Q:1 Write the differences between Certainty and Probability.
Ans. The exact point at which one ceases to be certain is the degree of certainty as opposed to the degree of belief measured as a probability function. Probability is quantified as a number between 0 and 1 (where 0 indicates impossibility and 1 indicates certainty). The higher the probability of an event, the more certain we are that the event will occur. The words “certainty” and “probability” do not apply to propositions that are either true or false. These propositions entertained by us with suspended judgment should never be qualified as either certain or probable. In American common law, there are degrees of certainty and doubt. Certainty attaches to judgments beyond the shadow of doubt; not certain are judgments made with a reasonable doubt, and less certain still are judgments made by a preponderance of the evidence. The last two are judgments to which some degree of probability must be attached, the former more probable, the latter less probable. The propositions in each of these two cases, when entertained with suspended judgment, are either true or false. Certainty and probability qualify our judgments about the matters under consideration on the propositions entertained with suspended judgment. This statement brings us to consider what happens by chance and what is causally determined. Here we must distinguish between the mathematical theory of probability and the philosophical theory of what happens by chance.

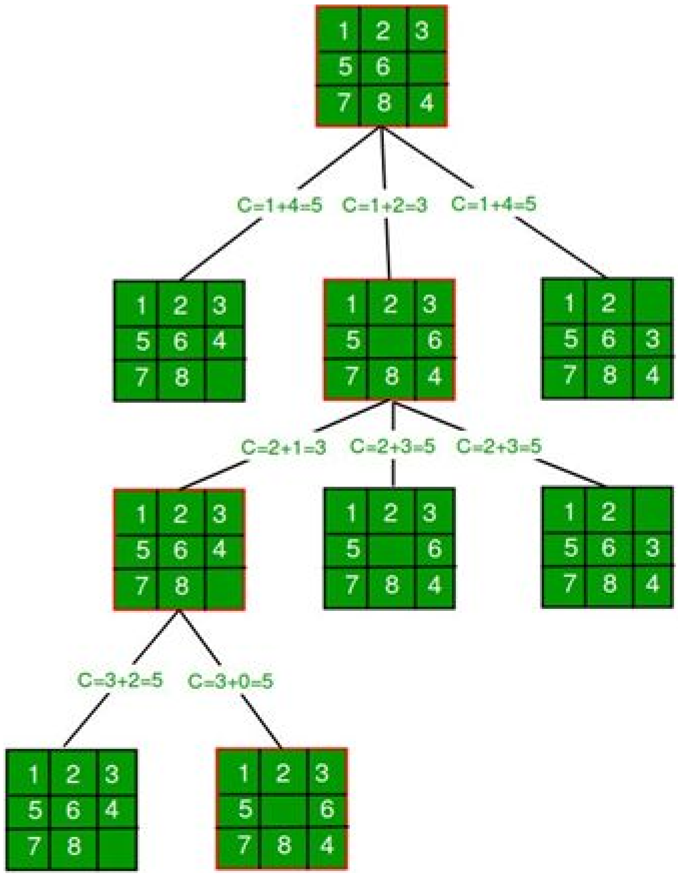
Q:2 Difference between complete and admissible algorithm.
Ans:
- Completeness: An algorithm is complete if it terminates with a solution when one exists.
- Admissibility: An algorithm is admissible if it is guaranteed to return an optimal solution whenever a solution exists.

Depth-First Search	Completeness: not guaranteed in general Admissibility: not guaranteed.
Breadth-First Search	Completeness: guaranteed. Admissibility: the algorithm will always find the shortest path
Depth-First Search Iterative-Deepening	Completeness: guaranteed. Admissibility: the algorithm will always find the shortest path

Q:3 Compute the complexity of the 8 puzzle problem.

Ans: Given a 3×3 board with 8 tiles (every tile has one number from 1 to 8) and one empty space. The objective is to place the numbers on tiles to match final configuration using the empty space. We can slide four adjacent (left, right, above and below) tiles into the empty space.

In this solution, successive moves can take us away from the goal rather than bringing closer. The search of state space tree follows leftmost path from the root regardless of initial state. An answer node may never be found in this approach.



<u>**Time Complexity= 9!**</u>

Q:4 Write the algorithm for IDFS.

