CS575 Homework 4

Due at the Beginning of the Class on May 5

PLEASE PRINT DOUBLE SIDED ONLY

Your Name: Aniruddha Tekade **Section:** 02

I promise to follow the academic honesty requirements of the Binghamton University. I agree that I will fill out and sign an official form that I have cheated if I get caught cheating. I understand that this form will be stored by the university. Furthermore, I understand that the minimum penalty for cheating is getting a grade of 0 for this assignment.

Sign: Aniruddha

1. [30%] You are given a 0-1 knapsack problem where the capacity of the knapsack W=30 and the set of items S ={(i1, 5,$50), (i2,20, $140), (i3,10,$60)} where each element in set S is a tuple of (itemID, weight, profit). Solve the given 0-1 knapsack problem using the dynamic programming method discussed in Chapter 12. Clearly show every step.

A1: Given Data: $W = 30$

$$S = \{(i_1, 5, \$50), (i_2, 20, \$140), (i_3, 10, \$60)\}$$

↳ Initially benefit matrix is all ∅. $B[0, w] = \emptyset$

↳ For $K = 1$:

Step I) $B[1,0] = 0$

$B[1,1] = 0$

$B[1,2] = 0$

$B[1,3] = 0$

$B[1,4] = 0$

$B[1,5] = 0$

⋮

$B[1,30] = 0$

Step II) For $K = 2$: Inserting i_1 into binary knapsack-

$B[2,0] = 0$

⋮

$B[2,4] = 0$

P.T.O →

$B[2,5] = \max\{ B[1,5], B[1,5-5] + b_2 \}$

$= \max\{ 0, 0+50 \}$

$\therefore \boxed{B[2,5] = 50}$

$B[2,6] = 50$

$B[2,7] = 50$

$\vdots$

$\boxed{B[2,30] = 50}$

Step III) $K=3$: Inserting item 2. (i_2)

$B[3,0] = 0$

$B[3,1] = 0$

$\vdots$

$B[3,5] = 50$

$\vdots$

$B[3,10] = 50$

$\vdots$

$B[3,19] = 50$

$B[3,20] = \max\{ B[2,20], B[2,0] + b_3 \}$

$= \max\{ 50, 0+140 \}$

$\boxed{B[3,20] = 140}$

$B[3,21] = \max\{ B[2,21], B[2,1] + b_3 \}$

$= \max\{ 50, 0+140 \} = 140$

$\vdots$

$B[3,22] = 140$

$\vdots$

$B[3,24] = 140.$

$B[3,25] = \max\{ B[2,25], B[2,5] + b_3 \}$

$= \max\{ 50, 50+140 \} = 190$

$\therefore \boxed{B[3,25] = 190}$

$B[3,30] = 190$

Step IV) K=4 : Inserting item 3 (i_3)

A1:

$B[4,0] = 0$

$B[4,10] = \max\{B[3,10], B[3,0]+b_4\}$
$= \max\{50, 0+60\} = 60$

$\therefore B[4,10] = 60$

$B[4,14] = 60$

$B[4,15] = \max\{B[3,15], B[3,5]+b_4\}$
$= \max\{50, 50+60\} = 110$

$B[4,15] = 110$

$B[4,16] = 110$

$B[4,20] = \max\{B[3,20], B[3,10]+b_4\}$
$= \max\{140, 50+60\}$

$B[4,20] = 140$

$B[4,25] = \max\{B[3,25], B[3,15]+b_4\}$
$= \max\{190, 50+60\}$

$B[4,25] = 190$

$B[4,26] = 190$
$B[4,27] = 190$
$B[4,28] = 190$
$B[4,29] = 190$
$B[4,30] = \max\{B[3,30], B[3,20]+b_4\}$
$= \max\{190, 140+60\} = 200$

$\therefore B[4,30] = 200 \longrightarrow$ maximum profit after adding all items.

B - Matrix ⇒

Weight	Max Profit $\{\}$	Max Profit $\{i_1\}$	Max Profit $\{i_1, i_2\}$	Max Profit $\{i_1, i_2, i_3\}$
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	(50)	50	50
6	0	50	50	50
7	0	50	50	50
8	0	50	50	50
9	0	50	50	50
10	0	50	50	(60)
11	0	50	50	60
12	0	50	50	60
13	0	50	50	60
14	0	50	50	60
15	0	50	50	(110)
16	0	50	50	110
17	0	50	50	110
18	0	50	50	110
19	0	50	50	110
20	0	50	(140)	140
21	0	50	140	140
22	0	50	140	140
23	0	50	140	140
24	0	50	140	140
25	0	50	(190)	190
26	0	50	190	190
27	0	50	190	190
28	0	50	190	190
29	0	50	190	190
30	(0)	(50)	(190)	(200)

← This is the table that represents all the previous calculations made during dynamic programming procedure for 0/1 Knapsack.

