Chapter: 3

Search Techniques

Production Techniques :

Production Technologies is divided to three Techniques:

- Deduction Techniques تقنيات الاستدراج
- Abduction Techniques تقنيات الاسترشاد
- Induction Techniques تقنيات الاستدلال

Deduction Techniques

- Is to prove that the desired goal is true
- This technique is called Backward Reasoning

Abduction Techniques

- It analyzes the target into parts and prove that all these parts belong to the knowledge base.
- This technique is called Forward Reasoning

Induction Techniques

- **It is used in building and programming neural networks.**

Search Techniques In KB

- Inference engine commonly uses two modes:

1. Forward chaining
2. Backward chaining

Search Techniques In KB

- **Forward chaining** is the logical process of inferring unknown facts from known data and moving forward using determined conditions and rules until a goal is reached.

Search Techniques In the KB

- **Backward chaining** is a form of reasoning, which starts with the goal and works backward chaining through rules to find known facts that support the goal.(Is to prove that the desired goal is true)

Example

Consider the following KB, and explain how to reach the following goal:

Goal: goal1 ; goal2 (Z)

Example

K.B:

f1 : fact2 .

f2 : fact3 .

R1 : goal1 :- fact1 .

R2 : goal1 :- a, b .

R3 : goal2 (X) :- c(X).

R4 : a :- not (d) .

Example

R5 : b :- d.

R6 : b :- e.

R7 : c(2) :- not (e).

R8 : d :- fact2 , fact3 .

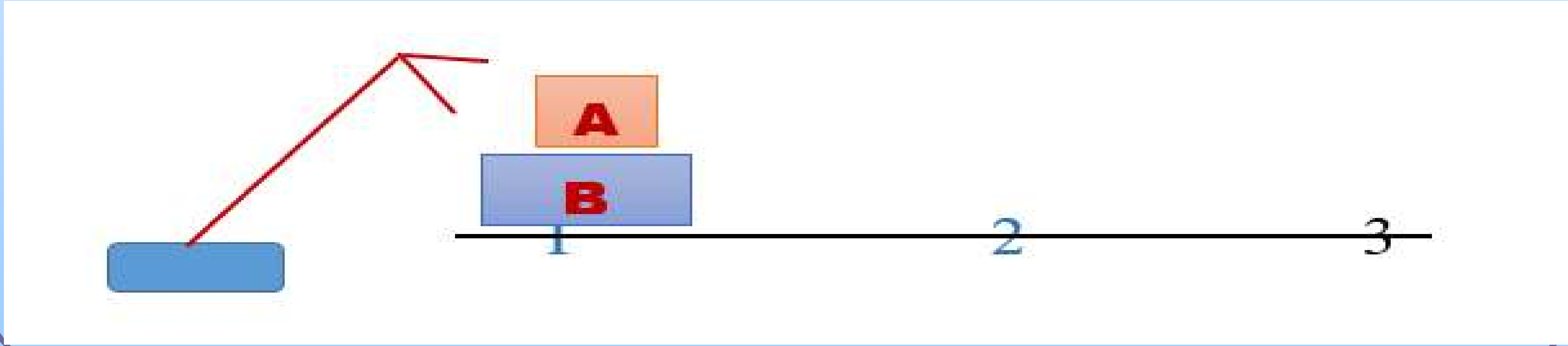
R9 : e :- fact2 , fact4.

Search Techniques for case-based knowledge

- **It is a searching Technique for the target in the tree structure.**
- **The tree structure is generated gradually by levels, in each level the target is searched**

Example

There are two boxes (A,B), the size of box B is greater than A and there is a winch and we need to move the boxes from location 1 to location 3 same as the same such that the winch can move only one box in each time.



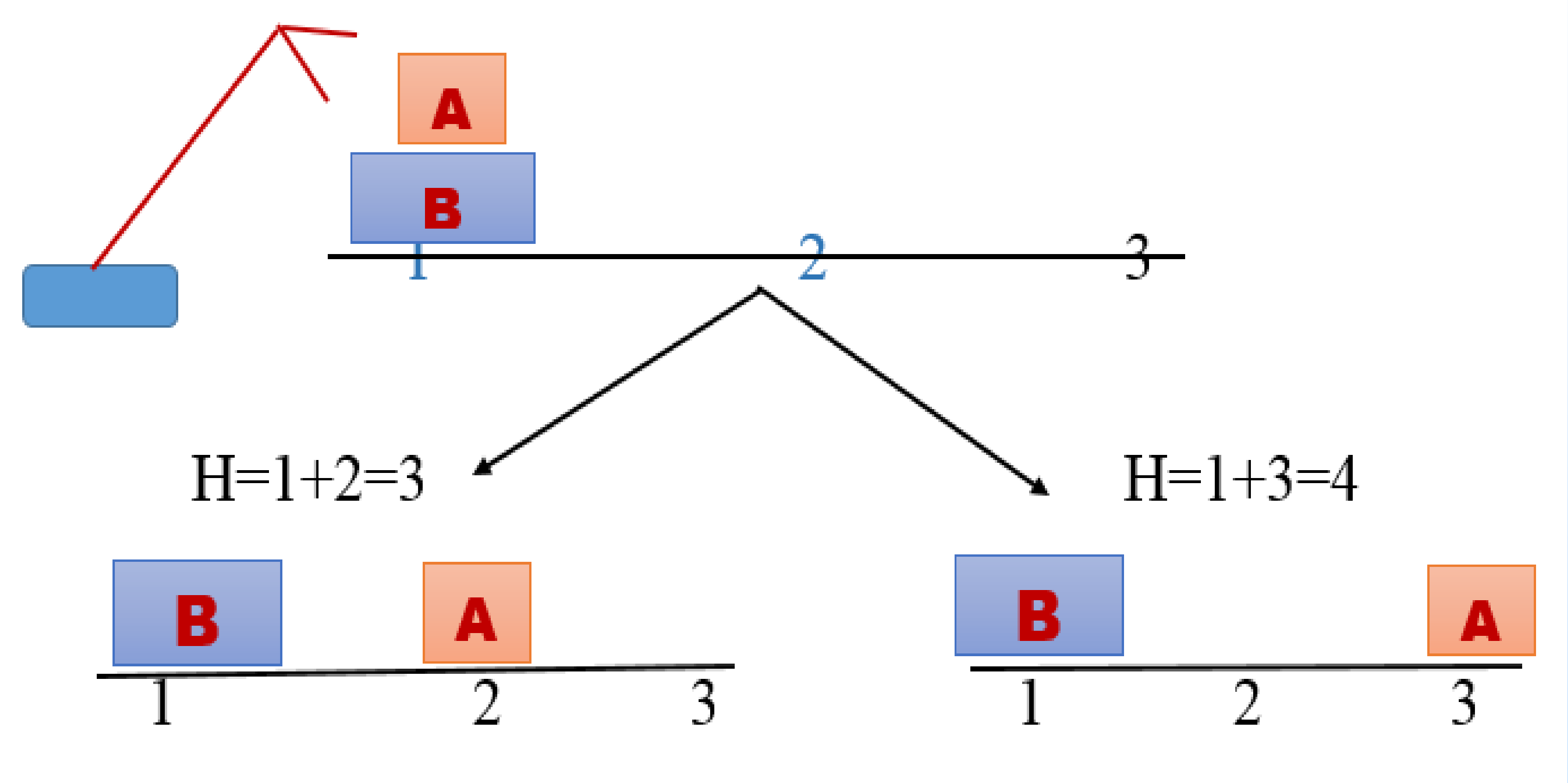
Example

And use the following equation to reach the target:

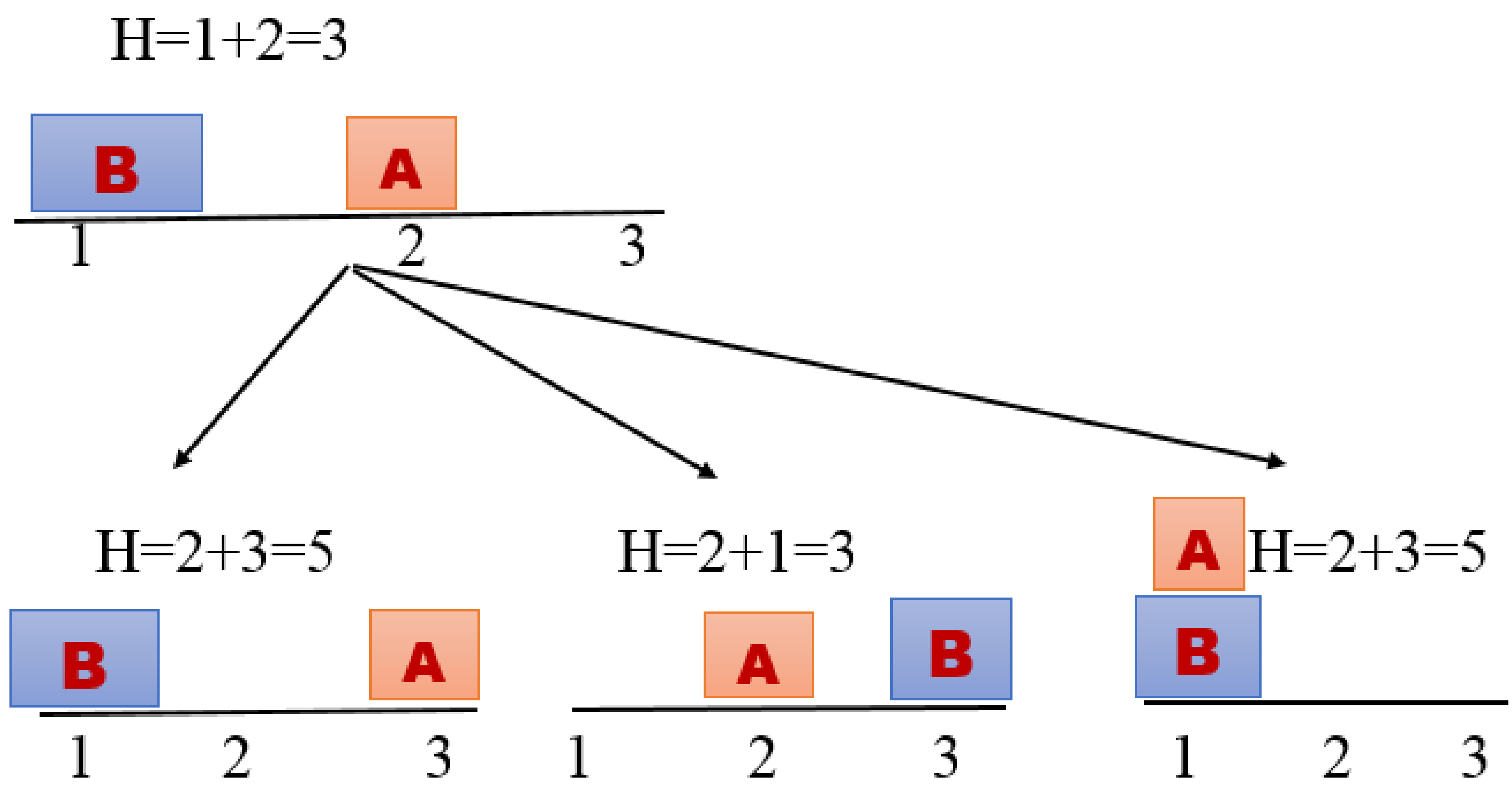
$$H = T + E; \qquad T = 0, 1, 2, 3, 4, ..$$

Box	E
B	1
A	3
A,B	0
Null	2

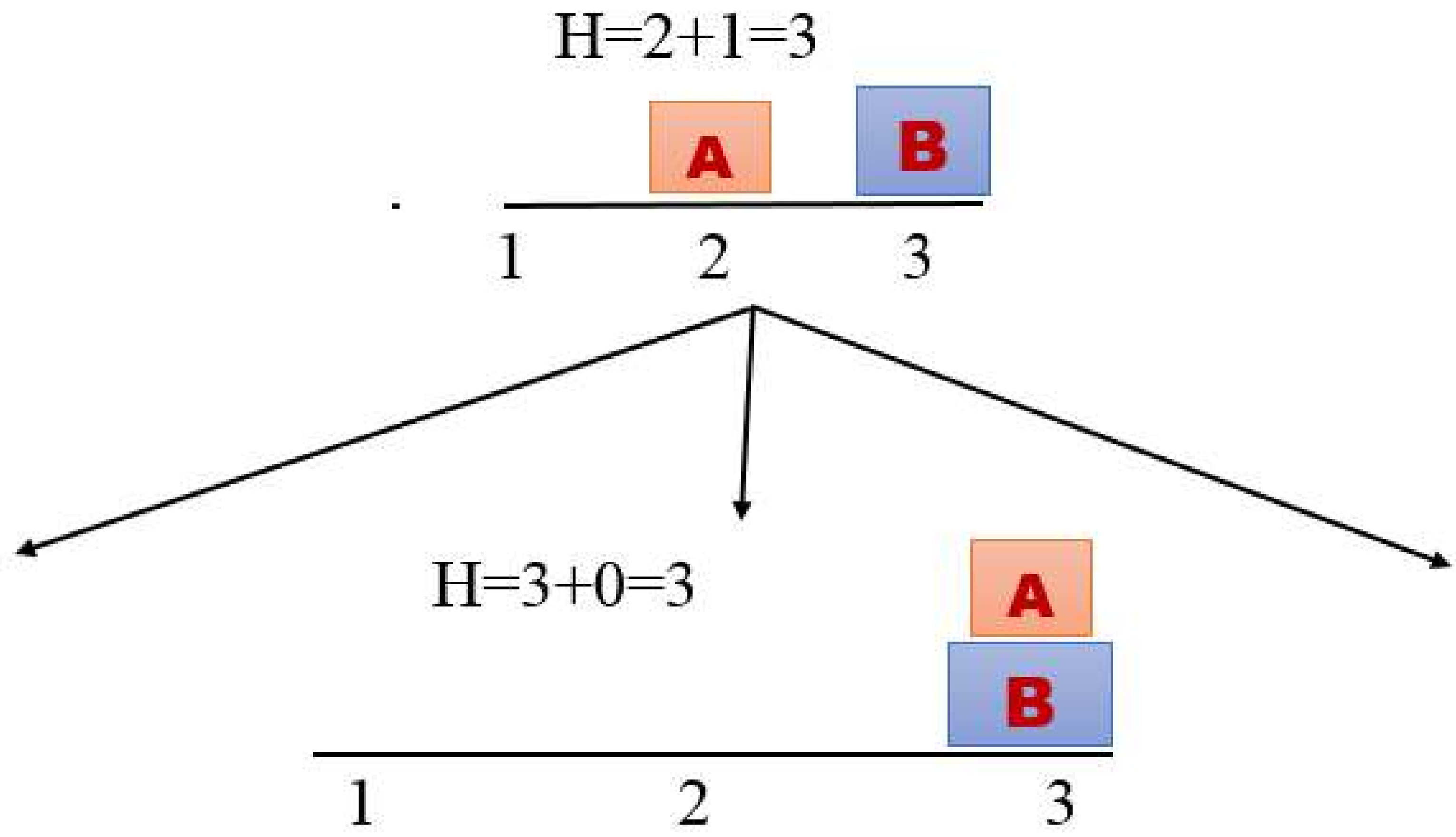
Example

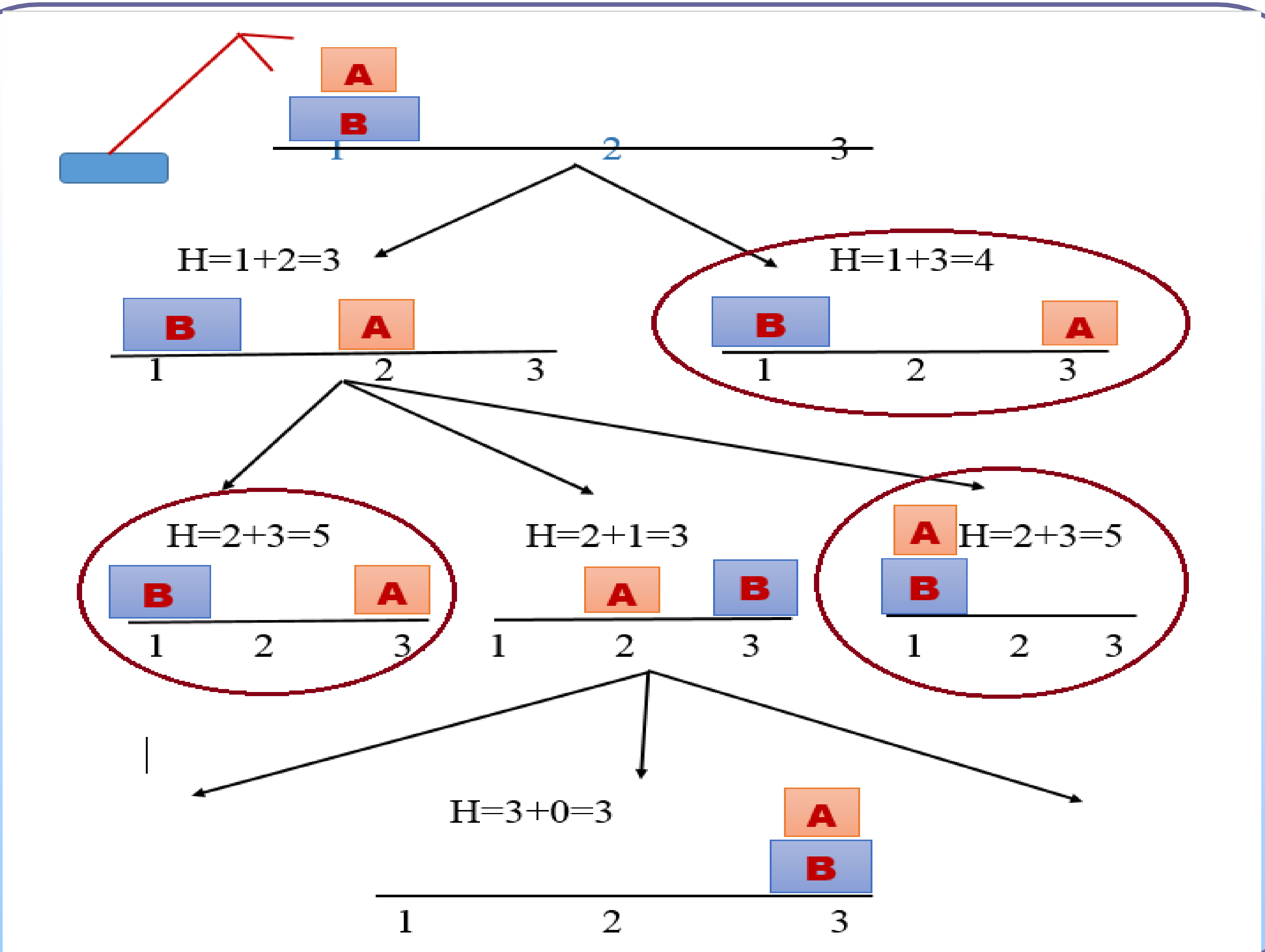


Example



Example




A
B
1
2
3
H=1+2=3
H=1+3=4
B
A
1
2
3
B
A
1
2
3
H=2+3=5
H=2+1=3
H=2+3=5
B
A
1
2
3
A
B
1
2
3
A
B
1
2
3
H=3+0=3
A
B
1
2
3

Generative Search Techniques

1. Systematic Techniques
   - Breadth-First Search Algorithm
   - Depth-First Search Algorithm
2. Heuristic Searches(Optimal Techniques)
   - Hill Climbing Algorithm
   - Best First Algorithm
   - A Algorithm
   - A* Algorithm

