Lecture 6

Analysis of A*, B&B Search Refinements

(Ch 3.7.1 - 3.7.4)

Lecture Overview

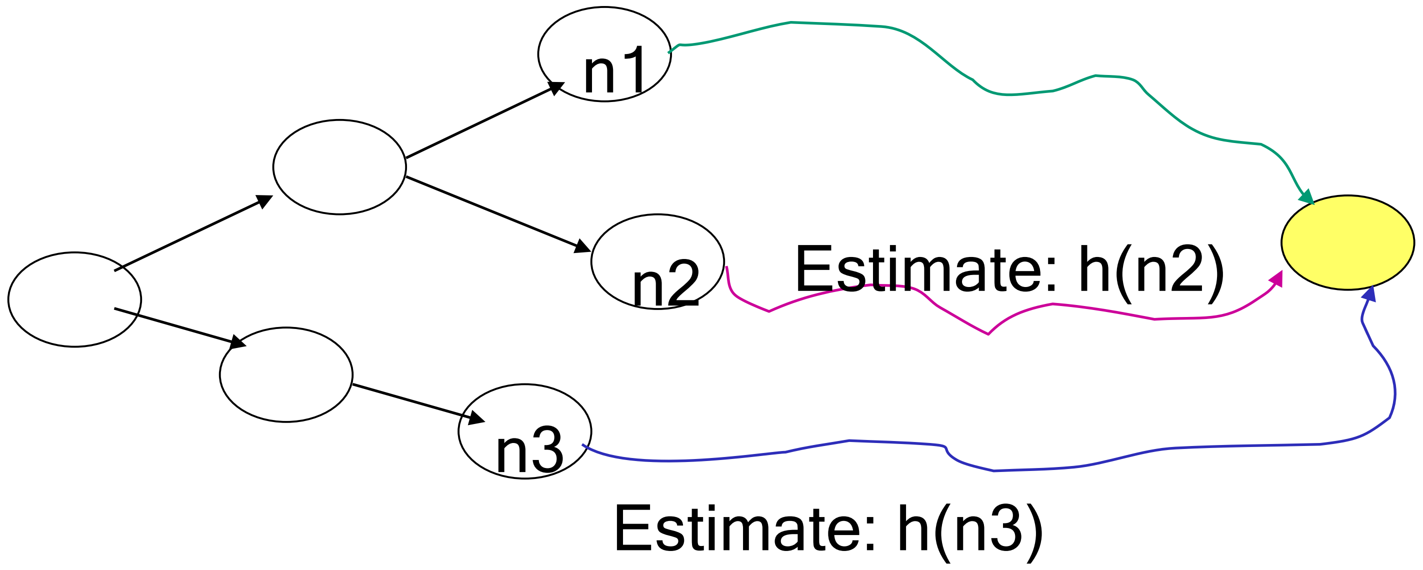
- ➡ Recap of previous lecture
- Analysis of A*
- Branch-and-Bound

- Cycle checking, multiple path pruning
- Stored Graph - Dynamic Programming

How to Make Search More Informed?

Def.: A search heuristic h(n)is an estimate of the cost of the optimal (cheapest) path from node nto a goal node.

Estimate: h(n1)

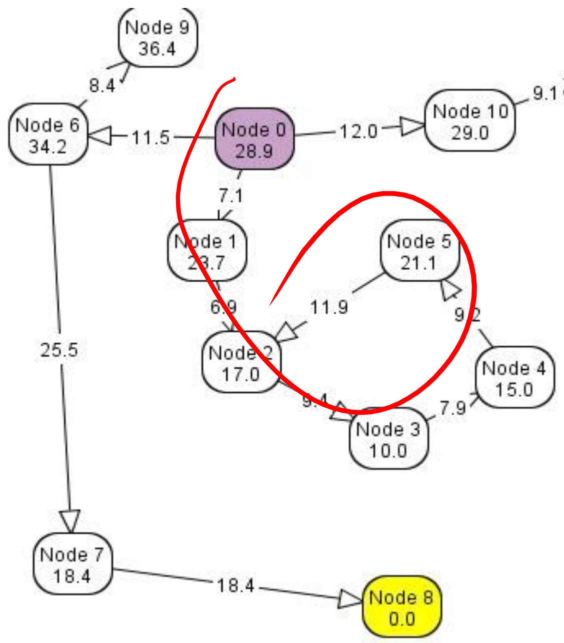


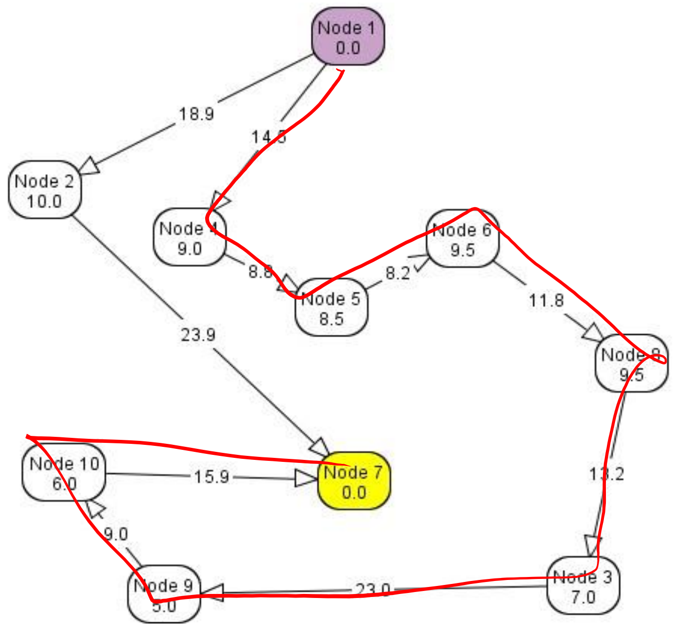
- hcan be extended to paths: h(n$_0$,…,n$_k$)=h(n$_k$)
- h(n)should leverage readily obtainable information (easy to compute) about a node.

Best First Search (BestFS)

- Always choose the path on the frontier with the smallest h value.
- BestFS treats the frontier as a priority queue ordered by h.
- Can get to the goal pretty fast if it has a good h but…

  It is not complete, nor optimal





- still has time and space worst-case complexity of O(b^m)

Learning Goal for Search

Apply basic properties of search algorithms:

- completeness, optimality, time and space complexity

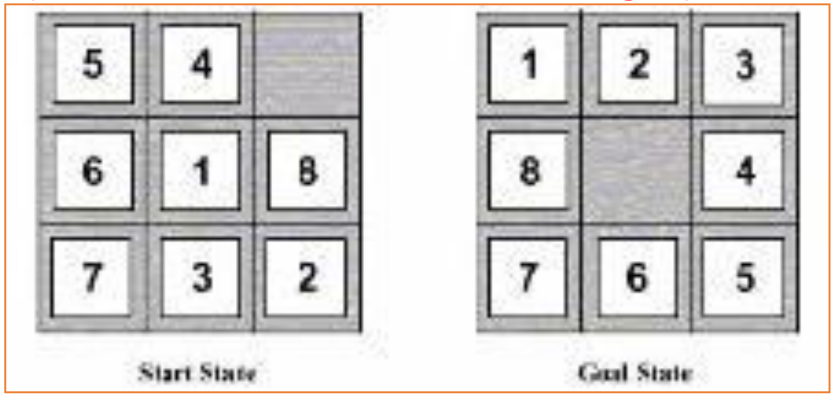
	Complete	Optimal	Time	Space
DFS	N (Y if no cycles)	N	$O(b^m)$	$O(mb)$
BFS	Y	Y	$O(b^m)$	$O(b^m)$
IDS	Y	Y	$O(b^m)$	$O(mb)$
LCFS (when arc costs available)	Y Costs > ε	Y Costs >=0	$O(b^m)$	$O(b^m)$
Best First (when havailable)	N	N	$O(b^m)$	$O(b^m)$

