Walchand College of Engineering

(Government Aided Autonomous Institute)

Vishrambag, Sangli. 416415

Computer Algorithms

M1: Introduction

23-24

D B Kulkarni

Information Technology Department

Schedule

- 3 Lect/week

Tue (01:15-02:15PM) , Wed (01:15-02:15PM), Thu (09:00-10:00AM)

- 60 mins session
  - 5 mins Doubts/discussion/warmup
  - 20 mins session I
  - 5 mins break
  - 20 mins session II
  - 10 Discussion & attendance

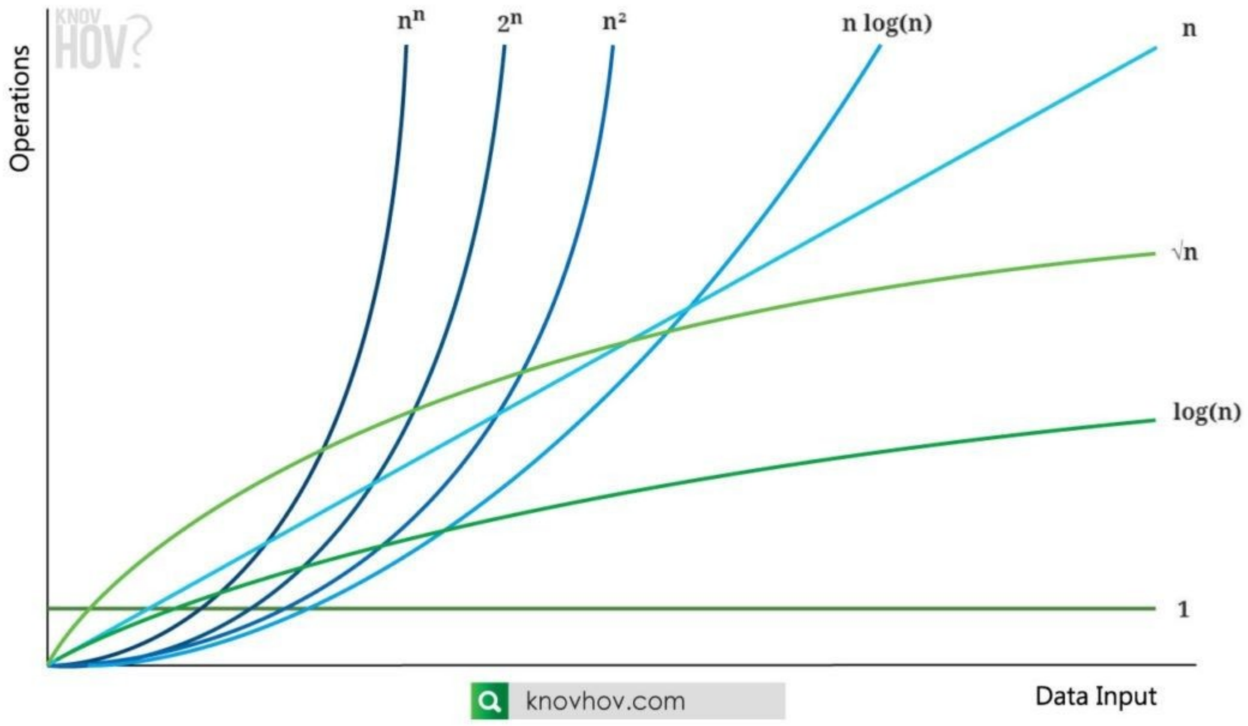
Algorithm

- What is an algorithm? –Pseudocode, logic
- Phases- Devise, Validate, Analyze, Test (DVAT)
- Properties:
  - Written in simple English, statement, description
  - Should accept all possible input- Generic
  - Platform (s/w, H/w independent)
  - Terminate
- Performance evaluation
  - Priori- Performance analysis
  - Posteriori- Performance measurement

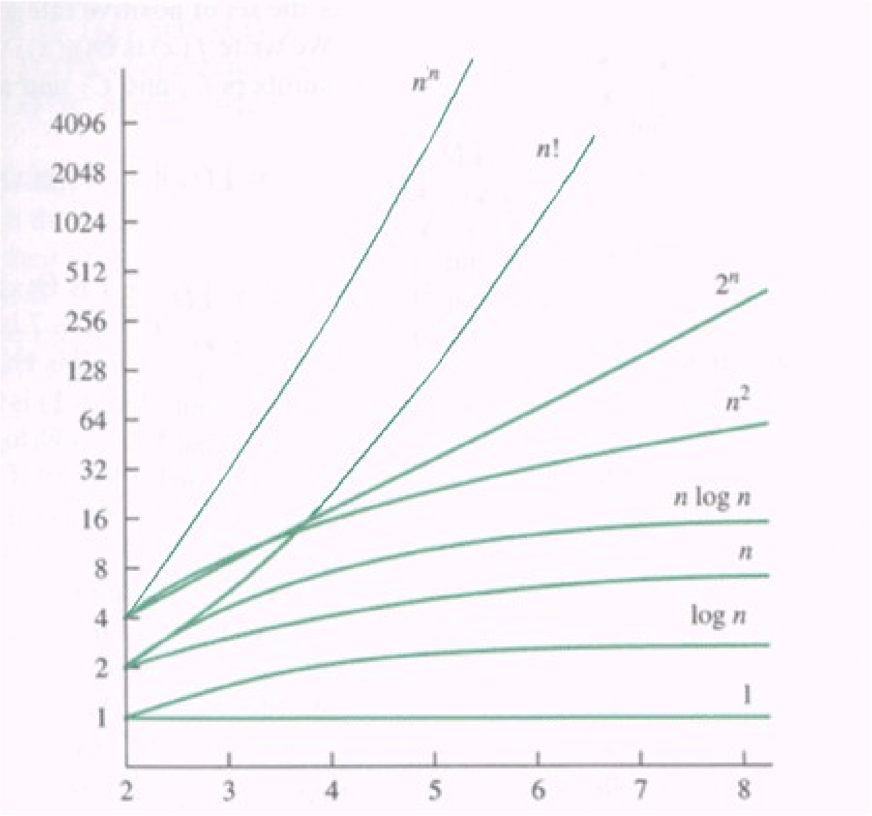
Algorithm: Analysis

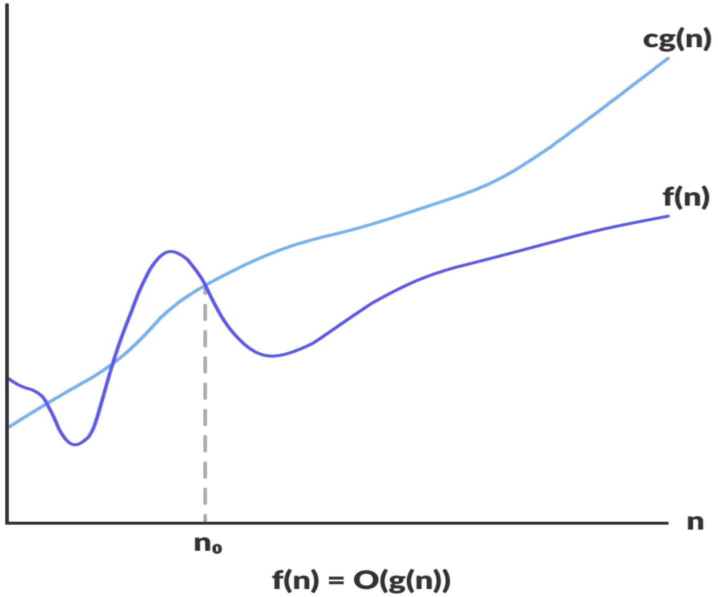
- Analysis: Primitive operation (comparison/item manipulations/#of use of function(module))
  - Complexity
    - Time
    - Space
- Asymptotic notations:
  - Big O (upper bound)
  - Omega (lower bound)
  - Theta (both bound)
- Strategies
  - Iterative
  - Recursive

