



TD1

Intelligence artificielle



Encadré par

Pr : Belcaid

Réalisé par le binôme :

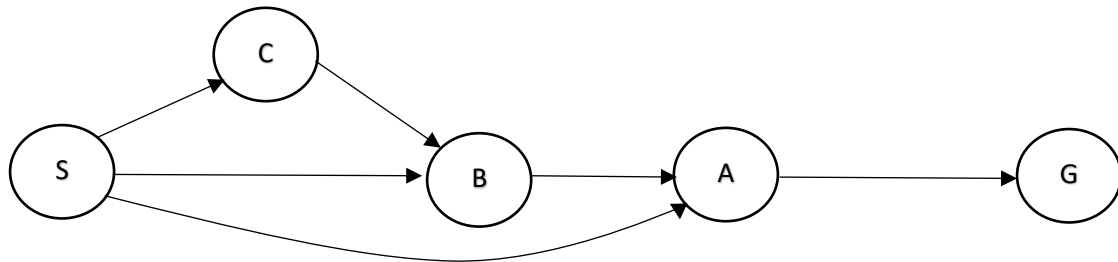
NOUAR Safae

ZINE Meryeme

Année universitaire : 2018/2019



Exercice 1 :



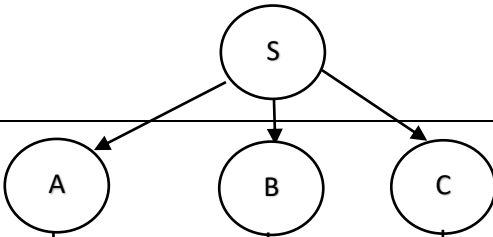
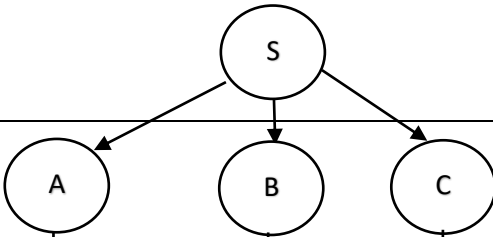
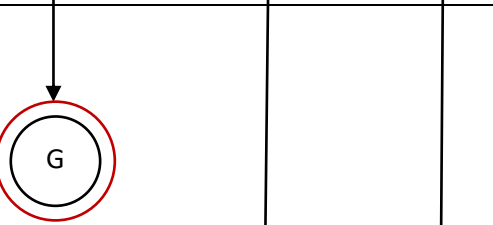
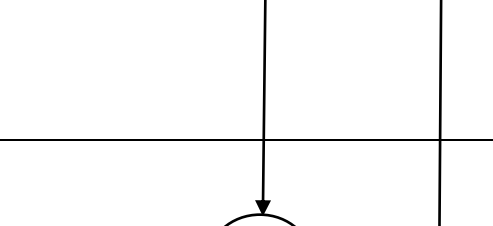
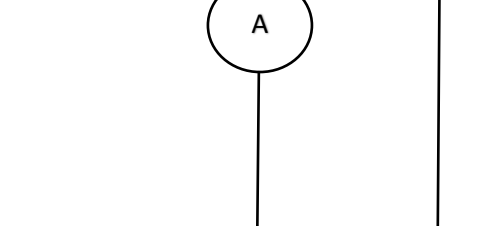
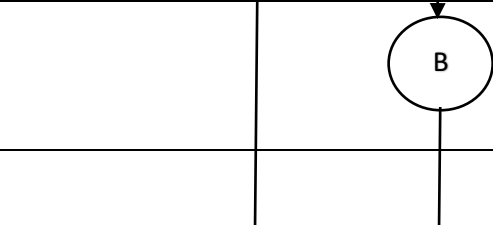
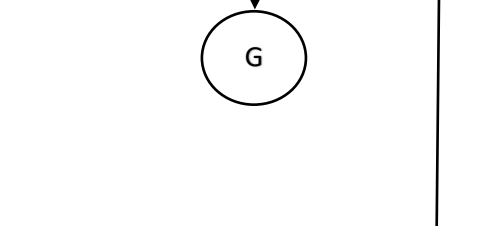
⇒ Depth-First Search :DFS

Frontière	Arbre	Profondeur	Commentaires
S		(0)	- On commence par une profondeur de 0.
S,A		(1)	- On développe notre arbre on trouve 3 nœuds possibles A,B et C.
S,B		(1)	
S,C		(1)	
S,A,G		(2)	- Puisque les 3 nœuds sont de même profondeur on développe l'arbre selon l'ordre alphabétique, donc on développe A et on trouve notre objectif G et le chemin S,A,G devient le chemin le plus profond par suite on arrête le développement .