Ans:
```
function IDFS(root)
        for depth from 0 to ∞
              found, remaining <-DLS(root,depth)
                 if found ≠ null
                       return found
                 else if not remaining
                       return null
    function DLS(node,depth)
         if depth= 0
                if node is goal
                      return (node,true)
                 else
                       retun(null,true)
         else if depth>0
                 any_remaining <- false
                 foreach child of node
                       found, remaining <- DLS(child,depth-1)
                             if found ≠ null
                                   return(found,true)
                             if remaining
                                   any_remaining <- true
                 return (null,any_remaining)
```

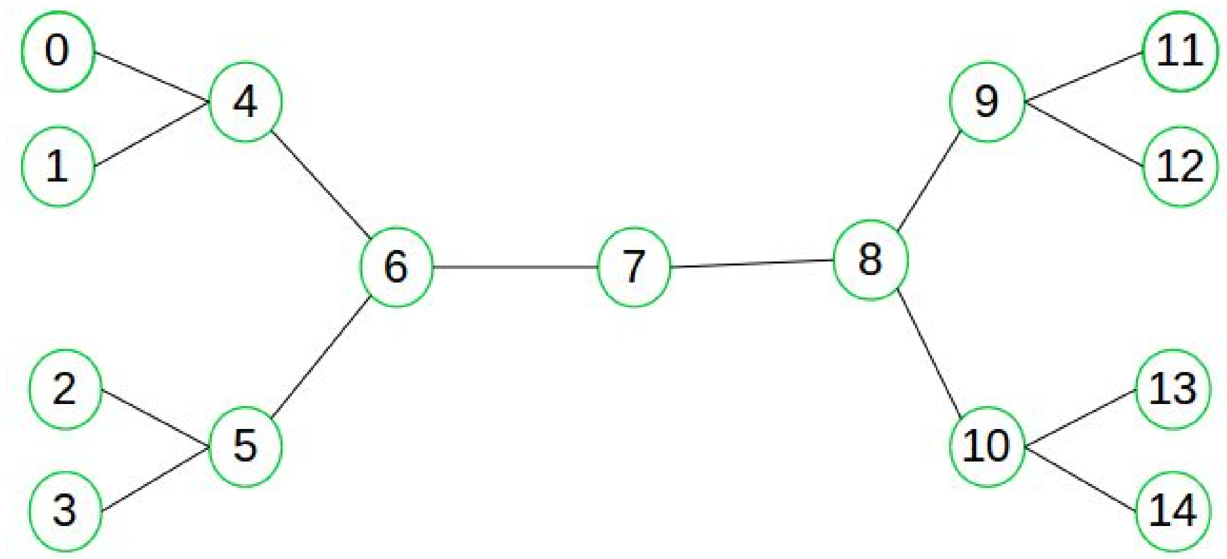
Q:5 Write the algorithm for bidirectional search.

Ans: Bidirectional search is a graph search algorithm which find shortest path from source to goal vertex. It runs two simultaneous search –

1. Forward search form source/initial vertex toward goal vertex
2. Backward search form goal/target vertex toward source vertex

Bidirectional search replaces single search graph(which is likely to grow exponentially) with two smaller sub graphs – one starting from initial vertex and other starting from goal vertex.

The search terminates when two graphs intersect.
Just like A* algorithm, bidirectional search can be guided by a heuristic estimate of remaining distance from source to goal and vice versa for finding shortest path possible. Consider following simple example-



Suppose we want to find if there exists a path from vertex 0 to vertex 14. Here we can execute two searches, one from vertex 0 and other from vertex 14. When both forward and backward search meet at vertex 7, we know that we have found a path from node 0 to 14 and search can be terminated now. We can clearly see that we have successfully avoided unnecessary exploration.

Why bidirectional approach?
Because in many cases it is faster, it dramatically reduce the amount of required exploration. Suppose if branching factor of tree is **b** and distance of goal vertex from source is **d**, then the normal BFS/DFS searching complexity would be $O(b^d)$. On the other hand, if we execute two search operation then the complexity would be $O(b^{d/2})$ for each search and total complexity would be $O(b^{d/2} + b^{d/2})$ which is far less than $O(b^d)$
When to use bidirectional approach?
We can consider bidirectional approach when-
1. Both initial and goal states are unique and completely defined.
2. The branching factor is exactly the same in both directions.

Performance measures
- Completeness : Bidirectional search is complete if BFS is used in both searches.
- Optimality : It is optimal if BFS is used for search and paths have uniform cost.

- Time and Space Complexity : Time and space complexity is $O(b^{d/2})$

Q:6 Write the algorithm for optimal A* search technique.

