**1)A_STAR:**

```
from queue import PriorityQueue
def get_weight(A, dit):
    sumi = 0
    for i, ch in enumerate(A):
        sumi += abs(i - dit[ch])
    return sumi
def search(parent, goal, dit):
    if parent == goal: return (True, [goal])
    child_no = len(parent) - 1
    visited, path = set([parent]), []
    pq = PriorityQueue()
    pq.put((110000, 'maxi'))
    A = list(parent)
    pq.put((get_weight(A, dit), parent))
    while pq.qsize() > 1:
        _, S = pq.get()
        path.append(S)
        if S == goal: break
        A = list(S)
        for i in range(child_no):
            A[i], A[i + 1] = A[i + 1], A[i]
            Ch = ''.join(A)
            if Ch not in visited:
                pq.put((get_weight(A, dit), Ch))
                visited.add(Ch)
            A[i], A[i + 1] = A[i + 1], A[i]
    return path
def a_star(start, goal):
    maxi, dit = len(start), {}
    for i, ch in enumerate(goal):
        dit[ch] = min(dit.get(ch, maxi), i)
    result = search(start, goal, dit)
    if result[-1] == goal: return result
    else: return ['No solution exists']
if __name__ == '__main__':
    start = 'HEMA'
    goal = 'MAHE'
    print('Starting ...\n')
    result = a_star(start, goal)
    for i, val in enumerate(result):
        print(f'{i + 1}) {val}')
    print()
```

**18)LENGTH AND REVERSE**

```
a) get_len([],L):-L is 0.
get_len([_|T],L):-
get_len(T,L1),
L is L1+1.

b) reverse([],L,L).
reverse([H|T],X,L):-
reverse(T,X,[H|L]).
```

**2)K-MEANS:**

```
from sklearn import datasets
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
dataset = datasets.load_iris()
x = datasets.load_iris()['data']
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-
means++', max_iter = 300, n_init = 10, random_state
= 0)
    kmeans.fit(x)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('The elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
kmeans = KMeans(n_clusters=3, init='k-means++',
max_iter=300, n_init=10, random_state=0)
y_kmeans = kmeans.fit_predict(x)
plt.scatter(x[y_kmeans == 0, 0], x[y_kmeans == 0, 1],
s = 100, c = 'red', label = 'Iris-setosa')
plt.scatter(x[y_kmeans == 1, 0], x[y_kmeans == 1, 1],
s = 100, c = 'blue', label = 'Iris-versicolour')
plt.scatter(x[y_kmeans == 2, 0], x[y_kmeans == 2, 1],
s = 100, c = 'green', label = 'Iris-virginica')
plt.scatter(kmeans.cluster_centers_[:, 0],
kmeans.cluster_centers_[:,1], s = 100, c = 'yellow',
label = 'Centroids')
plt.legend()
```

**KNN:**

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
irisData = load_iris()

X = irisData.data
y = irisData.target
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size = 0.1, random_state=42)
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train, y_train)
predictions = knn.predict(X_test)
print(knn.predict(X_test))
count = 0
for input, prediction, label in zip(X_test, predictions,
y_test):
  if prediction != label: count += 1
  print(f'{input} has been classified as {prediction}
and should be {label}')
print(f'Efficiency : {count / len(X_test)}')
```

**3)NAÏVE BAYES:**

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn import metrics
iris = datasets.load_iris()
iris.data
X=iris.data
y=iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.5, random_state=9)
nv = GaussianNB()
nv.fit(X_train, y_train)
y_pred = nv.predict(X_test)
for input, prediction, original in zip(X_test, y_pred,
y_test):
  print(f'{input} is predicted as {prediction} which is
{original}')
accuracy_score(y_test, y_pred)
print("acuracy:", metrics.accuracy_score(y_test,
y_pred))
```

**6)LOCATION,STAYS:**

```
location(city1, state1).
location(city2, state2).
location(city3, state3).
location(city4, state4).
location(city5, state5).
location(city6, state6).
location(city7, state7).
stays(person1, city1).
stays(person2, city2).
stays(person3, city3).
stays(person4, city4).
stays(person5, city5).
stays(person6, city6).
stays(person7, city7).
stays(person8, city1).
get_state(X, Y):-
   stays(X, Z),
   location(Z, Y).
get_person(X, Y):-
   location(Z, X),
   stays(Y, Z).
   print(X), nl,
   print(Y), nl,
   print(Z), nl.
```

