Finish Search

Computer Science cpsc322, Lecture 10

(Textbook Chpt 3.6)

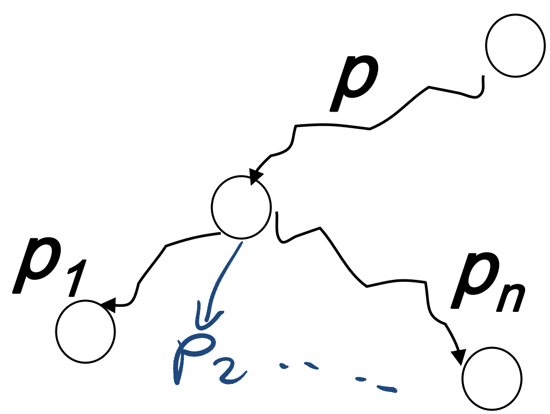
Sep, 26, 2010

Lecture Overview

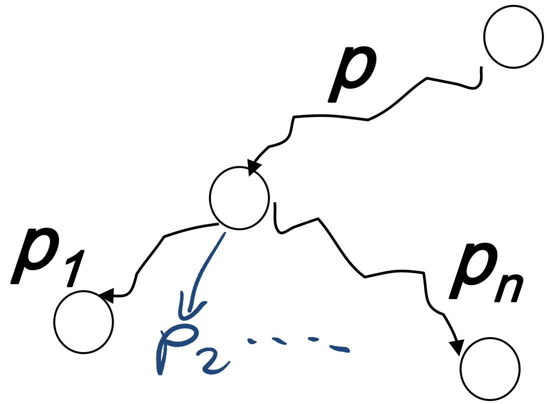
- Finish MBA*
- Pruning Cycles and Repeated states Examples
- Dynamic Programming
- Search Recap

Memory-bounded A^*

- Iterative deepening A* and B & B use a tiny amount of memory (but have their own problems…)
- **what if we've got more memory to use?**
- keep as much of the fringe in memory as we can
- if we have to delete something:
  - delete the worst paths (with ……highest……………f……)
  - ``back them up'' to a common ancestor



MBA*: Compute New *h(p)*



$$\text{New } h(p) = \min\left(\cancel{\max_i \left[\left(\text{cost}(p_i) - \text{cost}(p) \right) + h(p_i) \right]}, \text{Old } h(p) \right)$$

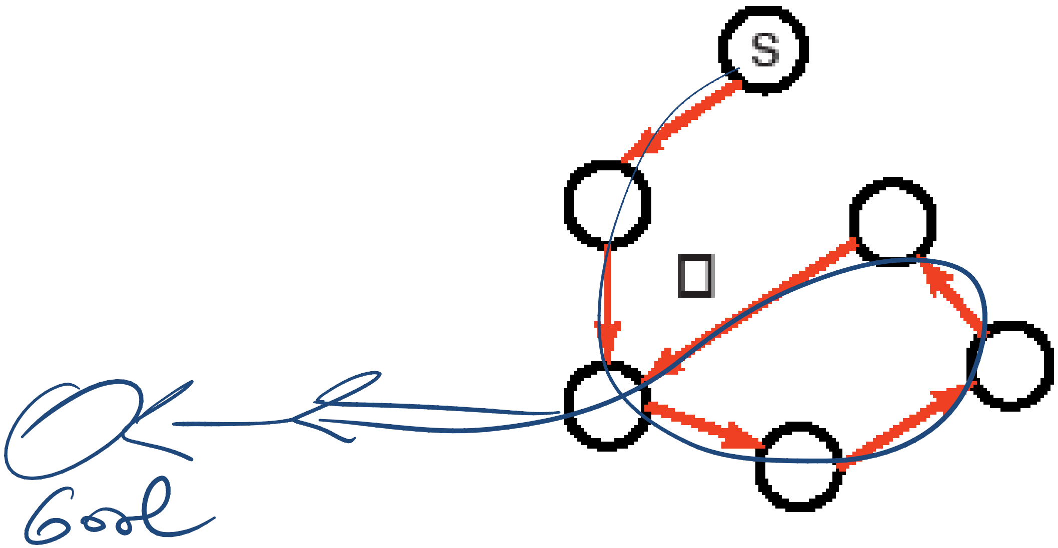
$$\text{New } h(p) = \max\left(\min_i \left[\left(\text{cost}(p_i) - \text{cost}(p) \right) + h(p_i) \right], \text{Old } h(p) \right)$$

