Algorithms, Design & Analysis

Lecture 19: Dijkstra's Algorithm, Huffman Compression & Floyd-Warshall Algorithm

Abdullah Hussain Yasim & M. Ibrahim Butt

Information Technology University

April 11, 2025

About Your Fellows

- Hi there! We are **Abdullah Hussain Yasim** and **Muhammad Ibrahim Butt**.
- We are Associate Students at ITU.

Dijkstra's Algorithm

(Finding the Shortest Path in a Weighted Graph)

Dijkstra's Algorithm

- **What is it?**
  - Dijkstra's Algorithm is a greedy algorithm used to find the shortest path from a single source node to all other nodes in a weighted graph.
- **Input/Output:**
  - Input: Graph, Start node
  - Output: Shortest paths & distances to all other nodes
- **Characteristics:**
  - It gives better time complexity.
  - It is greedy algorithm.
  - Can work with negative weight edges (but can gives incorrect answer).
  - Fails with negative cycles (Stuck in loop).

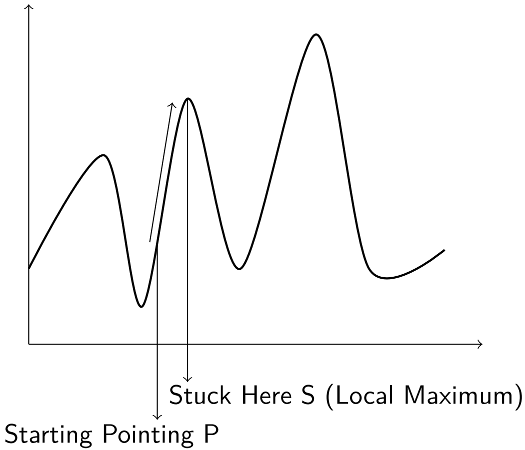
Greedy Algorithms

(Making the best local choice at every step)

Greedy Algorithm

- **Definition:**
  - An algorithm that makes **locally optimal choices** at each step.
- **Characteristics:**
  - Makes locally optimal choices at each step
  - No backtracking
  - Requires **greedy choice property** and **optimal substructure**
- **Examples:**
  - Dijkstra's Algorithm
  - Minimum Spanning Tree (Prim's, Kruskal's)
  - Huffman Compression
- **Note:** May not work for all problems (e.g., 0/1 Knapsack)

Local Maximum Problem



Local Optimization Limitation

- **Process:**
  - Starts at point P, evaluates immediate neighbors (left/right).
  - Always selects **best local move** (like hill-climbing)
  - Gets stuck at point S if no better neighbors exist (local optimum).
- **Why it fails:**
  - No memory of previous states (no backtracking).
  - No knowledge of global landscape (only sees local).
  - Works well only for convex problems.
- **Note:**
  - Local maximum (or local optimum) problems do not occur in Dijkstra's Algorithm and MST algorithms (Prim's, Kruskal's) if all edge weights are positive.

Types of Compression

Lossy vs Lossless Compression

What is Compression?

- **Definition:**
  Process of reducing file size by encoding data more efficiently

- **Two Main Types:**
  - **Lossy Compression**
    (Permanently removes some data)
  - **Lossless Compression**
    (Preserves all original data)

Lossy Compression

- **Definition:**
  Compression where some data is discarded, leading to quality loss
- **Key Points:**
  - Irreversible (original data cannot be perfectly reconstructed)
  - Prioritizes file size reduction over quality preservation
  - Best for human-perceived media (audio/video/images)
- **Examples:**
  - MP3 (Audio)
  - JPEG (Images)
  - MPEG-4 (Video)

Lossless Compression

- **Definition:**
  Compression with no data loss - exact original can be restored
- **Key Points:**
  - Fully reversible compression
  - Maintains data integrity and accuracy
  - Essential for sensitive/critical data
- **Examples:**
  - ZIP archives
  - PNG images
  - Sensor data storage
  - Text documents

Summary

Lossy Compression

- Some data is permanently lost
- Cannot restore original exactly
- Smaller file sizes
- Best for:
  Audio (MP3), Video (MP4), Images (JPEG)

Lossless Compression

- No data loss
- Perfect reconstruction
- Larger file sizes
- Best for:
  Text, Code, Sensor data, Archives (ZIP)

Huffman's Compression

(Lossless Encoding)

What is Huffman Compression?