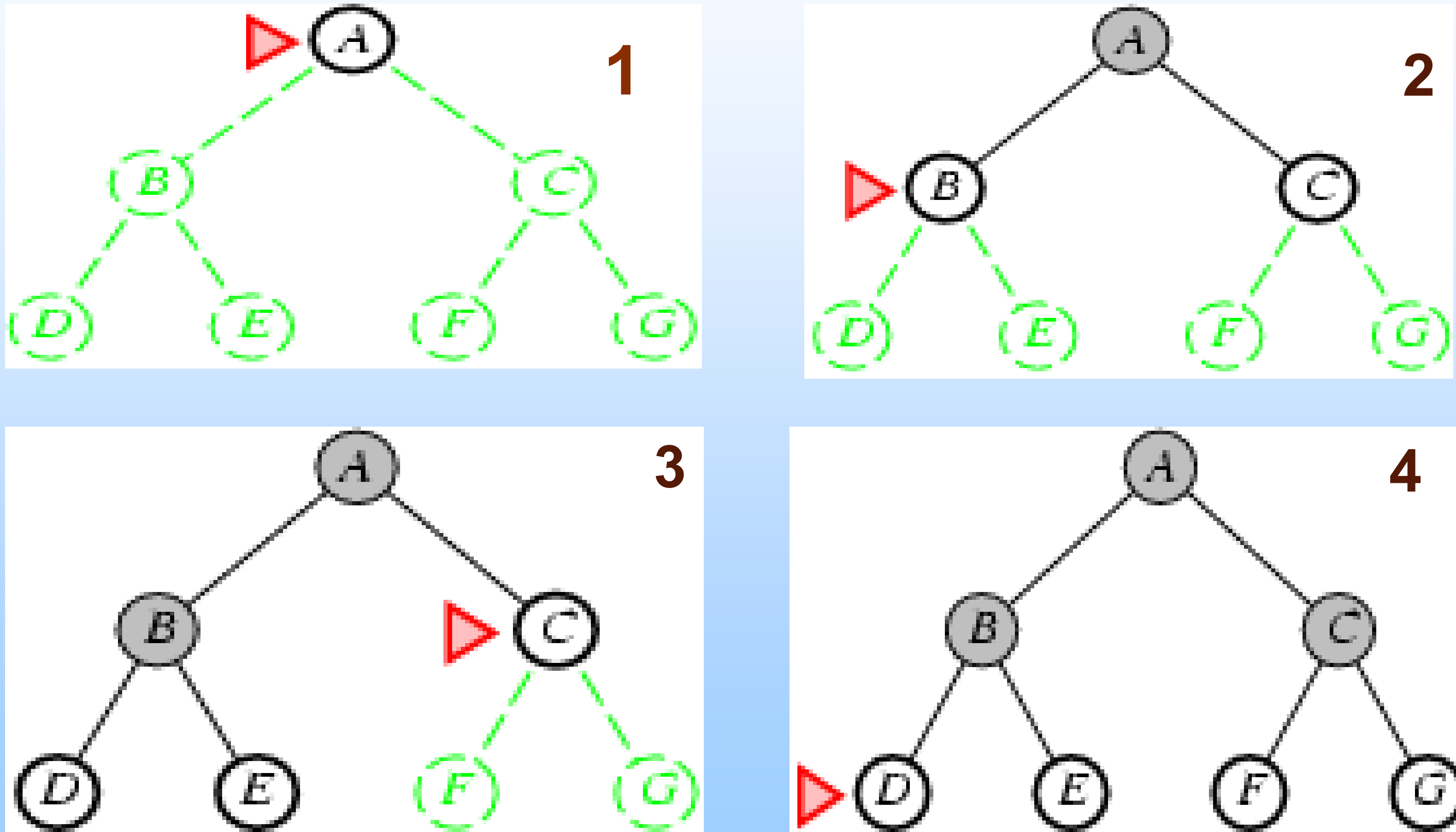
Generative Search Techniques

3. Genetics Algorithms
4. Bee Algorithm
5. Ant Colony Algorithm

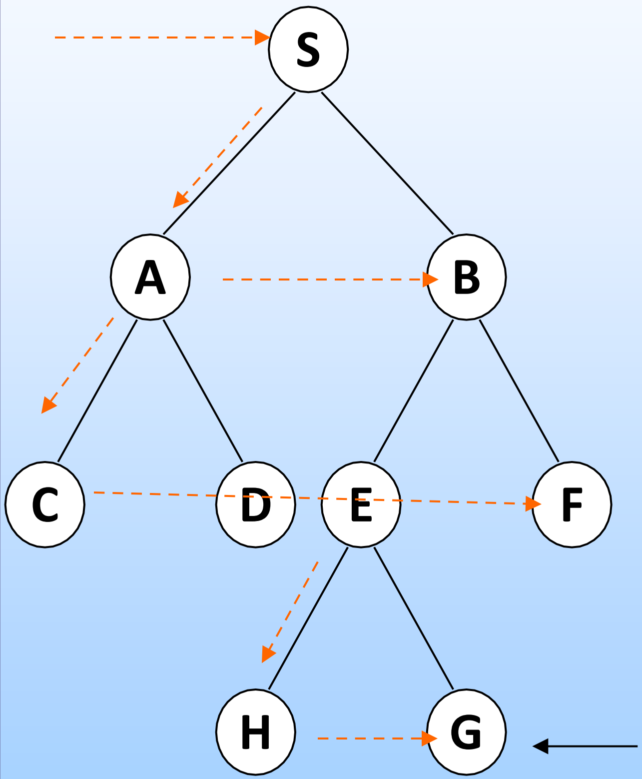
Breadth-First Search (BFS) Algorithm

- The breadth-first search (BFS) algorithm is used to search a tree or graph data structure for a node that meets a set of criteria. It starts at the tree's root or graph and searches/visits all nodes at the current depth level before moving on to the nodes at the next depth level. Breadth -first search can be used to solve many problems in graph theory.

Breadth-First Search (BFS) Algorithm



Breadth-First Search (BFS) Algorithm



الهدف
Goal

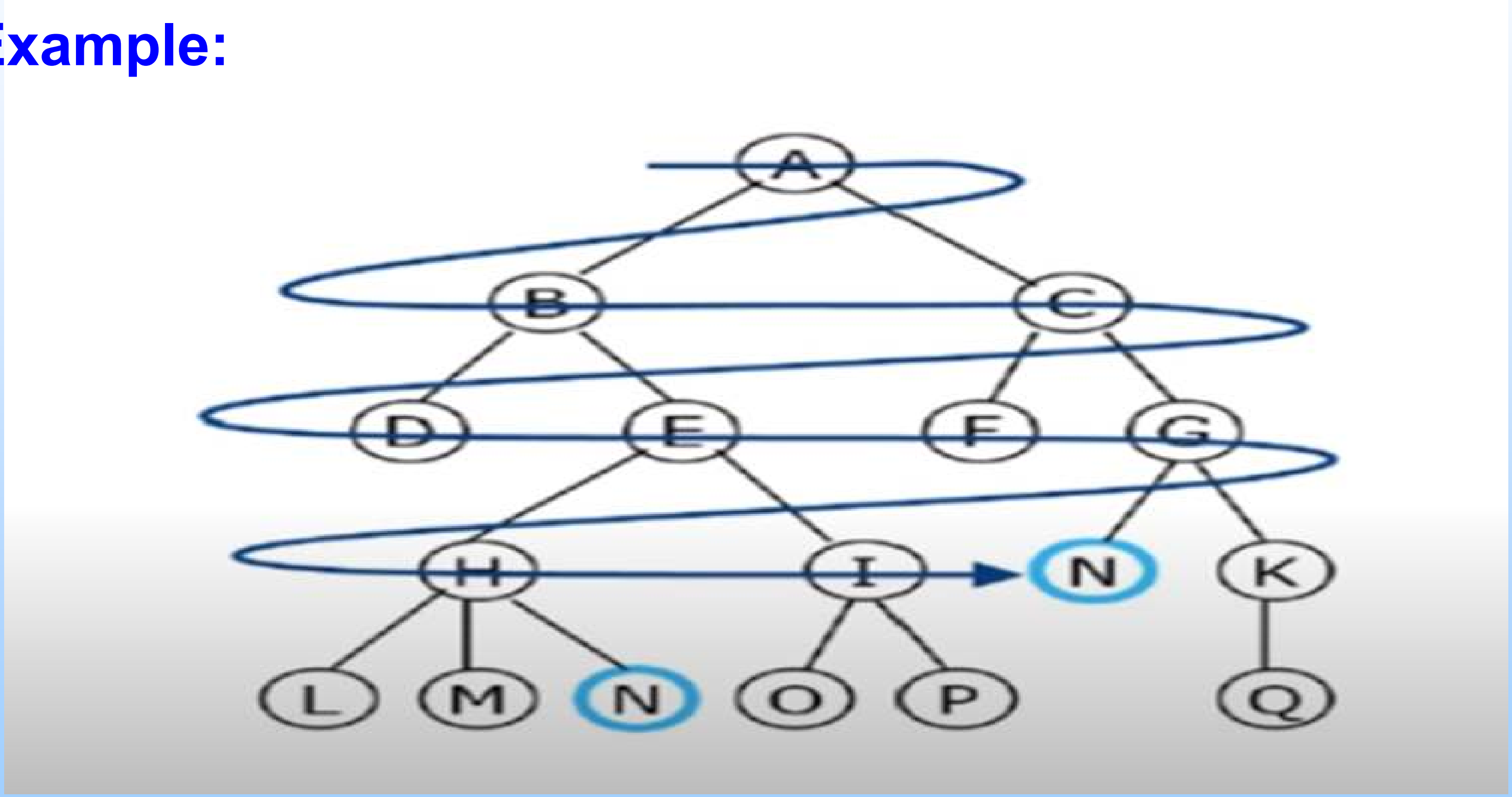
S
AB
BCD
CDEF
DEF
EF
FHG
HG
G

The search is done from one level to another, and the algorithm moves from one level to the next only when there are no other cases to explore at a certain level.

Goal Path: S,A,B,C,D,E,F,H,G

Breadth-First Search (BFS) Algorithm

Example:



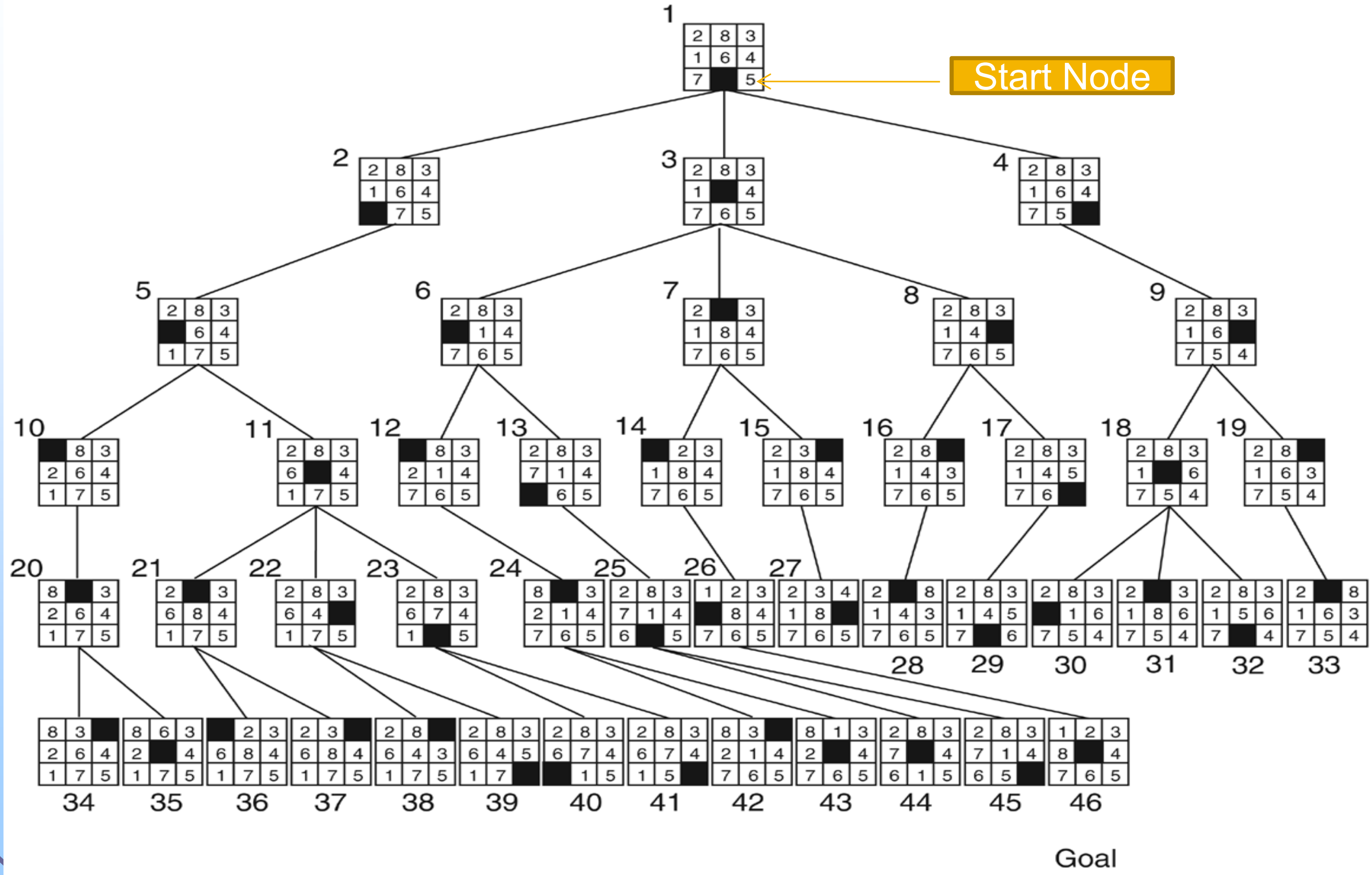
Goal Path: A, B, C, D, E, F, G, H, I, N

Breadth-First Search (BFS) Algorithm

```
begin
   open := [Start];
   closed := [ ];
   while open ≠ [ ] do
      begin
         remove leftmost state from open, call it X;
            if X is a goal then return SUCCESS
               else begin
                  generate children of X;
                  put X on closed;
                  discard children of X if already on open or closed;
                  put remaining children on right end of open
               end
      end
   return FAIL
end.
```

