28/10/19

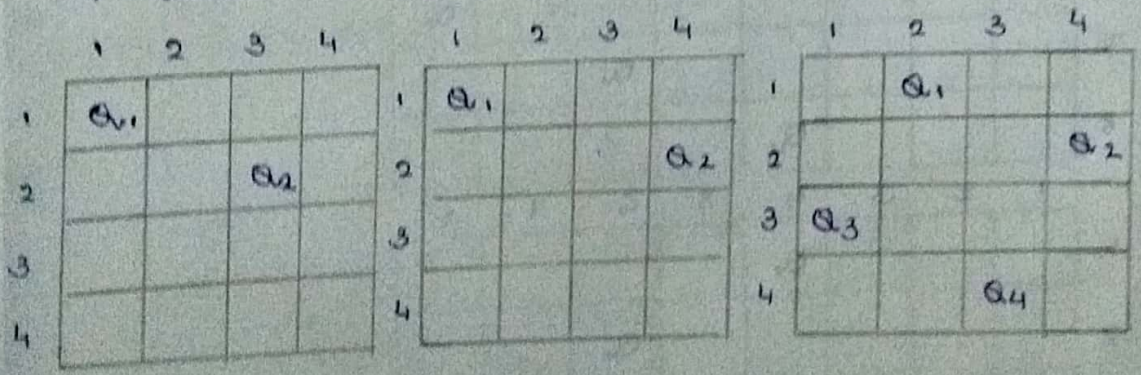
MODULE-V

Back tracking: It is an algorithmic design approach.

N-queen's problem

Given a n×n chess board. our problem is to place N queen's in n×n chess board without placing them in same row, same column and diagonal. The queen's are arranged in such a way that they won't attack each other by placing them in same row, same column and diagonal.

* 4 queens: Q_1, Q_2, Q_3, Q_4

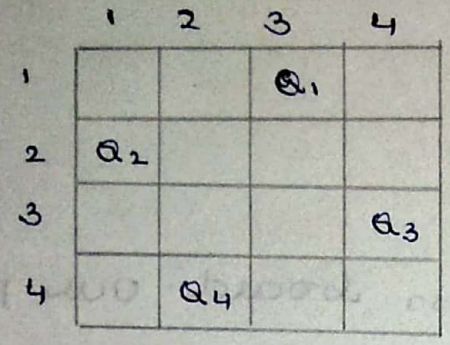


(ii) Here, Q_1 and Q_3 are placed, Q_3 cannot be placed. so, in next to 4×4 matrix, back tracking Q_2.

(iii) Back tracking. Q_1

∴ solution = (2, 4, 1, 3)

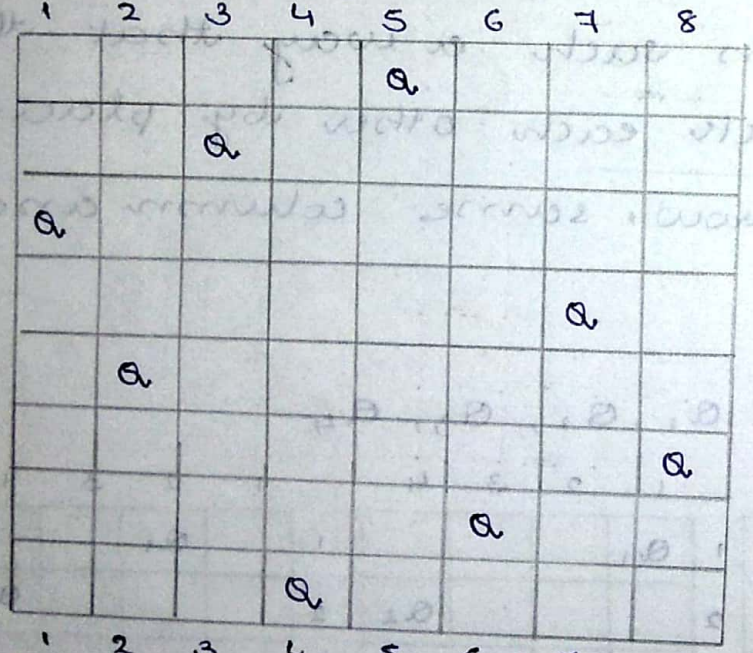
→ another solution, is the mirror image of the obtained solution.



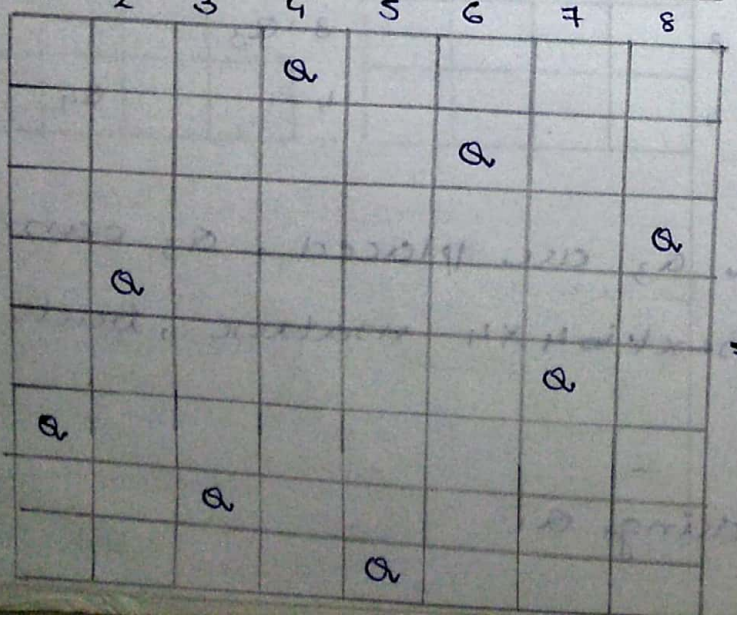
∴ solution = (3, 1, 4, 2)

* 8 queen's problem

(i)

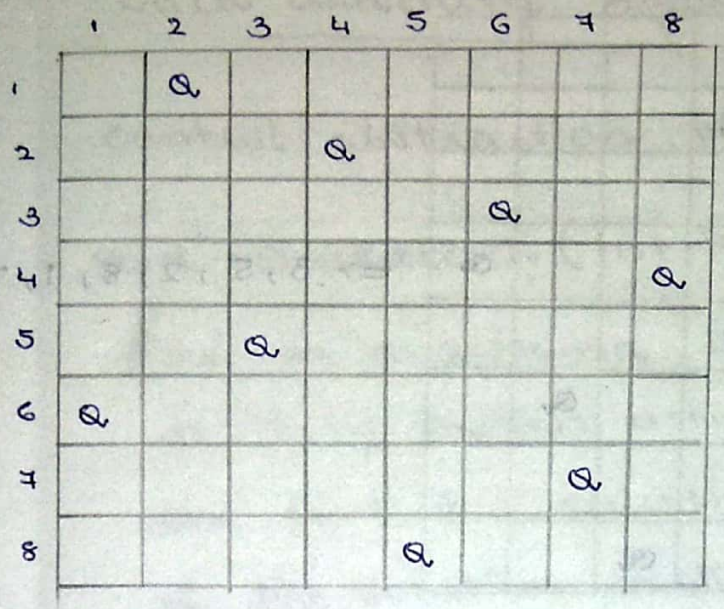


=> 5, 3, 1, 7, 2, 8, 6, 4

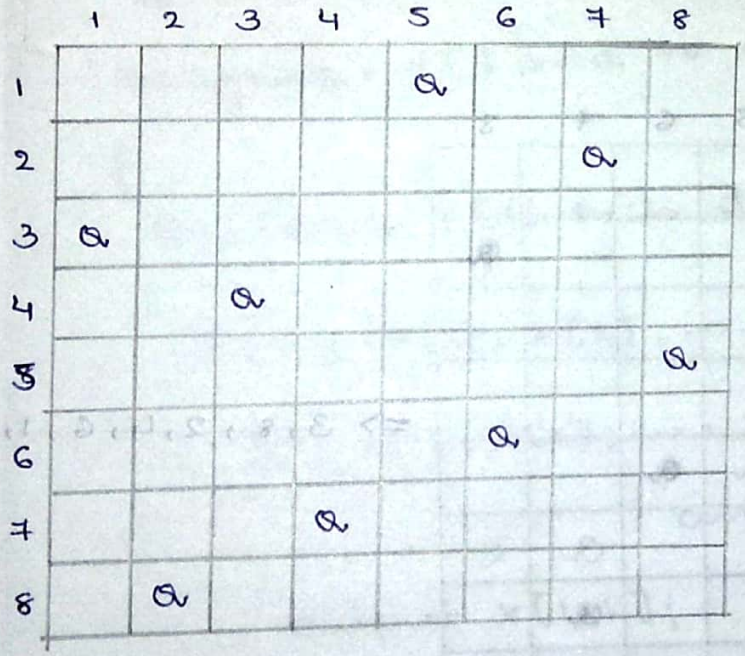


=> 4, 6, 8, 2, 7, 1, 3, 5

(iii)

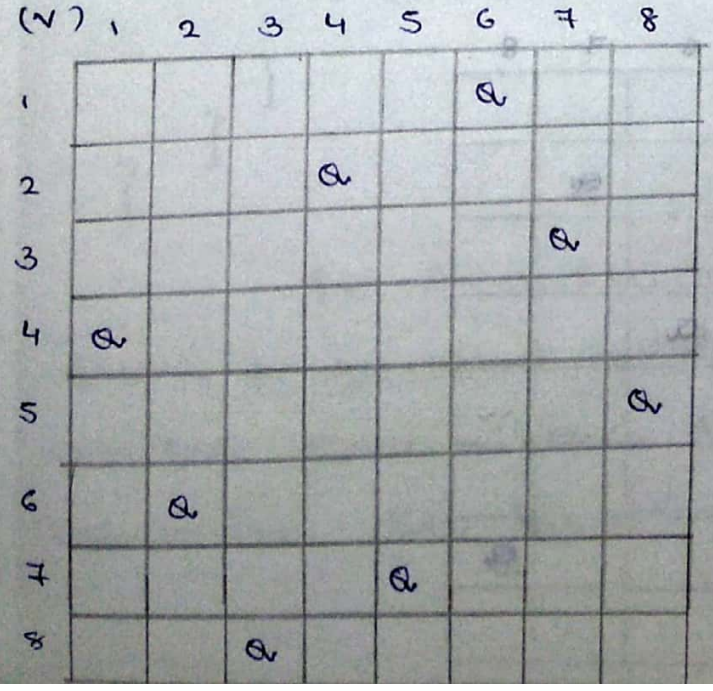


$\Rightarrow$ 2, 4, 6, 8, 3, 1, 7, 5

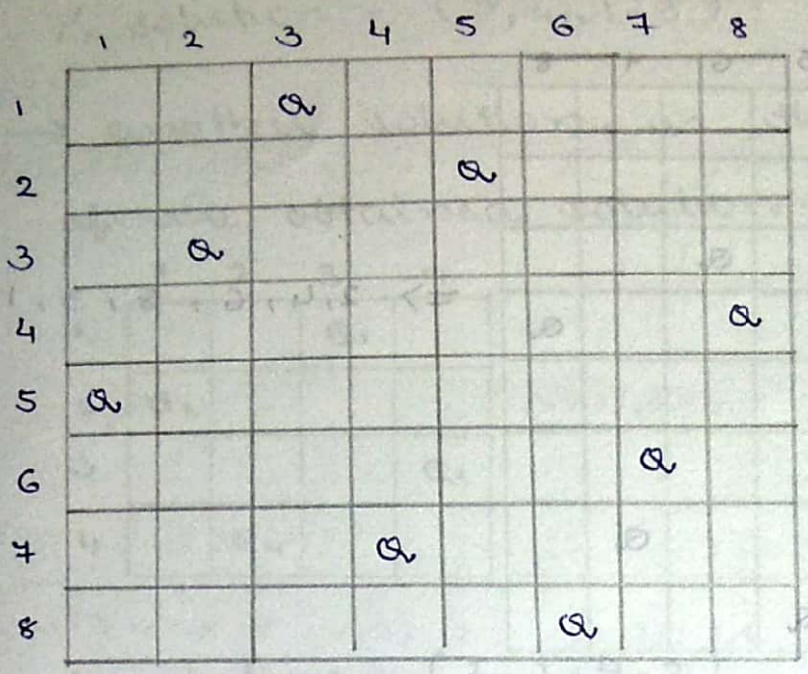


$\Rightarrow$ 5, 7, 1, 3, 8, 6, 4, 2

(iv)

