Module 1 MyNotes

08/08/2023

Page No.:
Date:

Greedy Algorithms:

- Approach to problem solving that involves making the best possible choice at each step without considering future steps.
- Idea of taking locally optimal choice in the hopes that it will lead to globally optimal soln.

Some greedy algorithms

1] Coin change
* 2] Fractional knapsack
3] Dijkstra's shortest path
4] kruskal's Minimum Spanning tree
5] Activity selection
* 6] Huffman Coding.

◉ Fractional knapsack problem:

▸ Problem Statement :

Given N items where each item has some weight and profit associated with it and also given a bag with capacity W. The task is to put the items into the bags such that the sum of profit associated with them is max possible.

▸ Constraint : You can take fraction of an item (unlike 0/1 knapsack where you can either take a item completely or not at all)

▶ Example :

Weight	Profit	Item
10	60	I1
20	100	I2
30	120	I3

capacity = 50

▶ Solution approach :

Steps 1] Calculate Profit / weight ratio.

2] Sort according to ratio.

3] Add items to knapsack from highest ratio until knapsack is full or there are no more items.

4] If fraction of an item is required to fill knapsack, take that fraction.

▶ Solution for the example :

1] ratios : I1 = 60/10 = 6

I2 = 100/20 = 5

I3 = 120/30 = 4

2] Sort : I1 > I2 > I3

3] i) add I1 = 10

ii) add I2 = 20

iii) add I3 = 20/30 ← from

20
20
10

50 -

∴ total profit = $60 + 100 + \left(\frac{20}{30} \times 120\right)$

= 240

Encoding message :

- Composed a message from string of characters
- Transforming regular text into special format
- Codes :

1) ASCII : Fixed length

- American standard code for information Interchange.
- A = 65
- a = 127
- represented using 8 bits per character
- Max : 256

2) UNICODE : fixed length

- No. of bits required to represent each character is 16-bits
- max : 65536

Fixed length Encoding :

- Here every character or symbol is given same number of bits.
- Fixed in sized
- Drawback :
  - Waste of space
  - character with small range or less frequecy, also gets same space as character with large range or high frequency
- Example :

Need to represent A to H

A : 0001 B : 0010 C : 0011 D : 0100
E : 0101 F : 0110 G : 0111 H : 1000

Message : CACABG
Encoding : 0011 0001 0011 0001 0010 0111
Frequencies : C(2) A(2) B(1) G(1)
decoding : CACABG

here characters B & G still require 4 bits each, even tough they could be represented using fewer bits, leads to space wastage.

- Solution : Use variable-length encoding.

Variable length Encoding :
- Method in which characters or symbols are assigned codes of different length based on their frequency of occurrence.
- Variable in size.
- Drawback :
  - Caused Confusion during decoding
  - challenging for the decoder to determine where one code ends and other begins.

- Example :
  Need to represent : A to D

A : 0 B : 1 C : 01 D : 11

Message : ABACD
Encode : 01001 11 (bits required = 7)
↳ if fixed length then bits : 10 (2 for each)

decoding : ABACD
CAABBB
ABAABBB
!
!

Confusion in decoding to determine start and end of single char.

- solution : Used variable-length encoding with prefix property.

Prefix property :

- Variable-length encoding with prefix property.
- where characters or symbols are assigned codes of different lengths based on their frequency occurence.
- Imp : Here no code is prefix of another code.
- avoids ambiguity during decoding.
- Drawbacks :
  - Might not always achieve best possible compression.

- Example :
  Need to represent A to D

A : 0
B : 10
C : 110
D : 111

message : ABCD
encoding : 0|0 1|1 0 1|1 1|1 — bits (9)
Decoding : AB CD

solution for the drawback : Huffman coding.

⊙ Huffman Coding tree :
Huffman coding is a method that efficiently encodes character or symbol based on their frequencies in a message.

Approach :
① Frequency Analysis -
→ Count the frequency of each character in given message. This helps in determining which characters are more common or less common

② Creating initial Nodes -
→ Create each character as separate Node.

③ Building Huffman Tree :
→ Combine two nodes with lowest frequencies to create a new parent node.
→ Repeat

④ Assigning binary codes :
→ Assign '0's on left side and '1's on right side
→ Path from root to each character node will give you binary code.

⑤ Creating table :

map each character with Binary code generated.

⑥ Encode the message using the same codes.

