Greedy Technique: Given **n** inputs, obtain a subset that satisfy some constraints.

- **feasible solution:** any subset that satisfies all the constraints.
- **optimal solution:** feasible solution which maximizes (or minimizes a) given objective function.

there is usually an obvious way to determine a feasible solution, but not necessarily an optimal solution.

- The greedy method suggests that one can device an algorithm which works in stages ***taking one input at a time***.
- At any stage, a decision is made whether or not a particular input is in optimal solution. This is done by considering the input in an order determined by some **selection procedure**.
- The selection procedure itself is based on some optimization measure.
- Several different optimization measures may be ***plausible*** for a given problem.

```
Global solution: set of items;
procedure Greedy (A:arraytype; n:integer);
{A[1..n] contains n inputs, output available in global "solution"}
var x: item;
begin
        solution ← ∅;  {initialize the solution to be empty}
        for i← 1 to n do
        begin
                x ← SELECT(A);
                if FEASIBLE(solution, x) then
                solution ← UNION(solution, x);
        end {required solution is now available in solution}
end.
```

- The function <u>SELECT</u> selects an input from A, removes it and assign its value to x.

- <u>FEASIBLE</u> is a boolean values function which determines if x can be included in the solution vector.

- <u>UNION</u> actually combines x with solution and updates the objective function.

Knapsack problem

Given **n** objects and a knapsack. Object **i** has weight $\mathbf{w}_i$ and knapsack has capacity **M**.

If a fraction x_i, $0 \le x_i \le 1$, of object i is placed into the knapsack, then a profit of $p_i\, x_i$ is earned.

Problem may be stated as-

Maximize $\sum p_i\, x_i, \quad 1 \le i \le n$ ....(1)

Subject to $\sum w_i\, x_i \le M, \quad 1 \le i \le n$ ....(2)

profits and weights are positive numbers

$0 \leq x_i \leq 1$, for $1 \leq i \leq n$ ....(3)

A ***feasible*** solution is any set (x_1, x_2, ..., x_n) satisfying (2) and (3).

An ***optimal*** solution is a feasible for which (1) is maximum.

Example: Consider the following instance of the knapsack problem:

n=3, M=20, (P_1, P_2, P_3) = (25, 24, 15), (w_1, w_2, w_3) = (18, 15, 10).

four feasible solutions are:

Sr.No	(x_1, x_2, x_3)	$\sum w_i x_i$	$\sum p_i x_i$
1	(1/2, 1/3, 1/4)	16.5	24.25
2	(1, 2/15, 0)	20	28.2
3	(0, 2/3, 1)	20	31
Maximum profit per unit of capacity used			

```
Global real P(1:n), W(1:n), X(1:n), M, cu;
procedure GREEDY_KNAPSACK(P, W, X, M, n)
// n objects ordered so that P(i)/W(i) ≥ P(i+1)/W(i+1). M is the
knapsack size and X(1:n) is the solution vector.//
var n: integer;
begin
        X ← 0;  //initialize solution to zero//
        cu ← M;    //cu = remaining knapsack capacity//
        for i ← 1 to n do
        begin
                if W(i) > cu then exit
                X(i) ← 1; cu ← cu - W(i);
        end
                if (i ≤ n) then X(i) ← cu/W(i);
end.
```

Huffman code:

- What is Morse code
- How characters are represented in the memory of the computer
- Which is a fixed length code

A byte is enough to store any number, there are 256 possible codes, which is more than enough!

We need less than that.

Assume the existence of the five letters ‘a’, ‘b’, ‘c’, ‘d’, ‘e’.
Three bits give us 8 possible different sequences.

letter	code
a	000
b	001
c	010
d	011
e	100

Average code length = 3

Possibility 1:

Suppose that we know the percentage of occurrence of a character

'a' = 35%, 'b' = 20%, 'c' = 20%, 'd' = 15%, 'e' = 10%, then we can define a variable length code.

letter	frequency	code
a	.35	0
b	.20	1
c	.20	00
d	.15	01
e	.10	10

Average code length = 1(.35)+1(.20)+2(.20)+2(.15)+1(.10) = 1.35, half that we did with last code. Unfortunately it does not work. What is 0010

aaba, aae, cba or ce?