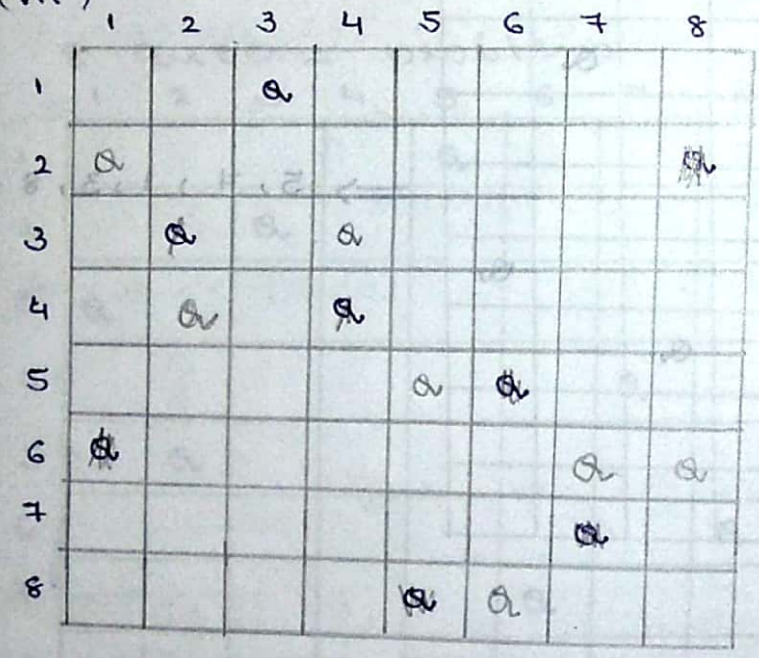


$\Rightarrow$ 6, 4, 7, 1, 8, 2, 5, 3

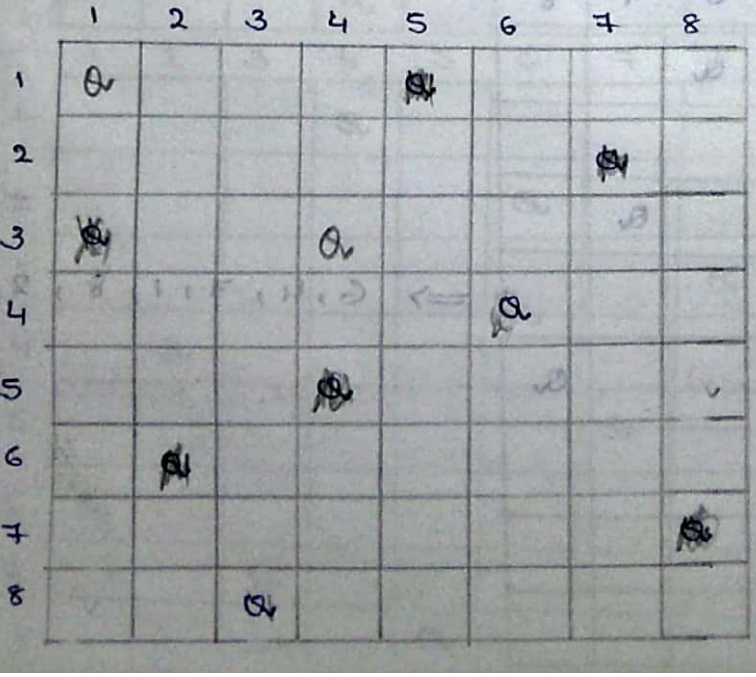


=> 3, 5, 2, 8, 1, 7, 4, 6

(vii)



=> 3, 8, 2, 4, 6, 1, 7, 5



29/10/19 Back tracking Algorithms

control abstraction of back-tracking

```
void Backtrack(int k)
// This is a schema that describes the back-
   tracking process using recursion. On entering,
   the first k-1 values x[1], x[2], ....., x[k-1]
   of the solution vector x[1:n] have been
   assigned. x[] and n are global.
{
   for (each x[k] such that x[k] ∈ T[x[1], ....
                                    x[k-1]] {
      if (Bk(x[1], x[2], ....., x[k])) {
         if (x[1], x[2], .... x[k] is a path to an
                                    answer node)
            output x[1:k];
            if (k<n) Backtrack(k+1);
      }
   }
}
```

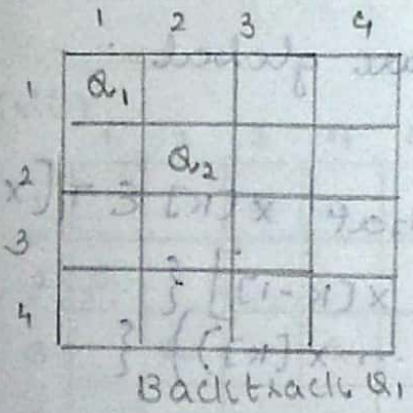
In backtracking, elements are chosen from a specified set, so that the sequence satisfies some criteria that maximizes or minimizes the functions.

Two type of constraint:

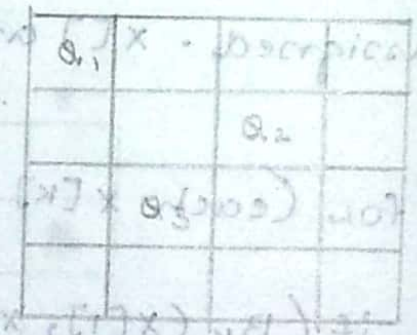
(i) Explicit and implicit constraint

Explicit constraints are ruled that restrict each x_i to take on values only from a given set. Implicit is vice versa.

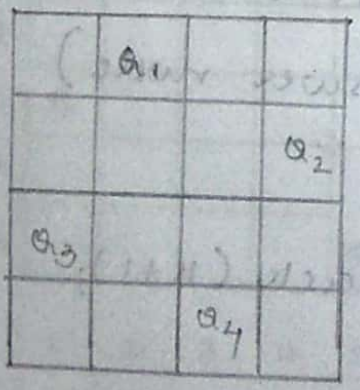
State space Tree of 4-queens problem.

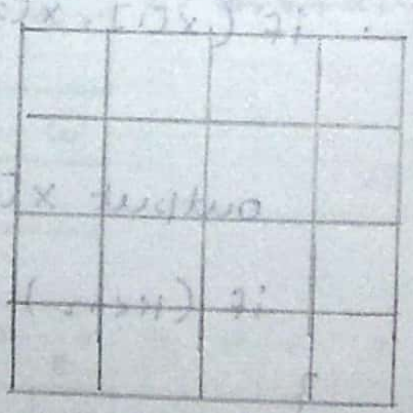


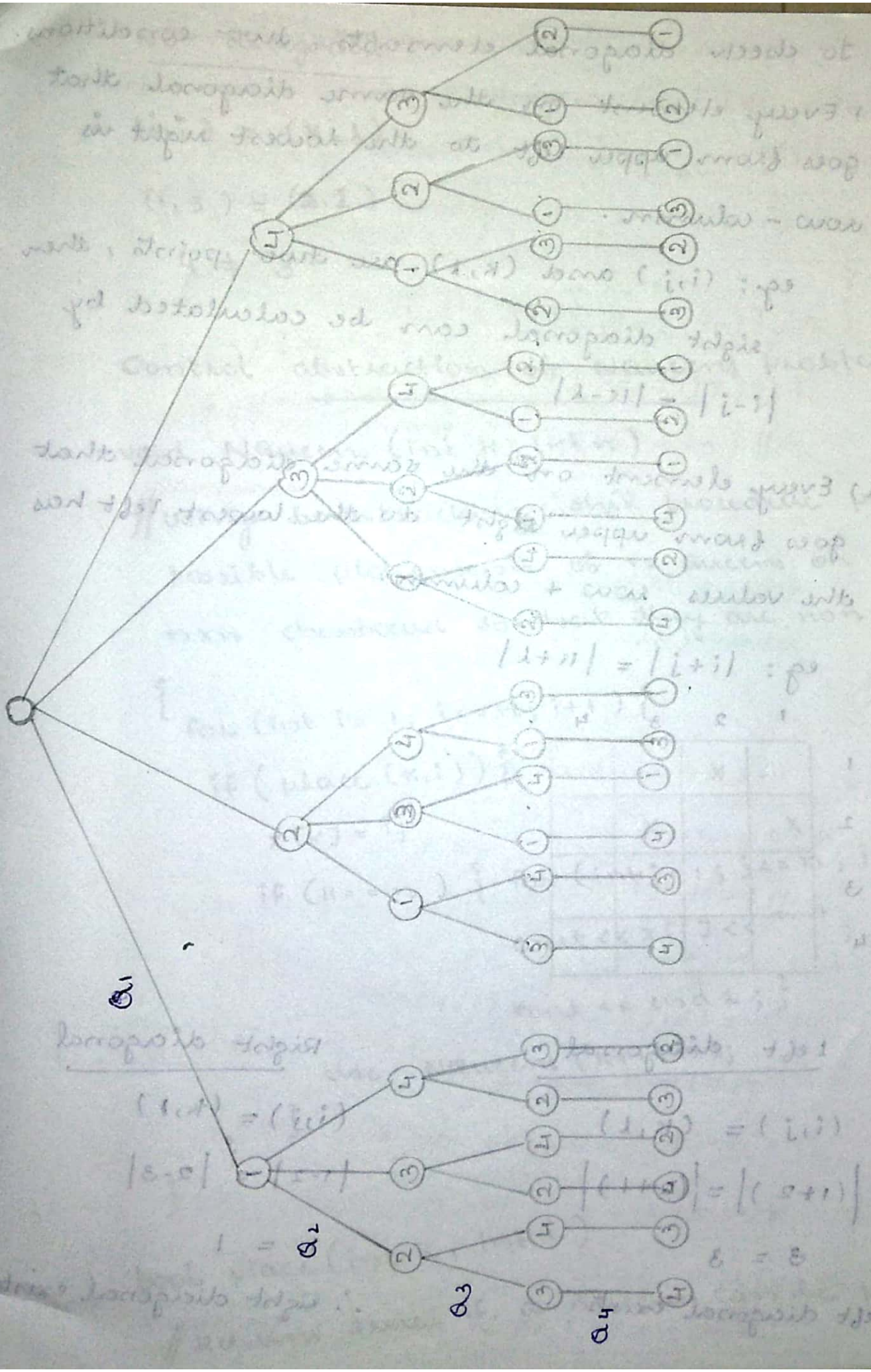
Backtrack Q_1



Backtrack Q_2







To check diagonal elements, two conditions

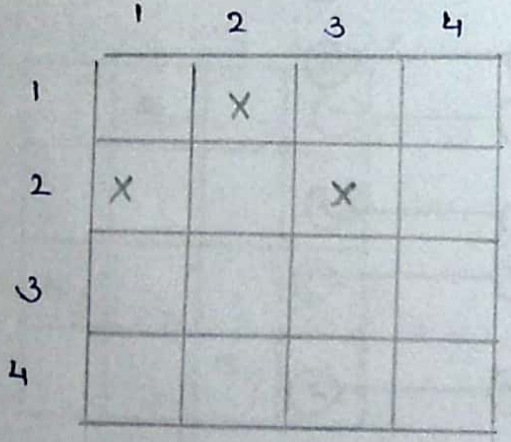
(i) Every element on the same diagonal that goes from upper left to the lowest right is row - column.

eg: (i,j) and (k,l) are two points, then right diagonal can be calculated by

$$|i-j| = |k-l|$$

(ii) Every element on the same diagonal that goes from upper right to the lowest left has the values row + column.

eg: $|i+j| = |k+l|$



Left diagonal

$(i,j) = (k,l)$

$|(1+2)| = |(2+1)|$

$3 = 3$

∴ left diagonal exists

Right diagonal

$(i,j) = (k,l)$

$|1-2| = |2-3|$

$1 = 1$

∴ right diagonal exists

Not diagonal

$(i, j) = (k, l)$

$(1, 3) = (2, 1)$

$4 \neq 3$

Control abstraction of NQueens problem

```
void NQueens (int k, int n)
// Using backtracking, this procedure prints all
   possible placements of n queens on an
   nxn chessboard so that they are non attacking
{
  for (int i = 1; i<=n; i++) {
    if (place (k,i)) {
      x[k] = i;
      if (k==n) { for (int j = 1; j<=n; j++)
                    cout << x[j] << ' ';
                  cout << endl; }
      else NQueens (k+1, n);
    }
  }
}

bool place (int k, int i)
// Returns true if a queen can be placed in
```

kth row and ith column. Otherwise it returns false. x[] is a global array whose first (k-1) values have been set.

abs(r) returns the absolute value of r.

```
{
for(int j = 1; j <= k-1; j++)
  if( (x[j] == i) // Two in the same column
     || (abs(x[j] - i) == abs(j - k)))
                  // or in the same diagonal
    return(false);
  return(true);
}
```

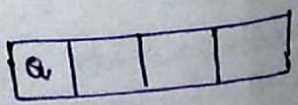
eg: 4-queens problem

k = 1, n = 4

(k, n) = (1, 4)

i = 1 to i <= 4

place(k, i) => (1, 1)



x[k] => i => x[1] = 1

j = 1 to k-1 => j = 1 to 1.

x[1] == 1

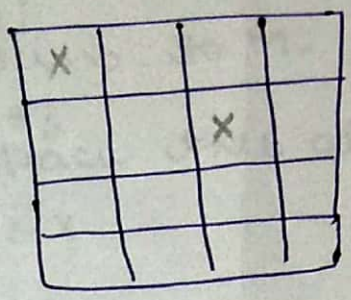
return true

if(k == n) => 1 == 4, false, Nqueens(2, n)

(k, n) = (2, 4)

i = 2

place (2, 2) => x[2] = 2



j = 2

x[2] = 2 true,

x[j - 1] = abs |j - k|

|2 - 2| = |2 - 2| , return false.

