Iterative Deepening and Branch & Bound

CPSC 322 – Search 6

Textbook § 3.7.3 and 3.7.4

January 24, 2011

Lecture Overview

- Recap from last week
- Iterative Deepening
- Branch & Bound

Search with Costs

- Sometimes there are costs associated with arcs.

Def.: The cost of a path is the sum of the costs of its arcs

$$\operatorname{cost}\left(\langle n_0, \ldots, n_k \rangle\right) = \sum_{i=1}^{k} \operatorname{cost}(\langle n_{i-1}, n_i \rangle)$$

- In this setting we often don't just want to find any solution
  – we usually want to find the solution that minimizes cost

Def.: A search algorithm is optimal if

when it finds a solution, it is the best one:

it has the lowest path cost

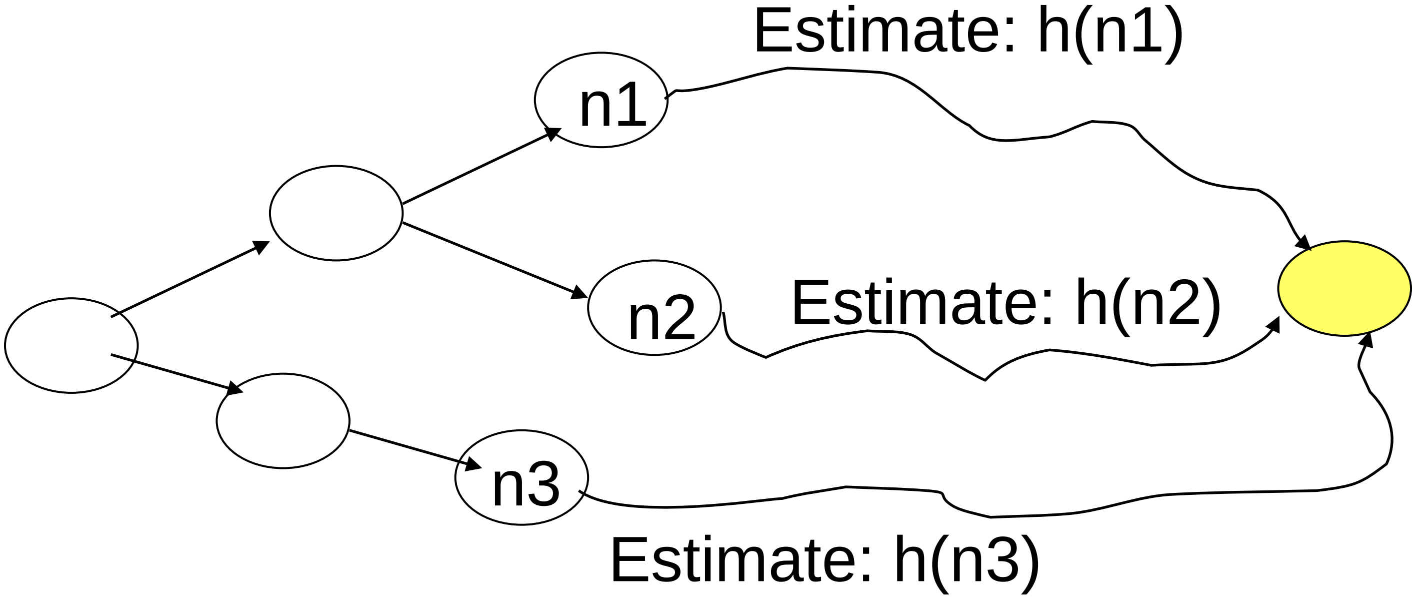
Lowest-Cost-First Search (LCFS)

- Expands the path with the lowest cost on the frontier.
- The frontier is implemented as a priority queue ordered by path cost.
- How does this differ from Dijkstra's algorithm?
  - The two algorithms are very similar
  - But Dijkstra’s algorithm
    - works with nodes not with paths
    - stores one bit per node (infeasible for infinite/very large graphs)
    - checks for cycles

Heuristic search

Def.:

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



Best-First Search (LCFS)

- Expands the path with the lowest h value on the frontier.
- The frontier is implemented as a priority queue ordered by h.
- Greedy: expands path that appears to lead to the goal quickest
  - Can get trapped
  - Can yield arbitrarily poor solutions
  - But with a perfect heuristic, it moves straight to the goal

A*

- Expands the path with the lowest cost + h value on the frontier
- The frontier is implemented as a priority queue ordered by $f(p) = cost(p) + h(p)$

Admissibility of a heuristic

Def.:

Let $c(n)$ denote the cost of the optimal path from node n to any goal node. A search heuristic $h(n)$ is called admissible if $h(n) \leq c(n)$ for all nodes n, i.e. if for all nodes it is an underestimate of the cost to any goal.