▭ uninformed 🟦 Uninformed but using arc cost 🟨 Informed (goal directed)

Remind definition of admissible… Example 3: Eight Puzzle

- Another possible h(n):

  Sum of number of moves between each tile's current position and its goal position (we can move over other tiles in the grid)

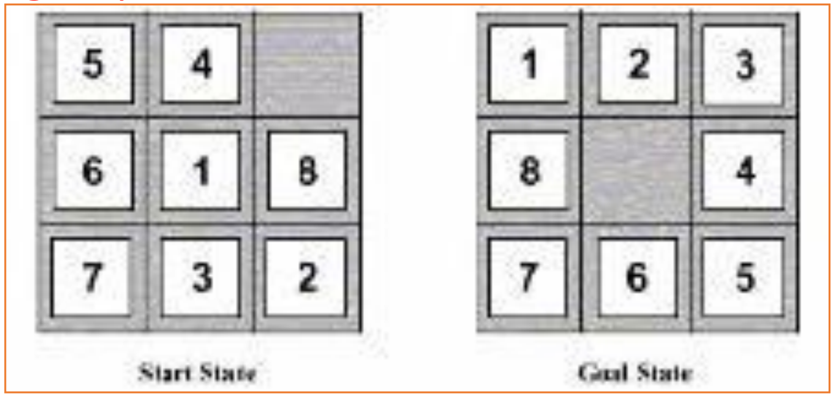


1	2	3	4	5	6	7	8

Sum (

Example 3: Eight Puzzle

- Another possible h(n):
  Sum of number of moves between each tile's current position and its goal position



$\boxed{1}$ $\boxed{2}$ $\boxed{3}$ $\boxed{4}$ $\boxed{5}$ $\boxed{6}$ $\boxed{7}$ $\boxed{8}$

sum (2 3 3 2 4 2 0 2) = 18

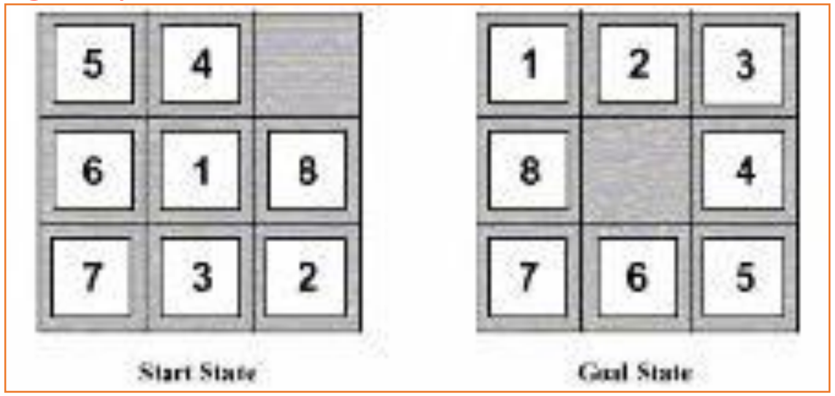
Admissible?

A. Yes B. No C. It depends

Example 3: Eight Puzzle

- Another possible h(n):

  Sum of number of moves between each tile's current position and its goal position



	1	2	3	4	5	6	7	8	
sum	(2	3	3	2	4	2	0	2)	= 18

Admissible? YES! One needs to make at least as many moves to get to the goal state when constrained by the grid structure

How can we effectively use h(n)

Maybe we should combine it with the cost. How?
Shall we select from the frontier the path pwith:

A. ~~Lowest cost(p) – h(p)~~ ←

B. Highest cost(p) – h(p) ←

C. Highest cost(p)+h(p) ←

[✓ D. Lowest cost(p)+h(p)] ← A*

Lecture Overview

• Recap of previous lecture • Analysis of A* • Branch-and-Bound • Cycle checking, multiple path pruning • Stored Graph - Dynamic Programming

Slide 11

A* Search Algorithm

- A* is a mix of:
- lowest-cost-first and
- best-first search

- A* treats the frontier as a priority queue ordered by f(p)= $g(p) + h(p)$

- It always selects the node on the frontier with the lowest f _______________ estimate~~d~~ total ________________________ distance.

Analysis of A*

If the heuristic is completely uninformative and the edge costs are all the same, A* is equivalent to...

A. [✓ BFS]
B. [✓ LCFS]
C. ~~DFS~~
D. None of the Above

Analysis of A^*

Let's assume that arc costs are strictly positive.

- Time complexity is $O(b^m)$
- the heuristic could be completely uninformative and the edge costs could all be the same, meaning that A^* does the same thing as...

BFS

- Space complexity is $O(b^m)$ like ……, A^* maintains a frontier which grows with the size of the tree

- Completeness: $\boxed{\text{yes}}$.

- $\boxed{\text{Optimality: ??}}$

Optimality of A*

If A* returns a solution, that solution is guaranteed to be optimal, as long as

When

- the branching factor is finite
- arc costs are strictly positive
- h(n)is an underestimate of the length of the shortest path from nto a goal node, and is non-negative

Theorem

If A^* selects a path p as the solution, p is the shortest (i.e., lowest-cost) path.

CPSC 322, Lecture 8

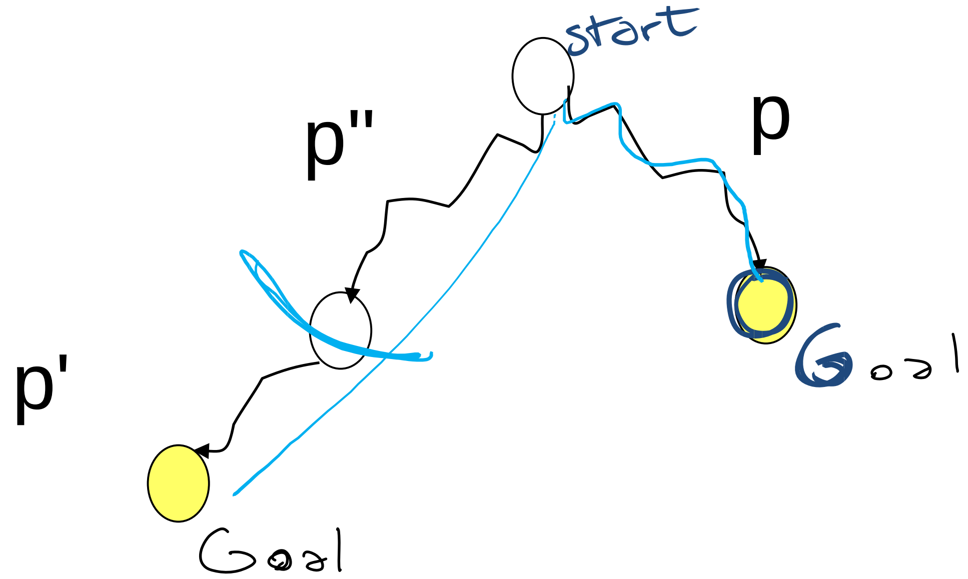
Why is A^* optimal?

$c(p) > c(p')$

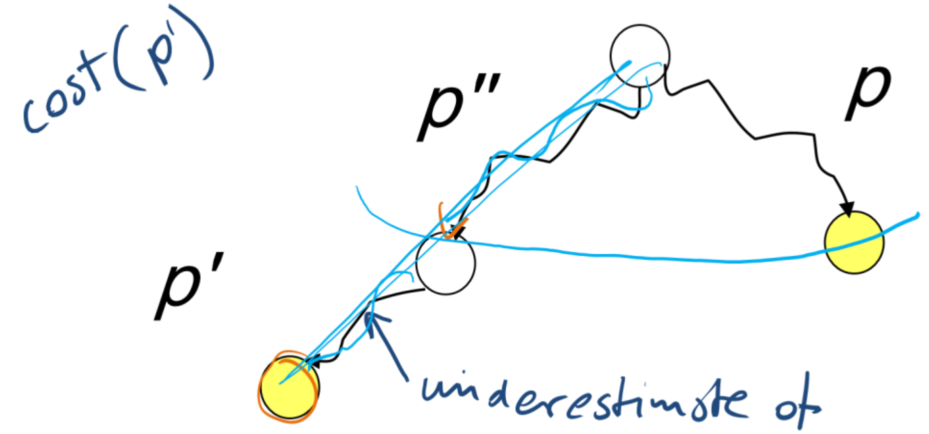
- A* returns p
- Assume for contradiction that some other path p' is actually the shortest path to a goal

- Consider the moment when pis chosen from the frontier. Some part of path p'will also be on the frontier; let's call this partial

path p''.



