Pandit Deendayal Energy University, Gandhinagar

School of Technology

Department of Computer Science & Engineering

Artificial Intelligence Lab (20CP313P)

B. Tech-Computer Science & Engineering (Sem-VI)

Name: Dhyey Rajveer
Enrolment No: 19BCP034
Sem: VI
Division: 1
Branch: Computer and Science Engineering

Index:

Contents

Assignment – 1

Aim: Q&A

Code:

```
import numpy as np
// Que 1
import AddSub
a,b = 12,7
print(AddSub.add(a,b))
print(AddSub.sub(a,b))
// Que 2
import os
print(len(os.listdir()))
// Que 3
a = np.ones((8,8),dtype="int32")
a[0::2,0::2] = 0
a[1::2,1::2] = 0
print(a)
// Que 4
b = np.random.randint(1,50,(3,4))
c = np.random.randint(1,50,(3,4))
np.random.seed(0)
print(b)
print(c)
print(f"Common Elements are,\n{np.intersect1d(b,c)}");
Que 5
r,c = 3,4
```

```
d = np.ones((r,c))
h = np.ones((1,c)) - 1
v = np.ones((r+2,1)) - 1
d = np.vstack([d,h])
d = np.vstack([h,d])
d = np.hstack([v,d])
d = np.hstack([d,v])
print(d)
```

Assignment – 2

Aim: Design Machine Learning model for the house price prediction. To train model, use data available on the below link.

https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data

Code:

```
### Importing libraries and data
# Adding needed libraries and reading data
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import ensemble, tree, linear_model
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.utils import shuffle

%matplotlib inline
import warnings
warnings.filterwarnings('ignore')

train = pd.read_csv('../input/train.csv')
test = pd.read_csv('../input/test.csv')
train.head()
### Checking for NAs
#Checking for missing data
NAs = pd.concat([train.isnull().sum(), test.isnull().sum()], axis=1, keys=['Train', 'Test'])
```

```
NAs[NAs.sum(axis=1) > 0]
### Importing my functions
# Prints R2 and RMSE scores
def get_score(prediction, lables):
    print('R2: {}'.format(r2_score(prediction, lables)))
    print('RMSE: {}'.format(np.sqrt(mean_squared_error(prediction, lables))))


# Shows scores for train and validation sets
def train_test(estimator, x_trn, x_tst, y_trn, y_tst):
    prediction_train = estimator.predict(x_trn)
    # Printing estimator
    print(estimator)
    # Printing train scores
    get_score(prediction_train, y_trn)
    prediction_test = estimator.predict(x_tst)
    # Printing test scores
    print("Test")
    get_score(prediction_test, y_tst)
### Splitting to features and labels and deleting variables I don't need
# Spliting to features and lables and deleting variable I don't need
train_labels = train.pop('SalePrice')

features = pd.concat([train, test], keys=['train', 'test'])
```

```
features.drop(['Utilities', 'RoofMatl', 'MasVnrArea', 'BsmtFinSF1',
'BsmtFinSF2', 'BsmtUnfSF', 'Heating', 'LowQualFinSF',
                'BsmtFullBath', 'BsmtHalfBath', 'Functional', 'GarageYrBlt',
'GarageArea', 'GarageCond', 'WoodDeckSF',
                'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch',
'PoolArea', 'PoolQC', 'Fence', 'MiscFeature', 'MiscVal'],
               axis=1, inplace=True)
### Filling NAs and converting features
# MSSubClass as str
features['MSSubClass'] = features['MSSubClass'].astype(str)

# MSZoning NA in pred. filling with most popular values
features['MSZoning'] =
features['MSZoning'].fillna(features['MSZoning'].mode()[0])

# LotFrontage  NA in all. I suppose NA means 0
features['LotFrontage'] =
features['LotFrontage'].fillna(features['LotFrontage'].mean())

# Alley  NA in all. NA means no access
features['Alley'] = features['Alley'].fillna('NOACCESS')

# Converting OverallCond to str
features.OverallCond = features.OverallCond.astype(str)

# MasVnrType NA in all. filling with most popular values
```

```
features['MasVnrType'] = features['MasVnrType'].fillna(features['MasVnrType'].mode()[0])


# BsmtQual, BsmtCond, BsmtExposure, BsmtFinType1, BsmtFinType2
# NA in all. NA means No basement
for col in ('BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2'):
    features[col] = features[col].fillna('NoBSMT')


# TotalBsmtSF  NA in pred. I suppose NA means 0
features['TotalBsmtSF'] = features['TotalBsmtSF'].fillna(0)


# Electrical NA in pred. filling with most popular values
features['Electrical'] = features['Electrical'].fillna(features['Electrical'].mode()[0])


# KitchenAbvGr to categorical
features['KitchenAbvGr'] = features['KitchenAbvGr'].astype(str)


# KitchenQual NA in pred. filling with most popular values
features['KitchenQual'] = features['KitchenQual'].fillna(features['KitchenQual'].mode()[0])


# FireplaceQu  NA in all. NA means No Fireplace
features['FireplaceQu'] = features['FireplaceQu'].fillna('NoFP')
```

```
# GarageType, GarageFinish, GarageQual  NA in all. NA means No Garage
for col in ('GarageType', 'GarageFinish', 'GarageQual'):
    features[col] = features[col].fillna('NoGRG')

# GarageCars  NA in pred. I suppose NA means 0
features['GarageCars'] = features['GarageCars'].fillna(0.0)

# SaleType NA in pred. filling with most popular values
features['SaleType'] = features['SaleType'].fillna(features['SaleType'].mode()[0])

# Year and Month to categorical
features['YrSold'] = features['YrSold'].astype(str)
features['MoSold'] = features['MoSold'].astype(str)

# Adding total sqfootage feature and removing Basement, 1st and 2nd floor features
features['TotalSF'] = features['TotalBsmtSF'] + features['1stFlrSF'] + features['2ndFlrSF']
features.drop(['TotalBsmtSF', '1stFlrSF', '2ndFlrSF'], axis=1, inplace=True)
### Log transformation
# Our SalesPrice is skewed right (check plot below). I'm logtransforming it.
ax = sns.distplot(train_labels)
```

```
## Log transformation of labels

train_labels = np.log(train_labels)

## Now it looks much better

ax = sns.distplot(train_labels)

### Standardizing numeric data

## Standardizing numeric features

numeric_features = features.loc[:,['LotFrontage', 'LotArea',
'GrLivArea', 'TotalSF']]

numeric_features_standardized = (numeric_features -
numeric_features.mean())/numeric_features.std()

ax = sns.pairplot(numeric_features_standardized)

### Converting categorical data to dummies

# Getting Dummies from Condition1 and Condition2

conditions = set([x for x in features['Condition1']] + [x for x in
features['Condition2']])

dummies = pd.DataFrame(data=np.zeros((len(features.index),
len(conditions))),

                       index=features.index, columns=conditions)

for i, cond in enumerate(zip(features['Condition1'],
features['Condition2'])):

    dummies.ix[i, cond] = 1

features = pd.concat([features, dummies.add_prefix('Condition_')],
axis=1)

features.drop(['Condition1', 'Condition2'], axis=1, inplace=True)


# Getting Dummies from Exterior1st and Exterior2nd
```

```
exteriors = set([x for x in features['Exterior1st']] + [x for x in features['Exterior2nd']])
dummies = pd.DataFrame(data=np.zeros((len(features.index), len(exteriors))),
                       index=features.index, columns=exteriors)
for i, ext in enumerate(zip(features['Exterior1st'], features['Exterior2nd'])):
    dummies.ix[i, ext] = 1
features = pd.concat([features, dummies.add_prefix('Exterior_')], axis=1)
features.drop(['Exterior1st', 'Exterior2nd', 'Exterior_nan'], axis=1, inplace=True)

# Getting Dummies from all other categorical vars
for col in features.dtypes[features.dtypes == 'object'].index:
    for_dummy = features.pop(col)
    features = pd.concat([features, pd.get_dummies(for_dummy, prefix=col)], axis=1)
### Obtaining standardized dataset
### Copying features
features_standardized = features.copy()

### Replacing numeric features by standardized values
features_standardized.update(numeric_features_standardized)
### Splitting train and test features
### Splitting features
```

```
train_features = features.loc['train'].drop('Id', axis=1).select_dtypes(include=[np.number]).values

test_features = features.loc['test'].drop('Id', axis=1).select_dtypes(include=[np.number]).values


### Splitting standardized features

train_features_st = features_standardized.loc['train'].drop('Id', axis=1).select_dtypes(include=[np.number]).values

test_features_st = features_standardized.loc['test'].drop('Id', axis=1).select_dtypes(include=[np.number]).values

### Splitting to train and validation sets

### Shuffling train sets

train_features_st, train_features, train_labels = shuffle(train_features_st, train_features, train_labels, random_state = 5)

### Splitting

x_train, x_test, y_train, y_test = train_test_split(train_features, train_labels, test_size=0.1, random_state=200)

x_train_st, x_test_st, y_train_st, y_test_st = train_test_split(train_features_st, train_labels, test_size=0.1, random_state=200)

## First level models

### Elastic Net

ENSTest = linear_model.ElasticNetCV(alphas=[0.0001, 0.0005, 0.001, 0.01, 0.1, 1, 10], l1_ratio=[.01, .1, .5, .9, .99], max_iter=5000).fit(x_train_st, y_train_st)

train_test(ENSTest, x_train_st, x_test_st, y_train_st, y_test_st)

# Average R2 score and standart deviation of 5-fold cross-validation
```

```
scores = cross_val_score(ENSTest, train_features_st, train_labels, cv=5)

print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))

### Gradient Boosting

GBest = ensemble.GradientBoostingRegressor(n_estimators=3000, learning_rate=0.05, max_depth=3, max_features='sqrt',
                                           min_samples_leaf=15, min_samples_split=10, loss='huber').fit(x_train, y_train)

train_test(GBest, x_train, x_test, y_train, y_test)

# Average R2 score and standart deviation of 5-fold cross-validation

scores = cross_val_score(GBest, train_features_st, train_labels, cv=5)

print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))

## Ensembling final model

# Retraining models

GB_model = GBest.fit(train_features, train_labels)

ENST_model = ENSTest.fit(train_features_st, train_labels)

## Getting our SalePrice estimation

Final_labels = (np.exp(GB_model.predict(test_features)) + np.exp(ENST_model.predict(test_features_st))) / 2

## Saving to CSV

pd.DataFrame({'Id': test.Id, 'SalePrice': Final_labels}).to_csv('2017-02-28.csv', index =False)
```

Assignment - 3

Aim: Design a Machine Learning model for the Heart Attack prediction. Download the Dataset along with the description

Code:

```
!pip install feature-engine

import numpy as np

import pandas as pd

# Data Analysis

data=pd.read_csv("heart.csv")

data.head()

data.tail()

data.isnull().sum()

# Train Test Split

from sklearn.model_selection import train_test_split # Import train_test_split function

dropcols=['oldpeak','slp','thall']

data=data.drop(dropcols,axis=1)

Y=data['output']

X=data.drop('output',axis=1)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=1) # 80% training and 20% test

# Standardization & Feature Selection

from sklearn.pipeline import Pipeline

from feature_engine.selection import DropConstantFeatures, DropDuplicateFeatures, SmartCorrelatedSelection

from sklearn.ensemble import RandomForestClassifier

rf=RandomForestClassifier(n_estimators=10, random_state=20)

pipe=Pipeline([('constant', DropConstantFeatures(tol=0.997)),('Duplicate', DropDuplicateFeatures()),('correlated', SmartCorrelatedSelection(selection_method="model_performance", estimator=rf, scoring="roc_auc", cv=3))])

pipe.fit(X_train, Y_train)
pipe.named_steps['correlated'].features_to_drop_
X_train=pipe.transform(X_train)
X_test=pipe.transform(X_test)
X_train.shape
from sklearn.linear_model import Lasso, LogisticRegression
from sklearn.feature_selection import SelectFromModel
from sklearn.preprocessing import StandardScaler
scalar=StandardScaler()
scalar.fit(X_train)
sel=SelectFromModel(LogisticRegression(C=0.8, penalty='l1',
solver='liblinear', random_state=10))
sel.fit_transform(scalar.transform(X_train), Y_train)
sel.get_support()
selected_feature = X_train.columns[sel.get_support()]
len(selected_feature)

# Logistic Regression
from sklearn.linear_model import LogisticRegression
reg=LogisticRegression()
reg.fit(X_train,Y_train)
ypred=reg.predict(X_test)
ypred
reg.score(X_test,Y_test)
# Decision Tree
from sklearn.tree import DecisionTreeClassifier
regdc=DecisionTreeClassifier(criterion='gini')
regdc.fit(X_train,Y_train)
regdc.predict(X_test)
regdc.score(X_test,Y_test)
# RandomForest
from sklearn.ensemble import RandomForestClassifier
regrf=RandomForestClassifier(n_estimators=200)
regrf.fit(X_train,Y_train)
regrf.predict(X_test)
regrf.score(X_test,Y_test)
# KNN
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=12)
knn.fit(X_train,Y_train)
knn.predict(X_test)
knn.score(X_test,Y_test)
# SVM
from sklearn.svm import SVC

model = SVC(kernel='linear')
model.fit(X_train, Y_train)
model.predict(X_test)
model.score(X_test,Y_test)
```

Assignment – 4

Aim:

WAP to implement DFS and BFS for traversing a graph from source node (S) to goal node (G), where source node and goal node is given by the user as an input.