(k, n) = (2, 4)

i = 3

place (2, 3) => x[2] = 3

j = 3

x[3] = 3 true, x[3 - 1] == x[3 - 2]

2 ≠ 1 , return false

Back tracking

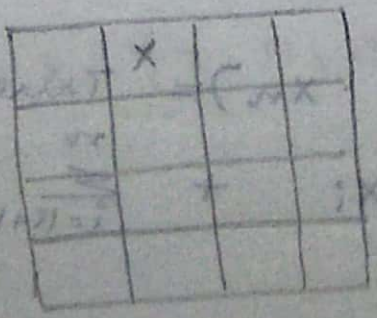
(k, n) = 3, 4

i = 4

place (3, 4) => x[3] = 4

j = 4

x[



k = 1 ; n = 4

(k, n) = (1, 4)

i = 2

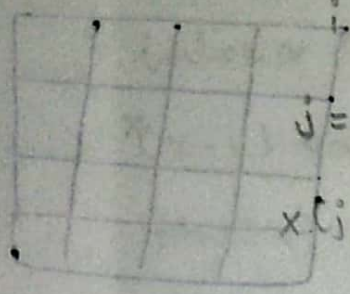
place (1, 2)

j = 1

x[1] = 2

return false

K = 2, n = 4 => (2,4)



i = 3

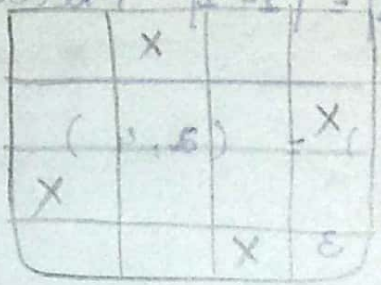
j = 2

x[j] ≠ 3 || x[2-3] ≠ x[3-2]

return false.

i = 3

j = 4



i = 3

j = 1

x[j].

31/10/19

Sum of subset Using Backtracking

Suppose we are given n distinct positive integers or numbers. Find subset of these nos. that sum up to M. $w_1, w_2, \ldots, w_n$ be n nos. The elements x_i of the solution vector is either 1 or 0 depending on whether the weight w_i is included or not.

A simple choice of bounding function,

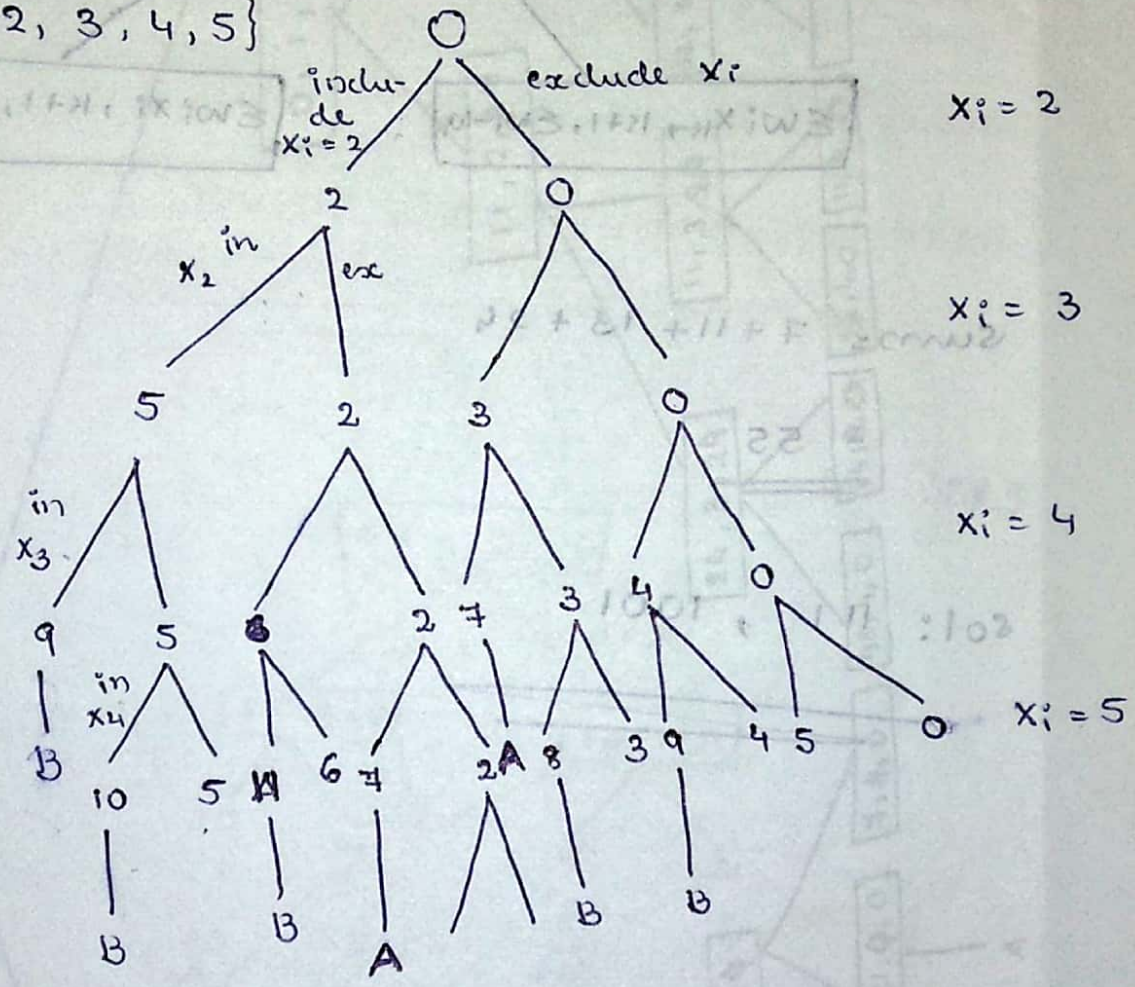
$B_k(x_1, x_2, x_3, \ldots, x_n)$ = True iff,

$$\sum_{i=1}^{k} w_i x_i + \sum_{i=k+1}^{n} w_i \geq M$$

eg: $W = \{4, 3, 2, 5\}$ and $M = 7$. Find all possible subset of W that sums to M.

Draw the portion of state space tree also.

