توصيف المسائل برمجيا:

هدف الجلسة:
القدرة على التعبير البرمجي عن المشاكل لنتمكن من الاستفادة من هذا التوصيف بربطها بأحد خوارزميات البحث الذكية في محاولة لإيجاد الحل.

بيئة العمل: لغة البرمجة المستخدمة هي Java يمكن العمل على أي بيئة تدعمها.

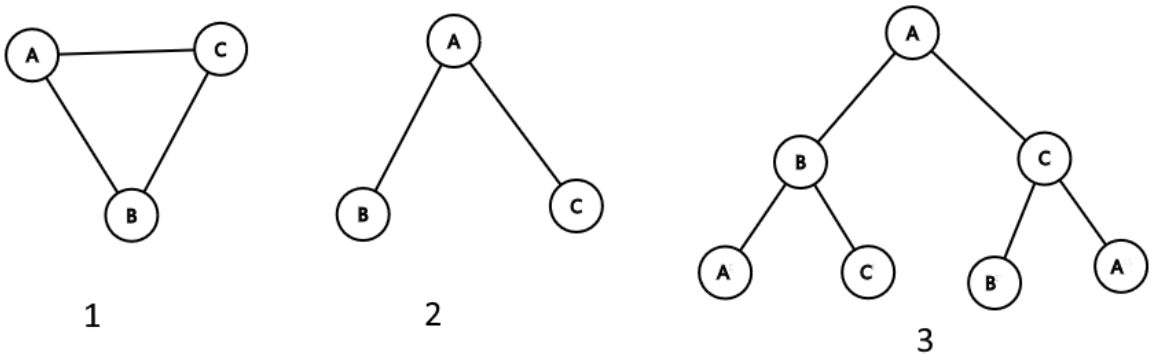
خوارزميات البحث الذكية هي نوع من أنواع الخوارزميات تعتمد استراتيجية ما لبلوغ هدف معين.

هناك نوعان من البحث:
البحث عن طريقPath Search : وتكون حالة الهدف معروفة فنبحث عن طريق لحالة الهدف.
Configuration Search:لا تكون حالة الهدف معروفة فيجب اختبار الحالة لمعرفة إن كانت حالة هدف ,أي نبحث عن الطريق و حالة الهدف.فالحل هو عبارة عن طريق من الحالة البدائية.

هناك نوعان من الخوارزميات:
خوارزميات بحث على مستوى شاملGlobal Search : نبحث به في كل الفضاء الاحتمالي وله القدرة على إيجاد حل أمثلي.
خوارزميات بحث على مستولا محليLocal Search : ننطلق من حل بدائي ليس بالضرورة أن يكون أمثليا ونبحث في المستوى المحيط بالحل فليس له القدرة على إيجاد حل أمثلي غالبا لأنه لا يبحث في كل الفضاء.
وتصنف الخوارزميات بطريقة أخرى:

خوارزميات عمياءBlind : لا تملك أي فكرة عما يتبقى للوصول للحل ولا تعرف كيفية الوصول لهذا الحل.
خوارزميات البحث مع معلوماتInformed : تملك فكرة عما تبقى للوصول للهدف فيمكن تطبيق توابع لتوقع المسافة المتبقية للهدف.
كما يمكن أن نميز بين نوعين للبحث:
بحث بيانيGraph Search : لا يقوم بتوليد الابن إن كان مكرراً مسبقاً.

بحث شجريTree Search : يقوم بتوليد الابن ولو كان مكرراً مسبقاً.



توضح الصورة التالية نمطي البحث حيث يمثل النمط 2 بحث بياني بالنسبة للبيان1 والنمط 3 هو بحث شجري في البيان1

كيف تعمل خوارزمية البحث(tree search) :

1. يكون لديها ما يسمى container (يمكن اعتبارها على أنها سلة) فتأخذ من المسألة في البداية الحاالة الابتدائيةونضعها في هذه السلة.
2. تفوم بما يسمى pick أي أنها تأخذ من السلة حالة (يكون اختيار الحالة تبعا للاستراتيجية التي سنتبعها وسنوضح ذلك لاحقا) ثم تفحصها check فيما إذا كانت حل أو لا فإذا كانت حل تتوقف الخوارزمية وتعيد الحل وإلا تنتقل إلى 3
3. تقوم بعمل expand للحالة الموجودة في السلة أي تعطينا كل الحالات التالية لهذه الحالة (أي الأبناء) وتضيفهم إلى السلة ونعود للخطوة2