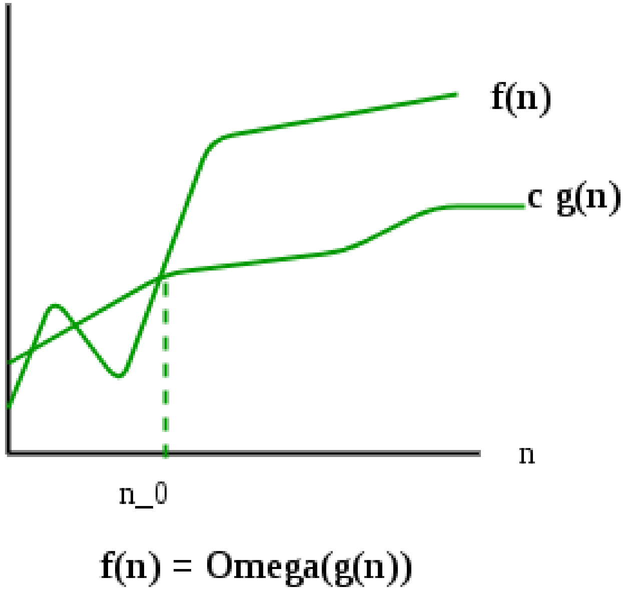
Algorithm Analysis: Growth of Function

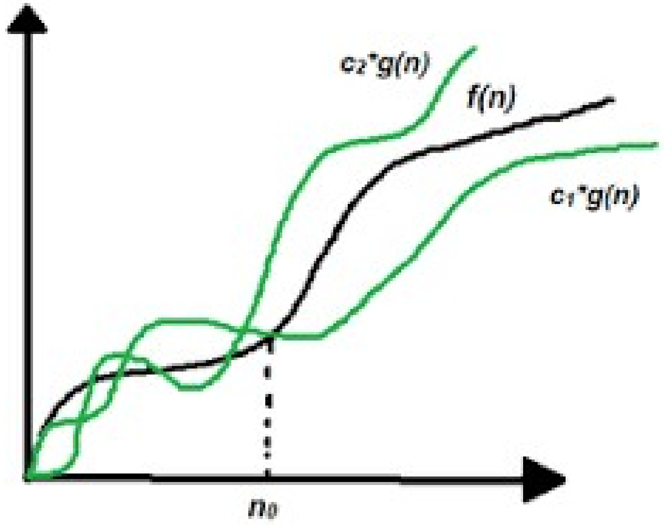


BIG O' NOTATION TIME COMPLEXITY









$f(n)=O(g(n))$

Upper Bound

$f(n)=\Omega(g(n))$

Lower Bound

$f(n)=\Theta(g(n))$

LU (both) Bound

Algorithms: Examples

1. Finding largest and smallest
2. Finding largest and second largest
3. Knapsack problem

Examples: Finding largest and

Finding largest and smallest

- (N-1) comparisons for each
- Total (2N-2)?
- (┌3N/2┐ -2)

Examples: Finding Winner and Runner-up

Finding Winner and Runner-up

(N-1) comparisons to find Winner, (N-2) for Runner-up= Total (3N-2)

- Find Winner (N-1)
- Runner-up could be found by finding winner between all payers who lost to the winner (┌log N┐ -1)
- Total N-2+(┌log N┐)

Examples: Knapsack problem

- YouTube Link https://www.youtube.com/watch?v=33k8EPNriTM
- Knapsack problem
  - Knapsack (Bag)
  - Weight handling capacity of Knapsack M
  - Items to be inserted
  - Item (wheat, jowar, rice...) Weight and profit factor given
- Optimization
  - Best the one which carries max profit!!!
  - Best the one which has least weight!!!

Fractional Knapsack

?Fractional Knapsack

- Fractional part of item/s are allowed to be added into the knapsack

Fractional Knapsack Problem: Put the items X_i (with weight w_i and profit p_i)in a knapsack of capacity M to get the maximum profit, given the weights and values of n items.

Maximise $\sum_{i=1}^{n} wipi$ with constraint of $(\sum_{i=1}^{n} wixi) \leq M$

- Strategies
    - Add item with min weight X
    - Add item with max profit X
    - Add item based on profit/weight ---Correct result

Fractional Knapsack: Example

Item (M=100)	**1**	**2**	**3**	4
Weight	60	10	30	80
Profit	30	40	10	20

Optimal:

Greedy Approach 1: Less weight: Sort items w˄- 2,3,1-p80

Greedy Approach 2: More profit: Sort items p˅-2,1,3/8(4)-p77.5

Correct Approach: Less weight/profit: Sort items (w/p) ˄-2,1,3-p80

0-1 Knapsack problem

- Fraction of item not allowed
  - Approach?
  - You cannot break an item, either pick the complete item or don't pick it (0-1 property).

- Brute force method: Consider all subsets of weight where sum is <M, select the one with highest profit
- Dynamic programming approach

- Determine the structure of an optimal solution
  - Decompose the problem into sub-problems
- Recursively define the value of an optimal solution.
- Compute the value of an optimal solution in a bottom-up fashion.
- Construct an optimal solution from computed information

Encoding messages

- **Encode a message composed of a string of characters**
- **Codes used by computer systems**
  - ASCII
    - uses 8 bits per character
    - can encode 256 characters
  - Unicode
    - 16 bits per character
    - can encode 65536 characters
    - includes all characters encoded by ASCII
- **ASCII and Unicode are *fixed-length codes***
  - all characters represented by same number of bits

Problem

Suppose that we want to encode a message constructed (only) from the symbols A, B, C, D, and E using a fixed-length code

- How many bits are required to encode each symbol?
  - at least 3 bits are required
  - 2 bits are not enough (can only encode four symbols)

How many bits are required to encode the message ACDBADDCDBAC?

- there are twelve symbols, each requires 3 bits
- 12*3 = 36 bits are required

A = 00
B = 01
C = 10
D = 11

A C D B A D D C D B A C

00 10 11 01 00 11 11 10 11 01 00 10

Fixed-length codes: Analysis

- **Benefit:**
  - Simple
- **Drawback:**
  - **Same number of bits used to represent all characters:** ‘a’ and ‘e’ occur more frequently than ‘q’ and ‘z’
    - **Wasted space:** Unicode uses twice as much space as ASCII, inefficient for plain-text messages containing only ASCII characters
- **Potential solution: use variable-length codes**
  - variable number of bits to represent characters when frequency of occurrence is known
  - short codes for characters that occur frequently

Variable-length codes

- **The advantage of variable-length codes over fixed-length is short codes can be given to characters that occur frequently**
  - on average, the length of the encoded message is less than fixed-length encoding
- **Potential problem: how do we know where one character ends and another begins?**
    - not a problem if number of bits is fixed!

- The following code **can not be used**

Symbol	Code
P	0
Q	1
R	01
S	10
T	11

- **As the pattern 1110 can be decoded as**
  - **1 1 1 0 (QQQP), 1 11 0 (QTP), 11 1 0 (QQS), or 11 10 (TS)**

Prefix property

- **A code has the prefix property if no character code is the prefix (start of the code) for another character**
- **Example:**

Symbol	Code
P	000
Q	11
R	01
S	001
T	10

$\boxed{01001101100010}$

$\boxed{\text{R S T Q P T}}$

- **000 is not a prefix of 11, 01, 001, or 10**
- **11 is not a prefix of 000, 01, 001, or 10 …**

- **Design a variable-length prefix-free code such that the message DEAACAAAAABA can be encoded using 22 bits**
- **Possible solution:**
  - **A** occurs eight times while **B**, **C**, **D**, and **E** each occur once
  - represent **A** with a one bit code, say 0
    - remaining codes cannot start with 0
  - represent **B** with the two bit code 10
    - remaining codes cannot start with 0 or 10
  - represent **C** with 110
  - represent **D** with 1110
  - represent **E** with 11110

Encoded message

DEAACAAAAABA

Symbol	Code
A	0
B	10
C	110
D	1110
E	11110

1110111100011000000100

22 bits

Another possible code

DEAACAAAAABA

Symbol	Code
A	0
B	100
C	101
D	1101
E	1111

11011111001010000001000

22 bits

Another possible code …contd

DEAACAAAAAB
A

Symbol	Code
A	0
B	100
C	101
D	110
E	111

1101110010100000100
0

20
bits

What code to use?

- **Question: Is there a variable-length code that makes the most efficient use of space?**

Answer: Yes!

