1. Rule Based Agent

a. Family Tree

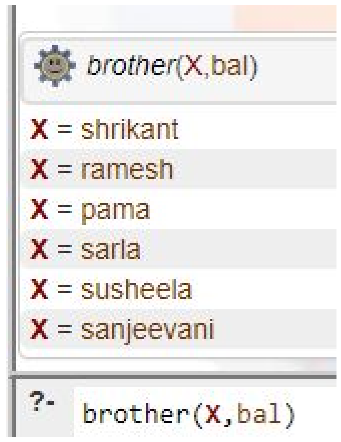
```
male(prabhodankar).
male(bal).
male(shrikant).
male(ramesh).
male(bindumadhav).
male(uddhav).
male(jaidev).
male(raj).
male(aditya).
male(amit).

female(ramabai).
female(meena).
female(kunda).
female(pama).
female(sarla).
female(susheela).
female(sanjeevani).
female(rashmi).
female(sharmila).
female(jaywanti).
female(anu).
female(urvashi).
female(mitali).

father(bal,prabhodankar).
father(shrikant,prabhodankar).
father(ramesh,prabhodankar).
father(pama,prabhodankar).
father(sarla,prabhodankar).
father(susheela,prabhodankar).
father(sanjeevani,prabhodankar).
father(bindumadhav,bal).
father(uddhav,bal).
father(jaidev,bal).
father(aditya,uddhav).
father(anu,uddhav).
father(raj,shrikant).
father(jaywanti,shrikant).
father(amit,raj).
father(urvashi,raj).

wife(prabhodankar,ramabai).
wife(bal,meena).
wife(shrikant,kunda).
wife(uddhav,rashmi).
wife(raj,sharmila).
wife(amit,mitali).
```

Rules:

```
mother(X,Y):- father(X,Z),wife(Z,Y).
daughter(X,Y):- father(X,Y),male(Y),female(X).
husband(X,Y):- wife(Y,X).

brother(X,Y):- father(X,Z),father(Y,Z),male(Y),(X\=Y).
sister(X,Y):- father(X,Z),father(Y,Z),female(Y),(X\=Y).

grandfather(X,Y):- father(X,Z),father(Z,Y),male(Y).
grandmother(X,Y):- father(X,Z),mother(Z,Y),female(Y).

daughter-in-law(X,Y):- father(Z,X),wife(Z,Y),male(X),female(Y).
father-in-law(X,Y):-  husband(X,Z),father(Z,Y).
mother-in-law(X,Y):-  husband(X,Z),mother(Z,Y).

great-grandfather(X,Y):- grandfather(X,Z),father(Z,Y).
great-grandmother(X,Y):- grandfather(X,Z),mother(Z,Y).

uncle(X,Y):- father(X,Z),brother(Z,Y).
aunt(X,Y):- father(X,Z),sister(Z,Y).

cousin(X,Y):- father(X,Z),father(Y,J),brother(Z,J).
```

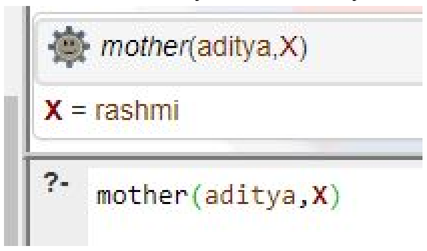
<u>Some Sample queries and Outputs:</u>

Find all brothers of Bal Thackeray.



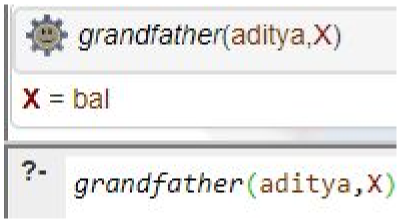
```
brother(X,bal)
X = shrikant
X = ramesh
X = pama
X = sarla
X = susheela
X = sanjeevani

?- brother(X,bal)
```

Mother of Aditya Thackeray



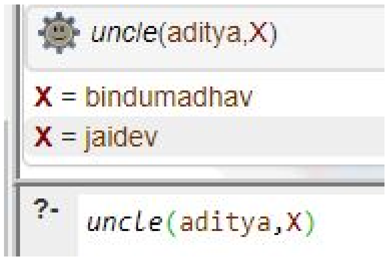
```
mother(aditya,X)
X = rashmi
?- mother(aditya,X)
```

Who is grandfather of Aditya



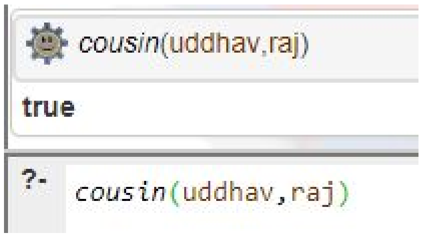
```
grandfather(aditya,X)
X = bal
?- grandfather(aditya,X)
```

Who are the uncle of Aditya



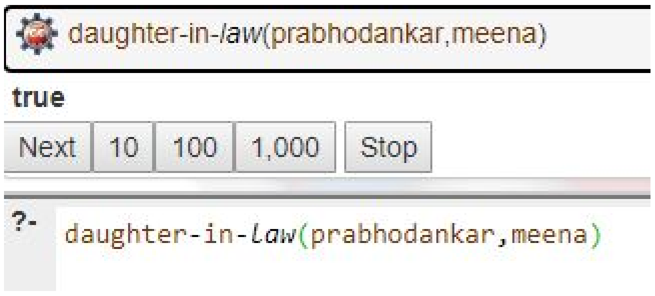
```
uncle(aditya,X)
X = bindumadhav
X = jaidev
?- uncle(aditya,X)
```

Is raj cousin of Uddhav Thackeray



```
cousin(uddhav,raj)
true
?- cousin(uddhav,raj)
```

Is meena Daughter-in-law of prabhodankar



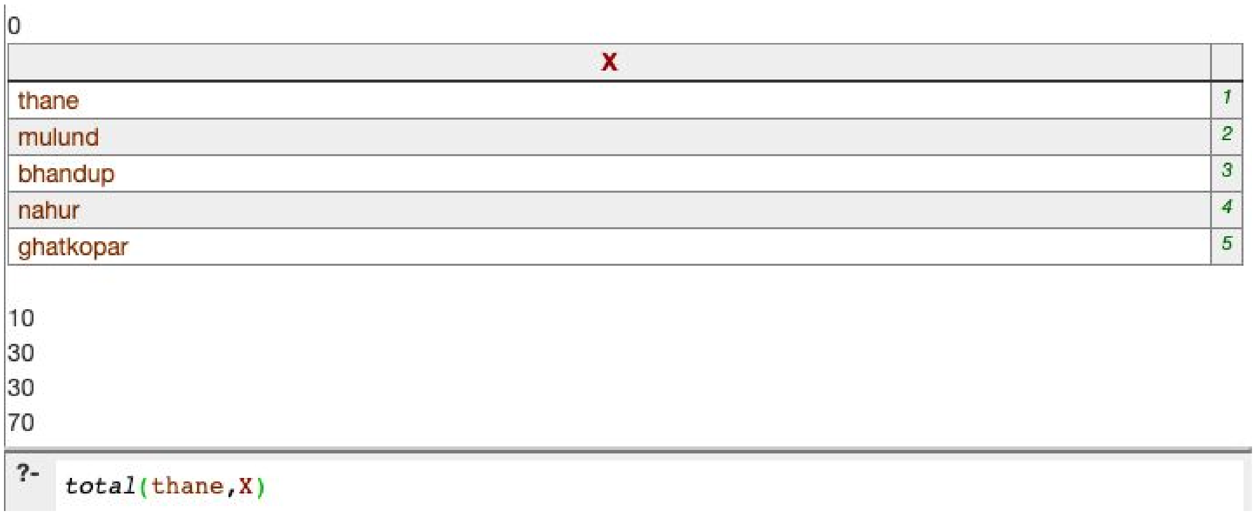
```
daughter-in-law(prabhodankar,meena)
true
?- daughter-in-law(prabhodankar,meena)
```

b. Distance between cities

```
distance(thane, thane, 0).
distance(thane, mulund, 10).
distance(thane, bhandup, 30).
distance(mulund, nahur, 20).
distance(nahur, bhandup, 20).
distance(bhandup, ghatkopar, 40).

connected(X,Y,Z):- distance(X,Y,Z).
%direct only
total(X,Y):- connected(X,Y,Z), write(Z).
%direct and indirect
total(X,Y):- connected(X,Z,A),connected(Z,Y,B), Sum is A+B,nl,write(Sum).
```

QUERY:

```
total(thane,X)
```



```
0
```

X	
thane	1
mulund	2
bhandup	3
nahur	4
ghatkopar	5

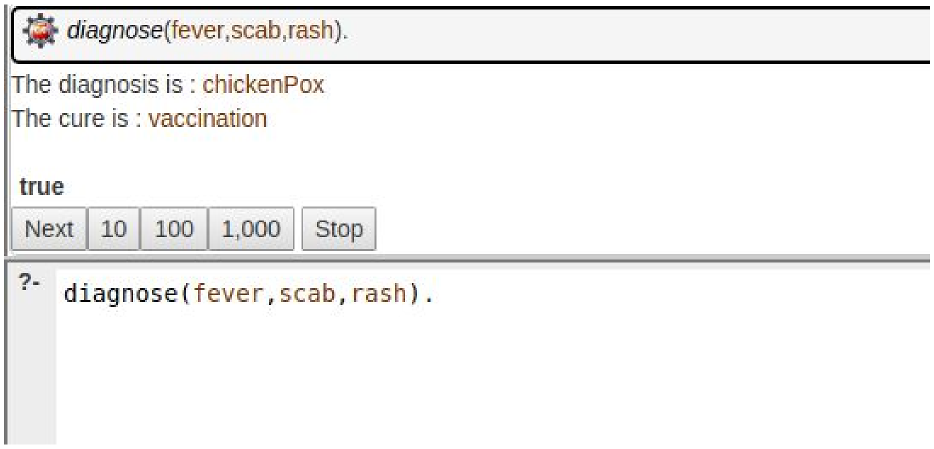
```
10
30
30
70
?- total(thane,X)
```

c. Medical Diagnosis system

Base Knowledgebase:

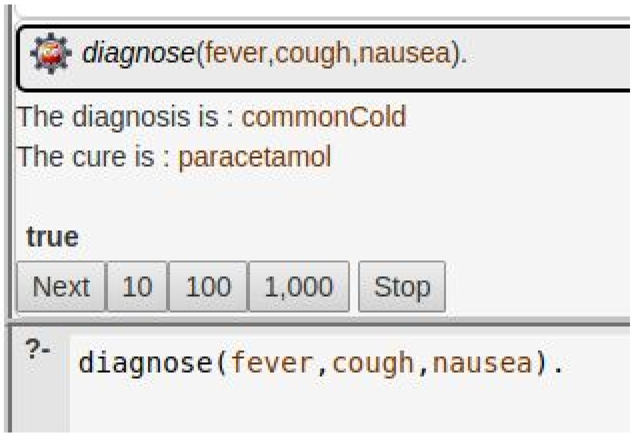
```
symptom(ChickenPox,malaise).
symptom(ChickenPox,fever).
symptom(ChickenPox,scab).
symptom(ChickenPox,rash).
cure(ChickenPox,vaccination).

symptom(Measles,cold).
symptom(Measles,fever).
symptom(Measles,conjunctivitis).
symptom(Measles,nausea).
cure(Measles,MMR_vaccination).

symptom(commonCold,cough).
symptom(commonCold,cold).
symptom(commonCold,fever).
symptom(commonCold,headache).
symptom(commonCold,nausea).
cure(commonCold,paracetamol).

diagnose(S1 , S2 , S3) :-
     symptom(D , S1),
     symptom(D , S2),
     symptom(D , S3),

     cure(D , C),
     write("The diagnosis is : "),
     write(D),
     nl,
     write("The cure is : "),
     write(C),
     nl   , nl .
```

Some Sample queries and Outputs:



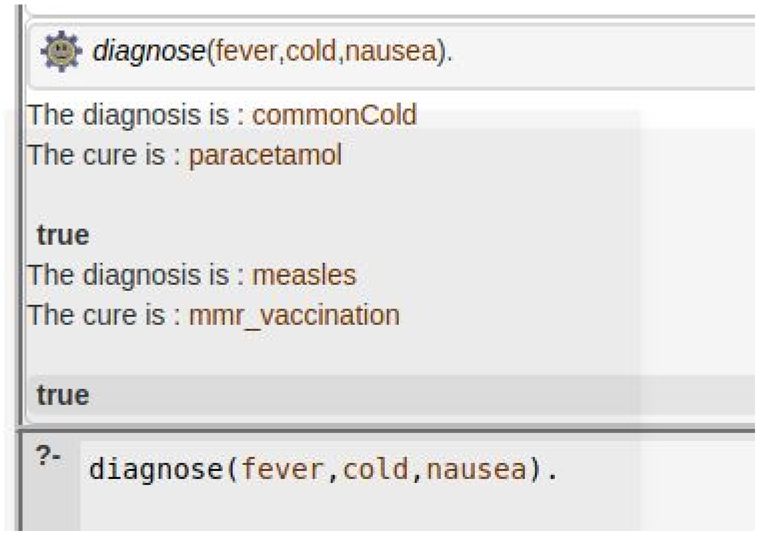
```
diagnose(fever,scab,rash).
The diagnosis is : chickenPox
The cure is : vaccination

true
```

```
?- diagnose(fever,scab,rash).
```



```
diagnose(fever,cough,nausea).
The diagnosis is : commonCold
The cure is : paracetamol

true
```

```
?- diagnose(fever,cough,nausea).
```



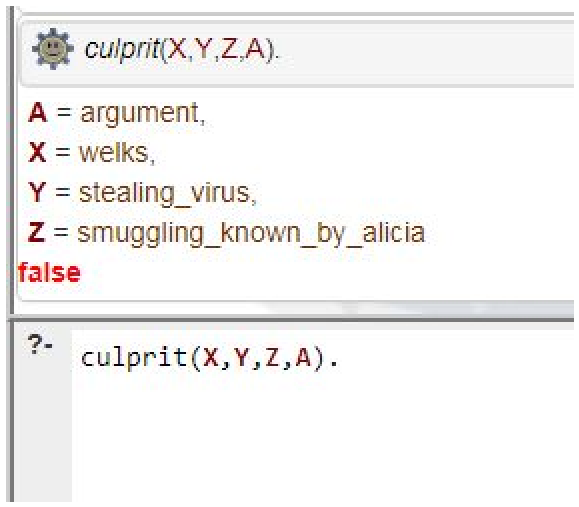
```
diagnose(fever,cold,nausea).
The diagnosis is : commonCold
The cure is : paracetamol

true
The diagnosis is : measles
The cure is : mmr_vaccination

true
?- diagnose(fever,cold,nausea).
```