Code:

BFS:

```
package Graph;
import java.util.*;

class Graph
{
   private static LinkedList<Integer> adj[];

   @SuppressWarnings("unchecked")
   public Graph(int v)
   {
      adj = new LinkedList[v];
      for(int i=0;i<v;i++)
      {
         adj[i] = new LinkedList<Integer>();
      }
   }
   public void addEdge(int source, int destination)
   {
      adj[source].add(destination);
      adj[destination].add(source);
   }
   public void BFS(int source, int destination)
   {
      boolean visited[] = new boolean[adj.length];

      Queue<Integer> q = new ArrayDeque<Integer>();

      q.add(source);
      visited[source] = true;

      while(!q.isEmpty())
      {
         int top = q.peek();
         System.out.println("Visited : " + top);
         if(top==destination)
         {
            break;
         }
         q.poll();

         for(int adjacent: adj[top])
         {
            if(!visited[adjacent])
            {
               visited[adjacent] = true;
               q.add(adjacent);
            }
         }
      }
   }
}

public class BFS_Solution {

   public static void main(String[] args) {

      Scanner sc = new Scanner(System.in);

      int v,e;
      System.out.print("Enter the number of vertices :
");
      v = sc.nextInt();
      System.out.print("Enter the number of edges : ");
      e = sc.nextInt();

      Graph g = new Graph(v);

      System.out.println("Enter "+e+" edges :");

      for(int i=0;i<e;i++)
      {
         int source = sc.nextInt();
         int destination = sc.nextInt();
         g.addEdge(source, destination);
      }

      int source, destination;
      System.out.print("Enter the source for BFS
Traversal : ");
      source = sc.nextInt();
      System.out.print("Enter the destination for BFS
Traversal : ");
      destination = sc.nextInt();

      g.BFS(source, destination);
      sc.close();
   }
}
```

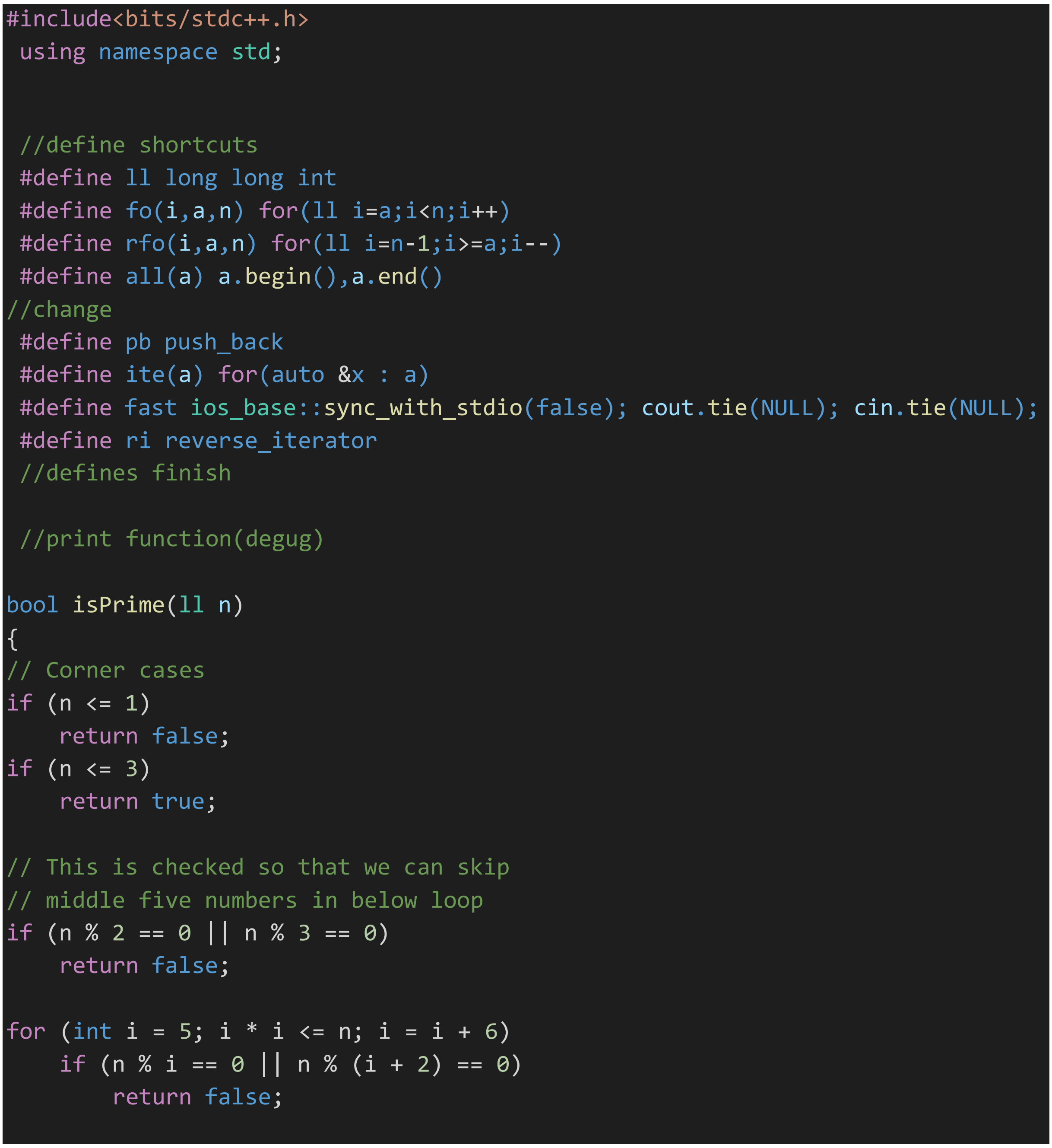
DFS:

```
package Graph;
import java.util.*;

class Graphs
{
   private static LinkedList<Integer> adj[];

   @SuppressWarnings("unchecked")
   public Graphs(int v)
   {
      adj = new LinkedList[v];
      for(int i=0;i<v;i++)
      {
         adj[i] = new LinkedList<Integer>();
      }
   }
   public void addEdge(int source, int destination)
   {
      adj[source].add(destination);
      adj[destination].add(source);
   }
   public void DFSStack(int source, int destination)
   {
      boolean visited[] = new boolean[adj.length];

      visited[source] = true;

      Stack<Integer> s = new Stack<Integer>();
      s.push(source);

      while(!s.isEmpty())
      {
         int current = s.peek();
         System.out.println("Visited : " + current);

         if(current==destination)
         {
            break;
         }

         s.pop();
         for(int adjacent: adj[current])
         {
            if(visited[adjacent]==false)
            {
               visited[adjacent] = true;
               s.push(adjacent);
            }
         }
      }
   }
}

public class DFS_Solution {

   public static void main(String[] args) {

      Scanner sc = new Scanner(System.in);

      int v,e;
      System.out.print("Enter the number of vertices :
");
      v = sc.nextInt();
      System.out.print("Enter the number of edges : ");
      e = sc.nextInt();

      Graphs graph = new Graphs(v);

      System.out.println("Enter "+e+" edges :");

      for(int i=0;i<e;i++)
      {
         int source = sc.nextInt();
         int destination = sc.nextInt();
         graph.addEdge(source, destination);
      }
      int source, destination;

      System.out.print("Enter the source for DFS
Traversal : ");
      source = sc.nextInt();
      System.out.print("Enter the destination for DFS
Traversal : ");
      destination = sc.nextInt();
      graph.DFSStack(source, destination);

      sc.close();
   }
}
```

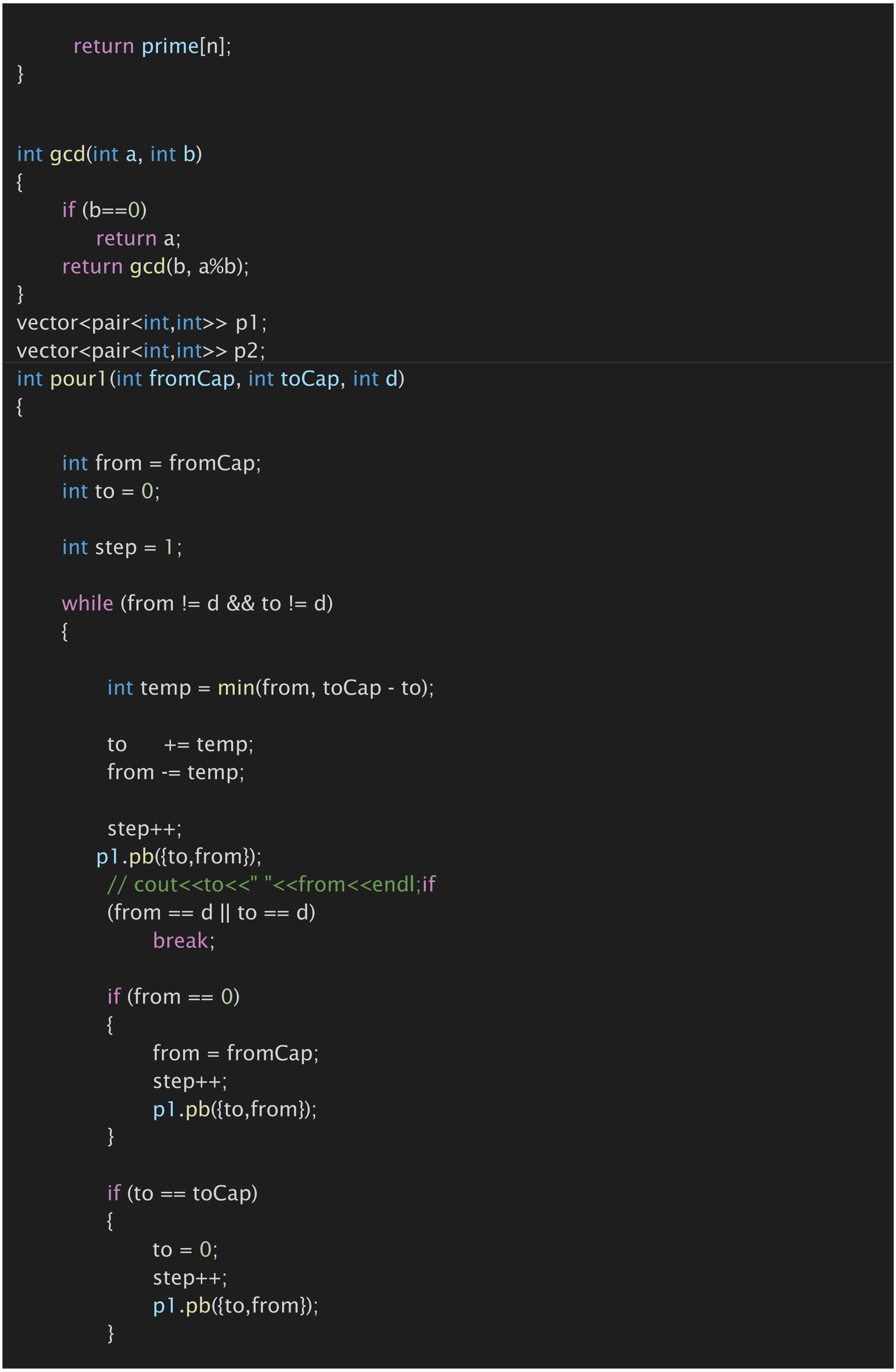
Assignment - 5

Aim: You are given two jugs with m liter and a n liter capacity. Both the jugs are initially empty. The jugs don’t have markings to allow measuring smaller quantities. You have to use the jugs to measure d liters of water where d is less than n.

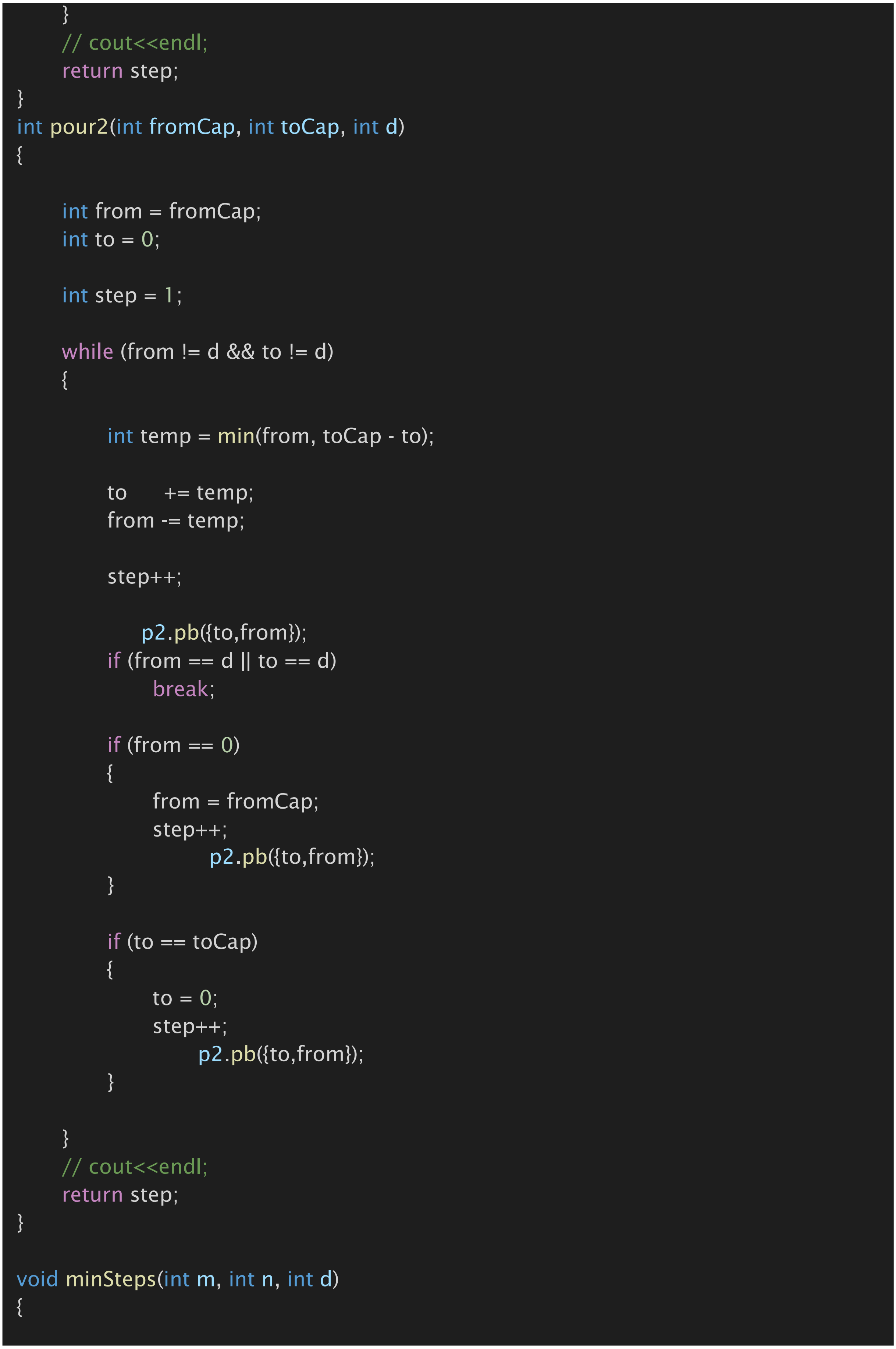
CODE:



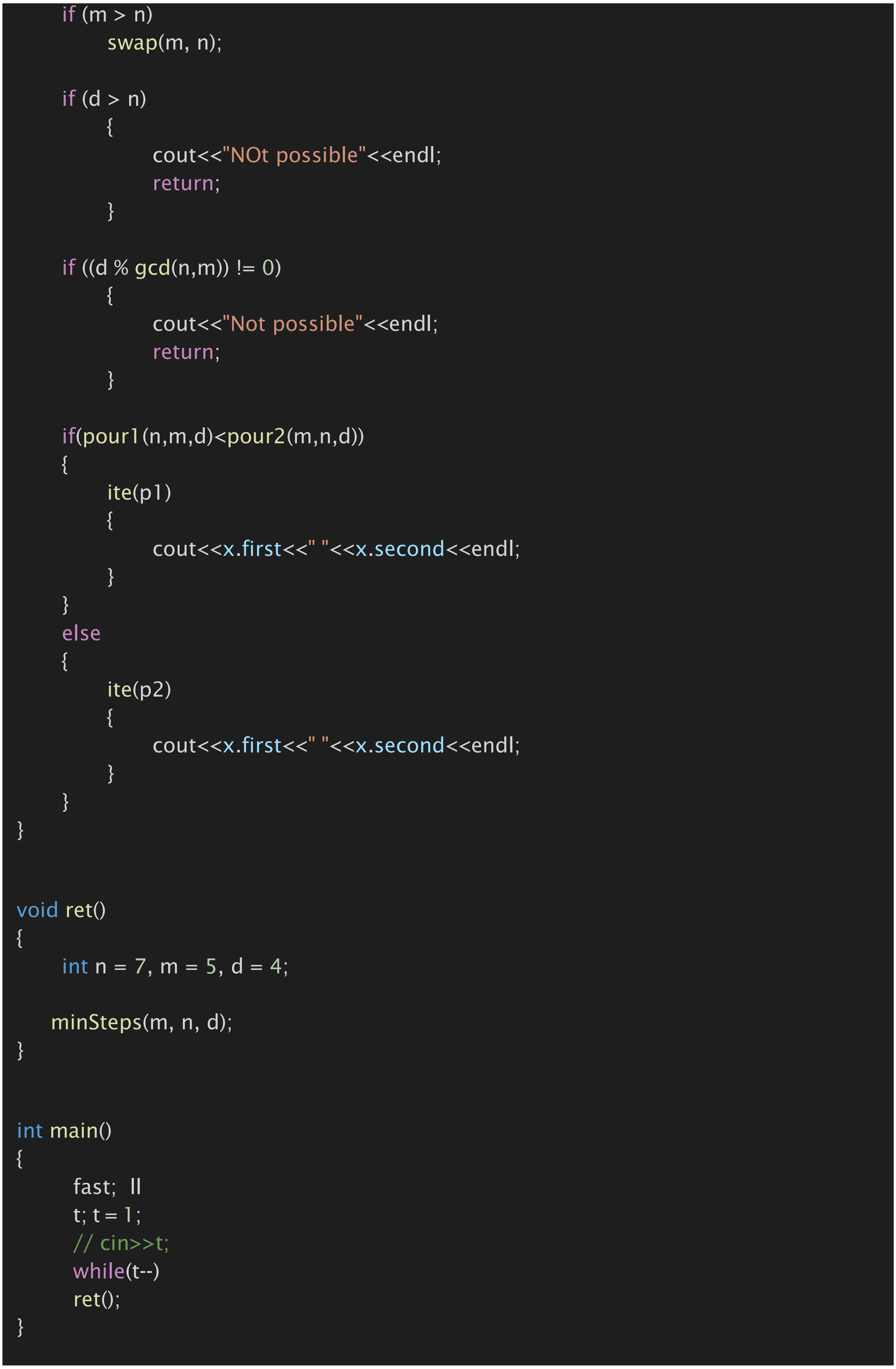
```
#include<bits/stdc++.h>
 using namespace std;


 //define shortcuts
 #define ll long long int
 #define fo(i,a,n) for(ll i=a;i<n;i++)
 #define rfo(i,a,n) for(ll i=n-1;i>=a;i--)
 #define all(a) a.begin(),a.end()
//change
 #define pb push_back
 #define ite(a) for(auto &x : a)
 #define fast ios_base::sync_with_stdio(false); cout.tie(NULL); cin.tie(NULL);
 #define ri reverse_iterator
 //defines finish

 //print function(degug)

bool isPrime(ll n)
{
// Corner cases
if (n <= 1)
    return false;
if (n <= 3)
    return true;

// This is checked so that we can skip
// middle five numbers in below loop
if (n % 2 == 0 || n % 3 == 0)
    return false;

for (int i = 5; i * i <= n; i = i + 6)
    if (n % i == 0 || n % (i + 2) == 0)
        return false;
return true;
}
 //floor and ceil for integer
 ll ceil(ll n,ll m) {if(n%m==0)return n/m; else return n/m+1;}

 //ncr Function
 ll mod = 1e9 + 7;ll
 MOD = mod;

 //ncr Functionll
 fact(ll n);

 ll nCr(ll n, ll r)
 {
        return fact(n) / (fact(r) * fact(n - r));
 }

 // Returns factorial of nll
 fact(ll n)
 {
        int res = 1;
        for (int i = 2; i <= n; i++)res =
                res * i;
        return res;
 }
 ll lcm(ll n,ll m)
 {

     return (n*m)/__gcd(n,m);



 }


 bool seive(ll n)
 {
          ll prime[n+1];
          memset(prime,1,sizeof(prime));
          for(int i = 2;i * i <= n; i++)
          {
                   if(prime[i] == 1)
                   {
                            for(int j = i * i; j <= n; j++)
                            {
                                     prime[i] = 0;
                            }
                   }
          }
```



```
     return prime[n];
}


int gcd(int a, int b)
{
    if (b==0)
       return a;
    return gcd(b, a%b);
}
vector<pair<int,int>> p1;
vector<pair<int,int>> p2;
int pour1(int fromCap, int toCap, int d)
{

    int from = fromCap;
    int to = 0;

    int step = 1;

    while (from != d && to != d)
    {

        int temp = min(from, toCap - to);

        to     += temp;
        from -= temp;

        step++;
       p1.pb({to,from});
        // cout<<to<<" "<<from<<endl;if
        (from == d || to == d)
            break;

        if (from == 0)
        {
            from = fromCap;
            step++;
            p1.pb({to,from});
        }

        if (to == toCap)
        {
            to = 0;
            step++;
            p1.pb({to,from});
        }
```

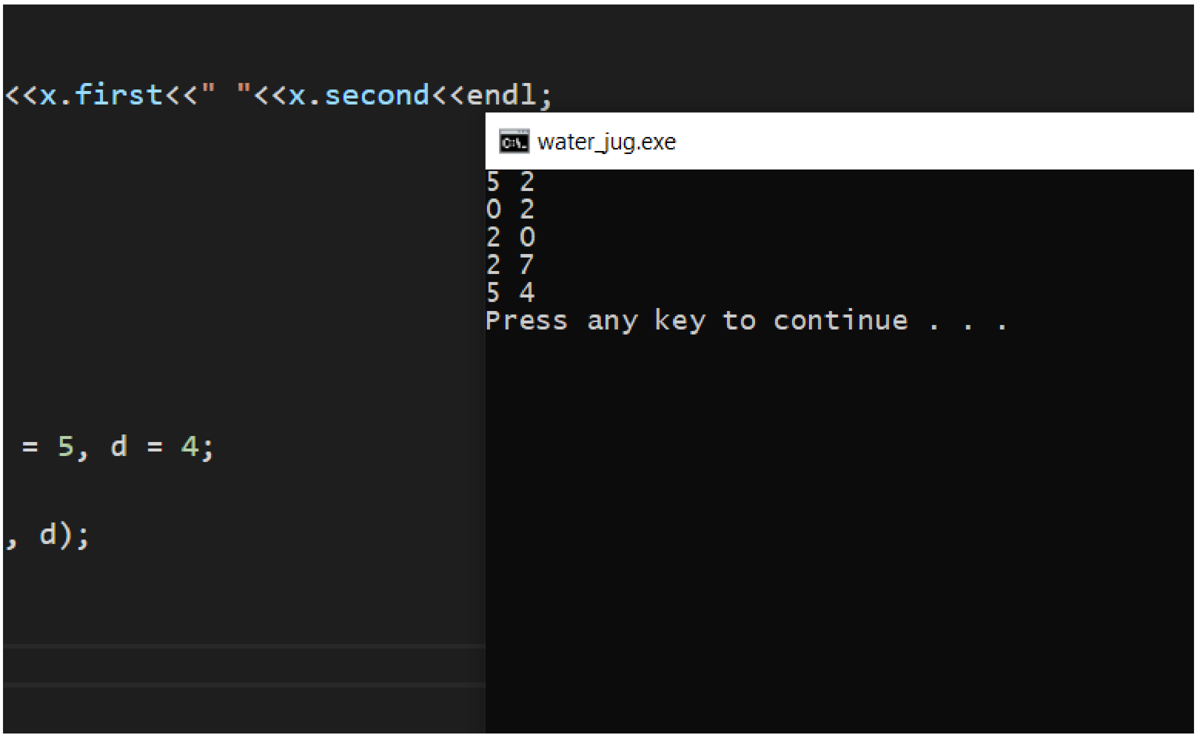


```
    }
    // cout<<endl;
    return step;
}
int pour2(int fromCap, int toCap, int d)
{

    int from = fromCap;
    int to = 0;

    int step = 1;

    while (from != d && to != d)
    {

        int temp = min(from, toCap - to);

        to   += temp;
        from -= temp;

        step++;

           p2.pb({to,from});
        if (from == d || to == d)
            break;

        if (from == 0)
        {
            from = fromCap;
            step++;
                 p2.pb({to,from});
        }

        if (to == toCap)
        {
            to = 0;
            step++;
                p2.pb({to,from});
        }

    }
    // cout<<endl;
    return step;
}

void minSteps(int m, int n, int d)
{
```



```
    if (m > n)
        swap(m, n);

    if (d > n)
        {
            cout<<"NOt possible"<<endl;
            return;
        }

    if ((d % gcd(n,m)) != 0)
        {
            cout<<"Not possible"<<endl;
            return;
        }

    if(pour1(n,m,d)<pour2(m,n,d))
    {
        ite(p1)
        {
            cout<<x.first<<" "<<x.second<<endl;
        }
    }
    else
    {
        ite(p2)
        {
            cout<<x.first<<" "<<x.second<<endl;
        }
    }
}


void ret()
{
    int n = 7, m = 5, d = 4;

   minSteps(m, n, d);
}


int main()
{
     fast;  ll
     t; t = 1;
     // cin>>t;
     while(t--)
     ret();
}
```

Output:



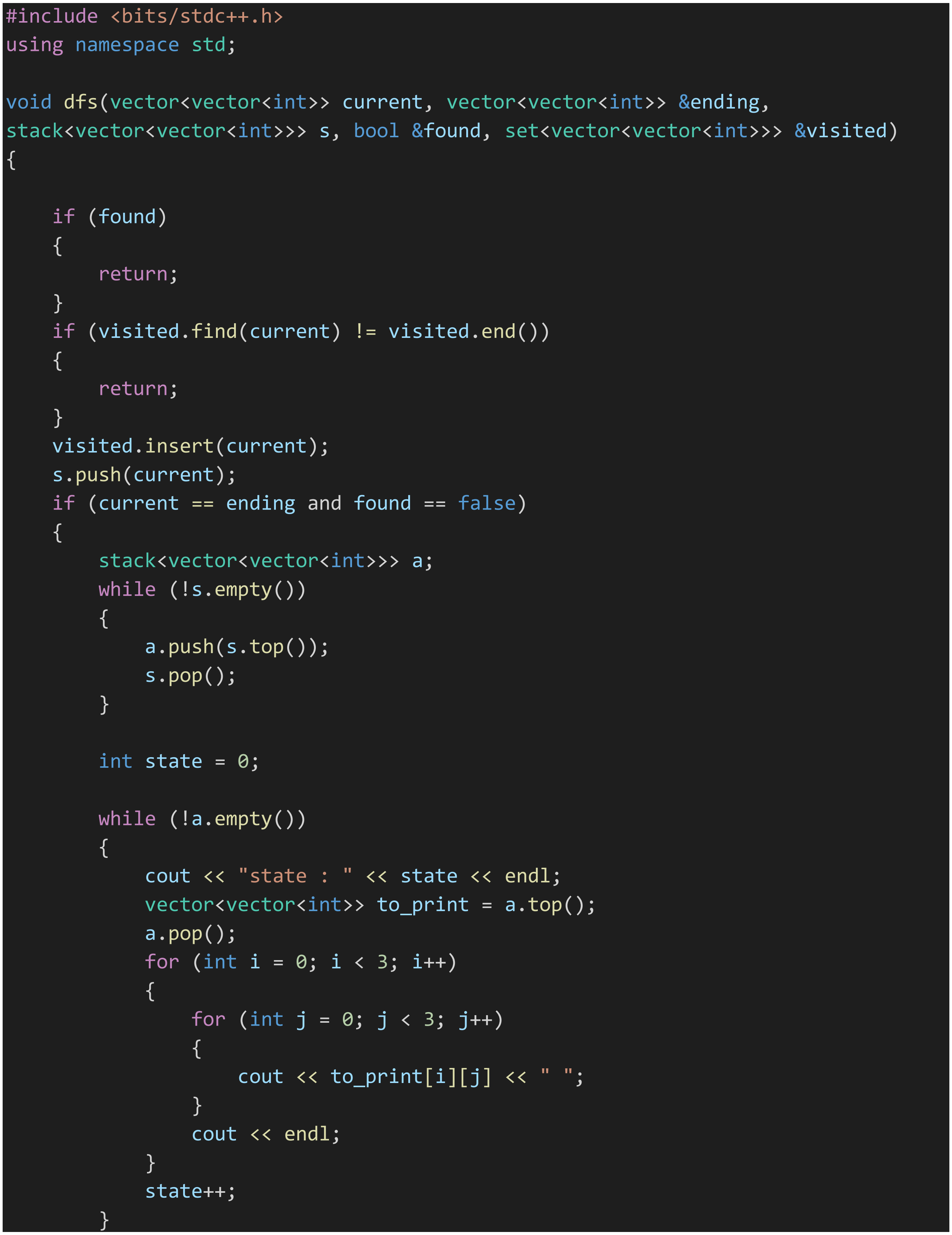
```
<<x.first<<" "<<x.second<<endl;

 = 5, d = 4;

, d);
```

```
5 2
0 2
2 0
2 7
5 4
Press any key to continue . . .
```