Example:1

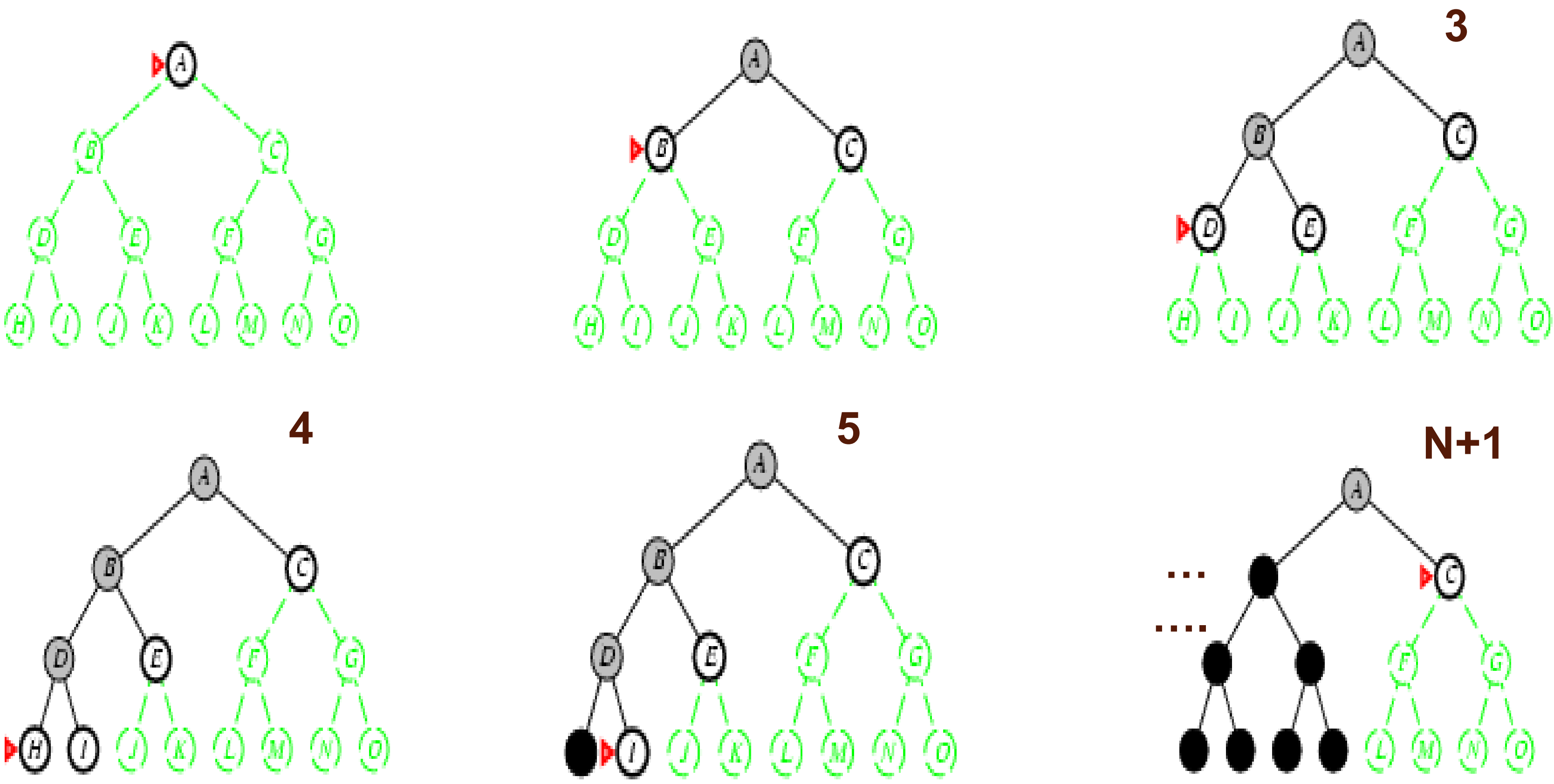
Example:2 (8-puzzle)



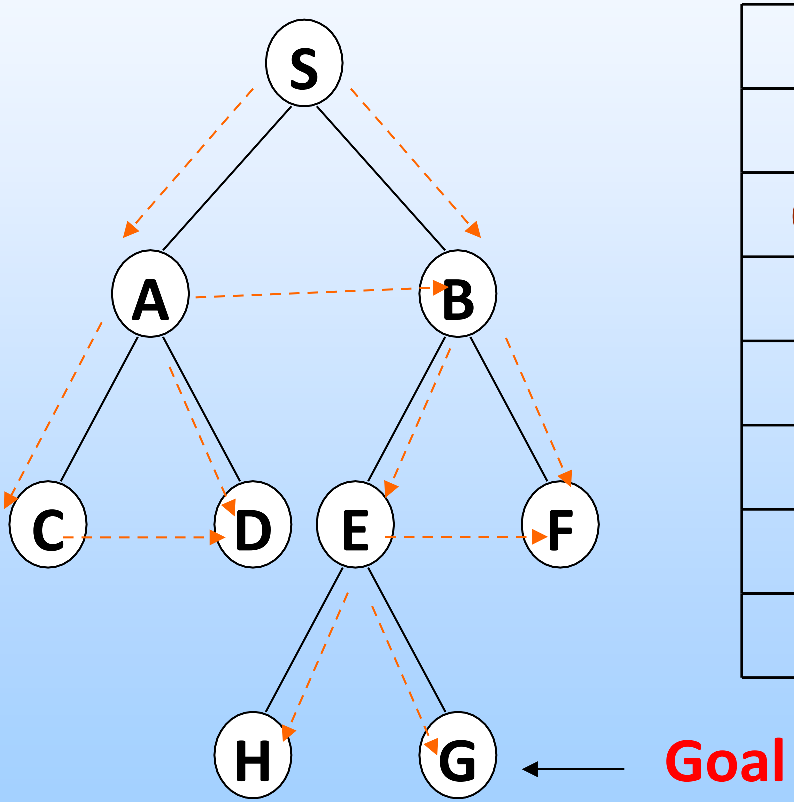
Depth-First Search (DFS) Algorithm

- It is a recursive algorithm to search all the vertices of a tree data structure or a graph. The depth-first search (DFS) algorithm starts with the initial node of graph G and goes deeper until we find the goal node or the node with no children.

Depth-First Search (DFS) Algorithm



Depth-First Search (DFS)

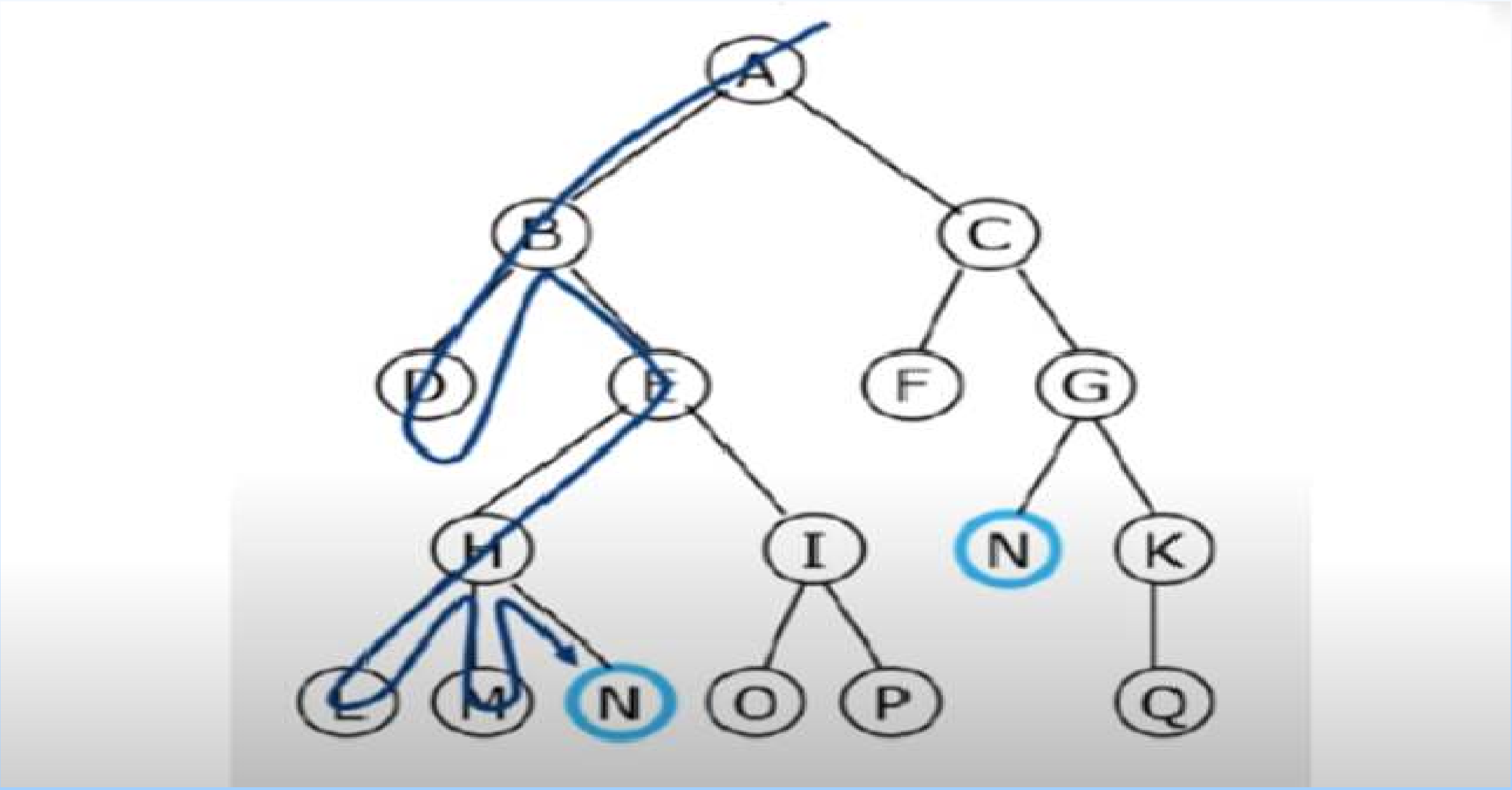


S
AB
CDB
DB
B
EF
HGF
GF

Here all children and their grandchildren are examined and tested before brothers are tested. In other words, depth-first search goes deeper into the research space whenever possible.

Goal Path: S,A,C,D,B,E,H,G

Depth-First Search (DFS)



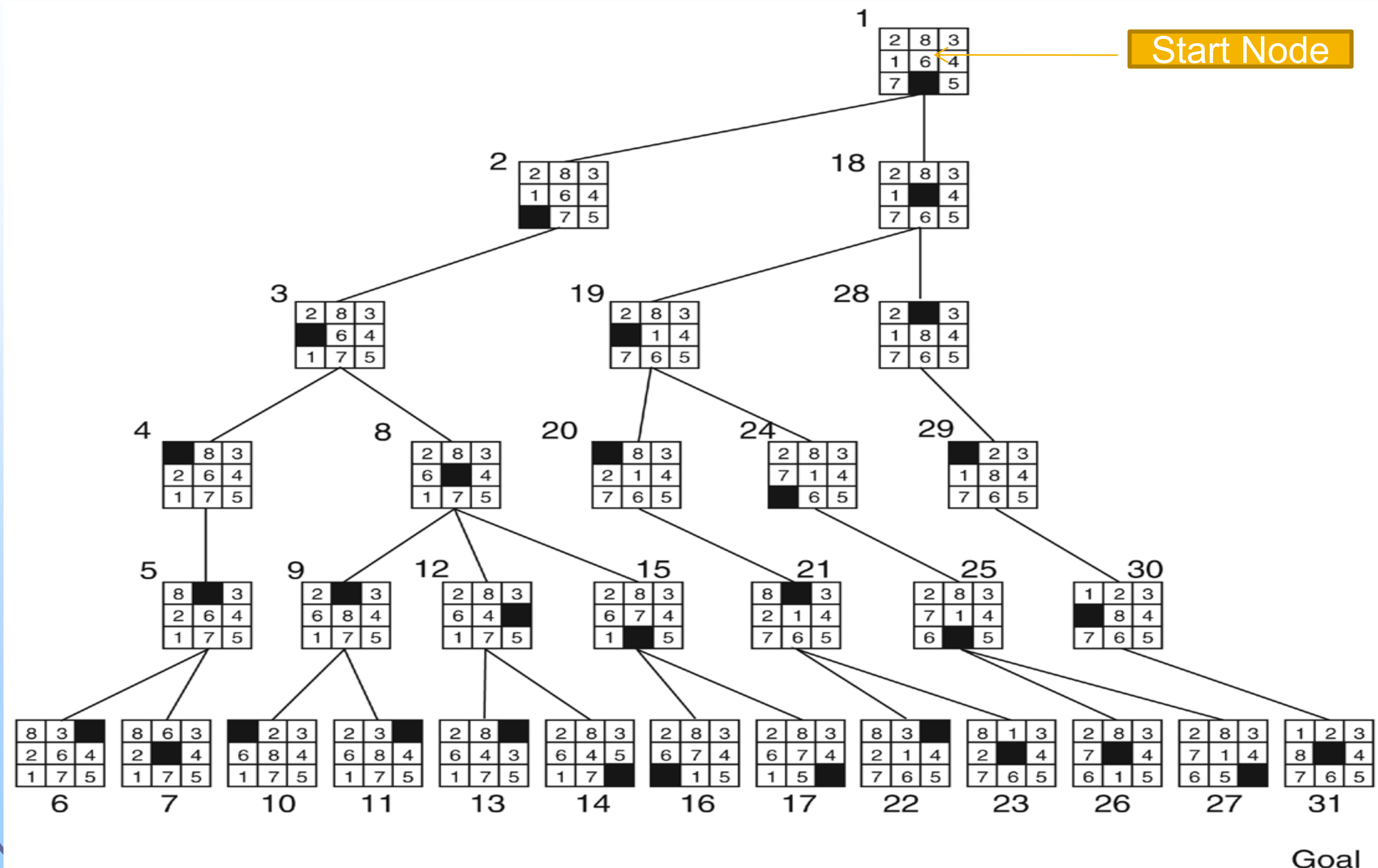
Goal Path:A, B, D, E, H,L, M,N

Depth-First Search (DFS) Algorithm

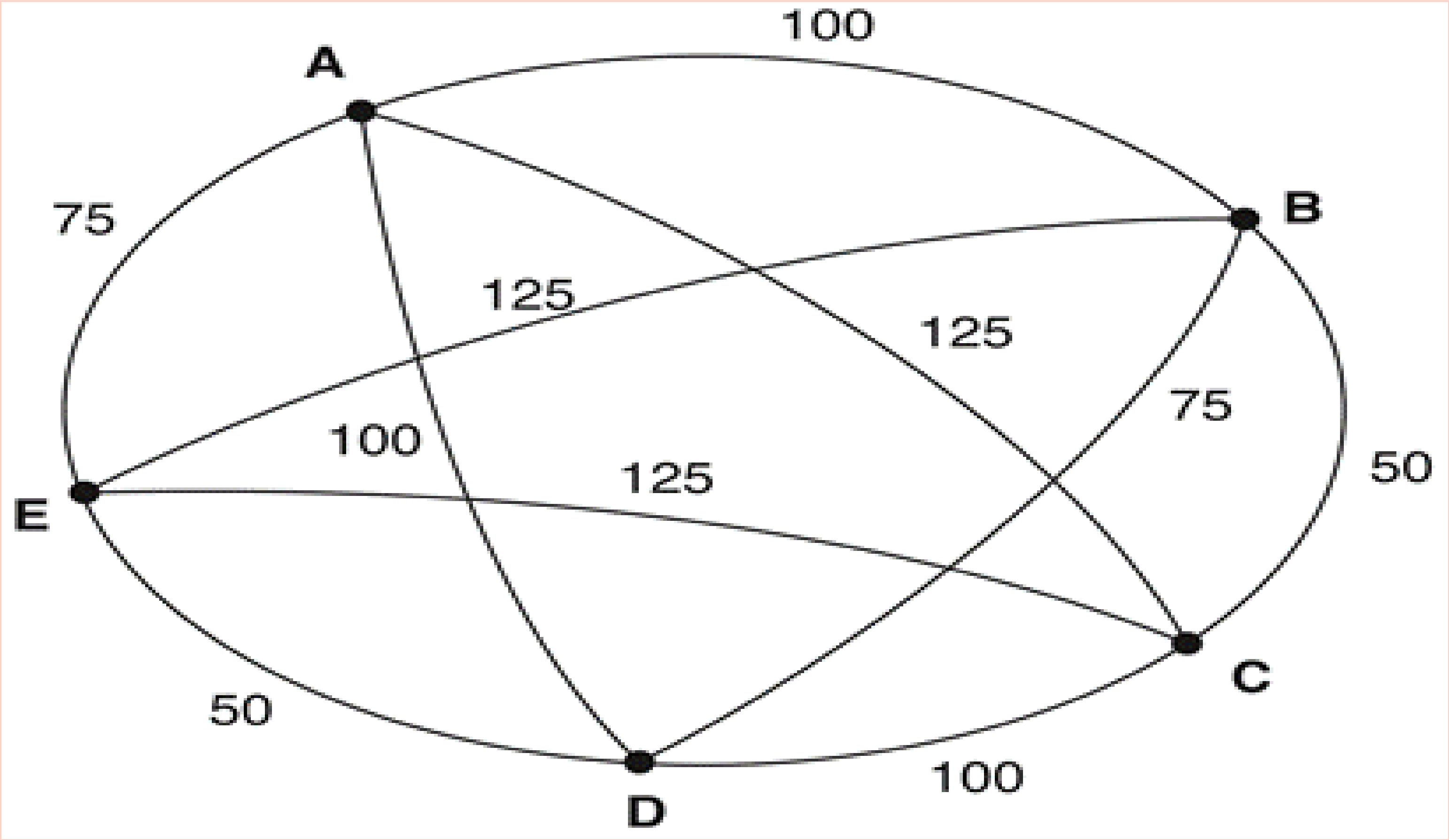
```
begin
   open := [Start];
   closed := [ ];
   while open ≠ [ ] do
      begin
         remove leftmost state from open, call it X;
         if X is a goal then return SUCCESS
            else begin
               generate children of X;
               put X on closed;
               discard children of X if already on open or closed;
               put remaining children on left end of open
            end
      end;
   return FAIL
end.
```