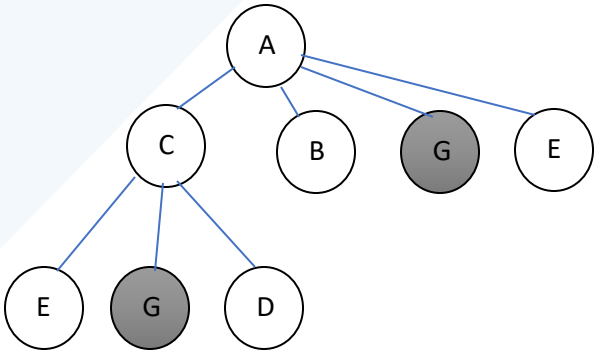
نلاحظ انه تبعا للخوارزمية السابقة تنتج شجرة لذلك سميت tree search

يجب التفريق بين حالة العقدة state والغقدة node حيث أن حالة العقدة هي أحد معاملات العقدة

State, actions, parent, depth and path_cost

والعقدة هي بنية بيانات حسابية تستخدم لتمثيل شجرة البحث فإذا كان المثال التالي :

العقدة ذات الحالة G في العمق 1 مختلفة عن العقدة ذات الحالة G في العمق 2 كل منهما مولد عن طريق طريقي بحث مختلفين.

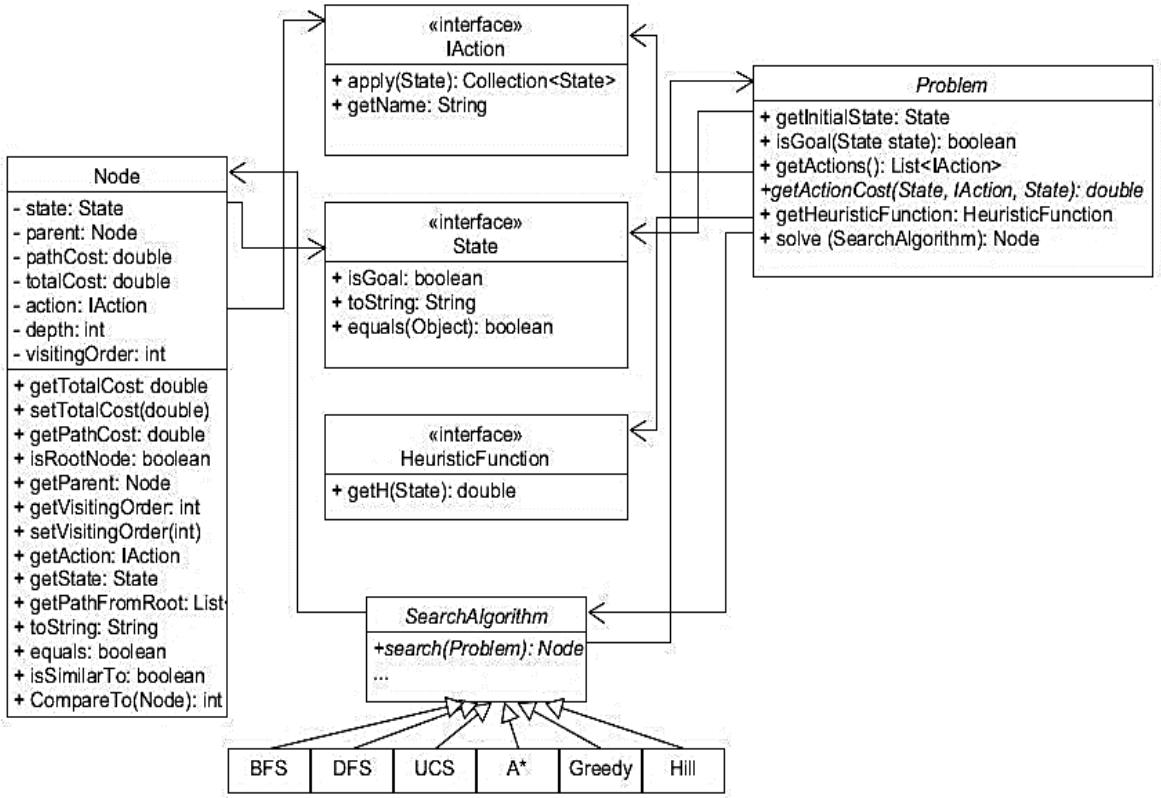


توصيف المسائل:

لتوصيف المسائل سنتعامل مع 5 مصطلحات رئيسية وهي:

1. فضاء الحالة State space
2. الحالة البدائية Initial state
3. الهدف Goal
4. الانتقالات Actions
5. كلفة الانتقال Action cost

الشكل التالي يوضح class يعبر عن المشكلة problem ومجموعة من الواجهات والصفوف التي تعبر برمجيا عن طريقة توصيف مسألة



سيتم شرح الواجهات والصفوف ضمن المخطط السابق وآلية الربط بينهما.