Huffman coding

YouTube video : https://www.youtube.com/watch?v=dM6us854Jk0

- **Binary tree**
  - each leaf contains symbol (character)
  - label edge from node to left child with 0
  - label edge from node to right child with 1
- **Code for any symbol obtained by following path from root to the leaf containing symbol**
- **Code has prefix property**
  - leaf node cannot appear on path to another leaf
  - *note*: fixed-length codes are represented by a complete Huffman tree and clearly have the prefix property

Huffman tree

- **Find frequencies of each symbol occurring in message**
- **Begin with a forest of single node trees**
  - each contain symbol and its frequency
- **Do recursively**
  - select two trees with smallest frequency at the root
  - produce a new binary tree with the selected trees as children and store the sum of their frequencies in the root
- **Recursion ends when there is one tree**
  - this is the Huffman coding tree

Huffman code & encoded message

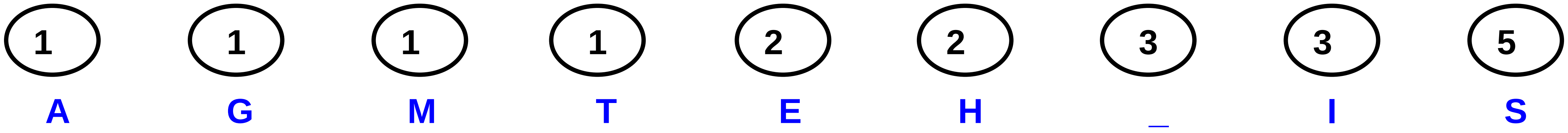
- **Build the Huffman coding tree for the message**