possibly better underestimate of distance from end node of p to goal

Lecture Overview

- Finish MBA*
- Pruning Cycles and Repeated states Examples
- Dynamic Programming
- Search Recap

Cycle Checking



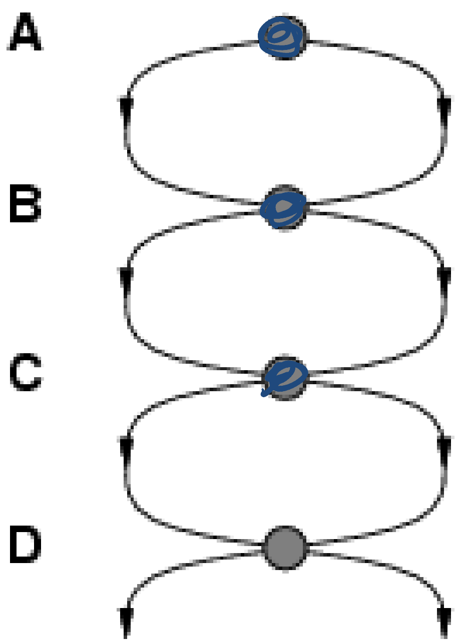
You can prune a path that ends in a node already on the path. This pruning cannot remove an optimal solution.

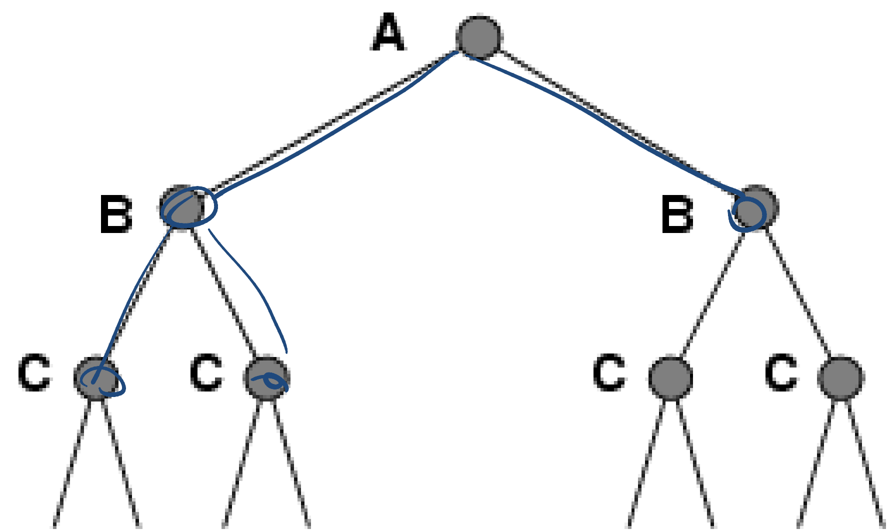
- The time is ....linear.............. in path length.