- E.g. Euclidian distance in routing networks
- General construction of heuristics: relax the problem, i.e. ignore some constraints
  - Can only make it easier
  - Saw lots of examples on Wednesday: Routing network, grid world, 8 puzzle, Infinite Mario

Admissibility of A*

- A* is complete (finds a solution, if one exists) and optimal (finds the optimal path to a goal) if:

  - *the branching factor is finite*
  - *arc costs are > 0*
  - *h is admissible.*

- This property of A* is called admissibility of A*

Why is A* admissible: complete

If there is a solution, A* finds it:

- f_{min} := cost of optimal solution path s (unknown but finite)

- Lemmas for prefix pr of s (exercise: prove at home)
  - Has cost $f(pr) \leq f_{min}$ (due to admissibility)
  - Always one such pr on the frontier (prove by induction)

- A* only expands paths with $f(p) \leq f_{min}$
  - Expands paths p with minimal f(p)
  - Always a pr on the frontier, with $f(pr) \leq f_{min}$
  - Terminates when expanding s

- Number of paths p with cost $f(p) \leq f_{min}$ is finite
  - Let $c_{min} > 0$ be the minimal cost of any arc
  - $k := f_{min} / c_{min}$. All paths with length > k have cost > f_{min}
  - Only b^k paths of length k. Finite b $\Rightarrow$ finite

Why is A* admissible: optimal

Proof by contradiction

- Assume (for contradiction):
  First solution s' that A* expands is suboptimal: i.e. cost(s') > f_{min}

- Since s' is a goal, h(s') = 0, and f(s') = cost(s') > f_{min}

- A* selected s' $\Rightarrow$ all other paths p on the frontier
  had f(p) $\geq$ f(s') > f_{min}

- But we know that a prefix pr of optimal solution path s is on the frontier, with f(pr) $\leq$ f_{min}
  $\Rightarrow$ Contradiction !

Summary: any prefix of optimal solution is expanded before suboptimal solution would be expanded

Learning Goals for last week

- Select the most appropriate algorithms for specific problems
  - Depth-First Search vs. Breadth-First Search vs. Least-Cost-First Search vs. Best-First Search vs. A*
- Define/read/write/trace/debug different search algorithms
  - With/without cost
  - Informed/Uninformed
- Construct heuristic functions for specific search problems
- Formally prove A* optimality
  - Define optimal efficiency

Learning Goals for last week, continued

- 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	$O(b^m)$	$O(b^m)$
LCFS (when arc costs available)	Y Costs > 0	Y Costs $\geq$ 0	$O(b^m)$	$O(b^m)$
Best First (when *h* available)	N	N	$O(b^m)$	$O(b^m)$
A* (when arc costs and *h* available)	Y Costs > 0 *h* admissible	Y Costs $\geq$ 0 *h* admissible	$O(b^m)$	$O(b^m)$