Example:1

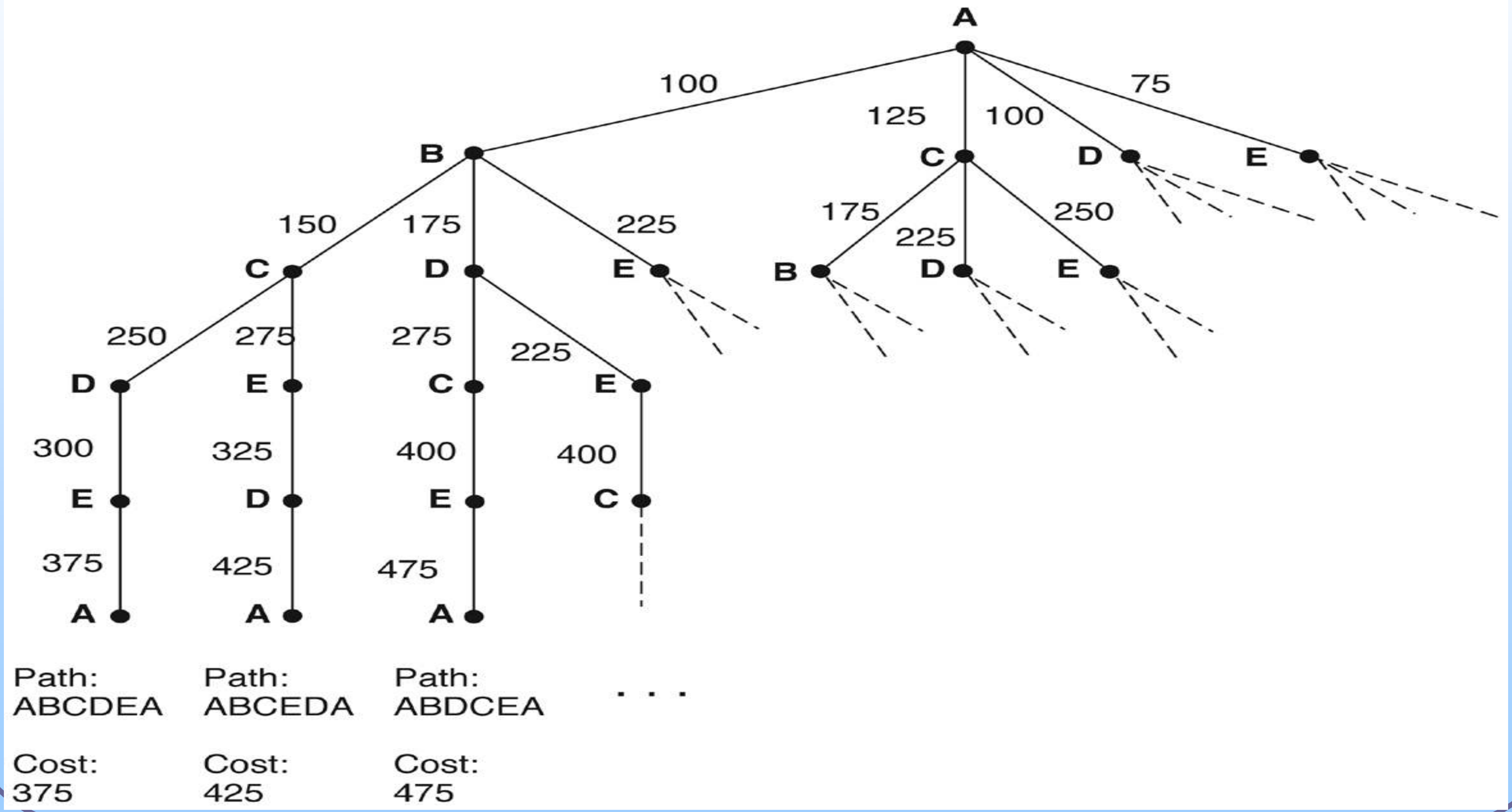
Example:2 (8-puzzle)



Example:3 (Traveling salesman)



Example:3 (Traveling salesman)



Depth-First Search(DFS).

- All paths lead to the goal, but what is an optimal path.
- The optimal path is the path of minimum cost.

Optimal Methods

- **The optimal search method requires a type of costs along with the cognitive structure. These costs represent a variety of values, which may be a distance between cities in kilometers or any other indication.**

- **So searching in this method means searching for the path that leads to the goal at the minimal cost.**

Optimal Methods

The process of calculating the cost and reaching the goal is done by two methods:

- **Hill Climbing:**

The cost is calculated by summing the cost of the path to the target.

- **Best-First:**

The cost of each node is the cost of accessing it from the previous node only.

Optimal Methods

- Calculate the costs of reaching the nodes.
- Arrange these nodes in the open matrix in ascending order.
- Select the beginning of the nodes and examine it is a target or not.
- When the target is not found, this node is deleted and calculate the cost of its children
- The matrix elements are arranged in ascending order with the new children, and so we continue until the target is found.

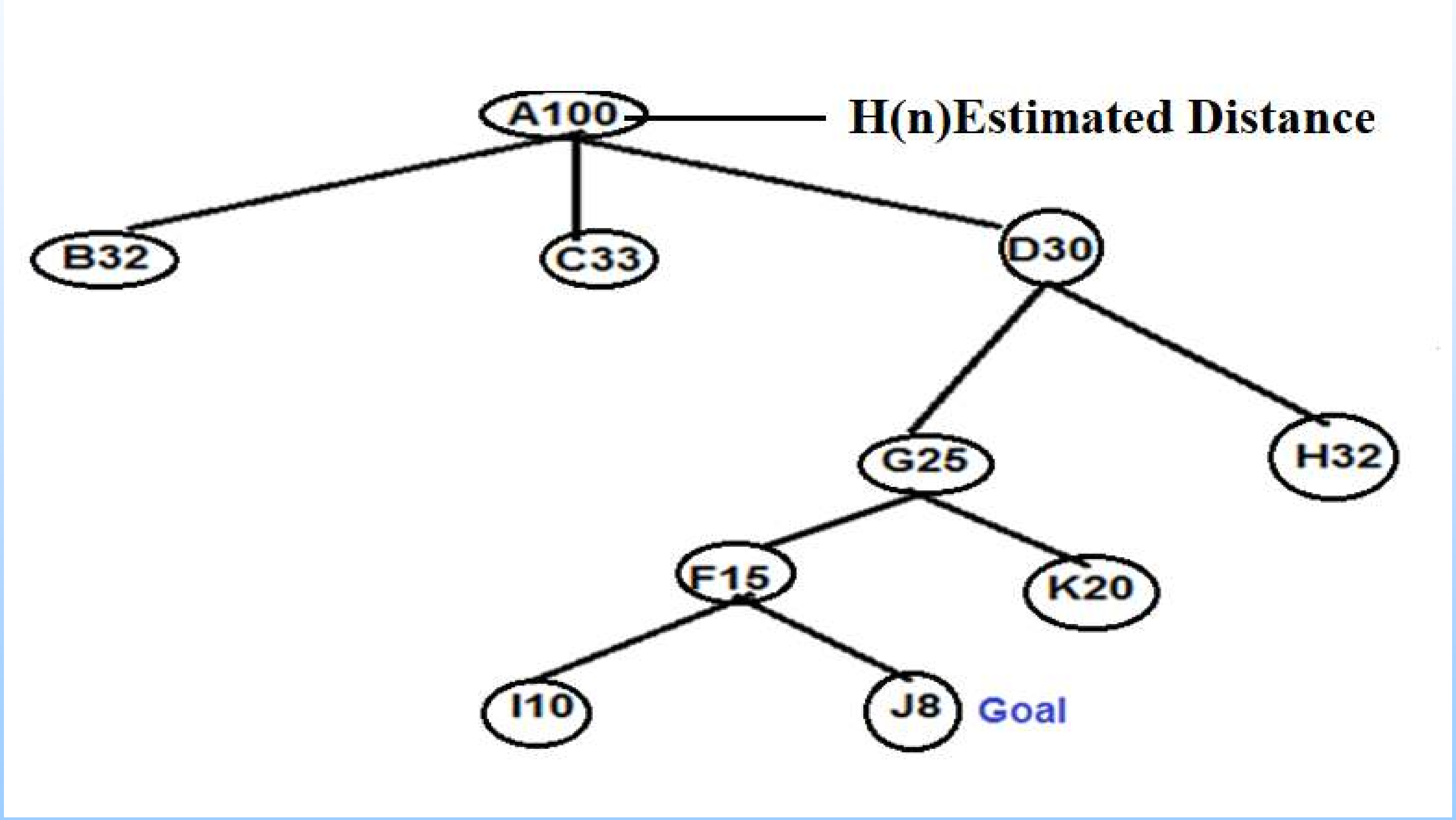
Hill Climbing

Important note:

- Hill Climbing algorithm does not have the ability to keep previous nodes because there is no memory, hence It is not possible to refer to the previous nodes and process them.

- Hill Climbing algorithm is faster to reach the target

Example:1

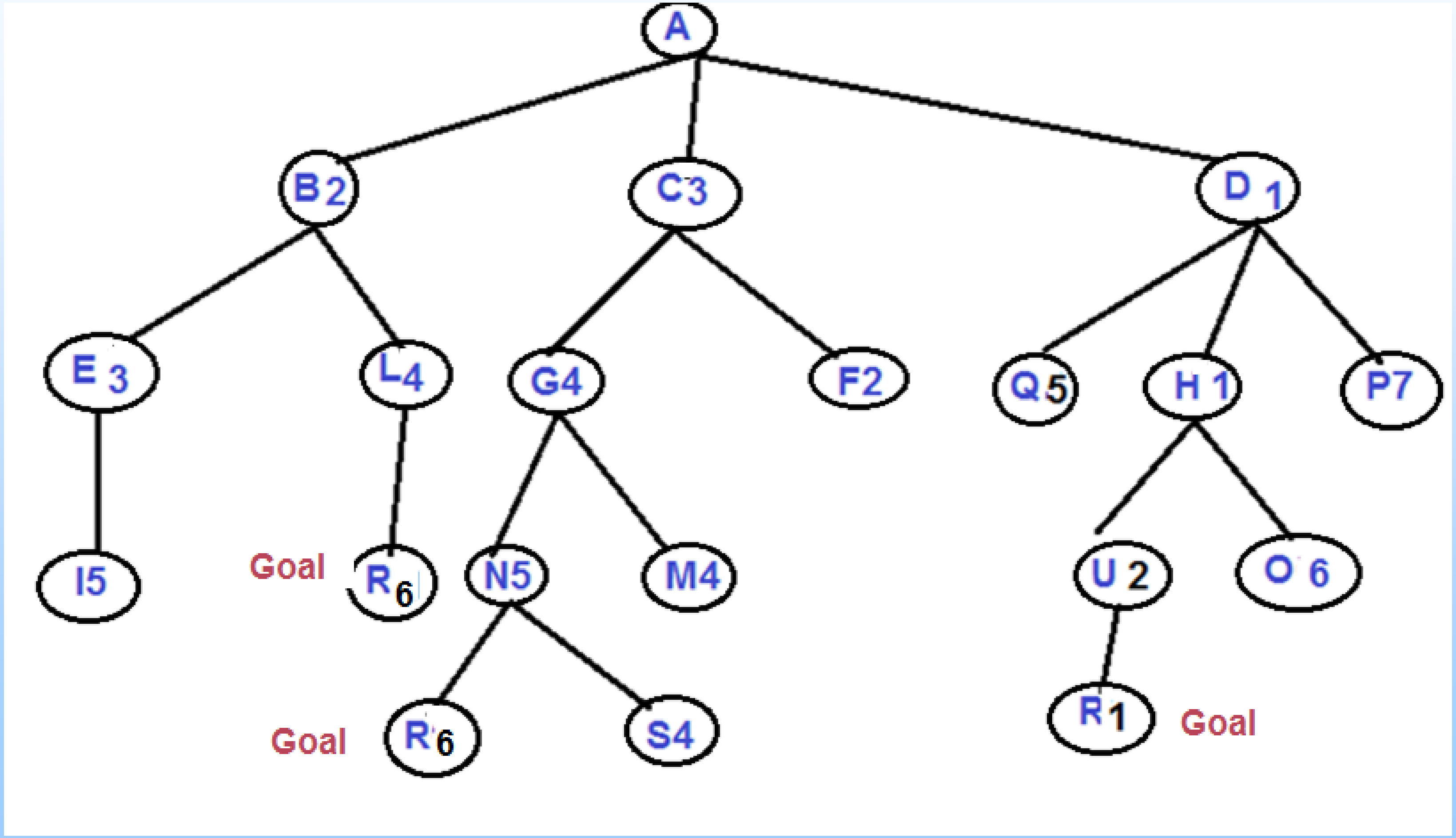

A100
H(n)Estimated Distance
B32
C33
D30
G25
H32
F15
K20
I10
J8
Goal

Example:1

Open	Close
[A100]	[ ]
[D30, ~~B32~~, ~~C33~~]	[A100]
[G25, ~~H32~~]	[A100, D30]
[F15, ~~K20~~]	[A100, D30, G25]
[J8, I10]	[A100, D30, G25, F15]
Stop	[A100, D30, G25, F15, J8]

Goal Path: A-> D-> G-> F-> J

Example:2


A
B2
C3
D 1
E 3
L4
G4
F2
Q5
H 1
P7
I5
Goal
R6
N5
M4
U 2
O 6
Goal
R6
S4
R1
Goal

Example:2

Open	Close
[A]	[ ]
[D1, ~~B2~~, ~~C3~~]	[A]
[H1, ~~Q5~~, ~~P7~~]	[A, D1]
[U2, ~~O6~~]	[A, D1, H1]
[<u>R1</u>, I10]	[A, D1, H1, U2]
Stop	[A, D1, H1, u2, R1]

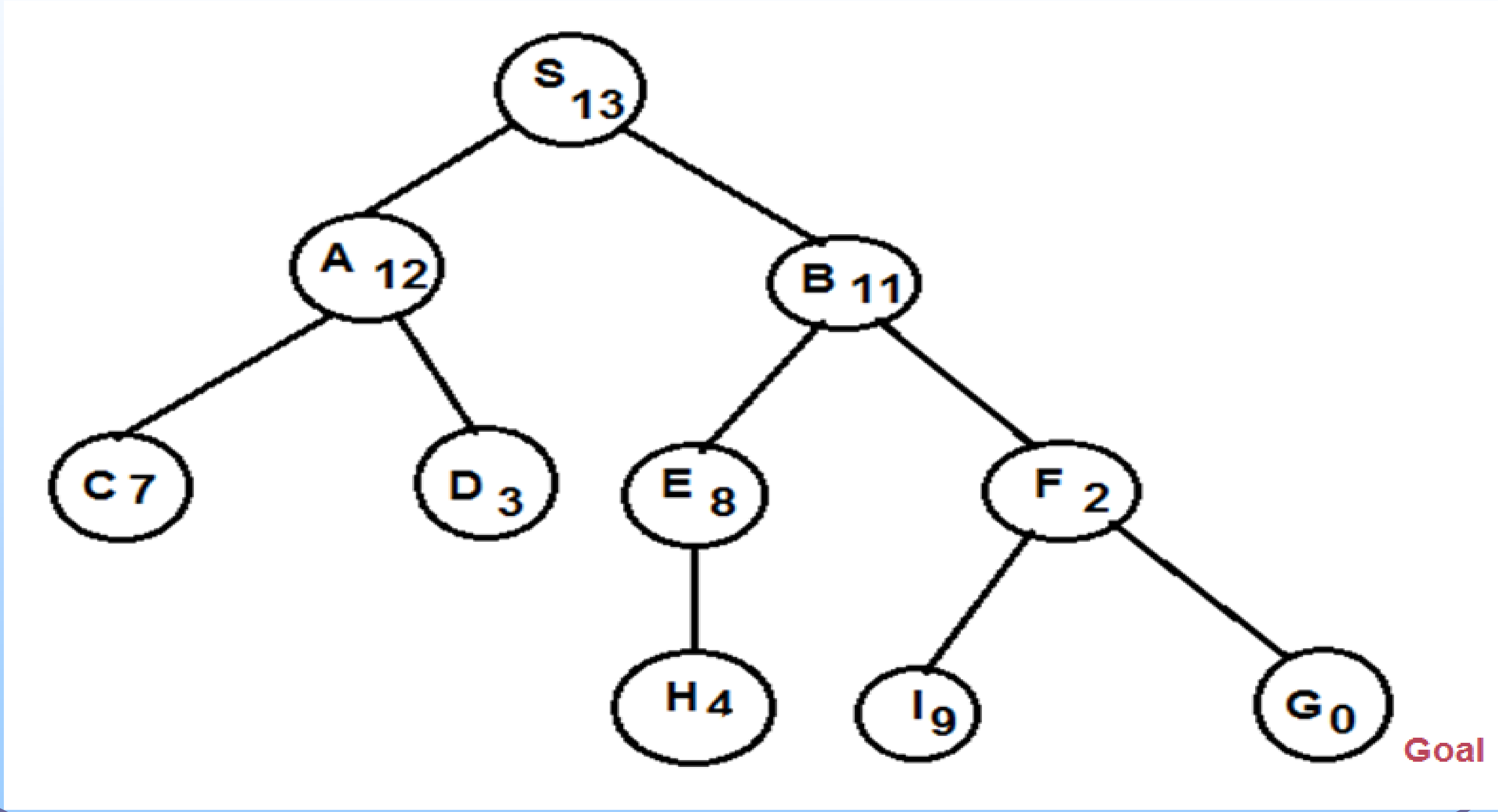
Goal Path: A-> D-> H-> U-> R

Best First

Important note:

