IREM de Clermont-Ferrand
Formation ISN
Algorithmique et programmation

03 mai 2012
Malika More
malika.more@u-clermont1.fr

Éléments de correction et remarques : Algorithmes du programme officiel d'ISN.

Si certaines questions ne vous paraissent pas claires, merci de ne pas hésiter à me poser des questions.

1 Additions en binaire

1. Effectuer l'addition en binaire ci-dessous :

$$\begin{array}{cccccccc} & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ + & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ \hline \end{array}$$

Éléments de réponse et remarques.

Pas de problème pour cette question, évidemment. Juste une remarque : dans mon esprit, il s'agissait simplement d'effectuer l'addition à la main, comme à l'école primaire, avec comme réponse 100010100. Peut-être influencés par le fait que j'avais écrit (oups...) le second opérande avec un zéro à gauche, ou alors obsédés par l'informatique, certains ont répondu 00010100 (et il reste une retenue de 1). Pourquoi pas !

2. Le *babinaire*, librement inspiré de la notation bibi-binaire de Bobby Lapointe.
On note BA pour 0 et BI pour 1. Ainsi, quatre se note $BIBABA$ et neuf se note $BIBABABI$. Écrivez un programme en Python3 effectuant l'addition de deux nombres en babinaire entrés au clavier (idéalement, directement avec des chaînes de caractères composées de BA et de BI, sans passer par le binaire ordinaire ou le décimal...)
Remarque : quand on a une chaîne de caractères mot, pour en extraire les lettres entre les indices i et j compris, en python on peut utiliser $extrait = mot[i : j + 1]$, sachant que la lettre d'indice 0 est la plus à gauche.

Éléments de réponse et remarques.

Dans cette question, j'attendais "simplement" une transposition d'un programme d'addition binaire utilisant les "chiffres" BA et BI au lieu de 0 et 1, comme dans le fichier `babinaire.py`. L'objectif était de voir si vous sauriez réécrire un algorithme bien connu dans un cadre un peu déstabilisant.
Par ailleurs, il est à noter que savoir mettre en œuvre un algorithme ne veut pas du tout dire savoir écrire (ou programmer) un algorithme. Selon moi, le programme ISN parle bien d'enseigner l'algorithme d'addition binaire, pas d'enseigner à faire des additions en base 2.

2 Dichotomie

1. Décrivez le principe d'un algorithme de recherche dichotomique d'une valeur dans un tableau trié. Comment la nature des éléments du tableau (des nombres, entiers ou flottants, des chaînes de caractères, ou encore des objets plus compliqués) intervient-elle ?

Éléments de réponse et remarques.

La première partie de la question n'a pas posé de problème, tant mieux puisqu'il vous faudra le faire pour vos élèves. Il est à noter que si cet algorithme est assez simple à décrire oralement ou sur le papier, et même à mettre en œuvre à la main, il est en revanche à mon avis plutôt difficile à programmer, car il faut être très précis et rigoureux sur les bornes des sous-tableaux. Il est peu probable que des élèves arrivent à s'en dépatouiller sans aide.

Concernant la deuxième partie de la question, cet algorithme s'applique sur n'importe quel type d'éléments susceptibles d'être ordonnés, que ça soit des nombres ou pas. L'ordre numérique, l'ordre alphabétique, ou encore autre chose, pourvu qu'on puisse ordonner (trier) les éléments. Certains d'entre vous ont ajouté des remarques judicieuses :

- dans certains cas, la comparaison de deux éléments peut prendre du temps (imaginez par exemple qu'il s'agisse de comparer deux fichiers texte de plusieurs milliers de pages, selon le nombre d'occurrences d'un mot donné) ;
- s'il s'agit de comparer des nombres réels représentés par des flottants, la question de l'égalité (ou de la précision) devra peut-être être gérée.

2. Si le tableau initial n'est pas trié, quels sont les avantages et les inconvénients respectifs des deux méthodes de recherche décrites ci-dessous ?
   (a) On commence par trier le tableau, puis on utilise une recherche dichotomique sur le tableau trié.
   (b) On parcourt linéairement toutes les cases du tableau. On s'arrête dès qu'on a trouvé l'élément cherché, mais si on arrive à la fin du tableau sans avoir trouvé cet élément, c'est qu'il n'est pas présent.

Éléments de réponse et remarques.

Question ouverte, et pas traitée par tout le monde.