Why is A* optimal? (cont')



$c(p') < c(p)$

$f(p) \le f(p'')$

- Because *p* was expanded before *p''*, $c(p) + h(p) \le c(p'') + h(p'')$
- Because *p* is a goal, $h(p) = 0$. Thus $c(p) \le c(p'') + h(p'')$
- Because *h* is admissible, *cost(p'') + h(p'')* $\le c(p')$ for any path *p'* to a goal that extends *p''*
- Thus $c(p) \le c(p')$ for any other path *p'* to a goal.

This contradicts our assumption that p'is the shortest path.

Optimal efficiency of A*

- In fact, we can prove something even stronger about A*: in a sense (given the particular heuristic that is available) no search algorithm could do better!

- Optimal Efficiency: Among all optimal algorithms that start from the same start node and use the same heuristich, A*expands the minimal number of paths.

Samples A* applications

- An Efficient A* Search Algorithm For Statistical Machine Translation. 2001
- The Generalized A* Architecture. Journal of Artificial Intelligence Research (2007)
- Machine Vision … Here we consider a new compositional model for finding salient curves.
- Factored A*search for models over sequences and trees International Conference on AI. 2003….

It starts saying… The primary challenge when using A* search is to find heuristic functions that ~~simultaneously are admissible, close to actual~~ completion costs, and efficient to calculate… applied to NLP and BioInformatics

(Natural Language Processing)

Samples A* applications (cont')

Aker, A., Cohn, T., Gaizauskas, R.: Multi-document summarization using A* search and discriminative training. Proceedings of the 2010 Conference on Empirical Methods in Natural Language

Processing.. ACL (2010)

Samples A* applications (cont')

EMNLP 2014 A* CCG Parsing with a Supertagfactored Model M. Lewis, M. Steedman

We introduce a new CCG parsing model which is factored on lexical category assignments. Parsing is then simply a deterministic search for the most probable category sequence that supports a CCG derivation. The parser is extremely simple, with a tiny feature set, no POS tagger, and no statistical model of the derivation or dependencies. Formulating the model in this way allows a highly effective heuristic for A∗ parsing, which makes parsing extremely fast. Compared to the standard C&C CCG parser, our model is more accurate out-of-domain, is four times faster, has higher coverage, and is greatly simplified. We also show that using our parser improves the performance of a state-of-the-art question answering system

Follow up ACL 2017 (main NLP conference – was held in Vancouver in August!) A* CCG Parsing with a Supertag and Dependency Factored

Model Masashi Yoshikawa, Hiroshi Noji, Yuji Matsumoto

A* advantages

What is a key advantage of A* ?

A. Does not need to consider the cost of the paths

B. Has a linear space complexity

C. It is often optimal

[✓ D.] None of the above

Lecture Overview

• Recap of previous lecture • Analysis of A* • Branch-and-Bound • Cycle checking, multiple path pruning • Stored Graph - Dynamic Programming

Branch-and-Bound Search

- Biggest advantages of A*...

$f = c + h$

- What is the biggest problem with A*?

b^m

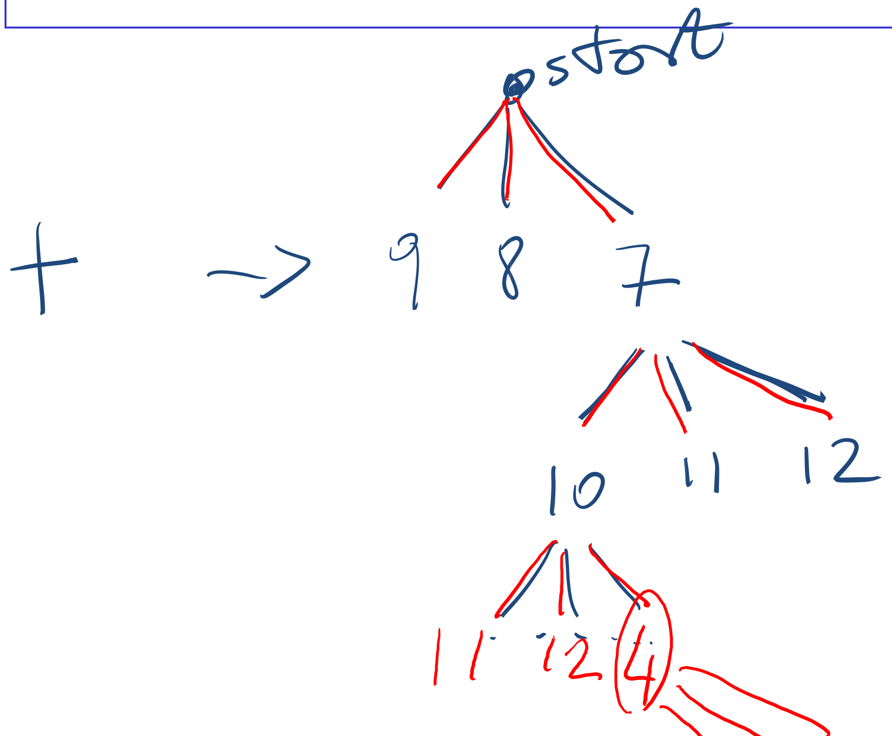
- Possible, preliminary Solution:

f

Branch-and-Bound Search Algorithm

- Follow exactly the same search path as depth-first search
  - treat the frontier as a stack: expand the most-recently added path first
  - the order in which neighbors are expanded can be governed by some arbitrary node-ordering heuristic

we can use $f = c + h$



Once this strategy has found a solution...

What should it do next ?

A. Keep running DFS, looking for deeper solutions?

[✓ B.] Stop and return that solution

C. Keep searching, but only for shorter solutions

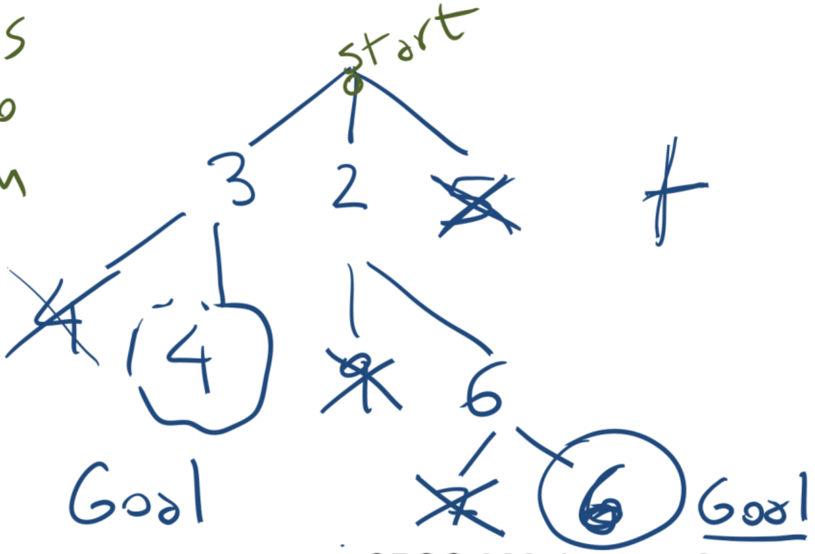
D. None of the above

Branch-and-Bound Search Algorithm

- Keep track of a lower bound and upper bound on solution cost at each path
- lower bound: LB(p) = f(p) = cost(p) + h(p)
- upper bound:UB = cost of the best solution found so far.
  - ✓ if no solution has been found yet, set the upper bound to .
- When a path pis selected for expansion:
- if LB(p) UB, remove pfrom frontier without expanding it (pruning)