  This is his message

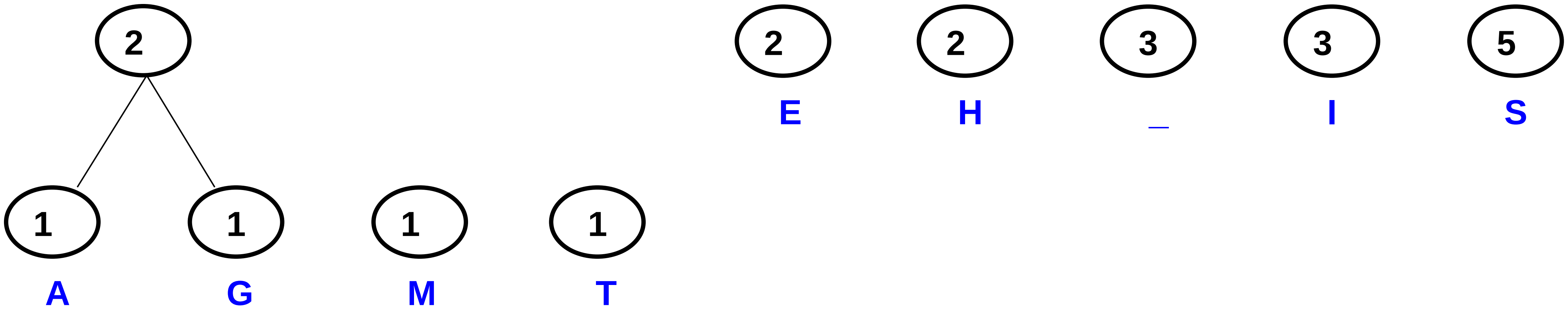
- **Character frequencies**

Char	A	G	M	T	E	H	_	I	S
Freq	1	1	1	1	2	2	3	3	5

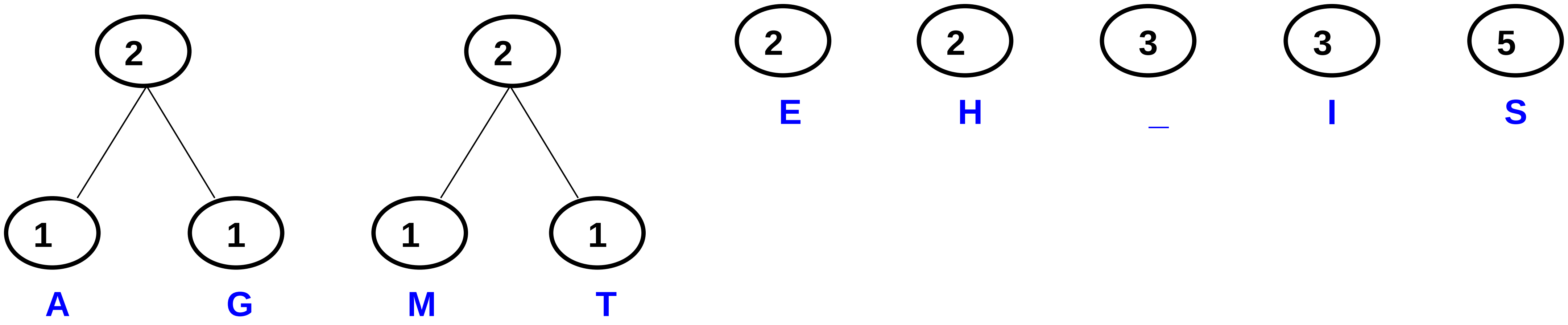
- **Begin with forest of single trees**



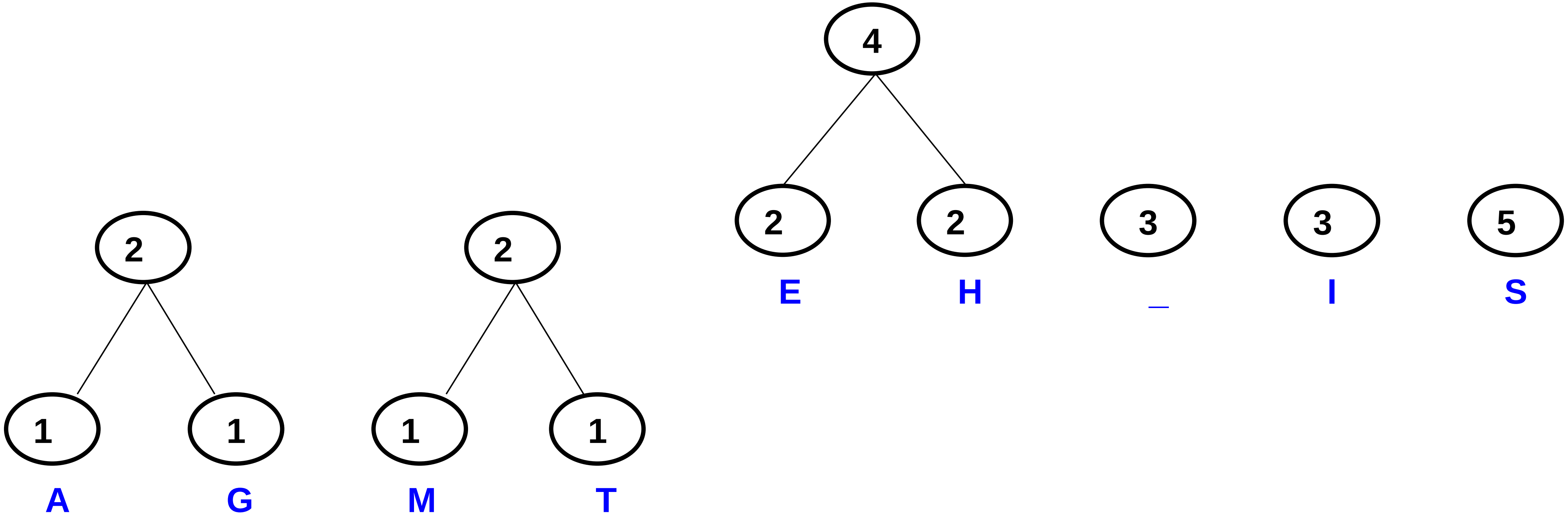
Huffman code & encoded message



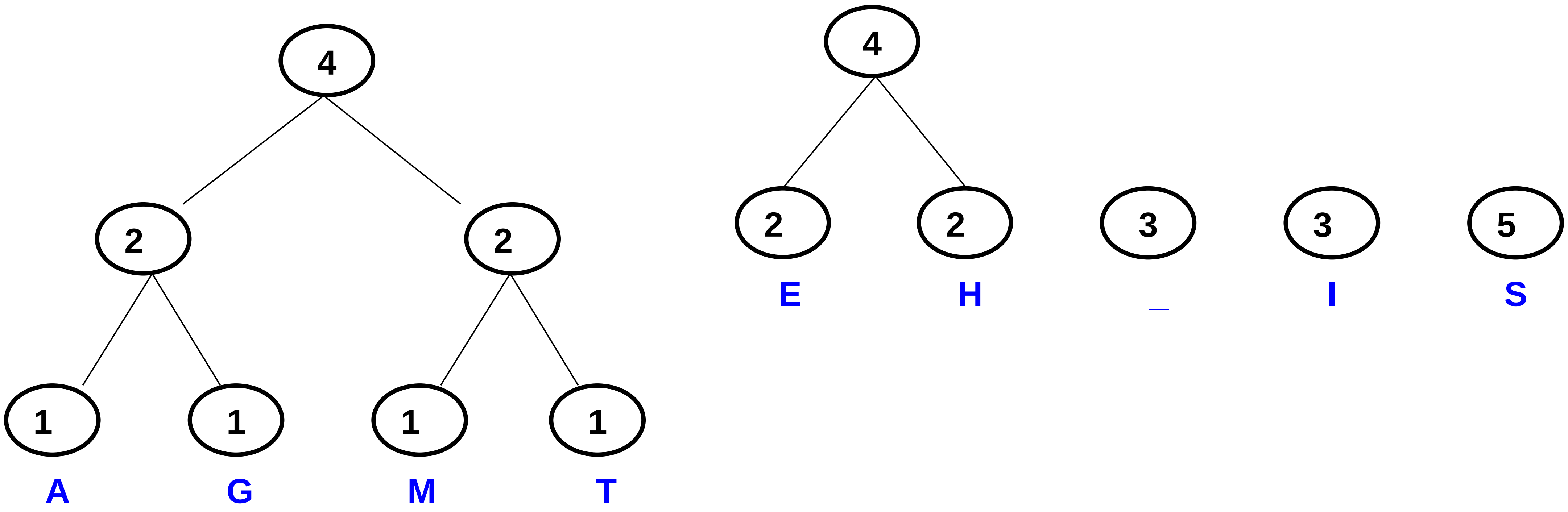
Huffman code & encoded message



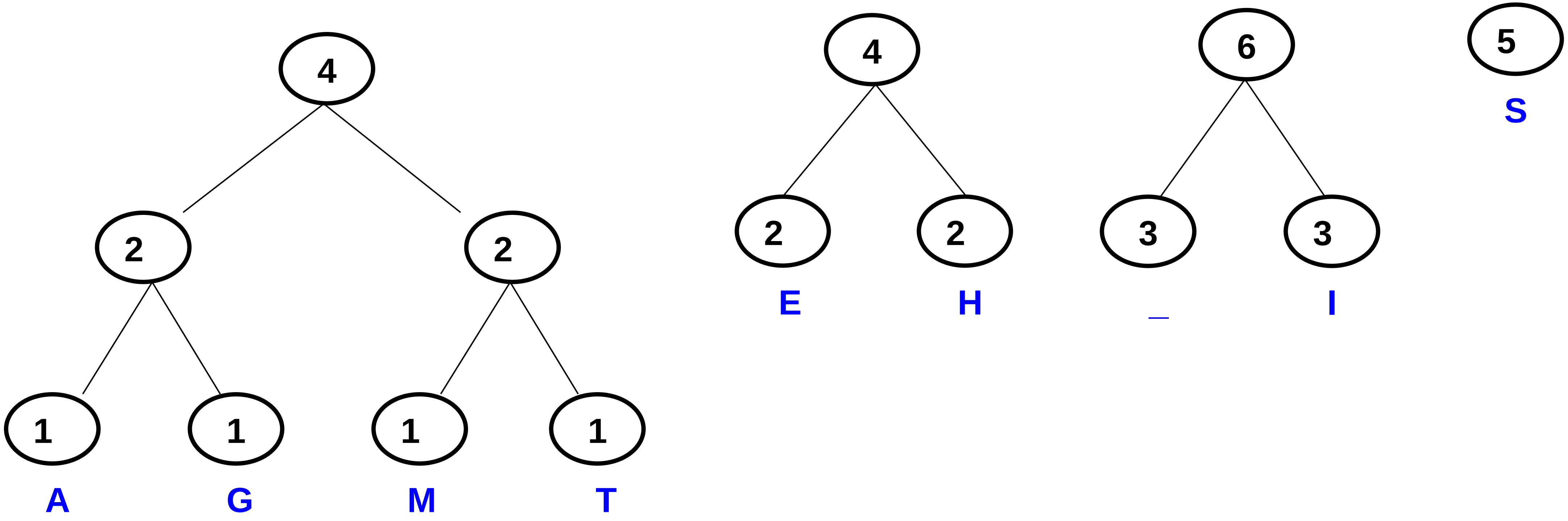
Huffman code & encoded message

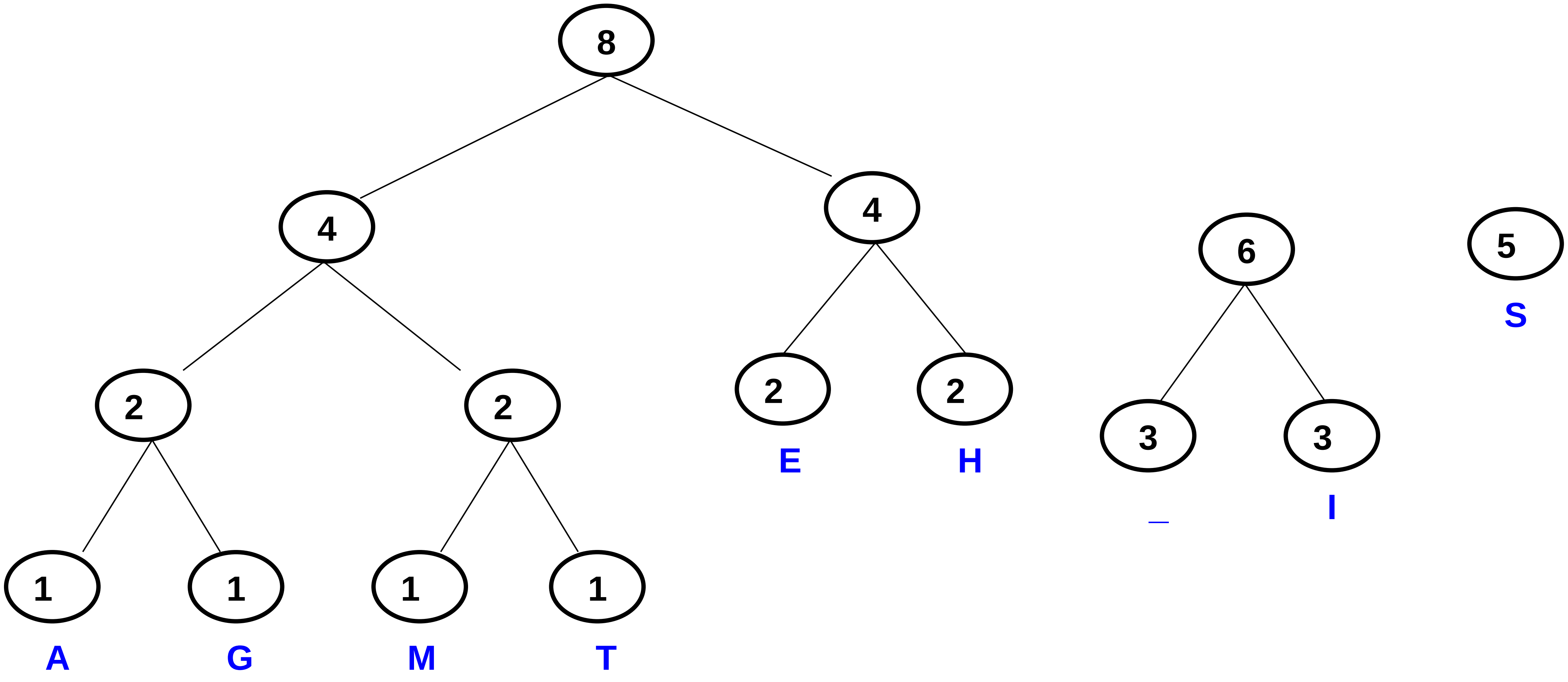


Huffman code & encoded message



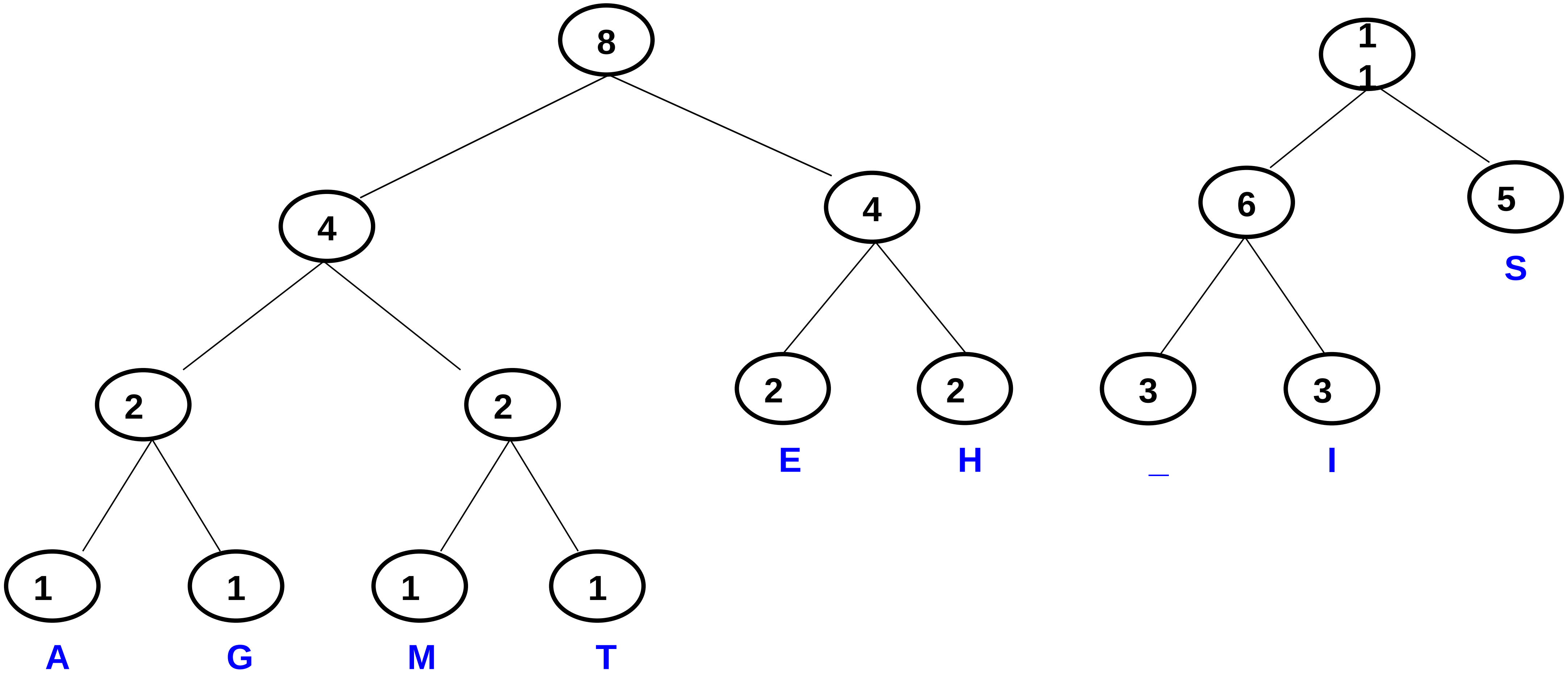
Huffman code & encoded message



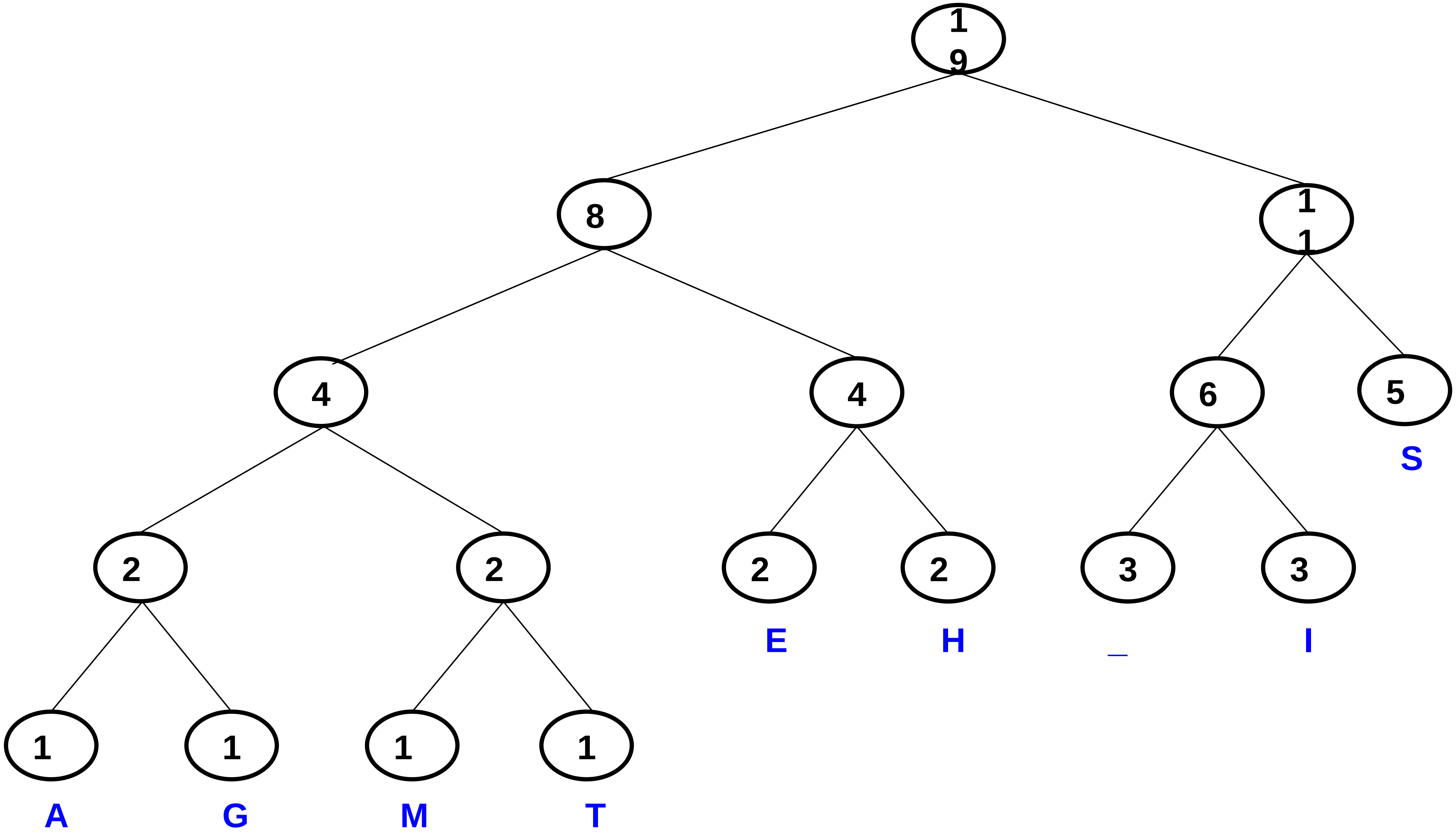


8
4
4
2
2
2
2
6
5
S
1
1
1
1
3
3
E
H
_
I
A
G
M
T

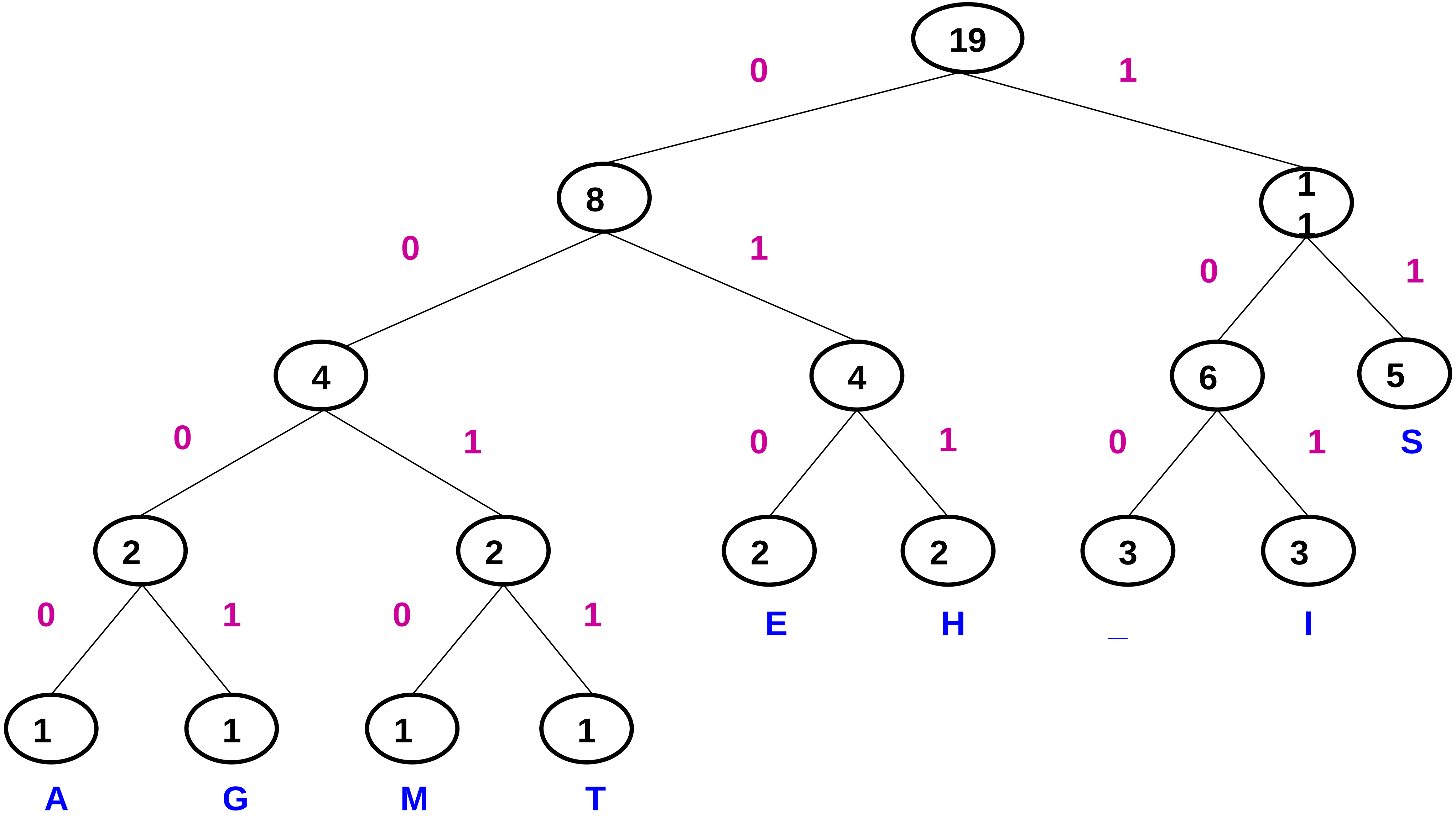
Huffman code & encoded message



Huffman code & encoded message



Huffman code & encoded message



Huffman code & encoded message

This is his message

S	11
E	010
H	011
_	100
I	101
A	0000
G	0001
M	0010
T	0011

0011 011 101 11 100 1011110001110111100001001011110000000001010

Matrix-chain Multiplication …contd

- Suppose we have A_1, A_2 matrices to be multiplied
  - That is, we want to compute the product $A_{1*}A_2$
  - Orders of matrix (rows*columns)
    - $A_1(pxq)$
    - $A_2(rxs)$
- Compatibility Condition for Mat-Mult q=r
- # Columns of A = # Rows of B

Matrix-chain Multiplication …contd

- Suppose we have a sequence or chain A_1, A_2, …, A_n of *n* matrices to be multiplied
  - That is, we want to compute the product $A_1A_2 \ldots A_n$
- There are many possible ways (parenthesizations) to compute the product

Matrix-chain Multiplication …contd

- Example: consider the chain A_1, A_2, A_3, A_4 of 4 matrices
  - Let us compute the product $A_1A_2A_3A_4$
- There are 5 possible ways:

1. $(A_1(A_2(A_3A_4)))$
2. $(A_1((A_2A_3)A_4))$
3. $((A_1A_2)(A_3A_4))$
4. $((A_1(A_2A_3))A_4)$
5. $(((A_1A_2)A_3)A_4)$