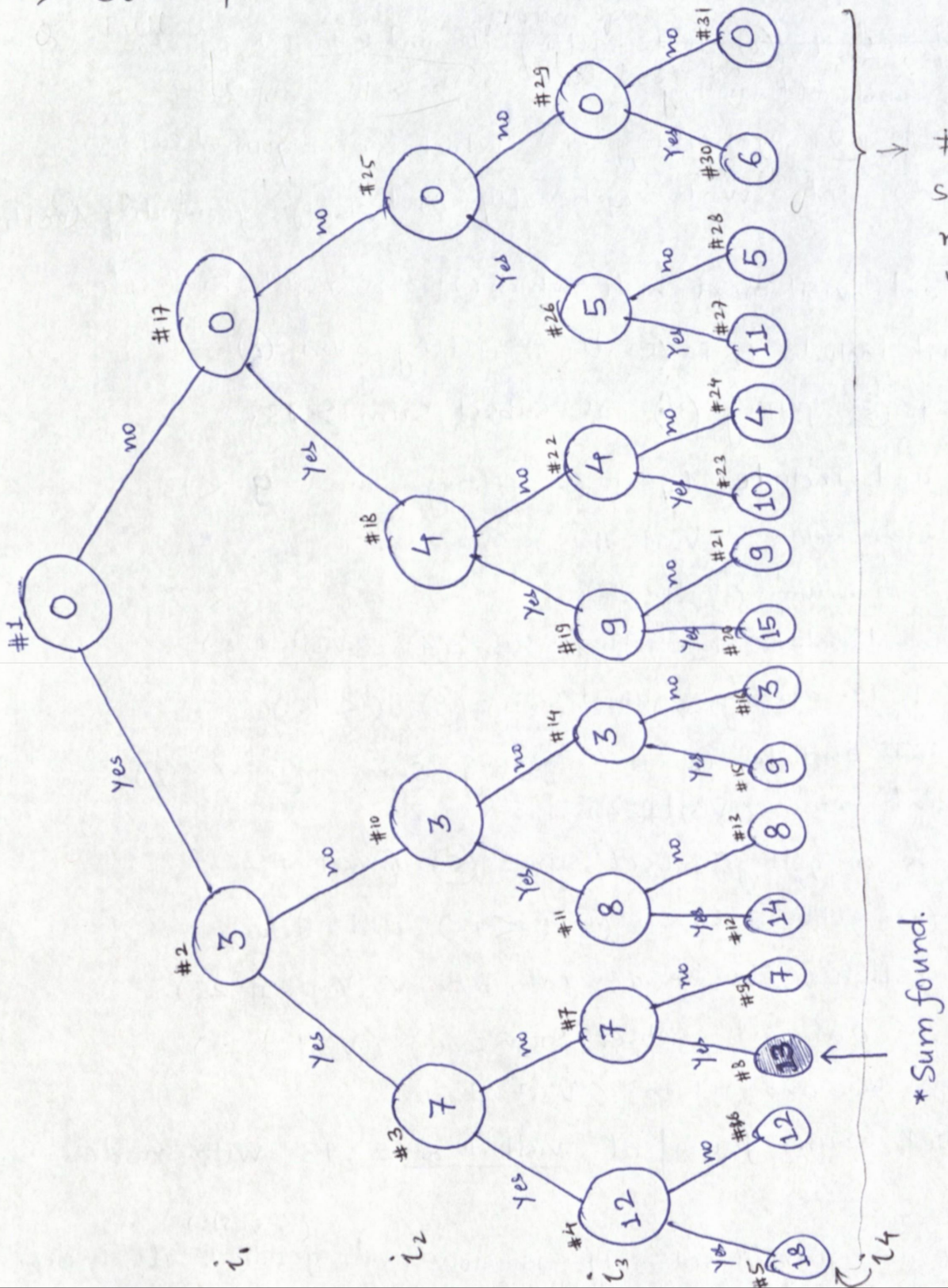
← The encircled values shows where calculation produced new values.