✓ Le chemin choisi par la **DFS** est **S,A,G** puisqu'elle priorise la profondeur



⇒ **Breadth-First Search: BFS**

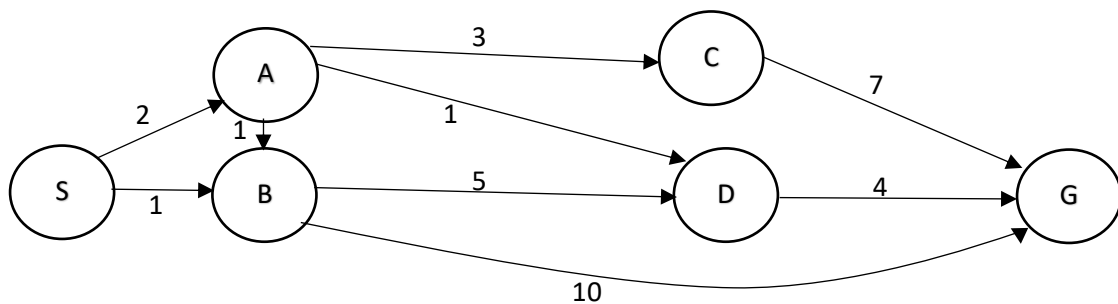
Frontière	Arbre	Largeur	Commentaires
S		(0)	- On commence par une profondeur de 0.
S,A		(1)	- On développe notre arbre on trouve 3 nœuds possibles A,B et C.
S,B		(1)	
S,C		(1)	
S,A,G		(2)	- Puisque les 3 nœuds sont de même profondeur on développe l'arbre selon l'ordre alphabétique, donc on développe A et on trouve notre objectif G mais on n'arrête le développement qu'après l'extraction de G.
S,B,A		(2)	- On compare les profondeurs et on développe l'arbre la plus petite. On les nœuds B et C possédant une même profondeur de valeur 1 donc selon l'ordre alphabétique on développe B.
S,C,B		(2)	- On développe le nœud C ayant une profondeur moins que celle de A et B.
S,B,A,G		(3)	- Tous les nœuds ont une autre fois la même profondeur donc par ordre alphabétique on développe A et on trouve notre objectif G mais on n'arrêtera le développement des nœuds qu'après l'extraction du G
S,C,B,A		(3)	- On compare entre le nœud B et G. On trouve qu'ils ont la même profondeur donc



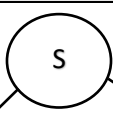

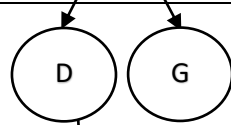
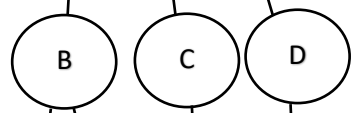
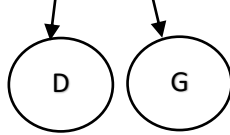
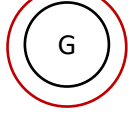
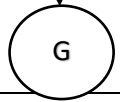
			<p>par ordre alphabétique on développera le nœud B. Après on compare les chemins on trouve que SAG possède la profondeur la plus petite du coup l'objectif G sera extrait.</p>
--	--	--	--

- ✓ Le chemin choisi par la **BFS** est **S,A,G** puisqu'elle priorise la largeur du coup le chemin ayant la moindre profondeur est le chemin convenable.

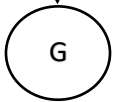
⇒ Uniform Cost Search: UCS





Frontière	Arbre	Coût Accumulé	Commentaires
S		(0)	- On commence par une profondeur de 0.
S,A		(2)	- On développe notre arbre on trouve 2 nœuds possibles A et B.
S,B		(1)	
S,B,D		(6)	-On compare les 2 couts (de A et B) et on développe celui ayant un cout minimal qui est le B et on trouve par suite 2 nœuds D et G avec un cout accumulé de 6 et 11 respectivement.
S,B,G		(11)	
S,A,B		(3)	-On compare maintenant le cout des 3 nœuds :A, D et G et on développe une autre celui possédant un cout accumulé minimal donc on développe A et on obtient 3 nœuds B,C et D avec un cout 3,5 et 3 respectivement.
S,A,C		(5)	
S,A,D		(3)	
SA,B,D		(8)	-Le même principe se répète mais cette fois ci on a 2 nœuds ayant le même coût donc on développe selon l'ordre alphabétique, donc on développe le B et on trouve D et G.
S,A,B,G		(13)	
S,A,D,G		(7)	-On compare les nœuds des dernières feuilles (SAD>3 ; SAC>5 ; SBD>6 ; SBG>11 ; SABD>8 ; SABG>13) Donc on développe SAD et on obtient notre objectif G avec un coût accumulé de 7 mais on n'arrêtera le développement qu'après l'extraction de l'objectif.
S,A,C,G		(12)	-On développe SAC qui a un coût accumulé minimal de 5 et on obtient une autre



			fois notre objectif G avec un coût de 12.
S,B,D,G		(10)	-On développe SBD et on obtient le G avec un coût de 10. On compare les coûts, on trouve que le chemin SADG est le chemin ayant le coût minimal du coup l'objectif G est extrait.

