Design and Analysis of Algorithm

Lecture-24:
Branch and Bound

Contents

Assignment problem

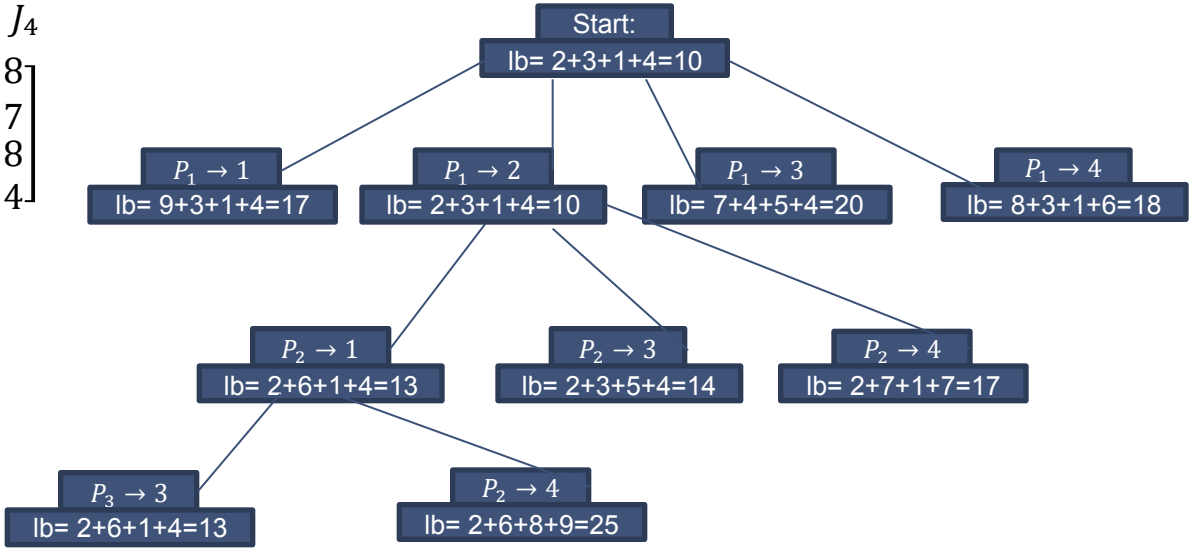
Consider the problem of assigning ***n*** people to ***n*** jobs so that the total cost of the assignment is as small as possible.

$$C = \begin{array}{c} \\ P_1 \\ P_2 \\ P_3 \\ P_4 \end{array} \begin{array}{c} \begin{array}{cccc} J_1 & J_2 & J_3 & J_4 \end{array} \\ \begin{bmatrix} 9 & 2 & 7 & 8 \\ 6 & 4 & 3 & 7 \\ 5 & 8 & 1 & 8 \\ 7 & 6 & 9 & 4 \end{bmatrix} \end{array}$$

It can be observed that the cost of any solution, including an optimal one, cannot be smaller than the sum of the smallest elements in each of the matrix's rows.

Assignment problem

$$C = \begin{array}{c} \\ P_1 \\ P_2 \\ P_3 \\ P_4 \end{array} \begin{array}{c} \begin{array}{cccc} J_1 & J_2 & J_3 & J_4 \end{array} \\ \begin{bmatrix} 9 & 2 & 7 & 8 \\ 6 & 4 & 3 & 7 \\ 5 & 8 & 1 & 8 \\ 7 & 6 & 9 & 4 \end{bmatrix} \end{array}$$



Conditions to terminate a path

- The value of the node's bound is not better than the value of the best solution seen so far.
- The node represents no feasible solutions because the constraints of the problem are already violated.
- The subset of feasible solutions represented by the node consists of a single point and hence no further choices can be made