← The max. profit achieved after adding all the 3 given items into the knapsack is 200 at the location B[3, 30].

2. [40%] A set{3,4,5, 6} is given. For the set, find every subset that sums to S =13.

a. [10%] Solve the problem using the depth first method. Draw a state space tree and clearly show every step. Also, number the nodes in the sequence of visiting them.

↳ State Space tree built via DFS method for subset problem ⇒



→ #no. are the sequence in which nodes are visited while DFS.

P.T.O →

Steps:

(1) Initially subset Sum = ∅ (Root of tree)

(2) we add item i_1 (3) on the left child of root and branch it.

(3) we add item i_2 (4) to above left child generating sum = (7)

(4) Add i_3 (5) getting sum = (12) (< 13)

(5) Add i_4 (6) getting sum = (18) (> 13) and return.

(6) Skip i_3 and ~~add i_4 (6) to 12~~ getting (12) and Return to parent.

(7) Return to (7) and add nothing.

(8) Add i_4 (6) to (7) generating (13) which is required soln.
But DFS do not work optimally and keep generating / visiting nodes.

(9) Return to (7) and visit next node which (7) on right hand side.

(10) Exclude (3) and include i_3 ie. (5) to (3) getting sum = (8)

(11) Include i_3 to (8) getting (14) as subset Sum > 13.

(12) Exclude i_3 and include i_4 to (3) getting sum = 9 < 13.

(13) Exclude i_4 and Return. (Visit # 15)

(14) Exclude i_3 and Return (Visit # 16)

(15) Exclude i_1 and include i_2 getting sum = (4) (Visit # 18)

(16) Include i_3 to (4) getting subset Sum = (9) (< 13.)

(17) Exclude i_3 and include i_4 to (4) getting sum = (10) (<13)

(18) Exclude i_2 and return. (Visit #24)

(19) Include i_3 to 0 getting subset sum = (5) (Visit # 26)

(20) Add i_4 to (5) getting Sum = (11) (<13) and Return.

(21) Exclude i_4 return, Exclude i_3 and return (Visits # 28)

(22) Add i_4 to (0) and get subset Sum = (6) (Visit # 30)

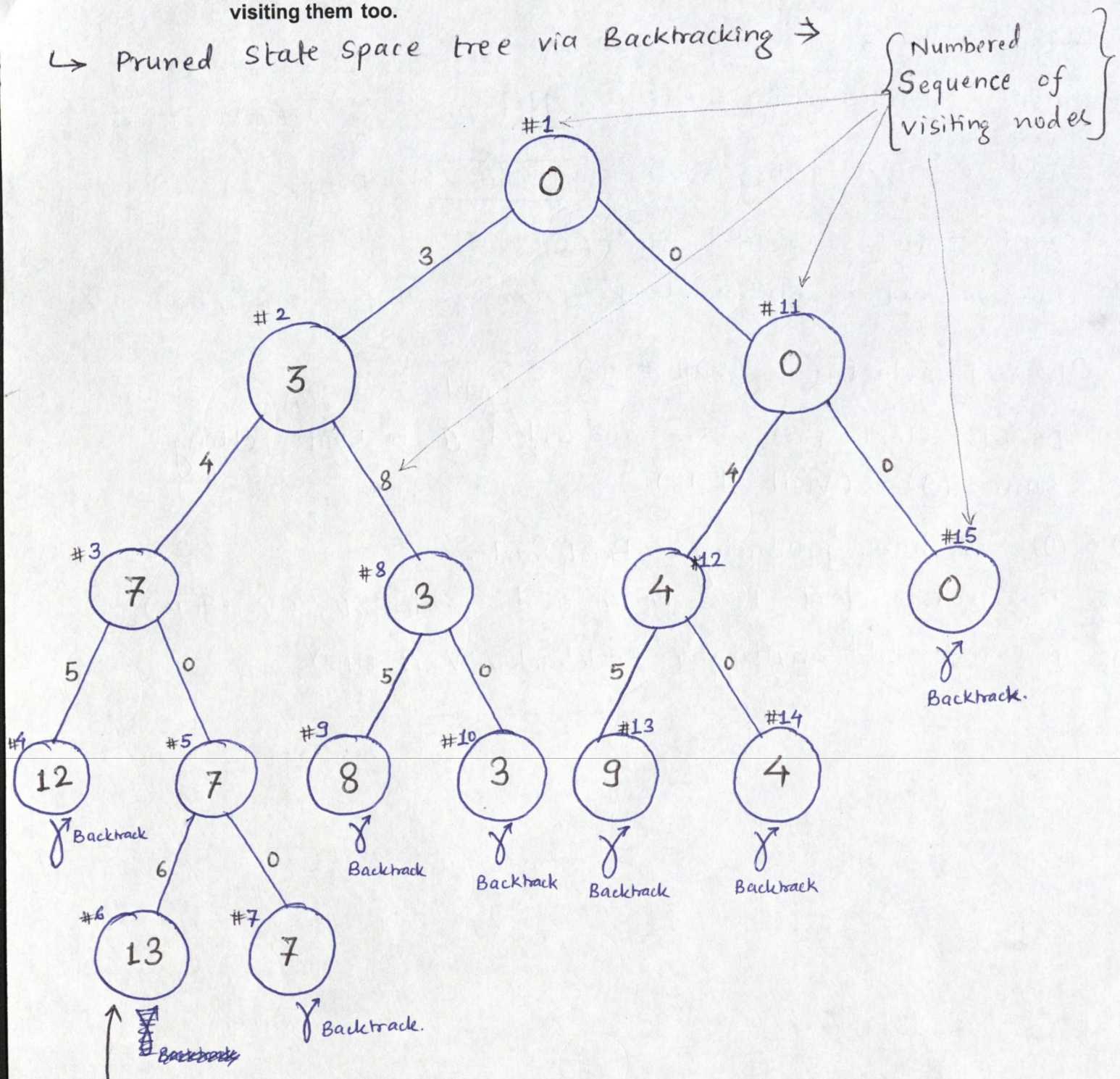
(23) Exclude item i_4 and return (Visit # 31)