Possibility 2:

Each row begins with 11 and 11 will not be used other than to coding space.

letter	frequency	code
a	.35	110
b	.20	111
c	.20	1100
d	.15	1101
e	.10	1110

Average code length = 3.35

Possibility 3:

Code has a prefix property. "**No code be prefix to another**".

letter	frequency	code
a	.35	00
b	.20	10
c	.20	010
d	.15	011
e	.10	111

Average code length = 2.45

On these discussions *D.A. Huffman* proposed minimal length prefix codes. We begin with forest of trees, correspondence to single character in the alphabet, weight associated with it.

procedure HUFFMAN_CODE

1. **repeat**

   i. find two trees of smallest weight, merge them together by making them the left and right children of a new root.

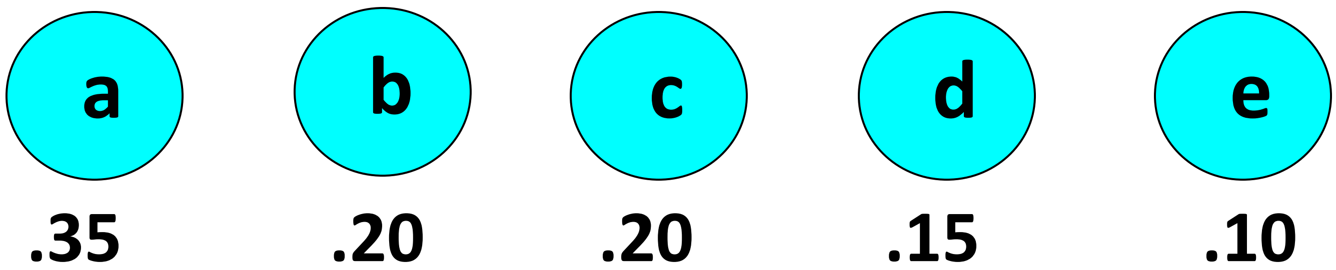
   ii. make the weight of the new tree equals to the the sum of the weights of two subtrees

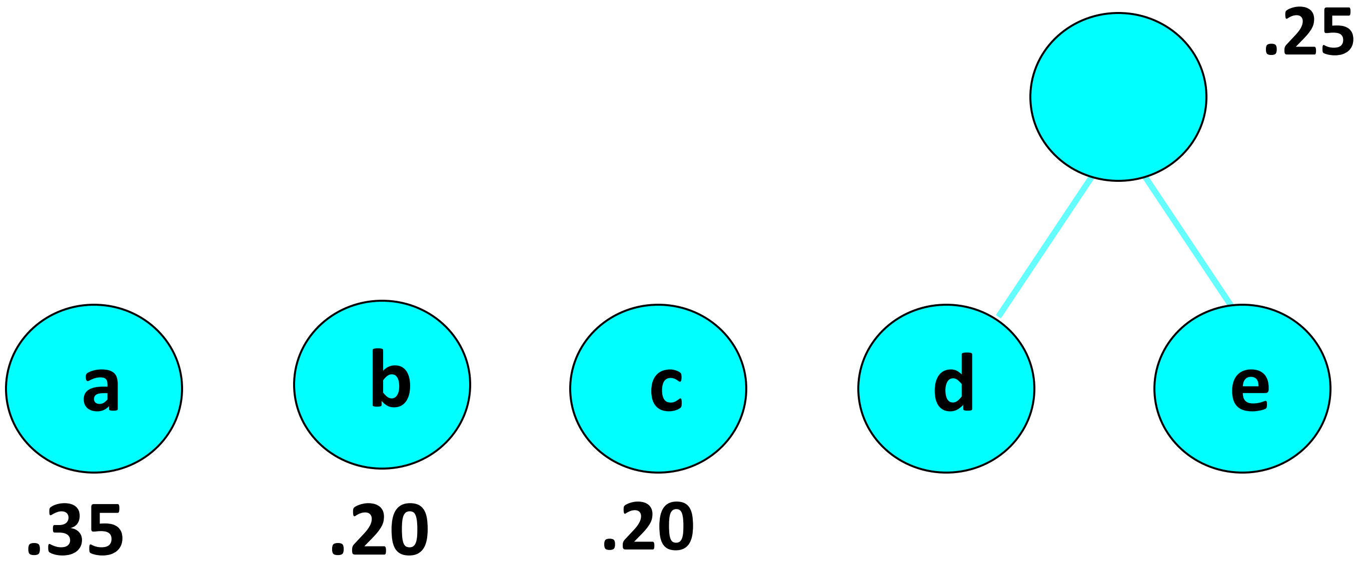
   until (the forest consists of a single tree)

