// of x.

{
for i=1 to k-1 do
if ((x[j] = i) // Two in the same column
or (Abs(x[j] - i) = Abs(j-k)))
// or in the same diagonal
then return false;

return true;

}

→ Place(K, i) returns a boolean value that is true if the kth queen can be placed in column i.

→ It tests whether i is distinct from all previous values x[1], ..., x[k-1] and whether there is no other queen on the same diagonal.

→ Its computing time is O(K-1).

→ The algorithm is invoked by NQueens(1, n).

Algorithm NQueens(K, n)

// This procedure prints all possible placements of n queens on an n×n chessboard so that they are nonattacking.

for i=1 to n do
{ if Place(K, i) then
{

```
        x[k] = i;
      if (k = n) then write(x[1:n]);
      else NQueens(k+1, n);
    }
  }
}
```

Graph Coloring:

-> Let G be a graph and m be a given positive integer.

-> The problem is to discover whether the nodes of G can be colored in such a way that no two adjacent nodes have the same color yet only m colors are used. This is termed as m-colorability decision problem.

-> The m-colorability optimization problem asks for the smallest integer m for which the graph G can be colored. This integer is referred to as the chromatic number of the graph.

-> Graph is represented by its adjacency matrix $G[1:n, 1:n]$ where $G[i,j] = 1$ if (i,j) is an edge of G, and $G[i,j] = 0$ otherwise.

-> The colors are represented by integers.

-> The solutions are given by the n tuple $(x_1, \ldots, x_n)$, where x_i is the color of node i.

→ Function mColoring is begun by first assigning the graph to its adjacency matrix and assinging array x[] to zero. Then invoke mColoring(1).

Algorithm mColoring(k)
// The graph is represented by its boolean adjacency
// matrix G[1:n, 1:n]. All assignments of 1, 2, m to
// the vertices of the graph such that adjacent vertices
// are assigned distinct integers are printed. k is the
// index of the next vertex to color.
{
  repeat
  { // Generate all legal assignments for x[k].
      NextValue(k); // Assign to x[k] a legal color.
      if (x[k] = 0) then return // No new color
                                              // Possible.
      if (k = n) then // At most m colors have been
                              // used to color the n vertices

          write (x[1:n]);
      else mColoring(k+1);
  } until(false);
}

→ Function NextValue produces the possible colors for x_k after x_1 through x_{k-1} have been defined.

Algorithm NextValue(k)
// x[1] ... x[k-1] have been assigned integer values

// in the range [1..m] such that adjacent
// vertices have distinct integers. A value for x[k] is
// determined in the range [0, m]. x[k] is assigned
// the next highest numbered color while maintaining
// distinctness from the adjacent vertices of vertex k.
// If no such color exits, x[k] = 0.

```
{
  repeat
  {
     x[k] = (x[k] + 1) mod (m+1); // Next highest color
    if (x[k] = 0) then return; // All colors have been used
    for j = 1 to n do
     { // check if this color is distinct from adj colors
      if ((G[k, j] ≠ 0 and (x[k] = x[j]))
      then break;
    }
   if (j = n+1) then return; // New color found
  } until (false); // otherwise try to find another color
}
```

→ An upper bound on the computing time of mColoring can be arrived at by noticing that the number of internal nodes in the state space tree is $\sum_{i=0}^{n-1} m^i$.

→ At each internal node, $O(mn)$ time is spent by NextValue to determine the children corresponding to legal coloring.

→ Hence the total time is $O(nm^n)$.

Sum of Subsets:

→ Given positive numbers w_i, $1 \le i \le n$ and m, the problem is to find all subsets of the w_i whose sums are m.

→ All solutions are k-tuples $(x_1, x_2, \ldots, x_k)$, $1 \le k \le n$ and different solutions may have different-sized tuples.

→ Each solution to the problem can also be represented by an n-tuple $(x_1, x_2, \ldots, x_n)$ such that $x_i \in \{0,1\}$, $1 \le i \le n$. $x_i = 0$ if w_i is not chosen and $x_i = 1$ if w_i is chosen

→ The bounding functions we use are

$$B_k(x_1, \ldots, x_k) = \text{true iff } \sum_{i=1}^{k} w_i x_i + \sum_{i=k+1}^{n} w_i \ge m$$

$$\text{and } \sum_{i=1}^{k} w_i x_i + w_{k+1} \le m$$

Algorithm SumofSub(s, k, r)