(24) Final subset sum found at visit # 8 = 13 with nodes {3, 4, 6}.

(25) DFS in not an optimal soln. It is slow and generates all nodes. (or visits)

b. [30%] Find the subsets via backtracking. Draw a (pruned) state space tree and clearly show every step. Number the nodes in the sequence of visiting them too.

↳ Pruned state space tree via Backtracking →



Optimal soln: Subset sum found = {3, 4, 6}.

P.T.O. →

Steps:

(1) Root = ⓪ Node. Promising. And its not a solution.

(2) Expand Root. Add ③ to Root's Left child (visit #2)

(3) Node promising. Not a solution. Therefore expand to left.

(4) Add 4 to ③ getting node ⑦. Not a sol^n, promising.

(5) Add 5 to ⑦ getting node ⑫. Not promising. Backtrack.

(6) Add 6 to ⑦ getting node ⑬. Solution. Return.

(7) Add 5 to ③ getting node ⑧.
This is not a solution. Looks promising. Therefore return/backtrack.

(8) Do not select ③ (visit # 10)

(9) Donot select first item and select 5 to Root getting Sum = ⑨ (visit # 13).

(10) ⑨ is non promising. Backtrack.

(11) Do not select t_2 i.e. 4 and backtrack. (Visit # 14)

(12) Do not add anything (Backtrack. Visit # 15).

3. [30%] When the capacity of the knapsack is 16, solve the following 0-1knapsack problem using the backtracking algorithm discussed in class that uses the optimal fractional knapsack algorithm to compute the possible upper bound of the profit.

i	p_i	w_i	p_i / w_i
1	$10	5	$2
2	$30	5	$6
3	$40	2	$20
4	$50	10	$5

A3 KWF needs the i/p table sorted for calculating the Upper Bound -

i	Pi	wi	Pi/wi
3	$40	2	$20
2	$ 30	5	$ 6
4	$ 50	10	$5
1	$ 10	5	$2

Node 1: Profit = 0
weight = 0
UB (Upper Bound) = 0 + 40 + 30 + (16-7) * 5
= $115

Node 2: Profit = $10
weight = 5
UB = $10 + 40 + 30 + $\frac{50}{10}$ (16-12) = $100
∴ MaxProfit = $10

Node 3: Profit = $30
weight = 5
MaxProfit = $40
UB = $ 10 + $ 30 + 40 + (16-12) * 5 = $100

Node 4: Profit = $40
Weight = 2
UB = $40 + $10 + 30 + 5 * 4 = $100
MaxProfit = $80

Node 5: Profit = $50 → MaxProfit = 130
Weight = 10 But Total weight > C
~~UB = $50+~~
∴ Not feasible.