2. Goal Based Agent

```
people(cliff).
people(welks).
people(tripp).
people(lilith).
people(price).
people(cleaner).
people(accountant).

weapon_access(cliff).
weapon_access(welks).
weapon_access(tripp).
weapon_access(alicia).
weapon_access(lilith).

motive(cliff,jealousy).
motive(welks,argument).
motive(lilith,workload).

location(cliff,home).
location(welks,research_center).
location(tripp,research_center).
location(lilith,home).

digging_past(welks, smuggling_known_by_alicia).
digging_past(lilith, jealousy_Due_To_Promotions).
digging_past(cliff, open_Relationship).
digging_past(price, backmailed_By_Alicia).

antidote_taken(welks).

evidence_found_against(price, camera_not_working).
evidence_found_against(lilith, tried_to_run).
evidence_found_against(cliff, mairage_not_working).
evidence_found_against(welks, stealing_virus).

suspect_By_Weapon_Access(X) :- people(X), weapon_access(X).

suspect_By_Motive(X,Y) :- people(X), motive(X,Y).

suspect_By_Location(X) :- people(X), location(X, research_center).

suspect(X,Y) :- suspect_By_Weapon_Access(X), suspect_By_Motive(X,Y),
suspect_By_Location(X).
```

```
narrowing_suspect(X,Y,Z,P) :- suspect(X,P), evidence_found_against(X, Y),digging_past(X,Z).

culprit(X,Y,Z,P) :-
    narrowing_suspect(X,Y,Z,P),
    antidote_taken(X).
```

OUTPUT



```
culprit(X,Y,Z,A).

A = argument,
X = welks,
Y = stealing_virus,
Z = smuggling_known_by_alicia
false

?- culprit(X,Y,Z,A).
```

3. Uninformed Search

a. 8 Puzzle

```
import java.util.*;


class Main {

 public static ArrayList<State> states = new ArrayList<>();

 public static ArrayList<Integer> visited = new ArrayList<>();

 public static ArrayList<State> path = new ArrayList<>();

 public static int counter = 0;

 public static int max=21;

 public static int depth = 0;

        public    static    State    initialState    =    new    State(new
int[]{1,2,3,4,5,6,7,8,0},0);

         public    static    State    finalState    =    new    State(new
int[]{1,2,3,4,0,5,7,8,6});

      public     static     State     end     =     new     State(new
int[]{-1,-1,-1,-1,-1,-1,-1,-1,-1});

 public static void main(String[] args) {

   System.out.print("\nINITIAL STATE:");

   for(int j=0;j<9;j++){

     if(j==0||j==3||j==6){

       System.out.println();

     }

     System.out.print(initialState.pos[j]+" ");

   }

   System.out.print("\n\nGOAL STATE:");

   for(int j=0;j<9;j++){

     if(j==0||j==3||j==6){
       System.out.println();
     }
     System.out.print(finalState.pos[j]+" ");
   }
   states.add(initialState);
   expand();
   int i=0;
   while(i<states.size()){
     System.out.println("\n\nLevel "+states.get(i).level);
     if(i==0){
       State st = states.get(i);
       for(int j=0;j<9;j++){
         if(j==0||j==3||j==6){
           System.out.println();
         }
         if(j==0){
           System.out.println(i);
         }
         System.out.print(st.pos[j]+" ");
       }
       i++;
     }
     else{
       for(int k=0;k<4;k++){
       State st = states.get(i);
         for(int j=0;j<9;j++){
```

```
          if(j==0||j==3||j==6){
            System.out.println();
          }
          if(j==0){
            System.out.println(i);
          }
          System.out.print(st.pos[j]+" ");
        }
      i++;
      System.out.println();
    }
    }
  }
  bfs();
  dfs();
  dls();
  ids();
}

static void expand(){
  while(counter<max){
    left(states.get(counter));
    right(states.get(counter));
    up(states.get(counter));
    down(states.get(counter));
    counter++;
  }
}

static void left(State state){
  State tempState = new State(state.pos, state.level+1);
  int found=-1;
  int f=0;
  for(int i=0;i<9;i++){
    if(tempState.pos[i]==0){
      found = i;
      break;
    }
  }
  if(found%3!=0&&found!=-1){
    int tempPos[] = new int[9];
    for(int j=0;j<9;j++){
      tempPos[j] = tempState.pos[j];
    }
    int t = tempPos[found];
    tempPos[found] = tempPos[found-1];
    tempPos[found-1] = t;
    tempState.pos = tempPos;
    for(int k=0;k<states.size();k++){
      if(Arrays.equals(states.get(k).pos, tempState.pos)){
        f=1;
      }
    }
    if(f==0){
      states.add(tempState);
    }
    else{
      tempPos = new int[]{-1,-1,-1,-1,-1,-1,-1,-1,-1};
      tempState.pos = tempPos;
      states.add(tempState);
    }
  }
  else{
    int tempPos[] = new int[]{-1,-1,-1,-1,-1,-1,-1,-1,-1};
    tempState.pos = tempPos;
    states.add(tempState);
  }
}

static void right(State state){
  State tempState = new State(state.pos, state.level+1);
  int found=-1;
  int f=0;
  for(int i=0;i<9;i++){
    if(tempState.pos[i]==0){
      found = i;
      break;
    }
```

```
}
if(found%3!=2&&found!=-1){
  int tempPos[] = new int[9];
  for(int j=0;j<9;j++){
    tempPos[j] = tempState.pos[j];
  }
  int t = tempPos[found];
  tempPos[found] = tempPos[found+1];
  tempPos[found+1] = t;
  tempState.pos = tempPos;
  for(int k=0;k<states.size();k++){
    if(Arrays.equals(states.get(k).pos, tempState.pos)){
      f=1;
    }
  }
  if(f==0){
    states.add(tempState);
  }
  else{
    tempPos = new int[]{-1,-1,-1,-1,-1,-1,-1,-1,-1};
    tempState.pos = tempPos;
    states.add(tempState);
  }
}
else{
  int tempPos[] = new int[]{-1,-1,-1,-1,-1,-1,-1,-1,-1};
```

```
      tempState.pos = tempPos;

      states.add(tempState);

    }

  }


   static void up(State state){

    State tempState = new State(state.pos, state.level+1);

    int found=-1;

    int f=0;

    for(int i=0;i<9;i++){

      if(tempState.pos[i]==0){

        found = i;

        break;

      }

    }

    if(found>2&&found!=-1){

      int tempPos[] = new int[9];

      for(int j=0;j<9;j++){

        tempPos[j] = tempState.pos[j];

      }

      int t = tempPos[found];

      tempPos[found] = tempPos[found-3];

      tempPos[found-3] = t;

      tempState.pos = tempPos;

      for(int k=0;k<states.size();k++){

        if(Arrays.equals(states.get(k).pos, tempState.pos)){
          f=1;

        }

      }

      if(f==0){

        states.add(tempState);

      }

      else{

        tempPos = new int[]{-1,-1,-1,-1,-1,-1,-1,-1,-1};

        tempState.pos = tempPos;

        states.add(tempState);

      }

    }

    else{

      int tempPos[] = new int[]{-1,-1,-1,-1,-1,-1,-1,-1,-1};

      tempState.pos = tempPos;

      states.add(tempState);

    }

  }


   static void down(State state){

    State tempState = new State(state.pos, state.level+1);

    int found=-1;

    int f=0;

    for(int i=0;i<9;i++){

      if(tempState.pos[i]==0){

        found = i;
  break;
}
}
if(found<6&&found!=-1){
int tempPos[] = new int[9];
for(int j=0;j<9;j++){
  tempPos[j] = tempState.pos[j];
}
int t = tempPos[found];
tempPos[found] = tempPos[found+3];
tempPos[found+3] = t;
tempState.pos = tempPos;
for(int k=0;k<states.size();k++){
  if(Arrays.equals(states.get(k).pos, tempState.pos)){
    f=1;
  }
}
if(f==0){
  states.add(tempState);
}
else{
  tempPos = new int[]{-1,-1,-1,-1,-1,-1,-1,-1,-1};
  tempState.pos = tempPos;
  states.add(tempState);
}
}
```

```
    else{
      int tempPos[] = new int[]{-1,-1,-1,-1,-1,-1,-1,-1,-1};
      tempState.pos = tempPos;
      states.add(tempState);
    }
  }

  static void bfs(){
    int posi=0;
    int found=0;
    visited = new ArrayList<>();
    visited.add(posi);
    int top=0;
    StringBuffer sb = new StringBuffer();
    System.out.println("\nBFS");
    //System.out.print("PATH: ");
    while(found==0){
      top = visited.get(0);
      sb.append(top+"  ");

 if(top<max){//!(Arrays.equals(states.get(top).pos,end.pos))&&top<max){
        if(Arrays.equals(states.get(top).pos, finalState.pos)){
          found=1;
        }
        visited.remove(0);
        posi = posi+1;
```

```
          // if(!(Arrays.equals(states.get(posi).pos,end.pos))){
              visited.add(0,posi);
          //}
          // for(int i=3;i>=0;i--){
          //   if(!(Arrays.equals(states.get(posi+i).pos,end.pos))){
          //     visited.add(0,posi+i);
          //   }
          // }
        }
        else{
          visited.remove(0);
        }
        if(visited.size()==0){
          break;
        }
      }
      if(found==1){
        System.out.print("PATH: "+sb);
        System.out.println("\nFOUND AT: "+top);
      }
      else{
        System.out.println("No solution found!");
      }
    }

     static void dfs(){
```

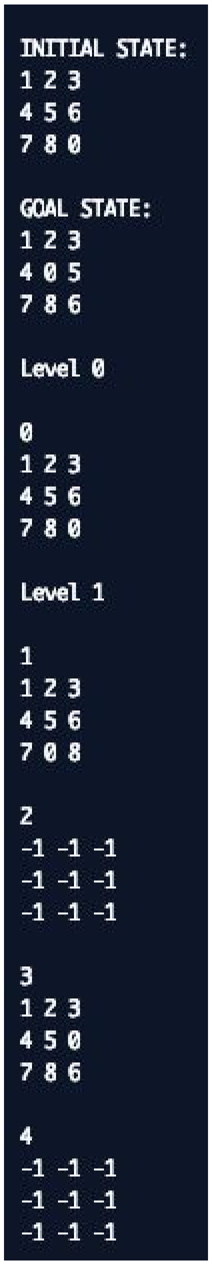
```
int posi=0;

int found=0;

visited = new ArrayList<>();

visited.add(posi);

int top=0;

StringBuffer sb = new StringBuffer();

System.out.println("\nDFS");

//System.out.print("PATH: ");

while(found==0){

  top = visited.get(0);

  sb.append(top+" ");

  if(!(Arrays.equals(states.get(top).pos,end.pos))&&top<max){

    if(Arrays.equals(states.get(top).pos, finalState.pos)){

      found=1;

    }

    visited.remove(0);

    posi = top*4+1;

    for(int i=3;i>=0;i--){

      if(!(Arrays.equals(states.get(posi+i).pos,end.pos))){

        visited.add(0,posi+i);

      }

    }

  }

  else{

    visited.remove(0);

  }
```

```java
        if(visited.size()==0){
          break;
        }
      }
      if(found==1){
        System.out.print("PATH: "+sb);
        System.out.println("\nFOUND AT: "+top);
      }
      else{
        System.out.println("No solution found!");
      }
    }

    static void dls(){
      max=0;
      int posi=0;
      int found=0;
      visited = new ArrayList();
      visited.add(posi);
      StringBuffer sb = new StringBuffer();
      int top=0;
      System.out.println("\n\nEnter the max depth allowed: ");
      Scanner sc = new Scanner(System.in);
      depth = sc.nextInt();
      for(int i=0;i<depth;i++){
        max = max+(int)Math.pow(4,i);
```

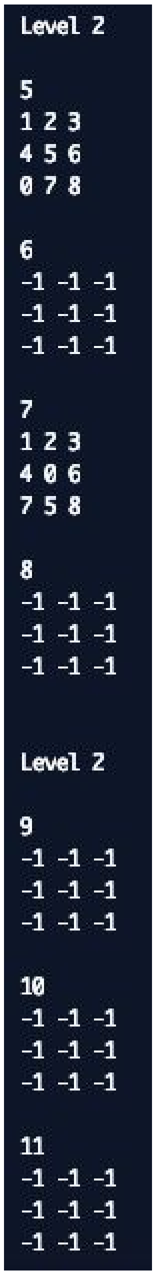
```
}
System.out.println("\nDLS");
while(found==0){
  top = visited.get(0);
  sb.append(top+" ");
  if(!(Arrays.equals(states.get(top).pos,end.pos))&&top<max){
    if(Arrays.equals(states.get(top).pos, finalState.pos)){
        found=1;
    }
    visited.remove(0);
    posi = top*4+1;
    for(int i=3;i>=0;i--){
      if(!(Arrays.equals(states.get(posi+i).pos,end.pos))){
        visited.add(0,posi+i);
      }
    }
  }
  else{
    visited.remove(0);
  }
  if(visited.size()==0){
    break;
  }
}
if(found==1){
  System.out.print("PATH: "+sb);
```

```
      System.out.println("\nFOUND AT: "+top);
    }
    else{
      System.out.println("No solution found!");
    }
  }

  static void ids(){
    System.out.println("\nIDS");
    depth = 0;
    int top=0;
    int found=0;
    StringBuffer sb = new StringBuffer();
    while(found==0){
    max=0;
    int posi=0;
    visited = new ArrayList();
    visited.add(posi);
    sb = new StringBuffer();
    top=0;
    depth++;
    if(depth==0){
      max=1;
    }
    else{
      for(int i=0;i<depth;i++){
        max = max+(int)Math.pow(4,i);
      }
    }
    while(found==0&&top<max){
      top = visited.get(0);
      sb.append(top+" ");
      if(!(Arrays.equals(states.get(top).pos,end.pos))&&top<max){
        if(Arrays.equals(states.get(top).pos, finalState.pos)){
            found=1;
        }
        visited.remove(0);
        posi = top*4+1;
        if(posi<max){
        for(int i=3;i>=0;i--){
          if(!(Arrays.equals(states.get(posi+i).pos,end.pos))){
            visited.add(0,posi+i);
          }
        }
        }
      }
      else{
        visited.remove(0);
      }
      if(visited.size()==0){
        break;
      }
    }
  }
  if(found==1){
      System.out.print("PATH: "+sb);
      System.out.println("\nFOUND AT: "+top);
    }
    else{
      System.out.println("No solution found!");
    }
  }
}

class State{
  int pos[] = new int[9];
  int level;
  public State(int pos[]){
    this.pos = pos;
  }
  public State(int pos[], int level){
    this.pos = pos;
    this.level = level;
  }
}
```