- A **lossless compression algorithm**
- Reduces file size using **variable-length encoding**
- Based on **frequency of characters**
- **No data is lost**, and original can be perfectly reconstructed

Input/Output

- **Input:**
  - Text file or string
- **Output:**
  - Compressed binary text
  - Huffman Tree or Lookup Table (for decoding)

Key Characteristics

- **Properties:**
  - Lossless (reversible)
  - Uses Priority Queue (Min-Heap)
  - Shorter codes for frequent characters
  - Optimal prefix code (no ambiguity in decoding)
  - Prefix-free code: No code is the prefix of another
  - Only leaf nodes contain characters
  - Each character is initially represented as a singular-node tree, where:
    - The node = character
    - The weight = frequency of the character

Steps of Huffman Algorithm

- **Steps:**
  - Build frequency table of all characters
  - Create nodes for each character
  - Insert nodes into a min-heap (priority queue)
  - Merge two nodes with the lowest frequency. The combined frequency is the sum of the two individual frequencies.
  - Repeat until one tree remains
  - Assign 0 to left, 1 to right during traversal
  - Encode each character using its path in the tree

Huffman Compression: Example

- **Example:**
  - Using the string:

    `"the average cost of each operation in an algorithm when spread"`

Character Frequency Analysis (Part 1)

- Frequency Table:

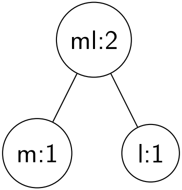
Character	Frequency
a	7
c	2
e	7
f	1
g	2
h	4
i	3
m	1
n	4
o	5

Character Frequency Analysis (Part 2)