Lecture Overview

- Recap from last week
- Iterative Deepening
- Branch & Bound

Iterative Deepening DFS (short IDS): Motivation

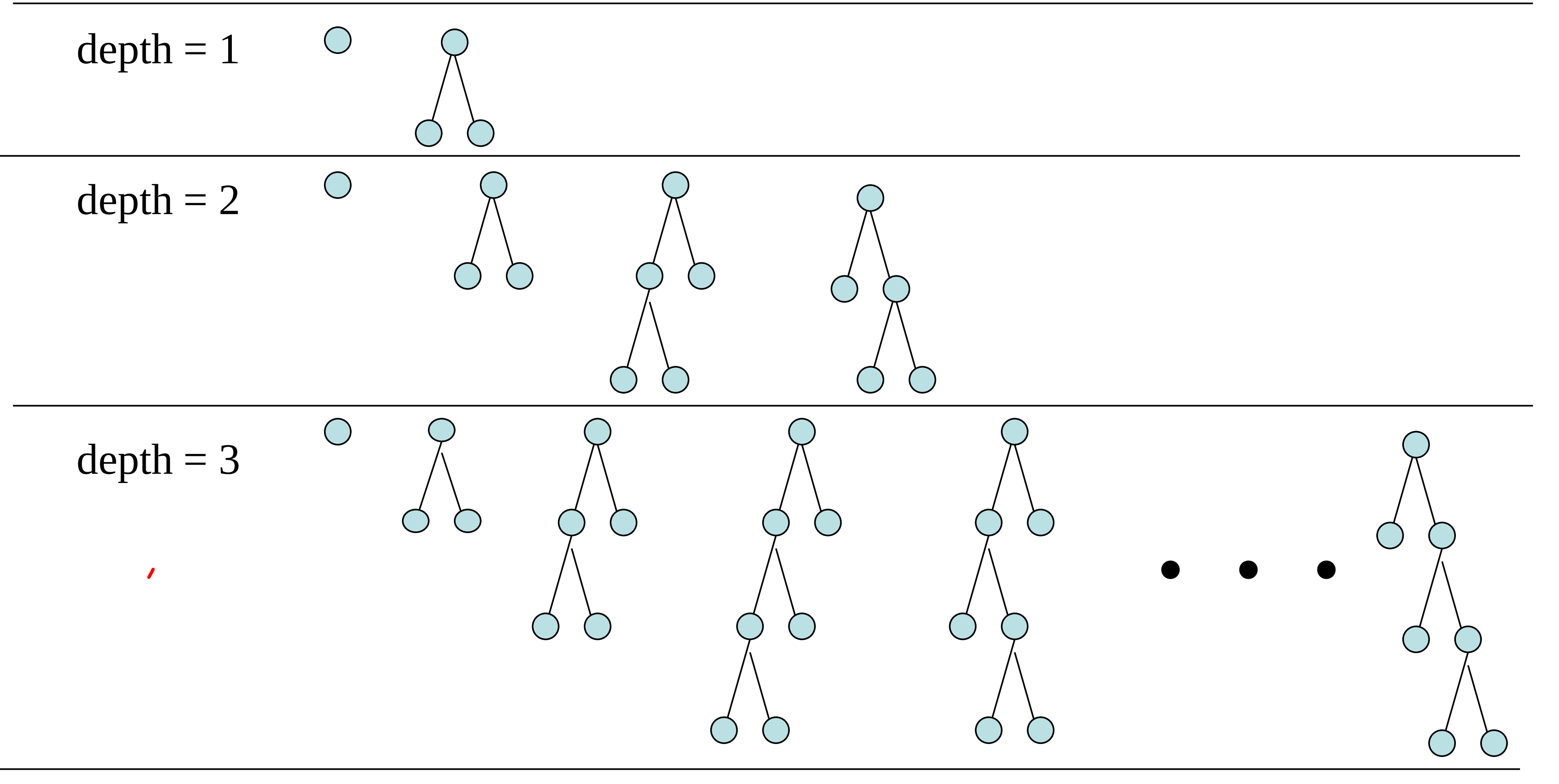
Want low space complexity but completeness and optimality

Key Idea: re-compute elements of the frontier
rather than saving them

	Complete	Optimal	Time	Space
DFS	N (Y if no cycles)	N	$O(b^m)$	$O(mb)$
BFS	Y	Y	$O(b^m)$	$O(b^m)$
LCFS (when arc costs available)	Y Costs > 0	Y Costs >=0	$O(b^m)$	$O(b^m)$
Best First (when *h* available)	N	N	$O(b^m)$	$O(b^m)$
A* (when arc costs and *h* available)	Y Costs > 0 *h* admissible	Y Costs >=0 *h* admissible	$O(b^m)$	$O(b^m)$

Iterative Deepening DFS (IDS) in a Nutshell

- Use DSF to look for solutions at depth 1, then 2, then 3, etc
  - For depth D, ignore any paths with longer length
  - Depth-bounded depth-first search



(Time) Complexity of IDS

- That sounds wasteful!
- Let's analyze the time complexity
- For a solution at depth *m* with branching factor *b*

Depth	Total # of paths at that level	#times created by BFS (or DFS)	#times created by IDS	Total #paths for IDS
1	b	1	m	mb
2	b^2	1	$m-1$	$(m-1)\ b^2$
. . .	. . .	. . .	. . .	. . .
m-1	b^{m-1}	1	2	$2\ b^{m-1}$
m	b^m	1	1	b^m