- else expand p, adding all of its neighbors to the frontier

The numbers correspond to f for the path from start to that node



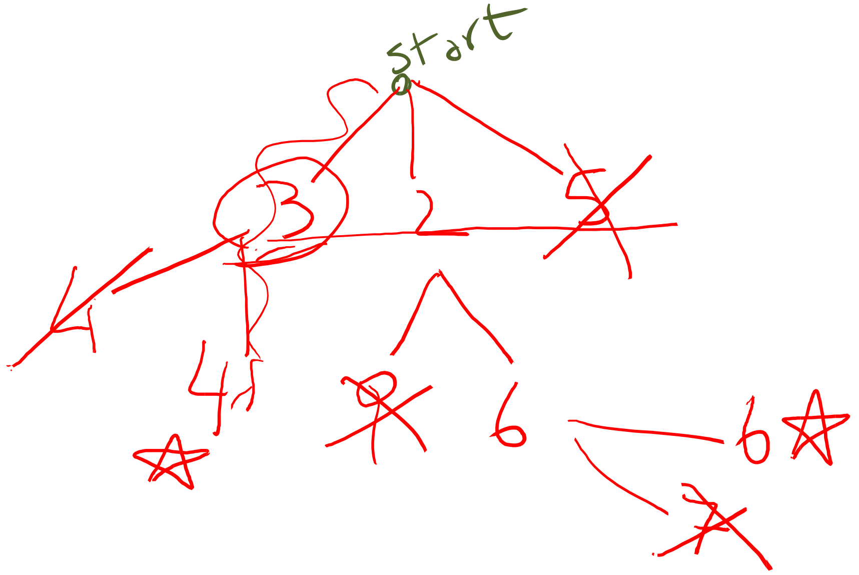
UB = ∞
6
4

Same for all paths at any given time

Branch-and-Bound Search Algorithm

- Keep track of a lower bound and upper bound on solution cost at each path
- lower bound: LB(p) = f(p) = cost(p) + h(p)
- upper bound:UB = cost of the best solution found so far.

- ✓ if no solution has been found yet, set the upper bound to □.
- When a path pis selected for expansion:
- if LB(p) □UB, remove pfrom frontier without expanding it (pruning)
- else expand p, adding all of its neighbors to the frontier



$UB = \infty$

6

4

•Arc cost = 1
•

Before expanding a path p,

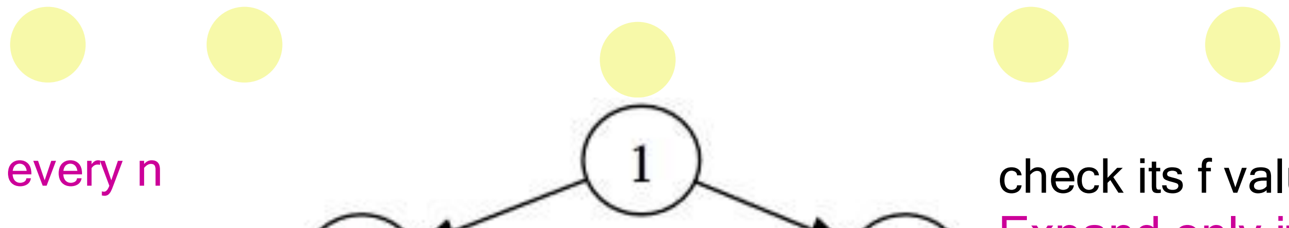
JUST TO SIMPLIFY
THE EXAMPLE

h(n) = 0 for every n

check its f value f(p):

- Arc cost = 1
-

Before expanding a path p,



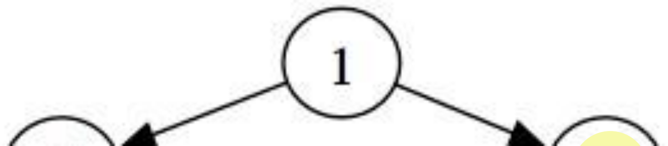
h(n) = 0 for every n

check its f value f(p):
Expand only if f(p) < UB

- Arc cost = 1
-

Before expanding a path p,

h(n) = 0 for every n



check its f value f(p):

Expand only if f(p) < UB

Solution!

•h(n) = 0 for every n

•

•UB = 3

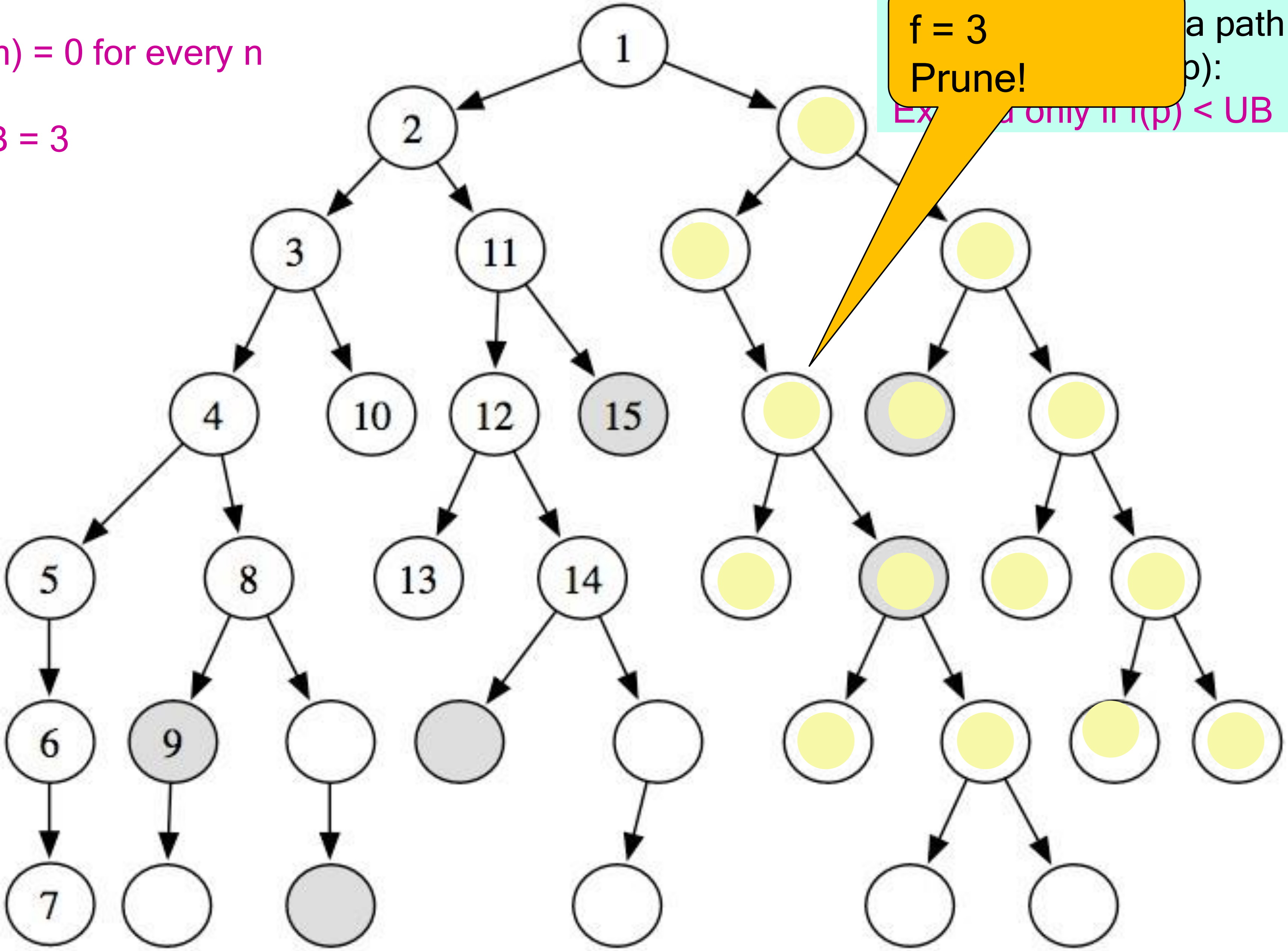
a path p,

p):