Character	Frequency
p	2
r	4
s	2
d	1
t	4
v	1
w	1
spaces	10

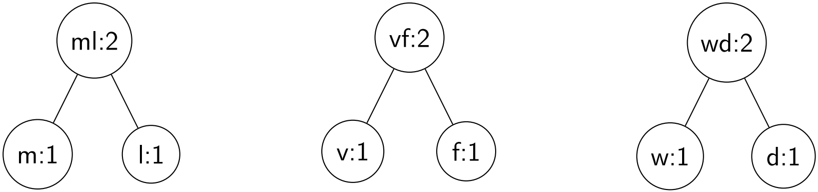
Huffman Tree Example

- Now using the min-heap, merge the singular node tree

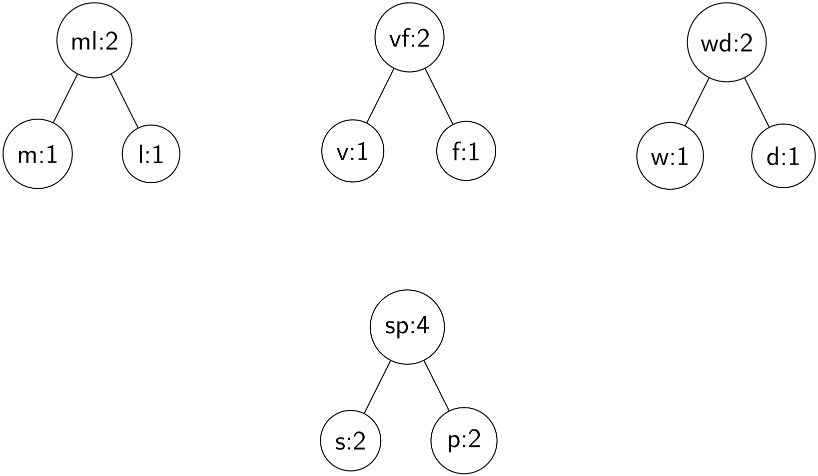


Huffman Tree Example

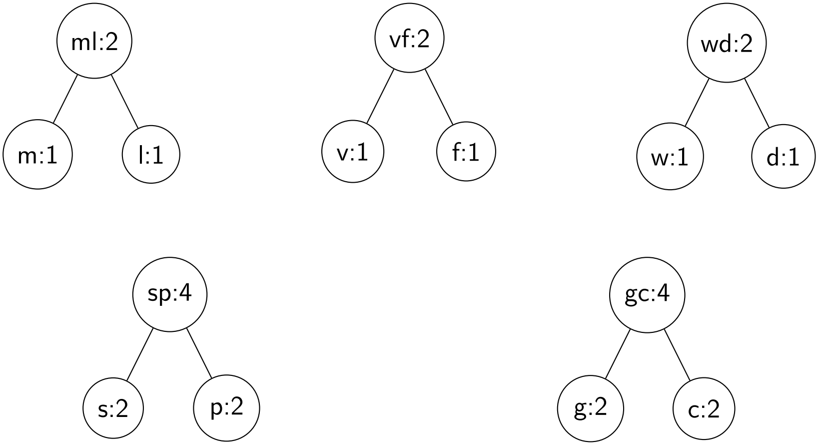
Huffman Tree Example



Huffman Tree Example



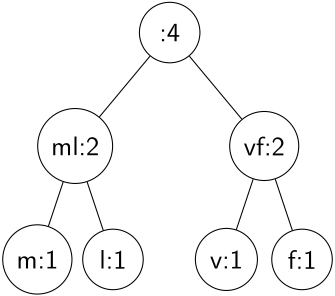
Huffman Tree Example



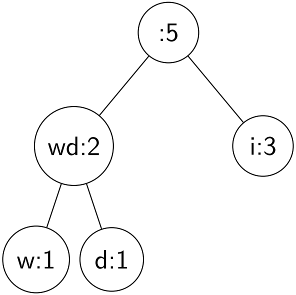
Character Merge Tracking Table

Character/Node	Frequency	Used in Merge
m	1	✓
l	1	✓
v	1	✓
f	1	✓
w	1	✓
d	1	✓
s	2	✓
p	2	✓
g	2	✓
c	2	✓