- Best First have the ability to keep previous nodes because there is a memory, hence It is possible to refer to the previous nodes and process them

Example:1

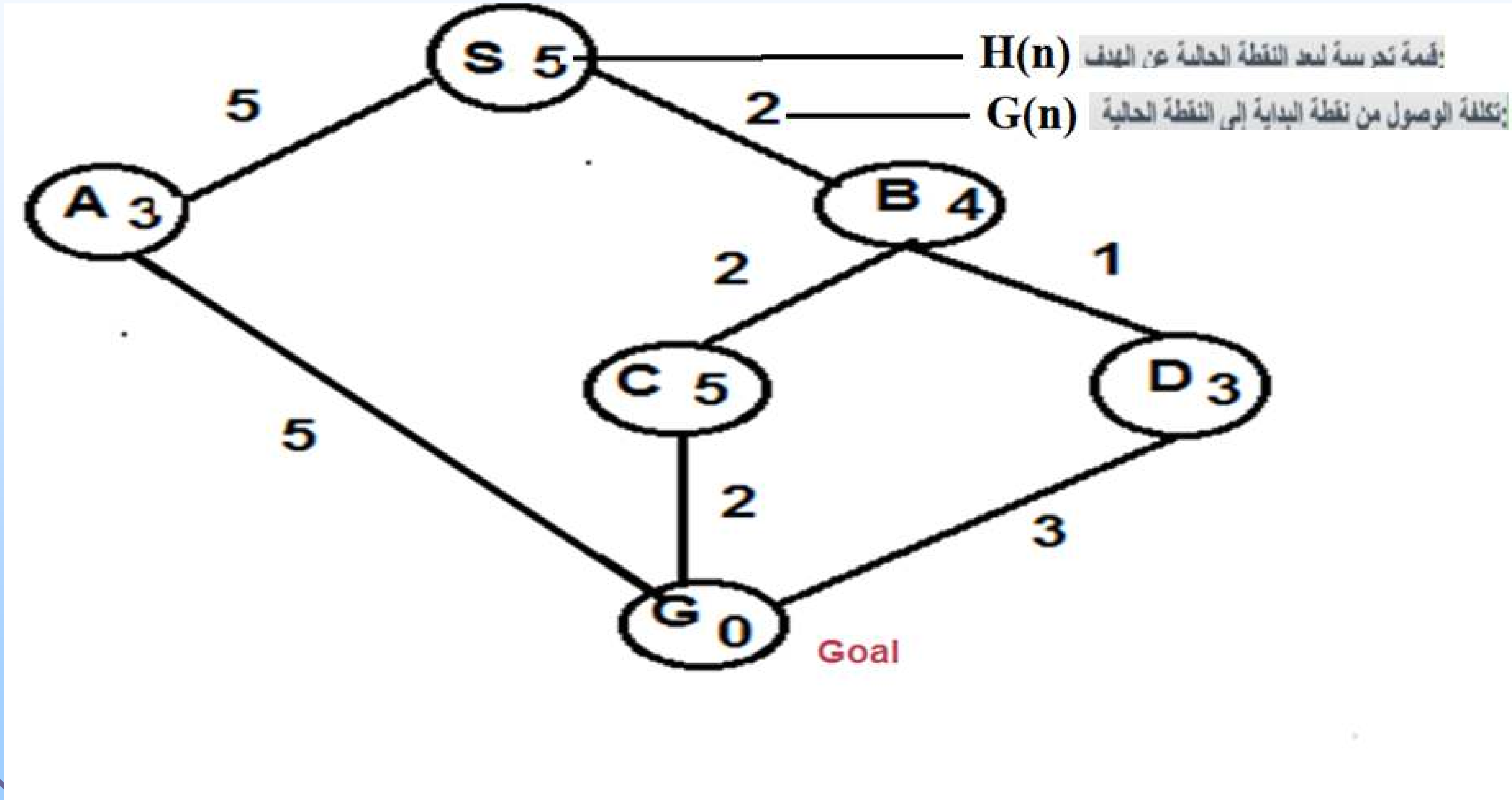


Example:1

Open	Close
[S13]	[ ]
[B11, A12]	[S13]
[F2, E8, A12]	[S5, B11]
[G0, E8, , I9, A12]	[S5, B11, F2]
STOP	[S5, B11, F2, G0]

Goal Path: S -> B -> F -> G

Example:2



S 5
A 3
B 4
C 5
D 3
G 0
Goal
5
2
2
1
5
2
3
H(n) :قيمة تحدسية لبعد النقطة الحالية عن الهدف
G(n) :تكلفة الوصول من نقطة البداية إلى النقطة الحالية

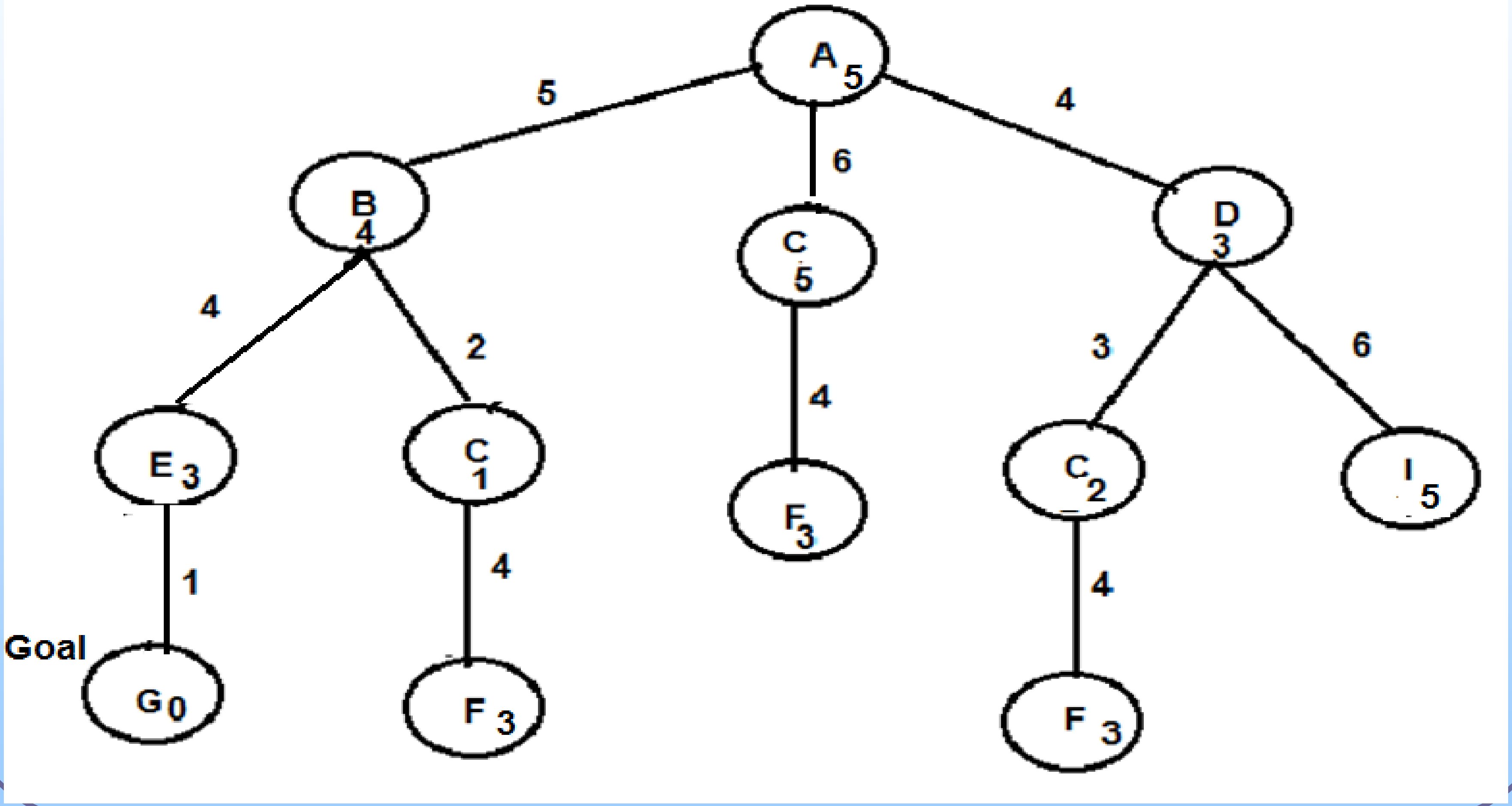
Example:2

Open	Close
[S5]	[ ]
[A3, B4]	[S5]
[G0, B4]	[S5, A3]
Stop	[S5, A3, G0]

Goal Path: S_0 -> A_5 -> G_5

Path Cost=0+5+5=10

Example:3



Example:3

Open	Close
[A5]	[ ]
[D3, B4, ~~C5~~]	[A5]
[C2, B4, I5]	[A5, D3]
[F3, B4, I5]	[A5, D3, C2]
[B4, I5]	[A5, D3, C2, F3]
[C1, E3, I5]	[A5, D3, ~~C2~~, F3, B4]
[E3, I5]	[A5, D3, F3, B4, C1]
[G0, I5]	[A5, D3, F3, B4, C1, E3, G0]

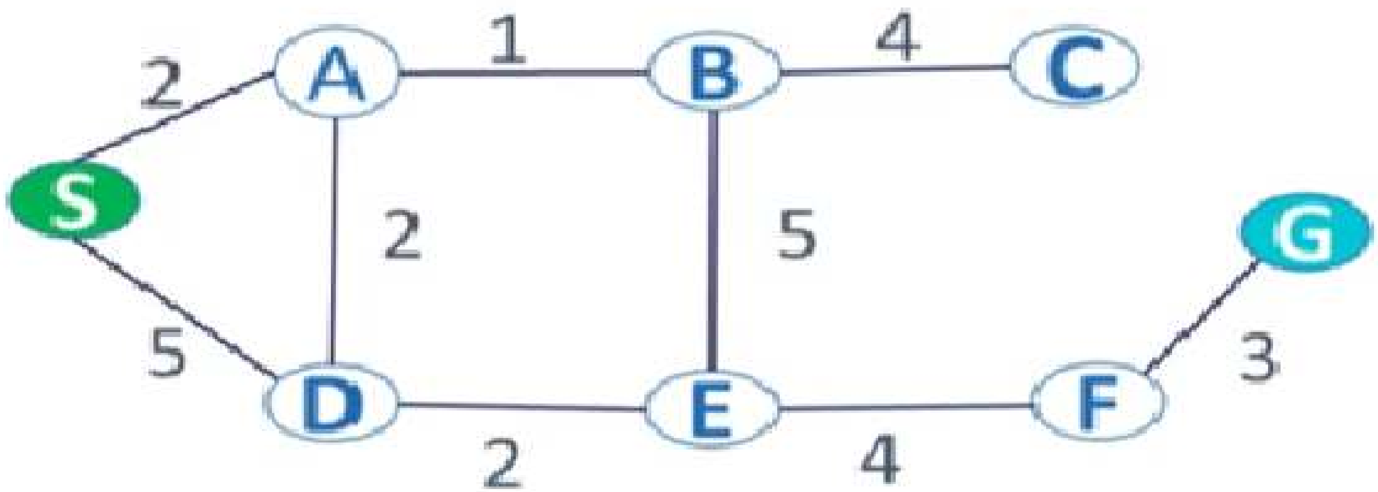
Goal Path: A_0 -> D_4 -> F_7 ->B_{16} ->C_2 ->E_6 ->G_1

Path cost=0+4+7+16+2+6+1= 36

A* Algorithm

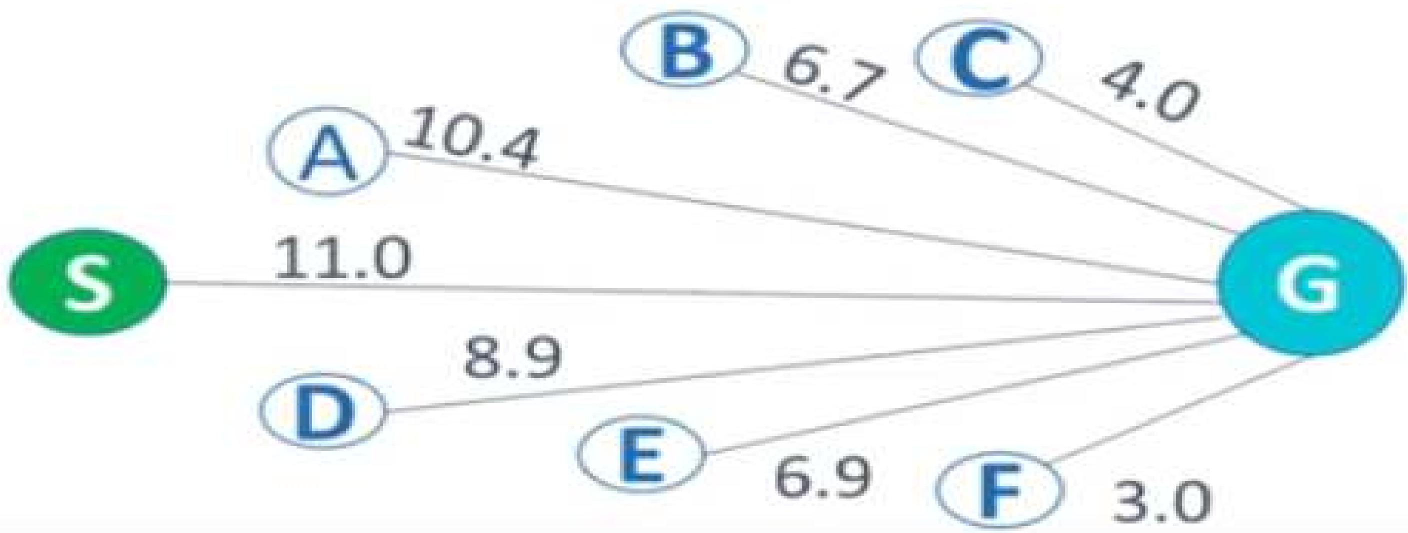
- It is a searching algorithm that is used to find the shortest path between an initial and a final point.
- It is a handy algorithm that is often used for map traversal to find the shortest path to be taken. A* was initially designed as a graph traversal problem, to help build a robot that can find its own course. It still remains a widely popular algorithm for graph traversal.