Example :

This is his message

char	T	h	i	s	m	e	a	g	_
freq.	1	2	3	5	1	2	1	1	3

original bits required = 4 bits for each character in message

= 4 x 19

= 76 bits (without Huffman)

* Apply Huffman

① Frequency Analysis :

T : 1

m : 1

a : 1

g : 1

h : 2

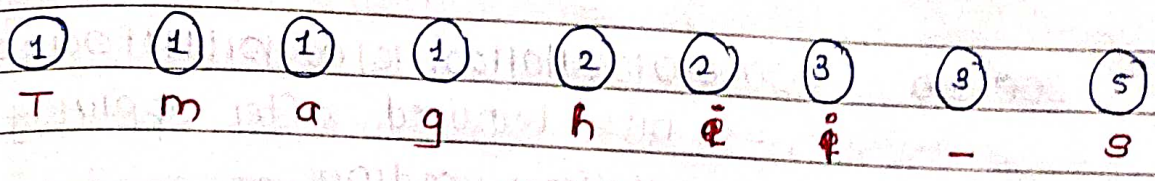
e : 2

i : 3

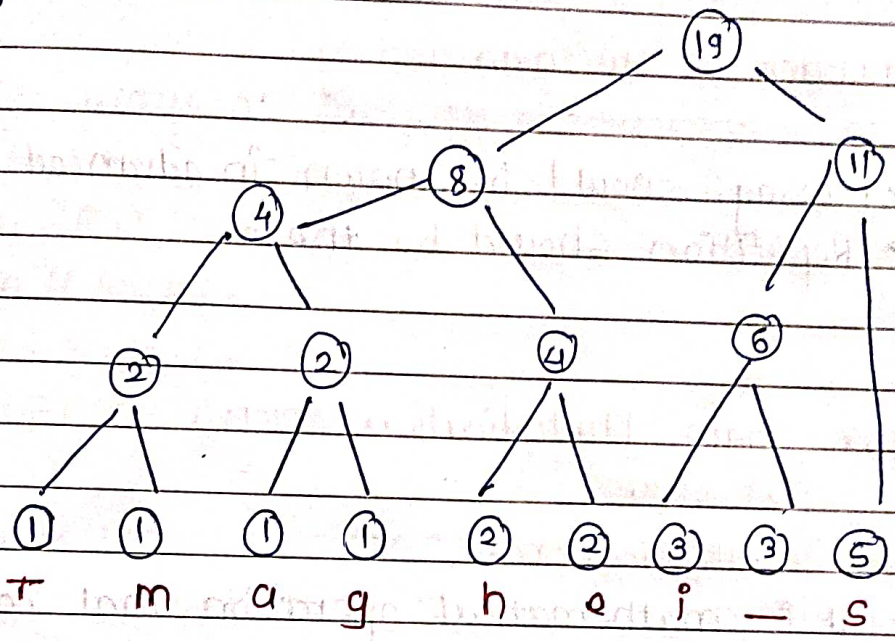
_ : 3

s : 5

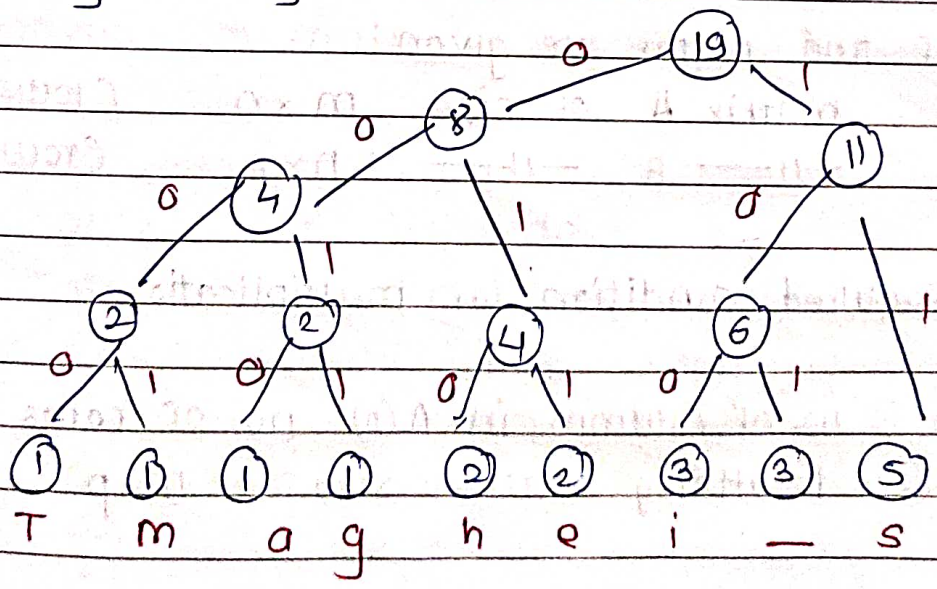
② Initial Node Creation



② Building Huffman tree :



④ Assigning Binary codes.



