CS370

Symbolic Programming
Declarative Programming

LECTURE 12: Basic Problem-Solving Strategies

Jong C. Park
park@cs.kaist.ac.kr
Computer Science Department
Korea Advanced Institute of Science and Technology
http://nlp.kaist.ac.kr/~cs370

Basic Problem-Solving Strategies

- Introductory concepts and examples
- Depth-first search and iterative deepening
- Breadth-first search
- Analysis of basic search techniques

Introductory concepts and examples

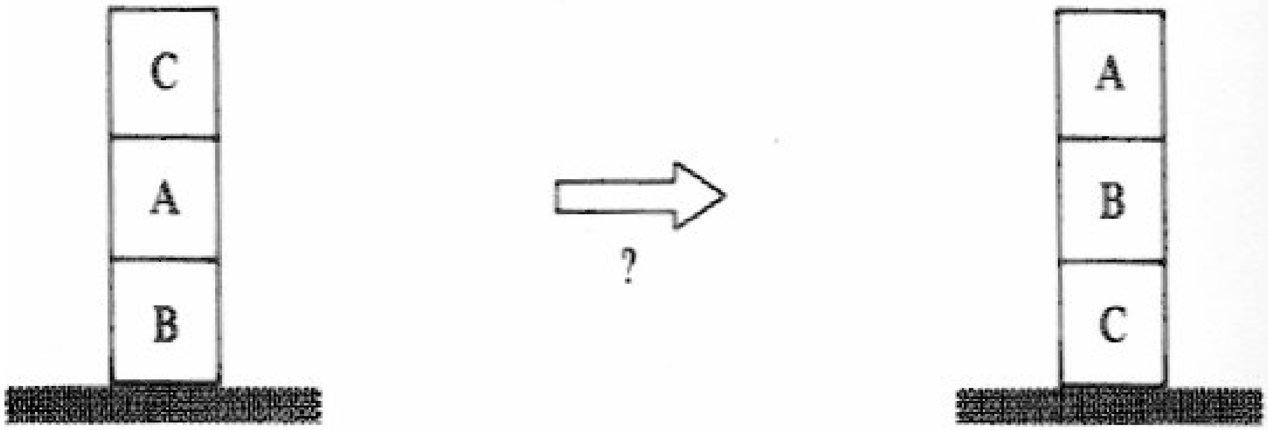


Figure 11.1 A blocks rearrangement problem.

- Two types of concept
  - problem situations
  - legal moves, or actions

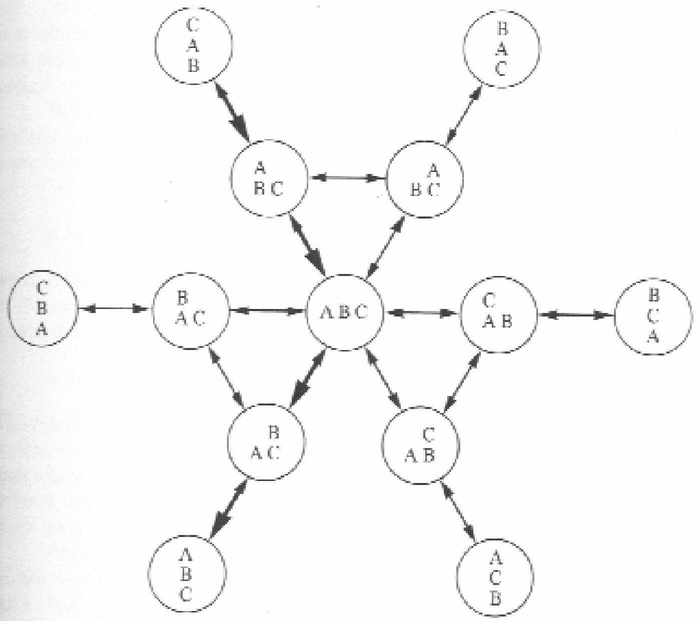
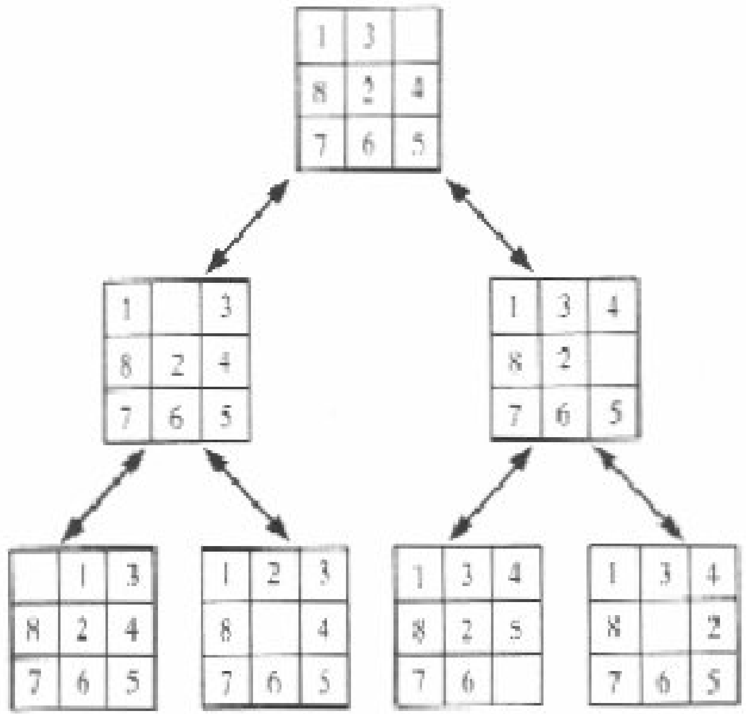