Node 6: Profit: $80
Weight = 12
UB = $80

Node 10: Profit = $10
Weight = 5
UB = 10 + 40 + 9*5 = 95

Node 11: Profit = $~~4~~0
Weight = 2
UB = $95

Node 12: Profit = 50
Weight = 10
UB = —
Total Profit = $100
Weight = 17 > 16
∴ Not feasible.

Node 16: weight = ~~$30~~5
Profit = $30
UB = $30 + 40 + (16 - 7) * 5
= $115

Node 17: Weight = $40
Profit = 2
UB = $115

Node 22: weight = 2
Profit = $40
UB = $40 + $50 = $98
+ $\frac{10}{5}$ (16 - 12)

Node 23: Weight = 10
Profit = $50
Upper Bound = $50 + $40 + $2*4 = $98
Total Profit = $90 (MaxProfit)
Weight = ~~8~~ 12
∴ Optimal Solution.

Node 25: weight = ~~$~~0
Profit = $0
~~wei~~ MaxProfit = $50
∴ Not able to produce output.

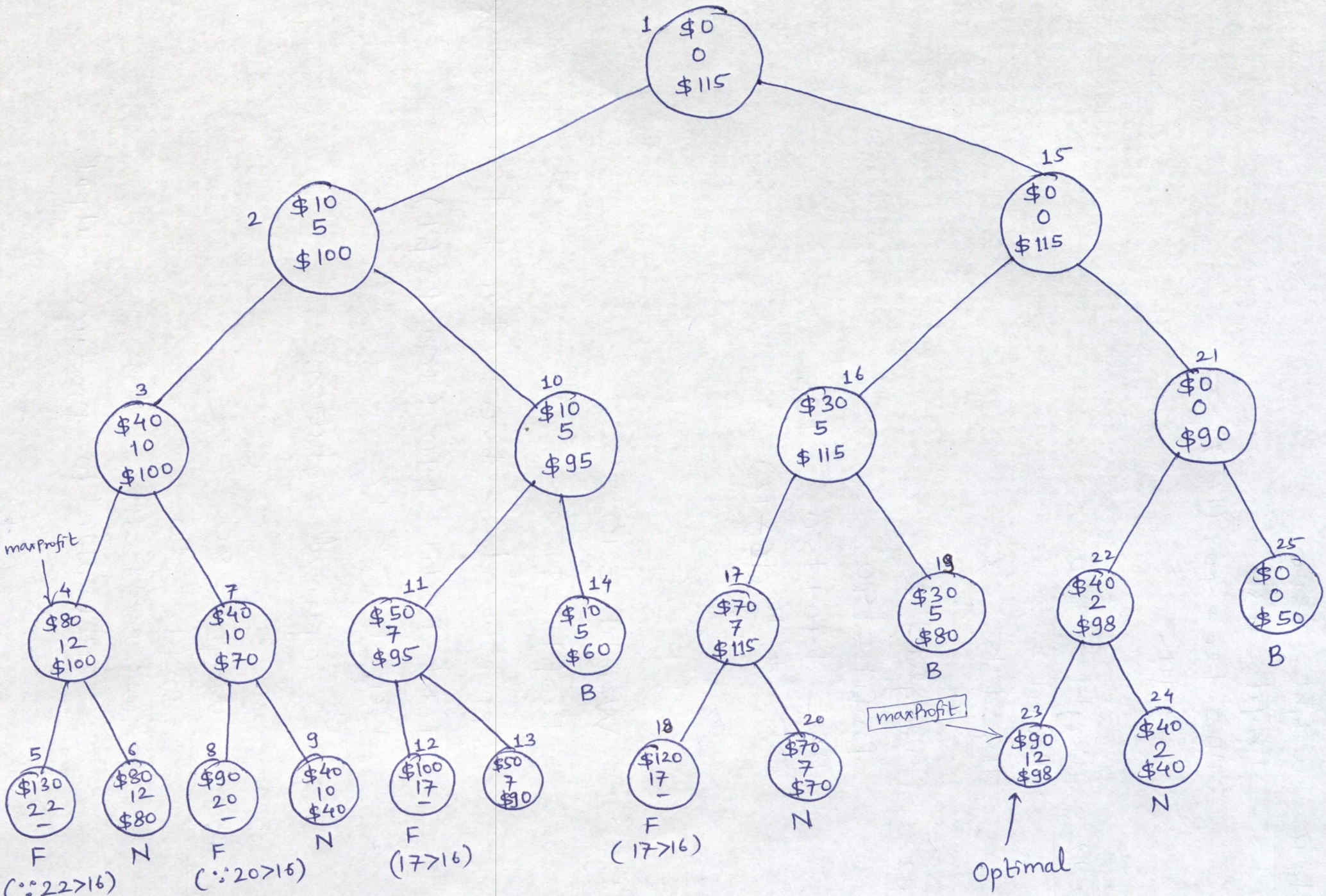
Node 16: weight = ~~$30~~5
Profit = $30
UB = $30 + 40 + (16 - 7) * 5
= $115