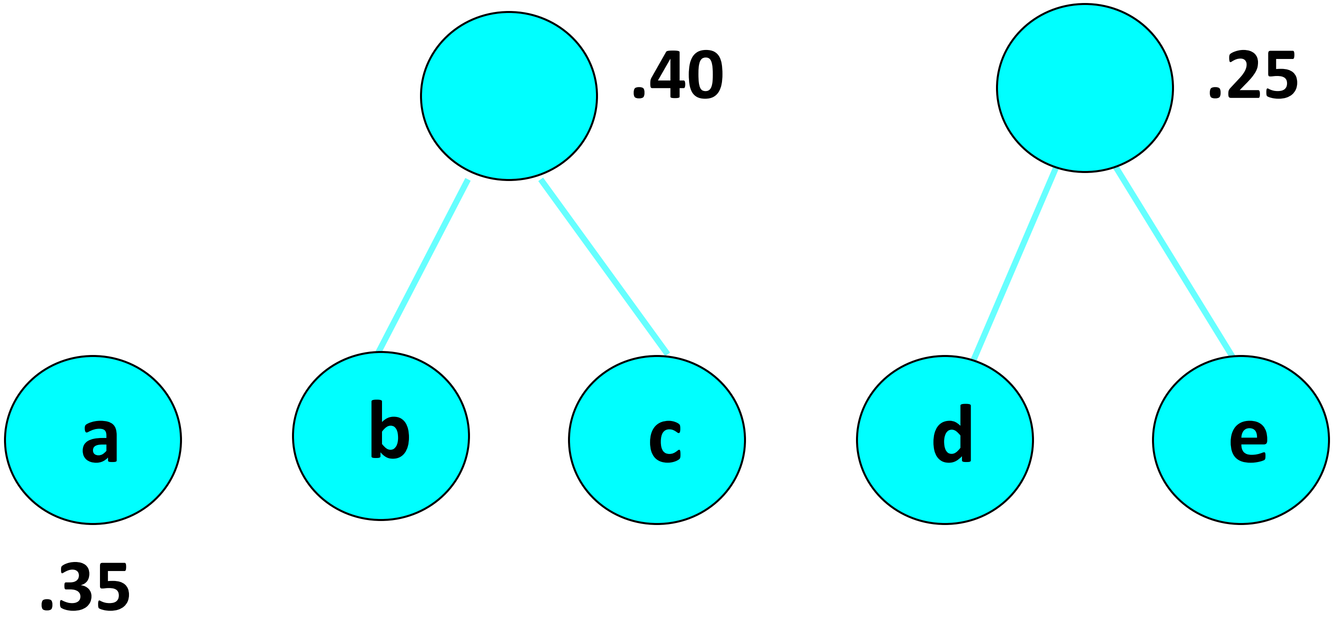
2. assign to each left edge a value 0 and each right edge the value 1.

