a goal, stop and report success.
  - "expand" *n* : For each n' in **successors**(n)….
    - Let $f' = g(n') + h(n') = g(n) + cost(n,n') + h(n')$
    - **If** *n'* not seen before, or *n'* previously expanded with $f(n') > f'$, or *n'* currently in *PQ* with $f(n') > f'$
    - **Then** Place/promote *n'* on priority queue with priority *f'* and update *V* to include (*state*=*n'*, *f '*, *BackPtr*=*n*).
    - **Else** Ignore *n'*

= $h(s)$ because $g(\text{start}) = 0$

use sneaky trick to compute $g(n)$

Is A* Guaranteed to Find the Optimal Path?

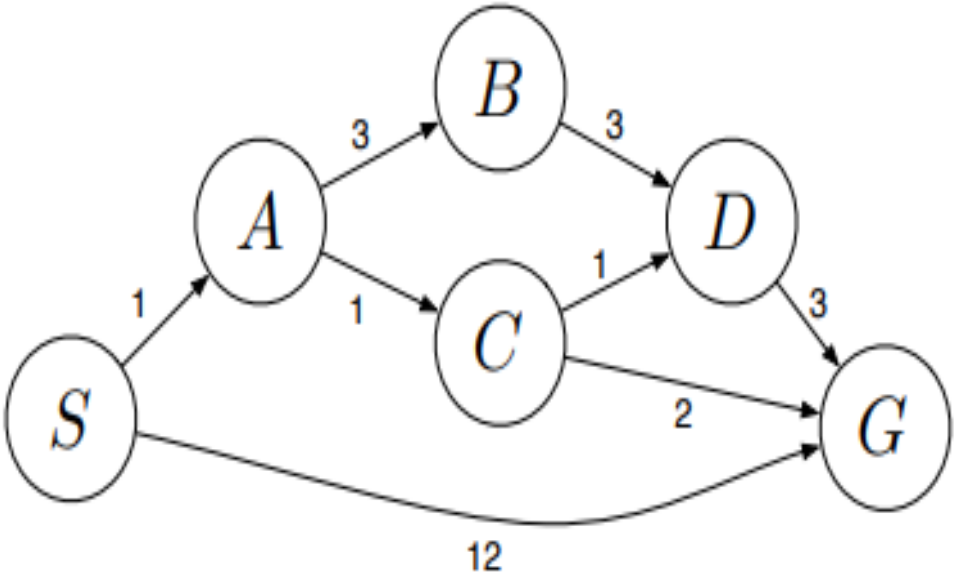


Nope. And this example shows why not.

Admissible Heuristics

- Write $h^*(n)$ = the true minimal cost to goal from n.
- A heuristic h is admissible if
  $h(n) <= h^*(n)$ for all states n.
- An admissible heuristic is guaranteed never to overestimate cost to goal.
- An admissible heuristic is optimistic.

A* with Admissible Heuristic Guarantees Optimal Path



State	h_1	h_2
S	5	4
A	3	2
B	6	6
C	2	1
D	3	3
G	0	0