GURU NANAK DEV ENGINEERING COLLEGE, LUDHIANA
(AN AUTONOMOUS COLLEGE u/s 2(f) & 12(B) OF UGC ACT - 1956)
AICTE Approved, Punjab Govt. Aided Status ISO - 9001:2008 Certified, Affiliated to I.K. Gujral PTU Jalandhar
(E) Accredited UG Programmes, Institute Accredited by NAAC (A Grade) & TCS

Harkirat K.

University Roll. No.	Signature of Invigilator
1905334	

Dated 06-06-22 Class B.Tech IT Sec A2

Class Roll No. 192036 Subject DAA

Q. No.	1	2	3	4	5	6	7	Total Marks	Sig. of Examiner
Marks								24	

Ans6. Boyer-Moore algorithm:

=> The basic Boyer-Moore algorithm is based on two heuristic:
(i) Looking glass heuristic
(ii) Bad Character Rule / heuristic.

-> In the looking glass heuristic, it left for the direction in which the pattern will be checked with respect to text.
-> In for Boyer-moore, the the pattern matching starts from last character i.e. from right to left, but the pattern moves in left to right.

-> Bad Character rule says if there are mismatch values, try to make it a match value.
** It is done by checking if there is a pattern character in text pattern which is similar to character of mismatched value of te

Algorithm:-

Step 1:- First he examine the character of the text & pattern and note the size of the same i.e. m (pattern) & n (text).

Step 2:- Now we will apply the looking glass heuristic and start checking from right to

left.

Step3:- Now we will match the last character of pattern with the same last character of window of text which is equal to mx.

Step4:- If it is matched, move to next character & check.

Step5:- If it is mismatched, check apply the bad character rule & try to make next mismatch value, a match value.

Step6:- The process will go on until we find the matching string.

For example:-

t: C A B D A C D B
p: A C D B
② ①

First comparison is a mismatch, but the 2nd comparison can be matched with text value of 1st the comparison.

t: C A B D A C D B
p: A C D B
② ③

Next comparison 3 is a mismatch but comparison 4

keep on applying bad character rule, until all characters are matched.

t: C A B D A C D B
p: A C D B
✓ ✓ ✓ ✓

All characters are matched

Time Complexity:- O(nm)

⇒ Improved Boyer Moore Algorithm

⇒ In this first two rules are same of basic Boyer moore & a new rule is added as Good Suffix.
⇒ The in good suffix, if it finds a match value but the rest are mismatched, it will skip altogether and start from a next character.

For example:-

t: C A B D A C D B
p: A C D B
② ①
✗ ✓ ✗ ✗

Here ① comparison is a mismatch so good suffix rule will skip altogether & start the pattern from next character of text.

t: C A B D A C D B
A C D B
A C D B
✓ ✓ ✓ ✓

So here all characters are matched.

left.

Step3:- Now we will match the last character of pattern with the last character of window of text which is equal to m.

Step4:- If it is matched, move to next character & check.

Step5:- If it is mismatched, check & apply the bad character rule & try to make next mismatch value, a match value.

Step6:- The process will go on until we find the matching string.

For example:-

t: C A B D A C D B
p: A C D B
(2) (1) ×

First comparison is a mismatch, but the 2nd comparison can be matched with text value of 1st the comparison.

t: C A B D A C D B
p: A C D B
(3) (2) (1)
✓ ×

Now comparison 3 is a mismatch but comparison 2

keep on applying bad character rule until all characters are matched.

t: C A B D A C D B
p: A C D B
✓ ✓ ✓ ✓

All characters are matched.

Time Complexity:- O(nm)

⇒ Improved Boyer Moore Algorithm

⇒ In this first two rules are same of basic Boyer moore & a new rule is added i.e. Good Suffix.

⇒ In good suffix, if it finds a match values but the rest are mismatched, it will skip altogether and start from next character.

For example:-

t: C A B D A C D B
p: A C D B
(2) (1)
× × × ×

Here (1) comparison is a mismatch so good suffix rule will skip altogether & start the pattern from next character of text.

t: C A B D A C D B
A C D B
A C D B
✓ ✓ ✓ ✓

So here all characters are matched

Algorithm for Improved Boyer moore:-

Step1:- I will examine the text & pattern size.

Step2:- We will apply looking glass heuristic & start checking from right character & move to left.

Step3:- Now we will check whether to apply bad character rule or good suffix rule.