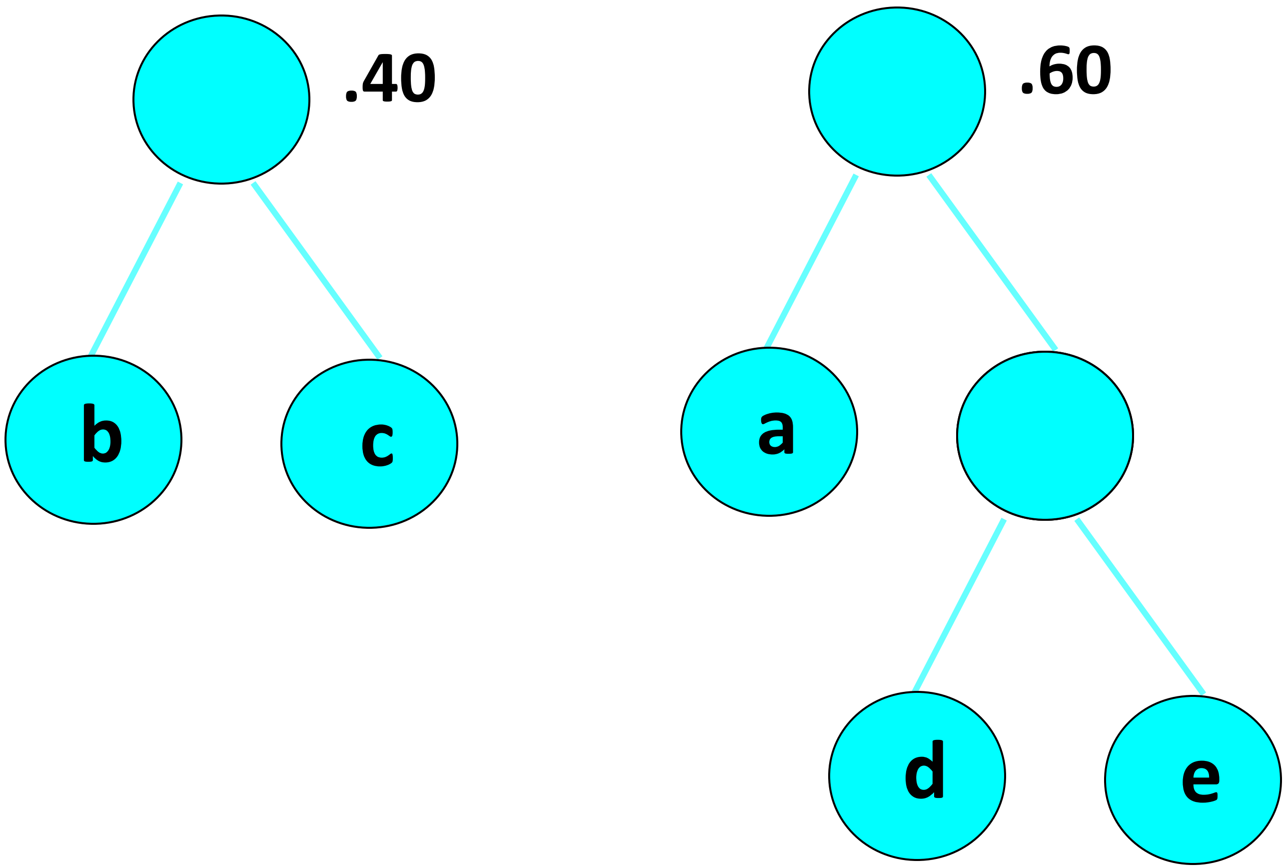
3. read the codes for the letters by tracking down the tree.