Solution Tree:

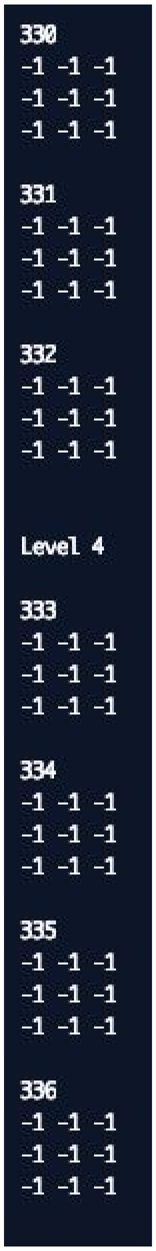


```
INITIAL STATE:
1 2 3
4 5 6
7 8 0

GOAL STATE:
1 2 3
4 0 5
7 8 6

Level 0

0
1 2 3
4 5 6
7 8 0

Level 1

1
1 2 3
4 5 6
7 0 8

2
-1 -1 -1
-1 -1 -1
-1 -1 -1

3
1 2 3
4 5 0
7 8 6

4
-1 -1 -1
-1 -1 -1
-1 -1 -1
```



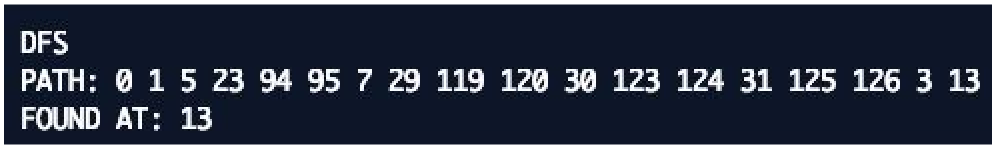
```
Level 2

5
1 2 3
4 5 6
0 7 8

6
-1 -1 -1
-1 -1 -1
-1 -1 -1

7
1 2 3
4 0 6
7 5 8

8
-1 -1 -1
-1 -1 -1
-1 -1 -1


Level 2

9
-1 -1 -1
-1 -1 -1
-1 -1 -1

10
-1 -1 -1
-1 -1 -1
-1 -1 -1

11
-1 -1 -1
-1 -1 -1
-1 -1 -1
```

and so on…..



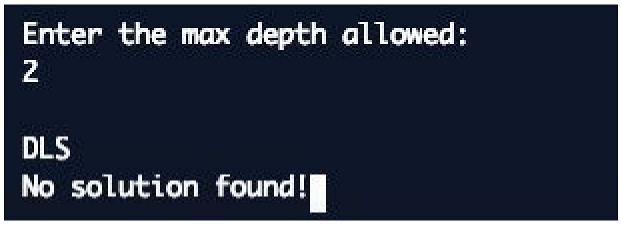
```
330
-1 -1 -1
-1 -1 -1
-1 -1 -1

331
-1 -1 -1
-1 -1 -1
-1 -1 -1

332
-1 -1 -1
-1 -1 -1
-1 -1 -1


Level 4

333
-1 -1 -1
-1 -1 -1
-1 -1 -1

334
-1 -1 -1
-1 -1 -1
-1 -1 -1

335
-1 -1 -1
-1 -1 -1
-1 -1 -1

336
-1 -1 -1
-1 -1 -1
-1 -1 -1
```

DFS Search:

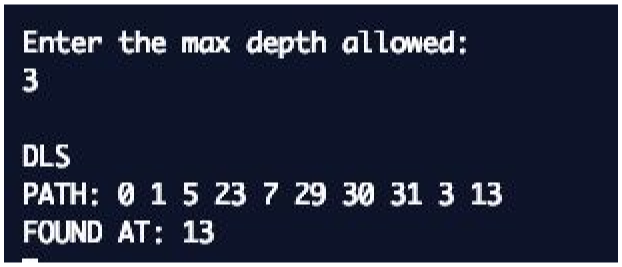


```
DFS
PATH: 0 1 5 23 94 95 7 29 119 120 30 123 124 31 125 126 3 13
FOUND AT: 13
```

DLS Search:

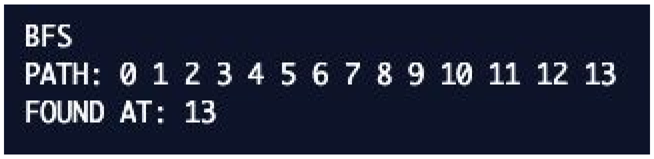


```
Enter the max depth allowed:
2

DLS
No solution found!
```



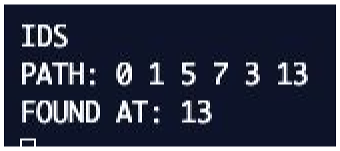
```
Enter the max depth allowed:
3

DLS
PATH: 0 1 5 23 7 29 30 31 3 13
FOUND AT: 13
```

BFS Search:



```
BFS
PATH: 0 1 2 3 4 5 6 7 8 9 10 11 12 13
FOUND AT: 13
```

IDS Search:

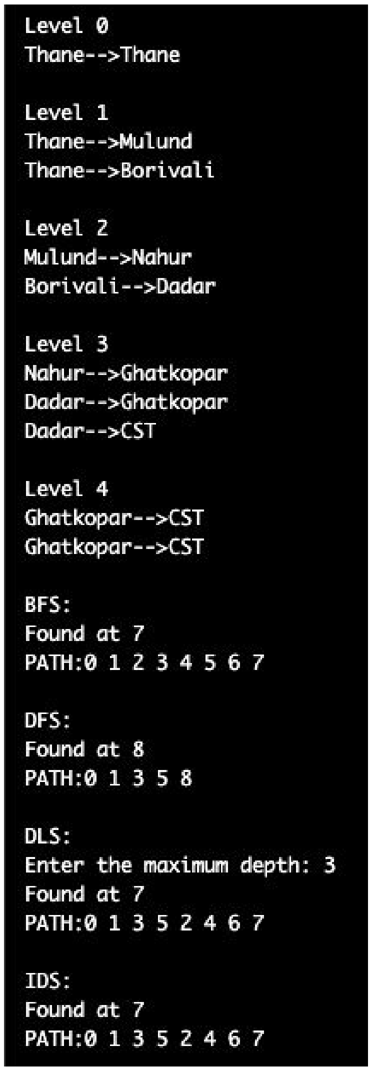


```
IDS
PATH: 0 1 5 7 3 13
FOUND AT: 13
```

b. Map Exploration

```
import java.util.*;

class Main {
 private static ArrayList<String> cities= new ArrayList<>();
 private static ArrayList<Distance> distances= new ArrayList<>();
 private static ArrayList<State> states= new ArrayList<>();
 private static ArrayList<State> tree= new ArrayList<>();
 static int count=0;
 static int max=8;
 static int start = 0;
 static int end = 6;
 public static void main(String[] args) {
   cities.add("Thane");
   cities.add("Mulund");
   cities.add("Borivali");
   cities.add("Nahur");
   cities.add("Dadar");
   cities.add("Ghatkopar");
   cities.add("CST");
   distances.add(new Distance(0,1,10));
   distances.add(new Distance(0,2,30));
   distances.add(new Distance(1,3,10));
   distances.add(new Distance(3,5,20));
   distances.add(new Distance(2,4,20));
   distances.add(new Distance(4,5,20));
   distances.add(new Distance(4,6,10));
   distances.add(new Distance(5,6,20));
   int l=0;
   states.add(new State(start,0,l));
   tree.add(new State(start,0,l));
   int curr;
   while(count<max){
     int flag=0,flag2=0;
     curr = states.get(0).city;
     l = states.get(0).level+1;
     //System.out.println(l);
     for(int i=0;i<distances.size();i++){
       if(distances.get(i).city1==curr){
         for(int j=0;j<tree.size();j++){

if(tree.get(j).city==distances.get(i).city2&&tree.get(j).level<l){
             flag=1;
             break;
           }
         }
         if(flag==0||distances.get(i).city2==end){
          //if(!states.contains(distances.get(i).city2)){
           states.add(new State(distances.get(i).city2,curr,l));
```

```
            tree.add(new State(distances.get(i).city2, curr,l));
           //}
          }
        }
        else if(distances.get(i).city2==curr){
          for(int j=0;j<tree.size();j++){

if(tree.get(j).city==distances.get(i).city1&&tree.get(j).level<l){
              flag2=1;
              break;
            }
          }
          if(flag2==0||distances.get(i).city1==end){
           //if(!states.contains(distances.get(i).city2)){
            states.add(new State(distances.get(i).city1,curr,l));
            tree.add(new State(distances.get(i).city1,curr,l));
          //}
          }
         }
      }
      //System.out.println("\ncount:"+count);

      states.remove(0);
      count++;
    }
    for(int i=0;i<tree.size();i++){
        if(i==0){
          System.out.println("\nLevel 0");
        }
        else if(tree.get(i-1).level!=tree.get(i).level){
          System.out.println("\nLevel "+tree.get(i).level);
        }

System.out.println(cities.get(tree.get(i).parent)+"-->"+cities.get(tree.ge
t(i).city)+" ");
      }

      bfs();
      dfs();
      dls();
      ids();
  }

  static void bfs(){
    StringBuffer sb = new StringBuffer();
    int location=-1;
    int found = 0;
    System.out.println("\nBFS:");
    for(int i=0;i<tree.size();i++){
      if(tree.get(i).city==end){
        sb.append(i);
        found=1;
        location=i;
        break;
      }
      else{
        sb.append(i+" ");
      }
    }
    if(found==1){
      System.out.println("Found at "+location);
      System.out.println("PATH:"+sb);
    }
    else{
      System.out.println("Not found");
    }
  }

  static void dfs(){
    int posi=0;
    ArrayList<State> dfsTree = new ArrayList<>();
    dfsTree.add(tree.get(posi));
    StringBuffer sb = new StringBuffer();
    int location=-1;
    int found = 0;
    System.out.println("\nDFS:");
    while(found==0){
      State curr =  dfsTree.get(0);
      dfsTree.remove(0);
      if(curr.city==end){
        sb.append(tree.indexOf(curr));
        location = tree.indexOf(curr);
        found=1;
        break;
      }
      else{
        sb.append(tree.indexOf(curr)+" ");
        //System.out.println(tree.indexOf(curr)+" ");
        for(int j=tree.size()-1;j>=1;j--){
          if(tree.get(j).parent==curr.city){
            dfsTree.add(0,tree.get(j));
          }
        }
      }
       if(dfsTree.size()==0){
        break;
      }
    }
    if(found==1){
      System.out.println("Found at "+location);
      System.out.println("PATH:"+sb);
    }
    else{
      System.out.println("Not found");
    }
  }

  static void dls(){
    int posi=0;
    ArrayList<State> dfsTree = new ArrayList<>();
    dfsTree.add(tree.get(posi));
    StringBuffer sb = new StringBuffer();
    int location=-1;
    int found = 0;
    System.out.println("\nDLS:");
    System.out.print("Enter the maximum depth: ");
    Scanner sc = new Scanner(System.in);
    int d = sc.nextInt();
    while(found==0){
      State curr =  dfsTree.get(0);
      dfsTree.remove(0);
      if(curr.city==end){
        sb.append(tree.indexOf(curr));
        location = tree.indexOf(curr);
        found=1;
        break;
      }
      else{
        sb.append(tree.indexOf(curr)+" ");
        //System.out.println(tree.indexOf(curr)+" ");
        for(int j=tree.size()-1;j>=1;j--){
          if(tree.get(j).parent==curr.city&&tree.get(j).level<=d){
            dfsTree.add(0,tree.get(j));
          }
        }
      }
       if(dfsTree.size()==0){
        break;
      }
    }
    if(found==1){
      System.out.println("Found at "+location);
      System.out.println("PATH:"+sb);
    }
    else{
      System.out.println("Not found");
    }
  }

  static void ids(){
    int posi=0;
    ArrayList<State> dfsTree = new ArrayList<>();
    dfsTree.add(tree.get(posi));
    StringBuffer sb = new StringBuffer();
    int location=-1;
    int found = 0;
    System.out.println("\nIDS:");
    int d = 1;
    while(found==0){
    d++;
    dfsTree = new ArrayList<>();
    dfsTree.add(tree.get(posi));
    sb = new StringBuffer();
    while(found==0){
      State curr =  dfsTree.get(0);
      dfsTree.remove(0);
      if(curr.city==end){
        sb.append(tree.indexOf(curr));
        location = tree.indexOf(curr);
        found=1;
        break;
      }
      else{
        sb.append(tree.indexOf(curr)+" ");
        //System.out.println(tree.indexOf(curr)+" ");
        for(int j=tree.size()-1;j>=1;j--){
          if(tree.get(j).parent==curr.city&&tree.get(j).level<=d){
            dfsTree.add(0,tree.get(j));
          }
        }
      }
      if(dfsTree.size()==0){
        break;
      }
    }
    }
    if(found==1){
      System.out.println("Found at "+location);
      System.out.println("PATH:"+sb);
    }
    else{
      System.out.println("Not found");
    }
  }

}

class Distance{
 int city1;
 int city2;
 int cost;
 public Distance(int city1, int city2, int cost){
   this.city1 = city1;
   this.city2 = city2;
   this.cost = cost;
 }
}

class State{
 int city;
 int level;
 int parent;
 public State(int city, int parent, int level){
   this.city = city;
   this.level = level;
   this.parent = parent;
 }
}
```



```
Level 0
Thane-->Thane

Level 1
Thane-->Mulund
Thane-->Borivali

Level 2
Mulund-->Nahur
Borivali-->Dadar

Level 3
Nahur-->Ghatkopar
Dadar-->Ghatkopar
Dadar-->CST

Level 4
Ghatkopar-->CST
Ghatkopar-->CST

BFS:
Found at 7
PATH:0 1 2 3 4 5 6 7

DFS:
Found at 8
PATH:0 1 3 5 8

DLS:
Enter the maximum depth: 3
Found at 7
PATH:0 1 3 5 2 4 6 7

IDS:
Found at 7
PATH:0 1 3 5 2 4 6 7
```