$\langle n_0, n_1, n_2 \ldots n_k \rangle$ n_2 n_i n_j

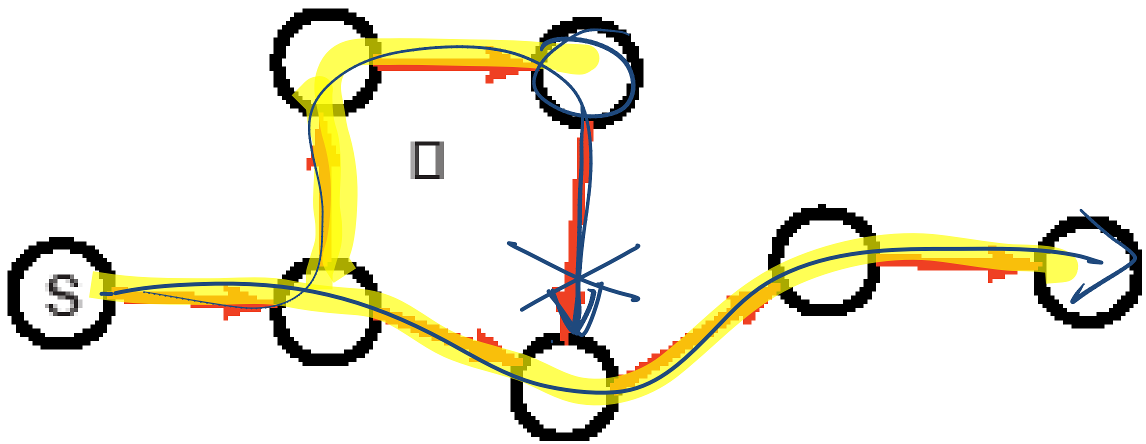
Repeated States / Multiple Paths

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





Multiple-Path Pruning



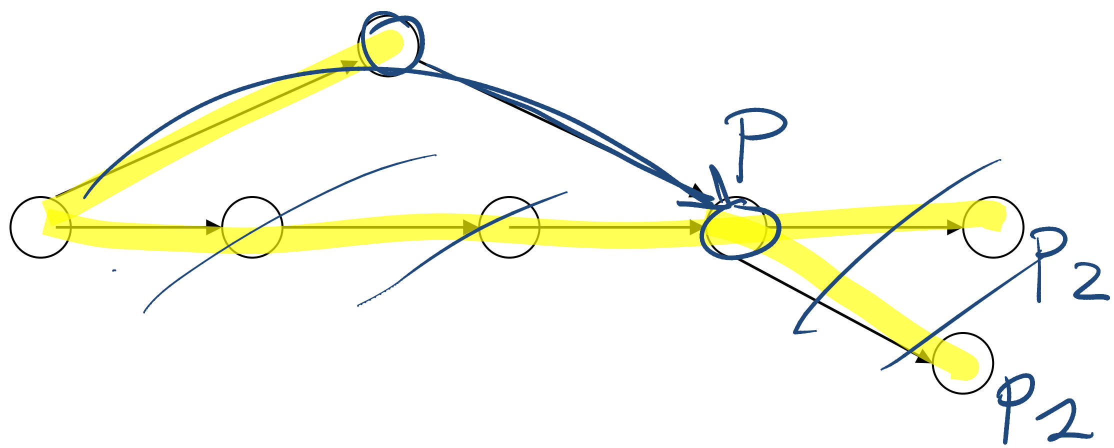
•You can prune a path to node *n* that you have already found a path to

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

Multiple-Path Pruning & Optimal Solutions

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

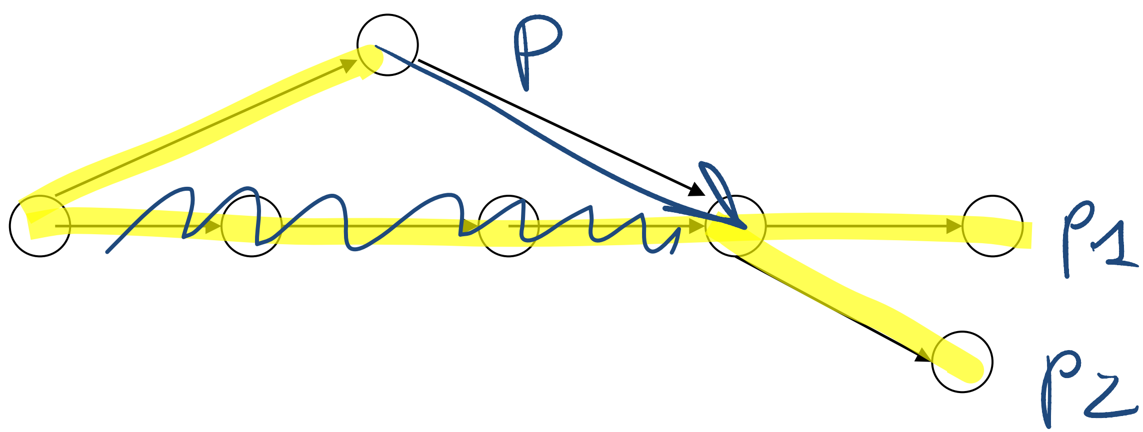
- You can remove all paths from the frontier that use the longer path. (as these can't be optimal)