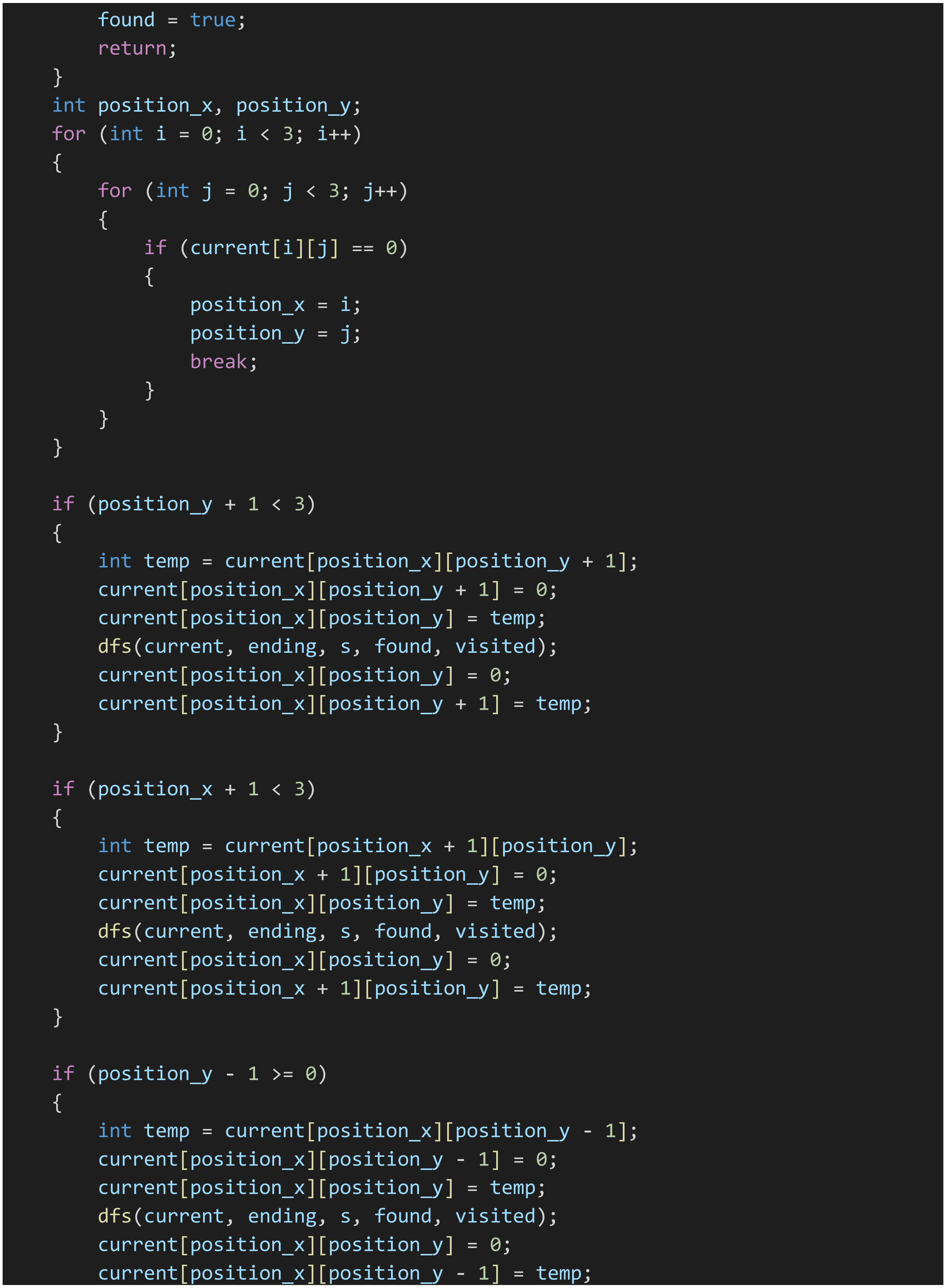
Assignment - 6

Aim: Solve 8 puzzle problem using BFS or DFS where initial state, final state and name of the method will be given by the users.

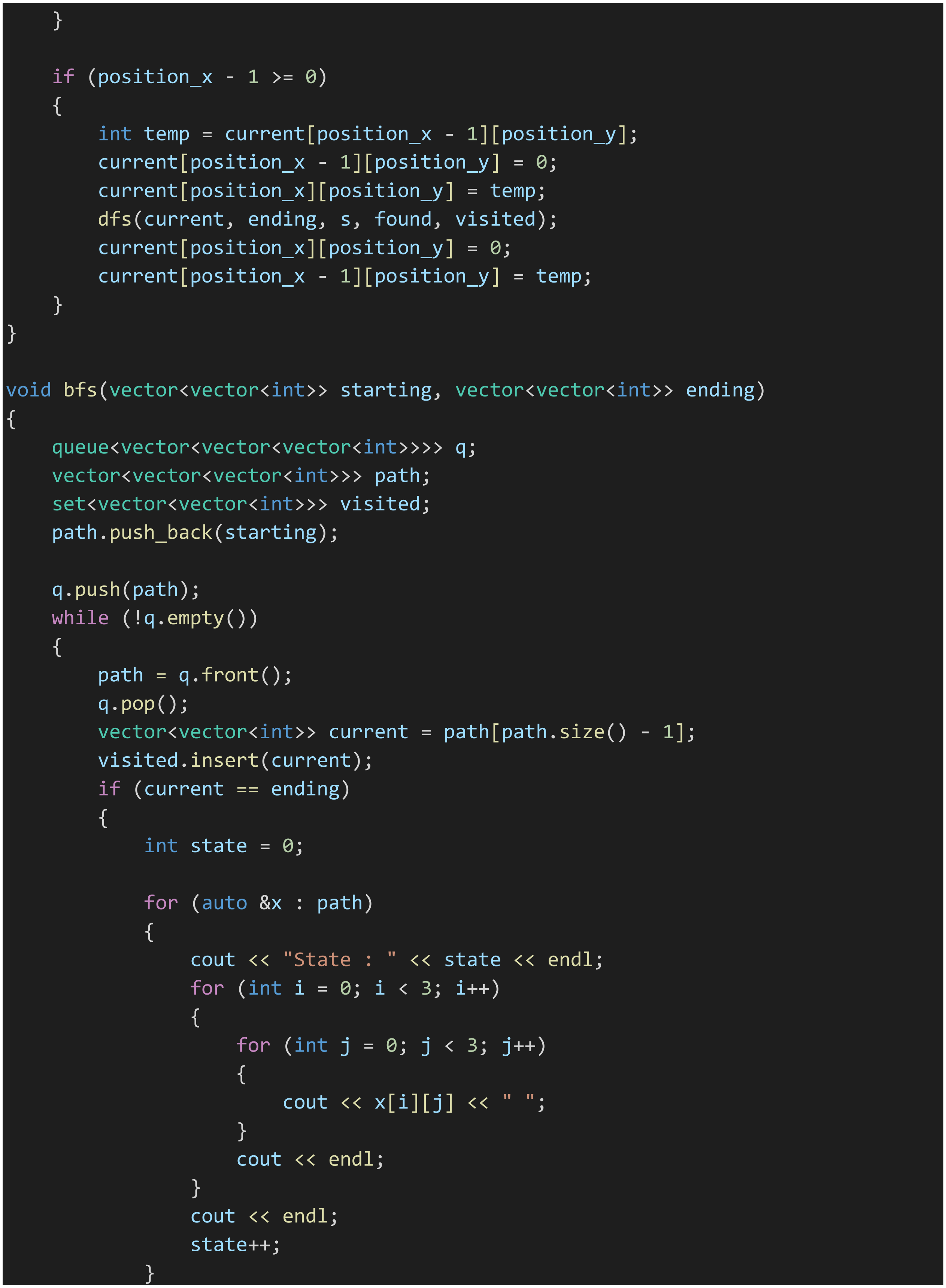
CODE: -



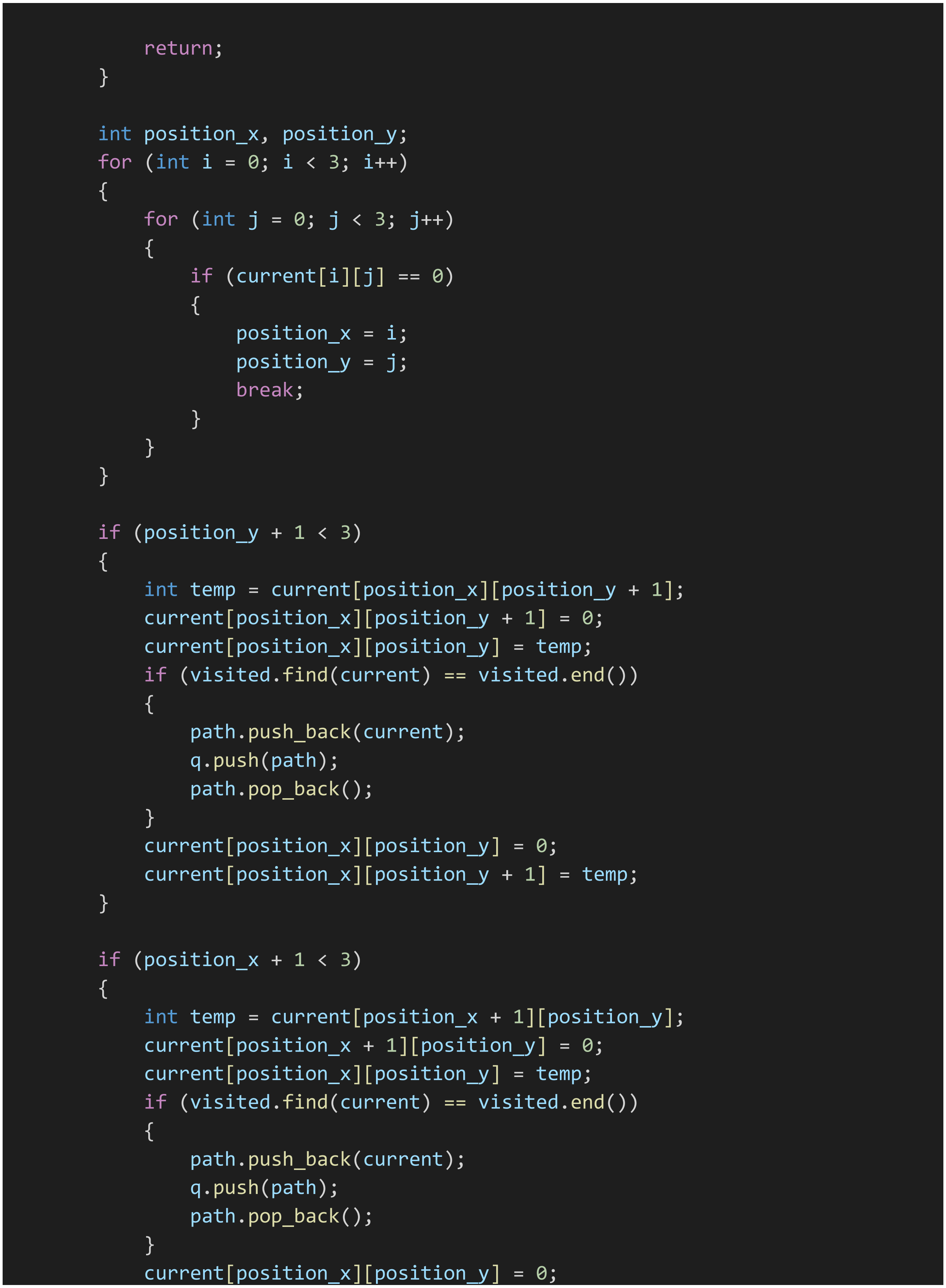
```
#include <bits/stdc++.h>
using namespace std;

void dfs(vector<vector<int>> current, vector<vector<int>> &ending,
stack<vector<vector<int>>> s, bool &found, set<vector<vector<int>>> &visited)
{

    if (found)
    {
        return;
    }
    if (visited.find(current) != visited.end())
    {
        return;
    }
    visited.insert(current);
    s.push(current);
    if (current == ending and found == false)
    {
        stack<vector<vector<int>>> a;
        while (!s.empty())
        {
            a.push(s.top());
            s.pop();
        }

        int state = 0;

        while (!a.empty())
        {
            cout << "state : " << state << endl;
            vector<vector<int>> to_print = a.top();
            a.pop();
            for (int i = 0; i < 3; i++)
            {
                for (int j = 0; j < 3; j++)
                {
                    cout << to_print[i][j] << " ";
                }
                cout << endl;
            }
            state++;
        }
```