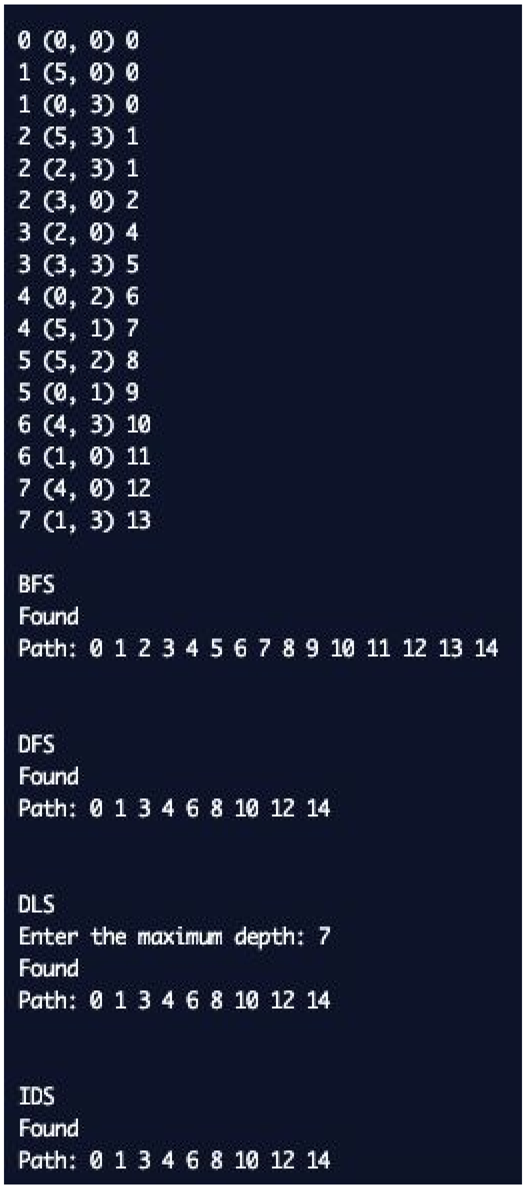
c. Water Jug

```
import java.util.*;

class Main {
 private static ArrayList<State> states = new ArrayList<>();
 private static int max = 100000;
 private static int count = 0;
 public static void main(String[] args) {
   states.add(new State(0,0,0));
   expand();
   for(int i=0;i<states.size();i++){
     if(states.get(i).left!=-1){
       System.out.print("\n"+states.get(i).level+"
("+states.get(i).left+", "+states.get(i).right+")"+ " "+
states.get(i).parent);
     }
   }
   bfs();
   dfs();
   dls();
   ids();
 }

 static void expand(){
   while(count<max&&count<states.size()){
       fill(states.get(count),0);
       fill(states.get(count),1);
       empty(states.get(count),0);
       empty(states.get(count),1);
       transfer(states.get(count),0);
       transfer(states.get(count),1);
       count++;
   }
 }

 static void fill(State s,int x){
   if(s.left!=-1){
   int left=0;
   int right=0;
   if(x==0&&s.left!=5){
     left = 5;
     right = s.right;
   }
   else if(x==1&&s.right!=3){
     left = s.left;
     right = 3;
   }
   int flag=0;
   for(int i=0;i<states.size();i++){
     if((states.get(i).left==left)&&(states.get(i).right==right)){
         flag=1;
         break;
     }
   }
   if(flag==0){
   states.add(new State(left, right,s.level+1, states.indexOf(s)));
   }
 }
}

 static void empty(State s,int x){
   if(s.left!=-1){
  int left=0;
   int right=0;
   if(x==0&&s.left!=0){
     left = 0;
     right = s.right;
   }
   else if(x==1&&s.right!=0){
     left = s.left;
     right = 0;
   }
   int flag=0;
   for(int i=0;i<states.size();i++){
     if((states.get(i).left==left)&&(states.get(i).right==right)){
         flag=1;
         break;
     }
   }
   if(flag==0){
   states.add(new State(left, right,s.level+1,states.indexOf(s)));
   }
 }
}



 static void transfer(State s,int x){
   if(s.left!=-1){
   int left=0;
   int right=0;
   if(x==0&&s.left!=5&&s.right!=0){
     if(s.left + s.right <= 5){
       left = s.left + s.right;
       right = 0;
     }
     else{
       left = 5;
       right = 3-(5-s.left);
       }
     }
     else if(x==1&&s.right!=3&&s.left!=0){
       if(s.left + s.right <= 3){
         left = 0;
         right = s.left + s.right;
       }
       else{
         left = 5-(3-s.right);
         right = 3;
       }
     }
    int flag=0;
     for(int i=0;i<states.size();i++){
       if((states.get(i).left==left)&&(states.get(i).right==right)){
           flag=1;
           break;
       }
     }
     if(flag==0){

     states.add(new State(left, right,s.level+1,states.indexOf(s)));
     }
   }
   }

   static void bfs(){
     StringBuffer sb = new StringBuffer();
     int found = 0;
     System.out.println("\n\nBFS");
     for(int i=0;i<states.size();i++){
       if(states.get(i).left==4&&states.get(i).right==0){
         sb.append(i+" ");
         found=1;
         break;
       }
       else{
         sb.append(i+" ");
       }
     }
     if(found==1){
       System.out.println("Found");
       System.out.println("Path: "+sb);
     }
     else{
       System.out.println("Not Found");
     }
   }

   static void dfs(){
```

```
    StringBuffer sb = new StringBuffer();
    ArrayList<State> visited = new ArrayList<>();
    visited.add(states.get(0));
    int found = 0;
    System.out.println("\n\nDFS");
    while(found==0){
      State curr = visited.get(0);
      visited.remove(0);
      if(curr.left==4&&curr.right==0){
        sb.append(states.indexOf(curr)+" ");
        found=1;
        break;
      }
      else{
        sb.append(states.indexOf(curr)+" ");
        for(int i=states.size()-1;i>0;i--){
          if(states.get(i).parent==states.indexOf(curr)){
            visited.add(0,states.get(i));
          }
        }
      }
    }
    if(found==1){
      System.out.println("Found");
      System.out.println("Path: "+sb);
    }
    else{
      System.out.println("Not Found");
    }
  }

  static void dls(){
    StringBuffer sb = new StringBuffer();
    ArrayList<State> visited = new ArrayList<>();
    visited.add(states.get(0));
    int found = 0;
    System.out.println("\n\nDLS");
    System.out.print("Enter the maximum depth: ");
    Scanner sc = new Scanner(System.in);
    int d = sc.nextInt();
    while(found==0){
      State curr = visited.get(0);
      visited.remove(0);
      if(curr.left==4&&curr.right==0){
        sb.append(states.indexOf(curr)+" ");
        found=1;
        break;
      }
      else{
        sb.append(states.indexOf(curr)+" ");
```

```
          for(int i=states.size()-1;i>0;i--){
  
if(states.get(i).parent==states.indexOf(curr)&&states.get(i).level<=d){
              visited.add(0,states.get(i));
            }
          }
        }
        if(visited.size()==0){
          break;
        }
      }
      if(found==1){
        System.out.println("Found");
        System.out.println("Path: "+sb);
      }
      else{
        System.out.println("Not Found");
      }
    }

    static void ids(){
      StringBuffer sb = new StringBuffer();
      ArrayList<State> visited = new ArrayList<>();
      visited.add(states.get(0));
      int found = 0;
      System.out.println("\n\nIDS");
      int d = 1;
      while(found==0){
      d++;
      visited = new ArrayList<>();
      visited.add(states.get(0));
      sb = new StringBuffer();
      while(found==0){
        State curr = visited.get(0);
        visited.remove(0);
        if(curr.left==4&&curr.right==0){
          sb.append(states.indexOf(curr)+" ");
          found=1;
          break;
        }
        else{
          sb.append(states.indexOf(curr)+" ");
          for(int i=states.size()-1;i>0;i--){
  
if(states.get(i).parent==states.indexOf(curr)&&states.get(i).level<=d){
              visited.add(0,states.get(i));
            }
          }
        }
        if(visited.size()==0){
```

```java
        break;
      }
    }
    }
    if(found==1){
      System.out.println("Found");
      System.out.println("Path: "+sb);
    }
    else{
      System.out.println("Not Found");
    }
 }

}

class State{
 int left;
 int right;
 int level;
 int parent;
 public State(int left, int right){
   this.left = left;
   this.right = right;
 }
 public State(int left, int right, int level){
   this.left = left;
   this.right = right;
   this.level = level;
 }
 public State(int left, int right, int level, int parent){
   this.left = left;
   this.right = right;
   this.level = level;
   this.parent = parent;
 }
}
```



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

BFS
Found
Path: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14


DFS
Found
Path: 0 1 3 4 6 8 10 12 14


DLS
Enter the maximum depth: 7
Found
Path: 0 1 3 4 6 8 10 12 14


IDS
Found
Path: 0 1 3 4 6 8 10 12 14
```

4. Informed Search

a. 8 Puzzle

```
import java.util.*;

class Main {
 public static ArrayList<State> states = new ArrayList<>();
 public static ArrayList<AstarObject> visited = new ArrayList<>();
 public static ArrayList<gbfsObject> visited2 = new ArrayList<>();
 public static ArrayList<State> path = new ArrayList<>();
 public static int counter = 0;
 public static int max=21;
 public static int depth = 0;
  public static State initialState = new State(new
int[]{1,2,3,4,5,6,7,8,0},0);
  public static State finalState = new State(new
int[]{1,2,3,4,0,5,7,8,6});
 public static State end = new State(new
int[]{-1,-1,-1,-1,-1,-1,-1,-1,-1});
 public static void main(String[] args) {
   System.out.print("\nINITIAL STATE:");
   for(int j=0;j<9;j++){
     if(j==0||j==3||j==6){
       System.out.println();
     }
     System.out.print(initialState.pos[j]+" ");
   }
   System.out.print("\n\nGOAL STATE:");
   for(int j=0;j<9;j++){
     if(j==0||j==3||j==6){
       System.out.println();
     }
     System.out.print(finalState.pos[j]+" ");
   }
   states.add(initialState);
   expand();
   int i=0;
   while(i<states.size()){
     System.out.println("\n\nLevel "+states.get(i).level);
     if(i==0){
       State st = states.get(i);
       for(int j=0;j<9;j++){
         if(j==0||j==3||j==6){
           System.out.println();
         }
         if(j==0){
           System.out.println(i + " cost="+st.f);
         }
         System.out.print(st.pos[j]+" ");
```

```
          }
          i++;
        }
        else{
          for(int k=0;k<4;k++){
          State st = states.get(i);
            for(int j=0;j<9;j++){
              if(j==0||j==3||j==6){
                System.out.println();
              }
              if(j==0){
                System.out.println(i + " cost="+st.h);
              }
              System.out.print(st.pos[j]+" ");
            }
          i++;
          System.out.println();
        }
        }
      }
      gbfs();
      astar();
    }

    static void expand(){
      while(counter<max){
        left(states.get(counter));
        right(states.get(counter));
        up(states.get(counter));
        down(states.get(counter));
        counter++;
      }
    }

    static void left(State state){
      State tempState = new State(state.pos, state.level+1);
      int found=-1;
      int f=0;
      for(int i=0;i<9;i++){
        if(tempState.pos[i]==0){
          found = i;
          break;
        }
      }
      if(found%3!=0&&found!=-1){
        int tempPos[] = new int[9];
        for(int j=0;j<9;j++){
          tempPos[j] = tempState.pos[j];
        }
        int t = tempPos[found];
        tempPos[found] = tempPos[found-1];
        tempPos[found-1] = t;
        tempState.pos = tempPos;
        for(int k=0;k<states.size();k++){
          if(Arrays.equals(states.get(k).pos, tempState.pos)){
            f=1;
          }
        }
        if(f==0){
          tempState.h = countMisplaced(tempState);
          tempState.f = tempState.h + tempState.level;
          states.add(tempState);
        }
        else{
          tempPos = new int[]{-1,-1,-1,-1,-1,-1,-1,-1,-1};
          tempState.pos = tempPos;
          tempState.h = 999;
          tempState.f = 999;
          states.add(tempState);
        }
      }
      else{
        int tempPos[] = new int[]{-1,-1,-1,-1,-1,-1,-1,-1,-1};
        tempState.pos = tempPos;
        tempState.h = 999;
        tempState.f = 999;
        states.add(tempState);
      }
    }

    static void right(State state){
      State tempState = new State(state.pos, state.level+1);
      int found=-1;
      int f=0;
      for(int i=0;i<9;i++){
        if(tempState.pos[i]==0){
          found = i;
          break;
        }
      }
      if(found%3!=2&&found!=-1){
        int tempPos[] = new int[9];
        for(int j=0;j<9;j++){
          tempPos[j] = tempState.pos[j];
        }
        int t = tempPos[found];
        tempPos[found] = tempPos[found+1];
        tempPos[found+1] = t;
        tempState.pos = tempPos;
        for(int k=0;k<states.size();k++){
```