Figure 11.2 A state-space representation of the block manipulation problem. The indicated path is a solution to the problem in Figure 11.1.

- a state space
- a start node and a goal condition (goal nodes)

An eight puzzle



State Space

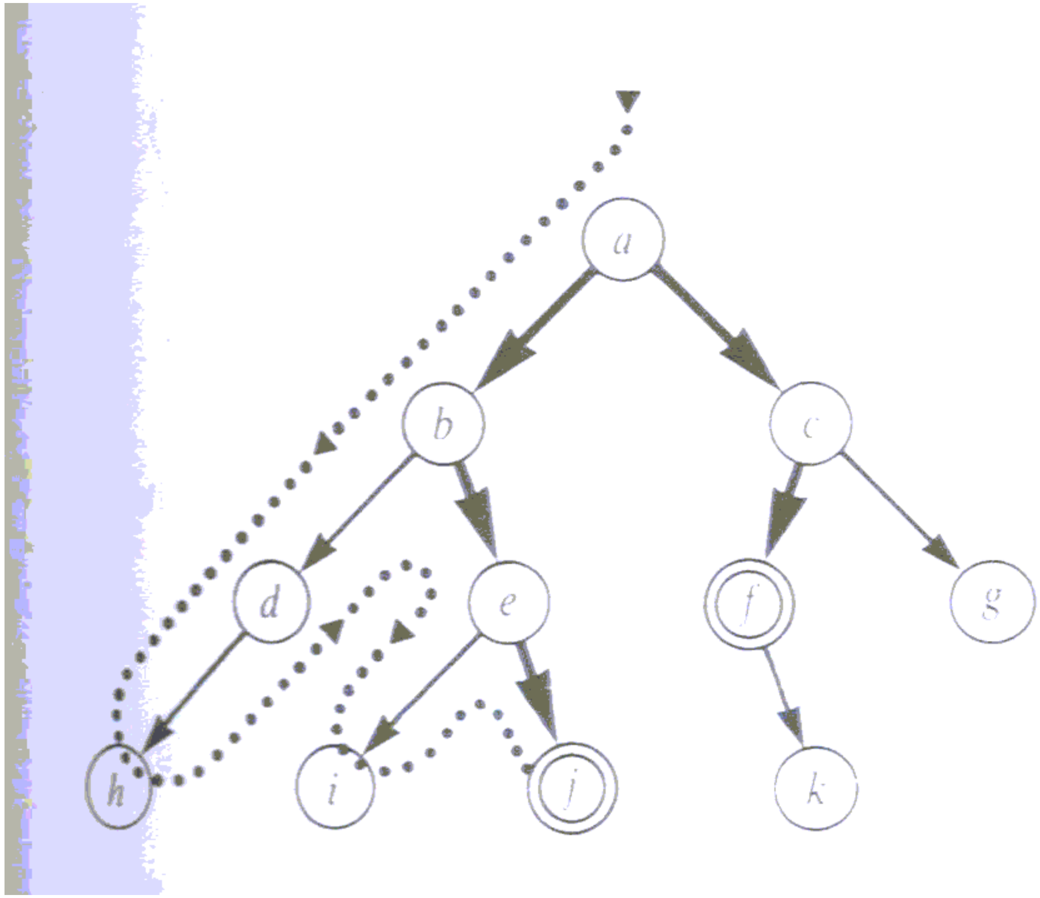
- Represented by relations
  - s(X,Y)
  - s(X,Y,Cost)
- Represented
  - explicitly by a set of facts, or
  - implicitly by stating the rules for computing the successor nodes of a given node