Matrix-chain Multiplication ...contd

- To compute the number of scalar multiplications necessary, we must know:
  - Algorithm to multiply two matrices
  - Matrix dimensions

- Can you write the algorithm to multiply two matrices?

Input: Matrices $A_{p\times q}$ and $B_{q\times r}$ (with dimensions $p\times q$ and $q\times r$)

Result: Matrix $C_{p\times r}$ resulting from the product $A\cdot B$

MATRIX-MULTIPLY$(A_{p\times q}, B_{q\times r})$

1. **for** $i \leftarrow 1$ **to** p
2.   **for** $j \leftarrow 1$ **to** r
3. $C[i, j] \leftarrow 0$
4.   **for** $k \leftarrow 1$ **to** q
5. $C[i, j] \leftarrow C[i, j] + A[i, k] \cdot B[k, j]$
6. **return** C

Scalar multiplication in line 5 dominates time to compute.
Number of scalar multiplications = *pqr*

- Example: Consider three matrices $A_{10\times100}$, $B_{100\times5}$, and $C_{5\times50}$
- There are 2 ways to parenthesize
  - $((AB)C) = D_{10\times5} \cdot C_{5\times50}$
    - $AB \Rightarrow 10 \cdot 100 \cdot 5 = 5{,}000$ scalar multiplications
    - $DC \Rightarrow 10 \cdot 5 \cdot 50 = 2{,}500$ scalar multiplications

    Total: 7,500
  - $(A(BC)) = A_{10\times100} \cdot E_{100\times50}$
    - $BC \Rightarrow 100 \cdot 5 \cdot 50 = 25{,}000$ scalar multiplications
    - $AE \Rightarrow 10 \cdot 100 \cdot 50 = 50{,}000$ scalar multiplications

    Total: 75,000

- For example, if A_1 is a 10×30 matrix, A_2 is a 30×5 matrix, and A_3 is a 5×60 matrix, then

$(A_1A_2) A_3$ (10×30×5) + (10×5×60) = 1500 + 3000 = 4500 operations

$A_1 (A_2A_3)$ (30×5×60) + (10×30×60) = 9000 + 18000 = 27000 operations

- To multiply n matrices $\{A_1,A_2,A_3.... A_n\}$, all combinations can be checked and least one can be accepted….brute force method…..complexity 2^n
- Dynamic programming approach has complexity of n^3

- Matrix-chain multiplication problem