Beaucoup de réponses ont tourné autour du temps d'exécution, tout en restant dans le vague, et souvent avec des conclusions erronées. Voici quelques précisions.

- Du point de vue de la complexité "théorique" (celle dont a parlé Vincent Limouzy), on peut donner la réponse suivante. Appelons n le nombre d'éléments des tableaux. Dans le pire des cas (càd si l'élément recherché n'est pas dans le tableau), la méthode (b) oblige à parcourir tout le tableau et à effectuer une comparaison pour chaque case. C'est donc un algorithme linéaire (càd en $\mathcal{O}(n)$). D'un autre côté, la méthode (a) est composée d'un tri et d'une recherche dichotomique. Les tris les plus rapides, comme le tri fusion, ont une complexité en $\mathcal{O}(n \log n)$ dans le pire des cas. Même si la méthode dichotomique utilisée ensuite est rapide (en $\mathcal{O}(\log n)$ dans le pire des cas), la succession des deux est en $\mathcal{O}(n \log n)$, ce qui est plus lent que $\mathcal{O}(n)$. Autrement dit, dans le pire des cas, la méthode (b) devrait être plus rapide que la méthode (a) sur de grands tableaux.
- Pour voir si cette approche théorique se vérifie expérimentalement, j'ai préparé un fichier Python intitulé `comparaisons.py`. Bien sûr, les temps de calcul expérimentaux ne correspondent pas à un "pire des cas", puisqu'on travaille sur des données aléatoires. Mais on voit bien que la théorie est confirmée. Bien sûr vous n'aviez pas le temps de le faire pendant l'examen, mais l'approche expérimentale est évidemment valide pour répondre à la question. (Attention, l'exécution est un peu longue...)

La moralité est évidemment qu'une méthode sophistiquée (ici la combinaison de deux algorithmes considérés comme rapides) n'est pas toujours préférable à une méthode simple (et l'inverse est vrai aussi).

On peut avoir de la chance dans les deux cas, et tomber directement sur l'élément cherché : s'il est au début du tableau pour la méthode (b) ou bien s'il est au milieu du tableau trié pour la méthode (a), et dans les deux cas, le pire des cas se produit lorsque l'élément recherché n'est pas dans le tableau.

D'autres paramètres que le temps d'exécution peuvent aussi jouer, comme l'ont fait remarquer plusieurs d'entre vous :

- Une fois que le tableau est trié, on peut aussi s'en servir pour autre chose. De même, si on doit effectuer de nombreuses recherches sur le même tableau, surtout dans le cas où on peut s'autoriser un temps de précalcul (pour le tri) qui n'a pas d'importance.
- Si la valeur cherchée n'est pas présente dans le tableau, la méthode (a) permet d'en trouver un encadrement par des éléments du tableau.
- Si la valeur cherchée est présente plusieurs fois dans le tableau, les différentes occurrences se retrouveront les unes à côté des autres si on utilise la méthode (a).
- S'il faut programmer à partir de rien, la méthode (b) est bien plus simple que la méthode (a), on risque moins de se tromper. D'un autre côté, si le tri et la recherche dichotomique sont des "briques" déjà prêtes, la combinaison des deux sera très simple.
- Si les éléments du tableau ne peuvent pas être ordonnés, évidemment on ne peut utiliser que la méthode (b), qui est donc plus générale que la méthode (a).

3 Tris

1. Parmi les trois exécutions d'algorithmes de tris ci-dessous, laquelle (lesquelles) corresponde(nt) à un tri par sélection du tableau 14 ;12 ;15 ;11 ;13 (on convient que l'indice de la première case du tableau est 0) ? Justifiez vos réponses.

