Algorithmique et Analyse d'Algorithmes

L3 Info

Cours 6 : Structures de données arborescentes

Benjamin Wack

2021 – 2022

La dernière fois

- Type Abstrait de Données
- Pile, File, File à Priorité
- Algorithmes de bon parenthésage

Aujourd'hui

- Structures arborescentes
- Type Abstrait Arbre Binaires (et autres)
- Partition Binaire de l'Espace

Plan

Les arbres omniprésents en informatique

Dans les données

- Système de fichiers (répertoires et fichiers)
- Expressions algébriques
- Document structuré (html, xml, tex...)

... et ailleurs (généalogie, organigrammes...)

Dans les algorithmes

- Arbre de décision
- Compression d'images, de textes
- Recherche de motifs

Arbre explicite ou sous-jacent

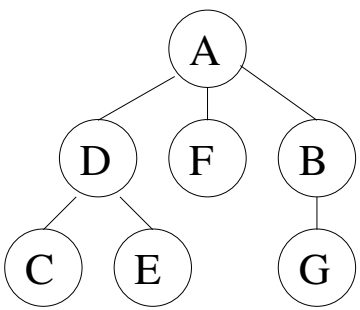
Vocabulaire général

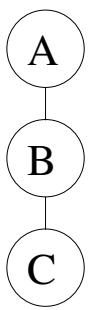
- Arbre en tant que graphe particulier : peu adapté à des manipulations algorithmiques (cf fin du semestre).
- On privilégie un sommet de référence :

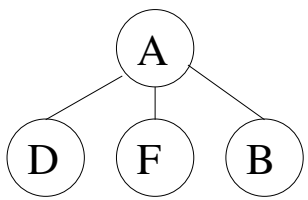
Arbre **enraciné**

Un arbre enraciné est un nœud constitué :

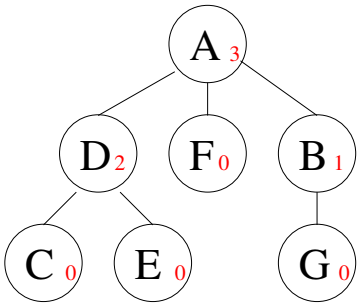
- d'une *étiquette* (ou clé)
- et de 0, 1 ou plusieurs *sous-arbres* enracinés (eux-mêmes constitués de nœuds etc.)







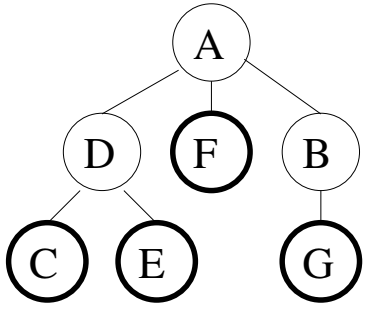
Vocabulaire général (2)



On appelle :

arité d'un nœud le nombre de sous-arbres qu'il porte

Vocabulaire général (2)

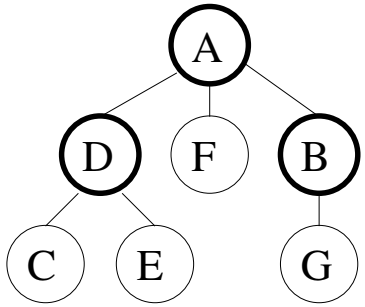


On appelle :

arité d'un nœud le nombre de sous-arbres qu'il porte

feuille un nœud d'arité 0

Vocabulaire général (2)



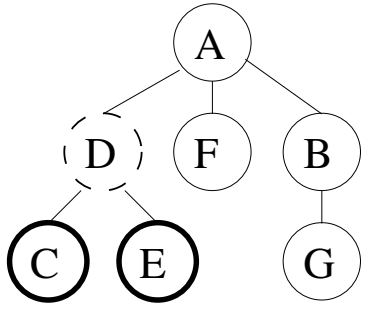
On appelle :

arité d'un nœud le nombre de sous-arbres qu'il porte

feuille un nœud d'arité 0

nœud interne un nœud qui n'est pas une feuille

Vocabulaire général (2)



On appelle :