Example:



G(n):تكلفة الوصول من نقطة البداية إلى النقطة الحالية

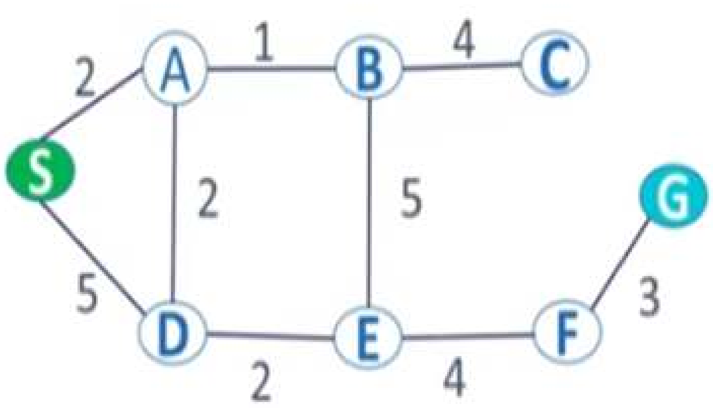
A* Algorithm

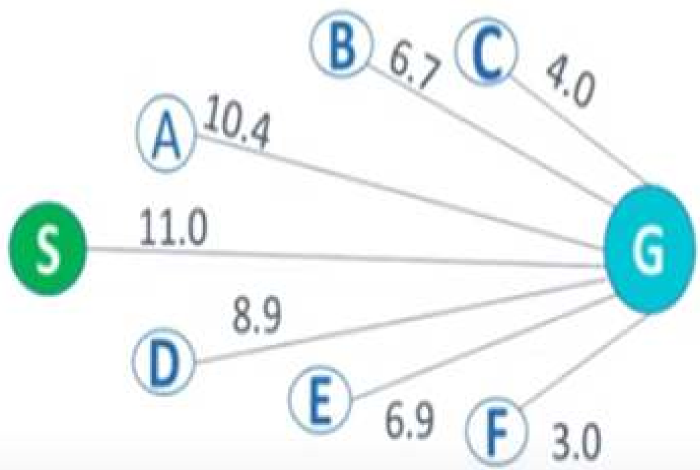


H(n):قيمة تقديرية لبعد النقطة الحالية عن الهدف

A* Algorithm

F(n)=G(n)+H(n)

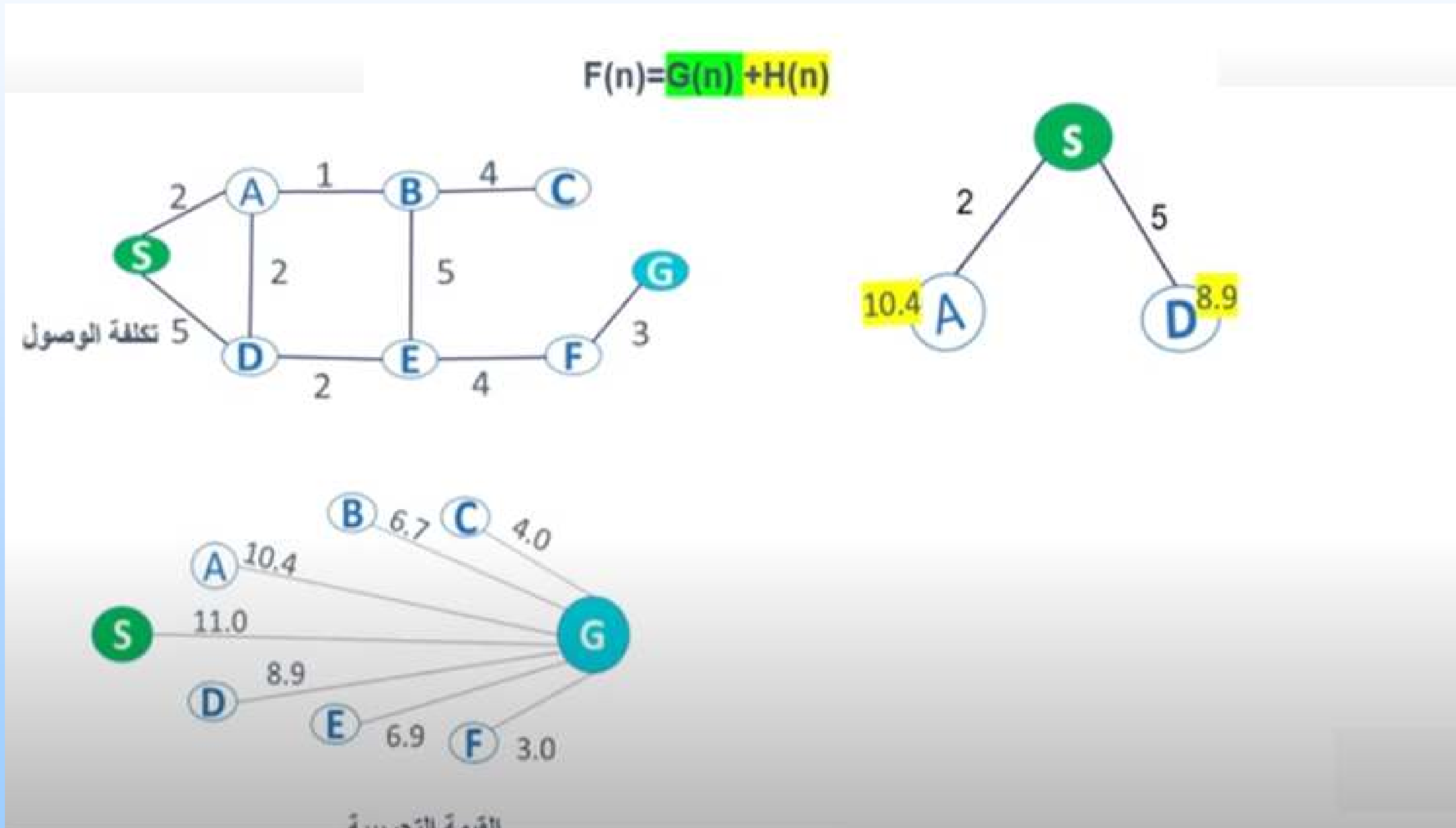




G(n):تكلفة الوصول من نقطة البداية إلى النقطة الحالية

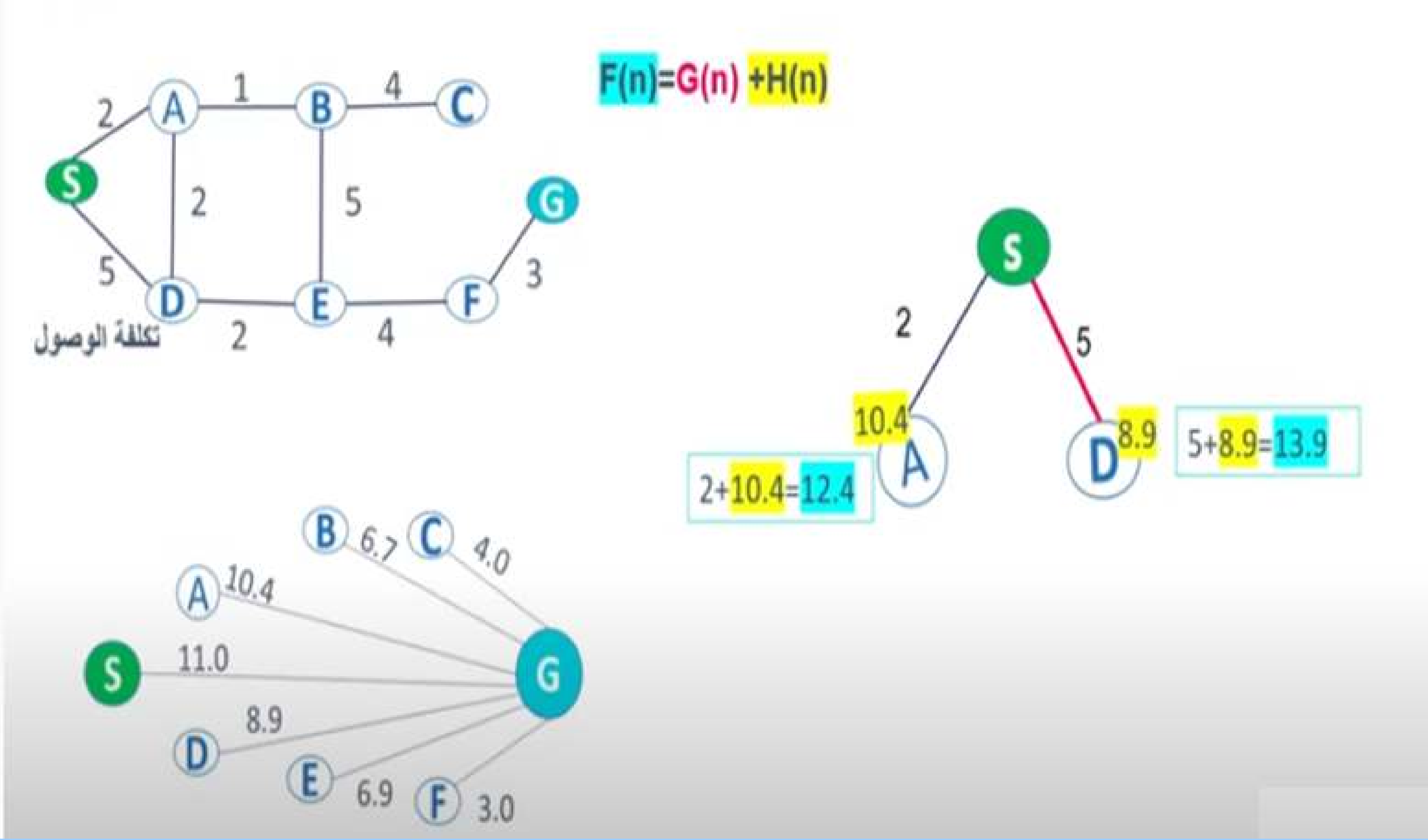
H(n):قيمة تجريبية لبعد النقطة الحالية عن الهدف

A^* Algorithm

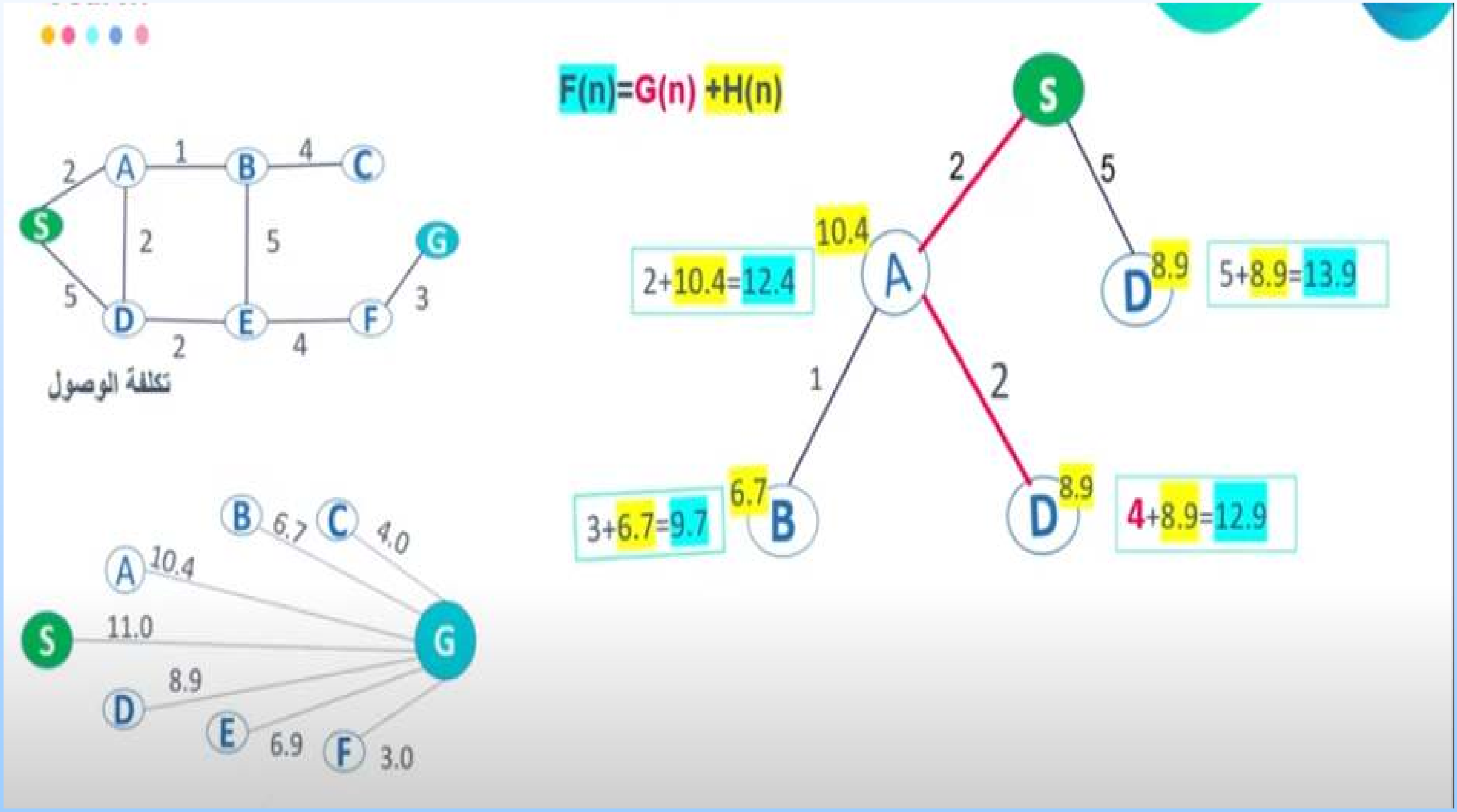


F(n)=G(n) +H(n)
S A B C D E F G
2 1 4 2 5 5 2 4 3
5 تكلفة الوصول
S 2 5 A D 10.4 8.9
B 6.7 C 4.0 A 10.4 S 11.0 G D 8.9 E 6.9 F 3.0