(a) *Première exécution*

- Je regarde les deux premiers éléments 14 et 12 : 12 est plus petit que 14, donc je les permute : 12 ;14 ;15 ;11 ;13
- J'avance d'une case, je regarde 14 et 15 : ils sont dans le bon ordre, donc je ne fais rien
- J'avance d'une case, je regarde 15 et 11 : 11 est plus petit que 15, donc je les permute : 12 ;14 ;11 ;15 ;13
- J'avance d'une case, je regarde 15 et 13 : 13 est plus petit que 15, donc je les permute : 12 ;14 ;11 ;13 ;15
- Le dernier élément est à sa place définitive, je n'y touche plus
- Je reviens au début du tableau
- Je regarde les deux premiers éléments 12 et 14 : ils sont dans le bon ordre, donc je ne fais rien
- J'avance d'une case, je regarde 14 et 11 : 11 est plus petit que 14, donc je les permute : 12 ;11 ;14 ;13 ;15
- J'avance d'une case, je regarde 14 et 13 : 13 est plus petit que 14, donc je les permute : 12 ;11 ;13 ;14 ;15
- Les deux derniers éléments sont à leur place définitive, je n'y touche plus
- Je reviens au début du tableau
- Je regarde les deux premiers élément 12 et 11 : 11 est plus petit que 12, donc je les permute : 11 ;12 ;13 ;14 ;15
- J'avance d'une case, je regarde 12 et 13 : ils sont dans le bon ordre, donc je ne fais rien
- Les trois derniers éléments sont à leur place définitive, je n'y touche plus
- Je reviens au début du tableau
- Je regarde les deux premiers élément 11 et 12 : ils sont dans le bon ordre, donc je ne fais rien
- Les quatre derniers éléments sont à leur place définitive, je n'y touche plus
- Je reviens au début du tableau et j'ai terminé : le tableau trié est 11 ;12 ;13 ;14 ;15

(b) *Deuxième exécution*

- Le premier élément du tableau est 14, je garde en mémoire son indice 0
- L'élément suivant du tableau est 12, il est plus petit que l'élément d'indice 0 donc il ne m'intéresse pas
- L'élement suivant du tableau est 15, il est plus grand que l'élément d'indice 0, donc j'oublie 0 et je garde en mémoire l'indice 2 de l'élement 15
- L'élement suivant du tableau est 11, il est plus petit que l'élément d'indice 2 donc il ne m'intéresse pas
- L'élement suivant du tableau est 11, il est plus petit que l'élément d'indice 2 donc il ne m'intéresse pas
- L'élement suivant du tableau est 13, il est plus petit que l'élément d'indice 2 donc il ne m'intéresse pas
- Je suis arrivée à la fin du tableau, le plus grand élément est celui qui a l'indice 2 : je le positionne en dernière position en le permutant avec le dernier élément : 14 ;12 ;13 ;11 ;15
- Le dernier élément est à sa place définitive, je n'y touche plus
- Je reviens au début du tableau
- Le premier élément du tableau est 14, je garde en mémoire son indice 0
- L'élément suivant du tableau est 12, il est plus petit que l'élément d'indice 0 donc il ne m'intéresse pas
- L'élément suivant du tableau est 13, il est plus petit que l'élément d'indice 0 donc il ne m'intéresse pas
- L'élément suivant du tableau est 11, il est plus petit que l'élément d'indice 0 donc il ne m'intéresse pas
- Je suis arrivée à la fin de la partie restante du tableau, le plus grand élément est celui qui a l'indice 0 : je le positionne en avant-dernière position en le permutant avec l'avant-dernier élément : 11 ;12 ;13 ;14 ;15
- Les deux derniers éléments sont à leur place définitive, je n'y touche plus
- Je reviens au début du tableau
- Le premier élément du tableau est 11, je garde en mémoire son indice 0
- L'élement suivant du tableau est 12, il est plus grand que l'élément d'indice 0, donc j'oublie 0 et je garde en mémoire l'indice 1 de l'élement 12
- L'élement suivant du tableau est 13, il est plus grand que l'élément d'indice 1, donc j'oublie 1 et je garde en mémoire l'indice 2 de l'élement 13
- Je suis arrivée à la fin de la partie restante du tableau, le plus grand élément est celui qui se trouve à l'indice 2 : il est déjà à la bonne place, donc je ne fais rien : 11 ;12 ;13 ;14 ;15
- Les trois derniers éléments sont à leur place définitive, je n'y touche plus
- Je reviens au début du tableau
- Le premier élément du tableau est 11, je garde en mémoire son indice 0
- L'élement suivant du tableau est 12, il est plus grand que l'élément d'indice 0, donc j'oublie 0 et je garde en mémoire l'indice 1 de l'élement 12
- Je suis arrivée à la fin de la partie restante du tableau, le plus grand élément est celui qui a l'indice 1 : il est déjà à la bonne place, donc je ne fais rien : 11 ;12 ;13 ;14 ;15
- Les quatre derniers éléments sont à leur place définitive, je n'y touche plus
- Je reviens au début du tableau
- Le premier élément du tableau est 11, je garde en mémoire son indice 0
- Je suis arrivée à la fin de la partie restante du tableau, le plus grand élément est celui qui a l'indice 0 :

il est déjà à la bonne place, donc je ne fais rien : 11 ;12 ;13 ;14 ;15
- Les cinq éléments sont à leur place définitive, donc j'ai terminé : le tableau trié est 11 ;12 ;13 ;14 ;15