**4)SUDOKO.PY:**

```
M = 9
def puzzle(a):
   for i in range(M):
      for j in range(M):
         print(a[i][j],end = " ")
      print()
def solve(grid, row, col, num):
   for x in range(9):
      if grid[row][x] == num:
         return False
   for x in range(9):
      if grid[x][col] == num:
         return False
   startRow = row - row % 3
   startCol = col - col % 3
   for i in range(3):
      for j in range(3):
         if grid[i + startRow][j + startCol] == num:
            return False
   return True
def Suduko(grid, row, col):
   if (row == M - 1 and col == M):
      return True
   if col == M:
      row += 1
      col = 0
   if grid[row][col] > 0:
      return Suduko(grid, row, col + 1)
   for num in range(1, M + 1, 1):
      if solve(grid, row, col, num):
         grid[row][col] = num
         if Suduko(grid, row, col + 1):
            return True
      grid[row][col] = 0
   return False
grid = [[2, 5, 0, 0, 3, 0, 9, 0, 1],
      [0, 1, 0, 0, 0, 4, 0, 0, 0],
      [4, 0, 7, 0, 0, 0, 2, 0, 8],
      [0, 0, 5, 2, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 9, 8, 1, 0, 0],
      [0, 4, 0, 0, 0, 3, 0, 0, 0],
      [0, 0, 0, 3, 6, 0, 0, 7, 2],
      [0, 7, 0, 0, 0, 0, 0, 0, 3],
      [9, 0, 3, 0, 0, 0, 6, 0, 4]]

if (Suduko(grid, 0, 0)):
   puzzle(grid)
else:
   print("Solution does not exist:(")
```

**7)a MARRIAGE,EMPLOYEE**

```
employee(spielberg, male, unmarried, us, director).
employee(allen, none, notFound, uk, manager).
employee(lee, female, married, india, supervisor).
status(unmarried, spielberg, male, us, director).
status(notFound, allen, none, uk, manager).
status(married, lee, female, india, supervisor).
gender(male, spielberg, unmarried, us, director).
gender(none, allen, notFound, uk, manager).
gender(female, lee, married, india, supervisor).
find_employee:-
write("enter employee name : "),
read(Input), nl,
employee(Input, O1, O2, O3, O4),
write("name : "), write(Input), nl,
write("Gender : "), write(O1), nl,
write("status : "), write(O2), nl,
write("country : "), write(O3), nl,
write("profession : "), write(O4),nl,
nl.
find_status:-
write("enter employee status: "),
read(Input), nl,
status(Input, O1, O2, O3, O4),
write("status : "), write(Input), nl,
write("name : "), write(O1), nl,
write("gender : "), write(O2), nl,
write("country : "), write(O3), nl,
write("profession : "), write(O4),nl,
nl.
find_gender:-
write("enter employee gender : "),
read(Input), nl,
gender(Input, O1, O2, O3, O4),
write("Gender : "), write(Input), nl,
write("name : "), write(O1), nl,
write("status : "), write(O2), nl,
write("country : "), write(O3), nl,
write("profession : "), write(O4),nl,
nl.
```

**15)travelling salesman:**

```
distance(a, b, 10).
distance(a, c, 20).
distance(a, d, 15).
distance(b, e, 14).
distance(b, f, 25).
distance(c, g, 30).
distance(c, h, 5).
find_distance(X, Y, Z):-
   distance(X, Y, D),
   Z is D.
find_distance(X, Y, Z1):-
   distance(X, Z, D1),
   find_distance(Z, Y, D2),
   Z1 is D1 + D2
```

**7)b.ANCESTORS**

```
male(kiran).
male(raj).
male(arun).
male(jagadeesh).
male(sailesh).
male(hari).
male(rao).
female(harika).
female(santhoshi).
female(lakshmi).
female(jhansi).
parent_of(kiran,lakshmi).
parent_of(harika,jhansi).
parent_of(harika,jagadeesh).
parent_of(jagadeesh,rao).
parent_of(jagadeesh,santhoshi).
parent_of(jhansi,sailesh).
parent_of(arun,sailesh).
parent_of(lakshmi,hari).
ancestor_of(X,Y):-
parent_of(X,Z),
parent_of(Z,Y).
```