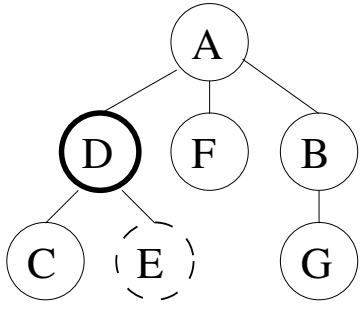
arité d'un nœud le nombre de sous-arbres qu'il porte

feuille un nœud d'arité 0

nœud interne un nœud qui n'est pas une feuille

fils d'un nœud, un des sous-arbres de ce nœud

Vocabulaire général (2)



On appelle :

arité d'un nœud le nombre de sous-arbres qu'il porte

feuille un nœud d'arité 0

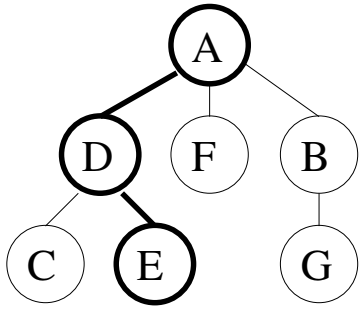
nœud interne un nœud qui n'est pas une feuille

fils d'un nœud, un des sous-arbres de ce nœud

père ou parent d'un nœud le nœud (s'il existe) dont il est un fils

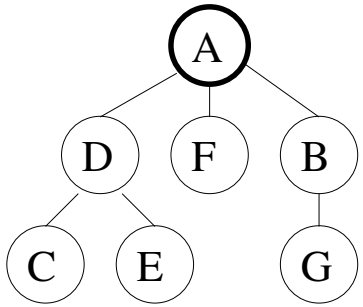
descendants, ancêtres...

Vocabulaire général (3)



chemin une suite de nœuds dont chacun est le fils du précédent

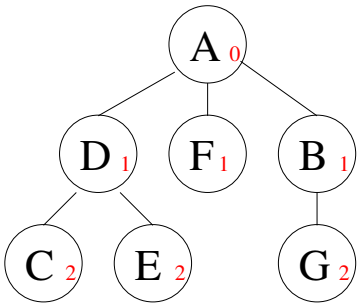
Vocabulaire général (3)



chemin une suite de nœuds dont chacun est le fils du précédent

racine d'un arbre **le** nœud qui n'a pas de parent

Vocabulaire général (3)

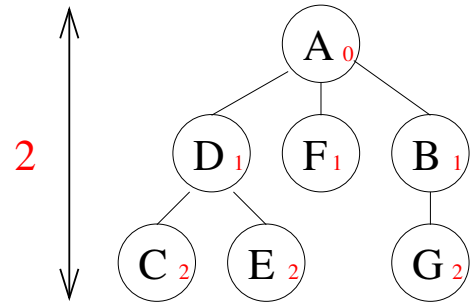


chemin une suite de nœuds dont chacun est le fils du précédent

racine d'un arbre **le** nœud qui n'a pas de parent

profondeur ou niveau d'un nœud la longueur du chemin le reliant à la racine (pour la racine c'est donc 0)

Vocabulaire général (3)



chemin une suite de nœuds dont chacun est le fils du précédent

racine d'un arbre **le** nœud qui n'a pas de parent

profondeur ou niveau d'un nœud la longueur du chemin le reliant à la racine (pour la racine c'est donc 0)

hauteur d'un arbre la profondeur maximale de ses nœuds

En particulier :