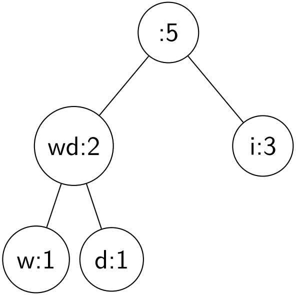
Huffman Tree Example

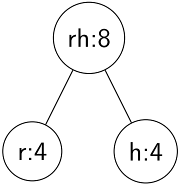


Huffman Tree Example

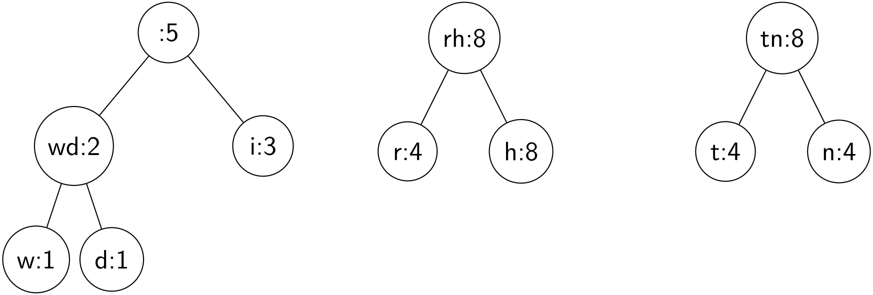


Huffman Tree Example

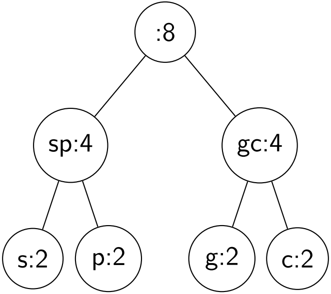




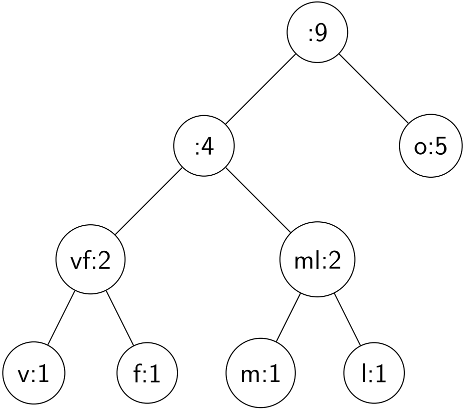
Huffman Tree Example



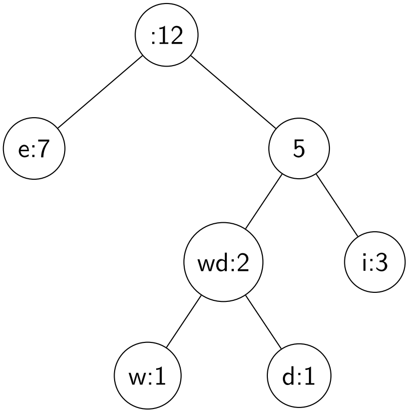
Huffman Tree Example



Huffman Tree Example



Huffman Tree Example



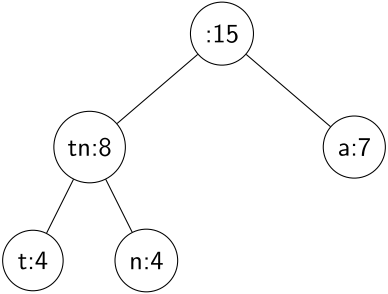
Character Merge Tracking Table

Character/Node	Frequency	Used in Merge
m	1	✓
l	1	✓
v	1	✓
f	1	✓
w	1	✓
d	1	✓
s	2	✓
p	2	✓
g	2	✓
c	2	✓