(c) *Troisième exécution*
- L'élément d'indice 1 du tableau est 12 : je mémorise l'indice 1
- Je remarque que l'élément d'indice 1 est plus petit que l'élément juste avant, qui est 14, donc je les permute : 12 ;14 ;15 ;11 ;13
- Je suis arrivée au début du tableau
- Je reviens à l'indice que j'avais mémorisé, à savoir 1
- Je passe à l'élément suivant du tableau : c'est 15, j'oublie l'indice 1 et je mémorise l'indice 2
- Je remarque que l'élément d'indice 2 est plus grand que l'élément juste avant, qui est 14, donc je ne fais rien d'autre
- Je passe à l'élément suivant du tableau : c'est 11, j'oublie l'indice 2 et je mémorise l'indice 3
- Je remarque que l'élément d'indice 3, qui est 11, est plus petit que l'élément juste avant, qui est 14 donc je les permute : 12 ;14 ;11 ;15 ;13
- Je remarque que 11 est plus petit que l'élément juste avant, donc je les permute : 12 ;11 ;14 ;15 ;13
- Je remarque que 11 est plus petit que l'élément juste avant, donc je les permute : 11 ;12 ;14 ;15 ;13
- Je suis arrivée au début du tableau
- Je reviens à l'indice que j'avais mémorisé, à savoir 3
- Je passe à l'élément suivant du tableau : c'est 13, j'oublie l'indice 3 et je mémorise l'indice 4
- Je remarque que l'élément d'indice 4, qui est 13 est plus petit que l'élément juste avant, qui est 15 donc je les permute : 11 ;12 ;14 ;13 ;15
- Je remarque que 13 est plus petit que l'élément juste avant, qui est 14, donc je les permute : 11 ;12 ;13 ;14 ;15
- Je remarque que 13 est plus grand que l'élément juste avant, qui est 12, donc je ne fais rien d'autre
- Je reviens à l'indice que j'avais mémorisé, à savoir 4
- C'est la fin du tableau donc j'ai terminé : le tableau trié est 11 ;12 ;13 ;14 ;15

Éléments de réponse et remarques.

La seule exécution qui correspond à un tri par sélection est la deuxième, comme tout le monde l'a remarqué. Tant mieux, puisqu'il vous arrivera sûrement d'avoir à dire à vos élèves s'ils ont bien effectué un tri par sélection.

La première exécution est un tri à bulle, et la troisième un tri par insertion (celui du joueur de cartes : vous avez une main de cartes déjà triée, on vous donne une nouvelle carte et vous l'insérez à sa place). Mais pour ce dernier, ma description n'était peut-être pas très claire, puisque peu d'entre vous l'ont reconnu !

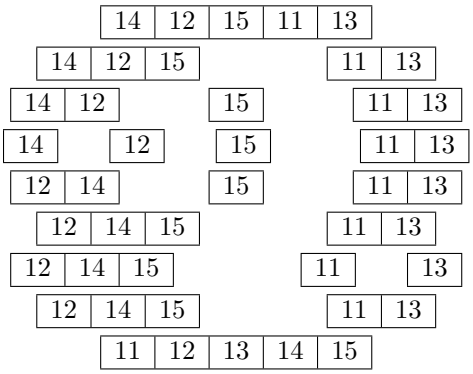
2. On donne les trois fonctions en python ci-dessous qui permettent de réaliser un tri fusion par l'appel de la fonction `trifusion` sur le tableau à trier. Décrivez en détails (restez raisonnable !) l'exécution de la commande `trifusion([14;12;15;11;13])`. En particulier, faites apparaître les différents appels récursifs, les valeurs des paramètres début, milieu et fin et l'évolution de l'état du tableau.

```
def interclassement(tabentree,temp,debut,milieu,fin):
    i=debut
    j=milieu+1
    for k in range(debut,fin+1):
        if (j>fin or (i<=milieu and tabentree[i]<tabentree[j])):
            temp[k]=tabentree[i]
            i=i+1
        else:
            temp[k]=tabentree[j]
            j=j+1
    for k in range(debut, fin+1):
        tabentree[k]=temp[k]


def fonctiontrifusion(tabentree,temp,debut,fin):
    if debut<fin:
        milieu=math.floor((debut+fin)/2)
        fonctiontrifusion(tabentree,temp,debut,milieu)
```

```
        fonctiontrifusion(tabentree,temp,milieu+1,fin)
        interclassement(tabentree,temp,debut,milieu,fin)

def trifusion(tabentree):
    n=len(tabentree)
    temp=[0 for i in range(n)]
    fonctiontrifusion(tabentree,temp,0,n-1)
```

Éléments de réponse et remarques.