لدينا واجهة باسم state - اجل تعريف فضاء الحالة للمسألة :

```
package core;
//implement this interface to define the state space of a problem
public interface State {
    //checks whether the state calling is a goal state or not
    Boolean isGoal();
    // the way to show the state in the output
    String to String();
    // checks calling state with passed state s for equality
    Boolean equals(Object s); }
```

الطرائق- :

- isGoal():لذي يرد True في حال كانت الحالة التي تستدعيه مساوية لحالة الهدف و False عدا ذلك
- toString(): سيتم تحقيقه بالصفوف القادمة و هو يرد قيمة String مكافئة للحالة التي لدينا (تابع خاص بإظهار الحالة بالخرج)
- equals():يرد True اذا كانت الحالة التي تستدعيه مساوية للكائن الممرر له و Falseعدا ذلك

الصف المجرد proplem

```
package core;
 import java.util.List;
 //this class describes a problem in general // extend it for a specific problem description // it needs a state space i.e. State and a search strategy i.e. Search Algorithm
 public abstract class Problem {
//returns the initial state of problem
public abstract State getInitialState();
// returns true when the the state passed is a goal state or when ever we reached a goal
 public abstract boolean isGoal(State state);
 // returns a list of actions (operators) that can be applied to any state of the problem public abstract List<IAction> get Actions();
 // returns the cost of applying the action a on state a to get state ss
 public abstract double getActionCost(State s, IAction a, State ss);
 //used to initiate the search strategy to be used to solve the problem and run the the algorithm sa on this problem.. //it returns the goal Node as result which satisfy the isGoal method
public Node solve (SearchAlgorithm sa){ return sa.search(this);}}
```

الطرائق :

- getInitialState(): هي طريقة مجردة تستدعيها problem و ترد الحالة الابتدائية عند تحقيقها
- isGoal(State state): طريقة مجردة تستدعيها problem و نمرر لها state فاذا كانت هي حالة الهدف فترد True و الا ترد False عند تحقيقها بصف ابن
- getActions():طريقة مجردة تستدعيها problem و ترد لائحة مين ال actions التي يمكن تطبيقها على أي state من حالات الproblem عند تحقيق الطريقة المجردة
- getActionCost():طريقة مجردة تستدعيهproblem يرد قيمة doubleاعتمادا على تمريرنا له ال state التي وصلنا لها ss و ال action الذي طبقناه للوصول الى هذه الحالة و الحالة السابقة ل ss و هيs
- solve(SearchAlgorithm sa):طريقة ليست مجردة يستدعيها كائن للصف نفسه (لكن معرف بيا ني صف ابين)تستدعيها problem لترد لنا عقدة الهدف وفق استراتيجية حل sa نختارها

```
package core;

import java.util.ArrayList;

import java.util.List;

//represents a node in the search tree //implements Comparable to be used in PriorityQueue ordering

public class Node implements Comparable<Node> {

    private UUID id;

//the corresponding state in state space

    private State state;

 //node parent which led us to this node

    private Node parent;

//the cost from initial state to this node i.e. g(n)

    private double pathCost;
```

```
//total estimated cost i.e. f(n)
private double totalCost;
//the action applied to get to this node
private IAction action;
//the node Depth;
 private int depth;
//the order of picking the node;
 private int visitingOrder;
public double getTotalCost() { return totalCost; }
public void setTotalCost(double c){ totalCost = c; }
//initial state ctor
 public Node(State state) {
        id= randomUUID();
         this.state = state;
        pathCost = 0.0;
        totalCost = Double.MAX_VALUE;
        visitingOrder = 0; }
```

```
public Node(State state, Node parent, IAction action, double pathCost) {
        this(state);
        this.parent = parent;
         this.action = action;
         this.pathCost = pathCost;
         this.totalCost = pathCost;
        //when total cost is not provided treat the pathCost as the totalCost in
        order to  be used correctly when using PriorityQueue as it  deals with
        totalCost.
        }
         public Node(State state, Node parent, IAction action, double
        pathCost, double totalCost) {
         this(state, parent, action, pathCost);
         this.totalCost = totalCost;
        }
         public double getPathCost() { return pathCost; }
        public boolean isRootNode() { return parent == null; }
        public Node getParent() { return parent; }
        public int getVisitingOrder(){ return visitingOrder; }
        public  void  setVisitingOrder(int  visitingvisitingOrder){
        this.visitingOrder = visitingOrder; }
         public IAction getAction() { return action; }
        public State getState() { return state; }
```

```
public List<Node> getPathFromRoot() {
 List<Node> path = new ArrayList<Node>();
 Node current = this;
 while (!current.isRootNode()) {
        path.add(0, current);
        current = current.getParent(); }
 // ensure the root node is added
        path.add(0, current); return path; }
 @Override
public String toString() {
return (parent == null?"initial State ":parent)+ "action: "+(action==null?"no action":action.getName())+ ", pathCost: "+ pathCost+ ", state: \n"+ getState() + "\n"; }
@Override
public boolean equals(Object o) {
 //two nodes are said to be equal when they hold same state Node t = (Node) o;
 return t.state.equals(state); }
 //this used when only when building search tree diagram..
public boolean isSimilarTo(Node n) {
//two nodes are said to be similar when they hold same info.
 return this.parent == n.parent && this.pathCost == n.getPathCost() && this.totalCost == n.getTotalCost() && this.state.equals(n.getState());
 }
@Override
public int compareTo(Node node) {
```

```
if (getTotalCost() > node.getTotalCost())
return 1;
if (getTotalCost() < node.getTotalCost())
                    return -1;
return 0; } }
```