$W = \{2, 3, 4, 5\}$

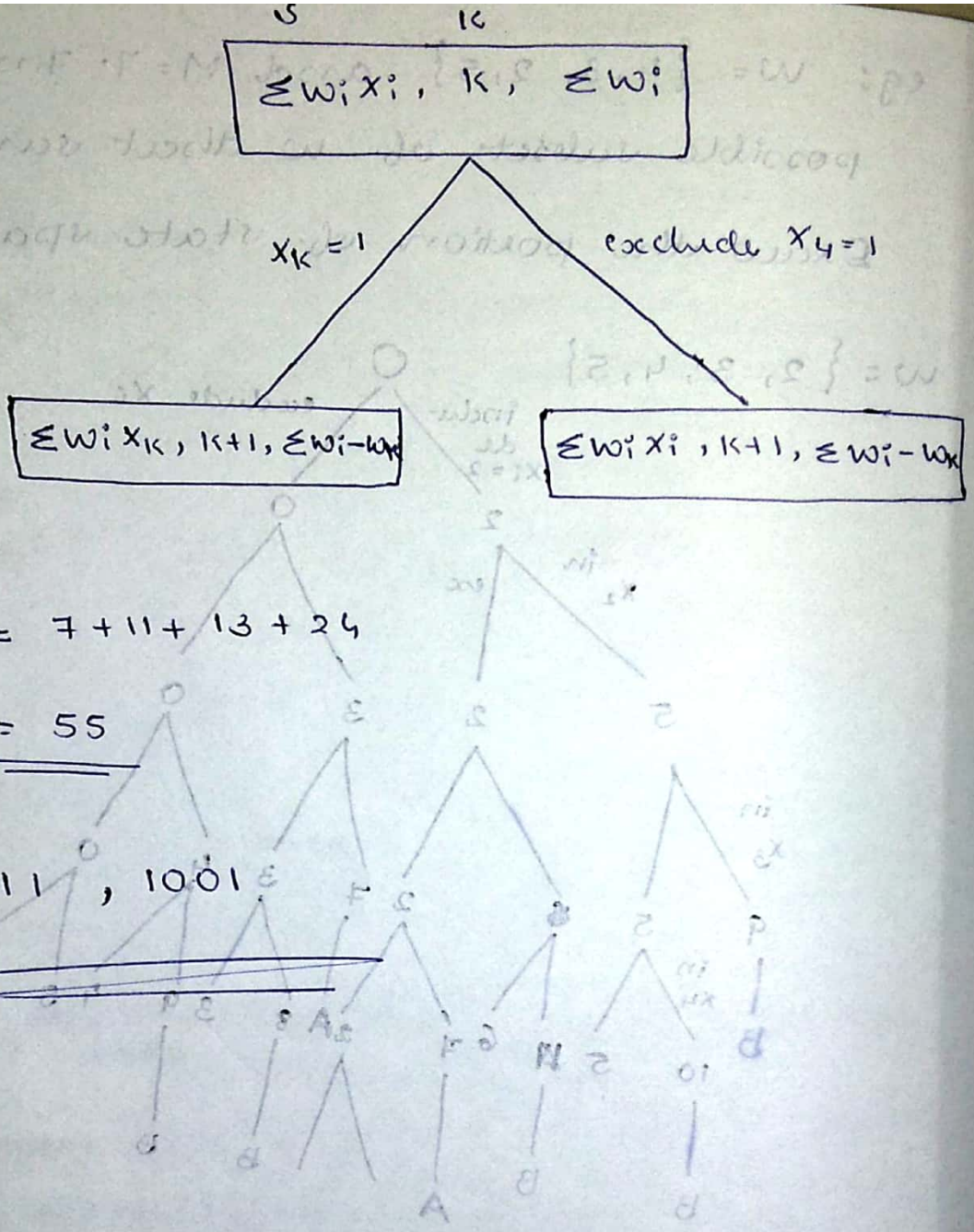


LHS => include x_i

RHS => exclude x_i

011 , 1001

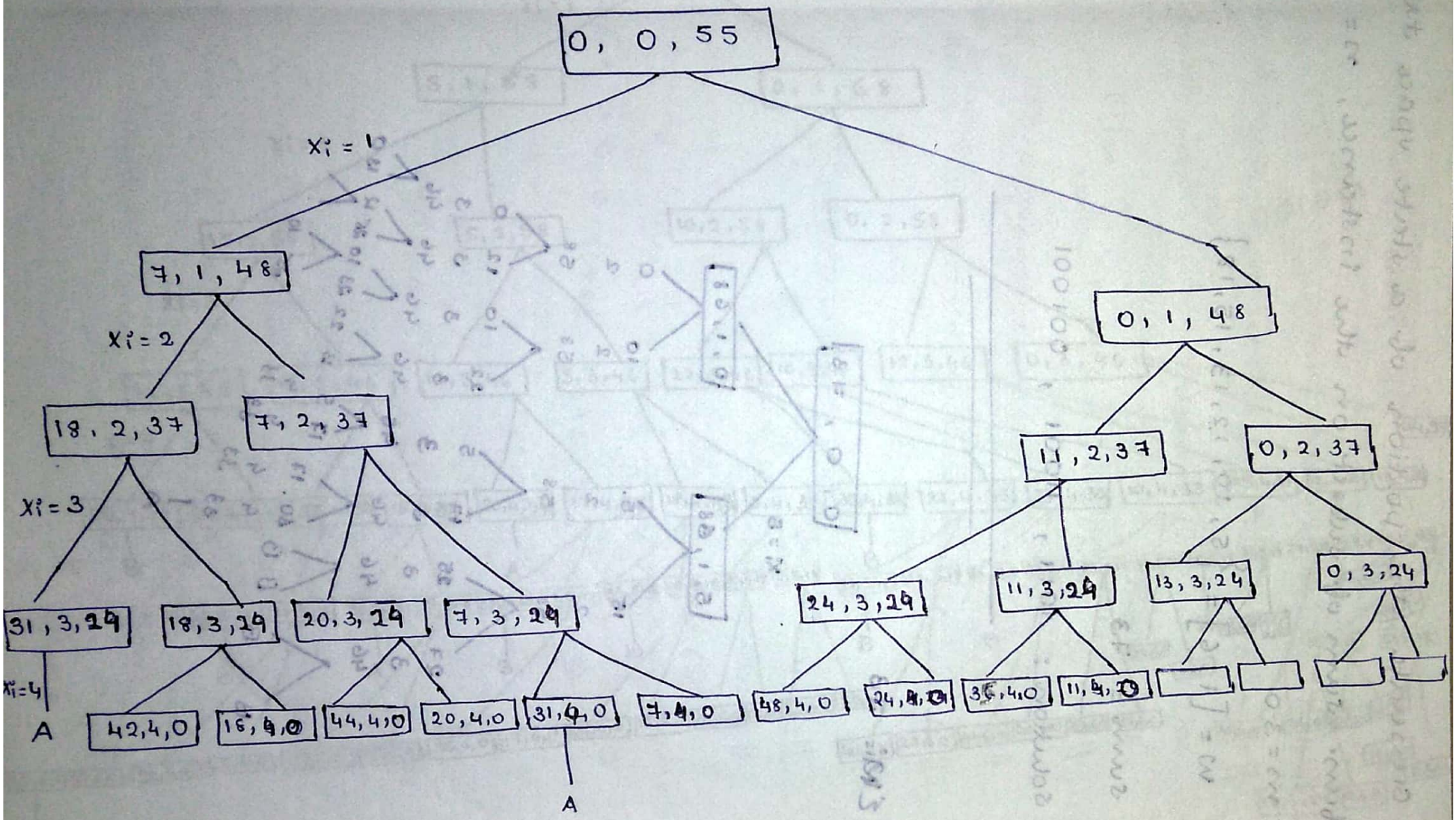
* Generate the portion of state space tree using sum of subset function for the instance $X = \{7, 11, 13, 24\}$ and $M = 31$.



Sum = 7 + 11 + 13 + 24

= 55

Sol: 1111 , 1001


0, 0, 55
Xi = 1
7, 1, 48
0, 1, 48
Xi = 2
18, 2, 37
7, 2, 37
11, 2, 37
0, 2, 37
Xi = 3
31, 3, 29
18, 3, 29
20, 3, 29
7, 3, 29
24, 3, 29
11, 3, 29
13, 3, 24
0, 3, 24
Xi=4
A
42, 4, 0
44, 4, 0
20, 4, 0
31, 4, 0
7, 4, 0
48, 4, 0
A

Generate the portion of a state space tree using fun. sums of subset on the instance, $n = 6$,

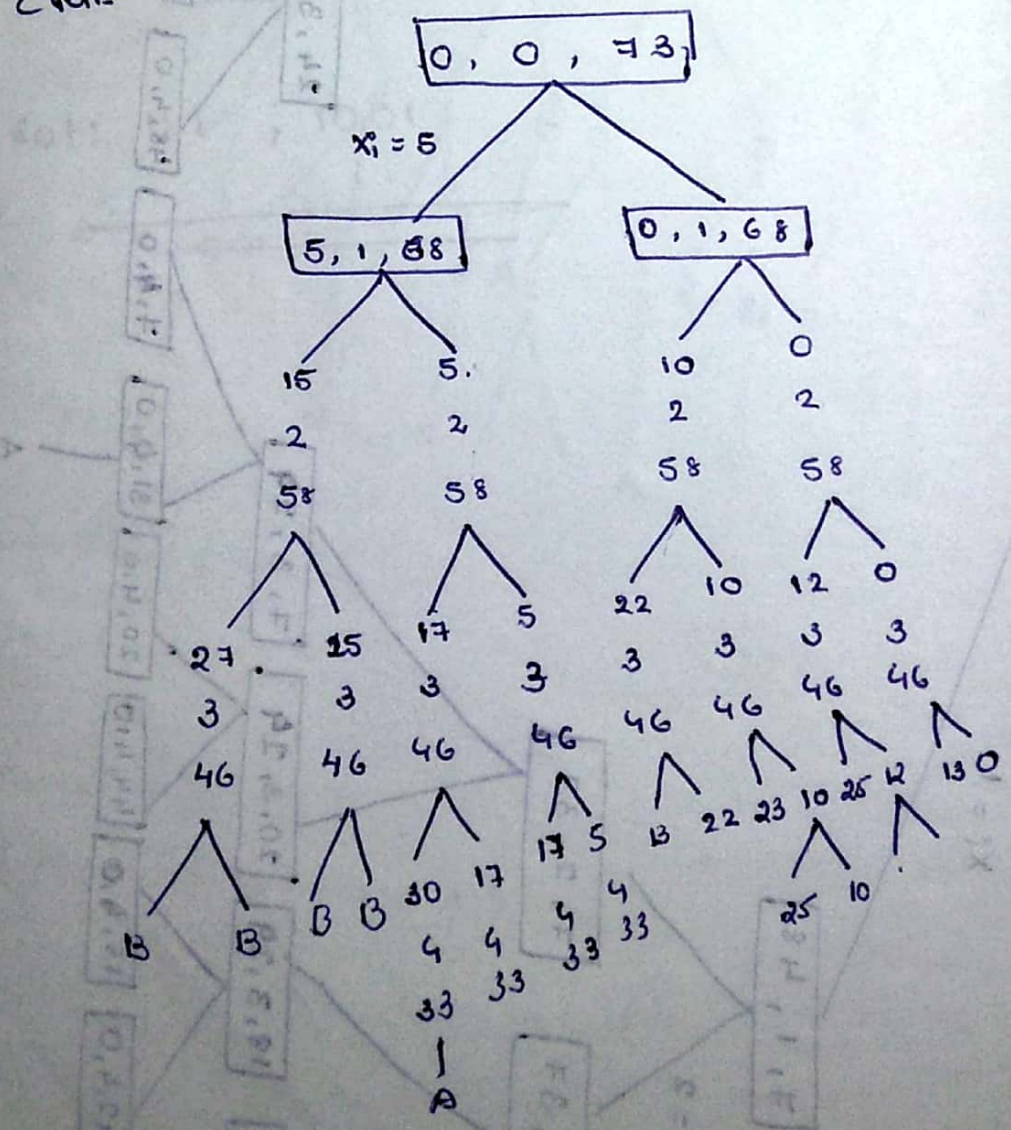
$m = 30$

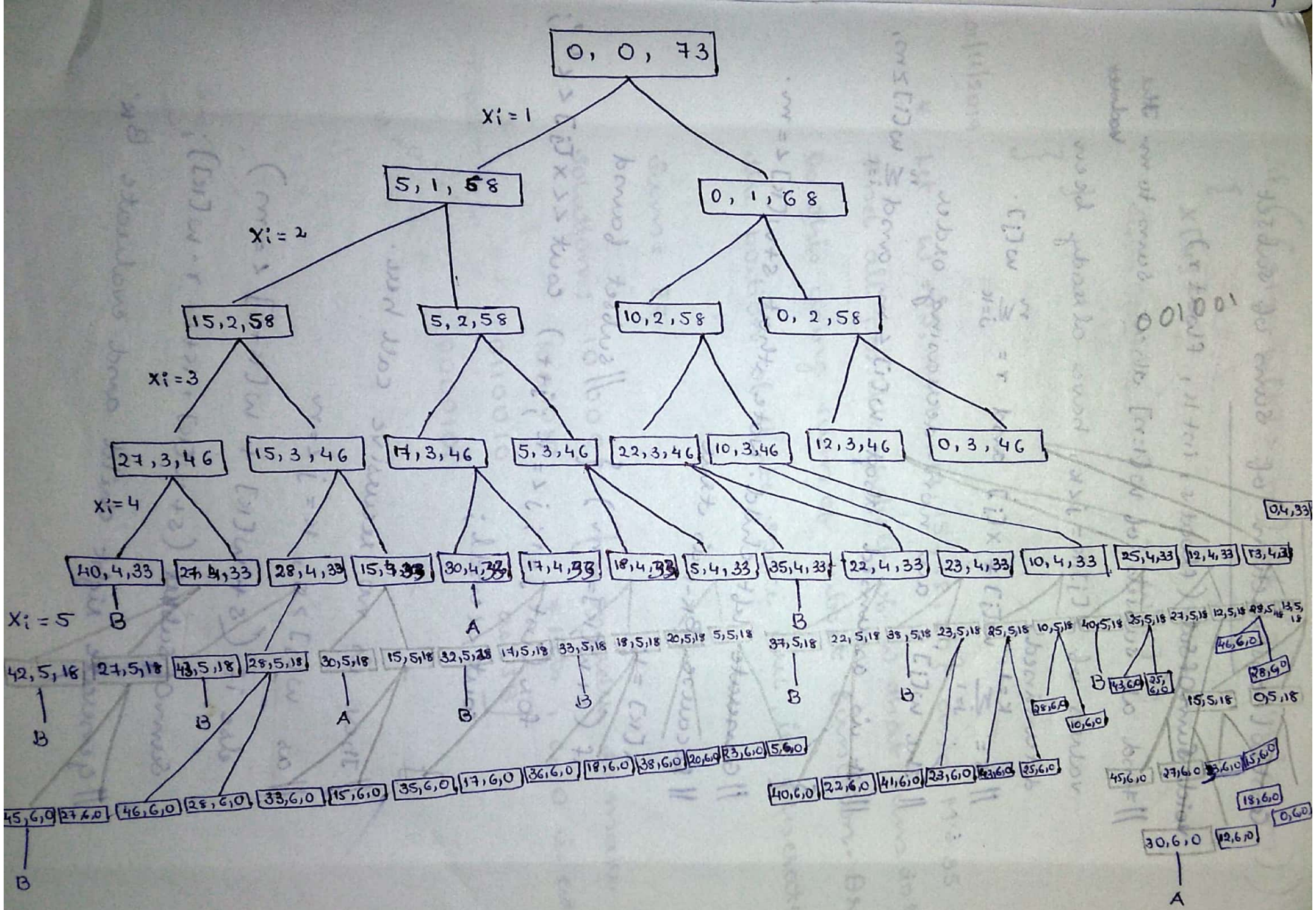
$w = [1:6] = \{5, 10, 12, 13, 15, 18\}$

sum = 73

solution : 1011 , 11001 , 001001

$\sum w_i = 73$




0, 0, 73
Xi = 1
5, 1, 58
0, 1, 68
Xi = 2
15, 2, 58
5, 2, 58
10, 2, 58
0, 2, 58
Xi = 3
27, 3, 46
15, 3, 46
17, 3, 46
5, 3, 46
22, 3, 46
10, 3, 46
12, 3, 46
0, 3, 46
Xi = 4
40, 4, 33
28, 4, 33
30, 4, 33
17, 4, 33
18, 4, 33
5, 4, 33
35, 4, 33
22, 4, 33
23, 4, 33
10, 4, 33
25, 4, 33
12, 4, 33
0, 4, 33
Xi = 5
B
A
B
42, 5, 18
27, 5, 18
28, 5, 18
30, 5, 18
15, 5, 18
32, 5, 18
17, 5, 18
33, 5, 18
18, 5, 18
20, 5, 18
5, 5, 18
37, 5, 18
22, 5, 18
38, 5, 18
23, 5, 18
25, 5, 18
10, 5, 18
40, 5, 18
25, 5, 18
27, 5, 18
12, 5, 18
15, 5, 18
0, 5, 18
45, 6, 0
27, 6, 0
46, 6, 0
28, 6, 0
33, 6, 0
15, 6, 0
35, 6, 0
17, 6, 0
36, 6, 0
18, 6, 0
38, 6, 0
20, 6, 0
23, 6, 0
5, 6, 0
40, 6, 0
22, 6, 0
41, 6, 0
23, 6, 0
25, 6, 0
45, 6, 0
27, 6, 0
15, 6, 0
18, 6, 0
0, 6, 0
30, 6, 0
12, 6, 0
A