D'un point de vue pédagogique, il est important de réfléchir à la façon dont vous allez expliquer ce point délicat à vos élèves, au delà du schéma général :



qui a été dans l'ensemble bien rendu.

En particulier, il me semble important de bien faire apparaître les différents appels récursifs. On pourrait imaginer quelque chose comme ça (j'ai enlevé le tableau "temp" qui n'a qu'un rôle technique) :

trifusion([14 ;12 ;15 ;11 ;13])

- fonctiontrifusion([14 ;12 ;15 ;11 ;13] ;0 ;4) milieu = 2, on plonge
  - ⋆ fonctiontrifusion([14 ;12 ;15 ;11 ;13] ;0 ;2) milieu = 1, on plonge
    - ◦ fonctiontrifusion([14 ;12 ;15 ;11 ;13] ;0 ;1) milieu = 0, on plonge
      - ∗ fonctiontrifusion([14 ;12 ;15 ;11 ;13] ;0 ;0)X , on ne fait rien, on passe à l'instruction suivante
      - ∗ fonctiontrifusion([14 ;12 ;15 ;11 ;13],1 ;1)X , on ne fait rien, on passe à l'instruction suivante
      - ∗ interclassement des éléments d'indices 0 et 1 , le tableau devient [12 ;14 ;15 ;11 ;13] et on remonte
    - ◦ fin du fonctiontrifusion([12 ;14 ;15 ;11 ;13] ;0 ;1), on passe à l'instruction suivante
    - ◦ fonctiontrifusion([12 ;14 ;15 ;11 ;13] ;2 ;2)X , on ne fait rien, on passe à l'instruction suivante
    - ◦ interclassement des éléments d'indices 0 à 2 , le tableau devient [12 ;14 ;15 ;11 ;13] et on remonte
  - ⋆ fin du fonctiontrifusion([12 ;14 ;15 ;11 ;13] ;0 ;2), on passe à l'instruction suivante
  - ⋆ fonctiontrifusion([12 ;14 ;15 ;11 ;13] ;3 ;4) milieu = 3 , on plonge
    - ◦ fonctiontrifusion([12 ;14 ;15 ;11 ;13] ;3 ;3)X , on ne fait rien, on passe à l'instruction suivante
    - ◦ fonctiontrifusion([12 ;14 ;15 ;11 ;13] ;4 ;4)X , on ne fait rien, on passe à l'instruction suivante
    - ◦ interclassement des éléments d'indices 3 à 4 , le tableau devient [12 ;14 ;15 ;11 ;13] et on remonte
  - ⋆ fin du fonctiontrifusion([12 ;14 ;15 ;11 ;13] ;3 ;4), on passe à l'instruction suivante
  - ⋆ interclassement des éléments d'indices 0 à 4 , le tableau devient [11 ;12 ;13 ;14 ;15] et on remonte
- fin de fonctiontrifusion([11 ;12 ;13 ;14 ;15] ;0 ;4)

fin du trifusion([11 ;12 ;13 ;14 ;15]), on a fini

4 Parcours

1. Citer quelques utilisations des parcours en largeur ou en profondeur des graphes ou des arbres.

Éléments de réponse et remarques.

Les deux types de parcours permettent une exploration systématique de tous les sommets accessibles à partir d'un sommet donné dans un graphe, et la construction d'un arbre de parcours représentant les arêtes visitées. Quoi que représente le graphe, et quelle que soit l'utilisation qu'on veut en faire, être capable de l'explorer paraît la moindre des choses ! Ensuite, le choix d'une méthode ou d'une autre pour cette exploration dépend plus précisément de l'objectif.

