## 1. Write a python program to implement constraint satisfaction problem.

```
import re
import time

from z3 import *

def csp(input: str, limit=None, unique=True):
    start_time  = time.perf_counter()
    solver      = Solver()
    token_words = re.findall(r'\b[a-zA-Z]\w*\b', input)

    letters = { l: Int(l) for l in list("".join(token_words)) }
    words   = { w: Int(w) for w in list(token_words)          }

    for l,s in letters.items():
        solver.add(0 <= s, s <= 9)
    if unique and len(letters) <= 10:
        solver.add(Distinct(*letters.values()))
    solver.add(Distinct(*words.values()))

    for word in words.keys():
        solver.add( letters[word[0]] != 0 )

    for word, word_symbol in words.items():
        solver.add(word_symbol == Sum(*[letter_symbol * 10**index
            for index,letter_symbol in enumerate(reversed([
                letters[l] for l in list(word)])) ]))

    solver.add(eval(input, None, words))
    solutions = []
    print(input)
    while str(solver.check()) == 'sat':
        solutions.append({ str(s):
                        solver.model()[s] for w,s in words.items() })

        print(solutions[-1])
        solver.add(Or(*[ s != solver.model()[s] for w,s in words.items() ]))
        if limit and len(solutions) >= limit:
            break

    run_time = round(time.perf_counter() - start_time, 1)
    print(f'== {len(solutions)} solutions found in {run_time}s ==\n')
csp('TWO + TWO == FOUR')
```

**OUTPUT**

```
TWO + TWO == FOUR
{'TWO': 765, 'FOUR': 1530}
{'TWO': 734, 'FOUR': 1468}
{'TWO': 836, 'FOUR': 1672}
{'TWO': 938, 'FOUR': 1876}
{'TWO': 928, 'FOUR': 1856}
{'TWO': 867, 'FOUR': 1734}
{'TWO': 846, 'FOUR': 1692}
== 7 solutions found in 6.2s ==
```

## 2. Write a python program to implement Greedy best-first search.

```
from queue import PriorityQueue
v=14
graph=[[] for i in range(v)]
def BestFS(src,target,n):
    visited=[False]*n
    pq=PriorityQueue()
    pq.put((0,src))
    visited[src]=True

    while pq.empty()==False:
        u=pq.get()[1]
        print(u,end=" ")
        if u==target:
            break
        for v,c in graph[u]:
            if visited[v]==False:
                visited[v]=True
                pq.put((c,v))
    print()

def addedge(x,y,cost):
    graph[x].append((y,cost))
    graph[y].append((x,cost))

addedge(0,1,3)
addedge(0,2,6)
addedge(0,3,5)
addedge(1,4,9)
addedge(1,5,8)
addedge(2,6,12)
addedge(2,7,14)
addedge(3,8,7)
addedge(8,9,5)
addedge(8,10,6)
addedge(9,11,1)
addedge(9,12,1)
addedge(9,12,10)
addedge(9,13,2)

source=0
target=9
BestFS(source,target,v)
```

**OUTPUT**

```
0 1 3 2 8 9
```

### 3. Write a Python code to implement alpha–beta pruning.

```
MAX, MIN = 1000, -1000

def minimax(depth, nodeIndex, maximizingPlayer,values, alpha, beta):
    if depth == 3:
        return values[nodeIndex]

    if maximizingPlayer:
        best = MIN
        for i in range(0, 2):
            val = minimax(depth + 1, nodeIndex * 2 + i,False, values, alpha, beta)
            best = max(best, val)
            alpha = max(alpha, best)

            if beta <= alpha:
                break
        return best
    else:
        best = MAX

        for i in range(0, 2):
            val = minimax(depth + 1, nodeIndex * 2 + i,True, values, alpha, beta)
            best = min(best, val)
            beta = min(beta, best)

            if beta <= alpha:
                break
        return best

if __name__ == "__main__":
    values = [3, 5, 6, 9, 1, 2, 0, -1]
    print("The optimal value is :", minimax(0, 0, True, values, MIN, MAX))
```