Introductory concepts and examples

- **A problem situation should be represented**
  - in a compact way, and
  - in a way that enables efficient execution of operations required
- **An example representation**
  - the block manipulation problem as a list of stacks

    [[c,a,b],[ ],[ ]], [[a,b,c],[ ],[ ]],
      [[ ],[a,b,c],[ ]],[[ ],[ ],[a,b,c]]

Depth-first search and iterative deepening



Depth-first search and iterative deepening

Depth-first search

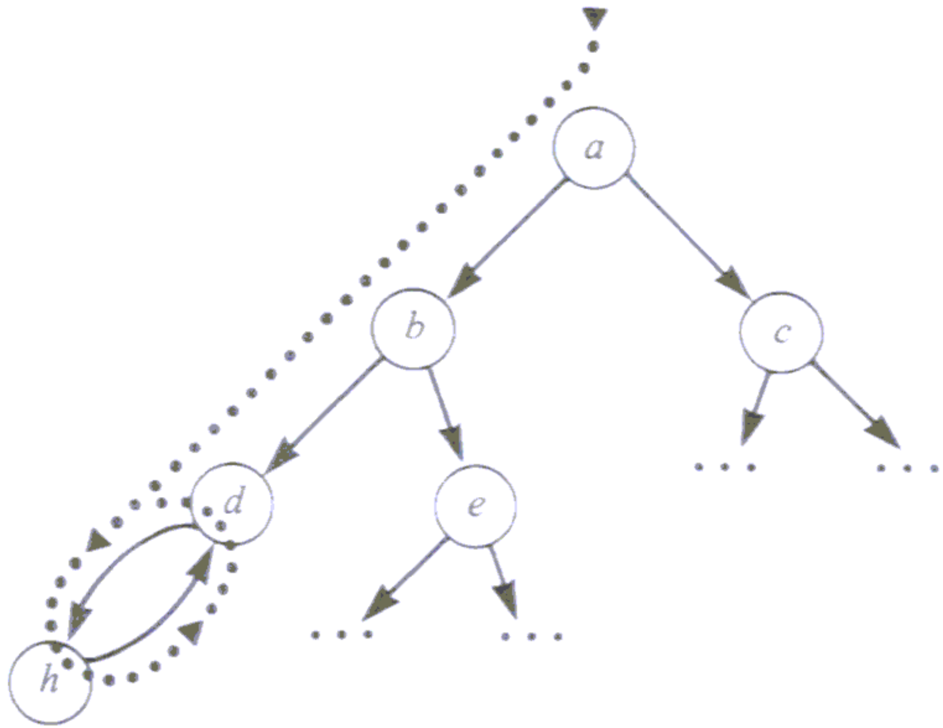
- To find a solution path **Sol** from a given node **N** to some goal node:
  - **N** is a goal node, or
  - There is a successor node **N1** of **N** such that there is a path **Sol1** from **N1** to a goal node.

```
solve(N,[N]) :- goal(N).
solve(N,[N|Sol1]) :- s(N,N1), solve(N1,Sol1).
?- solve(a,Sol).
```

Depth-first search and iterative deepening

⦿ Problems of DFS

- Cycle



Depth-first search and iterative deepening

⊙Problems of DFS

- Cycle: Detecting Cycles

depthfirst(Path,Node,Solution)

```
solve(Node,Solution) :-
            depthfirst([ ],Node,Solution).
depthfirst(Path,Node,[Node|Path]) :-
            goal(Node).
depthfirst(Path,Node,Sol) :-
            s(Node,Node1),
            not member(Node1,Path),
            depthfirst([Node|Path],Node1,Sol).
```

Depth-first search and iterative deepening

⦿ Problems of DFS

- Infinite non-cyclic branches

depthfirst2(Node,Solution,Maxdepth)

```
depthfirst2(Node,[Node],_) :- goal(Node).
depthfirst2(Node,[Node|Sol],Maxdepth) :-
        Maxdepth > 0,
        s(Node,Node1),
        Max1 is Maxdepth - 1,
        depthfirst2(Node1,Sol,Max1).
```