(Time) Complexity of IDS

Solution at depth m, branching factor b

Total # of paths generated:

$b^m + 2\,b^{m-1} + 3\,b^{m-2} + \ldots + mb$

$= b^m\,(1\,b^0 + 2\,b^{-1} + 3\,b^{-2} + \ldots + m\,b^{1-m})$

$$= b^m \left(\sum_{i=1}^{m} i b^{1-i}\right) = b^m \left(\sum_{i=1}^{m} i (b^{-1})^{i-1}\right)$$

$$\leq b^m \left(\sum_{i=0}^{\infty} i (b^{-1})^{i-1}\right) = b^m \left(\frac{1}{1-b^{-1}}\right)^2 = b^m \left(\frac{b}{b-1}\right)^2 \in O(b^m)$$

Geometric progression: for |r|<1: $\sum_{i=0}^{\infty} r^i = \frac{1}{1-r}$

$$\frac{d}{dr}\sum_{i=0}^{\infty} r^i = \sum_{i=0}^{\infty} i r^{i-1} = \frac{1}{(1-r)^2}$$

Further Analysis of Iterative Deepening DFS (IDS)

- Space complexity

  $O(b^m)$ $O(m^b)$ $O(bm)$ $O(b+m)$

  – DFS scheme, only explore one branch at a time

- Complete? Yes No

  – Only finite # of paths up to depth m, doesn't explore longer paths

- Optimal? Yes No

  – Proof by contradiction

Search methods so far

	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 > 0	Y Costs >=0	$O(b^m)$	$O(b^m)$
Best First (when *h* available)	N	N	$O(b^m)$	$O(b^m)$
A* (when arc costs and *h* available)	Y Costs > 0 *h* admissible	Y Costs >=0 *h* admissible	$O(b^m)$	$O(b^m)$

(Heuristic) Iterative Deepening: IDA*

- Like Iterative Deepening DFS
  - But the depth bound is measured in terms of the f value

- If you don't find a solution at a given depth
  - Increase the depth bound:
    to the minimum of the f-values that exceeded the previous bound

Analysis of Iterative Deepening A* (IDA*)

- Complete and optimal? Same conditions as A*
  - h is admissible
  - all arc costs > 0
  - finite branching factor

- Time complexity: $O(b^m)$

- Space complexity:

  $O(b^m)$ $O(m^b)$ $O(bm)$ $O(b+m)$

  - Same argument as for Iterative Deepening DFS

Lecture Overview

- Recap from last week
- Iterative Deepening
- Branch & Bound

Heuristic DFS

- Other than IDA*, can we use heuristic information in DFS?
  - When we expand a node, put all its neighbours on the stack
  - In which order?
    - Can use heuristic guidance: h or f
    - Perfect heuristic: would solve problem without any backtracking