A* Algorithm

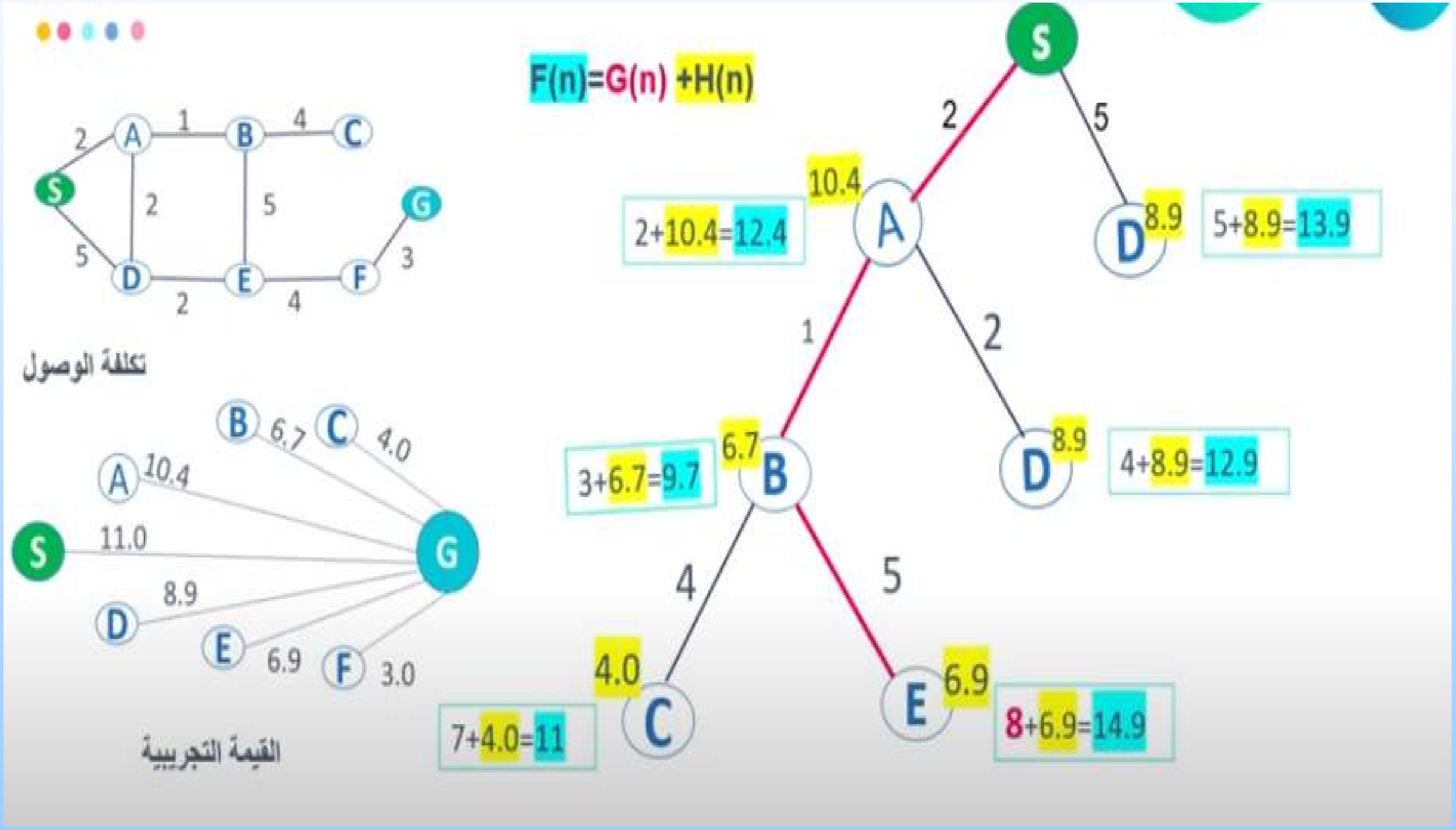


A* Algorithm



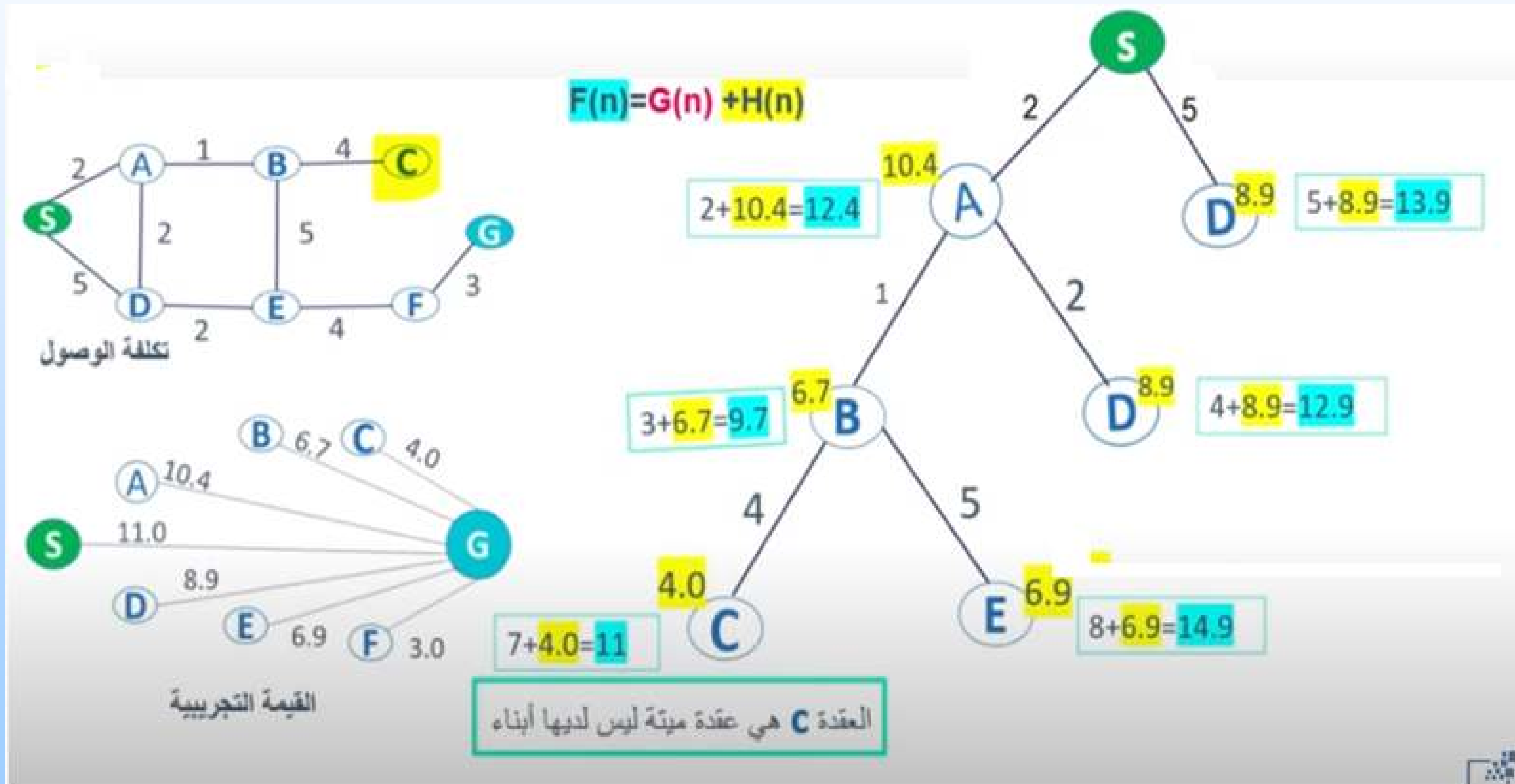
F(n)=G(n) +H(n)
S A B C D E F G
2 1 4 2 5 5 2 4 3
تكلفة الوصول
10.4 6.7 4.0 11.0 8.9 6.9 3.0
2+10.4=12.4
5+8.9=13.9
3+6.7=9.7
4+8.9=12.9

A* Algorithm



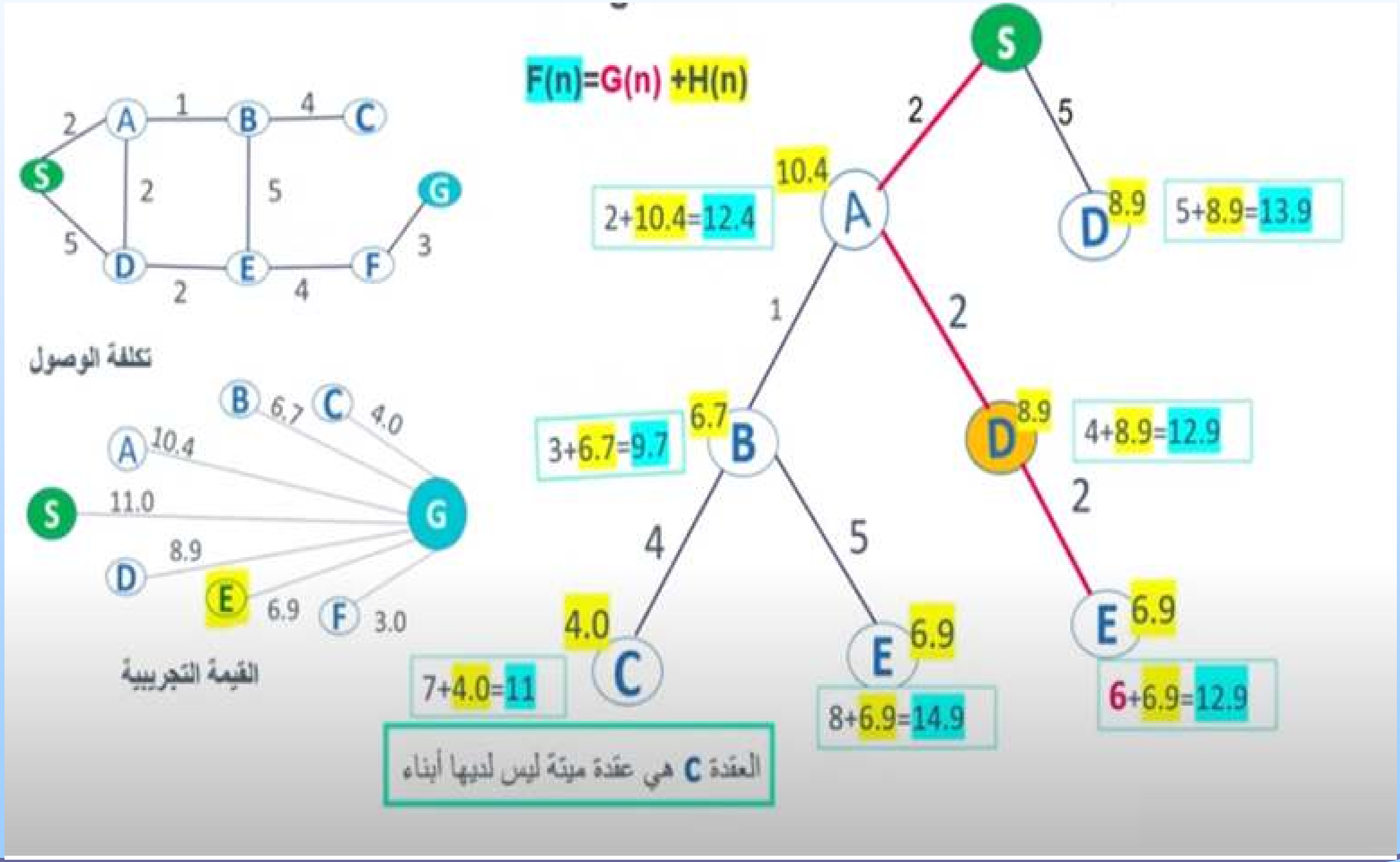
F(n)=G(n) +H(n)
تكلفة الوصول
القيمة التجريبية
2+10.4=12.4
5+8.9=13.9
3+6.7=9.7
4+8.9=12.9
7+4.0=11
8+6.9=14.9

A* Algorithm

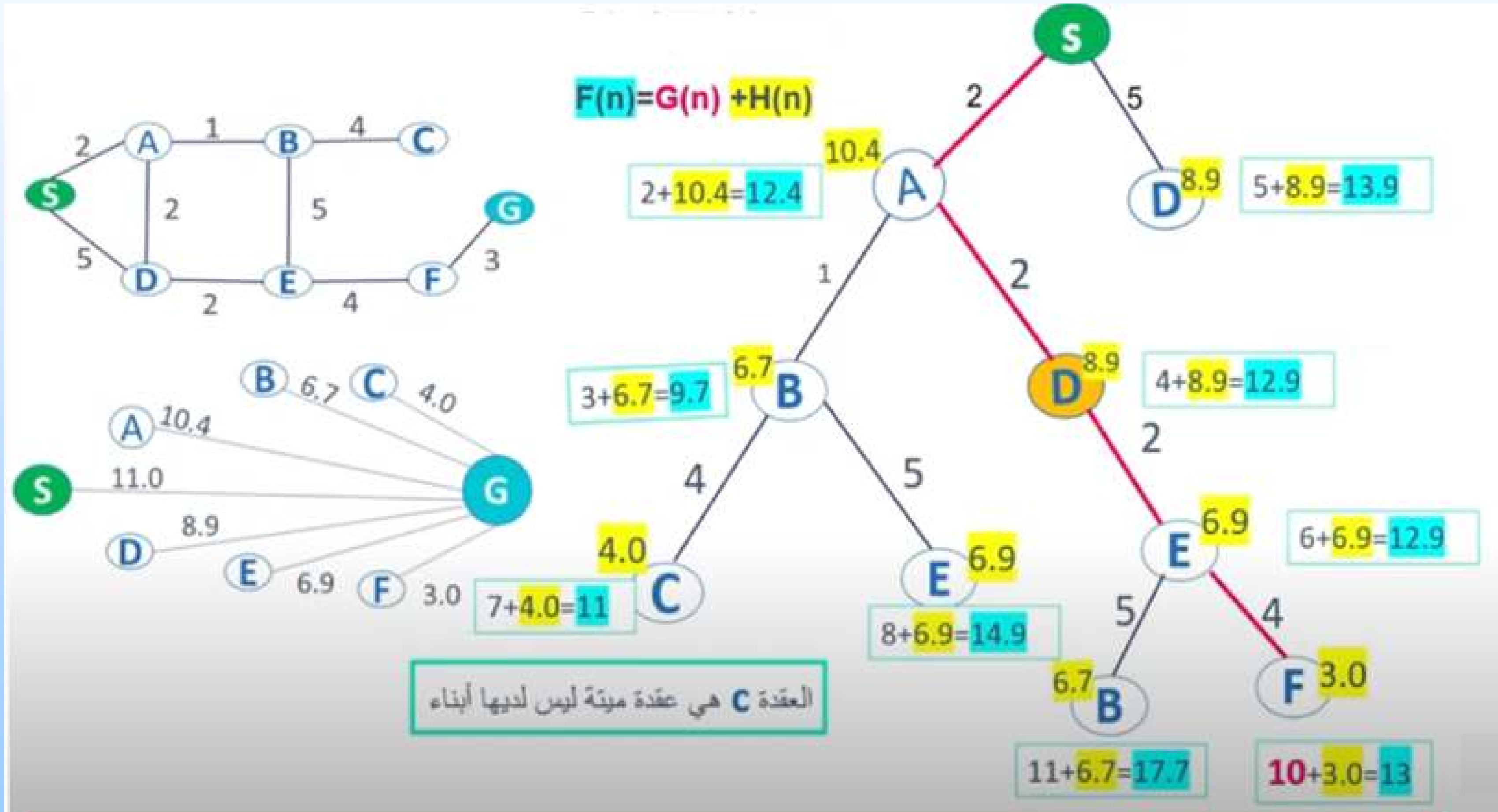


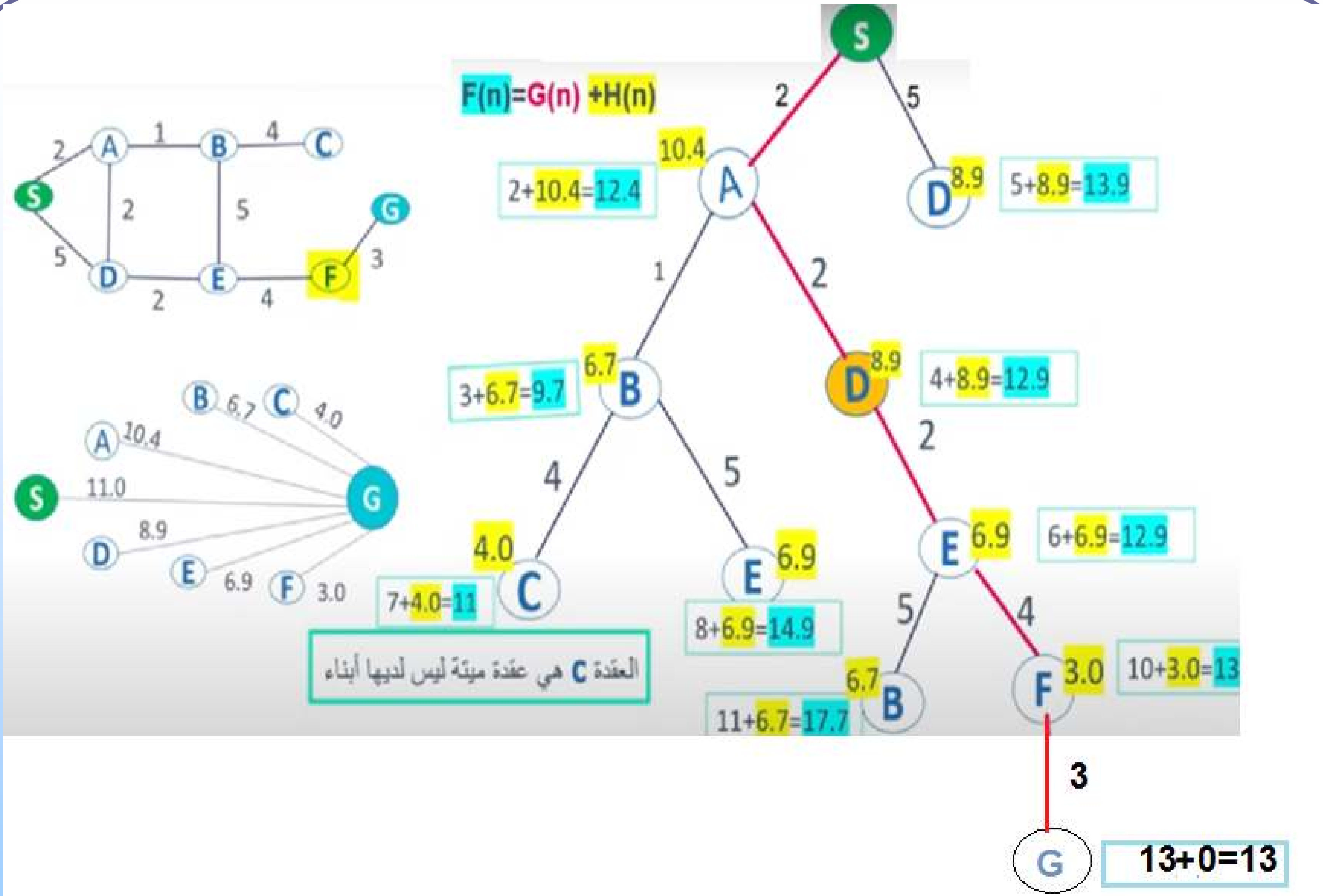
F(n)=G(n) +H(n)
تكلفة الوصول
القيمة التجريبية
2+10.4=12.4
5+8.9=13.9
3+6.7=9.7
4+8.9=12.9
7+4.0=11
8+6.9=14.9
العقدة C هي عقدة ميتة ليس لديها أبناء

A* Algorithm


F(n)=G(n) +H(n)
تكلفة الوصول
القيمة التجريبية
2+10.4=12.4
5+8.9=13.9
3+6.7=9.7
4+8.9=12.9
7+4.0=11
8+6.9=14.9
6+6.9=12.9
العقدة C هي عقدة ميتة ليس لديها أبناء

A* Algorithm


F(n)=G(n) +H(n)
S
A
B
C
D
E
F
G
2
1
4
2
5
5
2
4
3
10.4
6.7
4.0
11.0
8.9
6.9
3.0
2+10.4=12.4
5+8.9=13.9
3+6.7=9.7
4+8.9=12.9
7+4.0=11
8+6.9=14.9
6+6.9=12.9
11+6.7=17.7
10+3.0=13
العقدة C هي عقدة ميتة ليس لديها أبناء


F(n)=G(n) +H(n)
2+10.4=12.4
5+8.9=13.9
3+6.7=9.7
4+8.9=12.9
7+4.0=11
8+6.9=14.9
6+6.9=12.9
11+6.7=17.7
10+3.0=13
13+0=13
العقدة C هي عقدة ميتة ليس لديها أبناء

The Genetic Algorithms

- **Genetic algorithms are used to generate high quality solutions to optimization and search problems by depending on biologically operators such as mutation, crossover and selection.**

The Genetic Algorithms

- In a genetic algorithm, a population of candidate solutions to an optimization problem is evolved toward better solutions. Each candidate solution as a chromosome which can be represented in binary as strings of 0s and 1s, or other encodings.

The Components of a GA

- Initial Population.
- Fitness Function
- Reproduction
- Mutation

Initial Population

Population Initialization is the first step in the genetic algorithm process. Population is a subset of solutions in the current generation. Population can also be defined as a set of chromosomes and usually created randomly.

Chromosomes can be represented as:

- **Bit strings (01010001……..111100)**
- **Real numbers (43.2 -33.1 …. 0.0 89.2)**
- **Permutations of element (E11 E3 E7 ...... E1 E15)**
- **Lists of rules (R1 R2 R3 ..... R22 R23)**
- **any data structure …………………………..**

Fitness Function

- In the fields of genetic algorithms, each design solution is commonly represented as a string of numbers (referred to as a chromosome).
- After each round of testing, or simulation, the idea is to delete the n worst design solutions, and to breed n new ones from the best design solutions.