Ex d only if f(p) < UB

f = 3

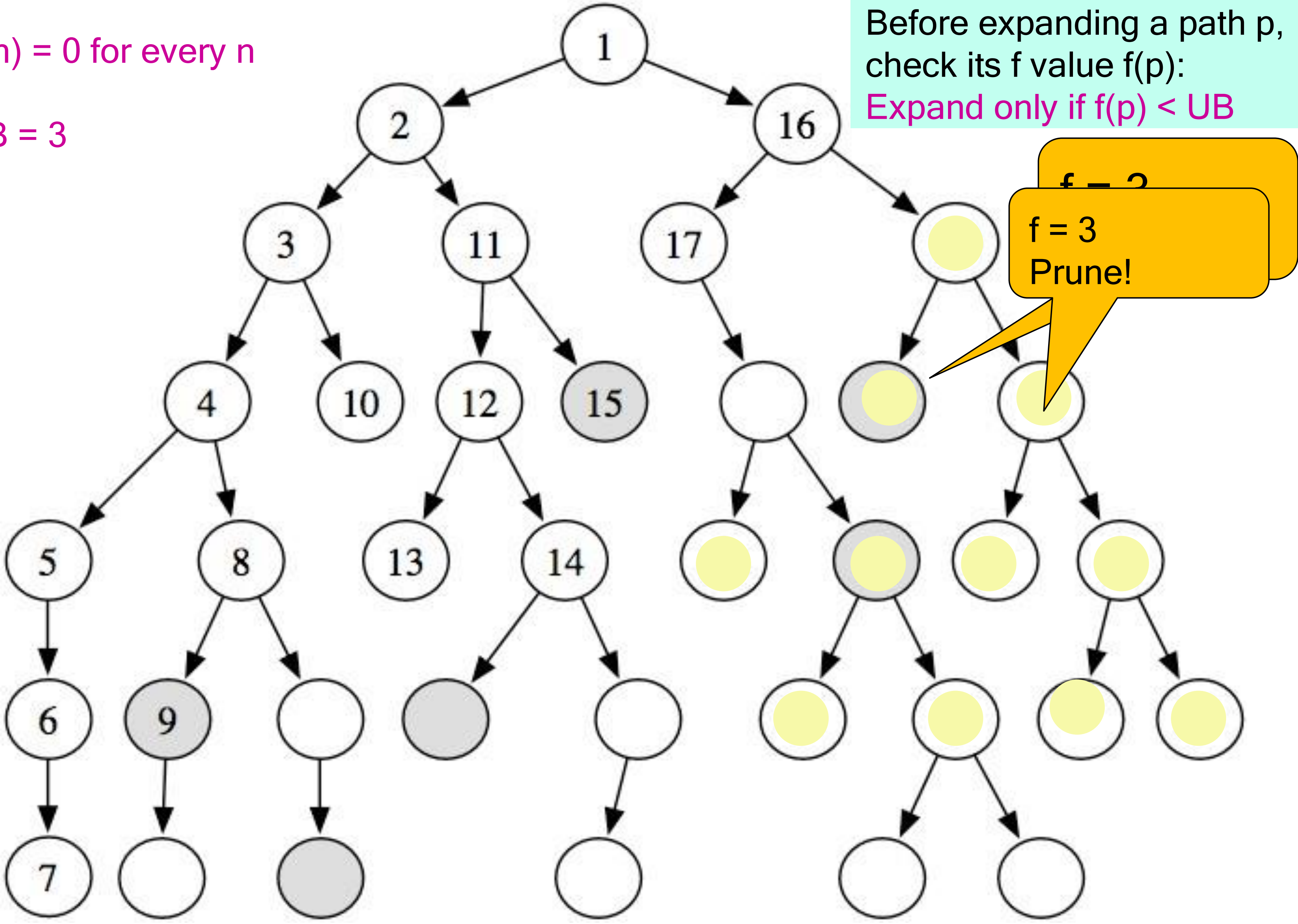
Prune!



h(n) = 0 for every n

•UB = 3

Before expanding a path p, check its f value f(p):
Expand only if f(p) < UB



•Arc cost = 1
•

Branch-and-Bound Analysis

- Completeness: ___________
- however, for many problems of interest ___________

- Time complexity: $\underline{O(b^m)}$
- Space complexity:___ bm ______________
- Branch & Bound has the same space complexity as___DFS______.

A^*

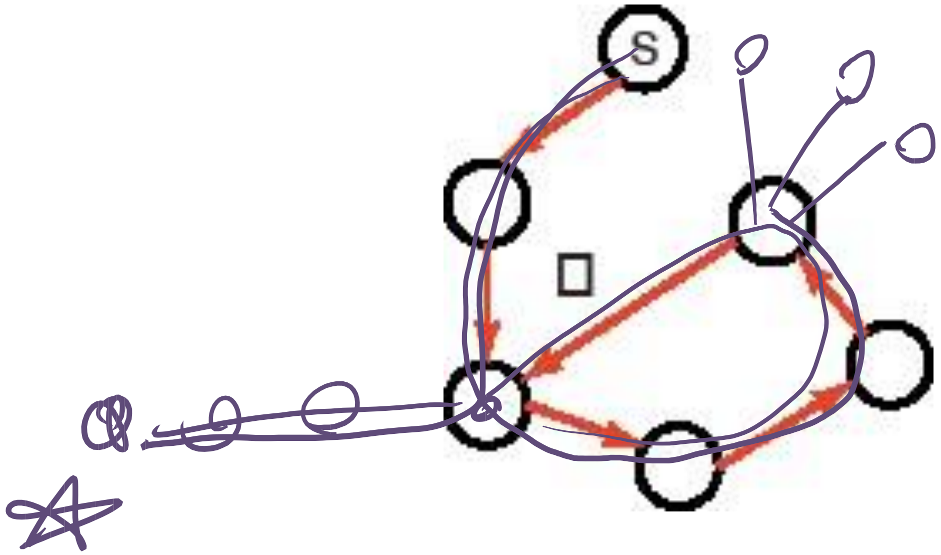
- this is a big improvement over ______________________________!
- Optimality: __Y__________________________

Lecture Overview

• Recap of previous lecture • Analysis of A* • Branch-and-Bound • Cycle checking, multiple path pruning • Stored Graph - Dynamic Programming