Step4:- If there are more skips in either of rules, that rule will be applied.

Step5:- For example if for a given string matching, no. of skips are more for bad character than good suffix, i.e BC > GS, Bad character rule will be applied, as there will be less comparisons, & vice versa.

Step6:- Step 3-4 will be repeated again & again until we find our matching string.

Time Complexity - O(n+m)

good

=> Boyer-Moore Horspool Algorithm:-

In this algorithm, string matching takes place on the basis of bad match table.

Algorithm:-

Step1:- Examine the size of text & pattern

Step 2:- Create a bad match table on the basis of letters & pattern size

Step3:- Update the table of bad match by checking values using formula (length-index-1)

GURU NANAK DEV ENGINEERING COLLEGE, LUDHI[ANA]
(AN AUTONOMOUS COLLEGE u/s 2(f) & 12(B) OF UGC ACT - 1956
AICTE Approved, Punjab Govt. Aided Status ISO : 9001:2008 Certified Affiliated to I.K. Gujral PTU Jalandhar
(E) Accredited UG Programmes, Institute Accredited by NAAC (A Grade) & TCS

Dated 6/6/22 Class D3 IT Sec A

Class Roll No. 1920036 Subject DAA

University Roll. No. 1905334

Total Marks

Q. No.	1	2	3	4	5	6	7
Marks							

Step4:- After last updation is completed, start matching the string from right to left.

Step5:- when there will be a mismatch, skip looking at the value of the letter in the bad match table.

Step6:- Repeat step 5, till we get the final result.

For example:-

t: t r u s t h a r d t o o t h b r u s h

p: t o o t h
[0] [1] [2] [3] [4]

Bad Match table:-

letters	t	o	h	*
values	4	3	5	5

value of t = length - index - 1
= 5 - 0 - 1 = 4

value of o = length - index - 1
= 5 - 1 - 1 = 5-2 = 3

value of h = same as length

Updating the table:-

letters	t	o	h	*
values	1	2	5	5

Value of t = 5-3-1
= 5-4 = 1

Value of o = 5-2-1
= 5-3 = 2

Again updated. The value of h will be same 5 as it is length & * is also 5 as it

represents set of the letters.

t : t r u s t h a r d t o o t h b r u s h
p : t o o t h
(1)
x

Now comparison 1 is a mismatch, and looking at the bad match table, we will skip it by (5) as the value mismatches at t.

If the value mismatched at 's', we would have skipped it by 5 skips as the value of 's' i.e. x is 5.

Following the same steps we will keep skipping until we reach the final string.

t : t r u e t h a r d t o o t h b r u s h
p : t o o t h
t o o t h
t o o t h

Time Complexity:

Average : O(n)
Worst case : O(nm)

Ans: The knapsack problem computes optimal solution if

$$\sum_{1 \le x_i < b} w_i x_i \le m \text{ & } \sum_{i=1}^{n} p_i x_i \text{ is maximized}$$

here, $0 \le x_i \le 1$ i.e. the value of x_i or solution is either 0 or 1.

The recursive backtracking algorithm uses the basis of bounding function (B_k) or recursive function.

```
Boundfunction (int k, float cp, float cw)
here int k = 0 , float cp = 1 , float cw = 1.
cp = current profit
cw = current weight.
```

Steps of solving 0/1 Knapsack using recursive backtracking:

Step 1:- Rearrange the items in decreasing order of there P_i/w_i

Step 2:- Find the upper bound ie. with the help of greedy approach.

Step 3:- Generate state space tree.

Step 4:- Expand the state space tree as to the upper bound calculated.

For example:-

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

m = 16

GURU NANAK DEV ENGINEERING COLLEGE, LUDHIANA

Upper Bound:

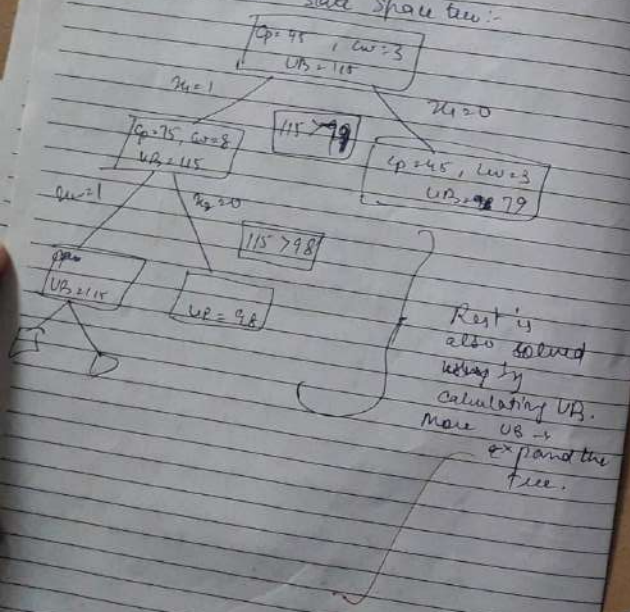
Step 1: Arrange all to pi/wi decreasing order:-

$P_1/w_1 > P_2/w_2 > P_3/w_3 > P_4/w_4$

Step 2: Calculate bound Upper Bound:-

	x_1	x_2	x_3
P	45	30	$8/9 \times 45 = 40 = 115$
W	3	5	$8/9 \times 9 = 16$

Step 3: Generate State Space tree:-



Rest is also solved using by calculating UB. More UB → expand the tree.

GURU NANAK DEV ENGINEERING COLLEGE

(AN AUTONOMOUS COLLEGE U/S 2(f) & 12(B) OF UGC ACT ...

AICTE Approved, Punjab Govt. Aided Status ISO : 9001:2008 Certified, Affiliated to I.K. Gujral PTU Jalandhar

(E) Accredited UG Programmes, Institute Accredited by NAAC (A Grade) & TCS

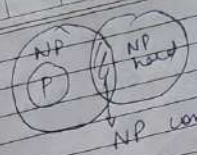
University Roll. No. 1905354 | Signature of ...

Dated ... Class ... Subject ... Sec ...

Class Roll No

Q. No	1	2	3	4	5	6	7	Total Marks	Sig. of Examiner
Marks									

Ans 4:



NP complete

#NP hard problems are based on satisfiability problems i.e. Is any satisfiability algorithm.

#NP complete problems are those problems which are converted to NP problems from NP hard decision problems.

NP hard problems can be decision problem or optimization problem.

Some NP hard problems can not be converted to NP complete.

NP hard decision problems can be converted to NP complete but these are some which cannot.

NP hard decision tree can't become NP complete for example Halting problem.

We can convert NP hard optimization problem to NP complete by decision problem.

i.e. by converting NP hard optimization problems to NP hard decision problem.

GURU NANAK DEV ENGINEERING COLLEGE, LUDHIANA

Upper Bound:

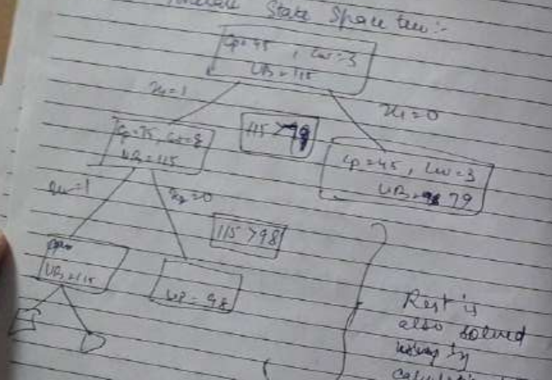
Step 1: Arrange in to p_i/w_i decreasing order:-

$p_1/w_1 > p_2/w_2 > p_3/w_3 > p_4/w_4$

Step 2: Calculate Upper Bound:-

	x_1	x_2	x_3	
P	45	30	$5/9 \times 45 = 40$	= 115
W	3	5	$5/9 \times 9$	= 16
			= 16.	

Step 3: Generate State Space tree:-



Rest is also solved using by calculating UB. More UB → expand the tree.

GURU NANAK DEV ...

(AN AUTONOMOUS COLLEGE ...)

AICTE Approved, Punjab Govt. Aided Status ISO : 9001:2008 Certified ... Accredited by NAAC (A Grade) ...

(E) Accredited UG Programmes, Institute ...

University Roll No.: 1905384

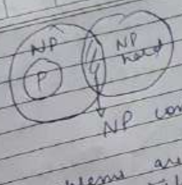
Dated ____ Class ____ Subject ____ Sec ____

Class Roll No. ____

Total Marks ____ Sig. of Examiner ____

Q. No.	1	2	3	4	5	6	7
Marks							

Ans 4.