```java
          if(Arrays.equals(states.get(k).pos, tempState.pos)){
            f=1;
          }
        }
        if(f==0){
          tempState.h = countMisplaced(tempState);
          tempState.f = tempState.h + tempState.level;
          states.add(tempState);
        }
        else{
          tempPos = new int[]{-1,-1,-1,-1,-1,-1,-1,-1,-1};
          tempState.pos = tempPos;
          tempState.h = 999;
          tempState.f = 999;
          states.add(tempState);
        }
      }
      else{
        int tempPos[] = new int[]{-1,-1,-1,-1,-1,-1,-1,-1,-1};
        tempState.pos = tempPos;
        tempState.h = 999;
        tempState.f = 999;
        states.add(tempState);
      }
    }

    static void up(State state){
      State tempState = new State(state.pos, state.level+1);
      int found=-1;
      int f=0;
      for(int i=0;i<9;i++){
        if(tempState.pos[i]==0){
          found = i;
          break;
        }
      }
      if(found>2&&found!=-1){
        int tempPos[] = new int[9];
        for(int j=0;j<9;j++){
          tempPos[j] = tempState.pos[j];
        }
        int t = tempPos[found];
        tempPos[found] = tempPos[found-3];
        tempPos[found-3] = t;
        tempState.pos = tempPos;
        for(int k=0;k<states.size();k++){
          if(Arrays.equals(states.get(k).pos, tempState.pos)){
            f=1;
          }
        }
        if(f==0){
          tempState.h = countMisplaced(tempState);
          tempState.f = tempState.h + tempState.level;
          states.add(tempState);
        }
        else{
          tempPos = new int[]{-1,-1,-1,-1,-1,-1,-1,-1,-1};
          tempState.pos = tempPos;
          tempState.h = 999;
          tempState.f = 999;
          states.add(tempState);
        }
      }
      else{
        int tempPos[] = new int[]{-1,-1,-1,-1,-1,-1,-1,-1,-1};
        tempState.pos = tempPos;
        tempState.h = 999;
        tempState.f = 999;
        states.add(tempState);
      }
    }

     static void down(State state){
      State tempState = new State(state.pos, state.level+1);
      int found=-1;
      int f=0;
      for(int i=0;i<9;i++){
        if(tempState.pos[i]==0){
          found = i;
          break;
        }
      }
      if(found<6&&found!=-1){
        int tempPos[] = new int[9];
        for(int j=0;j<9;j++){
          tempPos[j] = tempState.pos[j];
        }
        int t = tempPos[found];
        tempPos[found] = tempPos[found+3];
        tempPos[found+3] = t;
        tempState.pos = tempPos;
        for(int k=0;k<states.size();k++){
          if(Arrays.equals(states.get(k).pos, tempState.pos)){
            f=1;
          }
        }
        if(f==0){
          tempState.h = countMisplaced(tempState);
          tempState.f = tempState.h + tempState.level;
          states.add(tempState);
        }
        else{
          tempPos = new int[]{-1,-1,-1,-1,-1,-1,-1,-1,-1};
          tempState.pos = tempPos;
          tempState.h = 999;
          tempState.f = 999;
          states.add(tempState);
        }
      }
      else{
        int tempPos[] = new int[]{-1,-1,-1,-1,-1,-1,-1,-1,-1};
        tempState.pos = tempPos;
        tempState.h = 999;
        tempState.f = 999;
        states.add(tempState);
      }
    }

    static void gbfs(){     //check if it is the goal state when we expand it
      int posi=0;
      int found=0;
      visited2.add(new gbfsObject(0,states.get(0).h));
      int top=0;
      StringBuffer sb = new StringBuffer();
      System.out.println("\nGBFS");
      //System.out.print("PATH: ");
      while(found==0){
        top = visited2.get(0).index;
        sb.append(top+" ");
        if(!(Arrays.equals(states.get(top).pos,end.pos))&&top<max){
          if(Arrays.equals(states.get(top).pos, finalState.pos)){
            found=1;
            break;
          }
          visited2.remove(0);
          posi = top*4+1;
          for(int i=3;i>=0;i--){
            if(!(Arrays.equals(states.get(posi+i).pos,end.pos))){
              visited2.add(new gbfsObject(posi+i,states.get(posi+i).h));
            }
          }
          visited2.sort(Comparator.comparingInt(gbfsObject :: getCost));
        }
        else{
          visited2.remove(0);
        }
        if(visited2.size()==0){
          break;
        }
      }
    if(found==1){
      System.out.print("PATH: "+sb);
      System.out.println("\nFOUND AT: "+top);
    }
    else{
      System.out.println("No solution found!");
    }
  }

  static void astar(){     //check if it is the goal state when we expand it
    int posi=0;
    int found=0;
    visited.add(new AstarObject(0,states.get(0).f));
    int top=0;
    StringBuffer sb = new StringBuffer();
    System.out.println("\nA*");
    //System.out.print("PATH: ");
    while(found==0){
      top = visited.get(0).index;
      sb.append(top+" ");
      if(!(Arrays.equals(states.get(top).pos,end.pos))&&top<max){
        if(Arrays.equals(states.get(top).pos, finalState.pos)){
          found=1;
          break;
        }
        visited.remove(0);
        posi = top*4+1;
        for(int i=3;i>=0;i--){
          if(!(Arrays.equals(states.get(posi+i).pos,end.pos))){
            visited.add(new AstarObject(posi+i,states.get(posi+i).f));
          }
        }
        visited.sort(Comparator.comparingInt(AstarObject :: getCost));
      }
      else{
        visited.remove(0);
      }
      if(visited.size()==0){
        break;
      }
    }
    if(found==1){
      System.out.print("PATH: "+sb);
      System.out.println("\nFOUND AT: "+top);
    }
    else{
      System.out.println("No solution found!");
    }
  }
```

```
 static int countMisplaced(State st){
   int count = 0;
   for(int i=0;i<9;i++){
     if(st.pos[i]!=finalState.pos[i]){
       count++;
     }
   }
   return count;
 }
}

class State{
 int pos[] = new int[9];
 int level;
 int h;
 int f;
 public State(int pos[]){
   this.pos = pos;
 }
 public State(int pos[], int level){
   this.pos = pos;
   this.level = level;
 }
}

class AstarObject{
 int index;
 int cost;
 public AstarObject(int index, int cost){
   this.index = index;
   this.cost = cost;
 }
 int getCost(){
   return this.cost;
 }
}

class gbfsObject{
 int index;
 int cost;
 public gbfsObject(int index, int cost){
   this.index = index;
   this.cost = cost;
 }
 int getCost(){
   return this.cost;
 }
}
```

b. Map Exploration

```
import java.util.*;

class Main {
 private static ArrayList<City> cities= new ArrayList<>();
 private static ArrayList<Distance> distances= new ArrayList<>();
 private static ArrayList<State> states= new ArrayList<>();
 private static ArrayList<State> tree= new ArrayList<>();
 static int count=0;
 static int max=10;
 static int start = 0;
 static int end = 7;
 public static void main(String[] args) {
   cities.add(new City("Thane",50));
   cities.add(new City("Mulund",45));
   cities.add(new City("Borivali",30));
   cities.add(new City("Nahur",45));
   cities.add(new City("Dadar",20));
   cities.add(new City("Ghatkopar",30));
   cities.add(new City("Byculla",5));
   cities.add(new City("CST",0));
   distances.add(new Distance(0,1,10));
   distances.add(new Distance(0,2,30));
   distances.add(new Distance(1,3,10));
   distances.add(new Distance(3,5,20));
   distances.add(new Distance(2,4,20));
   distances.add(new Distance(4,5,20));
   distances.add(new Distance(4,7,10));
   //distances.add(new Distance(5,7,15));
   distances.add(new Distance(2,6,20));
   distances.add(new Distance(6,7,5));
   int l=0;
   states.add(new State(start,0,l,0));
   tree.add(new State(start,0,l,0));
```

```
    State curr;
    while(count<max){
      int flag=0,flag2=0;
      curr = states.get(0);
      l = states.get(0).level+1;
      //System.out.println(l);
      for(int i=0;i<distances.size();i++){
        if(distances.get(i).city1==curr.city){
          for(int j=0;j<tree.size();j++){

if(tree.get(j).city==distances.get(i).city2&&tree.get(j).level<l){
              flag=1;
              break;
            }
          }
          if(flag==0||distances.get(i).city2==end){
           //if(!states.contains(distances.get(i).city2)){
            states.add(new
State(distances.get(i).city2,curr.city,l,distances.get(i).cost+curr.g));
            System.out.println(i+ " " +(distances.get(i).cost+curr.g));
            tree.add(new State(distances.get(i).city2,
curr.city,l,distances.get(i).cost+curr.g));
           //}
          }
        }
        else if(distances.get(i).city2==curr.city){
          for(int j=0;j<tree.size();j++){

if(tree.get(j).city==distances.get(i).city1&&tree.get(j).level<l){
              flag2=1;
              break;
            }
          }
          if(flag2==0||distances.get(i).city1==end){
           //if(!states.contains(distances.get(i).city2)){
            states.add(new
 State(distances.get(i).city1,curr.city,l,distances.get(i).cost+curr.g));
            tree.add(new
 State(distances.get(i).city1,curr.city,l,distances.get(i).cost+curr.g));
          //}
          }
         }
      }
      //System.out.println("\ncount:"+count);

      states.remove(0);
      count++;
    }
    for(int i=0;i<tree.size();i++){
        if(i==0){
          System.out.println("\nLevel 0");
        }
        else if(tree.get(i-1).level!=tree.get(i).level){
          System.out.println("\nLevel "+tree.get(i).level);
        }
        System.out.println(i+"
 "+cities.get(tree.get(i).parent).name+"-->"+cities.get(tree.get(i).city).n
 ame+" "+tree.get(i).g);
      }
      gbfs();
      astar();
  }

  static void gbfs(){
    StringBuffer sb = new StringBuffer();
    int location=-1;
    int found = 0;
    ArrayList<State> temp = new ArrayList<>();
```

```
ArrayList<State> visited = new ArrayList<>();
visited.add(tree.get(0));
System.out.println("\nGBFS:");
for(int i=0;i<tree.size();i++){
  temp = new ArrayList<>();
  if(visited.size()==0){
      continue;
  }
  State curr = visited.get(0);
  if(i==0){
    i++;
  }
  if(curr.city==end){

    sb.append(i);
    found=1;
    location=i;
    break;
  }
  else{
      while(i!=0&&(tree.get(i).parent==curr.city)){
        State s = tree.get(i);
        if(s.city==end){
          sb.append(i);
          found=1;
          location=i;
          break;
        }
        s.h = cities.get(s.city).h;
        temp.add(s);
        i++;
      }
      if(found==1){
        break;
        }
        if(temp.size()==0){
          continue;
        }

        sb.append(curr.city+" ");
        visited.remove(0);
      temp.sort(Comparator.comparingInt(State :: getHeuristic));
      visited.add(temp.get(0));
    }
    if(visited.size()==0){
      break;
    }
  }
  if(found==1){
    System.out.println("Found at "+location);
    System.out.println("PATH:"+sb);
  }
  else{
    System.out.println("Not found");
  }
}

static void astar(){
  StringBuffer sb = new StringBuffer();
  int location=-1;
  int found = 0;
  ArrayList<State> temp = new ArrayList<>();
  ArrayList<State> visited = new ArrayList<>();
  visited.add(tree.get(0));
  System.out.println("\nA*:");
  for(int i=0;i<tree.size();i++){
    temp = new ArrayList<>();
    if(visited.size()==0){
            continue;
        }
        State curr = visited.get(0);
        System.out.println("Current: "+cities.get(curr.city).name);
        visited.remove(0);
        if(i==0){
          i++;
        }
        if(curr.city==end){
          sb.append(i);
          found=1;
          location=i;
          break;
        }
        else{
            while(i!=0&&(tree.get(i).parent==curr.city)){
              System.out.println("i is "+i);
              State s = tree.get(i);
              if(s.city==end){
                sb.append(i);
                found=1;
                location=i;
                break;
              }
              s.h = cities.get(s.city).h;
              s.f = s.g + s.h;
              System.out.println("City: "+cities.get(s.city).name+"Cost: 
"+s.f);
              visited.add(s);
              visited.sort(Comparator.comparingInt(State :: getCost));
              i++;
            }
            i--;
            if(found==1){
```

```
            break;
          }
          if(visited.size()==0){
            continue;
          }
        sb.append(curr.city+" ");
        //System.out.println("City:
"+cities.get(visited.get(0).city).name+"Cost: "+visited.get(0).f);
      }
      if(visited.size()==0){
        break;
      }
    }
    if(found==1){
      System.out.println("Found at "+location);
      System.out.println("PATH:"+sb);
    }
    else{
      System.out.println("Not found");
    }
  }
}


class Distance{
 int city1;
 int city2;
 int cost;
 public Distance(int city1, int city2, int cost){
   this.city1 = city1;
   this.city2 = city2;
   this.cost = cost;
 }
}
```

```
class City{
 String name;
 int h;
 public City(String name, int h){
   this.name = name;
   this.h = h;
 }
}

class State{
 int city;
 int level;
 int parent;
 int f;
 int g;
 int h;
 public State(int city, int parent, int level){
   this.city = city;
   this.level = level;
   this.parent = parent;
 }
  public State(int city, int parent, int level, int g){
   this.city = city;
   this.level = level;
   this.parent = parent;
   this.g = g;
 }
 int getHeuristic(){
   return this.h;
 }
  int getCost(){
   return this.f;
 }
}
```

c. Blocks World

```
import java.util.*;

class Main {
 private static ArrayList<State> states = new ArrayList<>();
 private static State finalState;
 public static void main(String[] args) {
   ArrayList<Integer> temp1 = new ArrayList<>();
   temp1.add(-1);
   ArrayList<Integer> temp2 = new ArrayList<>();
   temp2.add(-1);
   temp2.add(2);
   temp2.add(1);
   temp2.add(3);
   ArrayList<Integer> temp3 = new ArrayList<>();
   temp3.add(-1);
   finalState = new State(temp1, temp2, temp3, 0, 0, 0);
   temp1 = new ArrayList<>();
   temp1.add(-1);
   temp2 = new ArrayList<>();
   temp2.add(-1);
   temp2.add(1);
   temp2.add(2);
   temp3 = new ArrayList<>();
   temp3.add(-1);
   temp3.add(3);
   states.add(new State(temp1, temp2, temp3, 0, 0, -1));
   getHeuristic(states.get(0));
   for(int i=0;i<1000;i++){
     if(states.size()>i){
       // System.out.println(i);
       // System.out.println(states.get(i).one+" "+states.get(i).two+ " "
+ states.get(i).three);
       // System.out.println(states.get(i).g+" "+states.get(i).h);
```

```
        // System.out.println();
        shift(states.get(i));
      }
    }

    System.out.println("DONE");
    // for(int i=0;i<states.size();i++){
    //   System.out.println(states.get(i).g+" "+states.get(i).h);
    // }

  }

  static void shift(State s){
    ArrayList<Integer> temp1;
    ArrayList<Integer> temp2;
    ArrayList<Integer> temp3;
    if(s.one.get(s.one.size()-1)!=-1){
      temp1 = new ArrayList<>();
      for(int j=0;j<s.one.size();j++){
        temp1.add(s.one.get(j));
      }
      temp2 = new ArrayList<>();
      for(int j=0;j<s.two.size();j++){
        temp2.add(s.two.get(j));
      }
      temp3 = new ArrayList<>();
      for(int j=0;j<s.three.size();j++){
        temp3.add(s.three.get(j));
      }
      temp2.add(s.one.get(s.one.size()-1));
      temp1.remove(s.one.get(s.one.size()-1));
      int flag=0;
      for(int i=0;i<states.size();i++){
```

```
if((states.get(i).one.equals(temp1))&&(states.get(i).two.equals(temp2))&&(
states.get(i).three.equals(temp3))){
          flag=1;
        }
      }
      if(flag==0){
        states.add(new State(temp1, temp2, temp3, (s.g+1),0,
states.indexOf(s)));
        getHeuristic(states.get(states.size()-1));
        check(states.get(states.size()-1));
      }

      temp1 = new ArrayList<>();
      for(int j=0;j<s.one.size();j++){
        temp1.add(s.one.get(j));
      }
      temp2 = new ArrayList<>();
      for(int j=0;j<s.two.size();j++){
        temp2.add(s.two.get(j));
      }
      temp3 = new ArrayList<>();
      for(int j=0;j<s.three.size();j++){
        temp3.add(s.three.get(j));
      }
      temp3.add(s.one.get(s.one.size()-1));
      temp1.remove(s.one.get(s.one.size()-1));

      flag=0;
      for(int i=0;i<states.size();i++){

if((states.get(i).one.equals(temp1))&&(states.get(i).two.equals(temp2))&&(
states.get(i).three.equals(temp3))){
          flag=1;
        }
      }
      if(flag==0){
        states.add(new State(temp1, temp2, temp3,
(s.g+1),0,states.indexOf(s)));
        getHeuristic(states.get(states.size()-1));
        check(states.get(states.size()-1));
      }