  - Given a chain $A_1, A_2, \ldots, A_n$ of n matrices, where for i=1, 2, …, n, matrix A_i has dimension $p_{i-1} \times p_i$
  - Parenthesize the product $A_1A_2 \ldots A_n$ such that the total number of scalar multiplications is minimized
- Brute force method of exhaustive search takes time exponential in n

Dynamic Programming Approach …contd

- The structure of an optimal solution
  - Let us use the notation $A_{i..j}$ for the matrix that results from the product $A_i\, A_{i+1}\, \ldots\, A_j$
  - An optimal parenthesization of the product $A_1A_2\ldots A_n$ splits the product between A_k and A_{k+1} for some integer k where $1 \le k < n$
  - First compute matrices $A_{1..k}$ and $A_{k+1..n}$; then multiply them to get the final matrix $A_{1..n}$

- **Key observation**: parenthesizations of the subchains $A_1A_2\ldots A_k$ and $A_{k+1}A_{k+2}\ldots A_n$ must also be optimal if the parenthesization of the chain $A_1A_2\ldots A_n$ is optimal (why?)

- That is, the optimal solution to the problem contains within it the optimal solution to subproblems

- Recursive definition of the value of an optimal solution
  - Let $m[i, j]$ be the minimum number of scalar multiplications necessary to compute $A_{i..j}$
  - Minimum cost to compute $A_{1..n}$ is $m[1, n]$
  - Suppose the optimal parenthesization of $A_{i..j}$ splits the product between A_k and A_{k+1} for some integer k where $i \leq k < j$

Dynamic Programming Approach …contd

- $A_{i..j} = (A_i A_{i+1} \ldots A_k) \cdot (A_{k+1} A_{k+2} \ldots A_j) = A_{i..k} \cdot A_{k+1..j}$
- Cost of computing $A_{i..j}$ = cost of computing $A_{i..k}$ + cost of computing $A_{k+1..j}$ + cost of multiplying $A_{i..k}$ and $A_{k+1..j}$
- Cost of multiplying $A_{i..k}$ and $A_{k+1..j}$ is $p_{i-1} p_k p_j$

- $m[i, j] = m[i, k] + m[k+1, j] + p_{i-1} p_k p_j \qquad \text{for } i \le k < j$
- $m[i, i] = 0$ for $i = 1, 2, \ldots, n$

