BRANCH AND BOUND

- A Live Node
  - is one which has been generated and all of whose children have not been generated
- A Dead Node
  - is a generated node which is not to be expanded further or all of whose children have been generated
- E-Node
  - is a live node whose children are currently being generated

Methods of generating problem states

- Method 1
  - As soon as a child C of the current E-Node R is generated, this child will be the new E-Node. R will become the E-Node again when the subtree C has been fully explored
- Method 2
  - An E-Node remains so until it is dead

- In both methods, bounding functions are used to kill live nodes without generating all their children

The Depth First node generation with bounding functions is called *backtracking*

- The state generation in which E-Node remains as an E-Node until it is dead leads to the *branch and bound* method

FIFO/LIFO BB

- A BFS-like state space search is called FIFO search as the list of live nodes is a first-in-first-out list (i.e. queue)
- A D-search like state space search is called LIFO (Last in First out) search as the list of live nodes is a last-in-first-out list (i.e. stack)
- Bounding functions are used to avoid the generation of subtrees that do not contain an answer node

Least Cost (LC) Search

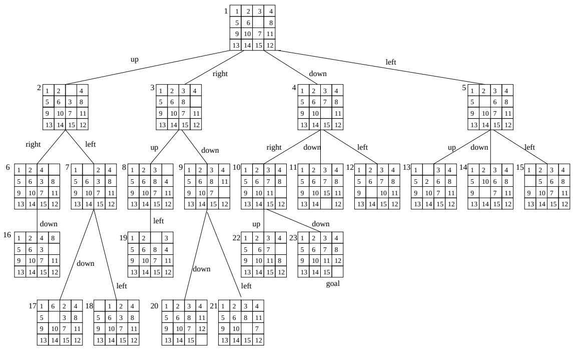
- The selection for FIFO or LIFO Branch and Bound is rigid and blind
- Using ranking function $c^*(x)$ for live nodes
- Next E-Node is selected on the basis of the ranking function (smallest $c^*(x)$)
- Let $g^*(x)$ be an estimate of the additional effort needed to reach an answer node from x, $h(x)$ to be the cost of reaching x from the root, and $f(.)$ to be any non-decreasing function, $c^*(x) = f(h(x)) + g^*(x)$

LC-Search

- BFS and D-search are special cases of LC search
- If we set $g^*(x) = 0$ and $f(h(x))$ to be level of node x, we have BFS
- If we set $f(h(x)) = 0$ and $g^*(x) \leq g^*(y)$ whenever y is a child of x then the search is essentially a D-search
- An LC-search coupled with bounding functions is known as LC-Branch and Bound Search

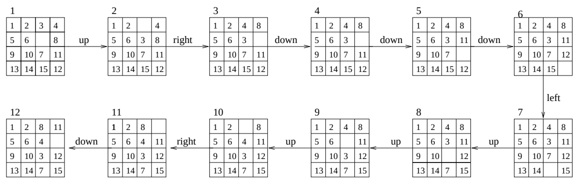
- Usually, if x ia an answer node, $c(x)$ is the cost of reaching x from the root. If x is not an answer node, then $c(x) = \infty$ if the subtree of x contains no answer node; otherwise $c(x)$ is the cost of a minimum cost answer node in the subtree x.
- $c^*(.)$ with $f(h(x)) = h(x)$ can be an approximation of $c(.)$

15-Puzzle State Space (partial)