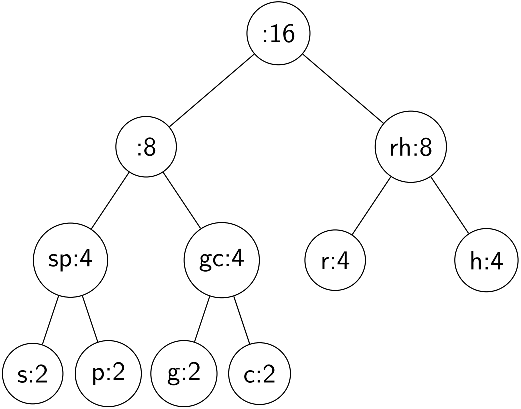
Character Merge Tracking Table

Character/Node	Frequency	Used in Merge
i	3	✓
r	4	✓
h	4	✓
t	4	✓
n	4	✓
o	5	✓
e	7	✓

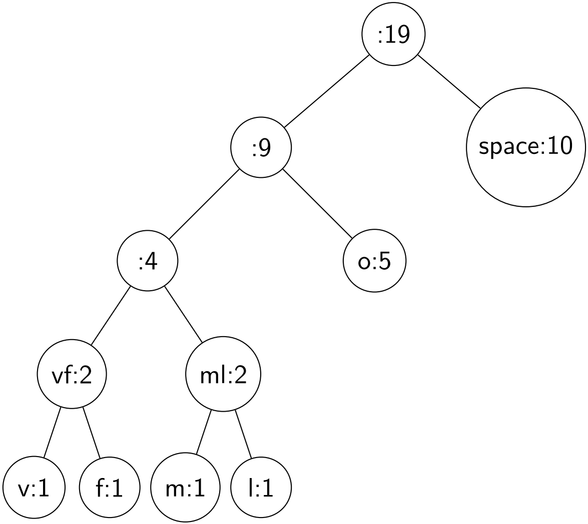
Huffman Tree Combination Example



Huffman Tree Combination Example



Huffman Tree Combination Example



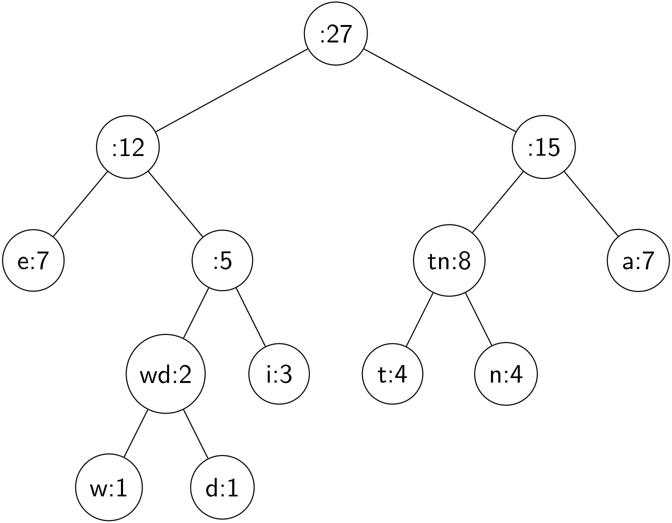
Character Merge Tracking Table

Character/Node	Frequency	Used in Merge
m	1	✓
l	1	✓
v	1	✓
f	1	✓
w	1	✓
d	1	✓
s	2	✓
p	2	✓
g	2	✓
c	2	✓

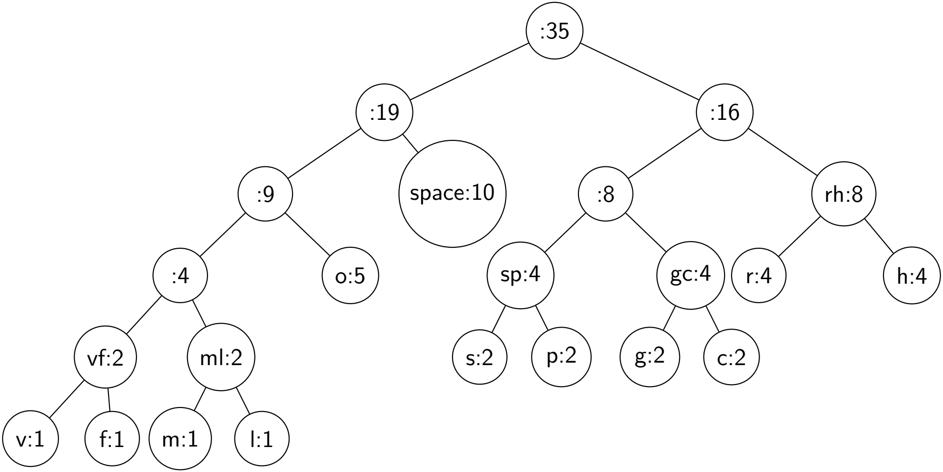
Character Merge Tracking Table

Character/Node	Frequency	Used in Merge
i	3	✓
r	4	✓
h	4	✓
t	4	✓
n	4	✓
o	5	✓
e	7	✓
a	7	✓
space	10	✓