Dynamic Programming Approach …contd

- But… optimal parenthesization occurs at one value of k among all possible $i \le k < j$
- Check all these and select the best one

$$m[i, j] = \begin{cases} 0 & \text{if } i=j \\ \min\limits_{i \le k < j} \{m[i, k] + m[k+1, j] + p_{i-1} p_k p_j\} & \text{if } i<j \end{cases}$$

Dynamic Programming Approach …contd

- To keep track of how to construct an optimal solution, we use a table *s*
- $s[i, j]$ = value of *k* at which $A_i A_{i+1} \ldots A_j$ is split for optimal parenthesization
- Algorithm: next slide
  - First computes costs for chains of length l=1
  - Then for chains of length l=2,3, … and so on
  - Computes the optimal cost bottom-up

Algorithm to Compute Optimal Cost

Input: Array $p[0\ldots n]$ containing matrix dimensions and n

Result: Minimum-cost table m and split table s

MATRIX-CHAIN-ORDER(p[ ], n)

```
for i ← 1 to n
        m[i, i] ← 0
for l ← 2 to n
        for i ← 1 to n-l+1
                j ← i+l-1
                m[i, j] ← ∞
                for k ← i to j-1
                        q ← m[i, k] + m[k+1, j] + p[i-1] p[k] p[j]
                        if q < m[i, j]
                                m[i, j] ← q
                                s[i, j] ← k
return m and s
```

Takes $O(n^3)$ time

Requires $O(n^2)$ space

Constructing Optimal Solution

- Our algorithm computes the minimum-cost table *m* and the split table *s*
- The optimal solution can be constructed from the split table *s*
  - Each entry $s[i, j]=k$ shows where to split the product $A_i A_{i+1} \ldots A_j$ for the minimum cost

Example

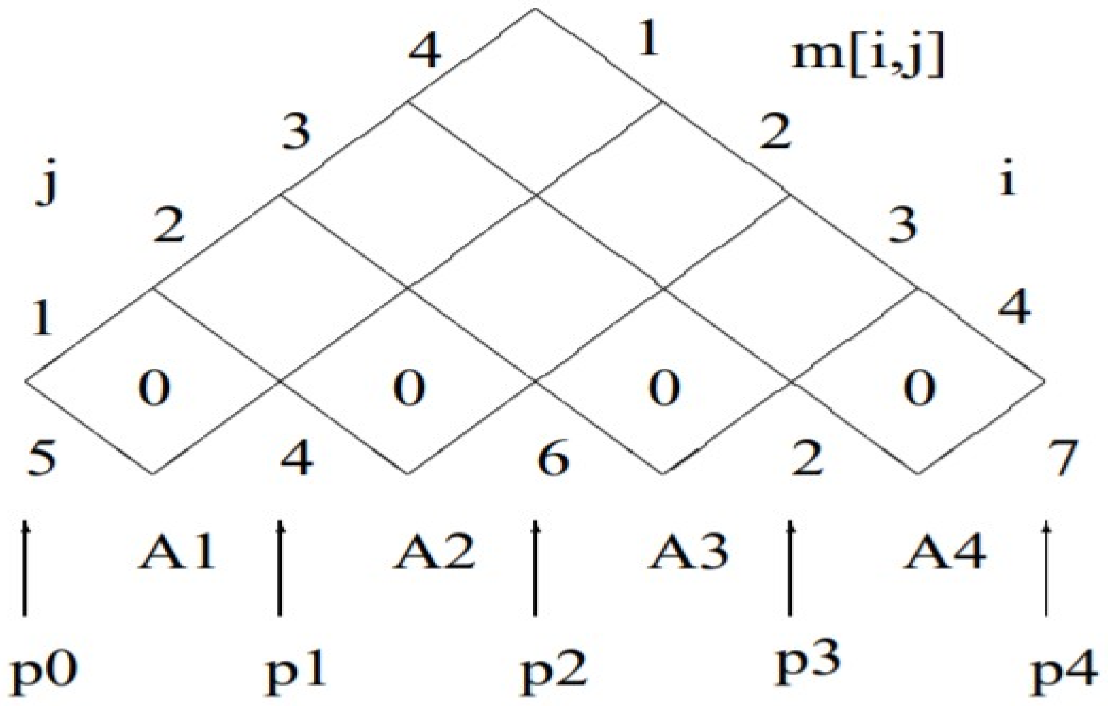
- Show how to multiply this matrix chain optimally

- Solution on the board
  - Minimum cost 15,125
  - Optimal parenthesization $((A_1(A_2A_3))((A_4 A_5)A_6))$

Matrix	Dimension
A_1	30×35
A_2	35×15
A_3	15×5
A_4	5×10
A_5	10×20
A_6	20×25

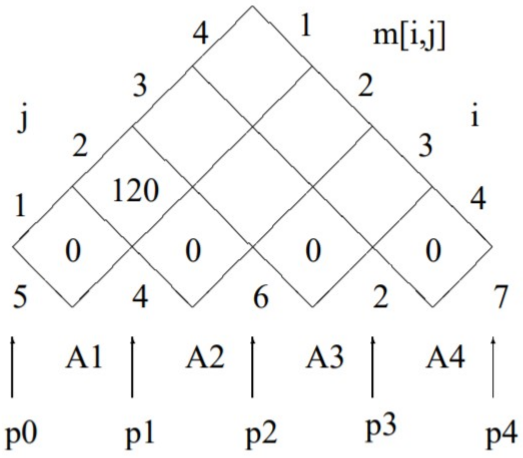
MCM Example

- For four matrices with vector p= {5,4,6,2,7} such that {A_1(5x4),A_2(4x6),A_3(6x2),A_4(2x7)} find anesthetization
- To find k such that m[1,4] is minimum

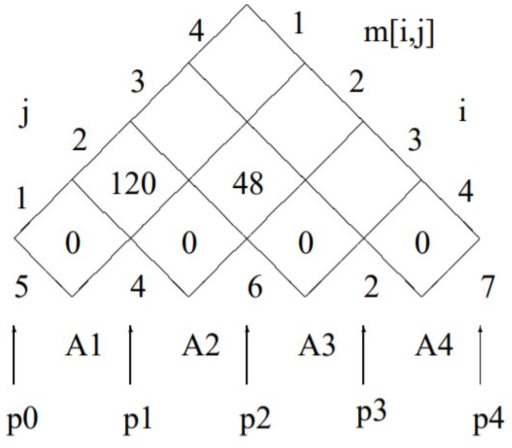


MCM Example..contd...

Compute	Step 1	Step 2	Step 3
Phase 1	m[1,2],m[2,3],m[3,4]	m[1,3], m[2,4]	m[1,4]
Record	s[1,4]	s[1,s[1,4]] or s[s[1,4],4]	Parentisize



$$m[1,2] = \min_{1\le k<2}(m[1,k]+m[k+1,2]+p_0p_kp_2) = m[1,1]+m[2,2]+p_0p_1p_2 = 120.$$



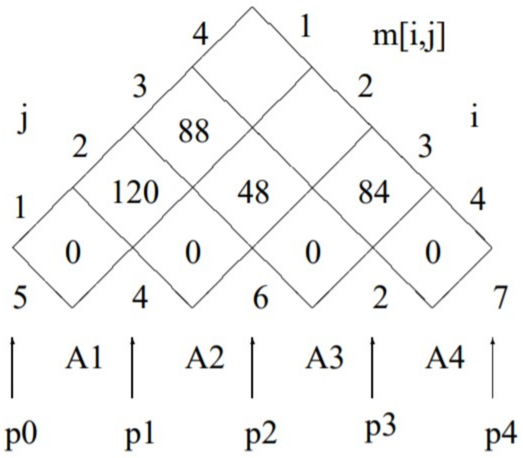
$$m[2,3] = \min_{2\le k<3}(m[2,k]+m[k+1,3]+p_1p_kp_3) = m[2,2]+m[3,3]+p_1p_2p_3 = 48.$$