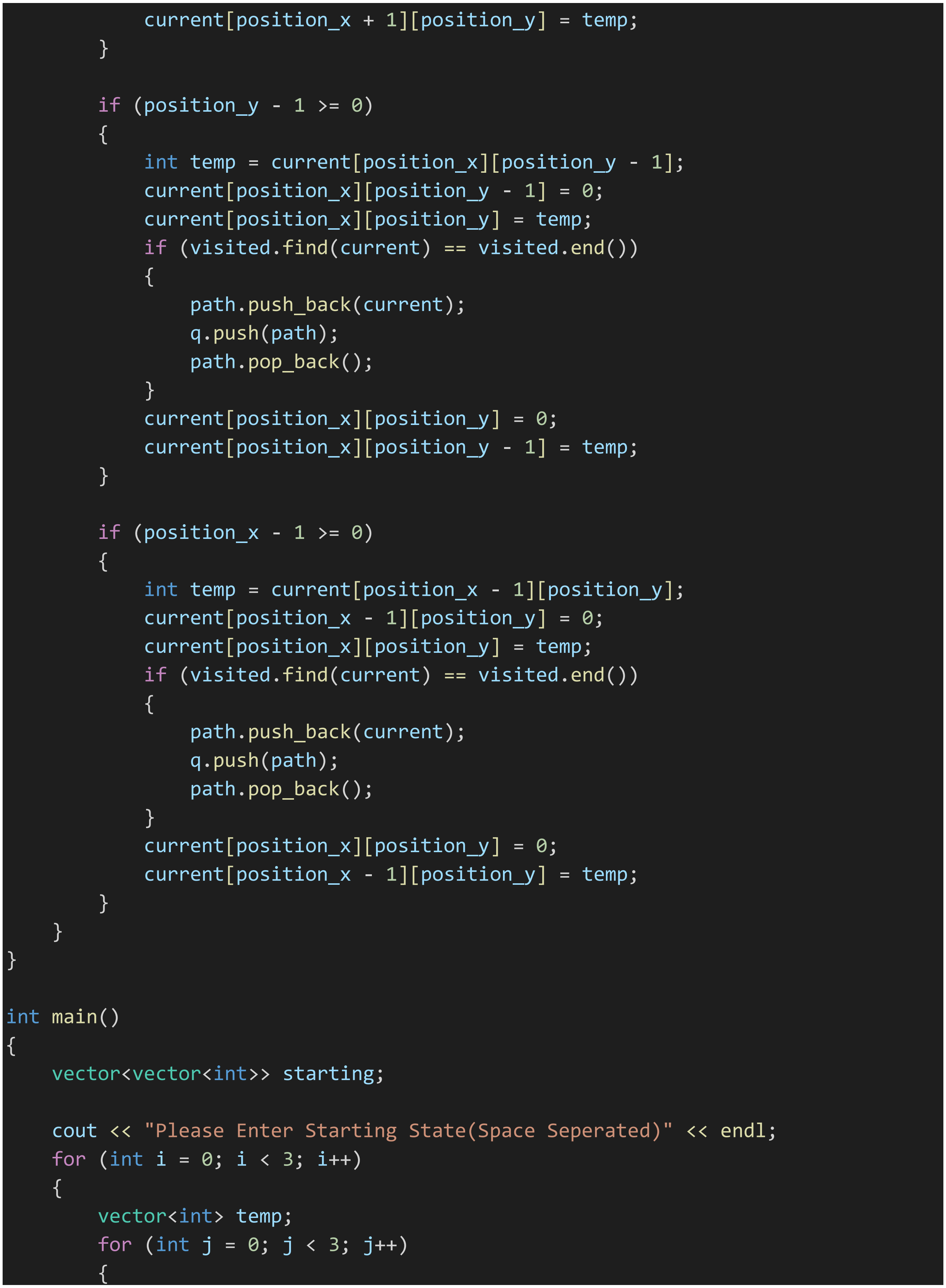
```
        found = true;
        return;
    }
    int position_x, position_y;
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            if (current[i][j] == 0)
            {
                position_x = i;
                position_y = j;
                break;
            }
        }
    }

    if (position_y + 1 < 3)
    {
        int temp = current[position_x][position_y + 1];
        current[position_x][position_y + 1] = 0;
        current[position_x][position_y] = temp;
        dfs(current, ending, s, found, visited);
        current[position_x][position_y] = 0;
        current[position_x][position_y + 1] = temp;
    }

    if (position_x + 1 < 3)
    {
        int temp = current[position_x + 1][position_y];
        current[position_x + 1][position_y] = 0;
        current[position_x][position_y] = temp;
        dfs(current, ending, s, found, visited);
        current[position_x][position_y] = 0;
        current[position_x + 1][position_y] = temp;
    }

    if (position_y - 1 >= 0)
    {
        int temp = current[position_x][position_y - 1];
        current[position_x][position_y - 1] = 0;
        current[position_x][position_y] = temp;
        dfs(current, ending, s, found, visited);
        current[position_x][position_y] = 0;
        current[position_x][position_y - 1] = temp;
```



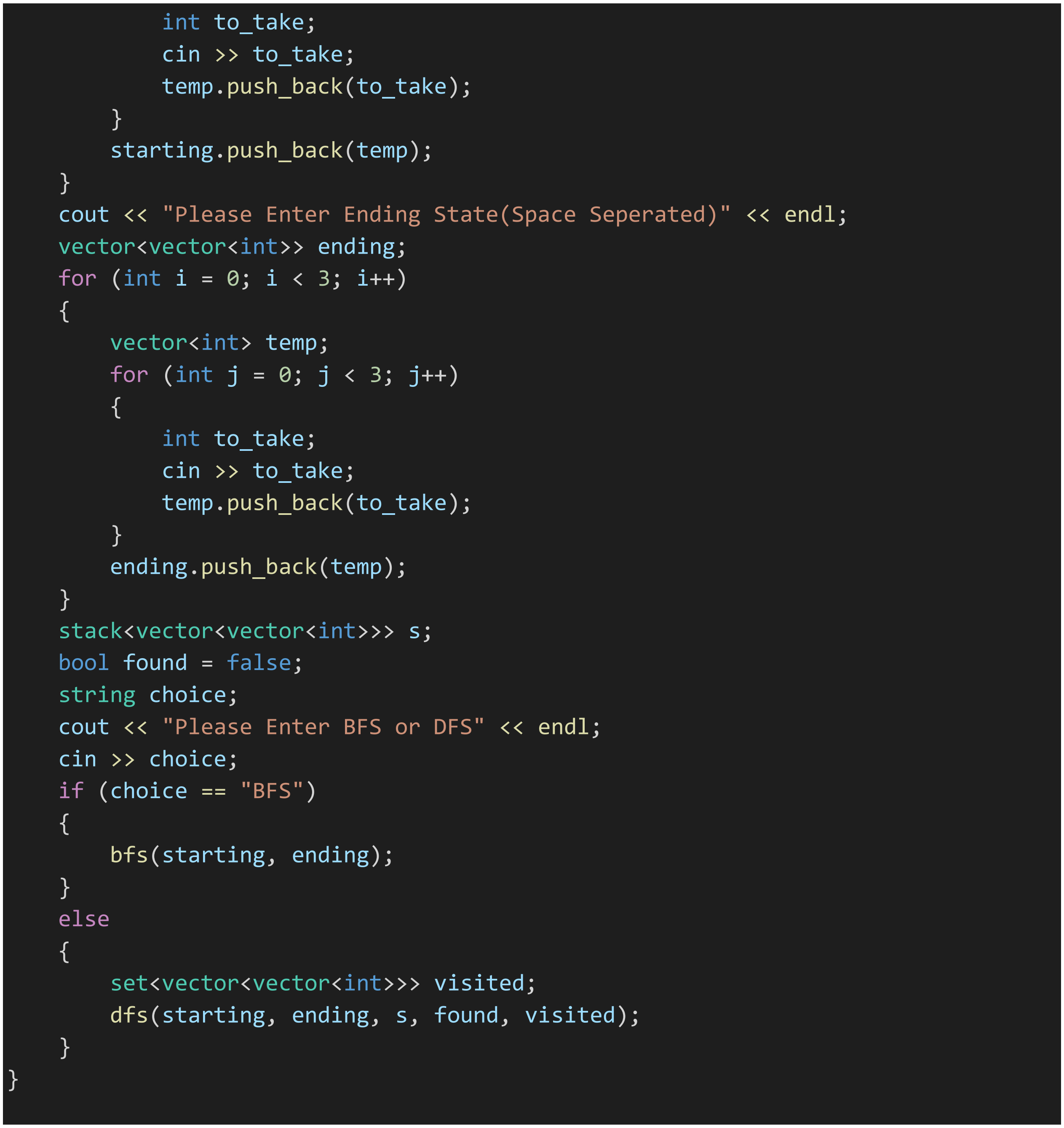
```
    }

    if (position_x - 1 >= 0)
    {
        int temp = current[position_x - 1][position_y];
        current[position_x - 1][position_y] = 0;
        current[position_x][position_y] = temp;
        dfs(current, ending, s, found, visited);
        current[position_x][position_y] = 0;
        current[position_x - 1][position_y] = temp;
    }
}

void bfs(vector<vector<int>> starting, vector<vector<int>> ending)
{
    queue<vector<vector<vector<int>>>> q;
    vector<vector<vector<int>>> path;
    set<vector<vector<int>>> visited;
    path.push_back(starting);

    q.push(path);
    while (!q.empty())
    {
        path = q.front();
        q.pop();
        vector<vector<int>> current = path[path.size() - 1];
        visited.insert(current);
        if (current == ending)
        {
            int state = 0;

            for (auto &x : path)
            {
                cout << "State : " << state << endl;
                for (int i = 0; i < 3; i++)
                {
                    for (int j = 0; j < 3; j++)
                    {
                        cout << x[i][j] << " ";
                    }
                    cout << endl;
                }
                cout << endl;
                state++;
            }
```



```
        return;
    }

    int position_x, position_y;
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            if (current[i][j] == 0)
            {
                position_x = i;
                position_y = j;
                break;
            }
        }
    }

    if (position_y + 1 < 3)
    {
        int temp = current[position_x][position_y + 1];
        current[position_x][position_y + 1] = 0;
        current[position_x][position_y] = temp;
        if (visited.find(current) == visited.end())
        {
            path.push_back(current);
            q.push(path);
            path.pop_back();
        }
        current[position_x][position_y] = 0;
        current[position_x][position_y + 1] = temp;
    }

    if (position_x + 1 < 3)
    {
        int temp = current[position_x + 1][position_y];
        current[position_x + 1][position_y] = 0;
        current[position_x][position_y] = temp;
        if (visited.find(current) == visited.end())
        {
            path.push_back(current);
            q.push(path);
            path.pop_back();
        }
        current[position_x][position_y] = 0;
```



```
            current[position_x + 1][position_y] = temp;
        }

        if (position_y - 1 >= 0)
        {
            int temp = current[position_x][position_y - 1];
            current[position_x][position_y - 1] = 0;
            current[position_x][position_y] = temp;
            if (visited.find(current) == visited.end())
            {
                path.push_back(current);
                q.push(path);
                path.pop_back();
            }
            current[position_x][position_y] = 0;
            current[position_x][position_y - 1] = temp;
        }

        if (position_x - 1 >= 0)
        {
            int temp = current[position_x - 1][position_y];
            current[position_x - 1][position_y] = 0;
            current[position_x][position_y] = temp;
            if (visited.find(current) == visited.end())
            {
                path.push_back(current);
                q.push(path);
                path.pop_back();
            }
            current[position_x][position_y] = 0;
            current[position_x - 1][position_y] = temp;
        }
    }
}

int main()
{
    vector<vector<int>> starting;

    cout << "Please Enter Starting State(Space Seperated)" << endl;
    for (int i = 0; i < 3; i++)
    {
        vector<int> temp;
        for (int j = 0; j < 3; j++)
        {
```



```
            int to_take;
            cin >> to_take;
            temp.push_back(to_take);
        }
        starting.push_back(temp);
    }
    cout << "Please Enter Ending State(Space Seperated)" << endl;
    vector<vector<int>> ending;
    for (int i = 0; i < 3; i++)
    {
        vector<int> temp;
        for (int j = 0; j < 3; j++)
        {
            int to_take;
            cin >> to_take;
            temp.push_back(to_take);
        }
        ending.push_back(temp);
    }
    stack<vector<vector<int>>> s;
    bool found = false;
    string choice;
    cout << "Please Enter BFS or DFS" << endl;
    cin >> choice;
    if (choice == "BFS")
    {
        bfs(starting, ending);
    }
    else
    {
        set<vector<vector<int>>> visited;
        dfs(starting, ending, s, found, visited);
    }
}
```

Output: -

```
Please Enter Starting State(Space Seperated)

3 2 1

5 4 7
```

```
8 0 6
Please Enter Ending State(Space Seperated)
2 3 1
5 6 7
0 8 4
Please Enter BFS or DFS
BFS
State : 0
3 2 1
5 4 7
8 0 6

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

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

State : 3
```