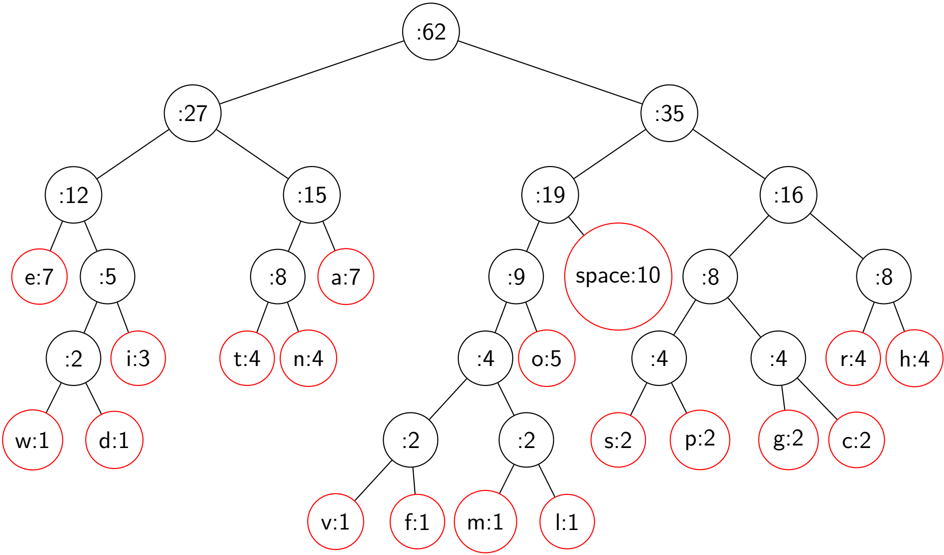
Huffman Tree Combination Example



Huffman Tree Combination Example



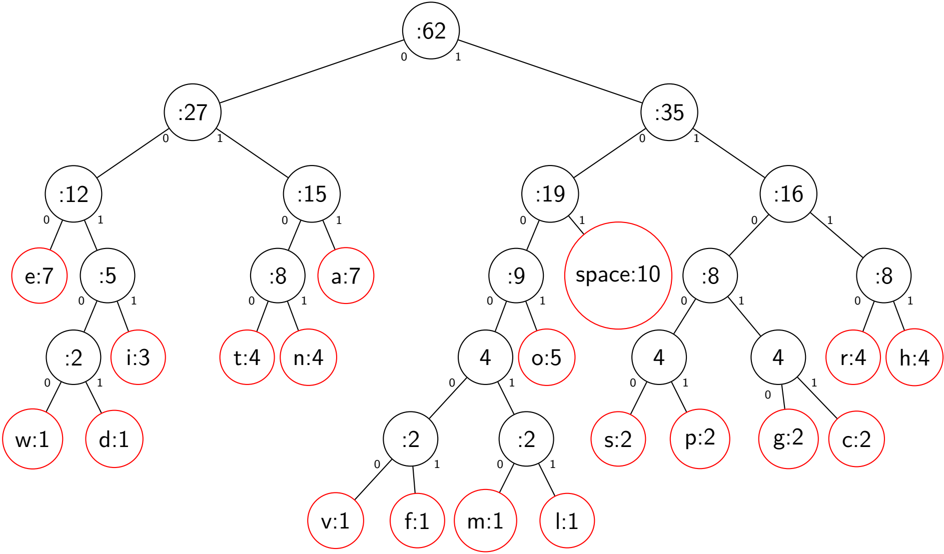
Final Huffman Tree



Huffman Tree

- Assigning 1 to the right traversal of each node
- Assigning 0 to the left traversal of each node
- We will traverse through the tree and set the traversal binary code as the encoding of the character.
- Doing so will assign smaller binary codes to more frequently occuring characters and larger binary codes to less frequent characters.
- For example The binary code for **a** will be 011 and for **v** will be 100000

Final Huffman Tree



Huffman Encoding Table

Character/Node	Frequency	Huffman Code
m	1	100010
l	1	100011
v	1	100000
f	1	100001
w	1	00100
d	1	00101
s	2	11000
p	2	11001
g	2	11010
c	2	11011

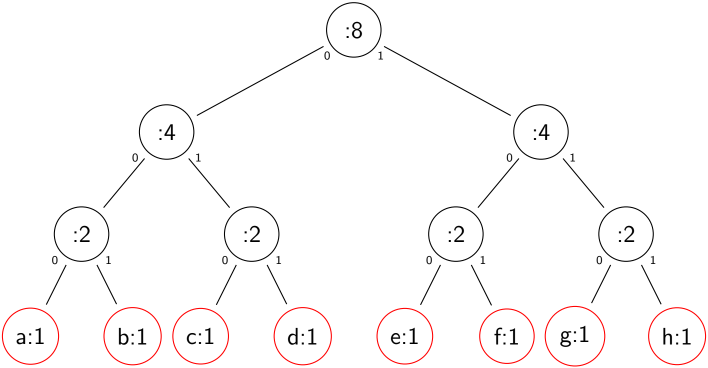
Huffman Encoding Table

Character/Node	Frequency	Huffman Code
i	3	0011
r	4	1110
h	4	1111
t	4	0100
n	4	0101
o	5	1001
e	7	000
a	7	011
space	10	101

