0/1 Knapsack Problem

(using backtracking)

When n elements are given which have w_i as the weights associated with them, & p_i as the profits associated with them.

It is required to fill the Knapsack of capacity m such that:-

$$\sum_{1 \le i \le n} w_i x_i \le m \quad \text{and} \quad \sum_{1 \le i \le n} p_i x_i \text{ is maximized}$$

and x_i = is either 0 or 1.

The solution space for this problem consists of 2^n distinct ways to assign 0 or 1 values to the x_i's.

The backtracking technique to solve this problem involves a recursive function:-

```
void BKnap (int k, float cp, float cw)
```

where, k is the index of the last element removed item
cp is the current profit total
cw is the current weight total

BKnap function uses another function which is :-

float Bound (float cp, float cw, int k)

Function Bound (cp, cw, k) determines an upperbound on the best solution obtainable by expanding any node Z at level k+1 of the state space tree.

If at node Z, the values of x_i, $(1 \le i \le k)$ have already been determined, then an upper bound for Z can be obtained by relaxing the requirement $x_i = 0$ or 1 to $0 \le x_i \le 1$ for $(k+1) \le i \le n$.

(means :- rather than values 0 or 1, value between 0 and 1 is allowed i.e. fractional value for the items from (k+1) to n, it is just to compute upper bound for the given situation.)

After relaxing the requirement, the greedy algorithm is used to solve the relaxed problem.

It is assumed that :-

$$\frac{p_i}{w_i} \geq \frac{p_{i+1}}{w_{i+1}}$$

(i.e. decreasing order of $\frac{p}{w}$ of items is considered).

Date
Page

```
float Bound (float cp, float cw, int k)
{
    // cp is current profit total
    // cw is current weight total
    // k is the index of the last removed item.
    // m is the knapsack size.

    float b = cp;
    float c = cw;

    for (int i = k+1; i <= n; i++)
    {
        c = c + w[i];          // weight if we add element in knapsack

        if (c < m)             // if feasible
        { b = b + p[i]; }      // also consider profit associated
        else                   // if not feasible
        {
            return (b + (1 - (c-m)/w[i]) * p[i]);
        }
    }
    return (b);
}
```

(1) consider the fraction of that element

(2) then multiply by profit

(3) then add it to the total profit gained.

true → { b = b + p[i]; }

false → else

Note: $\left[1 - \frac{(c-m)}{w_i}\right] = \frac{U}{w_i}$ as was in Knapsack algo. using greedy method.

CAMPUS 2 Date Page

The recursive backtracking algorithm for solving Knapsack Problem considers the following as values:-

m → size of the Knapsack
n → no. of elements (i.e. number of weights & profits)
w[] → weights
p[] → profits

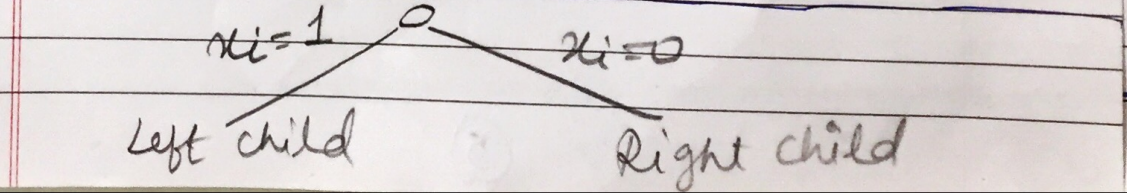
Decreasing order of :- P/w as follows:-

$$\frac{P_i}{w_i} >= \frac{P_{i+1}}{w_{i+1}}$$

fw → final weight of Knapsack
fp → final maximum profit
fp is initially set as -1. (i.e. fp = -1)

x[k] = = 0 , if w[k] is not in the Knapsack

x[k] = = 1 , if w[k] is in the Knapsack.