Quelques utilisations classiques du parcours en largeur :

- Recherche de plus courts chemins dans des graphes
- Lignes de niveau
- Exploration d'un arbre par niveaux

Quelques utilisations classiques du parcours en profondeur :

- En algorithmique, le tri topologique, la recherche de composantes (fortement) connexes, le calcul des blocs.
- Ordonnancement de tâches, par exemple méthode PERT, diagramme de Gantt
- Exploration d'un arbre par branche (parcours préfixe, infixe ou postfixe) : évaluation récursive des expressions arithmétiques, algorithme minimax

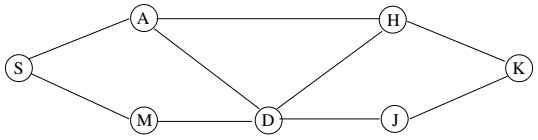
Les algorithmes de parcours sont considérés comme les plus importants de l'algorithmique des graphes, elle-même une des parties les plus importantes de l'algorithmique, tant théorique que pratique...

2. **Attention, plus difficile !**

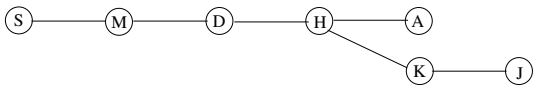
Le parcours d'un graphe à partir d'un sommet donné permet (entre autre) de construire un *arbre de parcours*. La racine de cet arbre est le sommet de départ, et un sommet y est le fils d'un sommet x lorsque y est visité à partir de x pendant le parcours. Cette construction peut se faire quel que soit le parcours effectué : en largeur, en profondeur ou ni l'un ni l'autre.

On considère le graphe G ci-dessous, et les arbres de parcours T_1, T_2, T_3 et T_4. Quels sont le ou les arbre(s) correspondant à un parcours en profondeur, à un parcours en largeur ou à un parcours quelconque du graphe G à partir du sommet S ? Justifiez vos réponses.

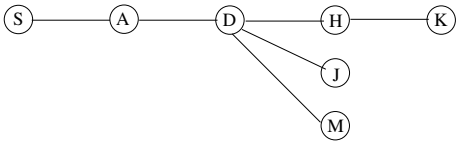
Remarque : Les arbres sont orientés de gauche à droite.



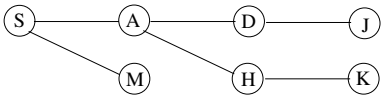
Le graphe G



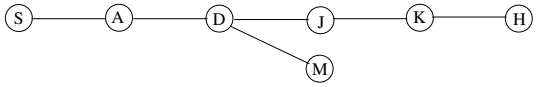
L'arbre T_1



L'arbre T_2



L'arbre T_3



L'arbre T_4

Éléments de réponse et remarques.

Les arbres T_1 et T_4 correspondent à des parcours en profondeur, l'arbre T_3 à un parcours en largeur, et l'arbre T_2 ni l'un ni l'autre. Cette question a posé plutôt moins de difficultés que ce que je pensais, tant mieux ! C'est (encore) un exercice classique de prof corrigeant une copie...

Réponses détaillées :

T_1 On part de S, puis on avance tant qu'on peut. À chaque carrefour, on choisit une des arêtes (pas forcément dans l'odre alphabétique). Quand on arrive en A, on ne peut plus avancer. On recule alors jusqu'à la première intersection permettant de prendre un nouveau chemin (c'est H), et on recommence à avancer. Quand on est encore bloqué, en J, on recule de nouveau, mais cette fois il n'y a plus de nouvelle branche à explorer, on retourne en S est le parcours est terminé.

C'est le principe d'un parcours en profondeur.

T_2 Ce n'est par un parcours en largeur car alors S aurait deux fils, A et M. Ce n'est pas non plus un parcours en profondeur, car arrivé en K, on pourrait continuer d'avancer en J.

Le parcours représenté par cet arbre n'est ni un parcours en largeur, ni un parcours en profondeur.

T_3 À partir de la racine S, on atteint ses deux voisins A et M. Puis à partir de A, on atteint ses deux voisins non encore explorés D et H. Le sommet M n'a plus de voisin qui n'a pas été exploré. Ensuite, on reprend les sommets à distance 2 de la racine : à partir de D, on peut aller en J seulement et à partir de H, on peut aller en K. Pour finir, on s'occupe des sommets à distance 3 de S : les sommets J puis K n'ont aucun voisin qui n'a pas été exploré. Le parcours est terminé.

C'est le principe d'un parcours en largeur.

T_4 C'est encore un parcours en profondeur, mais avec des choix différents aux carrefours.