Multiple-Path Pruning & Optimal Solutions

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

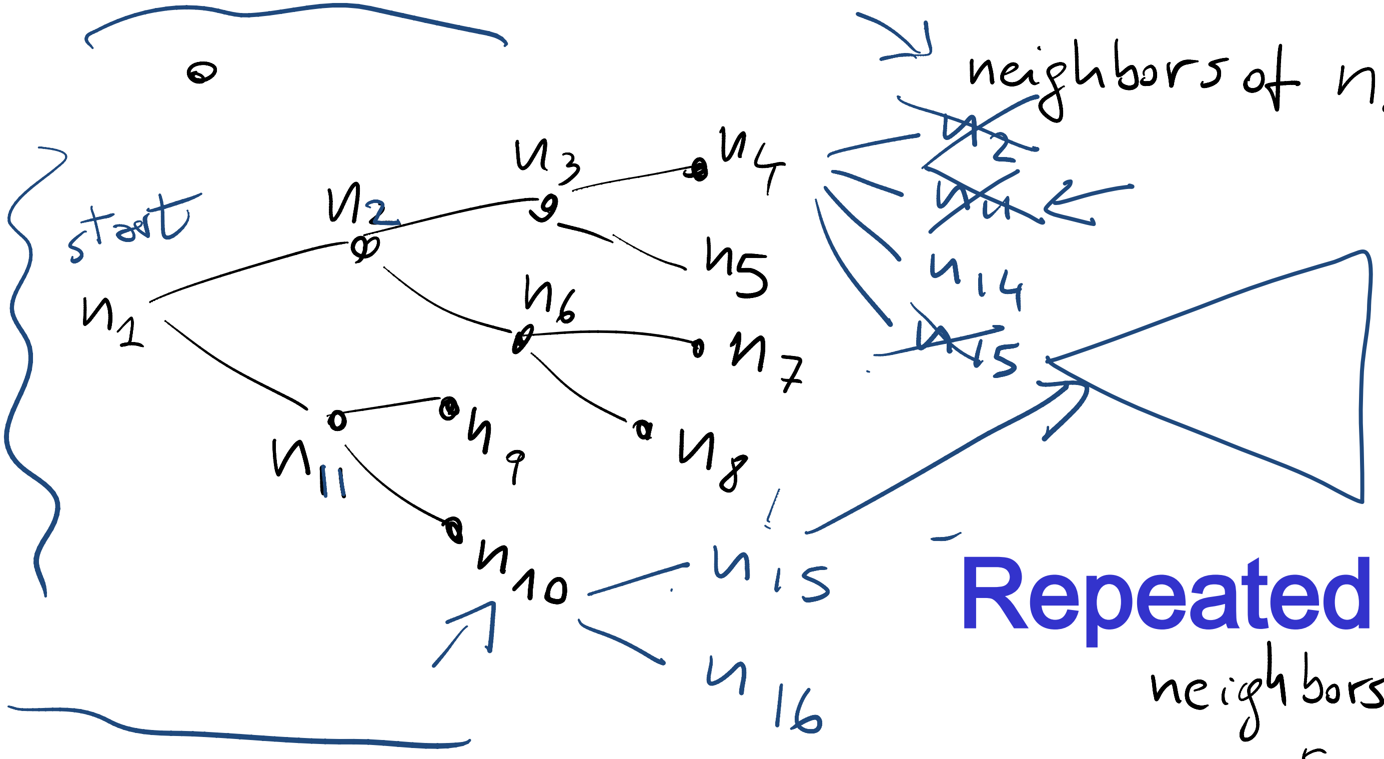
- You can change the initial segment of the paths on the frontier to use the shorter path.