Depth-first search and iterative deepening

⦿ Enhancing DFS

- Iterative deepening

%path(Node1,Node2,Path)

```
path(Node,Node,[Node]).
path(FirstNode,LastNode,[LastNode|Path]) :-
  path(FirstNode,OneButLast,Path),
  s(OneButLast,LastNode),
  not member(LastNode,Path).
```

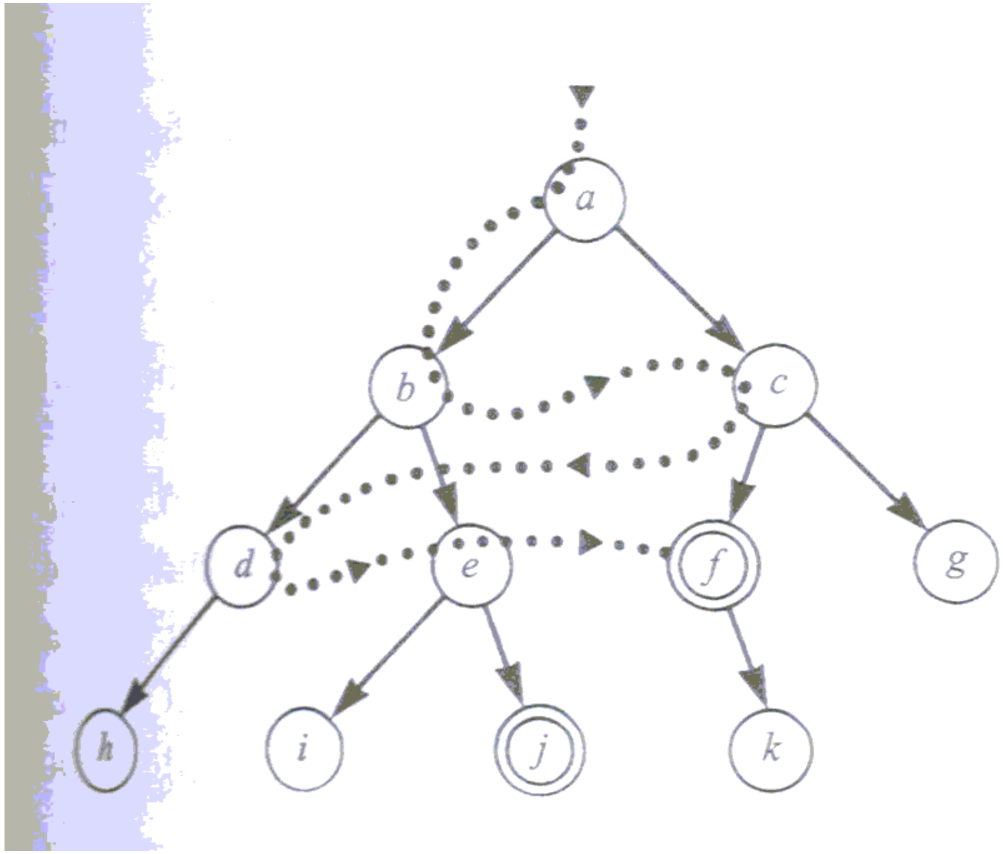
```
?- path(a,Last,Path).
Last = a      Last = b        Last = c        Last = d
Path = [a]; Path = [b,a]; Path = [c,a]; Path = [d,b,a];
```

Depth-first search and iterative deepening

⊙ Iterative deepening

```
path(Node,Node,[Node]).
path(FirstNode,LastNode,[LastNode|Path]) :-
  path(FirstNode,OneButLast,Path),
  s(OneButLast,LastNode),
  not member(LastNode,Path).
depthfirstiterativedeepening(Node,Solution) :-
  path(Node,GoalNode,Solution),
  goal(GoalNode).
```

Breadth-first search



Breadth-first search

- Given a set of candidate paths
  - if the first path contains a goal node as its head
    - then this is a solution of the problem, otherwise
  - remove the first path from the candidate set and generate the set of all possible one-step extensions of this path, adding this set of extensions at the end of the candidate set, and execute breadth-first search on this updated set.