    }
    if(s.two.get(s.two.size()-1)!=-1){
      temp1 = new ArrayList<>();
      for(int j=0;j<s.one.size();j++){
        temp1.add(s.one.get(j));
      }
      temp2 = new ArrayList<>();
      for(int j=0;j<s.two.size();j++){
        temp2.add(s.two.get(j));
      }
      temp3 = new ArrayList<>();
      for(int j=0;j<s.three.size();j++){
        temp3.add(s.three.get(j));
      }
      temp1.add(s.two.get(s.two.size()-1));
      temp2.remove(s.two.get(s.two.size()-1));
      int flag=0;
      for(int i=0;i<states.size();i++){

if((states.get(i).one.equals(temp1))&&(states.get(i).two.equals(temp2))&&(
states.get(i).three.equals(temp3))){
          flag=1;
        }
      }
      if(flag==0){
        states.add(new State(temp1, temp2, temp3, (s.g+1),0,
states.indexOf(s)));
        getHeuristic(states.get(states.size()-1));
        check(states.get(states.size()-1));
      }

      temp1 = new ArrayList<>();
      for(int j=0;j<s.one.size();j++){
        temp1.add(s.one.get(j));
      }
      temp2 = new ArrayList<>();
      for(int j=0;j<s.two.size();j++){
        temp2.add(s.two.get(j));
      }
      temp3 = new ArrayList<>();
      for(int j=0;j<s.three.size();j++){
        temp3.add(s.three.get(j));
      }
      temp3.add(s.two.get(s.two.size()-1));
      temp2.remove(s.two.get(s.two.size()-1));

      flag=0;
      for(int i=0;i<states.size();i++){

if((states.get(i).one.equals(temp1))&&(states.get(i).two.equals(temp2))&&(
states.get(i).three.equals(temp3))){
          flag=1;
        }
      }
      if(flag==0){
        states.add(new State(temp1, temp2, temp3, (s.g+1),0,
states.indexOf(s)));
        getHeuristic(states.get(states.size()-1));
        check(states.get(states.size()-1));
      }
    }
    if(s.three.get(s.three.size()-1)!=-1){
      temp1 = new ArrayList<>();
      for(int j=0;j<s.one.size();j++){
        temp1.add(s.one.get(j));
      }
      temp2 = new ArrayList<>();
      for(int j=0;j<s.two.size();j++){
        temp2.add(s.two.get(j));
      }
      temp3 = new ArrayList<>();
      for(int j=0;j<s.three.size();j++){
        temp3.add(s.three.get(j));
      }
      temp1.add(s.three.get(s.three.size()-1));
      temp3.remove(s.three.get(s.three.size()-1));

      int flag=0;
      for(int i=0;i<states.size();i++){

if((states.get(i).one.equals(temp1))&&(states.get(i).two.equals(temp2))&&(
states.get(i).three.equals(temp3))){
          flag=1;
        }
      }
      if(flag==0){
        states.add(new State(temp1, temp2, temp3, (s.g+1),0,
states.indexOf(s)));
        getHeuristic(states.get(states.size()-1));
        check(states.get(states.size()-1));
      }

     temp1 = new ArrayList<>();
```

```
      for(int j=0;j<s.one.size();j++){
        temp1.add(s.one.get(j));
      }
      temp2 = new ArrayList<>();
      for(int j=0;j<s.two.size();j++){
        temp2.add(s.two.get(j));
      }
      temp3 = new ArrayList<>();
      for(int j=0;j<s.three.size();j++){
        temp3.add(s.three.get(j));
      }
      temp2.add(s.three.get(s.three.size()-1));
      temp3.remove(s.three.get(s.three.size()-1));
      flag=0;
      for(int i=0;i<states.size();i++){

if((states.get(i).one.equals(temp1))&&(states.get(i).two.equals(temp2))&&(
states.get(i).three.equals(temp3))){
          flag=1;
        }
      }
      if(flag==0){
        states.add(new State(temp1, temp2, temp3,
(s.g+1),0,states.indexOf(s)));
        getHeuristic(states.get(states.size()-1));
        check(states.get(states.size()-1));
      }
    }


  }

  static void check(State s){
```

```
if(s.one.equals(finalState.one)&&s.two.equals(finalState.two)&&s.three.equ
als(finalState.three)){
     System.out.println("FOUND");
      System.out.println(s.one+" "+s.two+ " " + s.three);
       System.out.println(s.g+" "+s.h);
       System.out.println();
     System.exit(0);
   }
 }

 static void getHeuristic(State s){
   int h=0;
   for(int i=1;i<s.one.size();i++){
     if(i<finalState.one.size()){
       if(s.one.get(i-1)!=finalState.one.get(i-1)){
         h--;
         //System.out.println("Decrement 1");
       }
       else{
         h++;
         //System.out.println("Increment 1");
       }
     }
     else{
       h--;
       //System.out.println("Decrement 1");
     }
   }
   for(int i=1;i<s.two.size();i++){
     if(i<finalState.two.size()){
       if(s.two.get(i-1)!=finalState.two.get(i-1)){
         h--;
         //System.out.println("Decrement 2");
```

```
          }
          else{
            h++;
             //System.out.println("Increment 2");
          }
        }
        else{
          h--;
          //System.out.println("Decrement 2");
        }
      }
      for(int i=1;i<s.three.size();i++){
        if(i<finalState.three.size()){
          if(s.three.get(i-1)!=finalState.three.get(i-1)){
            h--;
            //System.out.println("Decrement 3");
          }
          else{
            h++;
             //System.out.println("Increment 3");
          }
        }
        else{
          h--;
          //System.out.println("Decrement 3");
        }
      }
      s.h = h;
      s.f = s.f + s.g;
    }
  }

class State{
 ArrayList<Integer> one;
```

```java
 ArrayList<Integer> two;
 ArrayList<Integer> three;
 int g;
 int h;
 int f;
 int parent;
 public State(ArrayList<Integer> one, ArrayList<Integer> two,
ArrayList<Integer> three, int g, int h, int parent){
   this.one = one;
   this.two = two;
   this.three = three;
   this.g = g;
   this.h = h;
   this.parent = parent;
 }
}
```

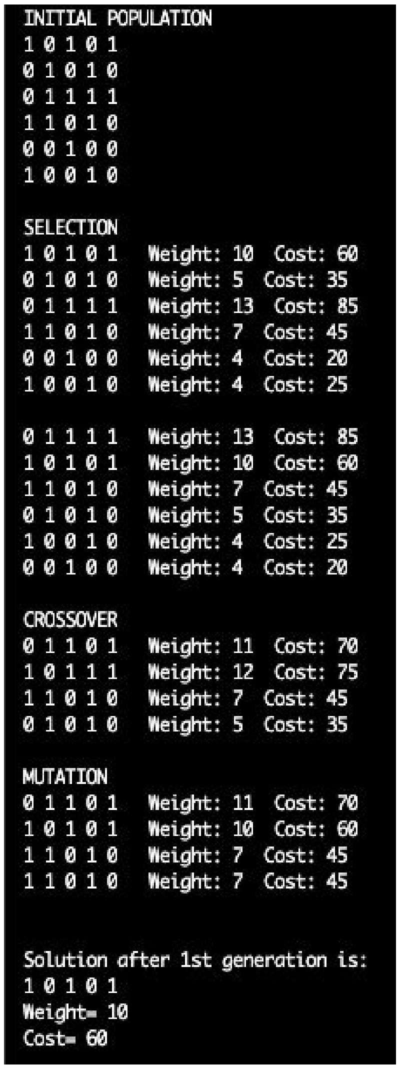
5. Genetic Algorithm

a. 0/1 Knapsack

```java
import java.util.*;

class Main {
 private static int max = 10;
 private static ArrayList<Item> items = new ArrayList<>();
 private static ArrayList<Entry> list = new ArrayList<>();
 public static void main(String[] args) {
   problem();
   initialize();
   selection();
   crossover();
   mutation();
   check();
 }

 static void print(){
   for(int i=0;i<list.size();i++){
     for(int j=0;j<5;j++){
       System.out.print(list.get(i).bin[j]+" ");
     }
     System.out.println("  Weight: "+list.get(i).weight+"  Cost: 
"+list.get(i).cost);
   }
   System.out.println();
 }

 static void problem(){
   items.add(new Item(2,10));
   items.add(new Item(3,20));
   items.add(new Item(4,20));
   items.add(new Item(2,15));
   items.add(new Item(4,30));
 }

 static void initialize(){
   System.out.println("INITIAL POPULATION");
   Random r = new Random();
   for(int i=0;i<6;i++){
     int temp[] = new int[5];
     int cost=0;
     int weight=0;
     for(int j=0;j<5;j++){
       temp[j] = r.nextInt(2);
```

```
        if(temp[j] == 1){
          cost = cost + items.get(j).cost;
          weight = weight + items.get(j).weight;
        }
      }
      list.add(new Entry(temp,cost,weight));
    }
    for(int i=0;i<list.size();i++){
      for(int j=0;j<5;j++){
        System.out.print(list.get(i).bin[j]+" ");
      }
      System.out.println();
    }
    System.out.println();
  }

  static void selection(){
    System.out.println("SELECTION");
    print();
    for(int i=0;i<list.size();i++){
      if(list.get(i).cost<=15){
        list.remove(i);
        i--;
      }
    }
    list.sort(Comparator.comparingInt(Entry :: getCost).reversed());
    print();
  }

  static void crossover(){
    System.out.println("CROSSOVER");
    if(list.size()>=4){
      for(int i=4;i<list.size();i++){
        list.remove(i);
        i--;
      }
      int crossoverPoint = 5/2 + 1;
      for(int i=0;i<list.size();i=i+2){
        Entry upper = list.get(i);
        Entry lower = list.get(i+1);

        for(int j=crossoverPoint;j<5;j++){
          int temp = upper.bin[j];
          upper.bin[j] = lower.bin[j];
          lower.bin[j] = temp;
        }
      }
      cost();
      print();
    }
```

```
 }

 static void mutation(){
   System.out.println("MUTATION");
   Random r = new Random();
   for(int i=0;i<list.size();i++){
     list.get(i).bin[r.nextInt(5)] = r.nextInt(2);
   }
   cost();
   print();
 }

 static void cost(){
   for(int i=0;i<list.size();i++){
     list.get(i).cost = 0;
     list.get(i).weight = 0;
     for(int j=0;j<5;j++){
       if(list.get(i).bin[j]==1){
         list.get(i).cost = list.get(i).cost + items.get(j).cost;
         list.get(i).weight = list.get(i).weight + items.get(j).weight;
       }
     }
   }
 }

 static void check(){
   int found=-1;
   list.sort(Comparator.comparingInt(Entry :: getCost).reversed());
   for(int i=0;i<list.size();i++){
     if(list.get(i).weight<=max){
       found = i;
       break;
     }
   }
   if(found!=-1){
     System.out.println("\nSolution after 1st generation is: ");
     for(int j=0;j<5;j++){
       System.out.print(list.get(found).bin[j]+" ");
     }
     System.out.println("\nWeight= "+list.get(found).weight);
     System.out.println("Cost= "+list.get(found).cost);
   }
   else{
     System.out.println("\nNo solution found after 1st generation");
   }

 }
}

class Item{
 int weight;
 int cost;
 public Item(int weight, int cost){
   this.weight = weight;
   this.cost = cost;
 }
}

class Entry{
 int bin[];
 int weight;
 int cost;
 public Entry(int bin[], int cost, int weight){
   this.bin = bin;
   this.cost = cost;
   this.weight = weight;
 }
 int getCost(){
   return this.cost;
 }
}
```

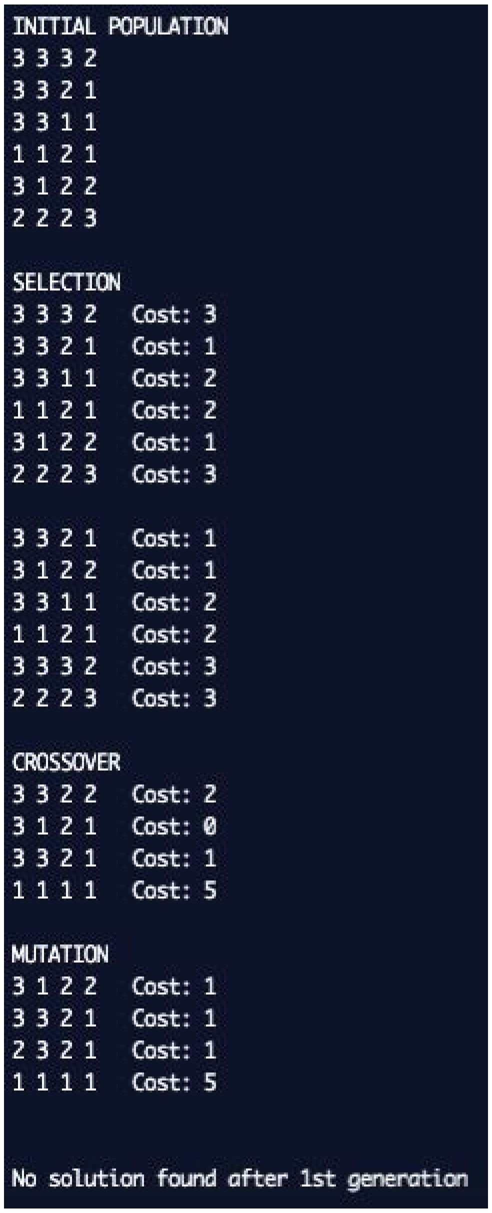


```
INITIAL POPULATION
1 0 1 0 1
0 1 0 1 0
0 1 1 1 1
1 1 0 1 0
0 0 1 0 0
1 0 0 1 0

SELECTION
1 0 1 0 1   Weight: 10  Cost: 60
0 1 0 1 0   Weight: 5  Cost: 35
0 1 1 1 1   Weight: 13  Cost: 85
1 1 0 1 0   Weight: 7  Cost: 45
0 0 1 0 0   Weight: 4  Cost: 20
1 0 0 1 0   Weight: 4  Cost: 25