- un arbre à un seul nœud est de hauteur 0
- l'arbre vide est de hauteur -1

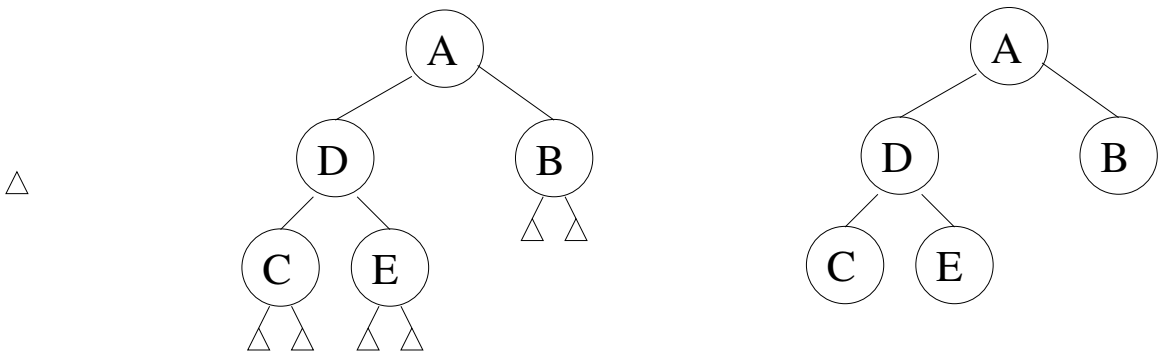
Arbre binaire

Arbre binaire

Un arbre **binaire** est :

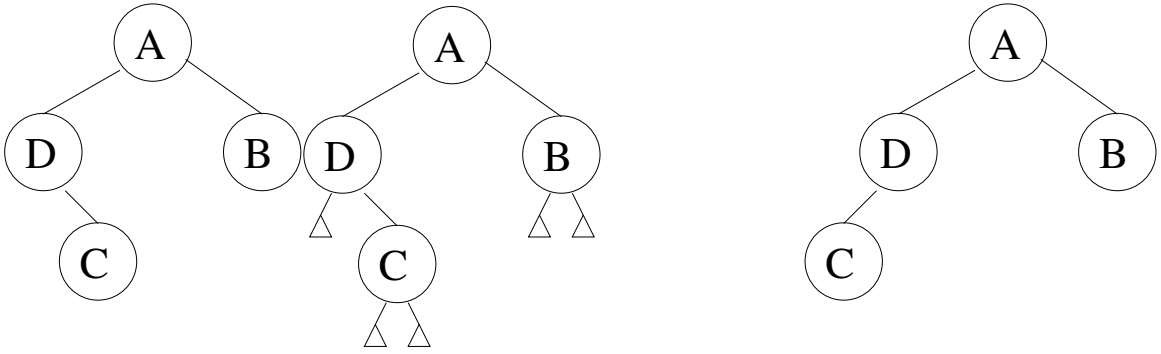
- soit l'arbre vide ;
- soit un nœud constitué d'une *étiquette* et de 2 *sous-arbres* binaires (gauche et droit).

En particulier tous ses nœuds sont d'arité 2.



On ne représente en général pas les sous-arbres vides.
On appelera **feuille** tout nœud ayant deux sous-arbres vides.

Arbre binaire (2)



Binaire VS complet

Tous les nœuds sont binaires, mais un nœud peut avoir un ou deux sous-arbres vides (nœud **incomplet**).

On distinguera le fils **gauche** du fils **droit** ; quand un nœud n'a qu'un seul fils non vide on doit « choisir » si c'est un fils gauche ou droit.

Pourquoi se focaliser sur les arbres binaires ?

- Toute décision peut être ramenée à des choix binaires
- La plupart des opérations prennent deux arguments
- Il est possible de simuler un arbre n-aire avec un arbre binaire.

Un peu de dénombrement

Nombre de nœuds d'un arbre binaire de hauteur h

Minimum : $h + 1$
Maximum : $1 + 2 + 4 + \ldots + 2^h = 2^{h+1} - 1$

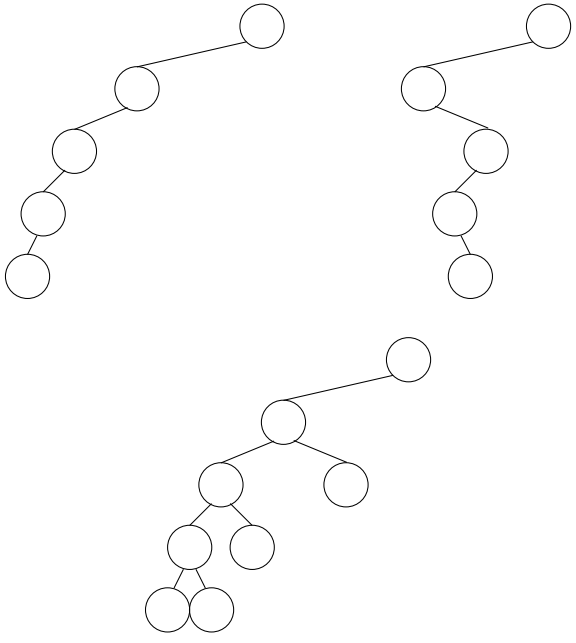
Hauteur d'un arbre binaire ayant n nœuds