- ✓ Le chemin choisi par la **UCS** est **S,A,D,G** puisqu'elle priorise le coût et le chemin possédant le coût minimal devient le chemin convenable



TP1

Intelligence artificielle



Encadré par

Pr : Belcaid

Réalisé par le binôme :

NOUAR Safae

ZINE Meryeme

Année universitaire : 2018/2019

TP1

Familiarisation avec l'environnement virtuel et l'autograder

Objectif :

L'objectif de ce premier TP est de se préparer pour les projets principaux du cours en :

- Installant et préparant un **environnement virtuel** .
- utilisant l'outil **autograder** pour vérifier et valider les réponses

Installation d'Anaconda :

- Puisqu'on a utilisé le langage python, il était recommandé d'installer le gestionnaire de paquetage anaconda qui facilite l'installation et la manipulation des modules et environnements python.
- Pour vérifier l'installation, on lance la commande :

conda -V

! ce n'est pas obligatoire d'utiliser anaconda dans le cas de python2.

Environnement virtuel :

Pour ne pas casser les dépendances avec le système. Nous avons créé un environnement AI qui utilise python2 par la commande **Erreur ! Signet non défini.:**

conda create -n AI python=2.7 anaconda

Puis on l'active par : **source activate AI**

Autograder :

Autograder est un outil qui permet la notation automatique de notre solution.

Pour l'utiliser il fallait télécharger le projet 'tutorial' où on trouve :

```
ls
addition.py      shopAroundTown.pyc  testParser.pyc
addition.pyc     shop.py              textDisplay.py
autograder.py    shop.pyc             textDisplay.pyc
buyLotsOfFruit.py shopSmart.py         town.py
buyLotsOfFruit.pyc shopSmart.pyc        town.pyc
grading.py       submission_autograder.py tutorialTestClasses.py
grading.pyc      test_cases           tutorialTestClasses.pyc
projectParams.py testClasses.py       util.py
projectParams.pyc testClasses.pyc      util.pyc
shopAroundTown.py testParser.py        VERSION
```

QUESTION 1 : Addition

On change le contenu de la fonction **add** qui se trouve dans le fichier : **addition.py**, pour calculer la bonne solution, comme suit :

```
def add(a, b):
    "Return the sum of a and b"
    "**** YOUR CODE HERE ****"
    return a+b
```

On exécute la commande "**python autograder.py -q q1**" pour vérifier notre solution:

```
python autograder.py -q q1
Starting on 3-30 at 12:59:57

Question q1
=====

*** PASS: test_cases/q1/addition1.test
*** add(a,b) returns the sum of a and b
*** PASS: test_cases/q1/addition2.test
*** add(a,b) returns the sum of a and b
*** PASS: test_cases/q1/addition3.test
*** add(a,b) returns the sum of a and b

### Question q1: 1/1 ###

Finished at 12:59:57

Provisional grades
=====
Question q1: 1/1
-----
Total: 1/1
```

Question 2: Acheter des Fruits :

On change la fonction **buyLotsOfFruit(orderList)** dans le fichier **buyLotsOfFruit.py**, qui prend comme argument une liste de couples (**fruit**, **pound**) et renvoie le **prix** de cette liste.

S'il y a un fruit dans la liste qui ne figure pas dans la liste de fruits offerts par le magasin, la fonction affiche un message d'erreur et renvoie la valeur **None**.

```
def buyLotsOfFruit(orderList):
    """
        orderList: List of (fruit, numPounds) tuples

    Returns cost of order
    """
    totalCost = 0.0
    prices = [ ]
    """ YOUR CODE HERE """
    for fruit , numPounds in orderList:
        if fruit in fruitPrices :
            prices.append(numPounds*fruitPrices[fruit])
        else :
            print 'le fruit :',fruit , 'ne figure pas dans la liste !'
            return None

    return sum(prices)
```

En utilisant la fonction append qui permet d'ajouter un élément à la fin d'une liste (dans notre cas la liste des prix totaux des fruits) et puis on retourne la somme des éléments de cette dernière en utilisant la fonction sum .