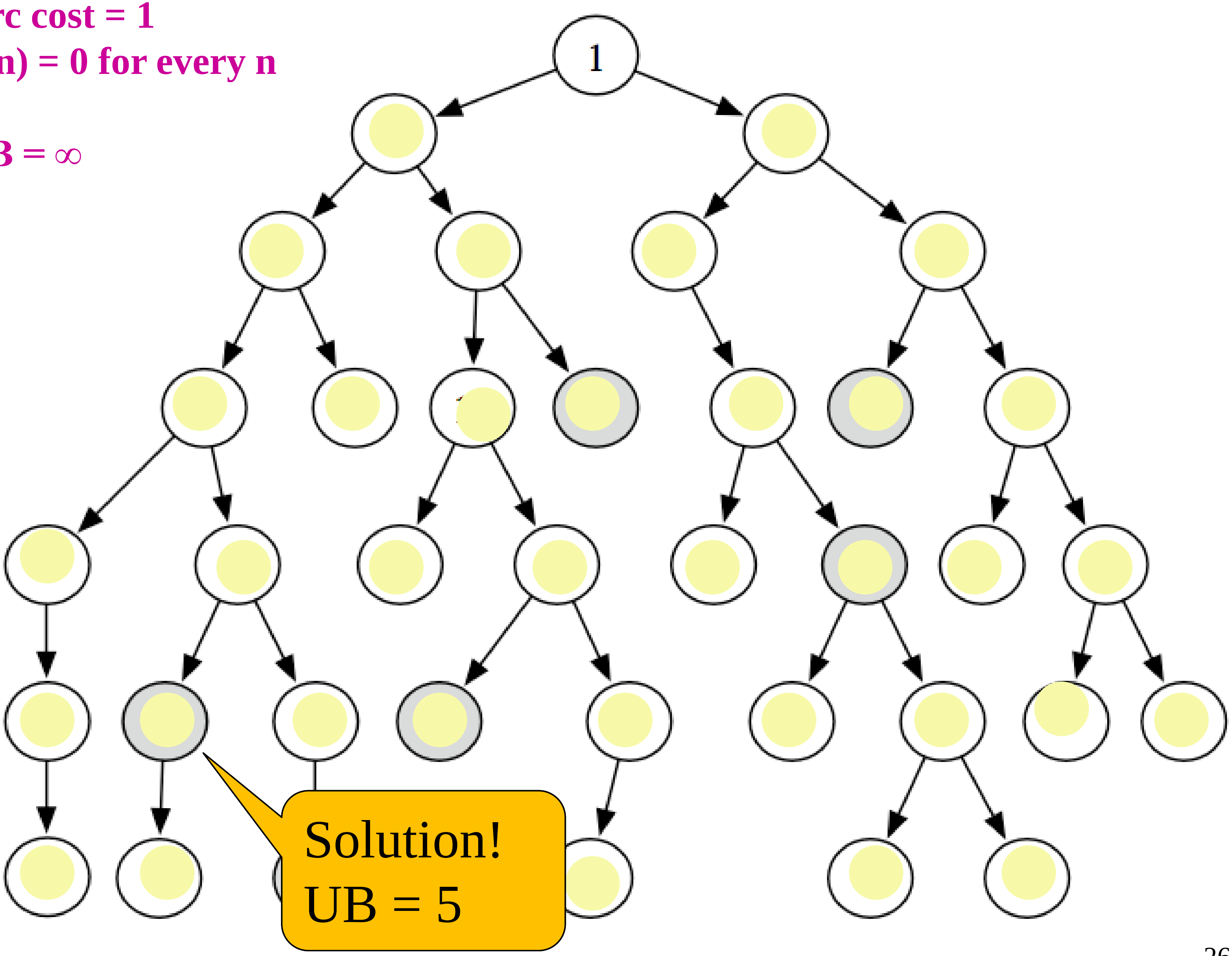
- Heuristic DFS is very frequently used in practice
  - Often don't need optimal solution, just some solution
  - No requirement for admissibility of heuristic
    - As long as we don't end up in infinite paths

Branch-and-Bound Search

- Another way to combine DFS with heuristic guidance
- Follows exactly the same search path as depth-first search
  - But to ensure optimality, it does not stop at the first solution found
- It continues, after recording upper bound on solution cost
  - upper bound: *UB* = cost of the best solution found so far
  - Initialized to ∞ or any overestimate of solution cost
- When a path *p* is selected for expansion:
  - Compute $LB(p) = f(p) = cost(p) + h(p)$
    - If $LB(p) \geq UB$, remove *p* from frontier without expanding it (pruning)
    - Else expand *p*, adding all of its neighbors to the frontier
  - Requires admissible h