Edges are labeled according to the direction
in which the empty space moves

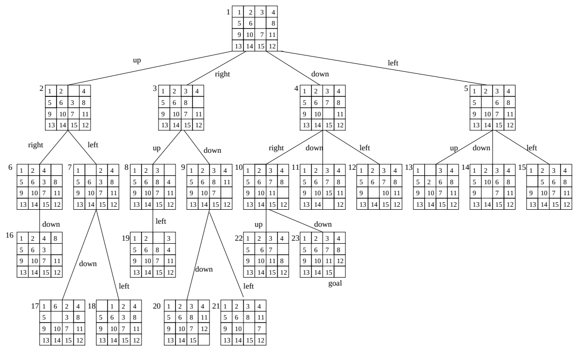
15-Puzzle Using Depth First



LC Search

- The cost $c(x)$ is the length of path from the root to the nearest goal node in the subtree with root x
- $c(1) = c(4) = c(10) = c(23) = 3$
- If such a cost function is known, a very efficient search can be carried out
- However, it is not possible to find $c(.)$
- So, find $c^*(.)$ to approximate $c(x)$
- $c^*(x) = f(x) + g^*(x)$ where $f(x)$ is the length of the path from root to node x and $g^*(x)$ is an estimate of the length of a shortest path from x to a goal node in the subtree with root x

LC-Search Process



Edges are labeled according to the direction in which the empty space moves

LC Search Algorithm

```
Algorithm LCSearch(t)
//Search t for an answer node
{
   if t is an answer node
      then output t and return;
   E := t ; //E-node
   Initialize the list of live nodes to be empty;
   repeat
   {
      for (each child x of E) do
      {
         if (x is an answer node)
            then output the path
            from x to t and return;
         ADD(x) ; // x is a new live node
         (x-> parent) := E //Pointer for path to root
      }
      if (there are no more live nodes) then
      {
         write ("No answer node"); return;
      }
   E := least()
   } until false;
}
```


Bounding

- $c^*(x) \leq c(x)$ will ensure correctness
- If $upper$ is an upper bound on the cost of a minimum cost solution, then all live nodes with $c^*(x) \leq upper$ may be killed

Job Sequencing with Deadlines

- We are given n jobs and one processor. Each job i has associated with it a three tuple (p_i, d_i, t_i). Job i requires t_i units of processing time. If its procesing is not completed by the deadline d_i then a penalty p_i is incurred.

- The objective is to select a subset J of the n jobs such that all jobs in J can be completed by their deadlines. A penalty can be incurred only on those jobs that are not in J. The subset J should be such that the penalty incurred is minimum among all possible subsets of J.

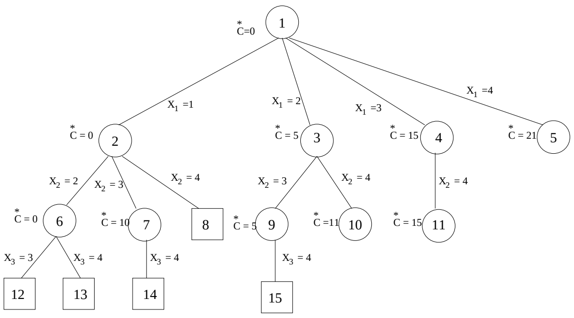
An Instance

- $n = 4$
  - $(p_1, d_1, t_1) = (5, 1, 1)$
  - $(p_2, d_2, t_3) = (10, 3, 2)$
  - $(p_3, d_3, t_3) = (6, 2, 1)$
  - $(p_4, d_4, t_4) = (3, 1, 1)$

- Using the variable tuple size formulation

- The state space tree is given in the next slide

State Space Tree

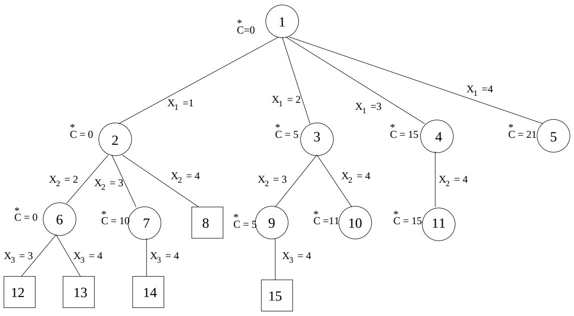


c^* and $upper$

Let S_x be the subset of jobs selected for J at node x.
If $m = \max\{i | i \in S_x\}$, then

$$c^*(x) = \textstyle\sum_{i<m, i \notin S_x} P_i$$
$$upper(x) = \textstyle\sum_{i \notin S_x} P_i$$

FIFO Branch and Bound



LC Branch and Bound