Minimum : $\lfloor \log_2(n) \rfloor$
Maximum : $n - 1$

Nombre de feuilles d'un arbre binaire ayant n nœuds

Minimum : 1
Maximum : $\lceil \frac{n}{2} \rceil$

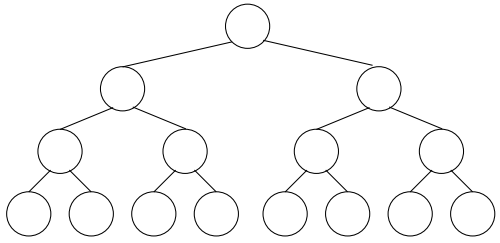
Cas particuliers à connaître



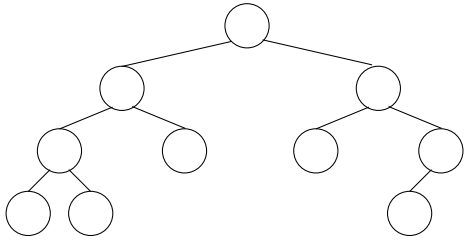
Peigne et arbres dégénérés

- $h = \mathcal{O}(n)$
- $n = \mathcal{O}(h)$

$\simeq$ équivalent à une structure linéaire



Arbre complet



Arbre équilibré

- $h = \mathcal{O}(\log n)$
- $n = \mathcal{O}(2^h)$
- $f = \mathcal{O}(n)$

Le poids d'un « vrai » arbre est dans ses feuilles.

Spécification

Opérations

$$\begin{aligned}
\text{ArbreVide} &: \textit{void} \rightarrow \textit{Arbre} \\
\text{Noeud} &: \textit{Arbre} \times \textit{Element} \times \textit{Arbre} \rightarrow \textit{Arbre} \\
\text{EstArbreVide} &: \textit{Arbre} \rightarrow \textit{bool} \\
\text{FilsGauche} &: \textit{Arbre} \rightarrow \textit{Arbre} \\
\text{FilsDroit} &: \textit{Arbre} \rightarrow \textit{Arbre} \\
\text{Clé} &: \textit{Arbre} \rightarrow \textit{Element}
\end{aligned}$$

Préconditions

$$\begin{aligned}
\text{FilsGauche}(a) &: \textit{non}\ \text{EstArbreVide}(a) \\
\text{FilsDroit}(a) &: \textit{non}\ \text{EstArbreVide}(a) \\
\text{Clé}(a) &: \textit{non}\ \text{EstArbreVide}(a)
\end{aligned}$$

Axiomes

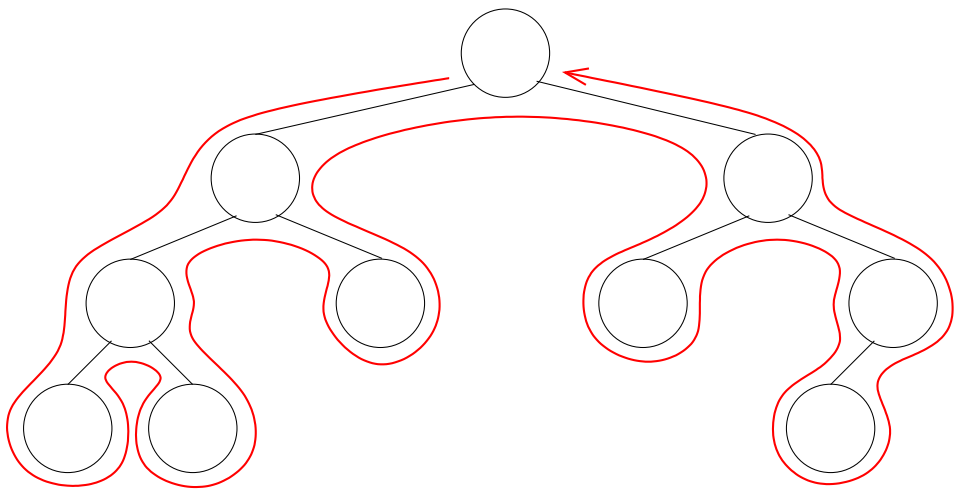
$$\begin{aligned}
\text{EstArbreVide}(\text{ArbreVide}()) &= \text{vrai} \\
\text{EstArbreVide}(\text{Noeud}(g, r, d)) &= \text{faux} \\
\text{FilsGauche}(\text{Noeud}(g, r, d)) &= g \\
\text{FilsDroit}(\text{Noeud}(g, r, d)) &= d \\
\text{Clé}(\text{Noeud}(g, r, d)) &= r
\end{aligned}$$