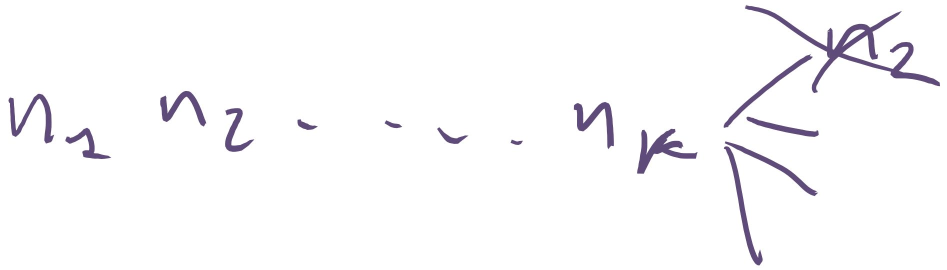
Slide 35

Cycle Checking



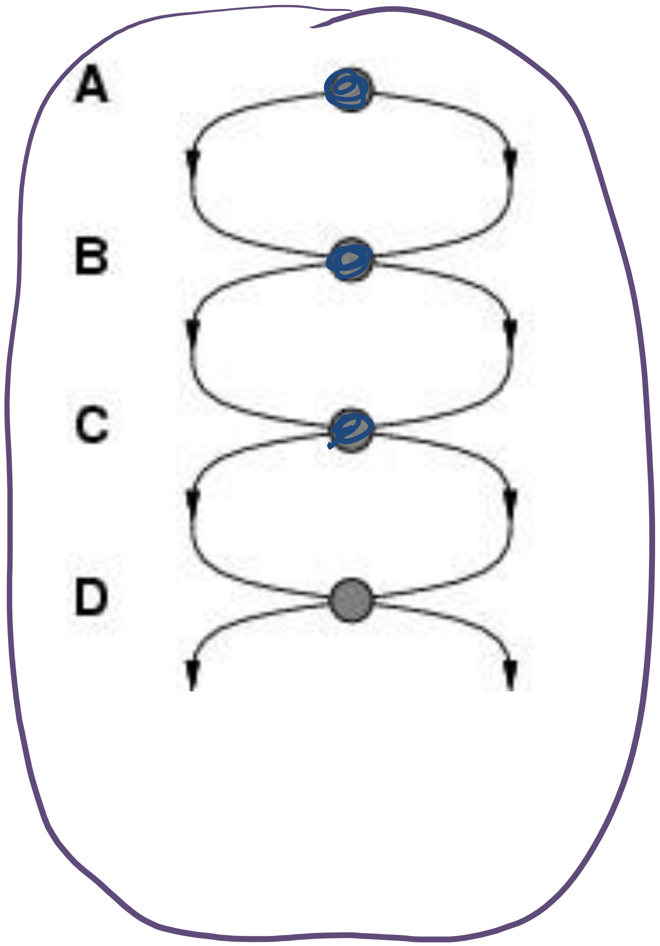
You can prune a path that ends in a node already on the path. This pruning cannot remove an optimal solution.

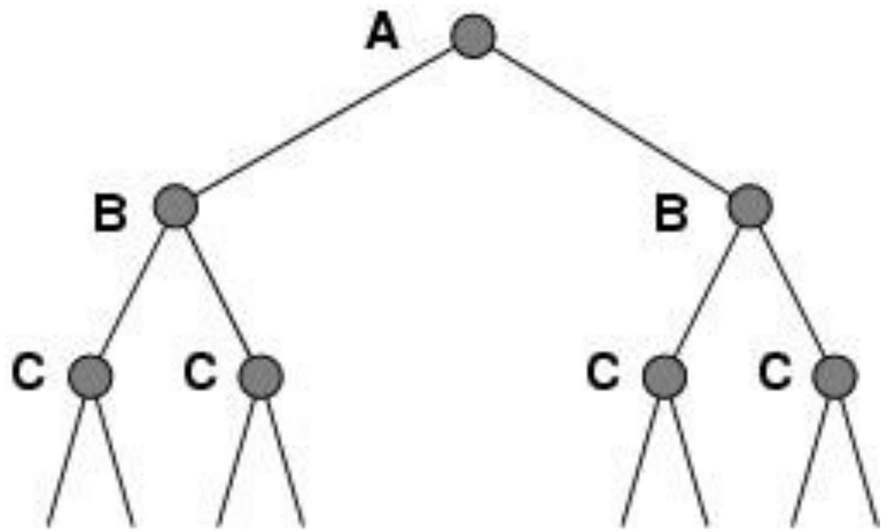
- The time for checking is ________________________________ in path length.



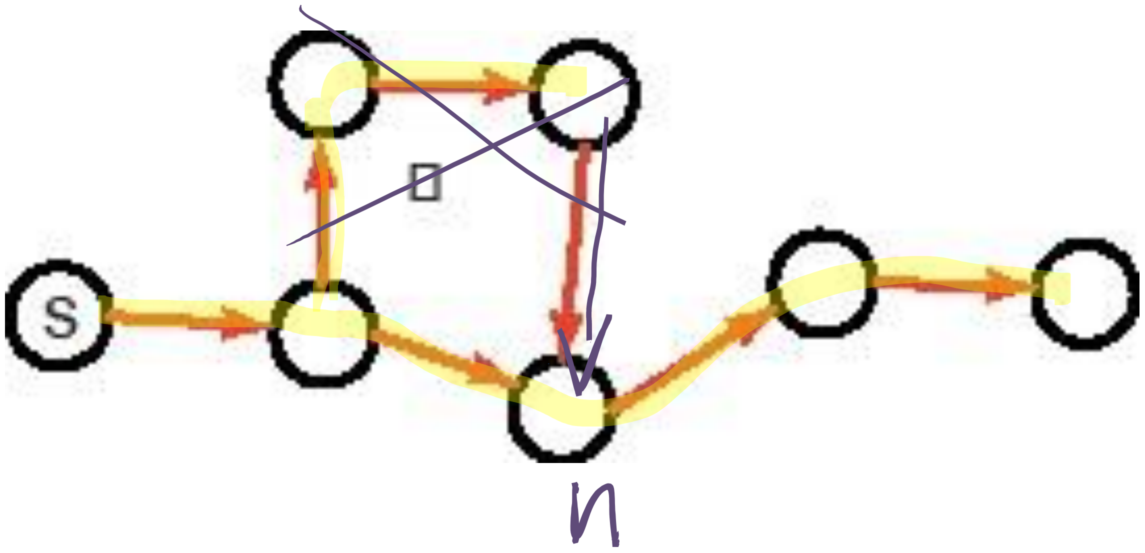
Repeated States / Multiple Paths

Failure to detect repeated states can turn a linear problem into an exponential one!





Multiple-Path Pruning



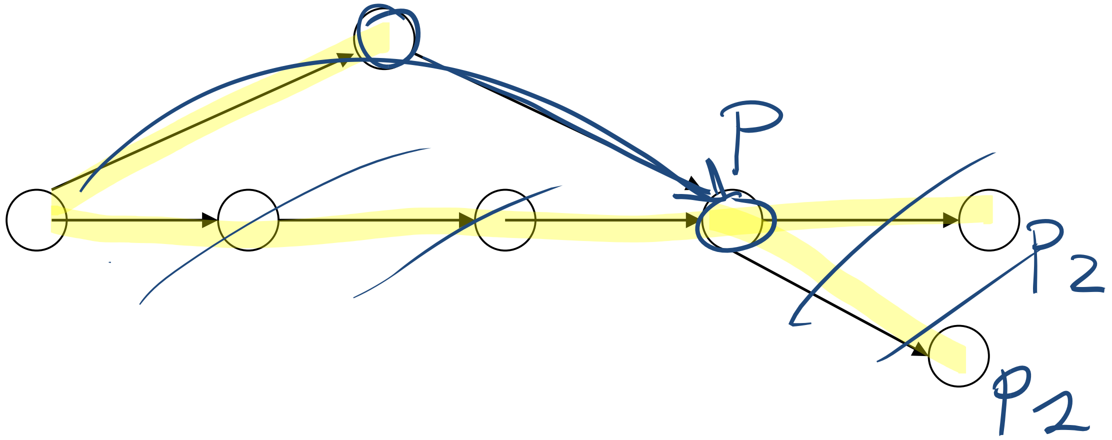
•You can prune a path to node n that you have already found a path to

• (if the new path is longer – more costly).

Multiple-Path Pruning & Optimal Solutions

Problem: what if a subsequent path to nis shorter than the first path to n ?

