[IA02] Examen Final – P19

Jeudi 27 juin 2019

Remarques liminaires

> Documents autorisés : deux feuilles A4 recto/verso **manuscrites**.
> Toute ambiguïté, erreur ou omission dans le sujet devra être résolue par le candidat et par lui seul.
> 1 point sera réservé au soin et à la clarté apportés à la copie. L'élégance des solutions proposées sera prise en compte dans la notation.
> Barème purement indicatif, susceptible de changement sans préavis : 2 + 3 + 2 + 12 + 1.

Exercice 1 : logique du premier ordre

Soient les 3 formules logiques suivantes :

- F1 = $\forall x\ (p(x) \rightarrow \neg q(x))$
- F2 = $\exists x\ (q(x) \wedge r(x))$
- F3 = $\exists x\ (r(x) \wedge \neg p(x))$

Mettre F1 et F2 sous forme de Skolem.

Montrer, en utilisant la résolution, que $\{F1,F2\} \vdash F3$.

Exercice 2 : recherche dans un espace d'états

2.1 Recherche arborescente

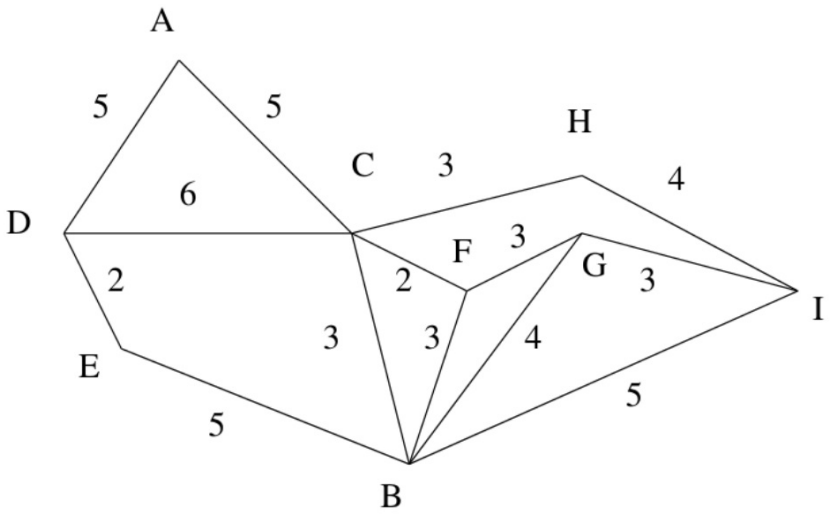
Questions

Faire un tableau présentant les avantages/défauts comparés des 3 algorithmes de recherche suivants :

- recherche en largeur ;
- recherche en profondeur ;
- recherche par approfondissement itératif.

2.2 Algorithme glouton et algorithme A*

Soit le graphe



On considère l'heuristique *h* suivante pour la recherche du plus court chemin permettant d'aller de A à I (c.-à.-d. *h* représente la valeur estimée restante pour aller à I) :

Nœud	A	B	C	D	E	F	G	H	I
h	10	4	5	10	10	3	3	4	0

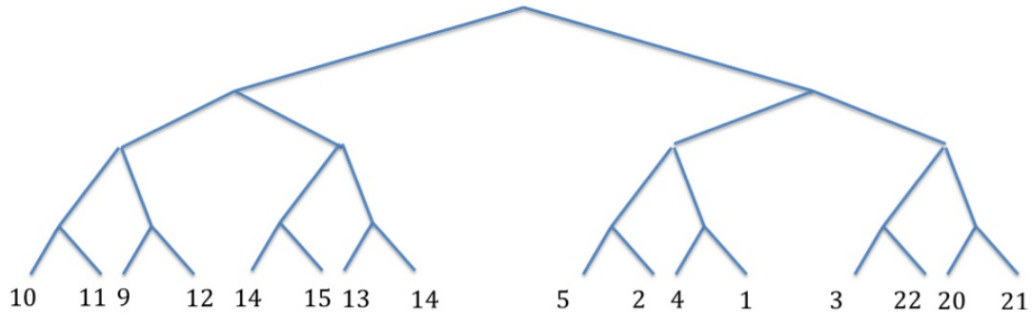
Questions

On recherche le plus court chemin permettant d'aller de A à I.

1. L'heuristique proposée est-elle admissible (minorante) ? Pourquoi ?
2. Appliquer l'algorithme de recherche glouton sur ce graphe.
3. Appliquer l'algorithme A*. La solution trouvée est-elle optimale ? Pourquoi ?

Exercice 3 : min-max et α-β

Soit l'arbre de jeu suivant, tel que le joueur Max joue en premier.



1. Appliquer l'algorithme min-max sur cet arbre.
2. Appliquer l'algorithme α-β sur cet arbre.

Problème : Mastermind en Prolog

L'objectif de ce problème est de faire un petit Mastermind en Prolog. Si besoin, on pourra réutiliser des fonctions citées précédemment dans le sujet mais non traitées.

À propos des prédicats demandés en Prolog.

On utilisera la notation suivante pour définir les prédicats à écrire : `p(+In, -Out, ?InOut)`

`+In` signifie que l'argument `In` doit être instancié dès l'appel du prédicat `p`, `-Out` signifie que l'argument `Out` sera instancié par l'appel du prédicat. `?` signifie que l'argument peut être instancié avant l'appel ou sera instancié par l'appel.