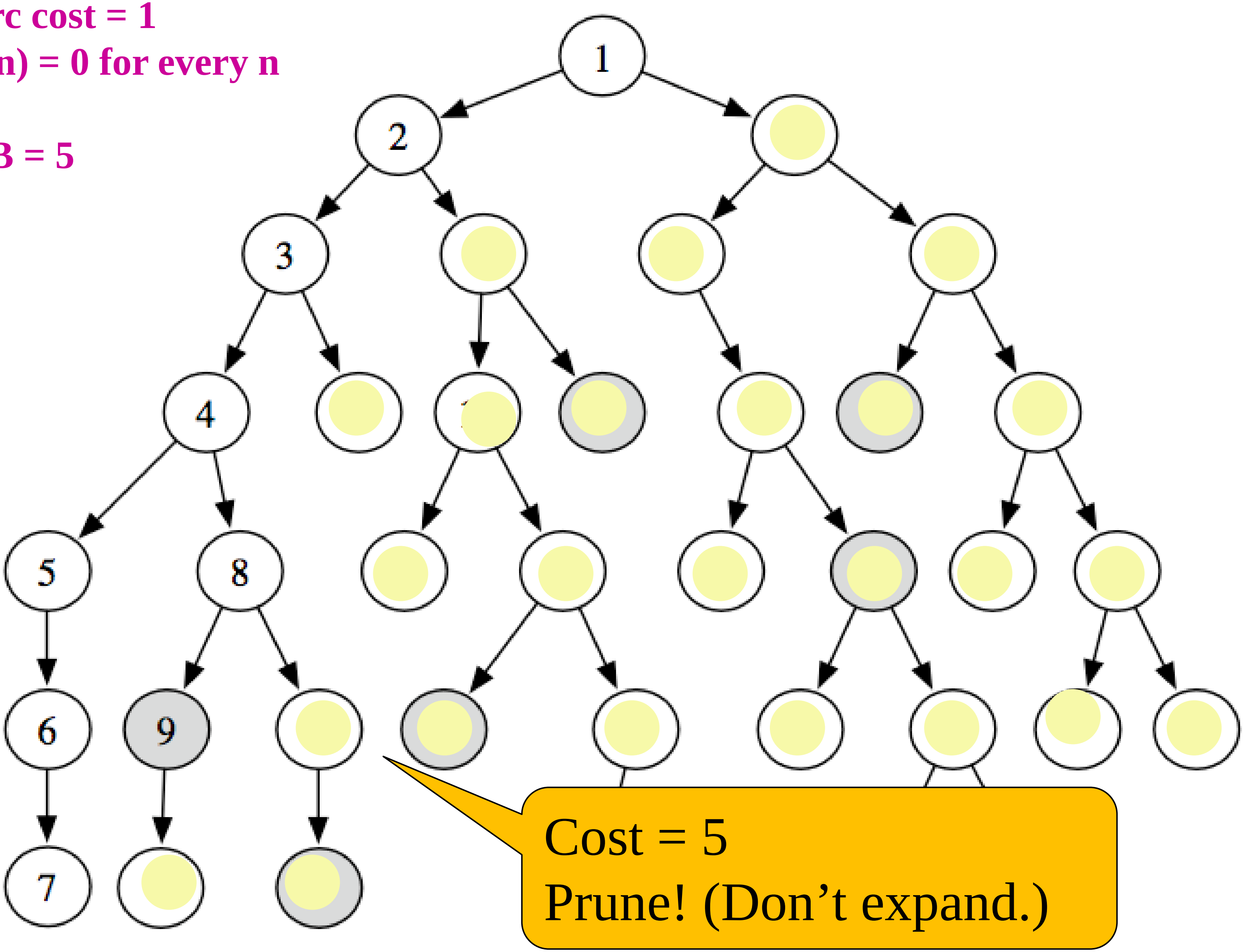
- **Arc cost = 1**
- **h(n) = 0 for every n**

