DIJKSTRA's ALGORITHM

→ Dijkstra's algorithm is used for solving single-source shortest path problem.

→ The single source shortest path problem finds the shortest paths to all its other vertices for a given vertex called source in a weighted connected graph

→ The single - source shortest path problem asks for a family of paths each leading from the source to a different vertex in the graph

→ Each vertex has two labels.

* 'd' - a numeric value which indicates the length of the shortest path from the source to this vertex.
* other label indicates the name of the next-to-last vertex on such a path.

→ Find the next nearest vertex u^* which is a fringe vertex with the smallest d value.

→ After identifying u^*, move u^* from the fringe to the set of tree vertices.

→ for each remaining fringe vertex u that is connected to u^* by an edge of weight $w(u^*, u)$ such that $d_{u^*} + w(u^*, u) \leq d_u$, update d_u.

→ ALGORITHM Dijkstra(G, s)
// Dijkstra's Alg. for single-source shortest paths
// Input: A weighted connected graph $G = \langle V, E \rangle$
// Output: The length d_v of the shortest path from s to v & its penultimate vertex p_v for every $v \in V$

Initialize (Q)
for every vertex v in V do
  $d_v \leftarrow \infty$; $p_v \leftarrow$ null
Insert (Q, v, p_v)
$d_s \leftarrow 0$; Decrease (Q, s, d_s)
$V_T \leftarrow \phi$

for $i \leftarrow 0$ to $|V|-1$ do

  $u^* \leftarrow$ DeleteMin (Q)

  $V_T \leftarrow V_T \cup \{u^*\}$

  for every vertex u in $V - V_T$ that is adjacent to u^* do

    if $d_{u^*} + w(u^*, u) < d_u$

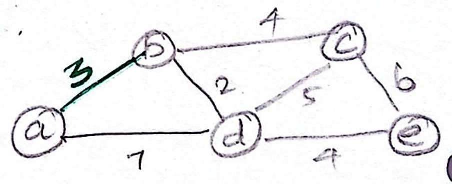
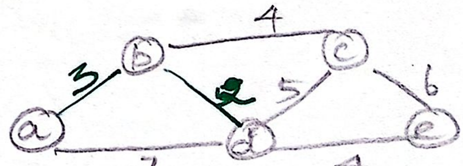
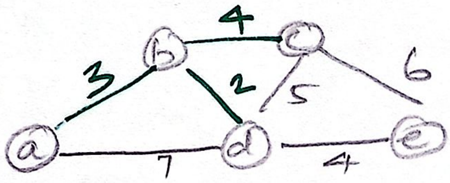
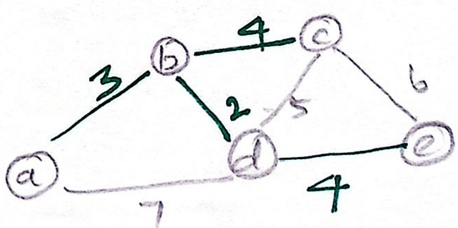
      $d_u \leftarrow d_{u^*} + w(u^*, u)$; $p_u \leftarrow u^*$

      Decrease (Q, u, d_u)

→ Efficiency when graph is implemented by weight matrix & PQ as an unordered array is $\Theta(|V|^2)$

→ Graph represented by adjacency lists & PQ using min heaps $O(|E| \log |V|)$.

→ Can still be improved if PQ is implemented by fibonacci heap.

Tree Vertices	Remaining Vertices	Illustration
a (-, 0)	b(a, 3) c(-, ∞) d(a, 7) e(-, ∞)	
b(a, 3)	c(b, 3+4), d(b, 3+2) e(-, ∞)	
d(b, 5)	c(b, 7), e(d, 5+4)	
c (b, 7)	e(d, 9)	
e (d, 9)		

The shortest paths and their lengths are

from a to b : a – b of length 3

from a to d : a-b-d of length 5

from a to c : a-b-c of length 7

from a to e : a-b-d-e of length 9

HUFFMAN TREES

- Two types of Encoding
  * Fixed-length Encoding : assigns to each character a bit string, of the same length. Sequence of characters assigned to each character is called codeword
  * Variable-length Encoding : assigns codewords of variable length to each word. Frequently used characters will be assigned smaller codewords while longer codewords will be assigned to less frequent characters. eg. Morse Code.

- The problem in variable length encoding is we can't tell how many bits of an encoded text represent the first character?

- The above problem can be overcome by using prefix-free codes / prefix codes which does not allow prefix of codeword same as another codeword of another character.

- So scan the bit-string until we got the first group of bits that is a codeword of some character, replace these bits by this character & repeat this operation until the bit-string's end is reached.

* This can be done by using a tree structure invented by David Huffman

* **Huffman's Algorithm**

Step 1: Initialize n one-node trees & label them with the characters of the alphabet. Record the frequency of each character in its tree's root to indicate tree's weight.

Step 2: Find two trees with smallest weight. Make them the left & right subtree of a new tree & record the sum of their weights in the root of the new tree as its weight.

Step 3: Repeat step 2 until a single tree is obtained.