Recursive BACKTRACKING ALGO. for Knapsack problem.

cp → current profit total
cw → current weight total

INVOKED AS :- BKnap(1, 0, 0)

```
Void BKnap(int k, float cp, float cw)
{
  // Generate left child
  if(cw + w[k] <= m)          // if feasible?
  {
    y[k] = 1;                 // include the element
    if (k < n)
    {
      BKnap(k+1, cp+p[k], cw+w[k]);
    }
    if ((cp+p[k] > fp) && (k == n))
    {
      fp = cp + p[k];
      fw = cw + w[k];
      for (int j=1; j<=k; j++)
      {
        x[j] = y[j];
      }
    }
  }
}
```

// Generate right child

```
if (Bound (cp, cw, k) >= fp)          // if not feasible
{                                     // compute upper Bound using greedy method
    y[k]=0;  // do not include element
    if (k<n)
        {
            BKnap (k+1, cp, cw);      // recursive call for other elements
        }
    if ((cp > fp) && (k==n))
    {   fp = cp;
        fw = cw;
        for (int j=1; j<=k; j++)
        {   x[j] = y[j]
        }
    }
}
}
```

CAMPUS 2 Date _____ Page _____

Note:- The bounding function need not be used whenever the backtracking algorithm makes a move to the left child of a node, since the bound for a feasible left child of a node x is the same as that for x.

* Steps to solve knapsack problem using backtracking:-

- Arrange items in decreasing order of $\frac{p_i}{w_i}$. (i.e. $\frac{p_i}{w_i} \geq \frac{p_{i+1}}{w_{i+1}}$) (i.e. non-increasing order)
- Find Upper Bound Value.

The upper bound is a value which can be generated by applying greedy algorithm and relaxing the requirement as $0 \leq x_i \leq 1$. i.e. allowing fractions.

- Generate State Space tree
- Follow Depth-first-generation while expanding state space tree.

Example of a 0/1 Knapsack problem being solved by using backtracking algorithm design technique :-

Item	x_1	x_2	x_3	x_4
p	45	30	45	10
w	3	5	9	5
p/w	15	6	5	2

Capacity of Knapsack :- $m = 16$

Sol :- Step 1 :-
The items are arranged in the non-increasing order of $\frac{p_i}{w_i}$ i.e. in decreasing order of $\frac{p_i}{w_i}$.

So, $\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \frac{p_3}{w_3} \geq \frac{p_4}{w_4}$

for all the items :-

x_1, x_2, x_3 and x_4

Then, we will generate state space tree and will compute upper bound when required by using greedy algorithm and relaxing the requirement as $0 \leq x_i \leq 1$

(Note:- UB is upper bound)

(Initial state)

$cp = 0$, $cw = 0$
$UB = 115$

Root Node

No item is selected yet.

So, cp = current profit total = 0

cw = current weight total = 0

UB calculation is done by using greedy approach as follows:—

* Selecting first object x_1

$p_1 = 45$, $w_1 = 3$ $\quad m = 16$.

$3 < 16$, so selected

Remaining weight of Knapsack is

$16 - 3 = 13$

* Selecting second item x_2

$p_2 = 30$, $w_2 = 5 < B$ ✓ selected

Remaining weight of Knapsack

$13 - 5 = 8$

* Selecting third element x_3

$p_3 = 45$, $w_3 = 9 < 8$ is false

So, fraction can be selected as follows:-

$$x_3 = \frac{\text{Remaining weight of Knapsack}}{\text{Actual weight of item}}$$

$$x_3 = \frac{8}{9}$$

So, profit earned will also be $p_i x_i = 45 \times \frac{8}{9}$

∴ Upper bound at Root Node is computed as follows using greedy approach:-

UB Calculation	x_1	x_2	x_3
p	45	30	$45 * \frac{8}{9} = 40$
w	3	5	8 out of 9 i.e. $\frac{8}{9}$
		wt =	$3 + 5 + (\frac{8}{9} \times 9) = 16 = m$

Means, if greedy approach is followed and fractions are also allowed, then the expected maximum profit i.e. Upper bound at root node is:-

UB = 45 + 30 + 40 = 115

Root Node

cp=0, cw=0
UB=115

$x_1 = 1$ (selecting item x_1)

cp=45, cw=3
UB=115

UB is same in Left child

$x_1 = 0$ (not selecting item x_1)

cp=0, cw=0
UB=79

UB is changed because we are not selecting item x_1. So, greedy approach is applied again to compute upper bound but starting selecting the items from x_2 onwards:-

not selecting x_1

So, upper bound for profit is

UB = 30+45+4
= 79

UB	x_1	x_2	x_3	x_4
P	X	30	45	$10 * \frac{2}{5} = 4$
W	X	5	9	2 out of 5 i.e. $\frac{2}{5}$