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

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

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

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

State : 7
3 1 7
5 2 6
0 8 4
```

```
State : 8

3 1 7

0 2 6

5 8 4


State : 9

3 1 7

2 0 6

5 8 4


State : 10

3 1 7

2 6 0

5 8 4


State : 11

3 1 0

2 6 7

5 8 4


State : 12

3 0 1
```

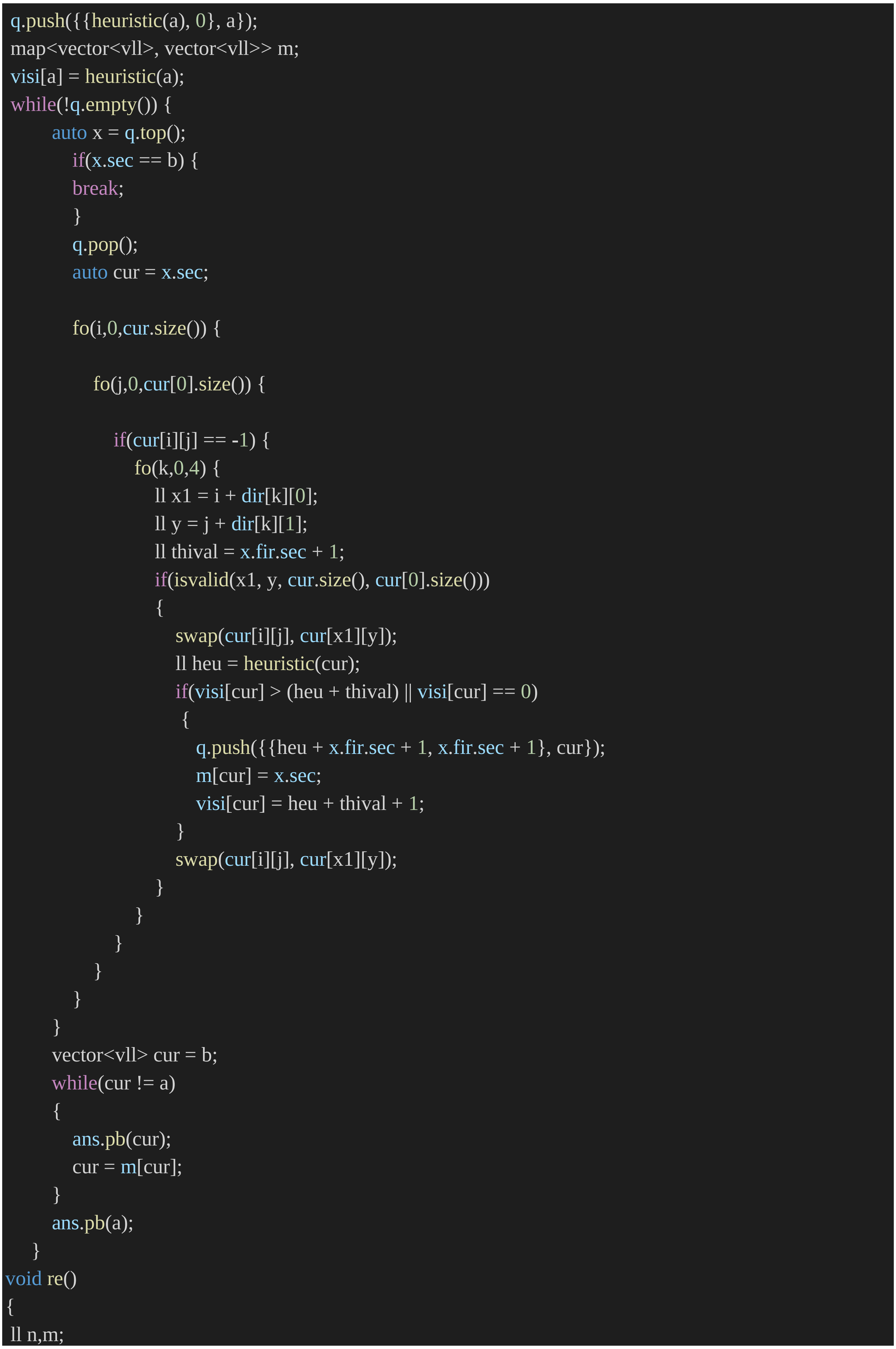
```
2 6 7
5 8 4

State : 13
0 3 1
2 6 7
5 8 4

State : 14
2 3 1
0 6 7
5 8 4

State : 15
2 3 1
5 6 7
0 8 4
```

Assignment - 7

Aim: Solve 8 puzzle problem using A* algorithm where initial state and Goal state will be given by the users.

Code:

```
#include<bits/stdc++.h>
using namespace std;
//define shortcuts
#define ll long long int
#define fo(i,a,n) for(ll i=a;i<n;i++)
#define rfo(i,a,n) for(ll i=n-1;i>=a;i--)
#define endl "\n"
#define all(a) a.begin(),a.end()
#define sz size
#define sec second
#define fir first
#define vll vector<ll>
#define pll pair<ll,ll>
#define mll map<ll,ll>
#define pb push_back
#define ite(a) for(auto &x : a)
#define read(a) ite(a) cin>>x;
#define fast ios_base::sync_with_stdio(false); cout.tie(NULL); cin.tie(NULL);
#define ri reverse_iterator
#define MOD 1000000007

//create factorial precalculate
const ll fact_size = 2e5 + 15;
vll fact1(fact_size);

vector<vll>a;
vector<vll>b;
vector<vector<vll>> ans;
map<vector<vll>, ll> visi;
vector<vll> dir = {{0, 1}, {1, 0}, {-1, 0}, {0, -1}};
bool isvalid(ll x, ll y, ll n, ll m) {
 if(x >= 0 && y >= 0 && x < n && y < m) return true;
 return false;
}
int heuristic(vector<vll> cur) {
    int val = 0;
    fo(i,0,cur.size()) {
        fo(j,0,cur[0].size()) {
            if(cur[i][j] != b[i][j]) val++;
           }
        }
    return val;
}
void sola() {
 priority_queue<pair<pll, vector<vll>>, vector<pair<pll, vector<vll>>>, greater<pair<pll, vector<vll>>>>
q;
```



```
q.push({{heuristic(a), 0}, a});
map<vector<vll>, vector<vll>> m;
visi[a] = heuristic(a);
while(!q.empty()) {
    auto x = q.top();
      if(x.sec == b) {
      break;
      }
      q.pop();
      auto cur = x.sec;

      fo(i,0,cur.size()) {

        fo(j,0,cur[0].size()) {

          if(cur[i][j] == -1) {
            fo(k,0,4) {
              ll x1 = i + dir[k][0];
              ll y = j + dir[k][1];
              ll thival = x.fir.sec + 1;
              if(isvalid(x1, y, cur.size(), cur[0].size()))
              {
                swap(cur[i][j], cur[x1][y]);
                ll heu = heuristic(cur);
                if(visi[cur] > (heu + thival) || visi[cur] == 0)
                 {
                  q.push({{heu + x.fir.sec + 1, x.fir.sec + 1}, cur});
                  m[cur] = x.sec;
                  visi[cur] = heu + thival + 1;
                }
                swap(cur[i][j], cur[x1][y]);
              }
            }
          }
        }
      }
    }
    vector<vll> cur = b;
    while(cur != a)
    {
      ans.pb(cur);
      cur = m[cur];
    }
    ans.pb(a);
  }
void re()
{
ll n,m;
```

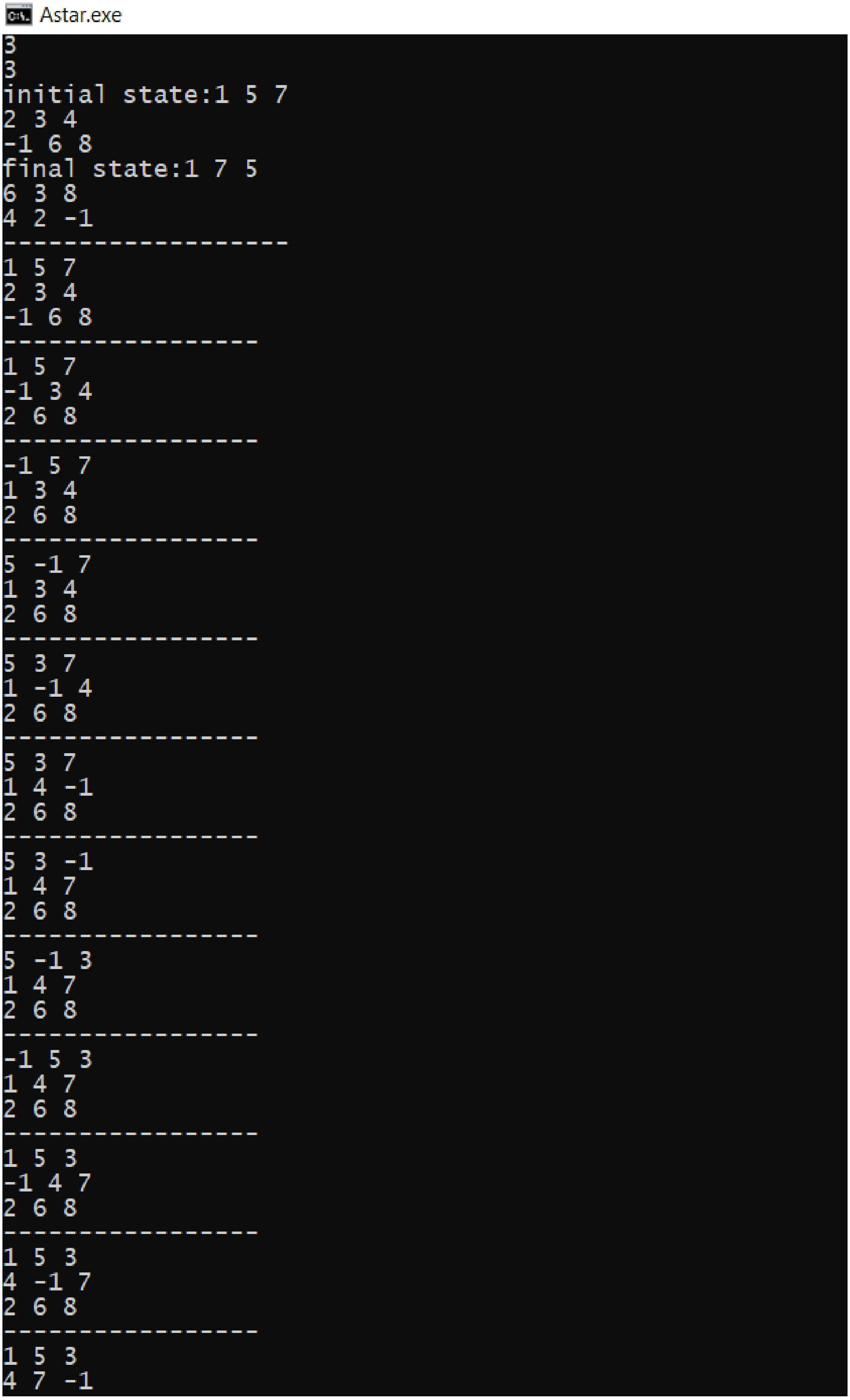
```
 cin>>n>>m;
 cout<<"initial state:";
 a.resize(n, vll(m));

 b.resize(n, vll(m));

 fo(i,0,n) read(a[i]);

 cout<<"final state:";

 fo(i,0,n) read(b[i]);

 sola();

 reverse(all(ans));

 cout<<"-------------------"<<endl;

 for(auto x: ans) {
     fo(i,0,x.size()) {
         fo(j,0,x[0].size()) {
             cout<<x[i][j]<<" ";
         }
         cout<<endl;
     }
     cout<<"-----------------"<<endl;
    }
    }
int main()
{
 ll t = 1;
 while(t--)
 {
     re();
 }
}
```

Output:

```
3
3
initial state:1 5 7
2 3 4
```

```
-1 6 8
final state:1 7 5
6 3 8
4 2 -1
--------------------
1 5 7
2 3 4
-1 6 8
------------------
1 5 7
-1 3 4
2 6 8
------------------
-1 5 7
1 3 4
2 6 8
------------------
5 -1 7
1 3 4
2 6 8
------------------
5 3 7
1 -1 4
2 6 8
------------------
5 3 7
1 4 -1
2 6 8
------------------
5 3 -1
1 4 7
2 6 8
------------------
5 -1 3
1 4 7
2 6 8
------------------
-1 5 3
1 4 7
2 6 8
------------------
1 5 3
-1 4 7
2 6 8
------------------
1 5 3
4 -1 7
2 6 8
```

```
-----------------
1 5 3
4 7 -1
2 6 8
-----------------
1 5 -1
4 7 3
2 6 8
-----------------
1 -1 5
4 7 3
2 6 8
-----------------
1 7 5
4 -1 3
2 6 8
-----------------
1 7 5
4 6 3
2 -1 8
-----------------
1 7 5
4 6 3
-1 2 8
-----------------
1 7 5
-1 6 3
4 2 8
-----------------
1 7 5
6 -1 3
4 2 8
-----------------
1 7 5
6 3 -1
4 2 8
-----------------
1 7 5
6 3 8
4 2 -1
-----------------
```