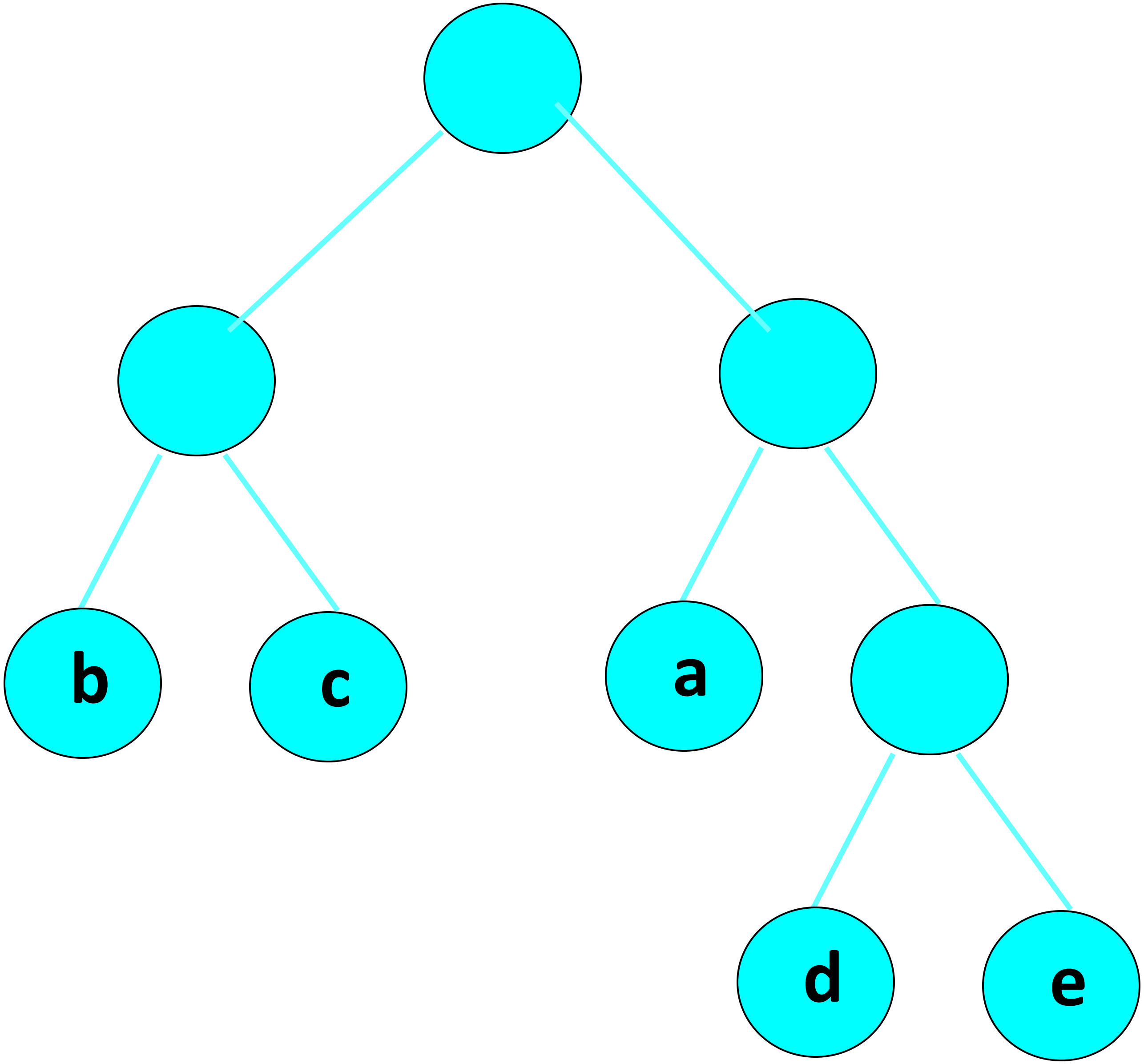
letter	frequency	code
a	.35	10
b	.20	00
c	.20	01
d	.15	110
e	.10	111