Attention, `+`, `-` et `?` ne sont pas utilisés par Prolog dans la définition des prédicats et ne doivent donc pas apparaître dans les réponses. Ils sont uniquement utilisés dans l'énoncé à des fins de désambiguïsation.

Rappel des règles du jeu

Mastermind est un jeu de stratégie à 2 joueurs dont le but est de trouver un code. Le premier joueur (le *codeur*) propose un code de N couleurs au deuxième joueur (le *décodeur*). À chaque proposition du décodeur, le codeur lui donne le nombre de couleurs bien placées (noté *BP*) et le nombre de couleurs mal placées (noté *MP*).

Codage utilisé

Les codes seront représentés par des listes d'entiers. On numérotera les couleurs de **1** à **M** et la taille des codes sera notée **N**.

Couleurs bien placées et mal placées

Prenons le cas où N = 4 et M = 5. Supposons que le code soit [1,2,3,4] et le décodeur propose [1,3,2,3]. Le nombre de bien placées sera de 1 (le premier 1) et le nombre de mal placées sera de 2 (le premier 3 et le 2). Une couleur ne peut pas intervenir deux fois dans un calcul de couleurs bien placées ou mal placées. Une couleur bien placée n'intervient plus pour le calcul des couleurs mal placées. Le tableau suivant apporte quelques exemples supplémentaires.

Code initial	Code proposé	BP	MP
[1,2,3,4]	[1,2,3,4]	4	0
[1,2,3,4]	[1,1,1,1]	1	0
[1,2,3,1]	[1,1,1,2]	1	2
[1,2,3,4]	[1,3,2,3]	1	2
[1,2,3,4]	[4,3,2,1]	0	4

Codage des Règles du jeu

On s'intéresse tout d'abord à calculer les valeurs *BP* et *MP*.

Question 1

1. Écrire un prédicat `nBienPlace(+Code1, +Code2, -BP)` qui, étant donnés 2 codes `Code1` et `Code2`, donne le nombre de couleurs bien placées.
2. Écrire un prédicat `gagne(+Code1, +Code2)` vérifiant que les codes 1 et 2 sont identiques. Utiliser `nBienPlace`. Définir pour cela un prédicat `longueur(+L, -N)` donnant la longueur `N` de la liste `L`.

```
| ?- nBienPlace([1,2,3,4], [1,2,3,5], BP).
BP = 3

| ?- gagne([1,2,3,4], [1,2,3,4]).
true
```

Question 2

1. Écrire un prédicat `element(+E, +L)` vérifiant que l'élément `E` appartient à la liste `L`.

2. Écrire un prédicat `enleve(+E, +L1, -L2)` qui construit la liste `L2` de telle sorte qu'elle soit identique à `L1` privée de la première occurrence de `E`.
3. Écrire un prédicat `enleveBP(+Code1, +Code2, -Code1Bis, -Code2Bis)` qui, étant donné `Code1` et `Code2`, est tel que `Code1Bis` (resp. `Code2Bis`) contiennent les éléments de `Code1` (resp. `Code2`) privé des éléments bien placés (communs) avec `Code2` (resp. `Code1`).
4. Écrire un prédicat `nMalPlaces(+Code1, +Code2, -MP)` qui étant donnés 2 codes donnent le nombre d'éléments mal placés (avec `MP` le nombre d'éléments mal placés). On pourra définir pour cela une fonction `nMalPlacesAux(+Code1, +Code2, -MP)` permettant de faire la même chose mais pour des listes ne contenant aucuns éléments bien placés communs.

```
| ?- element(2, [1,2,3,4]).
true

| ?- enleve(2, [1,2,3,4], L2).
L2 = [1,3,4]

| ?- enleveBP([1,2,3,4,5,6], [1,2,5,4,3,4], Code1Bis, Code2Bis).
Code1Bis = [3,5,6]
Code2Bis = [5,3,4]

| ?- nMalPlacesAux([1,2,3,4], [4,3,2,1], MP).
MP = 4

| ?- nMalPlaces([1,2,3,4], [1,3,2,3], MP).
MP = 2
```

Écriture d'un *codeur*

Question 3

1. Écrire une fonction `codeur(+M, +N, -Code)` qui produit aléatoirement un code de taille `N` basé sur `M` couleurs. Pour rappel, le prédicat `random(+Base, +Max, -Number)` unifie `Number` avec un nombre compris entre `Base` inclus et `Max` exclu.
2. Que fait la fonction `randomize` de l'exemple suivant ? Pourquoi l'utilise-t-on ?

```
| ?- randomize, codeur(4, 5, Code).
Code = [1,2,4,2,3]
```

Jouons…

Question 4

Écrire un prédicat `jouons(+M, +N, +Max)` permettant de jouer à Mastermind en tant que décodeur. Ce prédicat choisit aléatoirement un code de taille `N`, avec au plus `M` couleurs différentes puis demandera à un joueur humain un code. Ce dernier aura `Max` essais pour trouver le code. On pourra s'aider de prédicats auxiliaires pour simplifier le code. On supposera que l'utilisateur utilisera toujours des entrées valides.

```
| ?- jouons(5, 4, 6)
Il reste 6 coup(s).
Donner un code : [1,2,3,4].
BP: 0/MP: 2

Il reste 5 coup(s).
Donner un code : [2,1,5,1].
BP: 2/MP: 0

Il reste 4 coup(s).
Donner un code : [2,4,1,5].

BP: 3/MP: 0

Il reste 3 coup(s).
Donner un code : [2,4,5,5].
BP: 4/MP: 0
Gagné !!!
```