Control abstraction of sum of subset

```
void SumOfSub (float s, int k, float r)
// Find all subsets of w[1:n] that sums to m. The values
   values of x[j], 1 <= 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[j]'s are non-decreasing order
// It is assumed that w[1] <= m and sum_{i=1}^{n} w[i] >= m,
 {
// Generate left child. Note that s+w[k] <= m.
// because B_{k-1} is true.
   x[k] = 1;
   if (s+w[k]== m) {   // subset found
      for (int j = 1; j <= k ; j++ ) cout << x[j] << ' ';
      cout << endl;
   }
// There is no recursive call here.
// as w[j] > 0, 1 <= j <= n
   else if (s + w[k] + w[k+1] <= m)
   SumOfSubset (s+w[k], k+1, r-w[k]);
// Generate right child and evaluate B_k
```

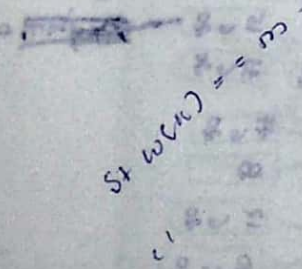
```
if ((s+r-w[k] >= m) && (s+w[k+1] <= m))
  {
    x[k] = 0;
    SumOfSub(s, k+1, r-w[k]);
  }
}
```

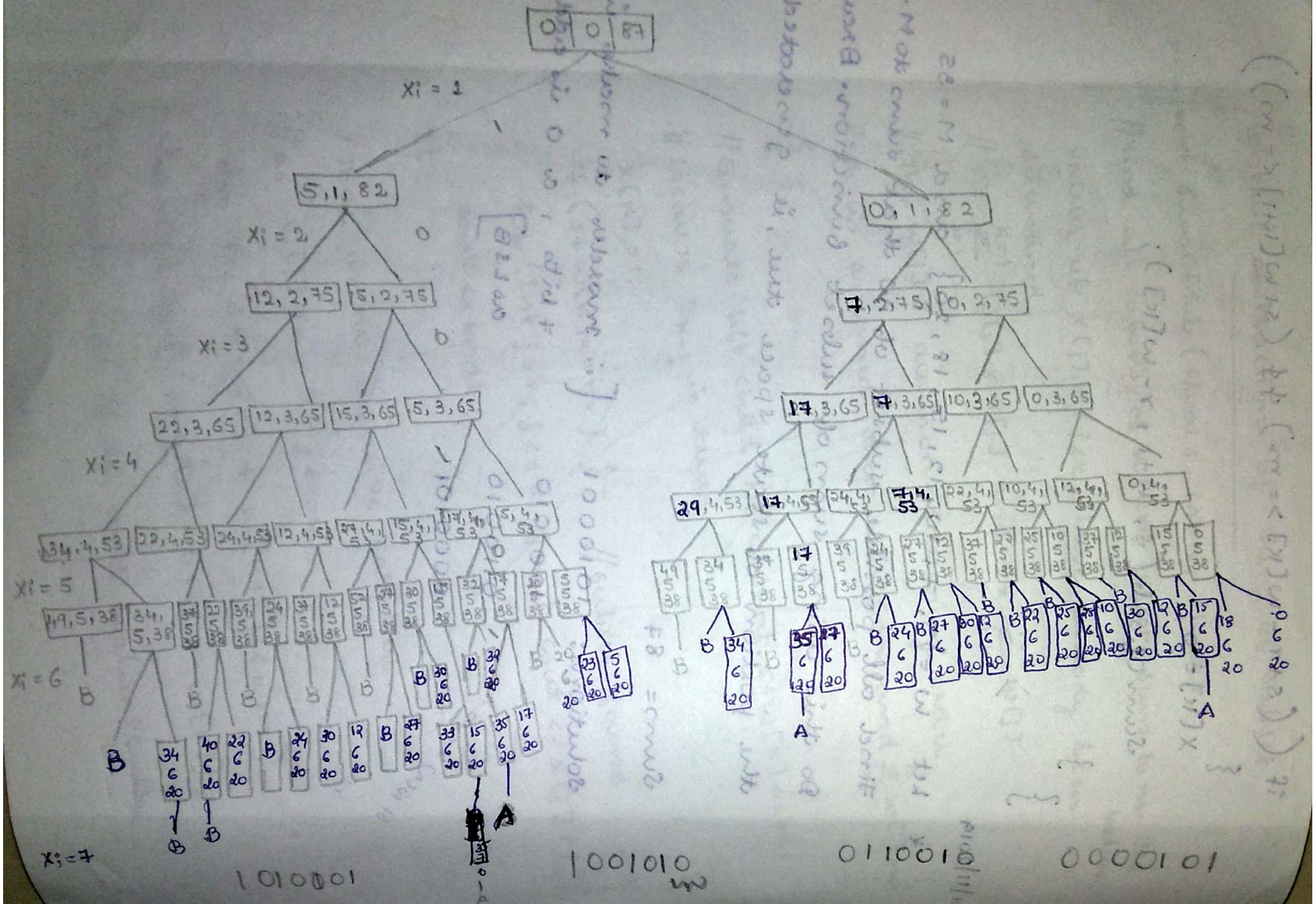
01/11/2019

* Let $w = \{5, 7, 10, 12, 15, 18, 20\}$ and $M = 35$ Find all possible subset of w that sums to M. Do this using sum of subset function. Draw the portion of state space tree, it generated.

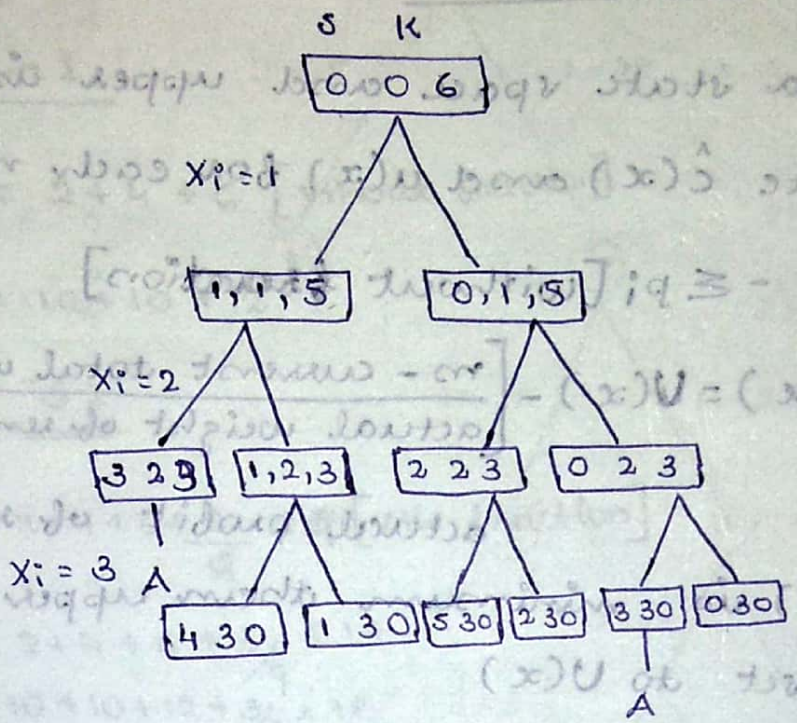
Sum = 87

Solutions: 1010001 [∵ Inserted to make it
1001010 7 bits, a 0 is added
0110010 as LSB]
0000101




Xi = 1
Xi = 2
Xi = 3
Xi = 4
Xi = 5
Xi = 6
Xi = 7
0, 0, 87
5, 1, 82
0, 1, 82
12, 2, 75
5, 2, 75
7, 2, 75
0, 2, 75
22, 3, 65
12, 3, 65
15, 3, 65
5, 3, 65
17, 3, 65
7, 3, 65
10, 3, 65
0, 3, 65
34, 4, 53
22, 4, 53
24, 4, 53
12, 4, 53
29, 4, 53
17, 4, 53
24, 4, 53
22, 4, 53
10, 4, 53
12, 4, 53
0, 4, 53
1010001
1001010
0110010
0000101

* $W = \{1, 2, 3\}$, $M = 3$



1 1 0 , 0 0 1

$x[k] = 1$, $0 + w[1] == 3 \Rightarrow 0 + 1 \neq 3$ false

RHS $0 + 1 + 2 <= 3 \Rightarrow 0 + 1, 2, 6-1$

1, 2, 5

LHS , $0 + 6 - 1 >= 3$ && $0 + 2 <= 3$

$5 >= 3$ && $2 <= 3$

$x[1] = 0$

$0, 2, 6-1 \Rightarrow 0, 1, 5$

$x[k=2]$, $1 + w[2] == 3 \Rightarrow 1 + 2 = 3$

Subset found, stop.

02/11/19

Branch and Bound

1. Draw a state space and upper is ∞
2. compute $\hat{c}(x)$ and $u(x)$ for each node
3. $u(x) = -\sum p_i$ [without fraction]

$$\hat{c}(x) = U(x) - \left[\frac{m - \text{current total weight}}{\text{actual weight of remaining}}\right] *$$

actual profit of remaining obj.

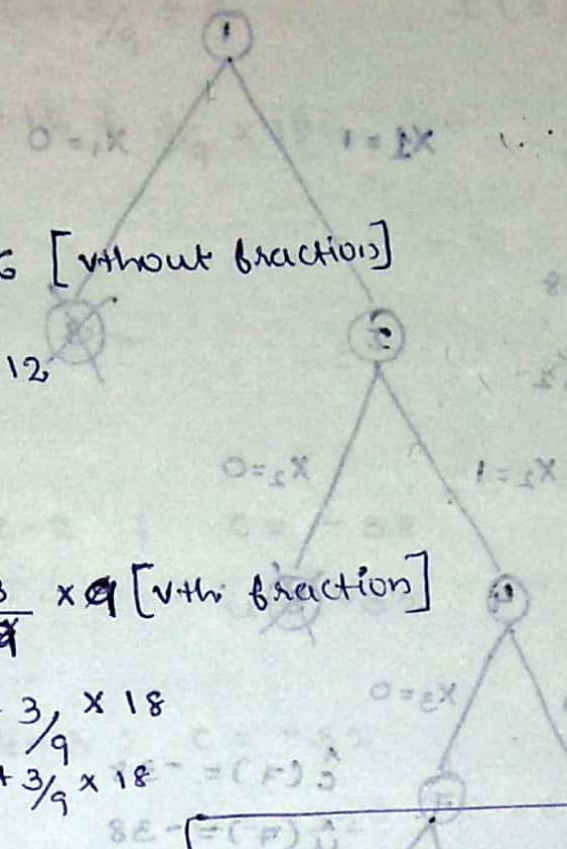
4. if $U(x)$ is minimum than upper then upper will set to $U(x)$.
5. if $\hat{c}(x) >$ upper kill node x.
6. Otherwise the mincost $\hat{c}(x)$ becomes E-node and generate children for E-node.
7. Repeat steps ② to ⑥ until all the nodes get covered.
8. Minimum cost $\hat{c}(x)$ become the answer node.

* Calculate 0/1 Knapsack for the following problem using branch and bound.