Parcours d'arbre

Beaucoup d'algorithmes se résument à traiter individuellement chaque nœud de l'arbre (pour en faire la liste, pour effectuer un calcul...).

On se donne donc une opération **Traiter** à appliquer à chaque nœud, et on écrit des algorithmes génériques de parcours utilisant cette opération.

Structure intimement récursive et donc liée à ce type d'algorithme... mais pas que.



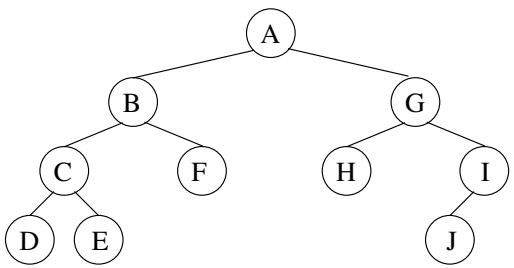
Question : dans quel ordre précisément traite-t-on les nœuds ?

Parcours en profondeur d'abord

PARCOURS_PROFONDEUR_PREFIXE (a)
Données : un arbre binaire a
if *non EstArbreVide(a)*
  Traiter(Clé(a))
  PARCOURS_PROFONDEUR_PREFIXE (FilsGauche(a))

  PARCOURS_PROFONDEUR_PREFIXE (FilsDroit(a))

- préfixe : traiter la racine d'abord



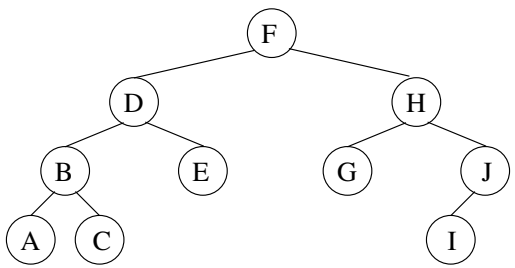
parcours préfixe

Parcours en profondeur d'abord

PARCOURS_PROFONDEUR_INFIXE (a)
Données : un arbre binaire a
if *non EstArbreVide(a)*

PARCOURS_PROFONDEUR_INFIXE (FilsGauche(a))
Traiter(Clé(a))
PARCOURS_PROFONDEUR_INFIXE (FilsDroit(a))

- infixe (ou symétrique) : traiter la racine entre les sous-arbres



parcours infixe

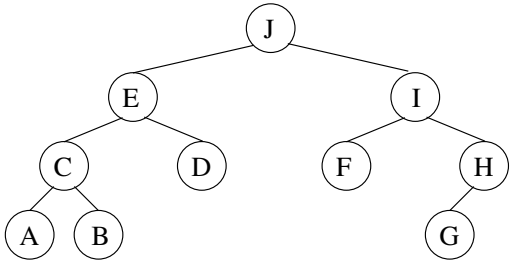
Parcours en profondeur d'abord

PARCOURS_PROFONDEUR_SUFFIXE (a)
Données : un arbre binaire a
if *non EstArbreVide(a)*

PARCOURS_PROFONDEUR_SUFFIXE (FilsGauche(a))