```
3
3
initial state:1 5 7
2 3 4
-1 6 8
final state:1 7 5
6 3 8
4 2 -1
-------------------
1 5 7
2 3 4
-1 6 8
-----------------
1 5 7
-1 3 4
2 6 8
-----------------
-1 5 7
1 3 4
2 6 8
-----------------
5 -1 7
1 3 4
2 6 8
-----------------
5 3 7
1 -1 4
2 6 8
-----------------
5 3 7
1 4 -1
2 6 8
-----------------
5 3 -1
1 4 7
2 6 8
-----------------
5 -1 3
1 4 7
2 6 8
-----------------
-1 5 3
1 4 7
2 6 8
-----------------
1 5 3
-1 4 7
2 6 8
-----------------
1 5 3
4 -1 7
2 6 8
-----------------
1 5 3
4 7 -1
```

Assignment - 8

Aim: WAP to design Tic Tac Toe game from O (Opponent) and X (Player) by using minimax algorithm.

About minimax algorithm:

Minimax is a kind of backtracking algorithm that is used in decision making and game theory to find the optimal move for a player, assuming that your opponent also plays optimally. It is widely used in two player turn-based games such as Tic-Tac-Toe, Backgammon, Mancala, Chess, etc.

In Minimax the two players are called maximizer and minimizer. The maximizer tries to get the highest score possible while the minimizer tries to do the opposite and get the lowest score possible.

Every board state has a value associated with it. In a given state if the maximizer has upper hand then, the score of the board will tend to be some positive value. If the minimizer has the upper hand in that board state then it will tend to be some negative value. The values of the board are calculated by some heuristics which are unique for every type of game.

Minimax algorithm for Tic-Tac-Toe:

A description for the algorithm, assuming X is the "turn taking player," would look something like:

- If the game is over, return the score from X's perspective.
- Otherwise get a list of new game states for every possible move
- Create a scores list
- For each of these states add the minimax result of that state to the scores list
- If it's X's turn, return the maximum score from the scores list
- If it's O's turn, return the minimum score from the scores list

Code for minimax algorithm for tic tac toe:

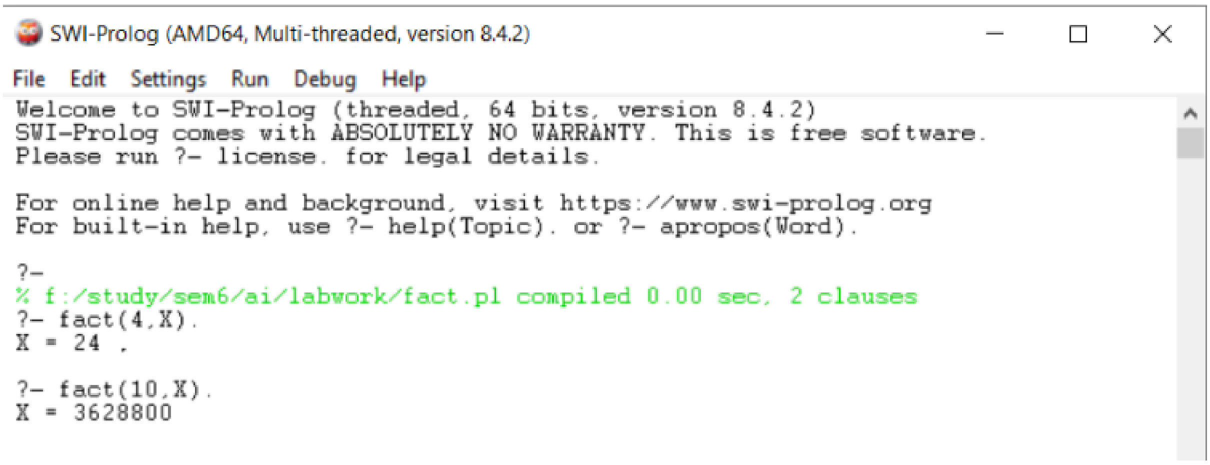
```
player, opponent = 'x', 'o'
def isMovesLeft(board) :


    for i in range(3) :
        for j in range(3) :
            if (board[i][j] == '_') :
                return True
    return False
def evaluate(b) :
    for row in range(3) :
        if (b[row][0] == b[row][1] and b[row][1] == b[row][2]) :
            if (b[row][0] == player) :
                return 10
            else if (b[row][0] == opponent) :
                return -10


    for col in range(3) :
        if (b[0][col] == b[1][col] and b[1][col] == b[2][col]) :

            if (b[0][col] == player) :
                return 10
            else if (b[0][col] == opponent) :
                return -10

    if (b[0][0] == b[1][1] and b[1][1] == b[2][2]) :

        if (b[0][0] == player) :
            return 10
        else if (b[0][0] == opponent) :
            return -10

    if (b[0][2] == b[1][1] and b[1][1] == b[2][0]) :

        if (b[0][2] == player) :
            return 10
        else if (b[0][2] == opponent) :
            return -10
    return 0
def minimax(board, depth, isMax) :
    score = evaluate(board)
    if (score == 10) :
        return score
    if (score == -10) :
        return score
    if (isMovesLeft(board) == False) :
        return 0
    if (isMax) :
        best = -1000
        for i in range(3) :
            for j in range(3) :
                if (board[i][j]=='_') :
                    board[i][j] = player
                    best = max( best, minimax(board, depth + 1,not
isMax) )
                    board[i][j] = '_'
        return best
    else :
        best = 1000
        for i in range(3) :
            for j in range(3) :
                if (board[i][j] == '_') :
                    board[i][j] = opponent
                    best = min(best, minimax(board, depth + 1, not
isMax))
                    board[i][j] = '_'
        return best
def findBestMove(board) :
    bestVal = -1000
    bestMove = (-1, -1)
    for i in range(3) :
        for j in range(3) :
            if (board[i][j] == '_') :
                board[i][j] = player
                moveVal = minimax(board, 0, False)
                board[i][j] = '_'
                if (moveVal > bestVal) :
                    bestMove = (i, j)
                    bestVal = moveVal
    print("The value of the best Move is :", bestVal)
    print()
    return bestMove
board = [
    [ 'x', 'o', 'x' ],
    [ 'o', 'o', 'x' ],
    [ '_', '_', '_' ]
]
bestMove = findBestMove(board)
print("The Optimal Move is :")
print("ROW:", bestMove[0], " COL:", bestMove[1])
```

Assignment - 9

Aim: WAP to calculate the factorial of a number by using Prolog.

Code:

```
fact(0,1).
fact(A,B):-
A>0,
C is A-1,
fact(C,D),
B is A*D.
```



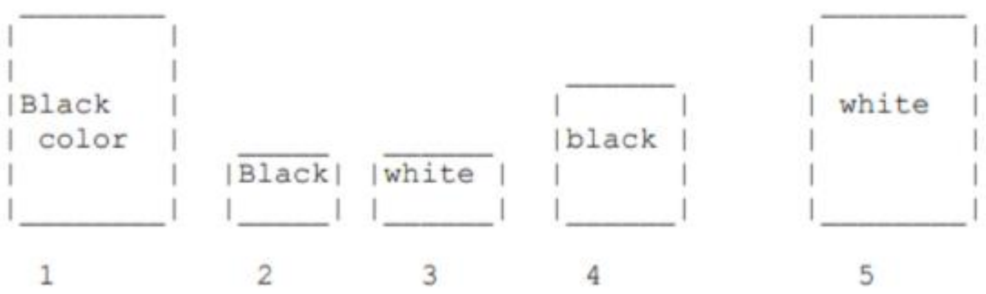
SWI-Prolog (AMD64, Multi-threaded, version 8.4.2)
File Edit Settings Run Debug Help

```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.2)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?-
% f:/study/sem6/ai/labwork/fact.pl compiled 0.00 sec, 2 clauses
?- fact(4,X).
X = 24 .

?- fact(10,X).
X = 3628800
```

Assignment - 10

Aim: WAP to solve Box Solver problem, which is given in the attached file

There are five boxes:



Anay, Bill, Charlie, Don, Eric own one box each but we don't know which box. Try to find each boxes' owner if you know that:

Anay and Bill have boxes with the same color.

Don and Eric have boxes with the same color.

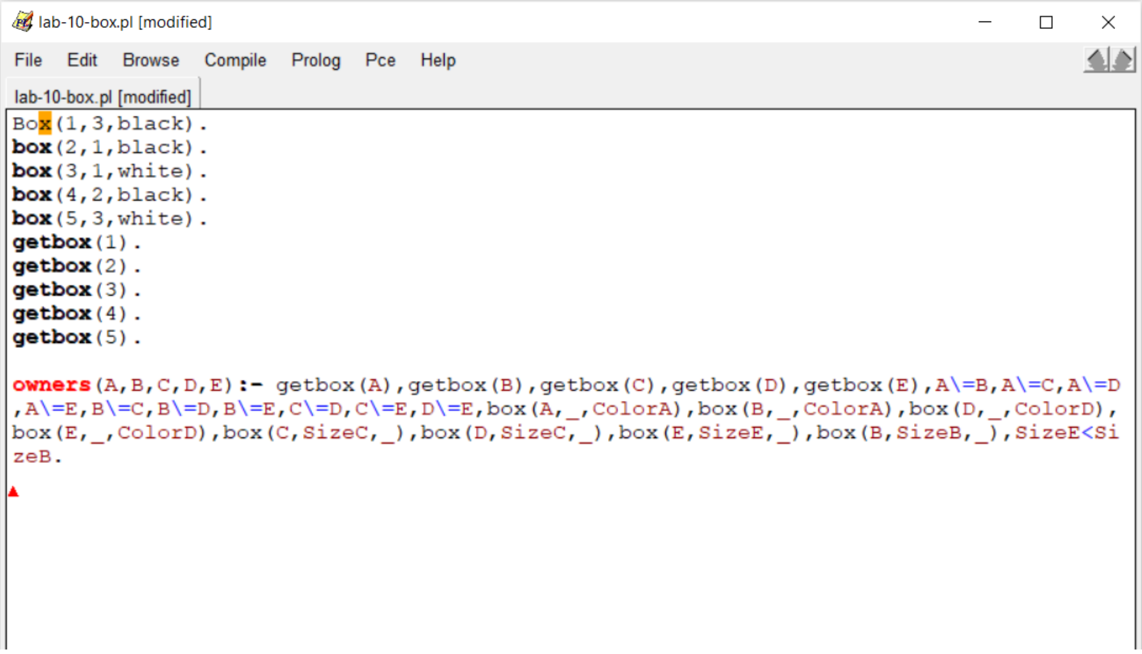
Charlie and Don have boxes with the same size.

Eric's box is smaller than Bill's

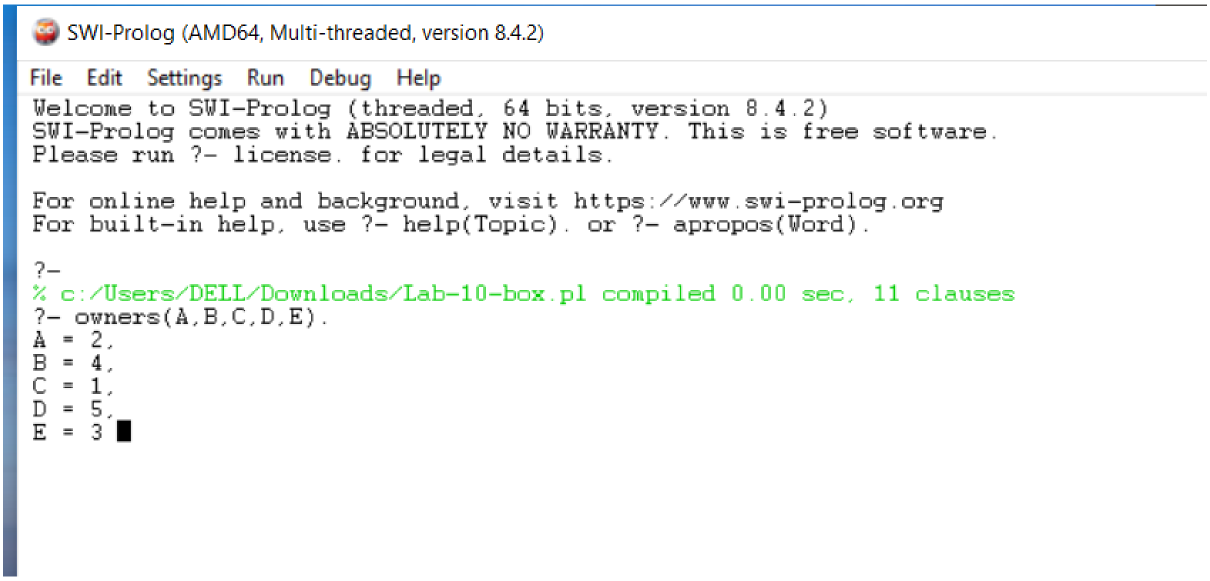
Code:

```
(1,3,black).

box(2,1,black).

box(3,1,white).

box(4,2,black).

box(5,3,white).

getbox(1).

getbox(2).

getbox(3).

getbox(4).

getbox(5).
```

owners(A,B,C,D,E):-
getbox(A),getbox(B),getbox(C),getbox(D),getbox(E),A\=B,A\=C,A\=D,A\=E,B\=C,B\=D,B\=E,C\=D,C\=E,D\=E,box(A,_,ColorA),box(B,_,ColorA),box(D,_,ColorD),box(E,_,ColorD),box(C,SizeC,_),box(D,SizeC,_),box(E,SizeE,_),box(B,SizeB,_),SizeE<SizeB.