O_i	1	2	3	4
P_i	10	10	12	18
W_i	2	4	6	9

$M = 15$

Sol: upper = ∞



① $\hat{c}(1) = -38$

$u(\hat{x}) = -32$

node - 1

$u(\overset{1}{x}) = 2 + 4 + 6$ [without fraction]

$= 10 + 10 + 12$

$= -32$

$c(\overset{1}{x}) = 2 + 4 + 6 + \frac{3}{9} \times 9$ [with fraction]

$= 2 + 4 + 6 + 3/9 \times 18$

$= 10 + 10 + 12 + 3/9 \times 18$

$= -38$

conditions

(i) check condition 1.

∞ , -32

min = -32

∴ upper = -32

(ii) $-38 > -32$, false.

so, no node is killed.

(i) upper = ∞, check u

(ii) compare $\hat{c}(x)$ with updated upper.

if $\hat{c}(x) >$ upper, kill

(iii) select least cost of $\hat{c}(x)$

No need of 3rd condition checking bcoz of a single node.

node - 2

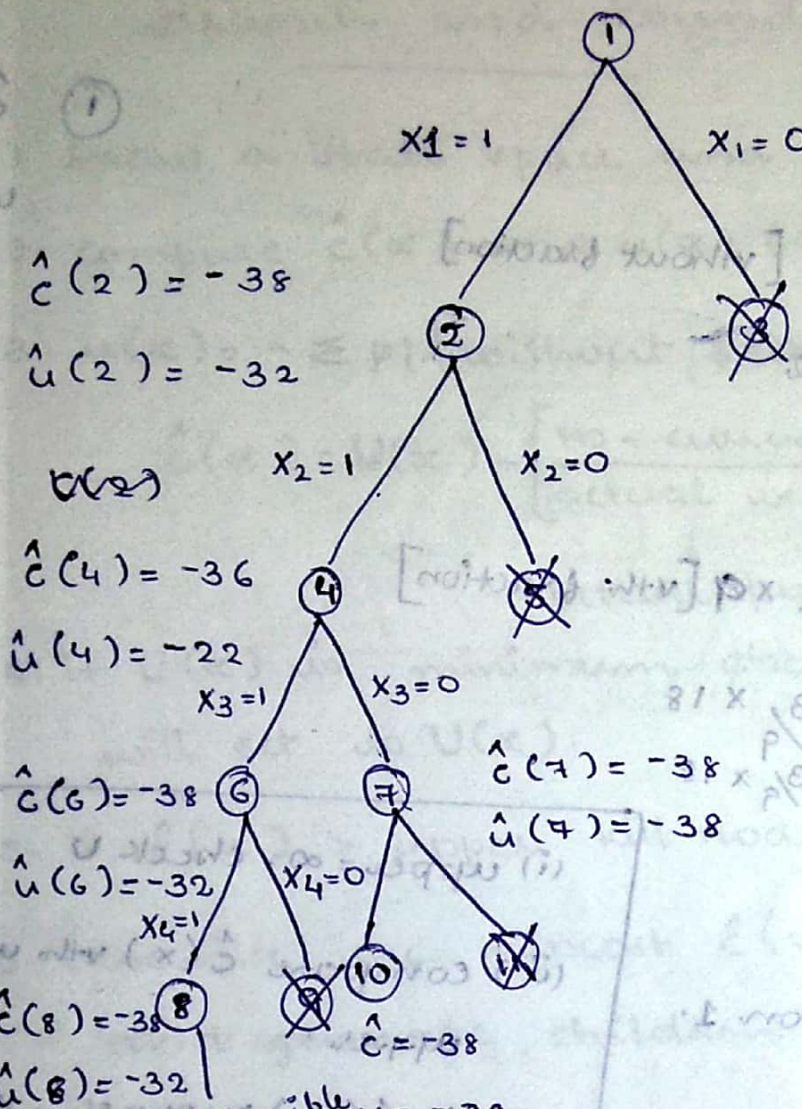
$u(2) = 2 + 4 + 6$

$= 10 + 10 + 12 = -32$

node - 3

$u(3) = 4 + 6$

$= 10 + 12 = -22$



$\hat{c}(1) = -38$

$\hat{u}(1) = -32$

$\hat{c}(2) = -38$

$\hat{u}(2) = -32$

$\hat{c}(3) = -32$

$\hat{u}(2) = -22$

$\hat{c}(4) = -36$

$\hat{u}(4) = -22$

$\hat{c}(5) = -36$

$\hat{u}(5) = -22$

$\hat{c}(6) = -38$

$\hat{u}(6) = -32$

$\hat{c}(7) = -38$

$\hat{u}(7) = -38$

$\hat{c} = -38$

$\hat{c}(8) = -38$

$\hat{u}(8) = -32$

$U = -38$

path = 1101

profit = 2 + 4 + 9 = 15

BFS : select among the live subproblems one of those with the lowest level.

DFS : select among the live subproblems one of those with the largest level.

$c(2) = 2+4+6+\frac{3}{9} \times 9$

$= 10+10+12+\frac{3}{9} \times 18$

$= -38$

$c(3) = 4+6+\frac{5}{9} \times 9$

$= 10+12+\frac{5}{9} \times 18$

$= 10+12+5 \times 2$

$= -32$

conditions

node-2 : $c = -38$

$u = -32$

node-3 : $c = -32$

$u = -22$

upper is checked with both node-2 and 3's u.

-32, node 2 u => -32, -32, node 3 u

-32, -32, -22

c => -38 > -32, -32 = -32, false.

so will -38 is least, so node 3's child.

node-4

$u(4) = 2+4+6$

$= 10+10+12 = -32$

$c(4) = 2+4+6+\frac{3}{9} \times 9$

$= 10+10+12+\frac{3}{9} \times 18 = -38$

node-5

$u(5) = 2+6$

$= 10+12 = -22$

$c(5) = 2+6+\frac{7}{9} \times 9$

$= 10+12+\frac{7}{9} \times 18$

$= -36$

node 4: $c = -32$

$u = -38$

node 5: $c = -22$

$u = -36$.

upper = -32,

$-32, u(4), u(5) \Rightarrow -32, -32, -22$

no change to upper.

$c \Rightarrow -38 \not> -32$ false

$-36 \not> -32$ false

no kill

least -38, node-4

node-6

$u(6) = 2+4+6$

$= 10+10+12$

$= -32$

$c(6) = 2+4+6+\frac{3}{9} \times 9$

$= 10+10+12+\frac{3}{9} \times 18$

$= -38$

node-7

$u(7) = 2+4+9$

$= 10+10+18$

$= -38$

$c(7) = 2+4+9$

$= 10+10+18$

$= -38$

node-6

$u(6) = -32$

$c = -38$

node-7: $u = -38$

$c = -38$

$(-32, -32, -38)$

upper $= -38$.

$-38 > -38$

$-38 > -38$

least, all c's same, so take 1st one.

node-8

$u(8) = 2+4+6$

$= 10+10+12 = -32$

$c(8) = 2+4+6+3/9 \times 9$

$= 10+10+12+3/9 \times 18$

$= -38$

$-38, -32, -32$

$u = -38$

node-9

$u(9) = 2+4+6$

$= 10+10+12 = -32$

$c(9) = 2+4+6$

$= 10+10+12 = -32$

$c \Rightarrow -38 \neq -38$

$\Rightarrow -32 > -38$, T

Kill node 9

node - 10

$u(10) = 2 + 4 + 9$

$= 10 + 10 + 18 = -38$

$c(10) = 2 + 4 + 9$

$= 10 + 10 + 18 = -38$

node - 11

$u(11) = 2 + 4$

$= 10 + 10 = -20$

$c(11) = 10 + 10 = -20$

upper $= -38$

$-38, -38, -20$

$\therefore$ upper $= -38$

$-38 > -38$, $-20 > -38$

Different types of nodes

Live node:

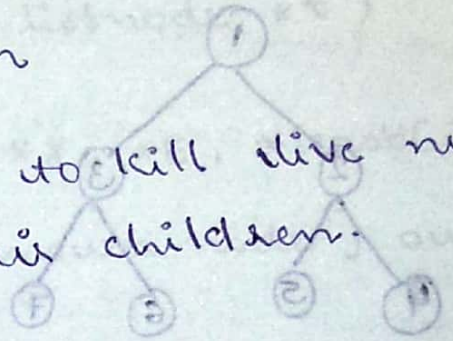
A node which has been generated and all of whose children have not being generated

E-node: The live node whose childsen are currently being generated is called a E node: (Node being Expanded).

Dead node: A dead node is a generated node which is not to be expanded further.

Bounding Function

BF are used to kill live node vthout generating all their childsen.



Depth-First node Generation with bounding function is called back tracking.

Branch and bound is a method of solving optimization problems. The objective of B & B is minimization. It also generate a state space tree. In B & B, there are two bounds upper (U). bound and lower (L) bound. Fractions are allowed in lower bound. Fractions are not allowed in upper bound for knapsack problems.

3 types of B & B.

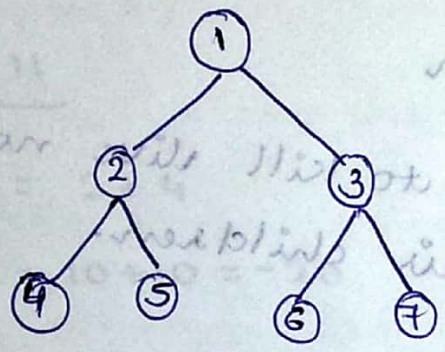
(i) FIFO B & B

(ii) LIFO B & B

(iii) LC B & B

FIFO B & B

It follows the method of First In First Out and expanded as BFS.



LIFO B & B

It follows the method of Last-In-First-Out and it is expanded from LIFO.

LC B & B

Node expanded according to the least cost or minimum weight.

* Draw the portion of state space tree generated by LC BB / BF for the following knapsack instance.

$n = 5$

$P_1, P_2, P_3, P_4, P_5 = \{10, 15, 6, 8, 4\}$

$w_1, w_2, w_3, w_4, w_5 = \{4, 6, 3, 4, 2\}$

Knapsack capacity => $M = 12$.

Control Abstraction of LC B&B

```
LCsearch (struct listnode *t)
{
  struct listnode *X, *E, *least();
  if (*t is an answer node) output *t and return;
  E = t;
  initialize the list of live node to be empty;
  do {
    for (each child X of E) {
      if (X is an answer node) output the path from X to t and return;
      Add(X);
      X -> parent = E;
    }
    if (there are no more live nodes) {
      cout << "No answer node\n";
      return;
    }
```

E= least ();

}while (1); }

Let T be state space Tree. C() be the cost function for the nodes in T. If x is a node in T, then C(T) be minimum cost of any answer node in T. Algorithm LC Search have two function least() and Add().

Least() function delete a live node. Add() function add a live node from or to the list of live node. In LC search algorithm variable E points to current E-node. By definition, current E-node be root node. So, E = T.

Then, initialize the list of live node, examine each child in List(E). If one of the children is answer node, the algorithm output the path from X→T and terminates. If a child of E is not an answer node, it becomes a live node and it is added to the list of live nodes and its parent field set to E.

when all children of E have been generated, E becomes dead node. This happens

only if none of E's children is an answer node.

If there is no live node, then entire state space tree has been searched and no answer node found and algorithm terminates. Otherwise, least function currently chooses next E-node and search continues.

Similarity btw LC (Breadth First search), LIFOBB, FIFOBB

If the list of live node is implemented as a queue with least function and add function. Then, LC search will be transformed to FIFO BFS / FIFO BB.

If the list of live node is implemented as a stack with the least function and add(), then LC is transformed to LIFO BFS / LIFO BB.

So, the control abstraction of LC BFS and DFS are same. Only difference is implementation of list of live node or selection rule used to obtain the next E-node.

Bounding Function

A branch and bound method searches a state space tree using any search mechanism in which all the children of the E-node are generated before another node becomes the E-node.

A cost function $\hat{c}(x)$ used to provide lower bound for any node x. For 'U' be upper bound. If a live node x with $\hat{c}(x) > U$, then the live node x may be killed. The starting value of upper can be obtained by heuristic value or can be set to infinity. Each time a new answer node is found, the value of upper can be updated with minimum.