- **UB = ∞**



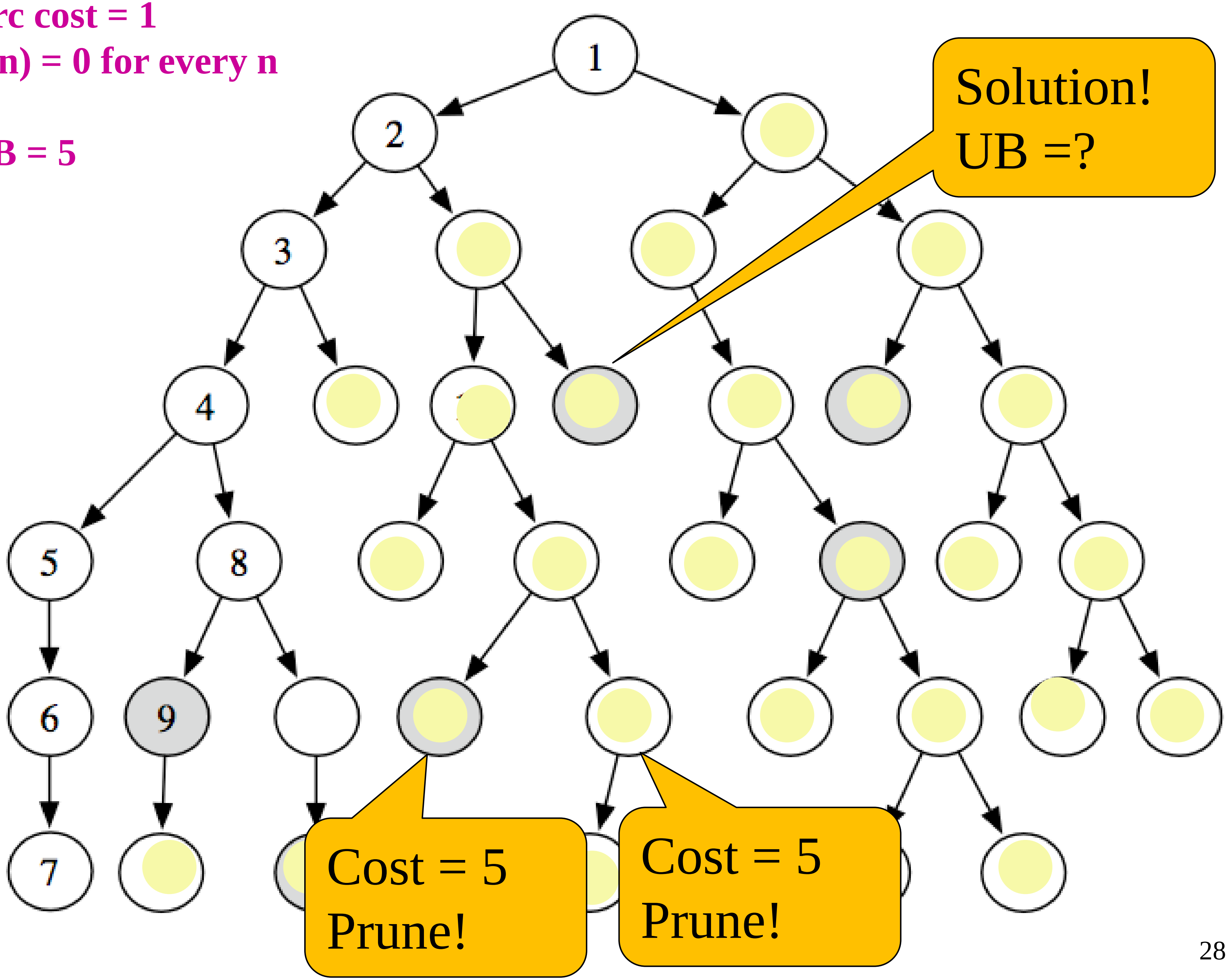
- **Arc cost = 1**
- **h(n) = 0 for every n**