**OUTPUT**

```
The optimal value is : 5
```

**4. Write a python program to construct a Bayesian network by considering anyexample data.**

**Step 1:** Install pgmpy package

- pip install pgmpy

**Step 2:** Goto https://www.kaggle.com/datasets/cherngs/heart-disease-cleveland-uci
**Step 3:** Download the dataset.

**Code**

```
import numpy as np
import csv
import pandas as pd
from pgmpy.models import BayesianModel
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.inference import VariableElimination

#read Cleveland Heart Disease data
heartDisease = pd.read_csv('heart.csv')
heartDisease = heartDisease.replace('?',np.nan)

#display the data
print('Few examples from the dataset are given below')
print(heartDisease.head())

#Model Bayesian Network
Model=BayesianModel([('age','trestbps'),('age','fbs'),
('sex','trestbps'),('exang','trestbps'),('trestbps','heartdisease'),('fbs','heartdisease'),('heartdisease','restecg')
,('heartdisease','thalach'),('heartdisease','chol')])

#Learning CPDs using Maximum Likelihood Estimators
print('\n Learning CPD using Maximum likelihood estimators')
model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)

# Inferencing with Bayesian Network
print('\n Inferencing with Bayesian Network:')
HeartDisease_infer = VariableElimination(model

#computing the Probability of HeartDisease given Age
print('\n 1. Probability of HeartDisease given Age=30')
q=HeartDisease_infer.query(variables=['heartdisease'],evidence={'age':28})
print(q['heartdisease'])

#computing the Probability of HeartDisease given cholesterol
print('\n 2. Probability of HeartDisease given cholesterol=100')
q=HeartDisease_infer.query(variables=['heartdisease'],evidence={'chol':100})
print(q['heartdisease'])
```

**OUTPUTS**

```
Few examples from the dataset are given below
   age  sex cp  trestbps   ...slope  ca  thal  heartdisease
0   63   1   1    145      ...  3    0    6            0
1   67   1   4    160      ...  2    3    3            2
2   67   1   4    120      ...  2    2    7            1
3   37   1   3    130      ...  3    0    3            0
4   41   0   2    130      ...  1    0    3            0

[5 rows x 14 columns]

Learning CPD using Maximum likelihood estimators

Inferencing with Bayesian Network:

1. Probability of HeartDisease given Age=28
```

| heartdisease | phi(heartdisease) |
|---|---|
| heartdisease_0 | 0.6791 |
| heartdisease_1 | 0.1212 |
| heartdisease_2 | 0.0810 |
| heartdisease_3 | 0.0939 |
| heartdisease_4 | 0.0247 |

```
2. Probability of HeartDisease given cholesterol=100
```

| heartdisease | phi(heartdisease) |
|---|---|
| heartdisease_0 | 0.5400 |
| heartdisease_1 | 0.1533 |
| heartdisease_2 | 0.1303 |
| heartdisease_3 | 0.1259 |
| heartdisease_4 | 0.0506 |

## 5. Write a Python program to implement A* search.

```
from collections import deque
class Graph:
    def __init__(self,adj_list):
        self.adj_list=adj_list
    def get_neigh(self,v):
        return self.adj_list[v]
    def heuristic(self,n):
        H={'A':1,'B':1,'C':1,'D':1}
        return H[n]
    def a_star(self,start_node,stop_node):
        openlist=set([start_node])
        closedlist=set([])
        g={}
        g[start_node]=0
        parents={}
        parents[start_node]=start_node
        while len(openlist)>0:
            n=None
            for v in openlist:
                if n==None or g[v]+self.heuristic(v)<g[n]+self.heuristic(n):
                    n=v;
            if n==None:
                print('Path does not exist!!!')
                return None
            if n==stop_node:
                reconstructpath=[]
                while parents[n]!=n:
                    reconstructpath.append(n)
                    n=parents[n]
                reconstructpath.append(start_node)
                reconstructpath.reverse()
                print('Path Found : {}'.format(reconstructpath))
                return reconstructpath
            for(m,weight) in self.get_neigh(n):
                if m not in openlist and m not in closedlist:
                    openlist.add(m)
                    parents[m]=n
                    g[m]=g[n]+weight
                else:
                    if g[m]>g[n]+weight:
                        g[m]=g[n]+weight
                        parents[m]=n
                        if m in closedlist:
                            closedlist.remove(m)
                            openlist.add(m)
            openlist.remove(n)
            closedlist.add(n)
        print('Path does not exist!!!')
        return None
adj_list={
    'A':[('B',1),('C',3),('D',7)],
    'B':[('D',5)],
    'C':[('D',12)]
}
graph1=Graph(adj_list)
graph1.a_star('A','D')
```

**OUTPUT**

```
Path Found : ['A', 'B', 'D']
```

**6. Write a Python code to solve traveling salesman problem.**

```
from sys import maxsize
from itertools import permutations
V = 4

def travellingSalesmanProblem(graph, s):
    vertex = []
    for i in range(V):
        if i != s:
            vertex.append(i)

    min_path = maxsize
    next_permutation=permutations(vertex)
    for i in next_permutation:
        current_pathweight = 0
        k = s
        for j in i:
            current_pathweight += graph[k][j]
            k = j
        current_pathweight += graph[k][s]
        min_path = min(min_path, current_pathweight)

    return min_path


if __name__ == "__main__":
    graph = [[0, 10, 15, 20], [10, 0, 35, 25],
            [15, 35, 0, 30], [20, 25, 30, 0]]
    s = 0
    print(travellingSalesmanProblem(graph, s))
```

**OUTPUT**

```
80
```