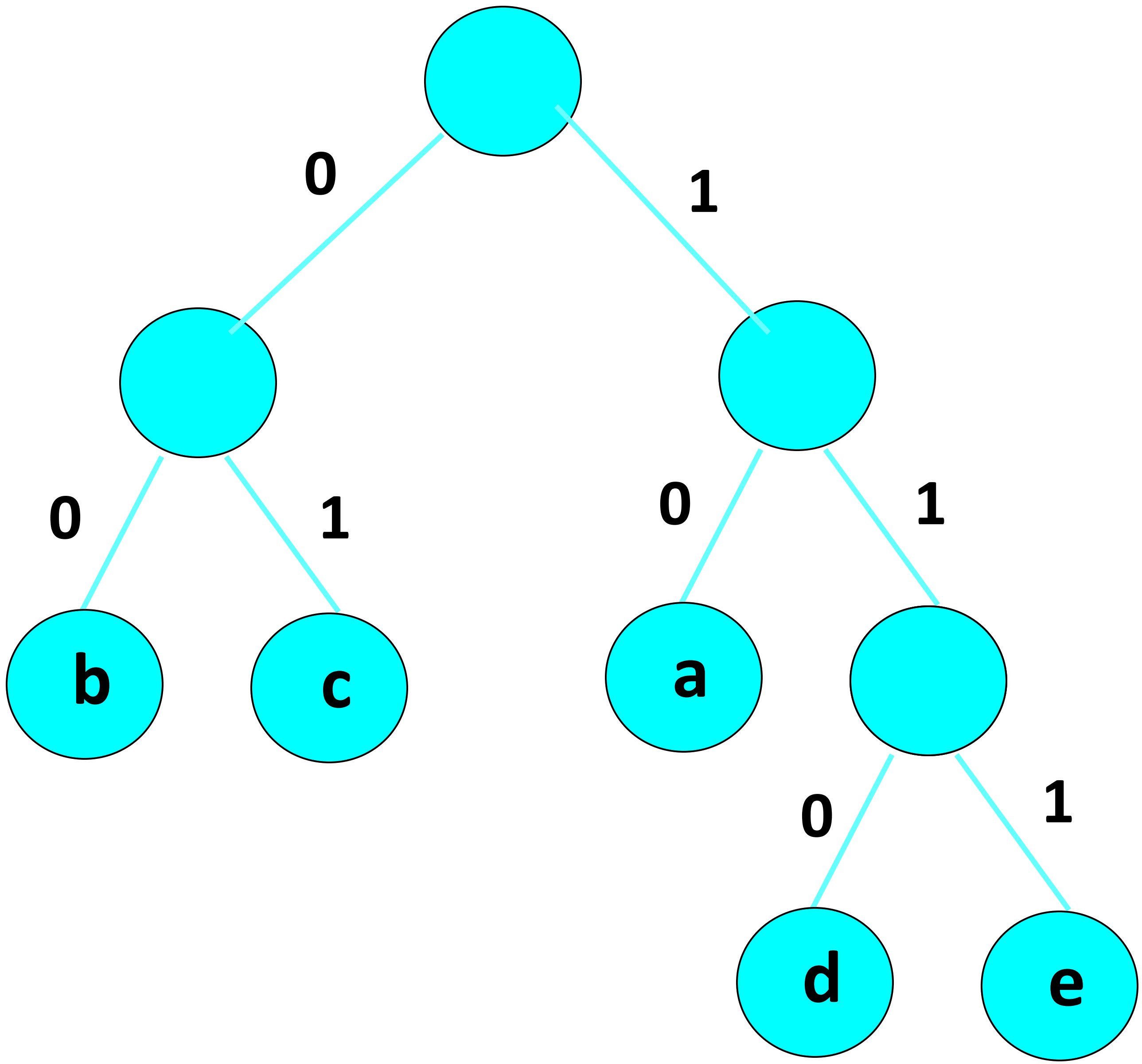

a
b
c
d
e
.35
.20
.20
.15
.10


.25
a
b
c
d
e
.35
.20
.20


.40
.25
a
b
c
d
e
.35


.40
b
c
.60
a
d
e


b
c
a
d
e



Average Code length = 2.25

Applications of Huffman Coding (few examples)

1. File Compression
    - Examples:
        - ZIP / GZIP (part of DEFLATE algorithm uses Huffman coding).
        - 7z, rar and other archive formats.
2. Image Compression
    - JPEG standard: after quantization and zig-zag scanning, Huffman coding is used to encode frequency coefficients efficiently.
3. Audio Compression
    - MP3 and AAC audio formats use Huffman coding for the entropy coding stage.
4. Video Compression
    - MPEG, H.264, HEVC (H.265) — Huffman coding (or variants like CABAC/CAVLC, which evolved from it) is used for entropy coding of video coefficients

SINGLE SOURCE SHORTEST PATH PROBLEM

In general shortest path problems are concerned with finding paths between vertices.