⑤ Table :

T = 0000	g = 0011	i = 100
m = 0001	h = 010	_ = 101
a = 0010	e = 011	s = 11

Encoded message :

~~0000100~~ 0000010 0100 110 100 110 000 101 1111 0010 0011 011

= 56 bits required after applying Huffman coding.

drawback of Huffman coding

→ Message should be known in advanced.
→ Repetition should be there.

Matrix Chain Multiplication (MCM) :

- Matrix multiplication -
  - It is mathematical operation that combines 2 matrices to produce third matrix.

- If two matrix are given
  matrix A of size m x n (rows x columns)
  —11— B —11— n x p (rows x colums)

- Required condition for multiplication
  1. no. of columns in A (n) = no. of rows in B (n)
  2. resulting matrix size = m x p

Example :

$$A = \begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix}_{2\times2} \quad B = \begin{bmatrix} 3 & 4 \\ 7 & 8 \end{bmatrix}_{2\times2}$$

$$C = \begin{bmatrix} 1*3+2*7 & 1*4+2*8 \\ 5*3+6*7 & 5*4+6*8 \end{bmatrix} = \begin{bmatrix} 17 & 20 \\ 39 & 46 \end{bmatrix}_{2\times2}$$

Matrix Chain Multiplication Concept :

- Suppose we have sequence or chain of n matrices to be multiplied.

Eg. A_1, A_2, A_3, A_4, are 4 matrices

possible ways to compute product

1. $((A_1 \cdot A_2)(A_3 \cdot A_4))$
2. $(A_1(A_2(A_3 \cdot A_4)))$
3. $(A_1((A_2 \cdot A_3) \cdot A_4))$
4. $(((A_1 \cdot A_2)A_3)A_4)$
5. $((A_1(A_2 A_3))A_4)$

To compute no. of scalar multiplications required we must know ① Algorithm to multiply 2 matrices
② matrix dimensions.

Algorithm to multiply 2 matrices : $A_{p \times q}$ and $B_{q \times r}$
Result $C() = C_{p \times r}$

```
1) For i ← 1 to p
2)     For j ← 1 to r
3)         C[i,j] ← 0
4)         For k ← 1 to q
5)             C[i,j] ← C[i,j] + A[i,k] · B[k,j]
6) return C
```

Find cost of matrix chain mutiplication.

eg A 10 x 100
B 100 x 5
C 5 x 50

2 ways of parenthesize

① ((AB)C) = $D_{10\times5}$. $C_{5\times50}$

↳ Cost of AB = 10.100.5 = 5000 scalar multiplications

Cost of DC = 10.5.50 = 2500 —11—

In total = 7500

② (A(BC)) = $A_{10\times100}$. $D_{100\times50}$

Cost of BC = 100.5.50 = 25000 scalar multiplications

Cost of AD = 10.100.50 = 50000 —11—

In total = 75,000

Two ways to solve MCM

1) Brute force : complexity 2^n

2) DP : —11— n^3

- Matrix Chain Multiplication Algorithm :

Problem Statement :

way1 - Algorithm applied to find the minimum cost way to multiply a sequence of matrices.

Input : sequence of matrices

output : lowest cost parenthesization.

way2 Given a chain $A_1, A_2, \ldots A_n$ of n matrices, where for $i = 1, 2 \ldots n$, matrix A_i with dimension $P_{i-1} \times P_i$ paranthesize the product such that total no. of scalar multiplications is

- DP Approach :

1) matrix chain A_{i_j} (i.e. $A_i, A_{i+1}, A_{i+2} \_\_ A_j$)
2) Split it betn A_k and A_{k+1} (k is $1 \le k < j$)
3) Compute $A_{i\_} A_k$ then $A_{k+1} \_\_ A_j$ and then multiply them to get final Result.