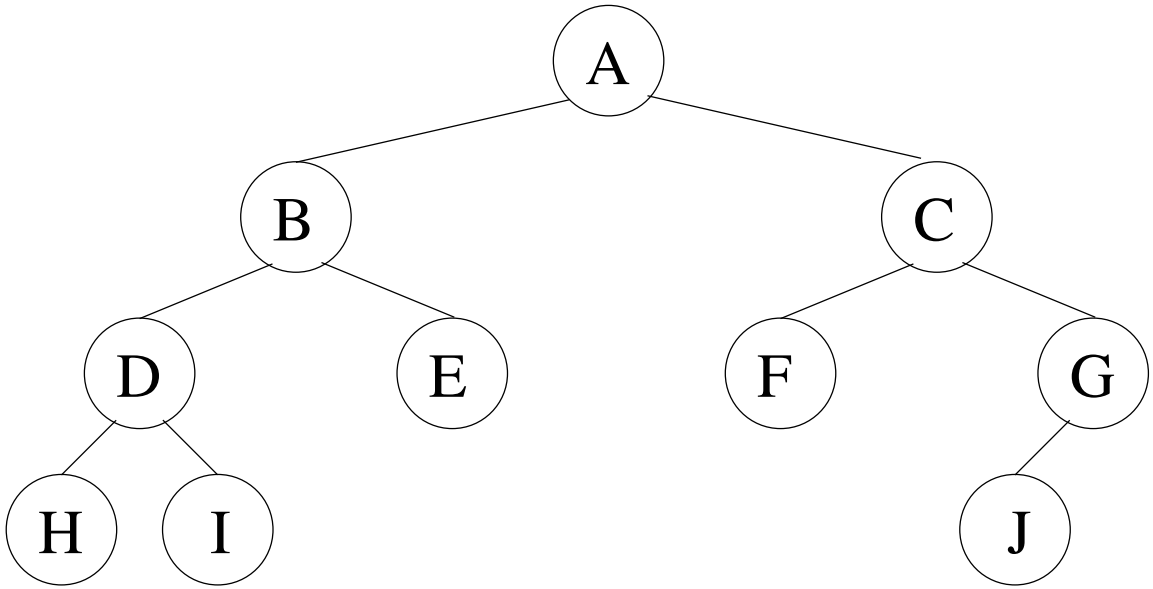
PARCOURS_PROFONDEUR_SUFFIXE (FilsDroit(a))
Traiter(Clé(a))

- suffixe (ou postfixe) : traiter la racine en dernier



parcours suffixe

Parcours en largeur d'abord



Besoin de « sauter » d'un sous-arbre à un autre : la récursivité n'est plus le bon outil.

⇒ On écrit un algorithme itératif et on utilise une file.

Parcours en largeur d'abord : l'algorithme

```
PARCOURS_LARGEUR( a )
F = FileVide()
Enfiler( a, F )
while non EstVide(F)
    n = Tête(F)
    F = Défiler(F)
    if non EstArbreVide(n)
        F = Enfiler( FilsGauche(n), F )
        F = Enfiler( FilsDroit(n), F )
        Traiter( n )
```

Remarque

L'utilisation similaire d'une *pile* permet d'écrire les parcours en profondeur sous forme d'algorithmes *itératifs*.

Feuilles étiquetées

Principe : remplacer les arbres vides par du contenu

On choisit deux types (en général différents) :

- pour les étiquettes des nœuds
- pour les étiquettes des feuilles

Les constructeurs deviennent :

- Feuille : $\mathit{EtiqFeuille} \rightarrow \mathit{Arbre}$
- Noeud : $\mathit{Arbre} \times \mathit{EtiqNoeud} \times \mathit{Arbre} \rightarrow \mathit{Arbre}$

Remarque : il n'y a alors plus de nœud unaire.

Applications

- Arbre d'expression algébrique
- Plus généralement arbre syntaxique

Arbre n-aire

Chaque nœud peut avoir un nombre n de fils :

- n fixé : Noeud(e, T) où T est un tableau de n fils
  (et on termine par des arbres vides)

  OU
- n variable : Noeud(e, L) où L est une liste de fils
  (et les feuilles sont les nœuds pour lesquels L est vide)

Intérêts

- Même hiérarchie mais structure plus compacte
- Permet de prendre en compte des opérateurs *n-aires* ou *variadiques*
- Avec une liste il y a « toujours de la place » sous un nœud donné

Applications

- Arborescence de répertoires
- Dictionnaire, T9...