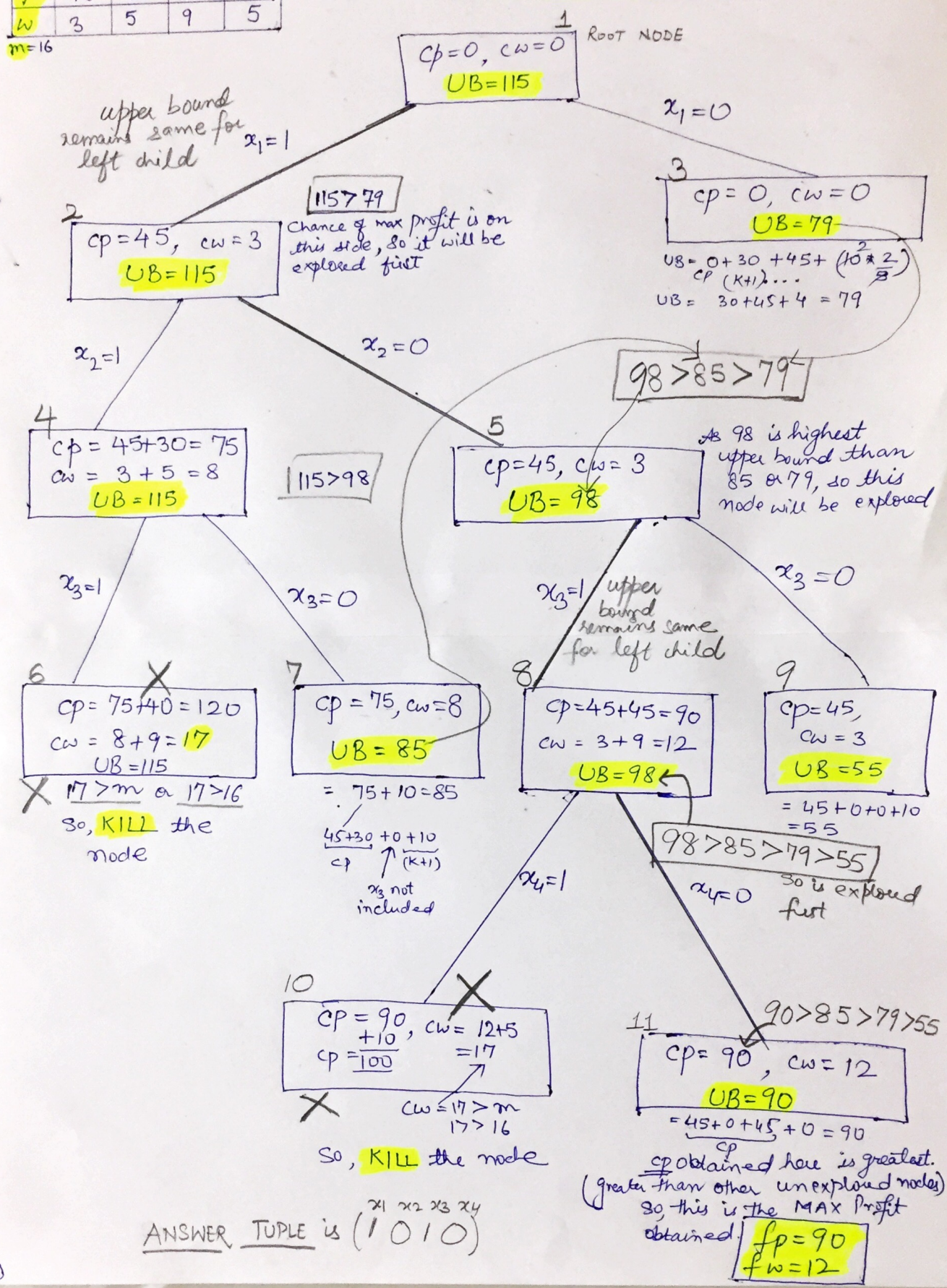
$W = 5+9+\left(\frac{2}{5} * 5\right) = 16 = m$

5+9=14 remaining wt = 16−14=2

0/1 Knapsack Problem being solved by backtracking technique:-

	x_1	x_2	x_3	x_4	Given values
p	45	30	45	10	
w	3	5	9	5	

$m = 16$



ANSWER TUPLE is $(x_1\ x_2\ x_3\ x_4) = (1\ 0\ 1\ 0)$