0 1 1 1 1   Weight: 13  Cost: 85
1 0 1 0 1   Weight: 10  Cost: 60
1 1 0 1 0   Weight: 7  Cost: 45
0 1 0 1 0   Weight: 5  Cost: 35
1 0 0 1 0   Weight: 4  Cost: 25
0 0 1 0 0   Weight: 4  Cost: 20

CROSSOVER
0 1 1 0 1   Weight: 11  Cost: 70
1 0 1 1 1   Weight: 12  Cost: 75
1 1 0 1 0   Weight: 7  Cost: 45
0 1 0 1 0   Weight: 5  Cost: 35

MUTATION
0 1 1 0 1   Weight: 11  Cost: 70
1 0 1 0 1   Weight: 10  Cost: 60
1 1 0 1 0   Weight: 7  Cost: 45
1 1 0 1 0   Weight: 7  Cost: 45


Solution after 1st generation is:
1 0 1 0 1
Weight= 10
Cost= 60
```

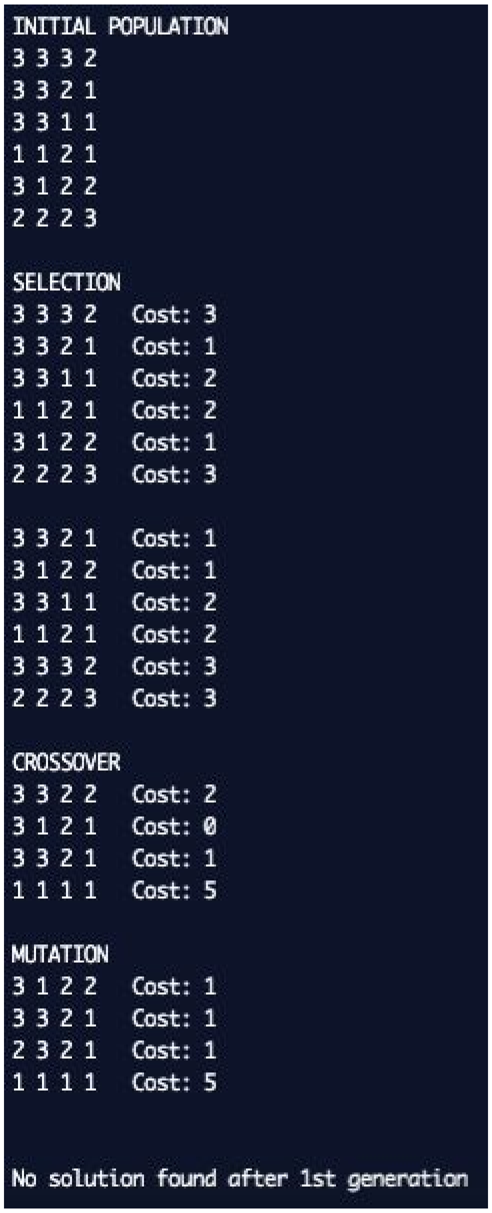
b. Graph Colouring

```
import java.util.*;

class Main {
 private static int graph[][] = new int[4][4];
 private static ArrayList<Entry> list = new ArrayList<>();
 public static void main(String[] args) {
   problem();
   initialize();
   selection();
   crossover();
   mutation();
   check();
 }

 static void problem(){
   graph[0][0] = 0;
   graph[0][1] = 1;
```

```
    graph[0][2] = 1;
    graph[0][3] = 1;
    graph[1][0] = 1;
    graph[1][1] = 0;
    graph[1][2] = 1;
    graph[1][3] = 0;
    graph[2][0] = 1;
    graph[2][1] = 1;
    graph[2][2] = 0;
    graph[2][3] = 1;
    graph[3][0] = 1;
    graph[3][1] = 0;
    graph[3][2] = 1;
    graph[3][3] = 0;
  }

  static void print(){
    for(int i=0;i<list.size();i++){
      for(int j=0;j<4;j++){
        System.out.print(list.get(i).bin[j]+" ");
      }
      System.out.println("  Cost: "+list.get(i).cost);
    }
    System.out.println();
  }

  static void initialize(){
    System.out.println("INITIAL POPULATION");
    Random r = new Random();
    for(int i=0;i<6;i++){
      int temp[] = new int[4];
      for(int j=0;j<4;j++){
        temp[j] = r.nextInt(3)+1;
      }
      list.add(new Entry(temp));
    }
    cost();
    for(int i=0;i<list.size();i++){
      for(int j=0;j<4;j++){
        System.out.print(list.get(i).bin[j]+" ");
      }
      System.out.println();
    }
    System.out.println();
  }

  static void selection(){
    System.out.println("SELECTION");
    print();
    for(int i=0;i<list.size();i++){
      if(list.get(i).cost>=4){
        list.remove(i);
        i--;
      }
    }
    list.sort(Comparator.comparingInt(Entry :: getCost));
    print();
  }

  static void crossover(){
    System.out.println("CROSSOVER");
    if(list.size()>=4){
      for(int i=4;i<list.size();i++){
        list.remove(i);
        i--;
      }
      int crossoverPoint = 4/2;
      for(int i=0;i<list.size();i=i+2){
        Entry upper = list.get(i);
        Entry lower = list.get(i+1);

        for(int j=crossoverPoint;j<4;j++){
          int temp = upper.bin[j];
          upper.bin[j] = lower.bin[j];
          lower.bin[j] = temp;
        }
      }
      cost();
      print();
    }
  }

  static void mutation(){
    System.out.println("MUTATION");
    Random r = new Random();
    for(int i=0;i<list.size();i++){
      list.get(i).bin[r.nextInt(4)] = r.nextInt(3)+1;
    }
    cost();
    print();
  }

  static void cost(){
    for(int i=0;i<list.size();i++){
      list.get(i).cost = 0;
      for(int j=0;j<4;j++){
        for(int k=j+1;k<4;k++){
          if(graph[j][k]==1&&(list.get(i).bin[j]==list.get(i).bin[k])){
            list.get(i).cost++;
        }
        }
      }
      }
  }

  static void check(){
    int found=-1;
    for(int i=0;i<list.size();i++){
      if(list.get(i).cost==0){
        found = i;
        break;
      }
    }
    if(found!=-1){
      System.out.println("\nSolution after 1st generation is: ");
      for(int j=0;j<4;j++){
        System.out.print(list.get(found).bin[j]+" ");
      }
    }
    else{
      System.out.println("\nNo solution found after 1st generation");
    }

  }
}

class Entry{
  int bin[];
  int cost;
  public Entry(int bin[]){
    this.bin = bin;
  }
  public Entry(int bin[], int cost){
    this.bin = bin;
    this.cost = cost;
  }
  int getCost(){
    return this.cost;
  }
}
```



```
INITIAL POPULATION
3 3 3 2
3 3 2 1
3 3 1 1
1 1 2 1
3 1 2 2
2 2 2 3

SELECTION
3 3 3 2   Cost: 3
3 3 2 1   Cost: 1
3 3 1 1   Cost: 2
1 1 2 1   Cost: 2
3 1 2 2   Cost: 1
2 2 2 3   Cost: 3

3 3 2 1   Cost: 1
3 1 2 2   Cost: 1
3 3 1 1   Cost: 2
1 1 2 1   Cost: 2
3 3 3 2   Cost: 3
2 2 2 3   Cost: 3

CROSSOVER
3 3 2 2   Cost: 2
3 1 2 1   Cost: 0
3 3 2 1   Cost: 1
1 1 1 1   Cost: 5

MUTATION
3 1 2 2   Cost: 1
3 3 2 1   Cost: 1
2 3 2 1   Cost: 1
1 1 1 1   Cost: 5


No solution found after 1st generation
```

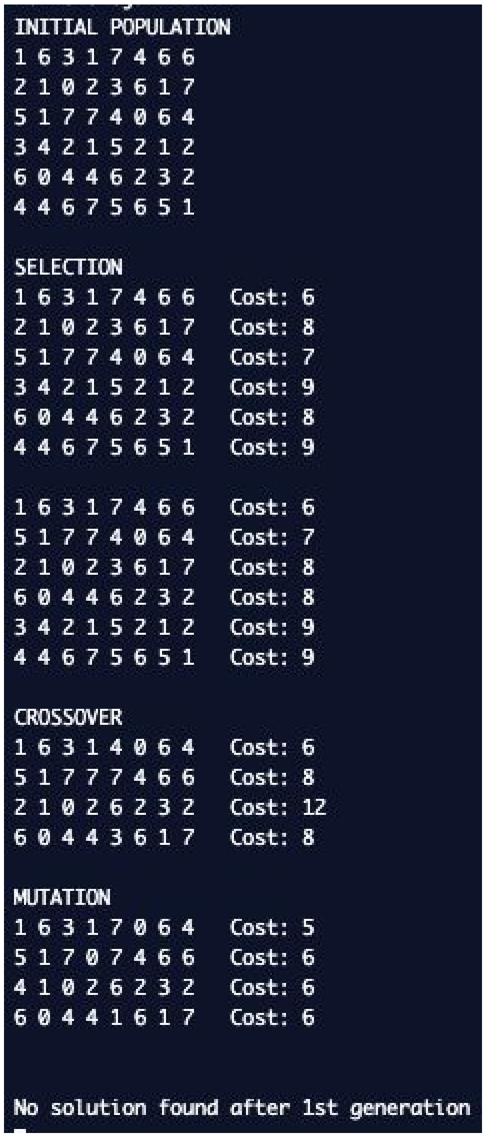


```
INITIAL POPULATION
3 3 3 2
3 3 2 1
3 3 1 1
1 1 2 1
3 1 2 2
2 2 2 3

SELECTION
3 3 3 2   Cost: 3
3 3 2 1   Cost: 1
3 3 1 1   Cost: 2
1 1 2 1   Cost: 2
3 1 2 2   Cost: 1
2 2 2 3   Cost: 3

3 3 2 1   Cost: 1
3 1 2 2   Cost: 1
3 3 1 1   Cost: 2
1 1 2 1   Cost: 2
3 3 3 2   Cost: 3
2 2 2 3   Cost: 3

CROSSOVER
3 3 2 2   Cost: 2
3 1 2 1   Cost: 0
3 3 2 1   Cost: 1
1 1 1 1   Cost: 5

MUTATION
3 1 2 2   Cost: 1
3 3 2 1   Cost: 1
2 3 2 1   Cost: 1
1 1 1 1   Cost: 5


No solution found after 1st generation
```

c. 8 Queens

```
import java.util.*;

class Main {
 private static ArrayList<Entry> list = new ArrayList<>();
 public static void main(String[] args) {
   initialize();
   selection();
   crossover();
   mutation();
   check();
 }

 static void print(){
   for(int i=0;i<list.size();i++){
     for(int j=0;j<8;j++){
       System.out.print(list.get(i).bin[j]+" ");
     }
     System.out.println("  Cost: "+list.get(i).cost);
   }
   System.out.println();
 }

 static void initialize(){
   System.out.println("INITIAL POPULATION");
   Random r = new Random();
   for(int i=0;i<6;i++){
     int temp[] = new int[8];
     int cost=0;
     //int weight=0;
     for(int j=0;j<8;j++){
       temp[j] = r.nextInt(8);
     }
     list.add(new Entry(temp));
   }
   cost();
   for(int i=0;i<list.size();i++){
     for(int j=0;j<8;j++){
       System.out.print(list.get(i).bin[j]+" ");
     }
     System.out.println();
   }
   System.out.println();
 }

 static void selection(){
   System.out.println("SELECTION");
   print();
   for(int i=0;i<list.size();i++){
     if(list.get(i).cost>=10){
       list.remove(i);
       i--;
     }
   }
   list.sort(Comparator.comparingInt(Entry :: getCost));
   print();
 }

 static void crossover(){
   System.out.println("CROSSOVER");
   if(list.size()>=4){
     for(int i=4;i<list.size();i++){
       list.remove(i);
       i--;
     }
     int crossoverPoint = 8/2;
     for(int i=0;i<list.size();i=i+2){
       Entry upper = list.get(i);
       Entry lower = list.get(i+1);

       for(int j=crossoverPoint;j<8;j++){
         int temp = upper.bin[j];
         upper.bin[j] = lower.bin[j];
         lower.bin[j] = temp;
       }
     }
     cost();
     print();
   }
 }

 static void mutation(){
   System.out.println("MUTATION");
   Random r = new Random();
   for(int i=0;i<list.size();i++){
     list.get(i).bin[r.nextInt(8)] = r.nextInt(8);
   }
   cost();
   print();
 }

 static void cost(){
   for(int i=0;i<list.size();i++){
     list.get(i).cost = 0;
     for(int j=0;j<8;j++){
       for(int k=j+1;k<8;k++){

if((list.get(i).bin[j]==list.get(i).bin[k])||((k-j)==Math.abs(list.get(i).
bin[k]-list.get(i).bin[j]))){
```

```
          list.get(i).cost++;
        }
      }
    }
    }
  }

  static void check(){
    int found=-1;
    for(int i=0;i<list.size();i++){
      if(list.get(i).cost==0){
        found = i;
        break;
      }
    }
    if(found!=-1){
      System.out.println("\nSolution after 1st generation is: ");
      for(int j=0;j<8;j++){
        System.out.print(list.get(found).bin[j]+" ");
      }
    }
    else{
      System.out.println("\nNo solution found after 1st generation");
    }

  }
}

class Entry{
  int bin[];
  int cost;
  public Entry(int bin[]){
    this.bin = bin;
  }
  public Entry(int bin[], int cost){
    this.bin = bin;
    this.cost = cost;
  }
  int getCost(){
    return this.cost;
  }
}
```



```
INITIAL POPULATION
1 6 3 1 7 4 6 6
2 1 0 2 3 6 1 7
5 1 7 7 4 0 6 4
3 4 2 1 5 2 1 2
6 0 4 4 6 2 3 2
4 4 6 7 5 6 5 1

SELECTION
1 6 3 1 7 4 6 6   Cost: 6
2 1 0 2 3 6 1 7   Cost: 8
5 1 7 7 4 0 6 4   Cost: 7
3 4 2 1 5 2 1 2   Cost: 9
6 0 4 4 6 2 3 2   Cost: 8
4 4 6 7 5 6 5 1   Cost: 9

1 6 3 1 7 4 6 6   Cost: 6
5 1 7 7 4 0 6 4   Cost: 7
2 1 0 2 3 6 1 7   Cost: 8
6 0 4 4 6 2 3 2   Cost: 8
3 4 2 1 5 2 1 2   Cost: 9
4 4 6 7 5 6 5 1   Cost: 9