- **UB = 5**



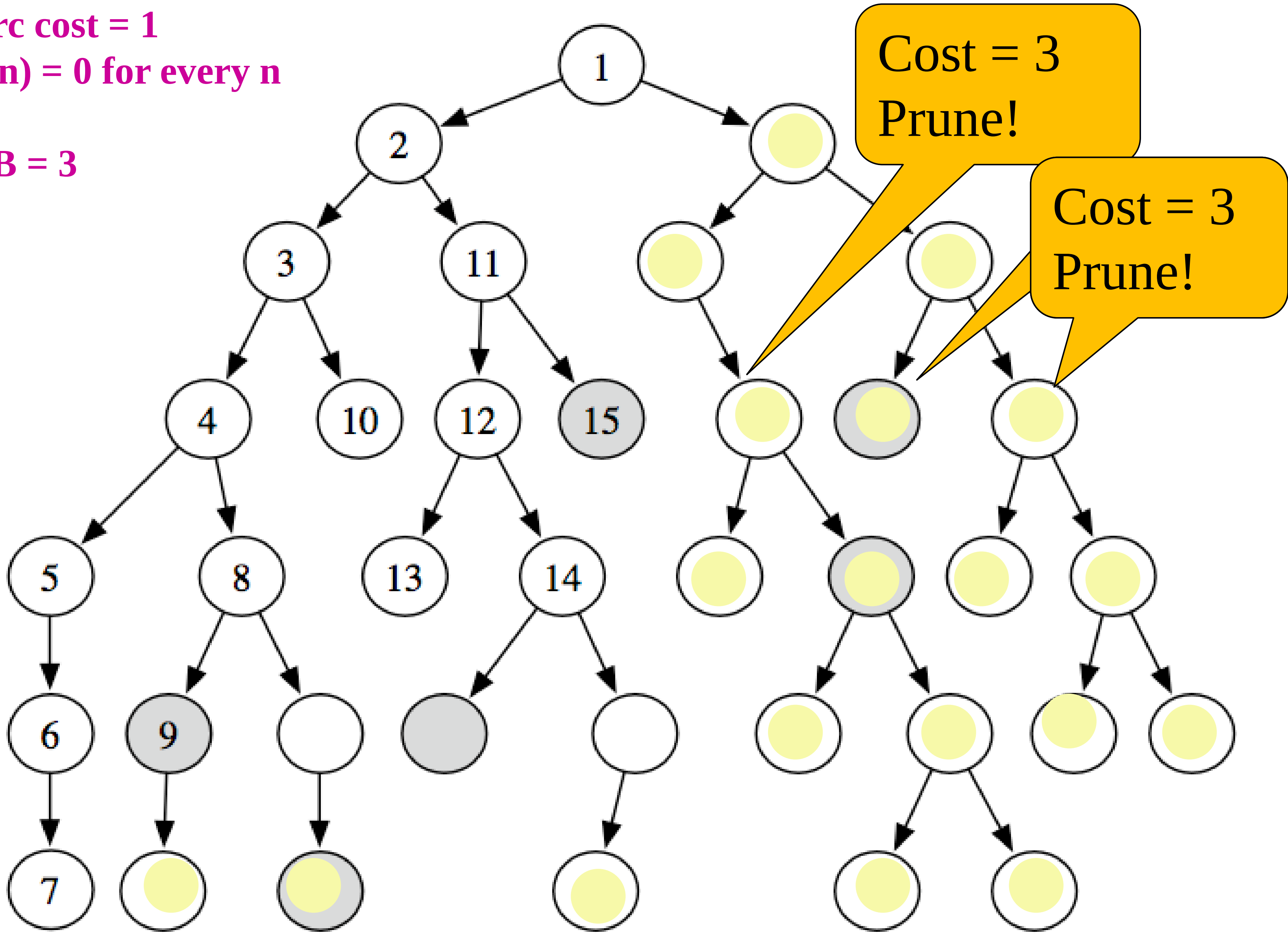
•Arc cost = 1
•h(n) = 0 for every n

•UB = 5



- **Arc cost = 1**
- **h(n) = 0 for every n**

- **UB = 3**



Branch-and-Bound Analysis

- Complete? **YES** **NO** **IT DEPENDS**
  - Can't handle infinite graphs (but can handle cycles)
- Optimal? **YES** **NO** **IT DEPENDS**
  - If it halts, the goal will be optimal
  - But it could find a goal and then follow an infinite path …
- Time complexity: $O(b^m)$
- Space complexity $O(b^m)$ $O(m^b)$ $O(bm)$ $O(b+m)$
  - It's a DFS

Combining B&B with heuristic guidance

- We said
  - "Follows exactly the same search path as depth-first search""
  - Let's make that heuristic depth-first search
- Can freely choose order to put neighbours on the stack
  - Could e.g. use a separate heuristic h' that is NOT admissible
- To compute LB(p)
  - Need to compute f value using an admissible heuristic h
- This combination is used a lot in practice
  - Sudoku solver in assignment 2 will be along those lines
  - But also integrates some logical reasoning at each node

Search methods so far

	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 > 0	Y Costs >=0	$O(b^m)$	$O(b^m)$
Best First (when *h* available)	N	N	$O(b^m)$	$O(b^m)$
A* (when arc costs and *h* available)	Y Costs > 0 *h* admissible	Y Costs >=0 *h* admissible	$O(b^m)$	$O(b^m)$
IDA*	Y (same cond. as A*)	Y	$O(b^m)$	$O(mb)$
Branch & Bound	Y (same cond. as A*)	Y	$O(b^m)$	$O(mb)$

Memory-bounded *A**

- Iterative deepening A* and B & B use little memory
- What if we've got more memory, but not $O(b^m)$?
- Do A* and keep as much of the frontier in memory as possible
- When running out of memory
    - delete worst path (highest f value) from frontier
    - Back its *f* value up to a common ancestor
- Subtree gets regenerated only when all other paths have been shown to be worse than the “forgotten” path

- Details are beyond the scope of the course, but
    - Complete and optimal if solution is at depth manageable for available memory

Learning Goals for today's class

- Define/read/write/trace/debug different search algorithms
  - New: Iterative Deepening,
    Iterative Deepening A*, Branch & Bound
- Apply basic properties of search algorithms:
  - completeness, optimality, time and space complexity

Announcements:

- New practice exercises are out: see WebCT
  - Heuristic search
  - Branch & Bound
  - Please use these! (Only takes 5 min. if you understood things…)
- Assignment 1 is out: see WebCT