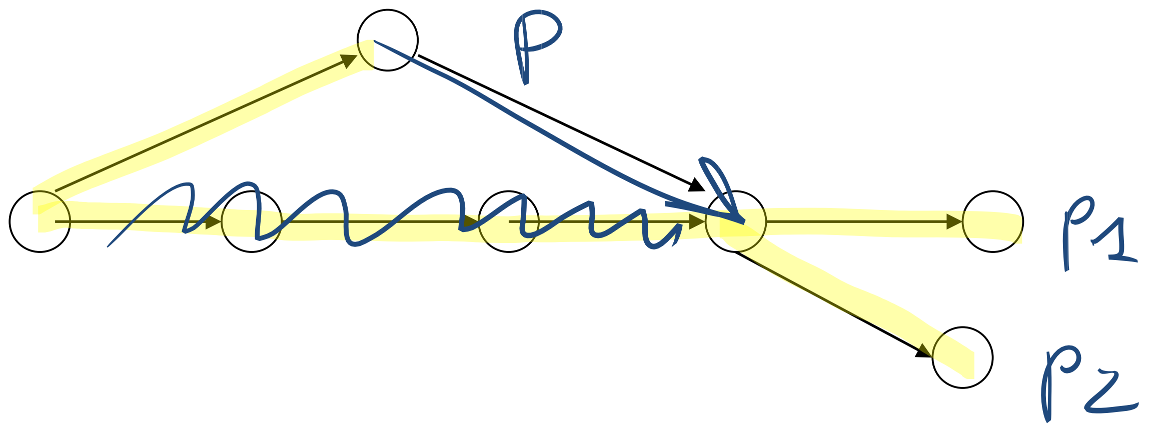
- You can remove all paths from the frontier that use the longer path. (as these can't be optimal)



Multiple-Path Pruning & Optimal Solutions

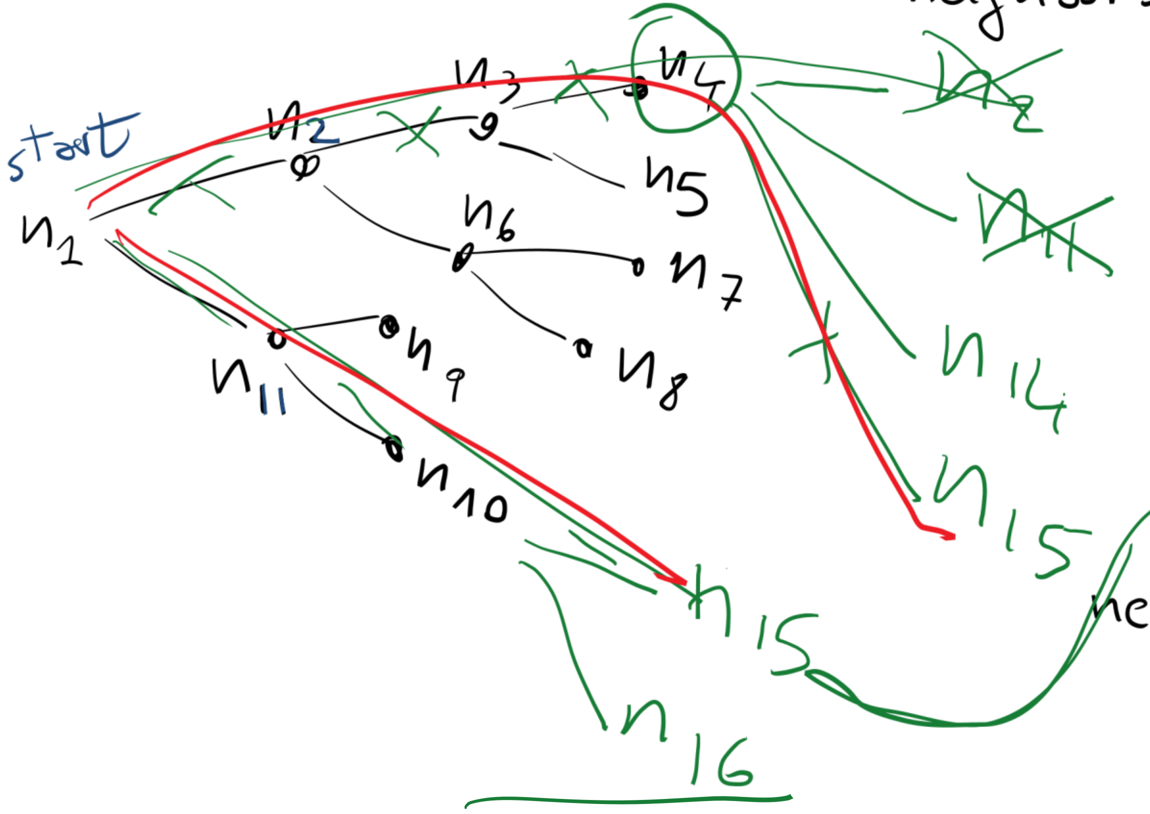
Problem: what if a subsequent path to nis shorter than the first path to n ?

- You can change the initial segment of the paths on the frontier to use the shorter path.



Example

Pruning Cycles   Repeated States



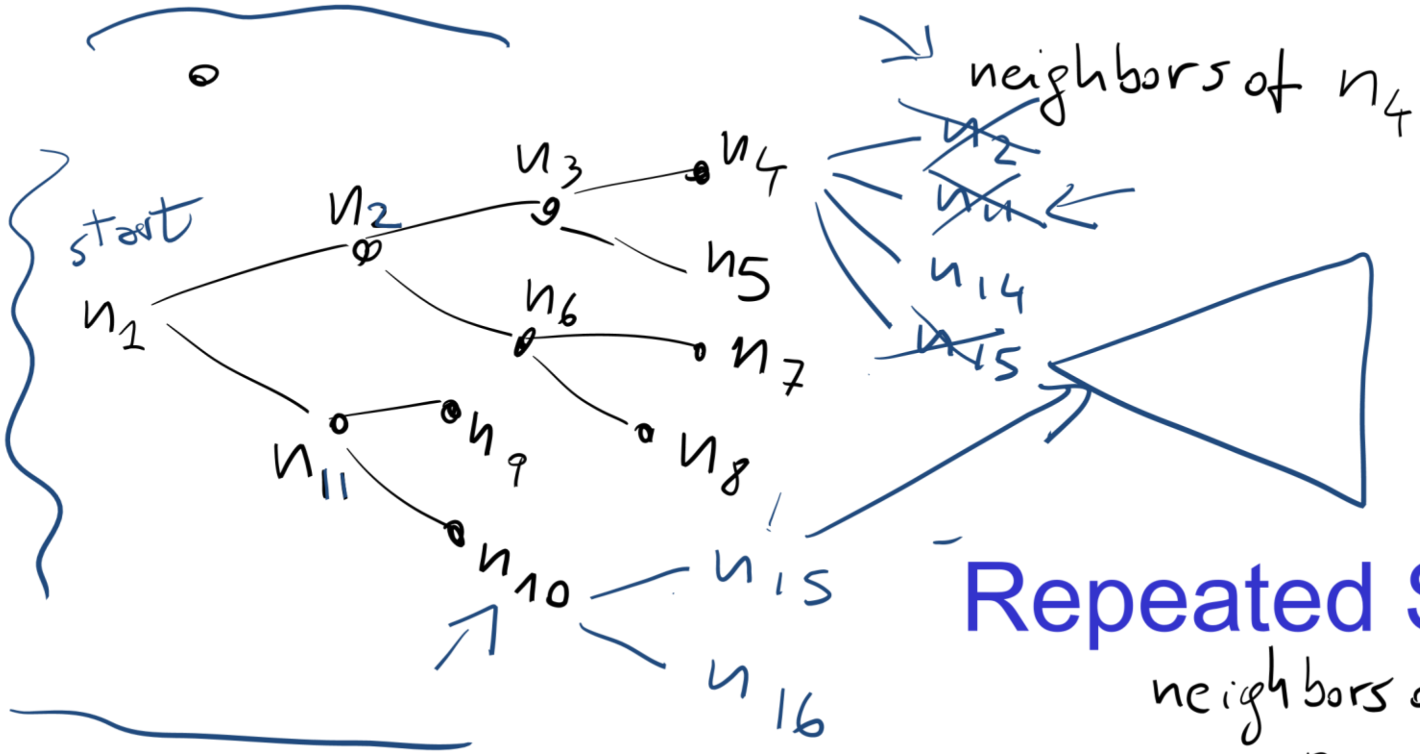
neighbors of $n_4 = \{n_2, n_{11}, n_{14}, n_{15}\}$