Autre solution :

```
def buyLotsOfFruit(orderList):
    """
        orderList: List of (fruit, numPounds) tuples

    Returns cost of order
    """
    totalCost = 0.0
    prices = [ ]
    """ YOUR CODE HERE """
    for fruit , numPounds in orderList:
        if fruit in fruitPrices :
            totalCost += numPounds*fruitPrices[fruit]

        else :
            print 'le fruit :',fruit , 'ne figure pas dans la liste !'
            return None

    return totalCost
```

On exécute la commande : **python buyLotsOfFruit.py** pour voir le résultat:

```
python buyLotsOfFruit.py
Cost of [('apples', 2.0), ('pears', 3.0), ('limes', 4.0)] is 12.25
```

On exécute la commande " **python autograder.py -q q2** " pour vérifier notre solution :

```
python autograder.py -q q2
Starting on 3-30 at 13:06:51

Question q2
=====

*** PASS: test_cases/q2/food_price1.test
*** buyLotsOfFruit correctly computes the cost of the order
*** PASS: test_cases/q2/food_price2.test
*** buyLotsOfFruit correctly computes the cost of the order
*** PASS: test_cases/q2/food_price3.test
*** buyLotsOfFruit correctly computes the cost of the order

### Question q2: 1/1 ###

Finished at 13:06:51

Provisional grades
=====
Question q2: 1/1
-----
Total: 1/1
```

Question 3 : acheter avec meilleur prix

On remplit la fonction **shopSmart(orders, shops)** qui se trouve dans le fichier **shopSmart.py**.

Cette fonction prend, comme argument, une liste « orders » comme fruits à acheter avec leur poids. Elle prend aussi une liste de magasins shops où on peut acheter ces fruits. Elle doit renvoie le magasin avec le **coût minimal** de notre ordre.

On l'a rempli comme suit :

```
def shopSmart(orderList, fruitShops):
    """
        orderList: List of (fruit, numPound) tuples
        fruitShops: List of FruitShops
    """
    "*** YOUR CODE HERE ***"
    totalCost = 0.0
    bestcost = 100
    best = ''
    "*** YOUR CODE HERE ***"
    for y in fruitShops :|
        totalCost = 0.0
        for fruit, numPound in orderList:
            totalCost += y.getCostPerPound(fruit)*numPound
        if totalCost < bestcost :
            bestcost = totalCost
            best = y
    return best
```

En utilisant la fonction : **getCostPerPound(self,fruit)** (qui prend comme paramètre le fruit de l'orderlist à chercher dans le shop et retourne son prix) du fichier **shop.py** qu'on a importé avec :

```
import shop
```

La fonction :

```
def getCostPerPound(self, fruit):  
    """  
        fruit: Fruit string  
        Returns cost of 'fruit', assuming 'fruit'  
        is in our inventory or None otherwise  
    """  
    if fruit not in self.fruitPrices:  
        return None  
    return self.fruitPrices[fruit]
```

Autre solution qui est plus optimale :

```
def shopSmart(orderList, fruitShops):  
    """  
        orderList: List of (fruit, numPound) tuples  
        fruitShops: List of FruitShops  
    """  
    """*** YOUR CODE HERE ***"""  
    totalCost = 0.0  
    bestcost = 100  
    best = ''  
    """*** YOUR CODE HERE ***"""  
    for y in fruitShops :  
        totalCost = 0.0  
        totalCost = y.getPriceOfOrder(orderList)  
        if totalCost < bestcost :  
            bestcost = totalCost  
            best = y  
    return best
```

En utilisant la fonction **getPriceOfOrder(self,orderList)** (qui prend comme paramètre l' « orderlist » et retourne le prix totale de ce qu'elle contient) du fichier **shop.py** :

```
def getPriceOfOrder(self, orderList):
    """
        orderList: List of (fruit, numPounds) tuples

        Returns cost of orderList. If any of the fruit are
    """
    totalCost = 0.0
    for fruit, numPounds in orderList:
        costPerPound = self.getCostPerPound(fruit)
        if costPerPound != None:
            totalCost += numPounds * costPerPound
    return totalCost
```

Le résultat qui nous montre le magasin avec le **coût minimal** de chacun de nos ordres :

```
python shopSmart.py
Welcome to shop1 fruit shop
Welcome to shop2 fruit shop
For orders [('apples', 1.0), ('oranges', 3.0)] , the best shop is shop1
For orders: [('apples', 3.0)] , the best shop is shop2
```

Autograder :

```
### Question q3: 1/1 ###

Finished at 13:28:04

Provisional grades
=====
Question q3: 1/1
-----
Total: 1/1
```

Total :

On utilise la commande : **python autograder.py**

Provisional grades

=====

Question q1: 1/1

Question q2: 1/1

Question q3: 1/1

Total: 3/3