Control Abstraction of Knapsack problem using Branch & Bound

```
float UBound (float cp, float cw, int k, float m)
// cp, cw, k and m have the same meaning
   as in pgm 7.11. w[i] and p[i] are
```

```
// respectively,
// The weight and profit of the ith object
{
    float b = cp, c = cw;
    for (int i = k+1; i <= n; i++) {
        if (c + w[i] <= m) {
            c += w[i]; b -= p[i];
        }
    }
    return (b);
}
```

Problem - solution

n = 5

M = 12

o_i	1	2	3	4	5
p_i	10	15	6	8	4
w_i	4	6	3	4	2

node-1

$v(x) = 4 + 6$

$= 10 + 15 = -25$

$\hat{c}(x) = 4 + 6 + 2/3 \times 3$

$= 10 + 15 + 2/3 \times 6$

$= -29$

upper = -25

node-2

$v(x) = 4 + 6$

$= 10 + 15 = -25$

$\hat{c}(x) = 6 + 3 + 2/3 \times 3$

$= 10 + 15 + 2/3 \times 6$

$= -29$

node-3

$v(x) = 6 + 3$

$= 15 + 6 = -21$

$\hat{c}(x) = 6 + 3 + 3/4 \times 8$

$15 + 6 + 3/4 \times 8^2$

$= 6 + 3 + 6$

$= -27$

node-4

$v(x) = 4 + 6$

$= 10 + 15 = -25$

$\hat{c}(x) = 4 + 6 + 2/3 \times 3$

$= 10 + 15 + 2/3 \times 6$

$= -29$

conditions

$u = -25$

$-25, -25, -21$

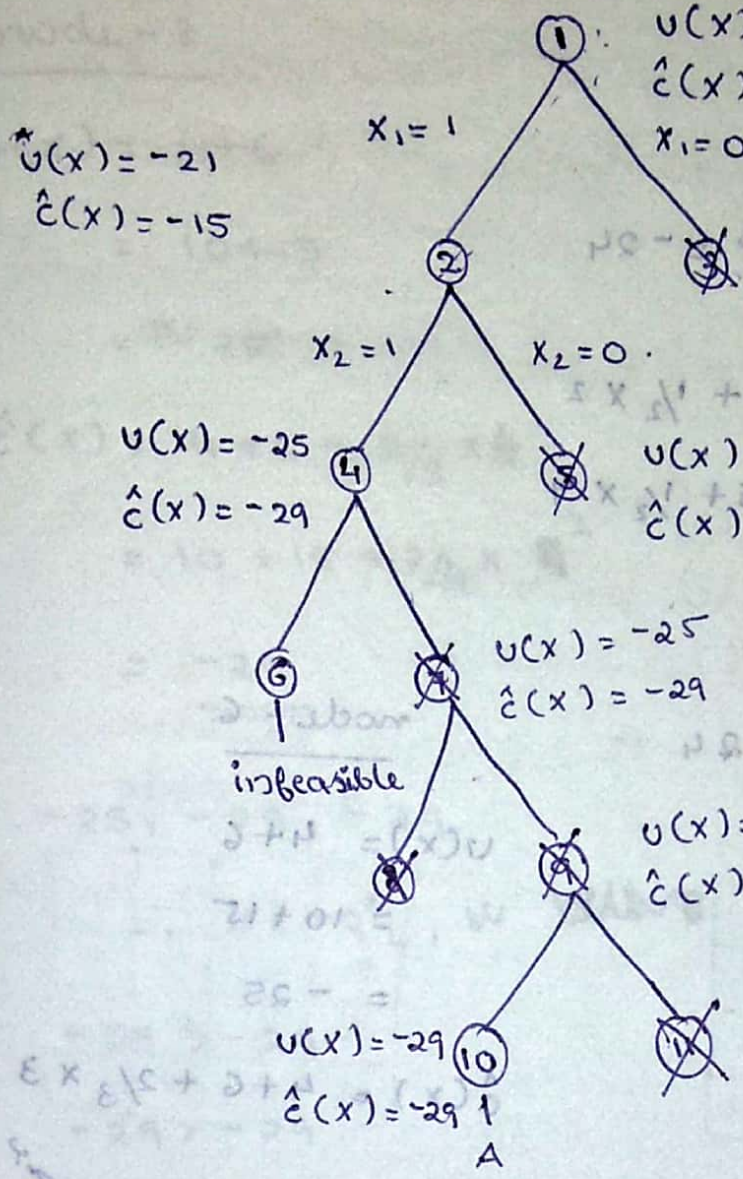
$\therefore v = -29$

(i) $-29 \nmid -25$

(ii) $-27 \nmid -25$

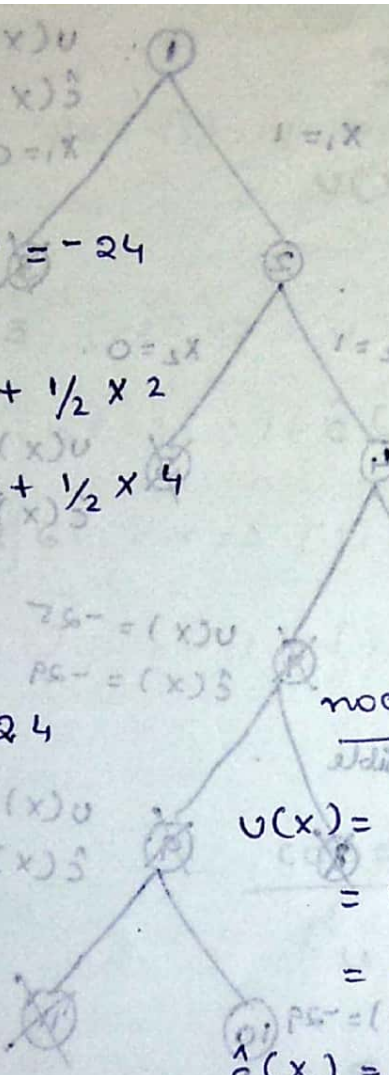
no kill.

(iii) least ②



Ans: 11001

{1, 2, 4, 7, 9, 10}



node - 5

$U(x) = 4+3+4$

$= 10+6+8 = -24$

$\hat{c}(x) = 4+3+4+\frac{1}{2} \times 2$

$= 10+6+8+\frac{1}{2} \times 4$

$= -26$

$\therefore -25, -25, -24$

$\therefore U = -25$

$-29 \not> -25$

$-26 \not> -25$

no kill

least - (4)

node - 6

$U(x) = 4+6$

$= 10+15$

$= -25$

$\hat{c}(x) = 4+6+\frac{2}{3} \times 3$

$= 10+15+\frac{2}{3} \times 6^{2}$

$= -29$

node - 7

$U(x) = 4+6$

$= 10+15$

$= -25$

$\hat{c}(x) = 4+6+\frac{2}{4} \times 4$

$= 10+15+\frac{2}{4} \times 8^{2}$

$= -29$

conclition

$-25, -25, -25$

$\therefore U = -25$

$-29 \not> -25$

$-29 \not> -25$

$\therefore$ no Kill

node - 8

$v(x) = 4 + 6$

$= 10 + 15$

$= 25$

$\hat{c}(x) = 4 + 6 + 2/5 \times 4$

$= 10 + 15 + 2/5 \times 8^2$

$= -29$

$-25, -25, -29.$

$\therefore v = -29,$

$-29 > -29$

$-29 > -29.$

node - 9

$v(x) = 4 + 6$

$= 10 + 15$

$= -25$

$\hat{c}(x) = 10 + 15 + 4$

$= -29$

node - 10

$v(x) = 4 + 6 + 2$

$= 10 + 15 + 4$

$= -29$

$\hat{c}(x) = 10 + 15 + 4$

$= -29$

$-29, -29, -25, \therefore v = -29.$

$-29 > -29$

$-25 > -29$

$\therefore$ kill 11

node - 11

$v(x) = 4 + 6$

$= 10 + 15$

$= -25$

$\hat{c}(x) = -25$

04/11/19

Draw the portion of state space tree generated by LCBB for the following knapsack instance

$n = 5$

$p_i = \{4, 4, 5, 8, 9\} = w_i$ and $M = 16$

$n^2 - 1$ (puzzle problem)

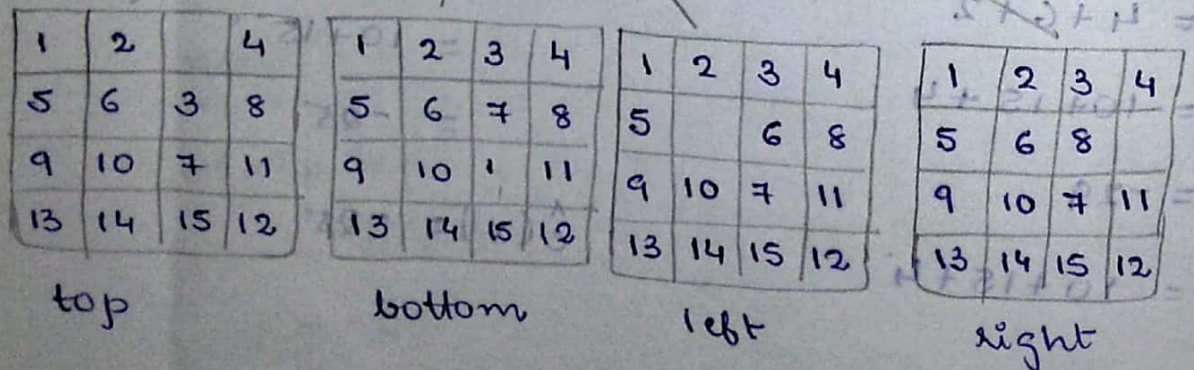
$n = 4 \Rightarrow 4^2 - 1 = 15$ puzzle problem.

Current state

1	2	3	4
5	6		8
9	10	7	11
13	14	15	12

goal state

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	



top: $\hat{c}(x) = 1 + 1 + 1 + 1 + 1$

$= 5$

bottom: $\hat{c}(x) = 1 + 1 + 1$

$= 3$

$\hat{f}(x)$: Length of the path from root node to x.

$\hat{g}(x)$: Length of the shortest path from root node to x.

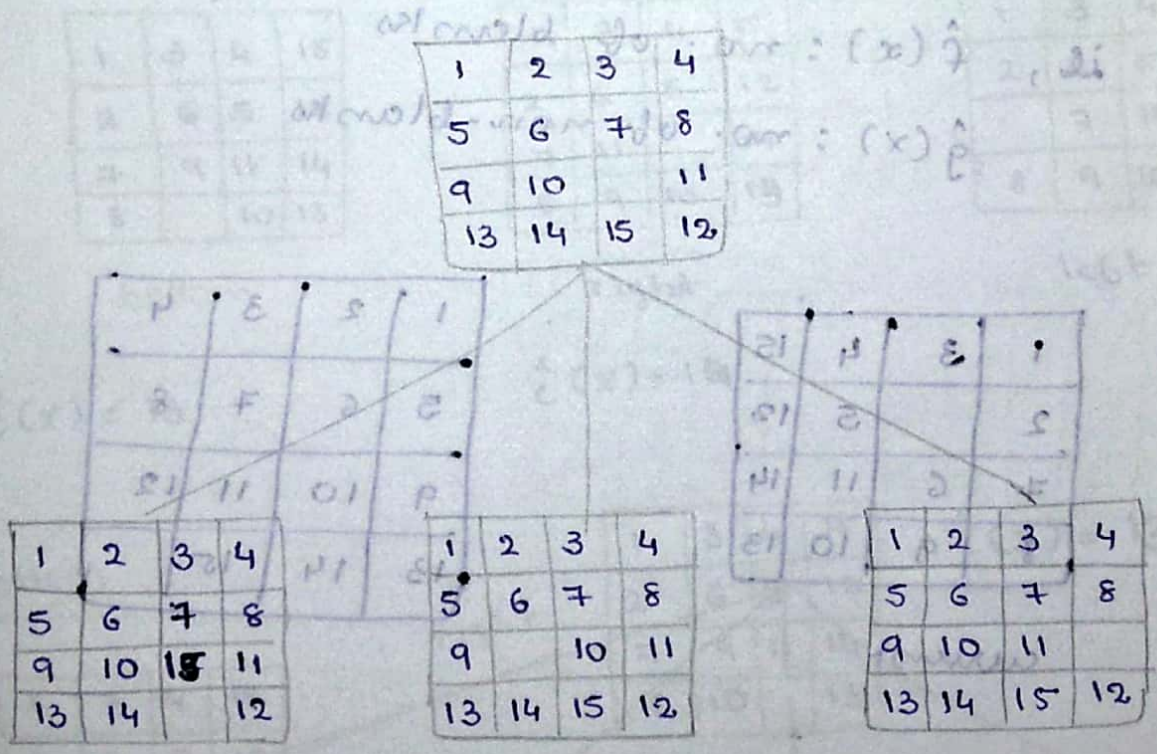
$$\hat{c}(x) = \hat{f}(x) + \hat{g}(x)$$

left : $\hat{c}(x) = 1 + 1 + 1 + 1 + 1$

$= 5$

Right : $\hat{c}(x) = 1 + 1 + 1 + 1 + 1$

$= 5$



bottom

left

right

$\hat{c}(x) = 1 + 1 + 1 + 1$

$= 4$

$\hat{c}(x) = 1 + 1 + 1 + 1$

$= 4$

$\hat{c}(x) = 1 + 1$

$= 2$

1	2	3	4
5	6	7	8
9	10	11	
13	14	15	12

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

bottom

KB pages. 401, 402, 403, 404

$\hat{g}(x)$ = no. of non-blank tiles not in their goal position.