Branches étiquetées

On choisit deux types (en général différents) :

- pour les étiquettes des nœuds
- pour les étiquettes des branches

Le constructeur Nœud devient :

Noeud : $Arbre \times EtiqBrche \times EtiqNoeud \times EtiqBrche \times Arbre \rightarrow Arbre$

Alors pour le parcours de l'arbre, à chaque nœud :

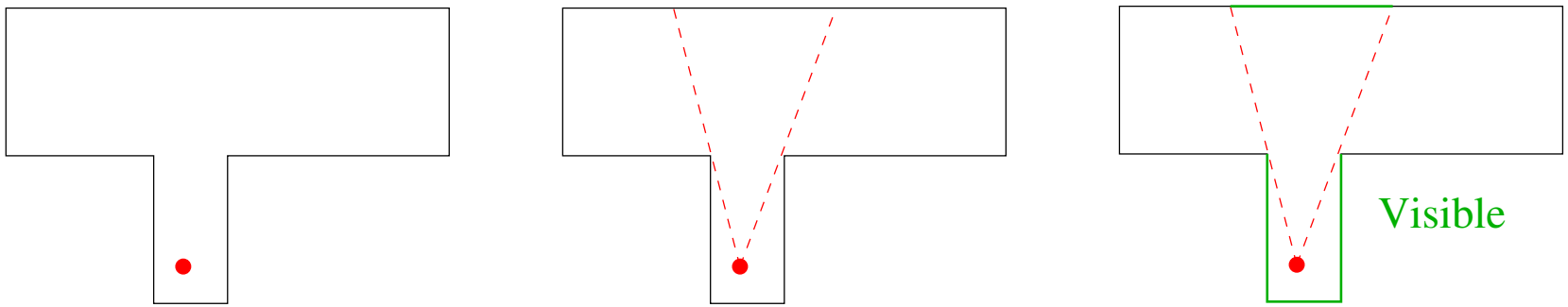
- on « pose une question »
- la réponse doit correspondre à une étiquette de branche
- on poursuit dans le fils correspondant

Application

- Arbre de décision, « automate » sans boucle

Doom

On considère un univers virtuel décrit comme un ensemble de polygones. Le problème est d'afficher ou non les polygones selon leur visibilité du point de vue du joueur.



L'algorithme du peintre

1. Trier les polygones par ordre de distance à l'observateur
2. Les afficher par distance décroissante

Naturel mais inefficace :

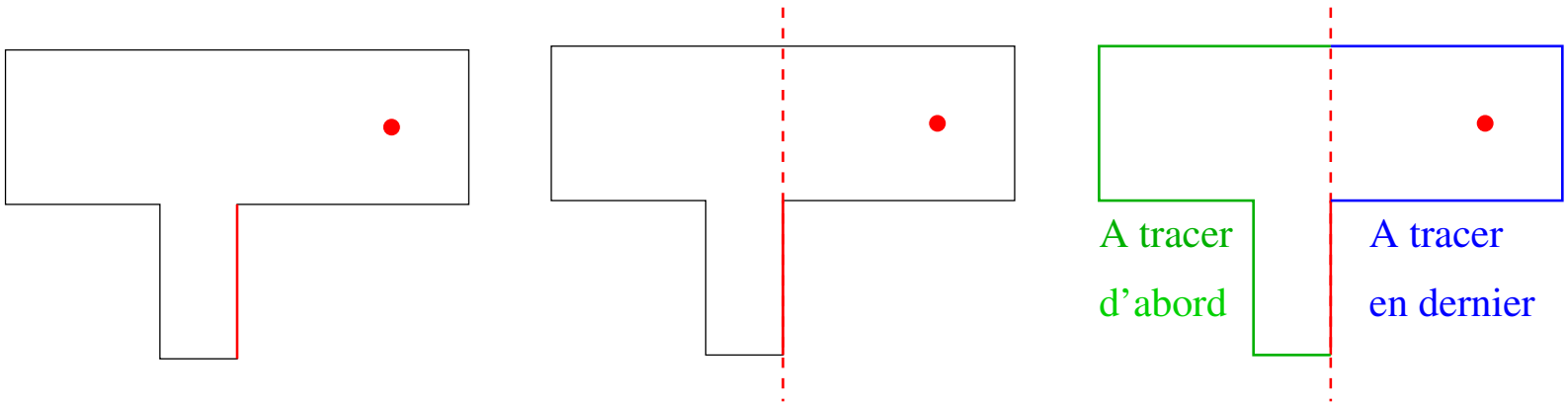
- tri préalable coûteux
- inadapté en cas d'intersection