Suppose

$m[i,j]$ = min scalar multiplications required for A_{i_j} calculations.

then,

$$m[i,j] = m[i,k] + m[k+1, j] + P_{i-1} \cdot P_k \cdot P_j$$

- $m[i,k]$: cost of computing A_{i_k}
- $m[k+1, j]$: cost of computing A_{k+1_j}
- $P_{i-1} \cdot P_k \cdot P_j$: cost of multiplying A_{i_k} & A_{k+1_j}

* $m[i,i] = 0$ — when $i = 1, 2 \_\_ j$
* $m[i,j] = m[i,k] + m[k+1, j] + P_{i-1} \cdot P_k \cdot P_j$
* $k \longrightarrow i \le k < j$

$$m[i,j] = \begin{cases} 0 & i = j \\ \min \{ m[i,k] + m[k+1, j] + P_{i-1} \cdot P_k \cdot P_j \} & i < j \end{cases}$$

$s[i,j]$ = value of k at which $A_i A_{i+1} \_\_ A_j$ is split for optimal parenthesization

(s) keeps track for how to construct optimal sol^n.

→ Steps :

① Compute minimum-cost table = m

② Then compute split table = s

└ where

s[i,j] = k — shows where to split product A_{i_j} for min cost.

Example :

P = { 5, 4, 6, 2, 7 }

→ Here : $P_0 = 5$

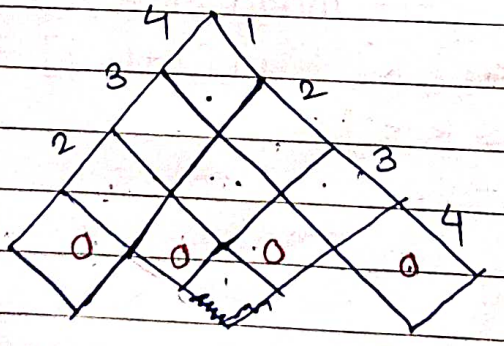
$P_1 = 4$

$P_2 = 6$

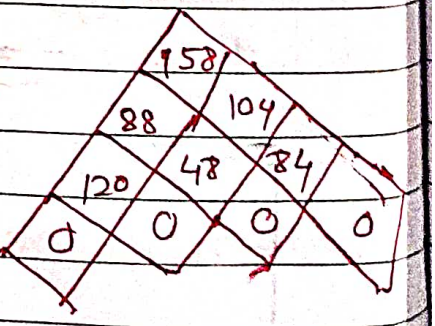
$P_3 = 2$

$P_4 = 7$

→ Compute m



→



i] as m[i,j] = 0 if i=j

So insert 0.

2] m[1,2] = m[1,1] + m[2,2] + $P_0 . P_1 . P_2$

= 0 + 0 + 5×4×6

= 120

(Here k = 1)

3] $m[2,3] = m[2,2] + m[3,3] + P_1 \cdot P_2 \cdot P_3$
$= 0 + 0 + 4 \times 6 \times 2$
$= 48$ (Here $k = 2$)

4] $m[3,4] = m[3,3] + m[4,4] + P_2 \cdot P_3 \cdot P_4$
$= 0 + 0 + 6 \times 2 \times 7$
$= 84$ (Here $k = 3$)

5] $m[1,3] = m[1,1] + m[2,3] + P_0 \cdot P_1 \cdot P_3$
$0 + 48 + 5 \times 4 \times 2 = (88)$

$= m[1,2] + m[3,3] + P_0 \cdot P_2 \cdot P_3$
$120 + 0 + 5 \times 6 \times 2 = 180$
$\min(180, 88) = 88$
$\therefore m[1,3] = 88$ (Here $k = 1$)

6] $m[2,4] = m[2,2] + m[3,4] + P_1 \cdot P_2 \cdot P_4$
$= 0 + 84 + 4 \times 6 \times 7$
$= (252)$
$m[2,3] + m[4,4] + P_1 P_3 P_4$
$= 48 + 0 + 4 \times 2 \times 7$
$= (104)$
$\therefore \min(252, 104) = 104$
$\therefore m[2,4] = 104$ (Here $k = 3$)

7) $m[1,4] = m[1,1] + m[2,4] + P_0 \cdot P_1 \cdot P_4$
$= 0 + 104 + 5 \times 4 \times 7 = (244)$

