SEARCH TREE

Node: State in state tree

Root node: Top of state tree

Children: Nodes that can be reached from a given node in 1 step (1 operator)

Expanding: Generating the children of a node

Open: Node not yet expanded

Closed: Node after expansion

Queue: Ordered list of open nodes

DEPTH 1st SEARCH

1. Put start state onto queue
2. If queue is empty then fail
3. If head of queue is goal then succeed
4. Else remove head of queue, expand it, place children in front of queue
5. Recurse to 2

DEPTH 1st (cont.)

When to use

- Depth limited or known beforehand
- All solutions at same depth
- Any solution will do
- Possibly fast

When to avoid

- Large or infinite subtrees
- Prefer shallow solution

BREADTH 1st SEARCH

1. Put start state onto queue
2. If queue is empty then fail
3. If head of queue is goal then succeed
4. Else remove head of queue, expand it, place children at end of queue
5. Recurse to 2

BREADTH 1st (Cont.)

When to use

- Large or infinite search tree
- Solution depth unknown
- Prefer shallow solution

When to avoid

- Very wide trees
- Generally slow
- May need to store b^d nodes

MODIFICATIONS TO DEPTH/BREADTH 1ST

Depth–limited Search:
Limit the total depth of the depth 1st search.

Iterative Deepening Search:
Repeat depth–limited search with limit 0, 1, 2, 3, . . . until a solution is found.

Uniform Cost Search:
Same as breadth 1st search but ordering the nodes on the queue according to their costs.

Bidirectional Search:
Simultaneously search forward from initial state and backward from goal state until both paths meet.

UNIFORM COST SEARCH

1. Put start state onto queue
2. If queue is empty then fail
3. If head of queue is goal then succeed
4. Else remove head of queue, expand it, place in queue, and **sort entire queue** by cost
5. Recurse to 2

UNIFORM COST SEARCH SUMMARY

When to use

- Desire good solution
- Almost always finds best solution

When to avoid

- Negative costs
- Any solution will do