Fitness Function

- Each design solution, therefore, needs to be awarded a figure of merit, to indicate how close it came to meeting the overall specification, and this is generated by applying the fitness function to the test, or simulation, results obtained from that solution.

Reproduction

Crossover

- Two parents produce two offspring
- There is a chance that the chromosomes of the two parents are copied unmodified as offspring
- There is a chance that the chromosomes of the two parents are randomly recombined (crossover) to form offspring
- Generally the chance of crossover is between 0.6 and 1.0

Reproduction

Crossover

Generating offspring from two selected parents by:

1. Single point crossover
2. Two point crossover
3. Uniform crossover

Reproduction

One-point Crossover

- Randomly one position in the chromosomes is chosen

Randomly chosen position

parent	1	1	0	1	0	0	1	0	1

parent	1	0	1	0	1	1	0	0	1

child	1	1	0	1	1	1	0	0	1
child	1	0	1	0	0	0	1	0	1

Reproduction

Two-points crossover

- Randomly two positions in the chromosomes are chosen

Randomly chosen position

parent	1	1	0	1	0	0	1	0	1
parent	1	0	1	0	1	1	0	0	1
child	1	1	0	0	1	1	1	0	1
child	1	0	1	1	0	0	0	0	1

Reproduction

Uniform crossover

- A random mask is generated
- The mask determines which bits are copied from one parent and which from the other parent
- Bit density in mask determines how much material is taken from the other parent (takeover parameter)

Mask : 0110011000 (Randomly generated)

Parents : 0 01 10 10 010 1 01 00 01 110

Offspring: 0011001010 1010010110

Mutation

- There is a chance that a gene of a child is changed randomly.
- Generally the chance of mutation is low (e.g. 0.001)

child	1	1	0	1	1	1	0	0	1

child	1	1	0	1	0	1	0	0	1

Algorithm

BEGIN

1. **Generate an initial population;**
2. **Compute the fitness function of each individual;**
3. **REPEAT /* New generation /***
4. **FOR population size; DO**

    Select two parents from old generation;

    Recombine parents for two offspring;

    Compute fitness of offspring;

    Insert offspring in new generation

  END FOR

5. **UNTIL population has converged**

END

Algorithm

- In the genetic algorithm process is as follows:
- Step 1.
  Determine the number of chromosomes, generation, and mutation rate and crossover rate value
- Step 2.
  Generate chromosome-chromosome number of the population, and the initialization value of the genes chromosome-chromosome with a random value
- Step 3.
  Process steps 4-7 until the number of generations is met

Algorithm

- **Step 4.**
  Evaluation of fitness value of chromosomes by calculating objective function
- **Step 5. Chromosomes selection**
- **Step 6. Crossover operation**
- **Step 7. Mutation operation**
- **Step 8. Solution (Best Chromosomes)**

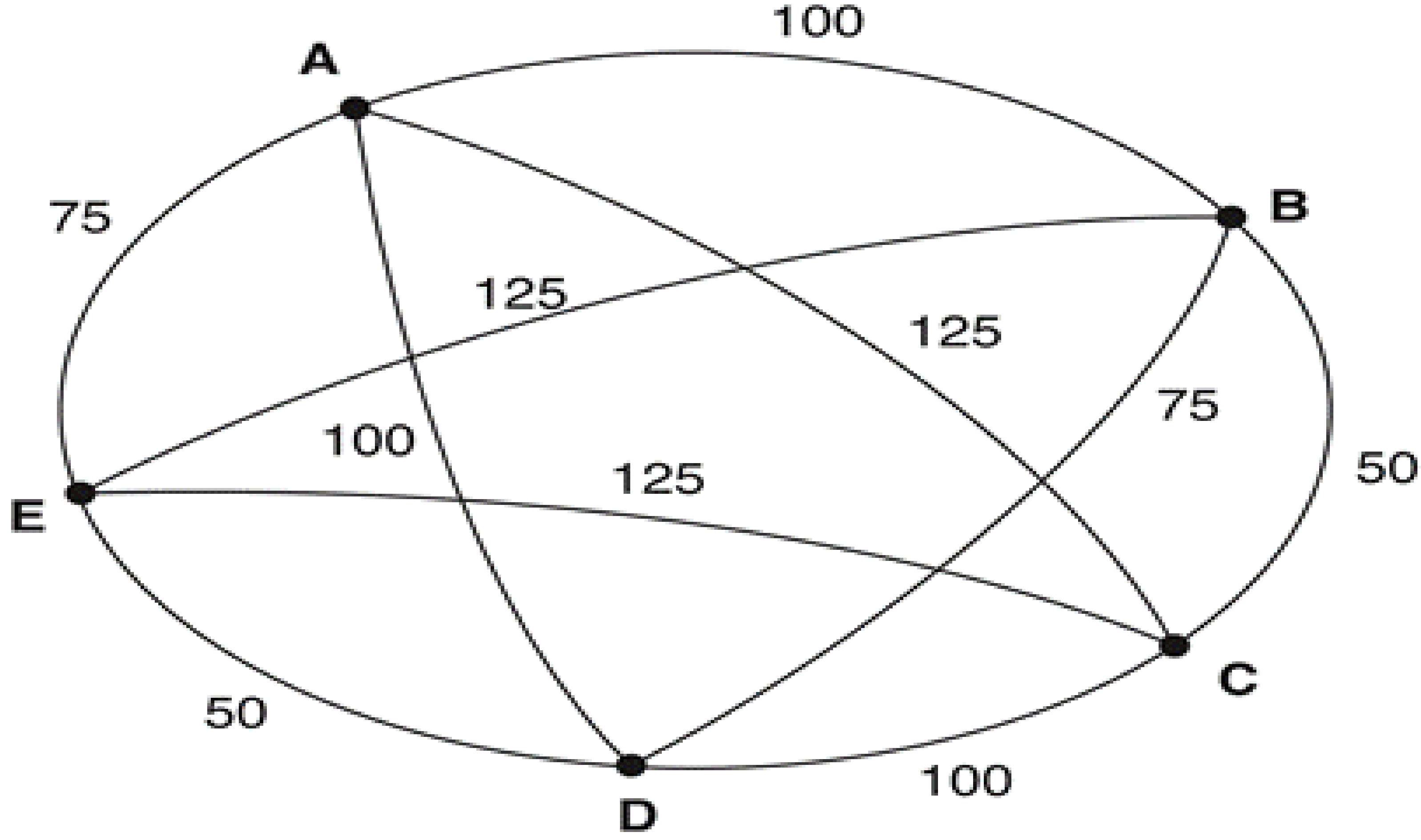
Example: 1

The Traveling Salesman Problem:

Find a tour of a given set of cities so that

- each city is visited only once
- the total distance traveled is minimized

Example: 1



Example: 1

1. **Representation**

The names of cities are A, B, C, D, E can be represented by numbers as follows:

1) A 2) B 3) C 4) D 5) E

Example: 1

2. Initialization

- Input the number of cities n, and their distances that are between them.
- Randomly generate a chromosome x as the following figure
- Check if x represents a real candidate path, i.e. begin and ends at the same city and contains all cities .
- Repeat from 2 to 3 to generate population size chromosomes.

1	3	4	5	2	1

Example:1

3. **Crossover Operation**

Select two chromosomes randomly as following:

			*		*	
Parent 1:	(1	2	4	5	3	1)
Parent 2:	(1	2	5	4	3	1)
Child	(1	2	4	5	3	1)

This operator is called the Order1 crossover.

Example:1

4. **Mutation Operation**

- **Mutation involves reordering of the list:**

		*		*		
Before:	(1	2	4	5	3	1)
After:	(1	5	4	2	3	1)

Example:1

5. **Compute the total cost**

- **Compute the total cost of he candidate solution by using the following equation.**

$$Cost\ (P) = \sum_{i} d_{ij}$$

The cost of the path P is the sum of the distances between the cities

Example:1

6. Repeat

- **Repeat the steps from 3 to 5 until get path of minimum cost.**

Example: 2

Suppose there is equality

$$a + 2b + 3c + 4d = 30 \qquad (1),$$

genetic algorithm will be used to find the value of a, b, c, and d that satisfy the above equation. First we should formulate.

Example:2

The objective function is minimizing the value of function f(x) where $f(x) = ((a + 2b + 3c + 4d) - 30)$.

Since there are four variables in the equation, namely a, b, c, and d, we can compose the chromosome as follow:

Example:2

- To speed up the computation, we can restrict that the values of variables a, b, c, and d are integers between 0 and 30.
- That is:

$$\begin{Bmatrix} 0 \le a \le 30 \\ 0 \le b \le 30 \\ 0 \le c \le 30 \end{Bmatrix} \text{-----------------------------} \rightarrow \quad (1)$$

Example:2

1. **Initialization:**

- **Randomly generate a chromosome x as the following figure:**
- **Check if the genes of x satisfy the condition (1) .**
- **Repeat from 2 to 3 to generate population size chromosomes.**

4	16	5	0

The form of chromosome

Example:2

2. **Crossover Operation**

Select two chromosomes randomly as following:

Cut point

Parent 1:	**(4 16 5 0)**
Parent 2:	**(10 2 5 4)**
Child	**(4 16 5 4)**

This operator is called the One point crossover.

Example:2

3. **Mutation Operation**

- **Mutation involves reordering of the list:**

	*		*	
Before:	**(4**	**16**	**5**	**4)**
After:	**(5**	**16**	**4**	**4)**

Eample:2

4. **Compute the Total Cost**

- **Compute the value of equation Eq. (1)**

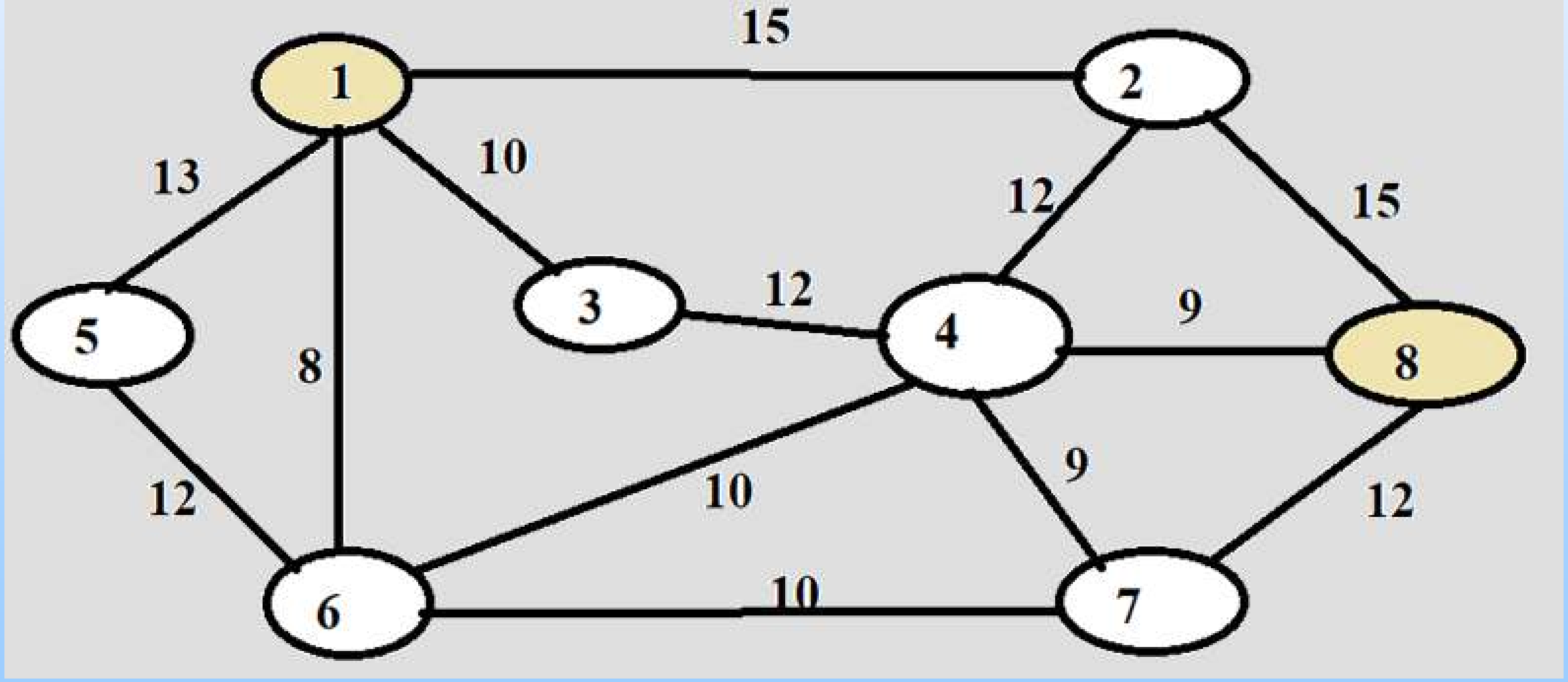
Example: 2

5. **Repeat**

- **Repeat the steps from 2 to 4 until satisfy the equation (1)**

Example:3

- We consider a network with 8 nodes as shows in the following figure. Each link has a corresponding bandwidth.



Example:3

To find the shortest path for the given network in the previous figure, we will follow the following steps:

Example:3

1. Create the connection matrix (con)

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

The connection matrix (con)

Example:3

2. **Create an Initial Population:**

chromosome

1	7	2	4	3	4	5	8
1	4	5	7	3	2	6	8
1	3	6	4	2	5	7	8
1	3	0	0	3	7	2	8
1	4	3	0	0	6	3	8
1	2	0	0	4	5	3	8

Matrix of a pop number chromosomes

Example:3 (create a pop number chromosomes)

```
cin>>dist;   cin>>nd;   cin>>pop;  n=1;
for(;;)
 {
   chrom[n][1]=1;
   for(j=2; j<nd; j++)
     chrom[n][j]=0;
   chrom[n][nd]= dist;
   test=connection_chrom(chrom[n]);
   if(test>0)n++;
```

Example:3 (create a pop number chromosomes)

```
for(;;)
  {
    chrom[n][1]=1;
    j=2;
    for(;;)
    {
      chrom[n][j]=random(8);
      if(chrom[n][j]==1)continue;
      if(j==nd-1)break;
      j++;
    }
```

Example:3 (create a pop number chromosomes)

```
chrom[n][nd]=dist;
sort_chrom(chrom[n]);// Check redundancy
test=connection_chrom(chrom[n]);// Check
 connectivity
if(test==0)continue;
print_chrom(chrom[n]);
if(n==pop)break;
n++;
}
```

Example:3 (Check Redundancy)

```
void sort_chrom(int m[50])
{
 for(int i=2; i<nd; i++)
 {
   if(m[i]==0)continue;
   for(int j=i+1; j<nd; j++)
   {
     if(m[j]==0)continue;
     if(m[i]==m[j]) m[j]=0;
   }
 }
}
```

Example:3 (Check The Connectivity)

```
int connection_chrom(int m[50])
{
   for(int i=2; i<nd; i++)
   {
    if(m[i]==0)continue;
    for(int j=i+1; j<=nd; j++)
    {
     if(m[j]==0)continue;
     if(con[m[i]][m[j]]==0)return 0; else  break;
    }
   }
   return 1;
}
```

Example:3 (Crossover Operation)

```
ng=1;
for(;;)
{
    n=1;
    for(;;)
    {
      float pc=float(random(10))/10;
      if(pc>=0.9)
      {
         for(;;)
           { t1=random(pop);  t2=random(pop);
              if(t1==0 || t2==0)continue; else break;  }
```

Example:3 (Crossover Operation)

```
x=random(7);
for(i=1; i<=x; i++)
  chrom1[n][i]=chrom[t1][i];
for(i=x+1; i<=nd; i++)
  chrom1[n][i]=chrom[t2][i];
}
```

Example:3 (Mutation Operation)

```
float pm=float(random(10))/10;
if(pm<=0.2)  {
  for(;;)
  {y=random(7);
    if(y==1)continue; else break; }
  for(;;)
  {z=random(7);
    if(z==1)continue; else break; }
    int sw= chrom1[n][y];
    chrom1[n][y] = chrom1[n][z];
    chrom1[n][z] = sw;
 }
```

Example:3 (Call compute Bandwidthn)

```
    test=connection_chrom(chrom1[n]);
    if(test==0)continue;
    band=compute_bandwidth(chrom1[n]);
    if (band>=10)  check_chrom(chrom1[n],band);
    if(n==pop)break;
    n++;
  }
  If(ng==600)break;
  ng++;
}
```

Example:3 (Compute the Bandwidth)

```
int compute_bandwidth(int m[50])
{
   int ban=100;
   for(i=1; i<nd; i++)
   {
    if(m[i]==0)continue;
    for(j=i+1; j<=nd; j++)
    {
     if(m[j]==0)continue;
     if(ban>con[m[i]][m[j]])ban=con[m[i]][m[j]];
     break;
    }   }   return ban; }
```

Example:3

- The parameters setting in this algorithm are: pop_size = 20, Pm = 0.2, Pc=0.9, maxgen =600. The source node n_0 is the node no. 1 and the destination node is 8, and the objective value of B is equal to 10.

Example:3

- The shortest path which obtained by the proposed genetic algorithm is shown as the following:

$$1 \longrightarrow 2 \longrightarrow 8$$