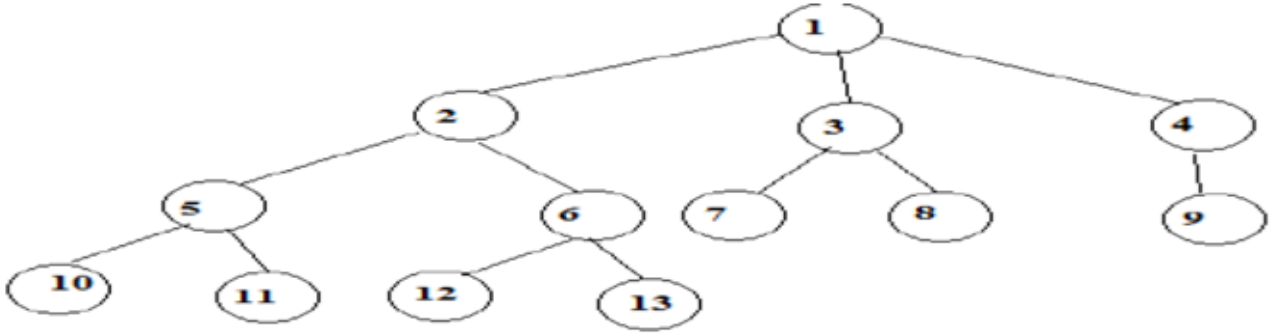
Question

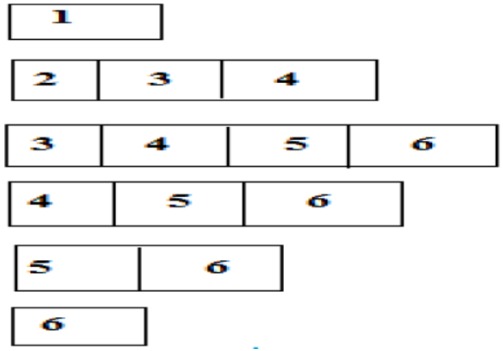
When using the branch and bound method in integer programming maximization problem, the stopping rule for branching is to continue until

1. the objective function is zero.
2. the new upper bound exceeds the lower bound.
3. the new upper bound is less than or equal to the lower bound or no further branching is possible.
4. the lower bound reaches zero.

FIFO Branch and Bound

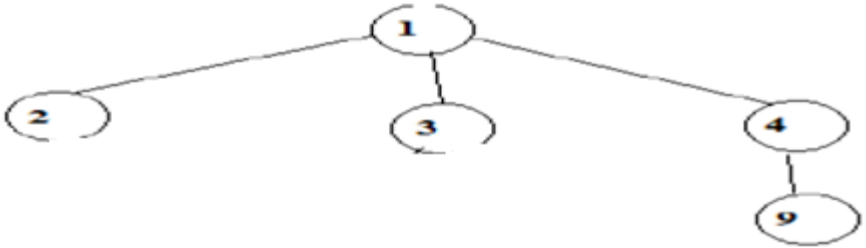
FIFO Branch and Bound is BFS.
In this approach, Children of E-node or live node are inserted in a queue.





LIFO Branch and Bound

LIFO Branch and Bound is DFS.
In this approach, Children of E-node or live node are inserted in a stack.



Least Cost Search

In both LIFO and FIFO branch-and-bound the selection rule for the next E-node is rather rigid and in a sense blind.

The selection rule for the next E-node does not give any preference to a node that has a very good chance of getting the search to an answer node quickly

The search for an answer node can often be speeded by using an “intelligent” ranking function c(-) for live node

Ranking of nodes

The ideal way to assign ranks would be on the basis of the additional computational effort (or cost)needed to reach an answer node from the live node

For any node x, this cost could be	(1) the number of nodes in the subtree x that need to be generated before an answer node is generated or, more simply,
	(2) the number of levels the nearest answer node (in the subtree x) is from x.

Ranking of nodes

The node x is assigned a rank using:

$$c(x) = (h(x) + g(x)$$

where, $c(x)$ *is the cost of* x

$h(x)$ is the cost of reaching x from the root

$g(x)$ is an estimate of the additional effort needed to reach an answer node from x

A search strategy that uses a cost function $c(x) = (h(x) + g(x)$ to select the next E-node would always choose for its next E-node a live node with least cost. Hence such a search is called least cost search.

Ranking of nodes

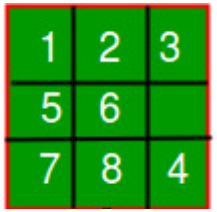
BFS and D-search are special cases of LC-search.

If g(x) = 0 and h(x) = level of node x, then an LC search generates nodes by levels. This is eventually the same as a BFS.

If h(x) = 0 and essentially a D-search. g(x) > g(y) whenever y is a child of x, then the search is

Example

8 Puzzle Problem: The 8-puzzle consists of 8 numbered tiles on a square frame with a capacity of 9 tiles



The only legal moves are ones in which a tile adjacent to the empty spot(ES) is moved to ES

Example

c(x) = f(x) + h(x)
where

f(x) is the length of the path from root to x (the number of moves so far)

h(x) is the number of non-blank tiles not in their goal position