الطرائق:

- getTotalCost(): يرد قيمة الكلفة الكلية للعقدة التي تستدعيه
- getpathCost(): يرد قيمة المتحول pathCost والذي هو الكلفة للوصول لهذه العقدة من الجذر.
- isRootNode(): تابع تستدعيه عقدة لنتأكد فيما اذا كانت هي العقدة الجذر او لا فنقوم باختبار اذا كان اب هذه العقدة هو null فيرد true و اذا لم يكن كذلك فيردfalse
- getParent(): يرد لنا العقدة الاب للعقدة الحالية
- getAction(): يرد ال action الذي اوصلنا الى هذه العقد ة
- getState(): يرد ال state الموافقة لهذه العقدة
- getPathFromRoot(): طريقة ترد لنا سلسلة ب اسم path تحتوي على العقد من العقدة الحالية رجوعا الى العقدة الجذر
- toString(): تابع يرد String أي يقوم برد سلسلة محرفية تتضمن حالة العقدة الحالية و العقدة الاب لها و ال action الذي اوصلنا اليها و كلفة المسار من العقدة الجذر الى العقدة الحالية
- equals():يختبر فيما اذا كانت عقدتين متساويتين و تكون عقدتين متساويتين اذا كانت حالتيهما متساويتين
- isSimilarTo(): تكون عقدتين متشابهتين اذا كان لديهما نفس المعلومات (نفس الاب, نفس الحالة,نفس ال, pathcost نفس ال) totalcost و يتم استخدام هذا التابع عند بناء شجرة البحث فيرد true اذا كانت تتشابه العقدة التي تستدعي التابع و العقدة الممررة له بنفس المعلومات واذا لم يكن كذ لك فيردfalse
- CompareTo(): يرد 1 اذا كانت الكلفة الكلية للعقدة التي تستدعي التابع اكبر من الكلفة الكلية للعقدة الممررة له و 1 - اذا كان العكس و 0 اذا كانتا متساويتين وهو التابع الذي يستخدم من أجل ترتيب العقد ضمن

ال priorityQueue وحصلنا عليه من تحقيق الواجهةComparable

الصف المجردSearchAlgorithm

```
package core;
import java.util.ArrayList;
 import java.util.List;
 public abstract class SearchAlgorithm {
        protected boolean useGraphSeach = false;
        //the search strategy .. return the goal node regarding the problem p; // it search for the goal using problem definition going from initial state of p applying actions to reach the goal state..
        public abstract Node search(Problem p)};
```

الواجهة IAction - سنستخدمها لتعريف ال actions التي سنطبقها:

```
package core;
import java.util.Collection;
 //implement this interface to define an action
 public interface IAction{
 //returns a collection of states as a result of applying the action to the state s
      Collection<State> apply(State s);
 //retuns actionName.. useful in some places
       String getName();
```