Example

Pruning Cycles

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



Repeated States

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

Lecture Overview

- Finish MBA*
- Pruning Cycles and Repeated states Examples
- **Dynamic Programming**
- Search Recap

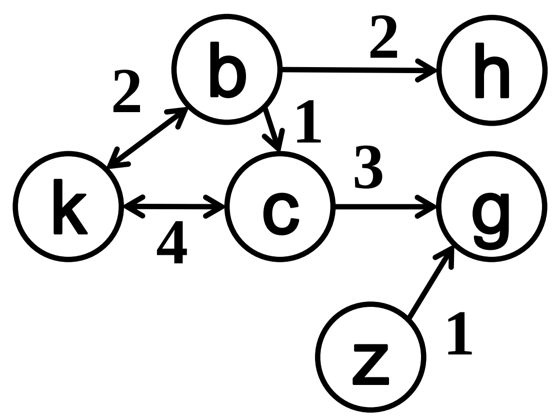
Dynamic Programming

- 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

f function

cost

heuristic



  - dist(g) = 0
  - dist(z) = 1
  - dist(c) = 3
  - dist(b) = 4 6 7 ∞
  - dist(k) = ? 6 7 ∞
  - dist(h) = ?

- How could we implement that?

Dynamic Programming

This can be built backwards from the goal:

This can be built backwards from the goal:

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

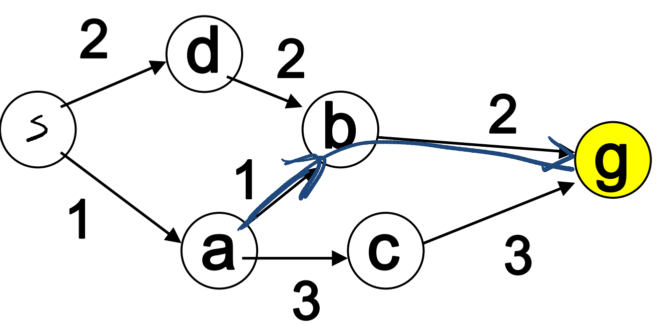
all the neighbors m

	dist(n)
g	0

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

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

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

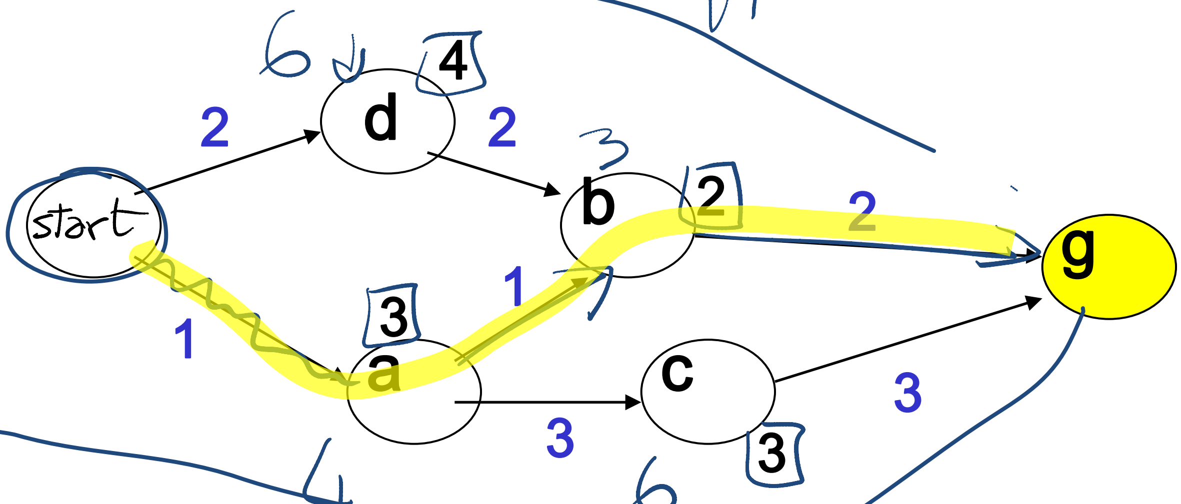


Dynamic Programming

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

Lecture Overview

- Finish MBA*
- Pruning Cycles and Repeated states Examples
- Dynamic Programming
- **Search Recap**