4

⇒ NP hard problems are based on satisfiability problems i.e. satisfiability ∝ L1, where L1 is any algorithm.

⇒ NP complete problems are those problems which are converted from NP hard decision problems to NP problems.

⇒ NP hard problems can be decision or optimization problem. NP hard optimization problem can not be converted to NP complete.

⇒ NP hard decision problems can be converted to NP complete but there are some which cannot.

⇒ Some NP hard decision tree can't become NP complete for example halting problem.

⇒ We can convert NP hard optimization to NP complete by:- decision optimization problem & decision problem

i.e. by converting NP hard optimization problem to NP hard decision problem.

* Then this decision problem can be converted to NP complete.

Ans3: The sum of subsets problem is that the subset of elements are added up to the sum of the subset i.e. m.
The bounding function for this problem is:-

$B_k(x_1 \ldots x_k) = \text{true}$

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

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

The first condition is the sum of can values of k in subset is added to ~~k~~ new weights of values of k, should be greater than or equal to m.

The second condition is the sum of already computed ~~as~~ weights is added to w_{k+1}, should be ~~less~~ greater than or equal to m

Time complexity :- $O(p(n)2^m)$
where $p(n)$ is polynomial

Eg: w_1 w_2 w_3 w_4 $m = 31$

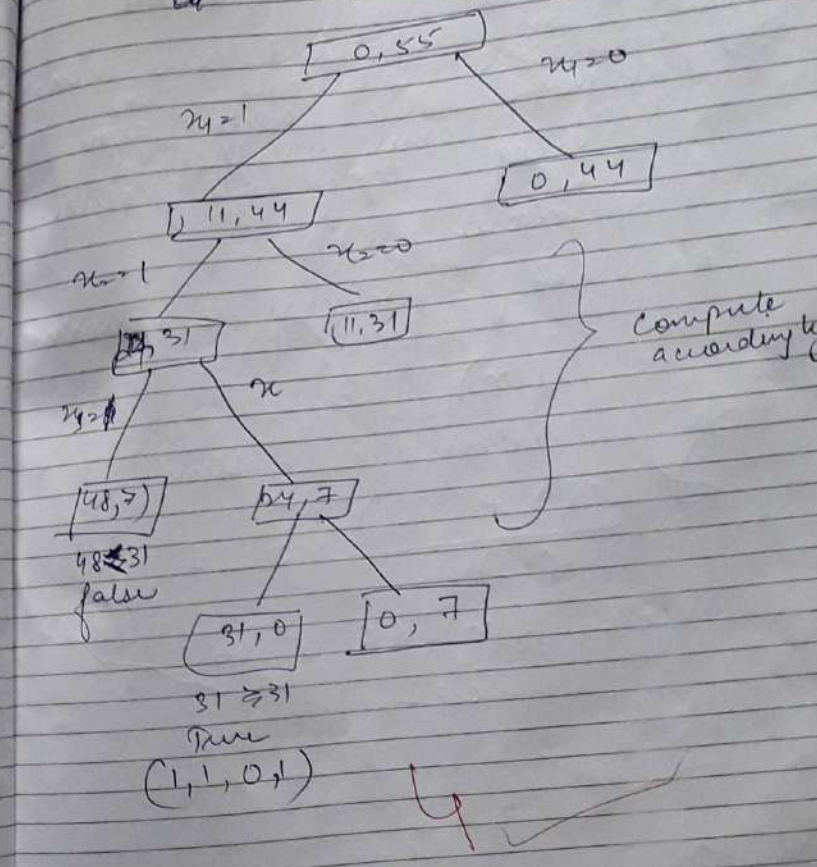
(11 13 24 7)

↓ ↓ ↓ ↓

1 1 0 1

or (0 0 1 1)

The sum should be equal to m.e. 31.



Ans 2: Satisfiability problem is that problem in which the values are checked with truth values, if the value is true, it is satisfiable. There are 2^n possible assignments.

This can be solved using non-deterministic algorithm in polynomial by checking some values & trying some values & not trying all values.

Ans 1: ε-approximate algorithm:

An algo algorithm is ε-approximate only & only if,

$f(n) \leq ε$.

For maximisation problem

$$\left| \frac{f^*(I) - f^\wedge(I)}{f^*(I)} \right| \leq 1$$

Here $f(n) \leq ε$, so

$ε = 1$.

Good