ie, $\hat{f}(x)$: no. of blanks

$\hat{g}(x)$: no. of non-blanks

*

1	3	4	15
2		5	12
7	6	11	14
8	9	10	13

current

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

1		4	15
2	3	5	12
7	6	11	14
8	9	10	13

top

1	3	4	15
2	6	5	12
7		11	14
8	9	10	13

1	3	4	15
	2	5	12
7	6	11	14
8	9	10	13

1	3	4	15
2	5		12
7	6	11	14
8	9	10	13

top : $\hat{c}(x) = 1 + 1 + 1 + 14$

$= 15$

bottom : $\hat{c}(x) = 13$

left : $\hat{c}(x) = 14$

right : $\hat{c}(x) = 14$

1	3	4	15
2	6	5	12
7		11	14
8	9	10	13

1	3	4	15
2	6	5	12
7	9	11	14
8		10	13

bottom

$\hat{c}(x) = 13$

1	3	4	15
2	6	5	12
7	11		14
8	9	10	13

right

$\hat{c}(x) = 14$

1	3	4	15
2	6	5	12
·	7	11	14
8	9	10	13

left

bottom

1	3	4	15
2	6	5	12
7	9	11	14
8		10	13

right

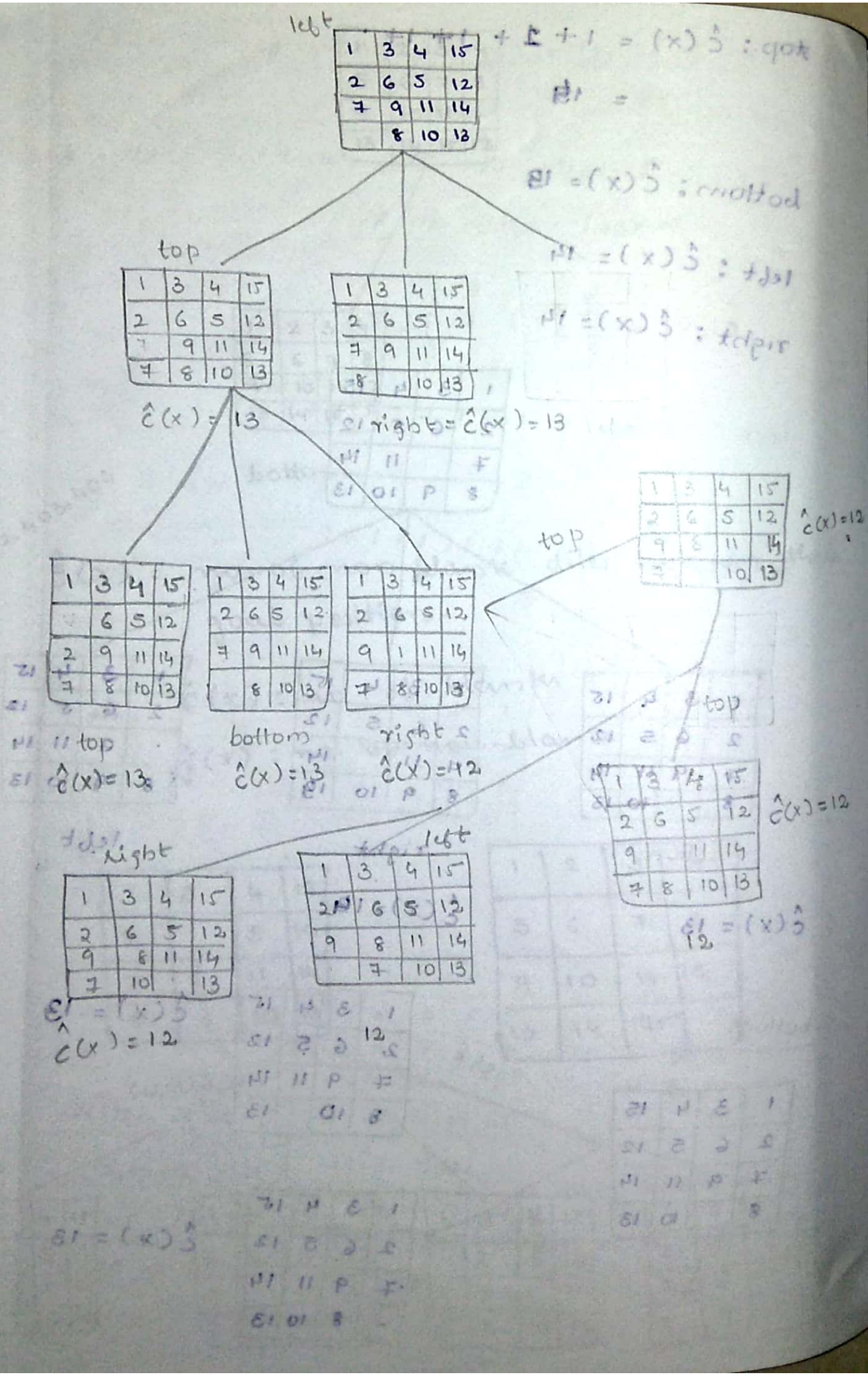
1	3	4	15
2	6	5	12
7	9	11	14
8	10		13

$\hat{c}(x) = 13$

left

1	3	4	15
2	6	5	12
7	9	11	14
-	8	10	13

$\hat{c}(x) = 13$


left
top
$\hat{c}(x) = 13$
right = $\hat{c}(x) = 13$
top
$\hat{c}(x) = 12$
top
$\hat{c}(x) = 13$
bottom
$\hat{c}(x) = 13$
right
$\hat{c}(x) = 12$
right
$\hat{c}(x) = 12$
left
top
$\hat{c}(x) = 12$
$= (x)\hat{c}$

I * $U(x) = 4+4+5 = 13$ $\quad U = \infty$

$\hat{c}(x) = 4+4+5+\frac{3}{8} \times 8 = 16$

$U = 13$

node-2

$U(x) = 4+4+5 = -13$

$\hat{c}(x) = -16$

$-13, -13, -9 \Rightarrow U = -13$

$-16 > -13$ - F with 2

$-16 > -13$ - F

node-3

$U(x) = 4+5 = -9$

$\hat{c}(x) = 4+5+\frac{3}{8} \times 8 = 16$

node-4

$U(x) = 4+4+5 = -13$

$\hat{c}(x) = 4+4+5+\frac{3}{8} \times 8 = -16$

$-13, -9, -13$

$U = -13$

$-16 > -13$ F , $-16 > -13$

node-5

$U(x) = 4+5 = -9$

$\hat{c}(x) = -16$

node-6

$U(x) = 4+5 = -9$

$\hat{c}(x) = -16$

$-13, -9, -13$

$U = -13$

node-7

$U(x) = 5+8 = -13$

$\hat{c}(x) = 5+8+\frac{3}{9} \times 9 = -16$

$\Rightarrow -16 > -13$

$-16 > -13$

node - 8

$v(x) = 4+4+5 = -13$

$\hat{c}(x) = -16$

$-13, -13, -16$

$v = -16$

$-16 > -16$

$-16 > -16$

node - 9

$v(x) = 4+4 = -16$

$\hat{c}(x) = 4+4+8 = -16$

node - 10

$v(x) = 4+5 = -9$

$\hat{c}(x) = 4+5+\frac{7}{8} \times 8 = -16$

$-16, -9, -12$

$-16 > -16$

$-16 > -16$

node - 11

$v(x) = 4+8 = -12$

$\hat{c}(x) = 4+8+\frac{4}{9} \times 9 = -16$

node - 12

$v(x) = 4+5 = -9$

$\hat{c}(x) = -16$

$-16, -9, -12$

$v = -16$

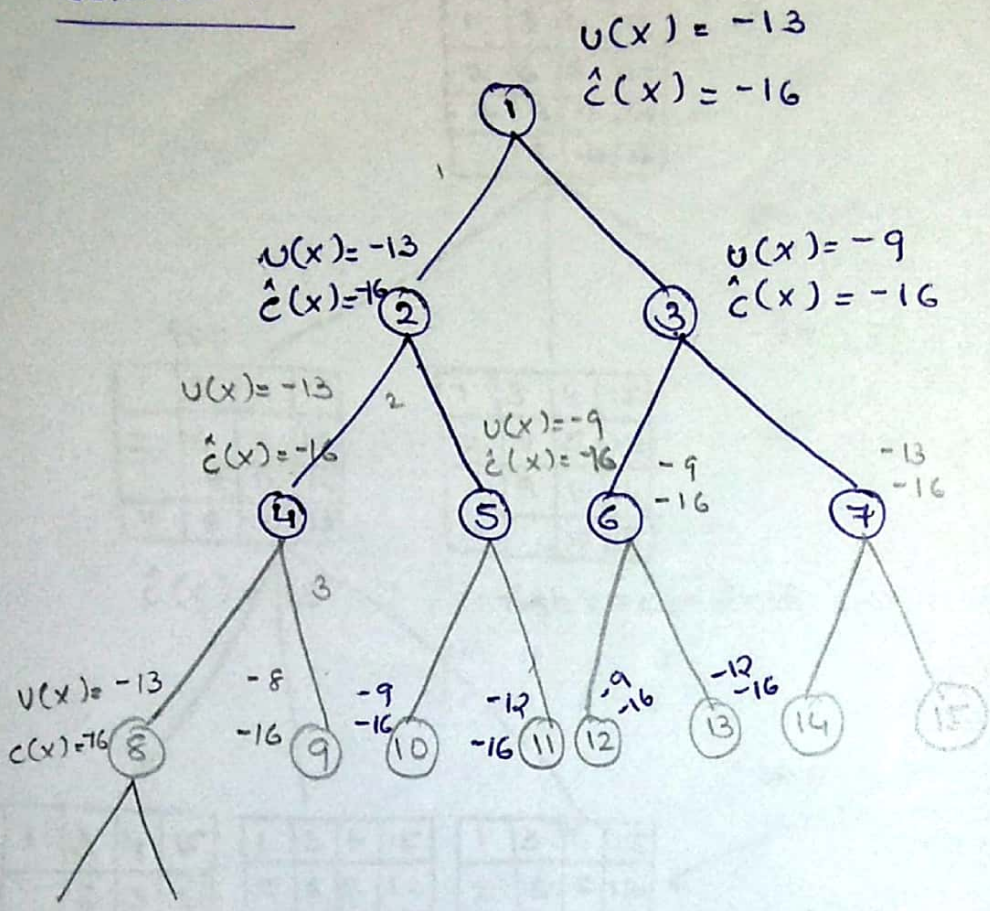
$-16 > -16, -16 > -16$

node - 13

$v(x) = 4+8 = -12$

$\hat{c}(x) = 4+8+\frac{4}{9} \times 9 = -16$

solution



The $N^2 - 1$ puzzle problems

- Invented by Sam Loyd.

We are given an initial arrangement of the tiles, and objective is to transforms this arrangement into the goal arrangement. Each move creates a new arrangement of tiles these are arrangements are called the state of the puzzle.

A state is reachable from the initial state iff there is a sequence of legal moves from the initial state to this state (goal state).

$$\hat{c}(x) = f(x) + \hat{g}(x)$$

$f(x)$ = is the length of the path from the root to node x.

$\hat{g}(x)$ = no. of non-blank tiles not in this goal position.