Recap Search

	Selection	Complete	Optimal	Time	Space
DFS	LIFO	N	N	$O(b^m)$	$O(mb)$
BFS	FIFO	Y	Y	$O(b^m)$	$O(b^m)$
IDS(C)	LIFO	Y	Y	$\boxed{O(b^m)}$	$O(mb)$
LCFS	min cost	Y	Y	$O(b^m)$	$O(b^m)$
BFS	min h	N	N	$O(b^m)$	$O(b^m)$
A*	min f=c+h	Y	Y	$O(b^m)$	$O(b^m)$
B&B	LIFO + pruning f	N	Y	$O(b^m)$	$O(mb)$
IDA*	LIFO	Y	Y	$O(b^m)$	$O(mb)$
MBA*	min f	N	Y	$O(b^m)$	$O(b^m)$

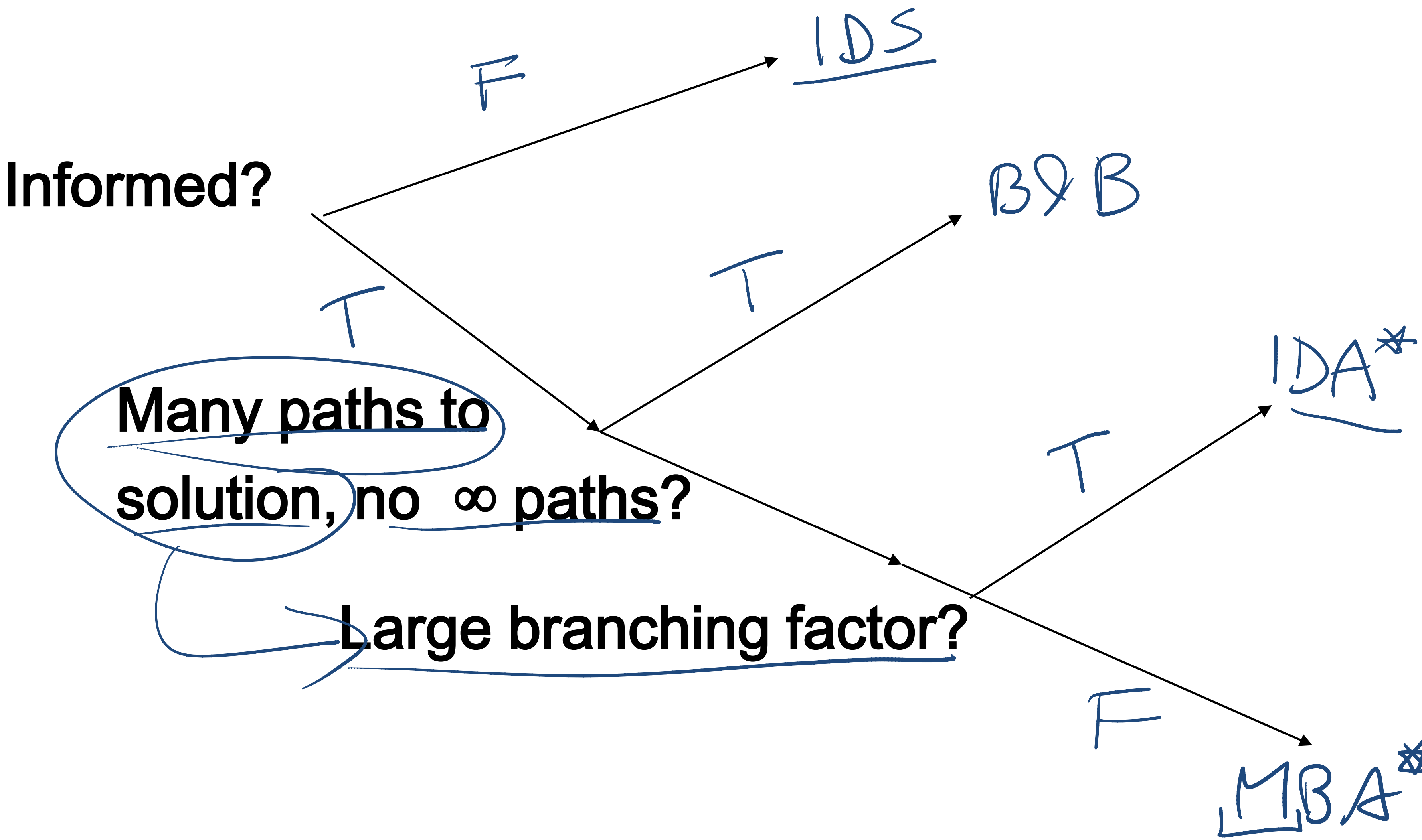
Recap Search (some qualifications)