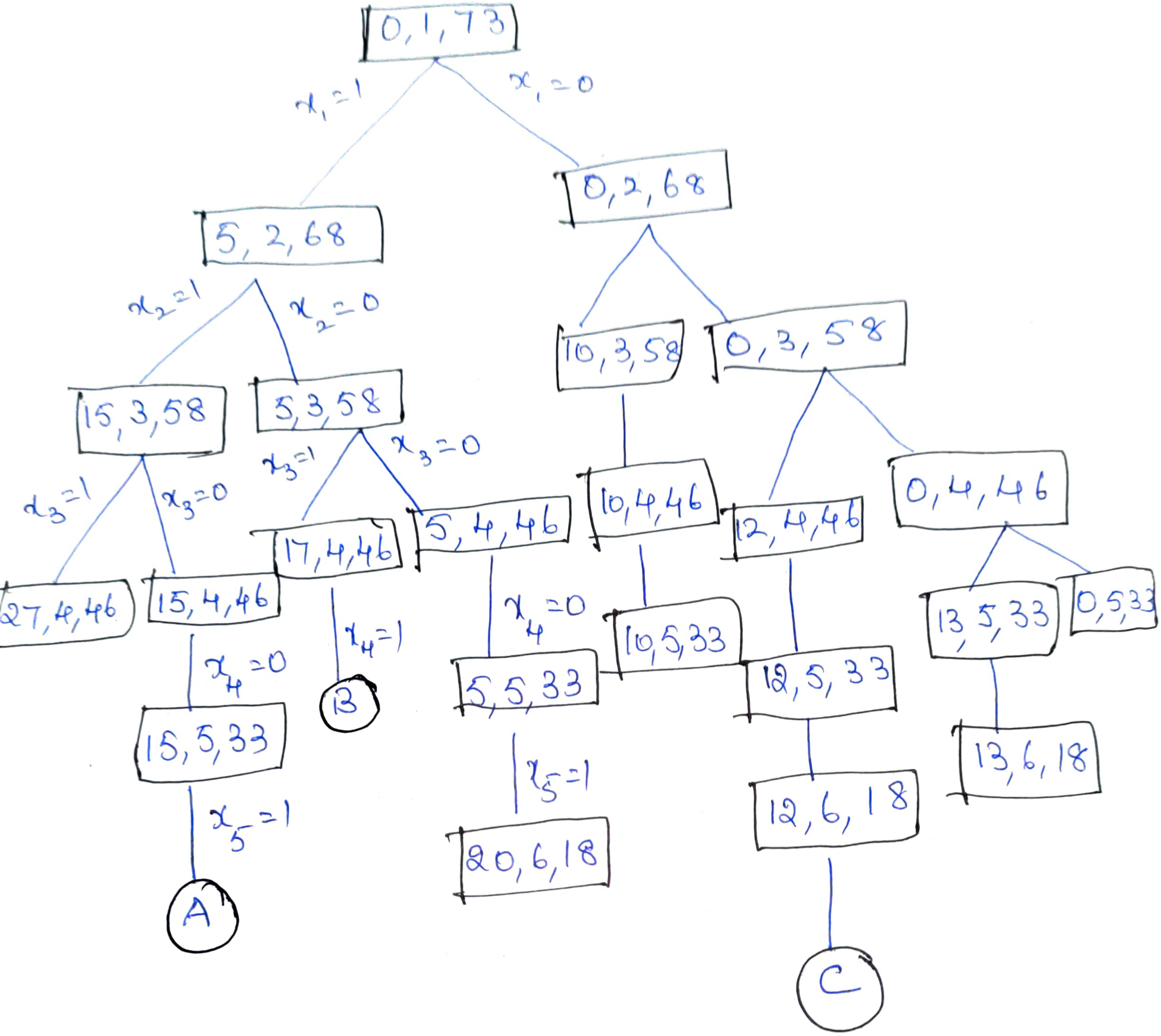
// Find all subsets of w[1:n] that sum to m. The
// values of x[j], $1 \le j < k$, have already been
// determined. $s = \sum_{j=1}^{k-1} w[j] * x[j]$ and $r = \sum_{j=k}^{n} w[j]$.

// The w[i]s are in non decreasing order. It
// is assumed that $w[1] \leq m$ and $\sum_{i=1}^{n} w[i] \geq m$

{
  // Generate left child. Note: $s + w[k] \leq m$ since
  // B_{k-1} is true.
  $x[k] = 1$;
  if $(s + w[k] = m)$ then write $(x[1:k])$;
      // subset found
  // There is no recursive call here as $w[j] > 0, 1 \leq j \leq n$
  else if $(s + w[k] + w[k+1] \leq m)$
  then SumOfSub$(s + w[k], k+1, r - w[k])$;
  // Generate right child and evaluate B_k.
  if $((s + r - w[k] \geq m)$ and $(s + w[k+1] \leq m))$ then
  { $x[k] = 0$;
    SumOfSub$(s, k+1, r - w[k])$;
  }
}

example

Let $w[1:6] = \{5, 10, 12, 13, 15, 18\}$ and $m = 30$.
Find all possible subsets of w that sum to m.
Do this using SumOfSub. Draw the portion of the
state space tree that is generated.



The solutions are

(1, 1, 0, 0, 1), (1, 0, 1, 1) and (0, 0, 1, 0, 0, 1).