CROSSOVER
1 6 3 1 4 0 6 4   Cost: 6
5 1 7 7 7 4 6 6   Cost: 8
2 1 0 2 6 2 3 2   Cost: 12
6 0 4 4 3 6 1 7   Cost: 8

MUTATION
1 6 3 1 7 0 6 4   Cost: 5
5 1 7 0 7 4 6 6   Cost: 6
4 1 0 2 6 2 3 2   Cost: 6
6 0 4 4 1 6 1 7   Cost: 6


No solution found after 1st generation
```

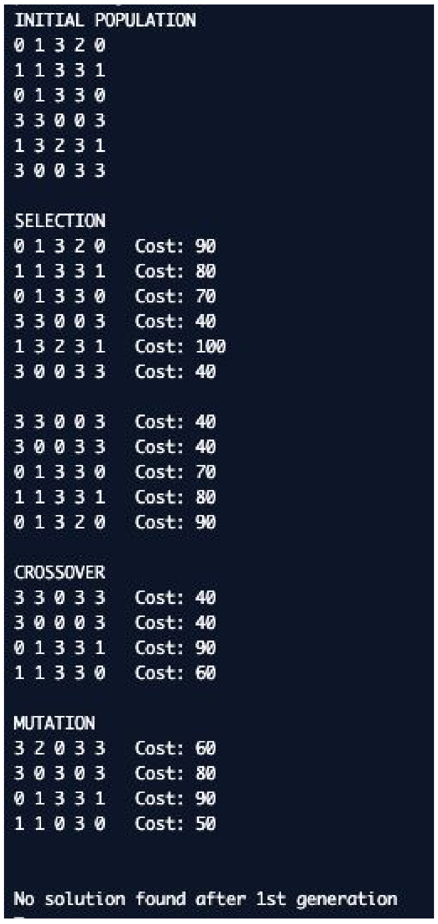
d. Travelling Salesman Problem

```
import java.util.*;

class Main {
 private static int graph[][] = new int[4][4];
 private static ArrayList<Entry> list = new ArrayList<>();
 public static void main(String[] args) {
   problem();
   initialize();
   selection();
   crossover();
   mutation();
   check();
 }

 static void problem(){
   graph[0][0] = 0;
   graph[0][1] = 10;
   graph[0][2] = 30;
   graph[0][3] = 20;
   graph[1][0] = 10;
   graph[1][1] = 0;
   graph[1][2] = 20;
   graph[1][3] = 40;
   graph[2][0] = 30;
   graph[2][1] = 20;
   graph[2][2] = 0;
   graph[2][3] = 10;
   graph[3][0] = 20;
   graph[3][1] = 40;
   graph[3][2] = 10;
   graph[3][3] = 0;
 }

 static void print(){
   for(int i=0;i<list.size();i++){
     for(int j=0;j<5;j++){
       System.out.print(list.get(i).bin[j]+" ");
     }
     System.out.println("  Cost: "+list.get(i).cost);
   }
   System.out.println();
 }

 static void initialize(){
   System.out.println("INITIAL POPULATION");
   Random r = new Random();
   for(int i=0;i<6;i++){
     int temp[] = new int[5];
```

```
      for(int j=0;j<4;j++){
        temp[j] = r.nextInt(4);
      }
      temp[4] = temp[0];
      list.add(new Entry(temp));
    }
    cost();
    for(int i=0;i<list.size();i++){
      for(int j=0;j<5;j++){
        System.out.print(list.get(i).bin[j]+" ");
      }
      System.out.println();
    }
    System.out.println();
  }

  static void selection(){
    System.out.println("SELECTION");
    print();
    for(int i=0;i<list.size();i++){
      if(list.get(i).cost>=100){
        list.remove(i);
        i--;
      }
    }
    list.sort(Comparator.comparingInt(Entry :: getCost));
    print();
  }

  static void crossover(){
    System.out.println("CROSSOVER");
    if(list.size()>=4){
      for(int i=4;i<list.size();i++){
        list.remove(i);
        i--;
      }
      int crossoverPoint = 5/2+1;
      for(int i=0;i<list.size();i=i+2){
        Entry upper = list.get(i);
        Entry lower = list.get(i+1);

        for(int j=crossoverPoint;j<5;j++){
          int temp = upper.bin[j];
          upper.bin[j] = lower.bin[j];
          lower.bin[j] = temp;
        }
      }
      cost();
      print();
    }
```

```
 }

 static void mutation(){
   System.out.println("MUTATION");
   Random r = new Random();
   for(int i=0;i<list.size();i++){
     list.get(i).bin[r.nextInt(5)] = r.nextInt(4);
   }
   cost();
   print();
 }

 static void cost(){
   for(int i=0;i<list.size();i++){
     list.get(i).cost = 0;
     for(int j=0;j<4;j++){
       int k = j+1;
       if(graph[list.get(i).bin[j]][list.get(i).bin[k]]!=0){
         list.get(i).cost = list.get(i).cost +
graph[list.get(i).bin[j]][list.get(i).bin[k]];
       }
     }
   }
 }

 static void check(){
   int found=-1;
   list.sort(Comparator.comparingInt(Entry :: getCost));
   for(int i=0;i<list.size();i++){
     int temp[] = new int[4];
     int flag=0;
     for(int j=0;j<4;j++){
       if( temp[list.get(i).bin[j]]==0){
         temp[list.get(i).bin[j]]=1;
       }
       else{
         flag=1;
         break;
       }
     }
     if(flag==0 && list.get(i).bin[0]==list.get(i).bin[4]){
       found = i;
       break;
     }
   }
   if(found!=-1){
     System.out.println("\nSolution after 1st generation is: ");
     for(int j=0;j<4;j++){
       System.out.print(list.get(found).bin[j]+" ");
     }
```

```
    }
    else{
      System.out.println("\nNo solution found after 1st generation");
    }

  }
}

class Entry{
  int bin[];
  int cost;
  public Entry(int bin[]){
    this.bin = bin;
  }
  public Entry(int bin[], int cost){
    this.bin = bin;
    this.cost = cost;
  }
  int getCost(){
    return this.cost;
  }
}
```



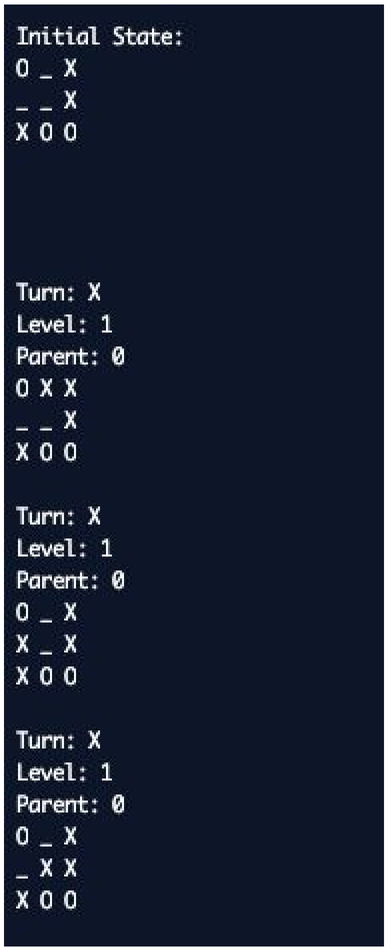
```
INITIAL POPULATION
0 1 3 2 0
1 1 3 3 1
0 1 3 3 0
3 3 0 0 3
1 3 2 3 1
3 0 0 3 3

SELECTION
0 1 3 2 0   Cost: 90
1 1 3 3 1   Cost: 80
0 1 3 3 0   Cost: 70
3 3 0 0 3   Cost: 40
1 3 2 3 1   Cost: 100
3 0 0 3 3   Cost: 40

3 3 0 0 3   Cost: 40
3 0 0 3 3   Cost: 40
0 1 3 3 0   Cost: 70
1 1 3 3 1   Cost: 80
0 1 3 2 0   Cost: 90

CROSSOVER
3 3 0 3 3   Cost: 40
3 0 0 0 3   Cost: 40
0 1 3 3 1   Cost: 90
1 1 3 3 0   Cost: 60

MUTATION
3 2 0 3 3   Cost: 60
3 0 3 0 3   Cost: 80
0 1 3 3 1   Cost: 90
1 1 0 3 0   Cost: 50


No solution found after 1st generation
```

6. Min Max - Tic Tac Toe

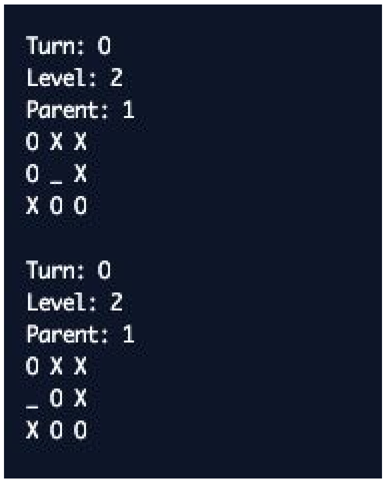
```
import java.util.*;

class Main {
 public static ArrayList<State> states = new ArrayList<State>();
 public static char turnval = 'X', blank = ' ';
 public static void main(String[] args) {
   Main.minmax();
   Main.backtrack();
 }

 public static void minmax(){
   states.add(new State(new char[]{'O', blank, 'X', blank, blank, 'X',
'X', 'O', 'O'}, -1, 0));
   for(int j=0;j<9;j++){
     if(j%3==0)
       System.out.println();
     System.out.print(states.get(states.size()-1).a[j]+" ");
   }
   System.out.println("\n\n");


   int l=1;
   int noOfBlanks = 3;
   int newNoOfBlanks=0;
   int count = noOfBlanks;
   for(int node=0; node<states.size(); node++){
     if(states.get(node).cost == 0){
       for(int i=0;i<9;i++){
         if(states.get(node).a[i]==blank){
           count--;

           char tempArray[] = new char[9];
           for(int j=0;j<9;j++){
             tempArray[j] = states.get(node).a[j];
           }

           tempArray[i] = turnval;

           State s = new State(tempArray, node, states.get(node).level+1);

           if(checkwin(s, turnval)){
             if(turnval =='X'){
               s.cost = 10;
             }
             else{
               s.cost = -10;
             }
```

```
            }
            else{
              newNoOfBlanks = newNoOfBlanks + (noOfBlanks-1);
            }
            states.add(s);


            System.out.print("\n"+turnval+" "+ node);
            for(int j=0;j<9;j++){
              if(j%3==0)
                System.out.println();
              System.out.print(states.get(states.size()-1).a[j]+" ");
            }
            System.out.print("\n "+states.get(states.size()-1).parent+"
"+states.get(states.size()-1).level+" \n");
          }
        }
        System.out.println("\n\n");

      }
      if(count==0){
        if(turnval == 'X'){
            turnval = 'O';
        }
        else{
            turnval = 'X';
        }
        noOfBlanks = newNoOfBlanks;
        newNoOfBlanks = 0;
        count = noOfBlanks;
      }
    }

 }

 public static boolean checkwin(State s, char turn){
   char temp[] = s.a;
   if((temp[0]==turn && temp[3]==turn && temp[6]==turn) || (temp[1]==turn
&& temp[4]==turn && temp[7]==turn) || (temp[2]==turn && temp[5]==turn &&
temp[8]==turn) || (temp[0]==turn && temp[4]==turn && temp[8]==turn) ||
(temp[2]==turn && temp[4]==turn && temp[6]==turn) || (temp[0]==turn &&
temp[1]==turn && temp[2]==turn) || (temp[3]==turn && temp[4]==turn &&
temp[5]==turn) || (temp[6]==turn && temp[7]==turn && temp[8]==turn)){
      return true;
   }
   return false;
 }

 public static void backtrack(){
   int i=states.size()-1;
```

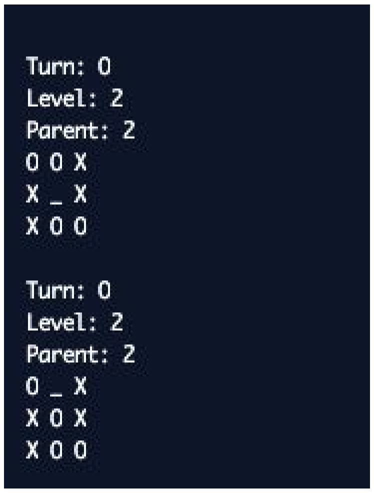
```
    while(i>0){
      int par = states.get(i).parent;
      int turn = states.get(i).level%2;
      ArrayList<Integer> tempStates = new ArrayList<>();
      while(states.get(i).parent == par){
        tempStates.add(states.get(i).cost);
        i--;
      }
      int newCost=0;
      if(turn==1){
        newCost = Collections.max(tempStates);
      }
      else{
        newCost = Collections.min(tempStates);
      }
      states.get(par).cost = newCost;
    }
    System.out.println(states.get(0).cost);
  }

}

class State{
 char a[]= new char[9];
 int parent, cost, level;
 public State(char a[], int parent, int level){
    this.a = a;
    this.parent = parent;
    this.cost = 0;
    this.level = level;
  }
}
```



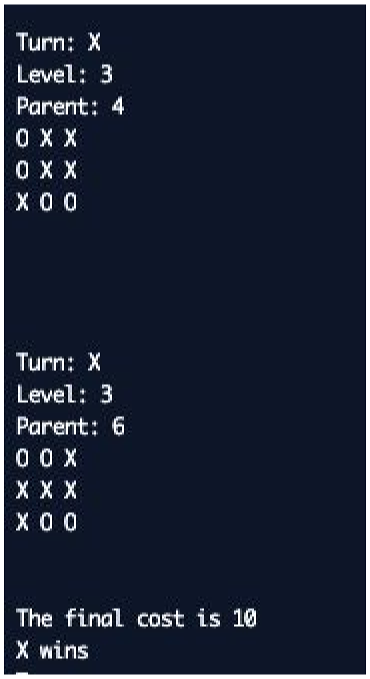
```
Initial State:
O _ X
_ _ X
X O O



Turn: X
Level: 1
Parent: 0
O X X
_ _ X
X O O

Turn: X
Level: 1
Parent: 0
O _ X
X _ X
X O O

Turn: X
Level: 1
Parent: 0
O _ X
_ X X
X O O
```



```
Turn: O
Level: 2
Parent: 1
O X X
O _ X
X O O

Turn: O
Level: 2
Parent: 1
O X X
_ O X
X O O
```



```
Turn: O
Level: 2
Parent: 2
O O X
X _ X
X O O

Turn: O
Level: 2
Parent: 2
O _ X
X O X
X O O
```



```
Turn: X
Level: 3
Parent: 4
O X X
O X X
X O O




Turn: X
Level: 3
Parent: 6
O O X
X X X
X O O


The final cost is 10
X wins
```