Ans:

```
1.Initialize the open list
2.  Initialize the closed list
        put the starting node on the open
        list (you can leave its f at zero)

3.  while the open list is not empty
        a) find the node with the least f on
        the open list, call it "q"

        b) pop q off the open list

        c) generate q's 8 successors and set their
        parents to q

        d) for each successor
        i) if successor is the goal, stop search
        successor.g = q.g + distance between
                              successor and q
        successor.h = distance from goal to
        successor (This can be done using many
         ways, we will discuss three heuristics-
        Manhattan, Diagonal and Euclidean
        Heuristics)

        successor.f = successor.g + successor.h

        ii) if a node with the same position as
         successor is in the OPEN list which has a
        lower f than successor, skip this successor

        iii) if a node with the same position as
         successor  is in the CLOSED list which has
        a lower f than successor, skip this successor
```

```
                   otherwise, add  the node to the open list
                  end (for loop)

                  e) push q on the closed list
                  end (while loop)
```

Q:7 Explain the algorithm for IDA*.

Ans:

```
root=initial node;
Goal=final node;
function IDA*()
{
threshold=heuristic(Start);
while(1)
 {
integer temp=search(Start,0,threshold); //function search(node,g score,threshold)
if(temp==FOUND)
      return FOUND;
if(temp== ∞)
      return;
threshold=temp;
}
}
function Search(node, g, threshold)
{
f=g+heuristic(node);
if(f>threshold)
      return f;
if(node==Goal)
      return FOUND;
integer min=MAX_INT;
foreach(tempnode in nextnodes(node))
{
integer temp=search(tempnode,g+cost(node,tempnode),threshold);
if(temp==FOUND)
return FOUND;
if(temp<min)
 threshold encountered
min=temp;
}
return min;  //return the minimum 'f' encountered greater than threshold
}
function nextnodes(node)
{
        return list of all possible next nodes from node;
}
```

Q:8 Explain resolution principle.

Ans: In mathematical logic and automated theorem proving, **resolution** is a rule of inference leading to a refutation theorem-proving technique for sentences in propositional logic and first-order logic. In other words, iteratively applying the resolution rule in a suitable way allows for telling whether a propositional formula is satisfiable and for proving that a first-order formula is unsatisfiable. Attempting to prove a satisfiable first-order formula as unsatisfiable may result in a nonterminating computation; this problem doesn't occur in propositional logic.

RESOLUTION IN PROPOSITIONAL LOGIC:

Resolution rule:

The **resolution rule** in propositional logic is a single valid inference rule that produces a new clause implied by two clauses containing complementary literals. A literal is a propositional variable or the negation of a propositional variable. Two literals are said to be complements if one is the negation of the other. The resulting clause contains all the literals that do not have complements. Formally:

$$\frac{a_1 \lor a_2 \lor \cdots \lor c, \quad b_1 \lor b_2 \lor \cdots \lor \neg c}{a_1 \lor a_2 \lor \cdots \lor b_1 \lor b_2 \lor \cdots}$$

where,
All a_i, b_i and c are literals,
the dividing line stands for "entails".

The above may also be written as:

$$\frac{(\neg a_1 \wedge \neg a_2 \wedge \cdots) \rightarrow c, \quad c \rightarrow (b_1 \vee b_2 \vee \cdots)}{(\neg a_1 \wedge \neg a_2 \wedge \cdots) \rightarrow (b_1 \vee b_2 \vee \cdots)}$$

The clause produced by the resolution rule is called the *resolvent* of the two input clauses. It is the principle of consensus applied to clauses rather than terms. When the two clauses contain more than one pair of complementary literals, the resolution rule can be applied (independently) for each such pair; however, the result is always a tautology.

Modus ponens can be seen as a special case of resolution (of a one-literal clause and a two-literal clause).

$$\frac{p \rightarrow q, \quad p}{q}$$

can be considered equal to

$$\frac{\neg p \vee q, \quad p}{q}$$

Resolution Technique:

When coupled with a complete search algorithm, the resolution rule yields a sound and complete algorithm for deciding the *satisfiability* of a propositional formula, and, by extension, the validity of a sentence under a set of axioms.

This resolution technique uses proof by contradiction and is based on the fact that any sentence in propositional logic can be transformed into an equivalent sentence in conjunctive normal form. The steps are as follows.

- All sentences in the knowledge base and the *negation* of the sentence to be proved (the *conjecture*) are conjunctively connected.
- The resulting sentence is transformed into a conjunctive normal form with the conjuncts viewed as elements in a set, *S*, of clauses.

  For example $(A_1 \vee A_2) \wedge (B_1 \vee B_2 \vee B_3) \wedge (C_1)$ gives rise to the set $S = \{A_1 \vee A_2, B_1 \vee B_2 \vee B_3, C_1\}$
- The resolution rule is applied to all possible pairs of clauses that contain complementary literals. After each application of the resolution rule, the resulting sentence is simplified by removing repeated literals. If the sentence contains complementary literals, it is discarded (as a tautology). If not, and if it is not yet present in the clause set *S*, it is added to *S*, and is considered for further resolution inferences.
- If after applying a resolution rule the *empty clause* is derived, the original formula is unsatisfiable (or *contradictory*), and hence it can be concluded that the initial conjecture follows from the axioms.