Question

Q. What if the frequency of all characters is same in the string during Huffman's compression?

Huffman Tree for a-h (Equal Frequencies)



Equal Frequency Huffmans Compression

- As we can see if the frequency of the characters is equal then there are no longer or shorter codes
- All encodings are of the same length, a = 000 , f = 101 and etc
- Thus it is not optimal to use huffmans compression for randomly generated strings or when frequency of characters is same because it provides negligible compression

Huffman Encoding Table

Character/Node	Frequency	Huffman Code
a	1	000
b	1	001
c	1	010
d	1	011
e	1	100
f	1	101
g	1	110
h	1	111

Question

Q. What if the frequency of the characters is an arithmetic progression during Huffman's compression?

Floyd-Warshall Algorithm

(Dynamic Programming)

Dynamic Programming

- A method for solving complex problems by breaking them down into simpler subproblems.
- Stores the results of subproblems to avoid redundant computations (memoization or tabulation).
- Widely used in algorithms like:
  - Fibonacci Number Calculation
  - Matrix Chain Multiplication
  - Longest Common Subsequence
  - Floyd-Warshall Algorithm

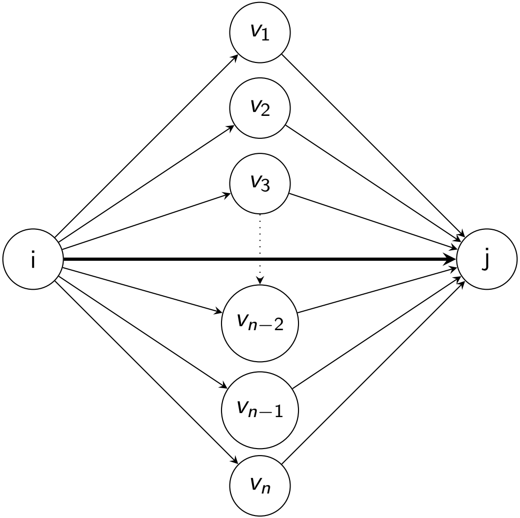
Floyd-Warshall Algorithm

- The Floyd-Warshall algorithm is used to find the shortest paths between all pairs of vertices in a weighted graph.
- It uses a dynamic programming approach by progressively improving an estimate of the shortest path between two vertices.
- The core idea is to check whether a path from i to j through an intermediate vertex k is shorter than the previously known shortest path.

Floyd-Warshall Algorithm

- It can be used to find the shortest path between two nodes in a graph which has negative weight edges
- **Input:** Graph G
- **Output:** Shortest path between all nodes.
- **Time Complexity:** The algorithm runs in $\mathcal{O}(n^3)$ time

Floyd-Warshall Algorithm



Floyd-Warshall Algorithm

- As we can see there are total n-1 paths from i to j
- d(i$\rightarrow$j)
- d(i$\rightarrow v_k$) $+$ d($v_k\rightarrow$j)
- The shortest possible path between i and j will be the one with the one with the minimum distance from these paths

Floyd-Warshall Algorithm (Dynamic Programming)

- d(i,j,n) will provide the shortest possible path between nodes i and j (where n is total number of vertices in the graph).
- Let d(i,j,k) be the shortest path between nodes i and j which allows traversal of first K nodes within the graph.
- We can determine d(i,j,k+1) from d(i,j,k) such as,
- $d(i,j,k+1) = \text{MIN}\ (d(i,j,k),\ d(i,k+1,k) + d(k+1,j,k))$

HomeWork

Q. Implement a dynamic array in C++ and analyze the amortized cost of its insertions.