Node 17: Weight = $40
Profit = 2
UB = $115

Node 12: weight = 2
Profit = $40
UB = $40 + $50 = $98
+ $\frac{10}{5}$ (16 - 12)

Node 23: Weight = 10
Profit = $50
Upper Bound = $50 + $40 + $2 * 4 = $98
Total Profit = $90 (MaxProfit)
Weight = ~~8~~ 12
∴ optimal Solution.

Node 25: weight = ~~0~~0
profit = $0
~~wei~~ MaxProfit = $50
∴ Not able to produce output.


1
$0
0
$115
2
$10
5
$100
15
$0
0
$115
3
$40
10
$100
10
$10
5
$95
16
$30
5
$115
21
$0
0
$90
maxProfit
4
$80
12
$100
7
$40
10
$70
11
$50
7
$95
14
$10
5
$60
B
17
$70
7
$115
19
$30
5
$80
B
22
$40
2
$98
25
$0
0
$50
B
5
$130
22
–
6
$80
12
$80
8
$90
20
–
9
$40
10
$40
12
$100
17
–
13
$50
7
$90
18
$120
17
–
20
$70
7
$70
maxProfit
23
$90
12
$98
24
$40
2
$40
F
(∵22>16)
N
F
(∵20>16)
N
F
(17>16)
F
(17>16)
N
Optimal
N

4. [20%] Assume that a hash table has 17 buckets where each buckethas only one slot. A simple hash function: home bucket = key % 17 (where % is a mod function) is used to compute the home bucket based on the key. You are supposed to insert the following keys to the hash table: 6,12,34,29,28,11,23,7,0,33,30,45 using the following overflow handling methods.

(1) Using linear probing method ⇒

Key = 6 : 6 % 17 = 06 (position. Available)

Key = 12 : 12 % 17 = 12 (position Available)

Key = 34 : 34 % 17 = 00 (position Available)

Key = 29 : 29 % 17 = 12 (Not available. ∴ Collision)

↳ (29 % 17) + 1 = 13 (Available)

Key = 28 : 28 % 17 = 11 (Available)

Key = 11 : 11 % 17 = 11 (Not available. ∴ Collision)

↳ (11 % 17) + 1 = 12 (Not available ∴ Collision)

↳ (11 % 17) + 2 = 13 (Collision)

↳ (11 % 17) + 3 = 14 (Available)

Key = 23 : 23 % 17 = 6 (Not available. ∴ Collision)

↳ (23 % 17) + 1 = 7 (Available)

Key = 7 : 7 % 17 = 07 (Collision) → (7 % 17) + 1 = 8 (Available)

Key = 0 : 0 % 17 = 0 (Collision) ⇒ 0 % 17 + 1 = 1 (Available)

Key = 33 : 33 % 17 = 16 (Available).

Key = 30 : 30 % 17 = 13 (Collision) → (30 % 17 + 2) = 15 (Available)

Key = 45 : 45 % 17 = 28 → 28 % 17 = 11 (Collision) → ~~45~~

↳ ~~28~~ % 17 + 5 = 16 (Not available) --- Inserted at location (2).