**11)DFS:**

```
:-op(500,xfx,'is_parent').
a is_parent b.
a is_parent c.
a is_parent d.
b is_parent e.
b is_parent f.
c is_parent g.
c is_parent h.
c is_parent i.
d is_parent j.
e is_parent k.
f is_parent l.
f is_parent m.
h is_parent n.
i is_parent o.
i is_parent p.
j is_parent q.
j is_parent r.
j is_parent s.
m is_parent t.
getchildren(Parent,Children):-
setof(Child,Parent^is_parent(Parent,Child),Children),
!.
getchildren(_,[]).
depthfirst([]):-!.
depthfirst([Node|Frontier]):-
format('~p',[Node]),
getchildren(Node,Children),
append(Children,Frontier,NewFrontier),
depthfirst(NewFrontier).
```

**10)8 queens:**

```
next(_, 1):- true.
next(_, 2):- true.
next(_, 3):- true.
next(_, 4):- true.
next(_, 5):- true.
next(_, 6):- true.
next(_, 7):- true.
next(_, 8):- true.

safe([], _, _):- true.
safe([(R, C)|T], R1, C1):-
   not(R == R1), not(C == C1),
   XD is R1 - R, YD is C1 - C, NXD is R - R1,
   not(XD == YD), not(NXD == YD),
   safe(T, R1, C1).
solve(9, L, L).
solve(R, L, A):-
   next(R, C),
   safe(L, R, C),
   R1 is R + 1,
   solve(R1, [(R, C)|L], A).
solve_8_queens:-
   solve(1, [], A).
```

**16)family representation:**

```
female(pam).
female(liz).
female(pat).
female(ann).
male(jim).
male(bob).
male(tom).
male(peter).
parent(pam,bob).
parent(tom,bob).
parent(tom,liz).
parent(bob,ann).
parent(bob,pat).
parent(pat,jim).
parent(bob,peter).
parent(peter,jim).
mother(X,Y):-parent(X,Y),female(X).
father(X,Y):-parent(X,Y),male(X).
sister(X,Y):-parent(Z,X),parent(Z,Y),female(X).
brother(X,Y):-parent(Z,X),parent(Z,Y),male(X).
```

**9,17)compound variables:**

```
go:-
readaddress(Address),nl,write(Address),nl,nl,write("
Accept(y
/n)?"),read(Reply),Reply=='y',!.
go:-
nl,write("please re-enter:"),nl,go.
readaddress(address(Name,Street,City,State,Zip)):-
write("Name:"),read(Name),
write("Street:"),read(Street),
write("City:"),read(City),
write("State:"),read(State),
write("Zip:"),read(Zip).
```

## 8-puzzle:

```
def print__(A):
  a, b, c = list(A[:3]), list(A[3:6]), list(A[6:])
  print(f'{a}\n{b}\n{c}\n')
def get_distance(G, state):
  A, count = list(state), 0
  for i, j in zip(A, G):
    if i != j: count += 1
  return count
def get_children(state, G, visited):
  pos, moves = state.find('0'), []
  if pos in [0, 1, 3, 4, 6, 7]: moves.append((1, 'left'))
  if pos in [1, 2, 4, 5, 7, 8]: moves.append((-1, 'right'))
  if pos in [3, 4, 5, 6, 7, 8]: moves.append((-3, 'down'))
  if pos in [0, 1, 2, 3, 4, 5]: moves.append((3, 'up'))
  children, p = [], list(state)
  for (i, j) in moves:
    p[pos], p[pos + i] = p[pos + i], p[pos]
    st = ''.join(p)
    val = get_distance(G, p)
    if st not in visited:
      children.append((val, j, st))
    p[pos], p[pos + i] = p[pos + i], p[pos]
  return sorted(children)
def solve(S, G, visited, path):
  print__(S)
  if S == ''.join(G): return True
  chl = get_children(S, G, visited)
  if len(chl) == 0: return False
  for i in chl:
    visited.add(S)
    path.append(i[1])
    if solve(i[2], G, visited, path) == True: return True
    path.pop()
  return False
if __name__ == '__main__':
  A = ['2', '3', '0', '1', '4', '6', '7', '5', '8']
  G = ['1', '2', '3', '4', '5', '6', '7', '8', '0']
  goal = ''.join(A)
  path, visited = [], set()
  visited.add(goal)
  print('start state : ')
  solve(goal, G, visited, path)
  print(path)
```