M[2,4] $=\min_{2<k<4}\{m[2,k]+m[k+1,4]+p_1p_kp_4\}$

$=\min\{(m[2,2]+m[3,4]+p_1p_2p_4)$, $\mathbf{(m[2,3]+m[4,4]+p_1p_3p_4)}\}$

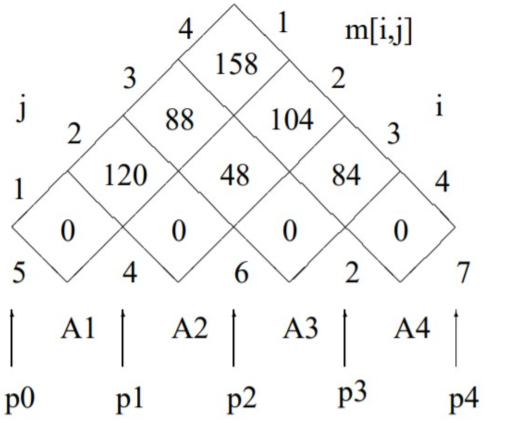
=Min{(254),**(104)**}

=104;



$$m[1,3] = \min_{1\le k<3}(m[1,k]+m[k+1,3]+p_0p_kp_3) = \min\left\{\begin{array}{l} m[1,1]+m[2,3]+p_0p_1p_3 \\ m[1,2]+m[3,3]+p_0p_2p_3 \end{array}\right\} = 88.$$

S[2,4]=2, s[1,3]=**1**, s[1,4]=**3**,



$$m[1,4] = \min_{1\le k<4}(m[1,k]+m[k+1,4]+p_0p_kp_4) = \min\left\{\begin{array}{l} m[1,1]+m[2,4]+p_0p_1p_4 \\ m[1,2]+m[3,4]+p_0p_2p_4 \\ m[1,3]+m[4,4]+p_0p_3p_4 \end{array}\right\} = 158.$$

(A1(A2A3))A4

MCM Example..contd...

- For m[i,j] let s[i,j]=k for optimal splitting (index for which m[i,j] is minimum)
- Lets split matrix chain A_iA_{i+1}.... A_j into (A_i..A_k)*(A_{k+1}..A_j) and compute m[i,k] and m[k+1,j]
  - For m[i,k] let s[i,k]=l for optimal splitting and for m[k+1,j] let s[k+1,j]=m be the value for optimal splitting
  - split matrix chain A_iA_{i+1}.... A_k into (A_i..A_l)*(A_{l+1}..A_k) and $A_{k+1}A_{k+2}$.... A_j into (A_{k+1}..A_m)*(A_{m+1}..A_j)
  - Recurse
- Example
  - Minimum from which m[1,4] is calculated, will give value of corresponding k
  - Record this value as s[1,4]
  - m[1,s[1,4]] OR m[s[1,4]+1,4]
- Parenthesize

Longest common sub-sequence (LCS)

- Given two sequences {X,Y}, finding the longest subsequence common to both. $X=\{x_1,x_2 \ldots x_n\}$ $Y=\{y_1,y_2 \ldots y_m\}$
  - Sub-sequences are different from sub-string

- Complexity NP Hard problem with complexity - 2^(lenth of sequence)

- Let $Z=\{z_1,z_2 \ldots z_k\}$ be the LCS of X and Y
  - If $z_k=x_n= y_{m,}$ then z_{k-1} is LCS of X_{n-1} and Y_{m-1}
  - X_n <> y_m
    - z_k <> $x_{n\ \text{then}}$ then Z_k is LCS of X_{n-1} and Y_m
    - z_k <> $y_{m\ ,}$ then z_k is LCS of X_n and Y_{m-1}

LCS: Formula

Let the two sequences be $X=\{x_1,x_2 \ldots x_i\}$ $Y=\{y_1,y_2 \ldots y_j\}$ where $i \le n$ and $j \le m$

- Then Longest common sub-sequence (LCS) of X and Y is given as
- $LCS(i,j) = 0,$ if $i=0$ OR $j=0$

  $= 1+LCS(i-1,j-1),$ if $x_i=y_j$

  $= \max\{LCS(i,j-1),LCS(i-1,j)\},$ if $x_i \neq y_j$
- To find
  - Length of LCS
  - LCS

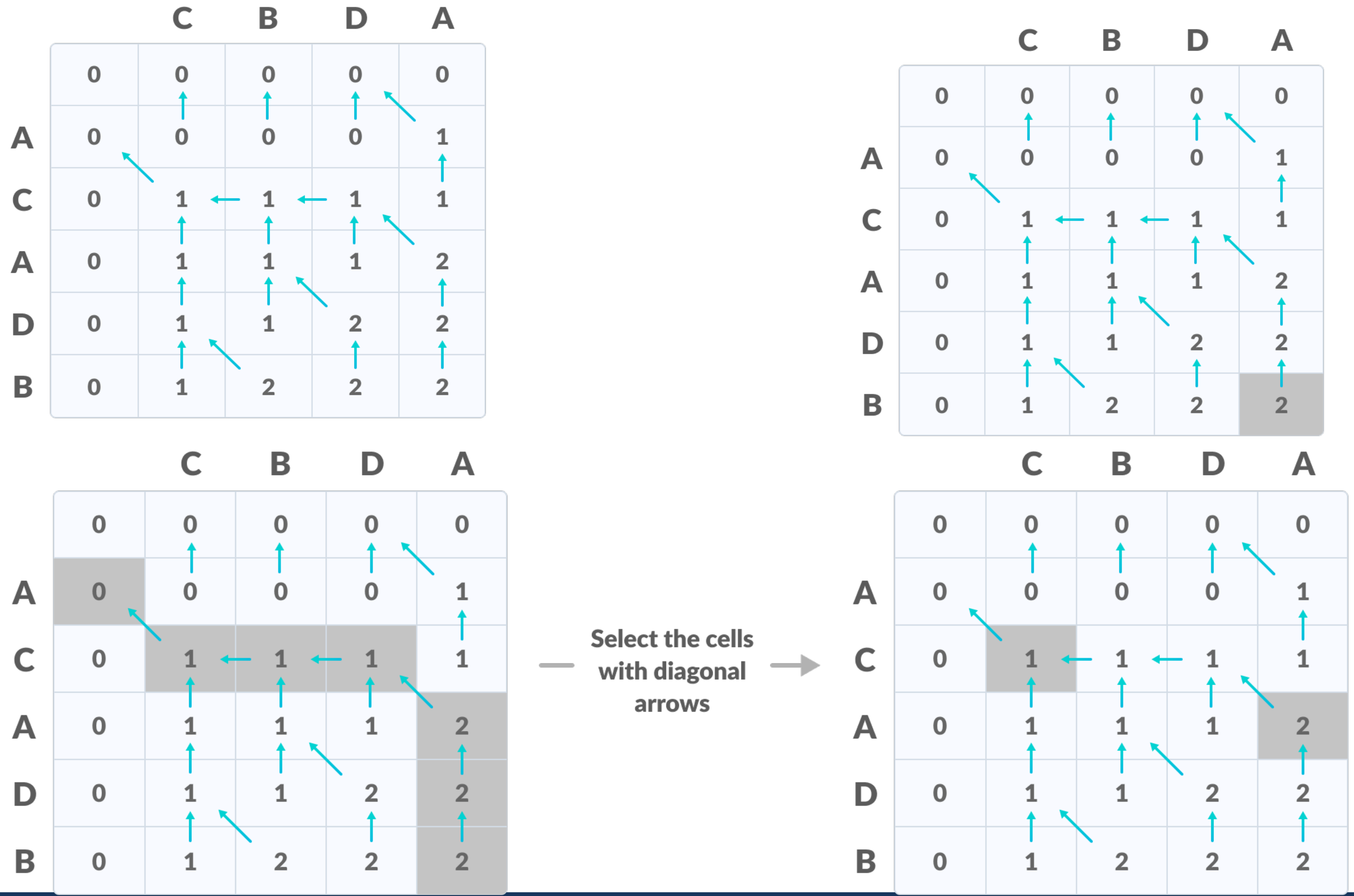
LCS: Example

$X_{i=5}$={A,C,A,D,B}, $Y_{j=4}$={C,B,D,A}

$$
\begin{aligned}
LCS(i,j) &= 0, && \text{if } i=0 \text{ OR } j=0 \\
&= 1+LCS(i-1,j-1), && \text{if } x_i=y_j \\
&= \max\{LCS(i,j-1),LCS(i-1,j)\} && \text{if } x_i \neq y_j
\end{aligned}
$$

	Y_j	y_1=C	y_2=B	y_3=D	y_4=A
X_i	0	0	0	0	0
x_1=A	0	0	0	0	1
x_2=C	0	1	1	1	1
x_3=A	0	1	1	1	2
x_4=D	0	1	1	2	2
x_5=B	0	1	2	2	2

LCS: Example X={A,C,A,D,B}, Y={C,B,D,A}



{C,A}{C,D}
{C,B}

LCS:Homework Example

- X={A,B,C,B,D,A,B}, Y{B,D,C,A,B,A}

- THANK YOU