- If, on the other hand, the empty clause cannot be derived, and the resolution rule cannot be applied to derive any more new clauses, the conjecture is not a theorem of the original knowledge base.

Q:9 Explain various operators over the fuzzy set.

Ans:

Given ‘X’ to be universe of discourse, A and B are two fuzzy sets with membership function μA(x) and μB(x) then,

1. Union:

The union of two fuzzy sets A and B is a new fuzzy set A ∪ B also on ‘X’ with membership function defined as follow:

$$\mu_{(AUB)} = \max(\mu_A(x), \mu_B(x)) = \mu_A(x)\hat{}\mu_B(x)$$

2. Intersection:

Intersection of fuzzy sets A & B, is a new fuzzy set A ∩ B also on ‘X’ whose membership function is defined by

$$\mu_{(A\cap B)} = \min\left(\mu_A(x), \mu_B(x)\right) = \mu_A(x) \underset{\substack{\downarrow \\ \text{minimum operator}}}{\wedge} \mu_B(x)$$

3. Compliment:

Compliment of a fuzzy set A is A with membership function

$$\mu_{\overline{A}}(x) = 1 - \mu_A(x)$$

4. Product of Two Fuzzy Sets:

The product of two fuzzy sets A & B is a new fuzzy set A.B with membership function:

$$\mu_{A.B}(x) = \mu_A(x).\mu_B(x)$$

5. Equality:

Two fuzzy sets A and B are said to be equal i.e, A = B if and only if μA(x) = μB(x) Which means their membership values must be equal.

6. Product of Fuzzy Sets with a Crisp Number:

Multiplying a fuzzy set A by a crisp number ‘n’ results in a new fuzzy set n.A, whose membership function is

$$\mu_{n\,.\,A}(x) = n\,.\,\mu_A(x)$$

7. Power of a Fuzzy Set:

The alpha power of a fuzzy set A is a new fuzzy set A^α whose membership function is: that is, individual memberships power of α

$$\mu_{A^\alpha}(x) = \left[\mu_A(x)\right]^\alpha$$

8. Difference of Fuzzy Sets

The differences of two fuzzy sets A and B is a new fuzzy set A-B which is defined as

$$A - B = \left(A \cap \overline{B}\right)$$

9. Disjunctive Sum of A & B

It is the new fuzzy set defined as follow:

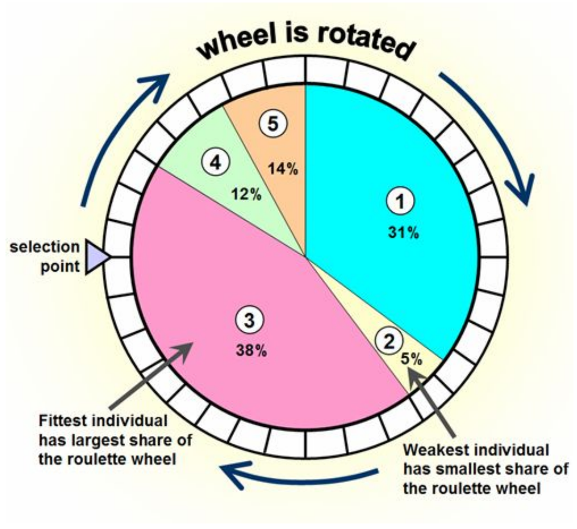
$$A \oplus B = \left(\overline{A} \cap B\right) \cup \left(A \cap \overline{B}\right)$$

Q:10 Types of Selection.

Ans.

Roulette wheel selection

The basic part of the selection process is to stochastically select from one generation to create the basis of the next generation. The requirement is that the fittest individuals have a greater chance of survival than weaker ones. This replicates nature in that fitter individuals will tend to have a better probability of survival and will go forward to form the mating pool for the next generation. Weaker individuals are not without a chance. In nature such individuals may have genetic coding that may prove useful to future generations.


wheel is rotated
selection point
1
31%
2
5%
3
38%
4
12%
5
14%
Fittest individual has largest share of the roulette wheel
Weakest individual has smallest share of the roulette wheel

Tournament Selection

Tournament Selection is a Selection Strategy used for selecting the fittest candidates from the current generation in a Genetic Algorithm. These selected candidates are then passed on to the next generation. In a K-way tournament selection, we select k-individuals and run a tournament among them. Only the fittest candidate amongst those selected candidates is chosen and is passed on to the next generation. In this way many such tournaments take place and we have our final selection of candidates who move on to the next generation. It also has a parameter called the selection pressure which is a probabilistic measure of a candidate's likelihood of participation in a tournament. If the tournament size is larger, weak candidates have a smaller chance of getting selected as it has to compete with a stronger candidate.