Breadth-first search

⦿ Implementation

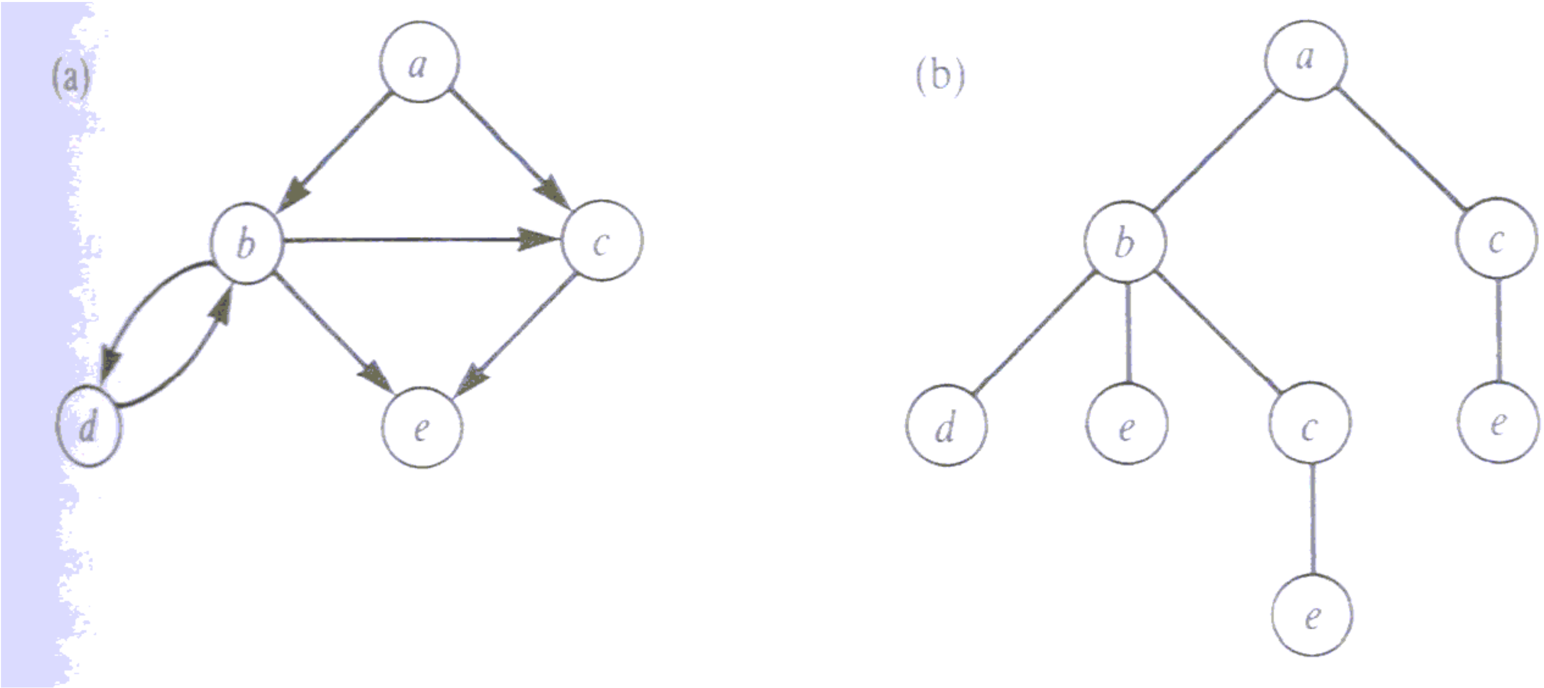
```
solve(Start,Solution) :- bfirst([[Start]],Solution).
bfirst([[Node|Path]|_],[Node|Path]) :- goal(Node).
bfirst([Path|Paths],Solution) :-
    extend(Path,NewPaths),
    conc(Paths,NewPaths,Paths1),
    bfirst(Paths1,Solution).
extend([Node|Path],NewPaths) :-
   bagof([NewNode,Node|Path], (s(Node,NewNode),
    not member(NewNode,[Node|Path])),NewPaths),
   !.
extend(Path,[ ]).
```

Breadth-first search

⦿ A more efficient implementation

```
solve(Start, Solution) :-
  breadthfirst([[Start] | Z]-Z, Solution).
breadthfirst([[Node | Path] | _]-_,  [Node | Path] ) :-
   goal(Node).
breadthfirst([Path | Paths]-Z, Solution) :-
   extend(Path, NewPaths),
   conc(NewPaths, Z1, Z),
   Paths \== Z1,
   breadthfirst(Paths – Z1, Solution).
```

Analysis of basic search techniques



Analysis of basic search techniques

⦿ Pros and cons of search techniques

- Breadth-first
- Depth-first
- Iterative deepening
- Bidirectional: breadth-first in both directions

Summary

- Introductory concepts and examples
- Depth-first search and iterative deepening
- Breadth-first search
- Analysis of basic search techniques