	Complete	Optimal	Time	Space
DFS	N	N	$O(b^m)$	$O(mb)$
BFS	Y	Y	$O(b^m)$	$O(b^m)$
IDS(C)	Y	Y	$O(b^m)$	$O(mb)$
LCFS	Y	Y ? $c \geq 0$	$O(b^m)$	$O(b^m)$
BFS	N	N	$O(b^m)$	$O(b^m)$
A*	Y	h admissible Y ?	$O(b^m)$	$O(b^m)$
B&B	N	Y ?	$O(b^m)$	$O(mb)$
IDA*	Y	Y	$O(b^m)$	$O(mb)$
MBA*	N	Y	$O(b^m)$	$O(b^m)$

Search in Practice

	Complete	Optimal	Time	Space
DFS	N	N	$O(b^m)$	$O(mb)$
BFS	Y	Y	$O(b^m)$	$O(b^m)$
IDS(C)	Y	Y	$O(b^m)$	$O(mb)$
LCFS	Y	Y	$O(b^m)$	$O(b^m)$
BFS	N	N	$O(b^m)$	$O(b^m)$
A*	Y	Y	$O(b^m)$	$O(b^m)$
B&B	N	Y	$O(b^m)$	$O(mb)$
IDA*	Y	Y	$O(b^m)$	$O(mb)$
MBA*	N	Y	$O(b^m)$	$O(b^m)$
BDS	Y	Y	$O(b^{m/2})$	$O(b^{m/2})$

Search in Practice (cont')



(Adversarial) Search: Chess

Deep Blue's Results in the second tournament:

- second tournament: won 3 games, lost 2, tied 1

- 30 CPUs + 480 chess processors
- Searched 126.000.000 nodes per sec
- Generated 30 billion positions per move reaching depth 14 routinely

- **Iterative Deepening** with evaluation function (similar to a **heuristic**) based on 8000 features (e.g., sum of worth of pieces: pawn 1, rook 5, queen 10)

Modules we'll cover in this course: R&Rsys

Problem		Environment: Deterministic	Environment: Stochastic
Static	Constraint Satisfaction	*Vars + Constraints* Arc Consistency Search	
Static	Query	*Logics* Search	*Belief Nets* Var. Elimination
Sequential	Planning	*STRIPS* actions precondts effects Search	*Decision Nets* Var. Elimination *Markov Processes* Value Iteration

Representation

Reasoning Technique

Standard Search vs. Specific R&R systems

Constraint Satisfaction (Problems):

- State
- Successor function
- Goal test
- Solution
- Heuristic function

Planning :

- State
- Successor function
- Goal test
- Solution
- Heuristic function

Inference

- State
- Successor function
- Goal test
- Solution
- Heuristic function

Next class

Start **Constraint Satisfaction Problems** (CSPs)

Textbook 4.1-4.3

I will be away for 2-3 classes

Alan Mackworth will sub for me

- Co-author of your textbook
- Pioneered work in CSP