Some characteristics which may help in finding the exact problem is-

1. Graph is finite or infinite
2. Graph is directed or undirected

3. Edges are all of length 1, or all lengths are non negative, or negative lengths are allowed.

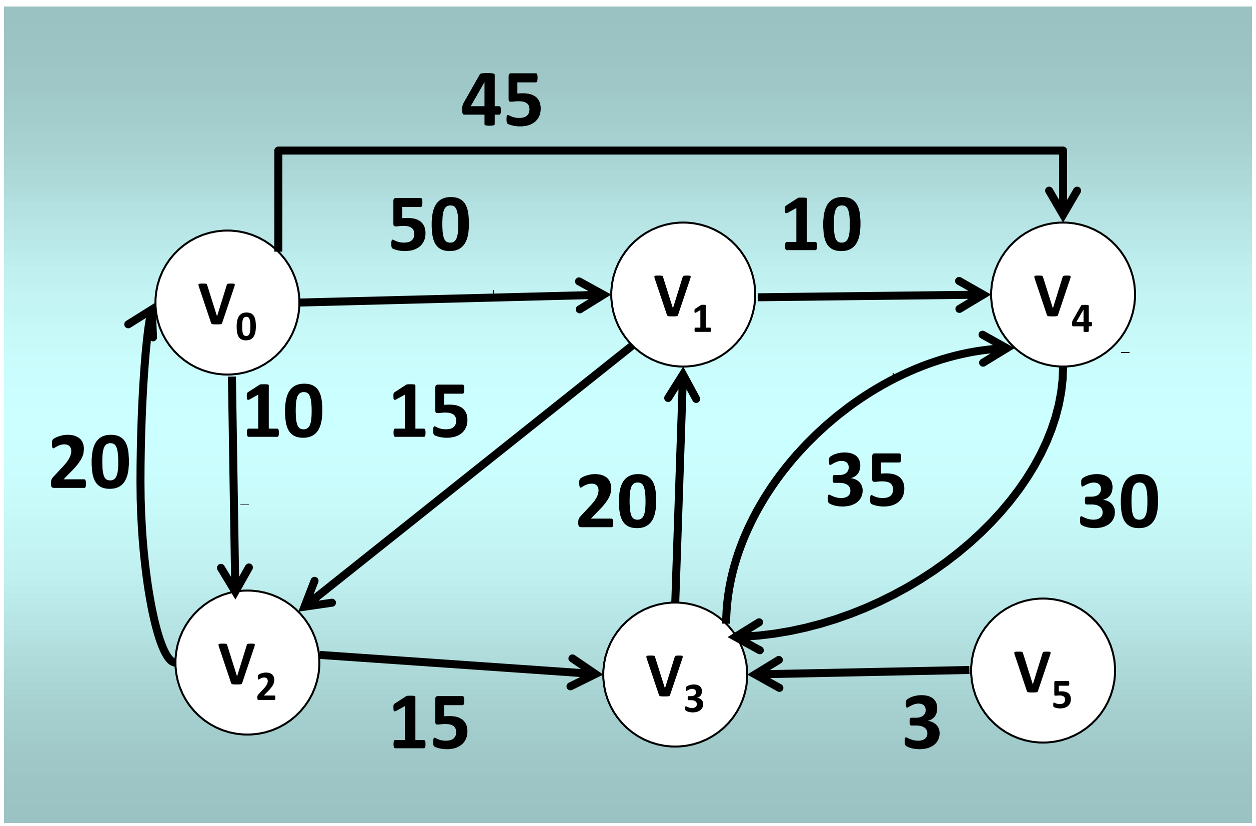
4. We may be interested in shortest path from a given vertex to another, or, from a given vertex to all other vertices, or, from each vertex to all the other vertices.

Is there a path from A to B?

Is there any other path from A to B which is shortest path?

length of the path = sum of weights of the edges on that path.

- We assume that starting vertex is the source and the last vertex is the destination, directed graph G = (V, E) a weighting function c(e) for the edges of G and a source vertex v_0.



Path	Length
$V_0 V_2$	10
$V_0 V_2 V_3$	25
$V_0 V_2 V_3 V_1$	45
$V_0 V_4$	45

In order to generate the shortest path we need to be able to determine

1. The next vertex to which the shortest path must be generated.

2. Shortest path to this vertex.

Let **S** denotes the set of all vertices (including v_0 to which the path has already been generated).

for **w** not in **S**, let **DIST(w)** be the length of the shortest path starting from v_0 going through only those vertices which are in **S** and ending at **w**, we observe-

1. If next shortest path is to **u**, then path begins at **v_0** end to **u**, going through only those vertices which are in **S**.

   *Comment: {Assume that there is a vertex **w** not in **S**, path **v_0** to **u** contains a path from **v_0** to **w** which is of length less than the **v_0** to **u** path, which contradicts the existence of such **w**.}*

2. The destination of the next path generated must be that vertex **u** which has the minimum distance, **DIST(u)**, among all vertices not in **S**.

3. Having selected such **u** from (2) above, and generated the shortest path $\mathbf{v_0}$ to **u**, **u** becomes the member of **S**. length of this path is DIST(u) + c(u,w).

The algorithms is known as Dijkstra's Algorithm

3. Having selected such **u** from (2) above, and generated the shortest path $\mathbf{v_0}$ to **u**, **u** becomes the member of **S**. length of this path is DIST(u) + c(u,w).

Procedure SHORTEST-PATHS(v, COST, DIST, n)

//DIST(j), $1 \le j \le n$ is set to the length of the Shortest path from v to j.//