Algorithm

1.Select k individuals from the population and perform a tournament amongst them

2.Select the best individual from the k individuals

3. Repeat process 1 and 2 until you have the desired amount of population

Q:11 Types of crossover.

Ans.

Crossover with Reduced Surrogate

This technique can limits the crossover points that can be chosen to positions where the values in the two parents differ. As a result, positions where the values are swapped will always cause a change, and hence produce new individuals whenever possible. This technique is performed in conjunction with the others

Arithmetic Crossover

Arithmetic crossover is used in case of real-value encoding. Arithmetic crossover operator linearly combines the two parent chromosomes [17]. Two chromosomes are particular randomly for crossover and create two offspring's which are linear mixture of their parents.

Partially mapped Crossover

Partially Matched or Mapped Crossover (PMX) is the most frequently used crossover operator. It was proposed by Goldberg and Lingle [19] for Travelling Salesman Problem. In Partially Matched Crossover, two chromosomes are associated and two crossover sites are chosen arbitrarily. The fraction of chromosomes between the two crossover points gives a corresponding selection that undergoes the crossover process through position-byposition exchange operations [2, 17]. PMX tends to respect the absolute positions.

Shuffle Crossover

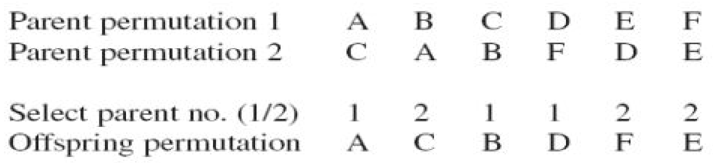
Shuffle Crossover helps in creation of offspring which have independent of crossover point in their parents. It uses the same 1-Point Crossover technique in addition to shuffle. Shuffle Crossover selects the two parents for crossover. It firstly randomly shuffles the genes in the both parents but in the same way. Then it applies the 1-Point crossover technique by randomly selecting a point as crossover point and then combines both parents to create two offspring. After performing 1-point crossover the genes in offspring are then unshuffled in same way as they have been shuffled.

Precedence Preservative Crossover (PPX)

Precedence Preservative Crossover was independently developed for vehicle routing problems by Blanton and Wainwright (1993) and for scheduling problems by Bierwirth et al. (1996). The operator passes on precedence relations of operations given in two parental permutations to one offspring at the same rate, while no new precedence relations are introduced. PPX is illustrated in below, for a problem consisting of six operations A–F. The operator works as follows:

• A vector of length Sigma, sub i=1 to m, representing the number of operations involved in the problem, is randomly filled with elements of the set {1, 2}.
• This vector defines the order in which the operations are successively drawn from parent 1 and parent 2.

• We can also consider the parent and offspring permutations as lists, for which the operations ‘append’ and ‘delete’ are defined.
• First we start by initializing an empty offspring.
• The leftmost operation in one of the two parents is selected in accordance with the order of parents given in the vector.
• After an operation is selected it is deleted in both parents. 2
• Finally the selected operation is appended to the offspring.
• This step is repeated until both parents are empty and the offspring contains all operations involved.
• Note that PPX does not work in a uniform-crossover manner due to the ‘deletion append’ scheme used.
Example is shown in Figure below



Parent permutation 1	A	B	C	D	E	F
Parent permutation 2	C	A	B	F	D	E
Select parent no. (1/2)	1	2	1	1	2	2
Offspring permutation	A	C	B	D	F	E

Reference:
1. http://www.edc.ncl.ac.uk/highlight/rhjanuary2007g02.php
2. https://www.geeksforgeeks.org/tournament-selection-ga/
3. http://ictactjournals.in/paper/IJSC_V6_I1_paper_4_pp_1083_1092.pdf
4. http://ijcsit.com/docs/Volume%205/vol5issue06/ijcsit2014050673.pdf
5. http://ictactjournals.in/paper/IJSC_V6_I1_paper_4_pp_1083_1092.pdf
6. http://repository.uobabylon.edu.iq/mirror/resources/paper_2_2712_124.pdf
7. http://www.uobabylon.edu.iq/uobcoleges/ad_downloads/4_10592_215.pdf