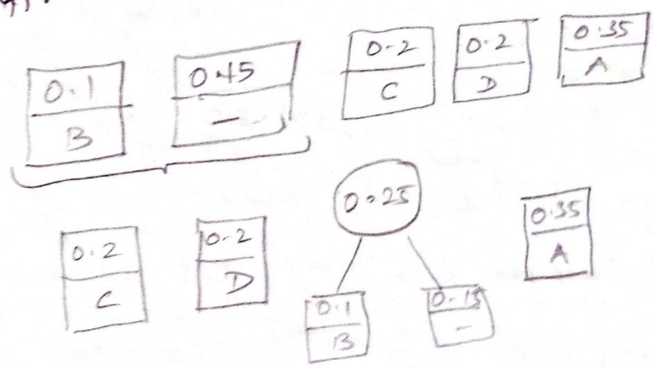
* The tree constructed like this is called Huffman tree.

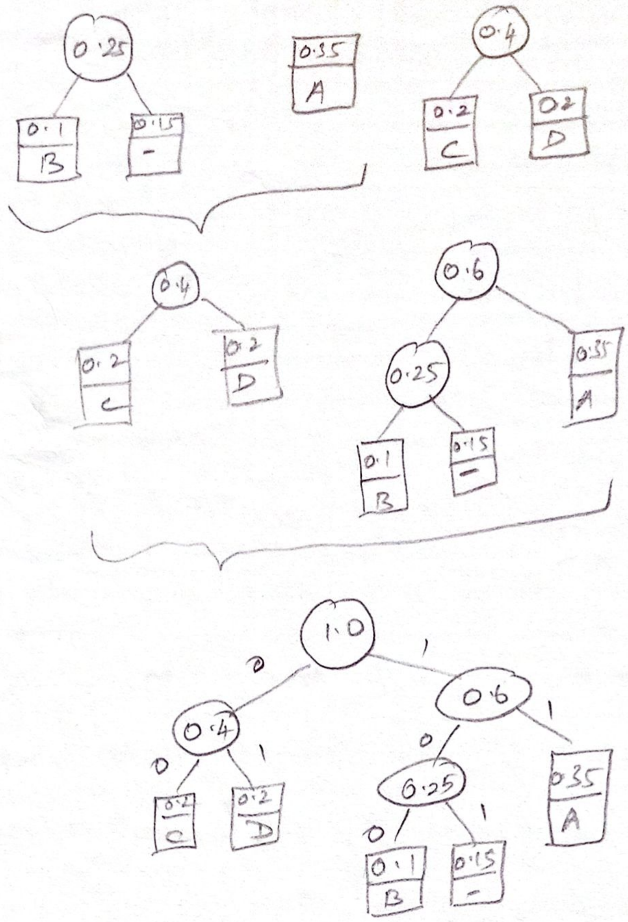
Example:

Consider five-character alphabets {A, B, C, D, -} with the following probabilities

Character	A	B	C	D	-
probability	0.35	0.1	0.2	0.2	0.15

* Huffman coding tree





Resulting Codeword

char	A	B	C	D	-
prob.	0.35	0.1	0.2	0.2	0.15
Code word	11	100	00	01	101

- DAD is encoded as 011101
- $\underbrace{100}_{B}\underbrace{11}_{A}\underbrace{01}_{D}\underbrace{101}_{-}\underbrace{11}_{A}\underbrace{01}_{D}$ is decoded as BAD-AD
- Expected number of bits per character in this code is

  $2 * 0.35 + 3 * 0.1 + 2 * 0.2 + 2 * 0.2 + 3 * 0.15 = 2.25$
- Effectiveness of a compression algorithm is its compression ratio

  For fixed length Encoding of 3 bits 3

  For variable " " 2.25

  $\therefore \frac{3 - 2.25}{3} * 100\% = 25\%$

  i.e. 25% less memory is needed for decoding using Huffman's algorithm.

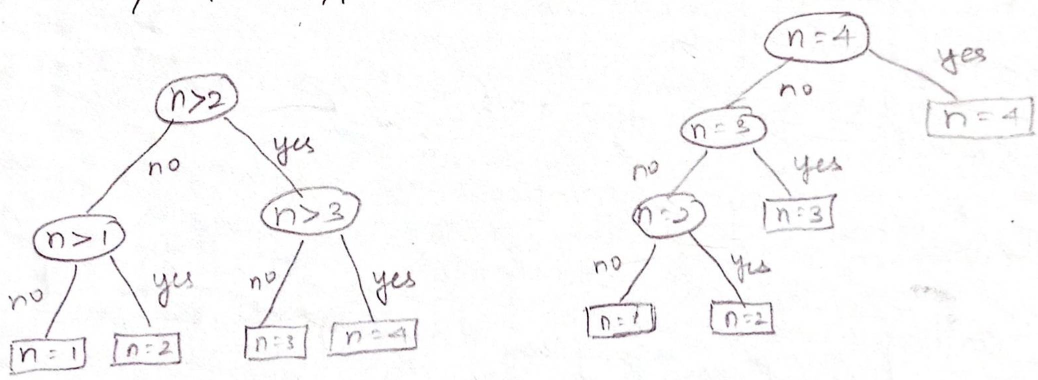
* Huffman's encoding is one of the most important file compression methods.

* Another version dynamic Huffman encoding updates the coding tree each time a new character is read from the source text.

* Huffman's algorithm is also used to construct a binary tree with minimum weighted path length

* Weighted path length of a tree is $\sum_{i=1}^{n} l_i w_i$, where l_i is the length of the simple path from the root to the i^{th} leaf & w_i ~~is~~ the positive number assigned to each leaf.

* Decision Trees a game for guessing an integer between 1 and 4 is also works on same Concept - Huffman trees



(1) (2)

if $P_1 = 0.1$, $P_2 = 0.2$, $P_3 = 0.3$ & $P_4 = 0.4$ then the second tree will give minimum weighted path tree.