//DIST(v) is set to 0//

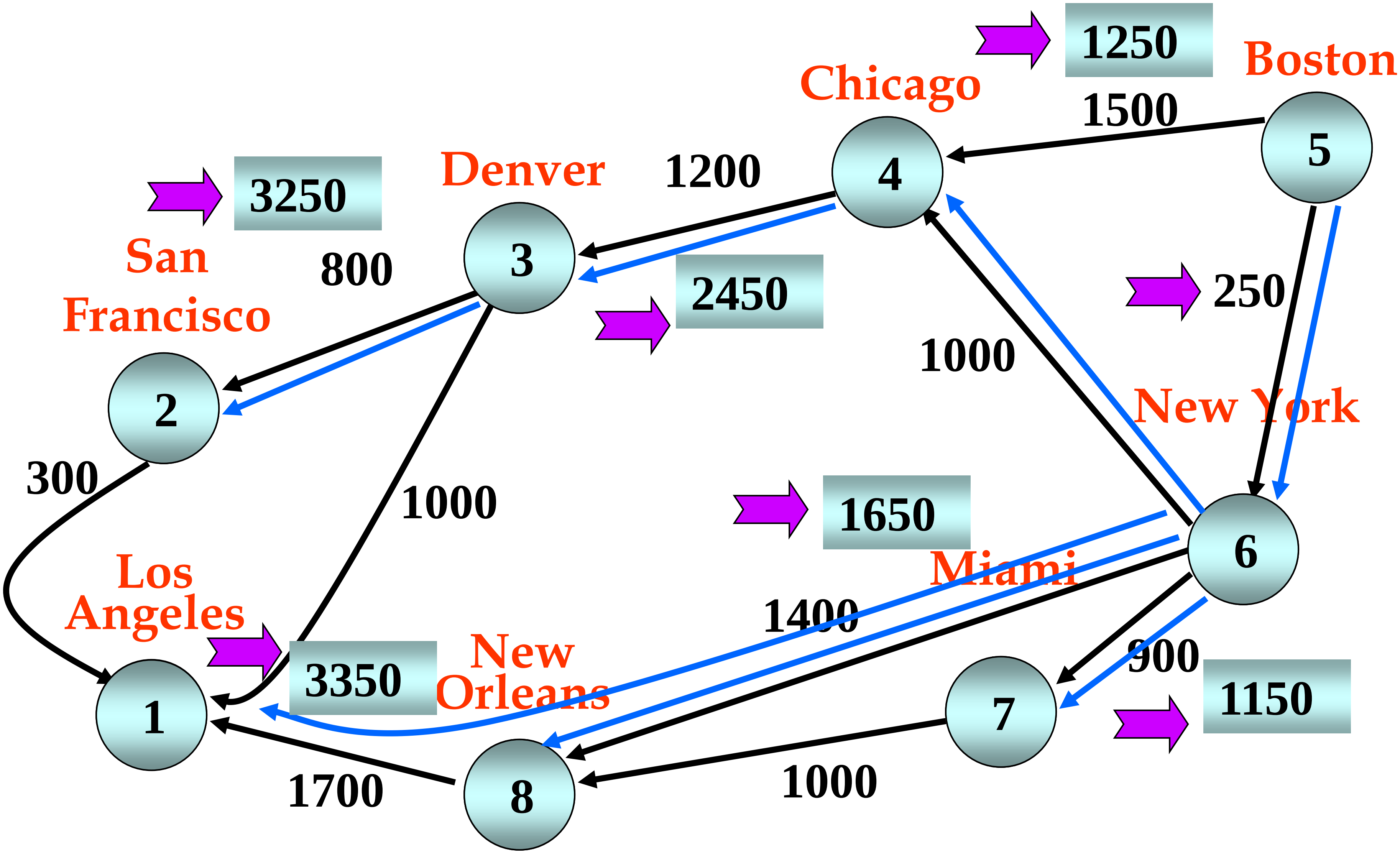
var boolean S(1:n); real COST(1:n, 1:n), DIST(1:n)

integer u, v, n, num, i, w;

begin

for i ← 1 to n do //initialize set S to empty//

```
begin
        S[i] ← 0; DIST(i) ← COST[v,i];
end
S[v] ← 1; DIST[v] ← 0;  //put the vertex v in set S//
for num ← 2 to n-1 do   //determine n-1 paths from
                                              vertex v//
begin
   choose u such that DIST[u] = min{DIST(w)}
   S[w] ← 0;
   S[u] ← 1;  //put vertex u in set S//
   for all w with S[w] = 0 do //update distances//
        DIST[w] ← min{DIST(w), DIST(u) +COST(u,w)}
end
end.
```


1250
Boston
Chicago
1500
5
4
3250
Denver
1200
3
San
Francisco
800
2450
250
1000
2
New York
300
1000
1650
6
Miami
Los
Angeles
1400
New
Orleans
900
3350
1150
7
1
8
1700
1000