- Hash Table ⇒

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
34	0	45	/	/	/	6	23	7	/	/	28	12	29	11	30	33

(2) Using Sorted chaining method ⇒

34 % 17 = 0 (first element @ 0'th index)

6 % 17 = 6 (first element @ bucket 6)

12 % 17 = 12 (First at bucket 12)

29 % 17 = 12 (2nd at bucket 12)

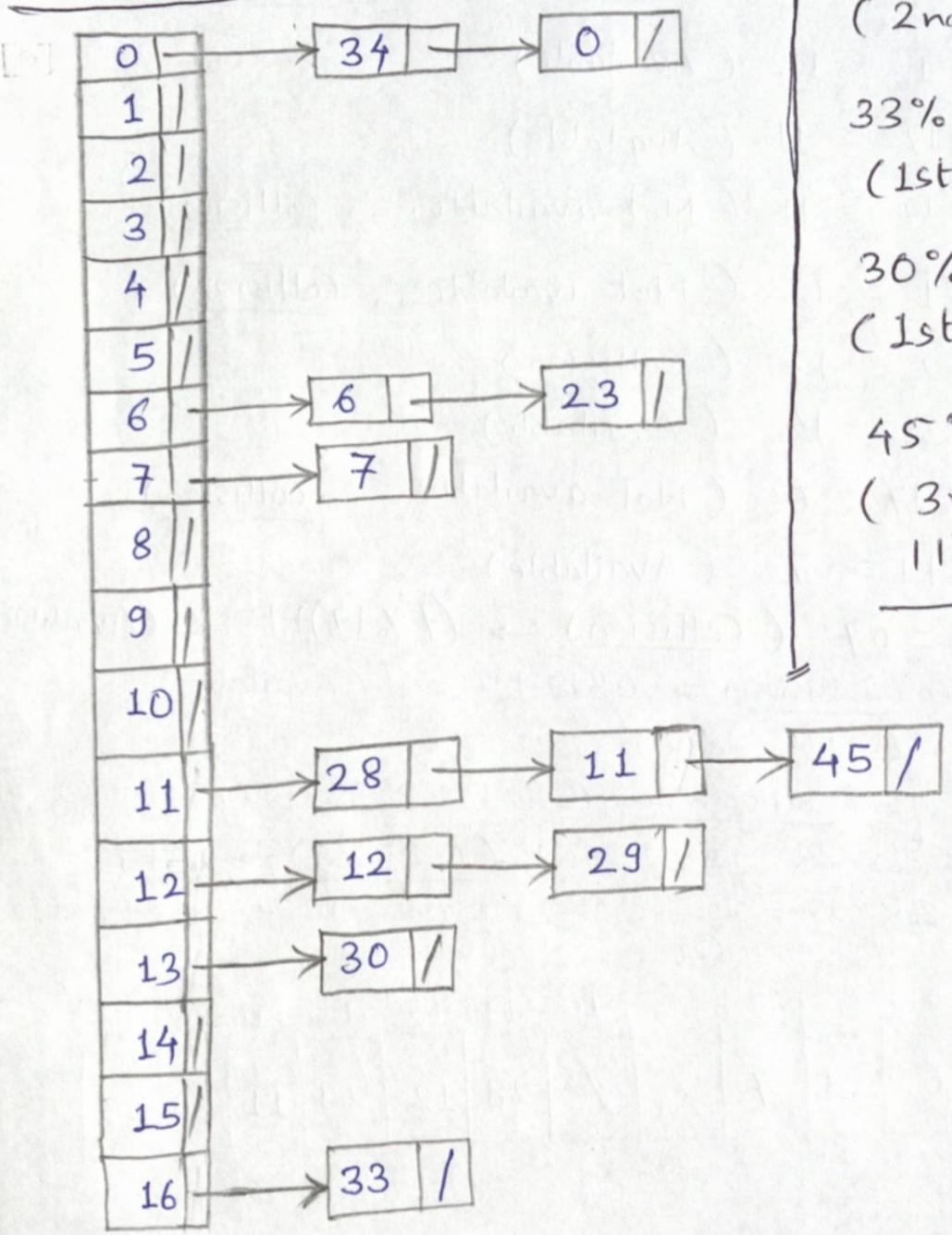
28 % 17 = 11 (1st at bucket 11)

11 % 17 = 11 (2nd at bucket 11)

23 % 17 = 6 (2nd at bucket 6)

7 % 17 = 7 (1st at bucket 7)

Sorted chain ⇒



0 % 17 = 0
(2nd at bucket 0)

33 % 17 = 16
(1st at bucket 16)

30 % 17 = 13
(1st at bucket 13)

45 % 17 = 11
(3rd element at bucket 11)