$= m[1,2] + m[3,4] + P_0 \cdot P_2 \cdot P_4$
$= 120 + 84 + 5 \times 6 \times 7 = (414)$

$= m[1,3] + m[4,4] + P_0 \cdot P_3 \cdot P_4$
$88 + 0 + 5 \times 2 \times 7 = (158)$

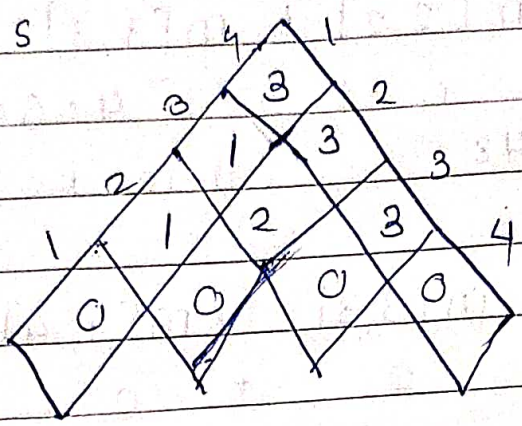
M T W T F S S

Page No.:

Date:

YOUVA

→ Compute s



22/6/1
My Notes

M T W T F S S
Page No.:
Date:
YOUVA

⊛ Longest Common Subsequence :

- Finding the longest subsequence that two sequences have in common.
- subsequence is a sequence that can be derived from another sequence by deleting some or no elements without changing the order of remaining elements.

Problem Statement :

Given two sequences, i.e sequence A of length n and sequence B of length m.
Find the longest subsequence present in both.

but should maintain their order.

- Longest Common sequence – not should be in continuous
- Longest Common Substring - Find longest string that is continous in both.

Example : String : ABCDE
substring : AB
subsequences : AC

- Formula to Find LCS :

If X and Y are two sequences then LCS of X and Y is given as

LCS (i,j) i] = 0 —— if i=j=0
ii] = 1 + LCS (i-1, j-1) —— if x[i] = y[j]
iii] = max (LCS(i,j-1), LCS (i-1,j)) —— if x[i] ≠ y[j]

it will give us :

1] Length of LCS

2] LCS

Example:

$X_{i=5} = \{A, C, A, D, B\}$

$Y_{j=4} = \{C, B, D, A\}$

Solution :

Y →		^	C	B	D	A
X ↓	^	0	0	0	0	0
	A	0	0	0	0	1
	C	0	1	1	1	1
	A	0	1	1	1	2
	D	0	1	1	2	2
	B	0	1	2	2	2

∴ Subsequences : CA — (1)

CD — (2)

CB — (3)

Example : $X_{i=7} = \{A, B, C, B, D, A, B\}$

$Y_{j=6} = \{B, D, C, A, B, A\}$

Y →		^	B	D	C	A	B	A
X ↓	^	0	0	0	0	0	0	0
	A	0	0	0	0	1	1	1
	B	0	1	1	1	1	2	2
	C	0	1	1	2	2	2	2
	B	0	1	1	2	2	3	3
	D	0	1	2	2	2	3	3
	A	0	1	2	2	3	3	4
	B	0	1	2	2	3	4	4

subsequences: BJAB — ①
BCBA — ②
BCAB — ③
[illegible] — [illegible]

⑥ knapsack 0/1 :

It involves selecting a subset of n items with specified weights and profit associated with it and also given a bag with capacity w. The task is put the items into the bag such that the sum of profit associated with them is max possible.

▶ Constraint / Variation that fractional knapsack :
Here each item can either be selected (0) or not selected(1), hence the name knapsack 0/1

① unlike fractional knapsack item is added to bag or not added.
② Greedy approach doesn't ensure optimal solution for every instance.
③ solution is using dp approach

Example :

Item	profit	weight	
①	2	1	Capacity = 8
②	4	3	
③	7	5	
④	10	7	

solution using dp :

- create m matrix

i	w → 0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1 (1,2)	0	1	2	2	2	2	2	2	2
2 (3,4)	0	2	2	4	6	6	6	6	6
3 (5,7)	0	2	2	4	6	7	9	9	11
4 (7,10)	0	2	2	4	6	7	9	10	12

weight must be in ascending manner

weight Profit

∴ max profit = 12

use ⇒ formula: $m[i,w] = \max(m[i-1,w], m[i-1, w-w[i]] + p[i])$