```
Box(1,3,black).
box(2,1,black).
box(3,1,white).
box(4,2,black).
box(5,3,white).
getbox(1).
getbox(2).
getbox(3).
getbox(4).
getbox(5).

owners(A,B,C,D,E):- getbox(A),getbox(B),getbox(C),getbox(D),getbox(E),A\=B,A\=C,A\=D
,A\=E,B\=C,B\=D,B\=E,C\=D,C\=E,D\=E,box(A,_,ColorA),box(B,_,ColorA),box(D,_,ColorD),
box(E,_,ColorD),box(C,SizeC,_),box(D,SizeC,_),box(E,SizeE,_),box(B,SizeB,_),SizeE<Si
zeB.
```



SWI-Prolog (AMD64, Multi-threaded, version 8.4.2)
File Edit Settings Run Debug Help

```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.2)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?-
% c:/Users/DELL/Downloads/Lab-10-box.pl compiled 0.00 sec, 11 clauses
?- owners(A,B,C,D,E).
A = 2,
B = 4,
C = 1,
D = 5,
E = 3
```

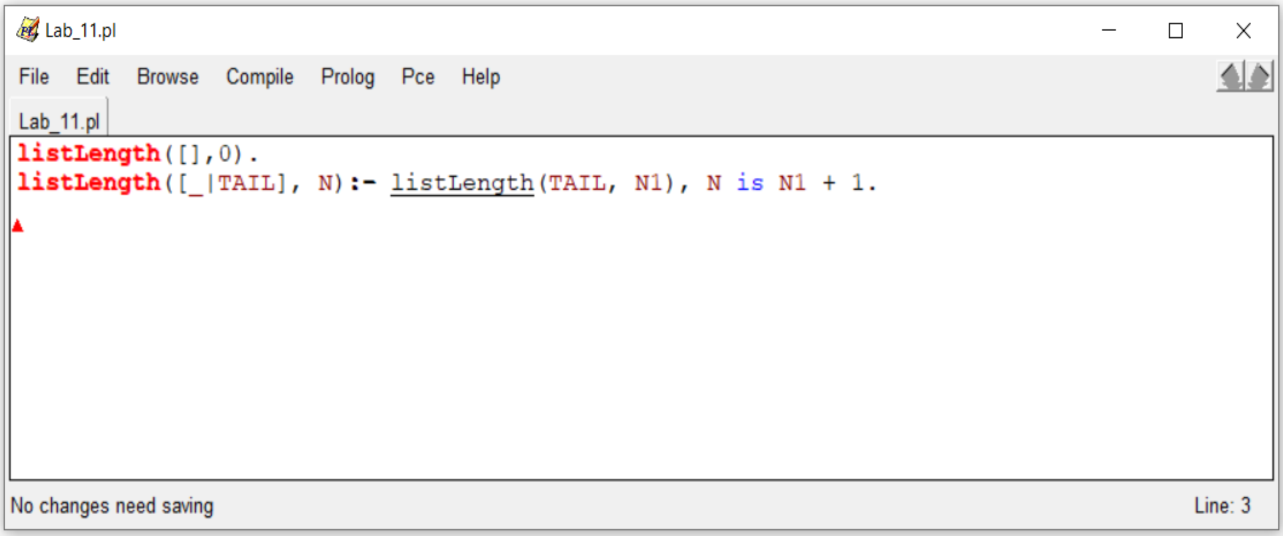
Assignment – 11

Aim: WAP to find the length of the list using Prolog.

Code:

listLength([],0).

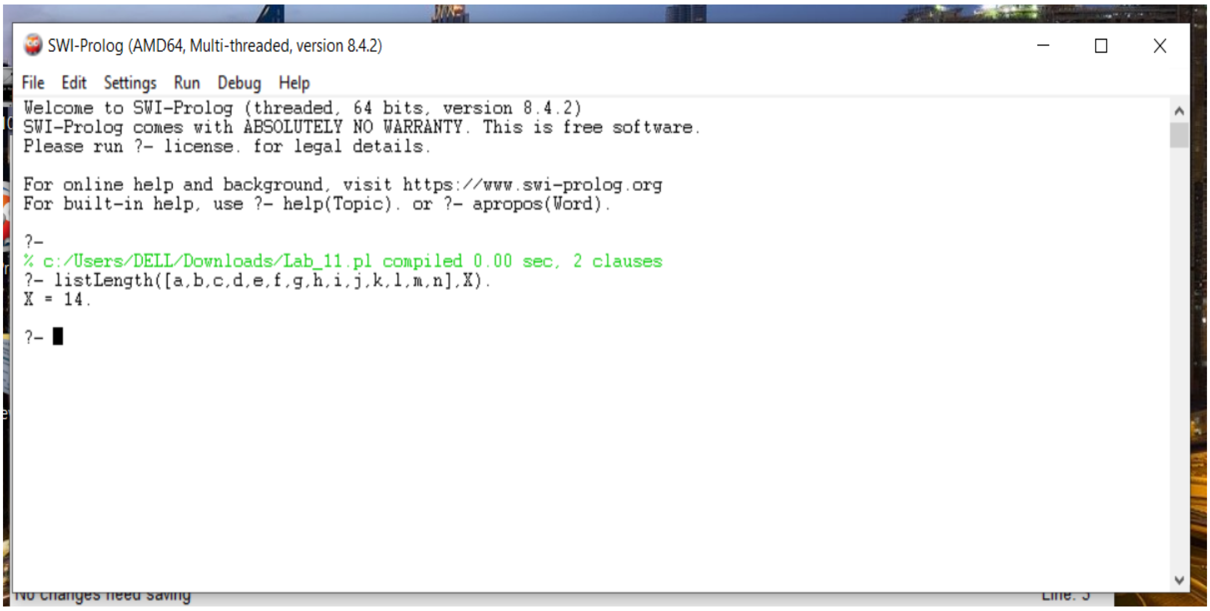
listLength([_|TAIL], N):- listLength(TAIL, N1), N is N1 + 1.



```
listLength([],0).
listLength([_|TAIL], N):- listLength(TAIL, N1), N is N1 + 1.
```

No changes need saving

Line: 3



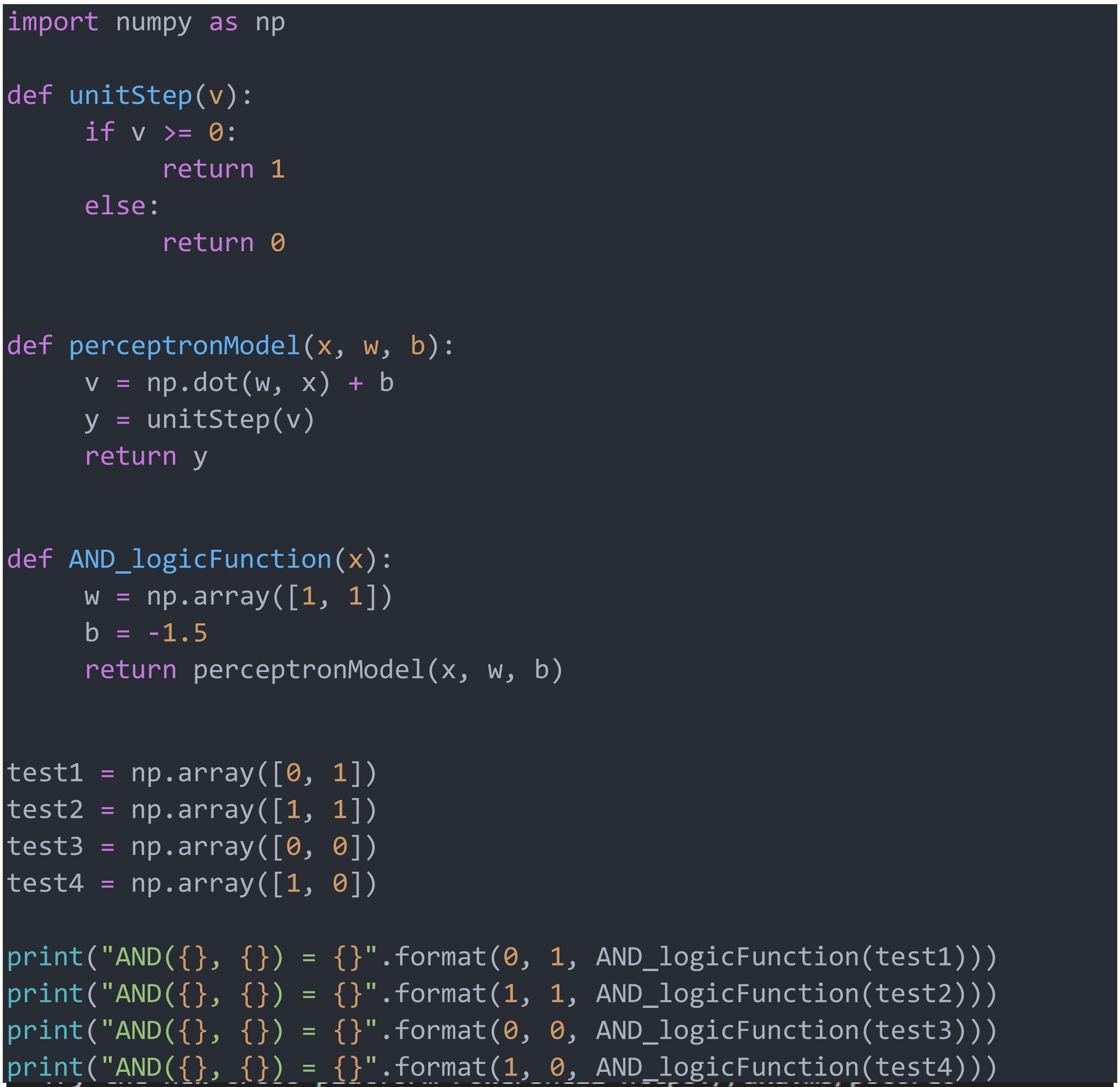
SWI-Prolog (AMD64, Multi-threaded, version 8.4.2)
File Edit Settings Run Debug Help

```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.2)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?-
% c:/Users/DELL/Downloads/Lab_11.pl compiled 0.00 sec, 2 clauses
?- listLength([a,b,c,d,e,f,g,h,i,j,k,l,m,n],X).
X = 14.

?-
```

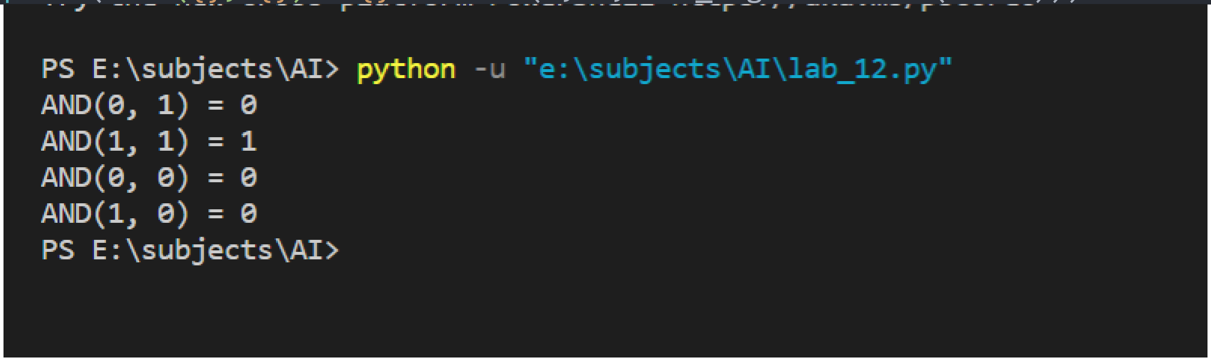
Assignment – 12

Aim: WAP to implement AND logic Gate using perceptron neural network.

Code:



```python
import numpy as np

def unitStep(v):
     if v >= 0:
          return 1
     else:
          return 0


def perceptronModel(x, w, b):
     v = np.dot(w, x) + b
     y = unitStep(v)
     return y


def AND_logicFunction(x):
     w = np.array([1, 1])
     b = -1.5
     return perceptronModel(x, w, b)


test1 = np.array([0, 1])
test2 = np.array([1, 1])
test3 = np.array([0, 0])
test4 = np.array([1, 0])

print("AND({}, {}) = {}".format(0, 1, AND_logicFunction(test1)))
print("AND({}, {}) = {}".format(1, 1, AND_logicFunction(test2)))
print("AND({}, {}) = {}".format(0, 0, AND_logicFunction(test3)))
print("AND({}, {}) = {}".format(1, 0, AND_logicFunction(test4)))
```



```
PS E:\subjects\AI> python -u "e:\subjects\AI\lab_12.py"
AND(0, 1) = 0
AND(1, 1) = 1
AND(0, 0) = 0
AND(1, 0) = 0
PS E:\subjects\AI>
```

Output:

AND(0, 1) = 0

AND(1, 1) = 1

AND(0, 0) = 0

AND(1, 0) = 0

Assignment - 13

Aim: Design a Convolutional Neural Network from Scratch for MNIST fashion dataset. Apply dropout technique to deal with the overfitting. Dataset can be downloaded from the below link

https://www.kaggle.com/datasets/zalando-research/fashionmnist

Code:

```
import tensorflow as tf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from google.colab import files
uploaded = files.upload()
import io
df = pd.read_csv(io.BytesIO(uploaded['fashion-mnist_test.csv']))
df1 = pd.read_csv(io.BytesIO(uploaded['fashion-mnist_train.csv']))
train = np.array(df1, dtype='float32')
test = np.array(df, dtype='float32')
xtrain = train[:,1:]/255
ytrain = train[:,0]

xtest = test[:,1:]/255
ytest = test[:,0]
xtrain=xtrain.reshape(xtrain.shape[0],*(28,28,1))
xtest=xtest.reshape(xtest.shape[0],*(28,28,1))
xtrain.shape
xtest.shape
from keras.models import Sequential
from keras.layers import Conv2D, MaxPool2D, Flatten, Dense, Dropout
model = Sequential()
model.add(Conv2D(32, (2,2), activation='relu', input_shape=(28,28,1)) )
model.add(MaxPool2D(2,2))

model.add(Conv2D(64, (2,2), activation='relu') )
model.add(MaxPool2D(2,2))

model.add(Conv2D(64, (2,2), activation='relu') )

model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(10, activation='softmax'))
model.summary()


model.compile(loss='sparse_categorical_crossentropy', optimizer='Adam',
metrics=['accuracy'])


# In[15]:


history = model.fit(xtrain, ytrain, batch_size=512, epochs=5, verbose=1)


# In[16]:


plt.plot(history.history['loss'])


# In[17]:
```

```
plt.plot(history.history['accuracy'])


# In[18]:


ypred = model.predict(xtest)


# In[19]:


ypred.shape


# In[20]:


yclass = np.argmax(ypred, axis=1)


# In[21]:


yclass.shape


# In[22]:


from sklearn.metrics import confusion_matrix, classification_report,
accuracy_score


# In[23]:


accuracy_score(ytest, yclass)


# In[24]:


mat = confusion_matrix(ytest, yclass)
mat


# In[25]:


sns.heatmap(mat, annot=True , fmt='d')


# In[26]:


plt.figure(figsize=(14,10))
sns.heatmap(mat, annot=True , fmt='d')
```

```
# In[27]:


classification_report(ytest, yclass)


# In[ ]:
```