neighbors of $n_{10} = \{n_{15}, n_{16}\}$

Example

Pruning Cycles

neighbors of $n_4 = \{n_2, n_{11}, n_{14}, n_{15}\}$



Repeated States

neighbors of $n_{10} =$

$\{n_{15}, n_{16}\}$

Lecture Overview

Dynamic Programming

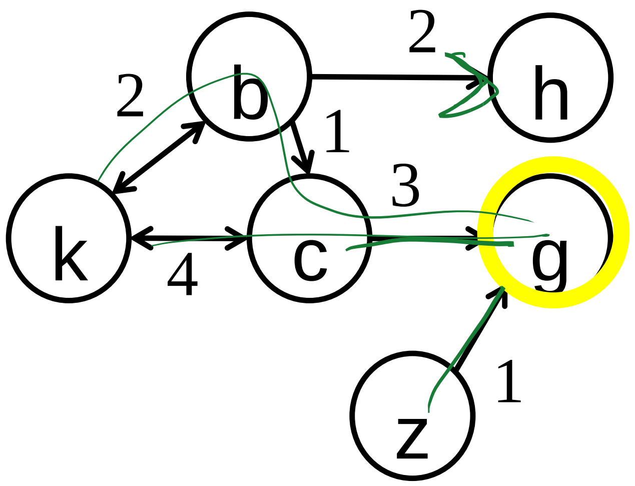
• Recap of previous lecture • Analysis of A* • Branch-and-Bound • Cycle checking, multiple path pruning ➡ • Stored Graph - Dynamic Programming

Dynamic Programming

- Idea: for statically stored graphs, build a table of dist(n):
- The actual distance of the shortest path from node n to a goal g
- This is the perfect__________________________.

Dynamic Programming

- dist(g) = 0
- dist(z) = 1
- dist(c) = 3
- dist(b) = 4 6
- dist(k) = ? ∞
- dist(h) = ?



- How could we implement that?

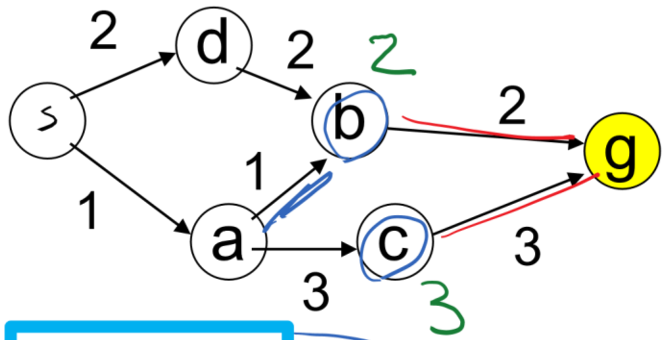
This can be built backwards from the goal:

□□

Dynamic Programming

allthe neighbors m

$$dist(n) = \begin{cases} 0 & if \quad is\_goal(n), \\ \min_{\langle n,m\rangle \in A}\left[(\text{cost}(n,m) + dist(m))\right] & otherwise \end{cases}$$



i▸clicker

dist(a)

A. *min(3,3)*

B. *min(6,3)*

$dist(g)$ 0

$dist(b)$ $\min[(2+0)] = \boxed{2}$

$dist(c)$ $\min[(3+0)] = \boxed{3}$

$dist(a)$ $= \min(2$

$\boxed{}$ 3

C.min(2,3)

Dynamic Programming

This can be built backwards from the goal:

This can be built backwards from the goal:

□□ $\underline{0 \qquad\qquad\qquad\qquad} \; if \quad \underline{is\_goal(n),}$

$\underline{}dist(n)$ $\langle \underline{\ } 0 \rangle$ $\left(\underline{cost(n,m)} + \underline{\boxed{dist(m)}}\right)$

□□□□min $_{n,m□A}$ dist(u)

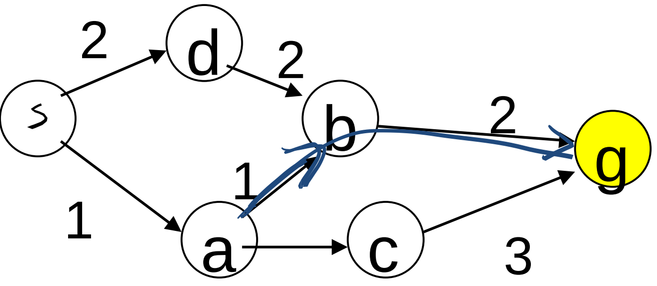
otherwise g 0

allthe neighbors m

dist(b) = min[(2 + 0)] = 2

dist(c) = min[(3 + 0)] = 3

dist(a) = min[(3+3), (1+2)] = 3



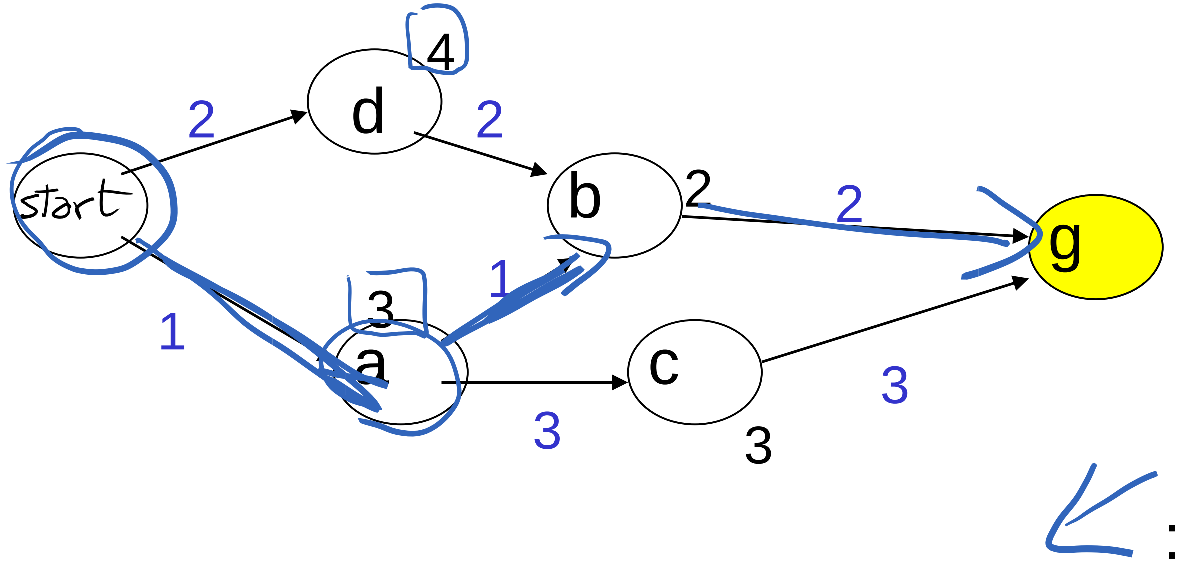
Dynamic Programming

This can be used locally to determine what to do.

From each node n go to its neighbor which minimizes

$$\left[\mathrm{cost}(n,m) + dist(m) \right]$$

But there are at least two main problems



- You need enough space to store the graph.
- The dist function needs to be recomputed for each goal

Learning Goals for today's class

- Define/read/write/trace/debug & Compare different Informed search algorithms Best First Search, A*, and Branch Bound

- Formally prove A* optimality.

Dynamic Programming

- Apply techniques to deal with cycles and repeated states
- Simplify search when full search graph can be stored

To Do for Next Class

- Read
- Chp 4.1-4.2 (Intro to Constraint Satisfaction Problems)

- Do Practice Exercise 3E

- Keep working on assignment-1 !

Next class

Finish Search (finish Chpt 3)

- Branch-and-Bound
- Informed IDS
- A* enhancements
- Non-heuristic Pruning
- Dynamic Programming