L'idée

Formalisation

Chaque polygone coupe l'espace en deux régions (devant/derrière).
(en réalité le plan qui contient ce polygone).

L'univers est donné sous forme d'une liste de polygones.



On crée un arbre **binaire** étiqueté par des listes de polygones : le Binary Space Partitioning Tree (BSPT).

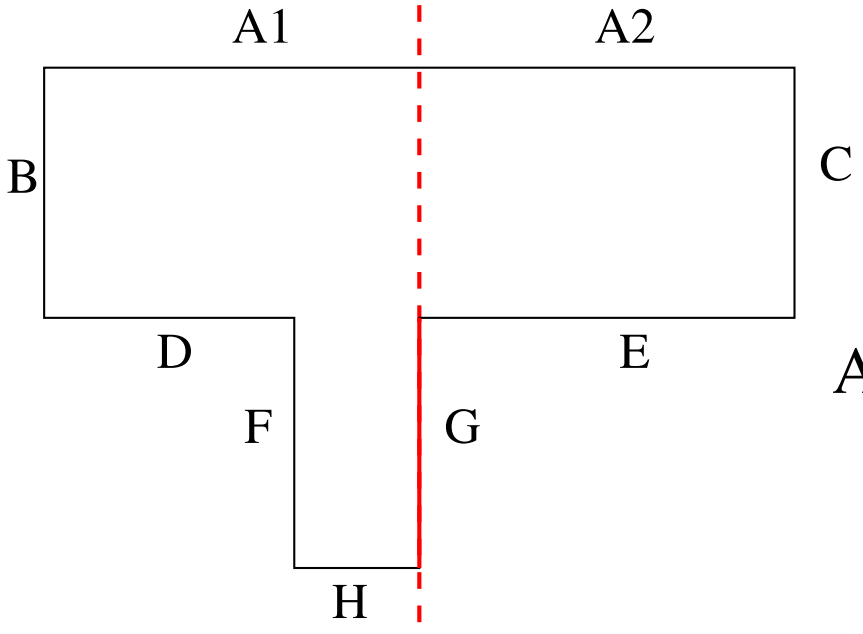
Génération du BSPT

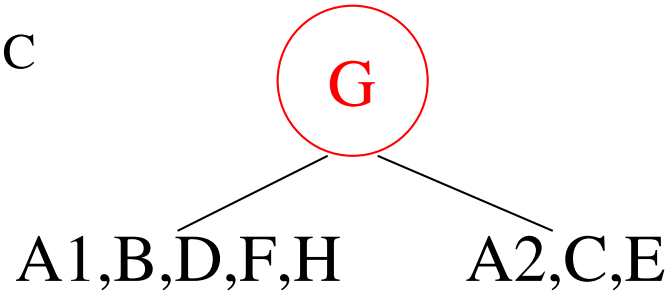
GENERATION_BSPT(L)
Données : une liste L de polygones

1. Si L est vide, renvoyer l'arbre vide.
2. Sinon, choisir un polygone P dans L.
3. Créer un nouveau nœud N étiqueté par la liste vide.
4. Créer deux listes vides L_{devant} et $L_{derriere}$.
5. Pour tout polygone Q de L :
   - Si Q est devant P, ajouter Q à L_{devant}.
   - Si Q est derrière P, ajouter Q à $L_{derriere}$.
   - Si Q coupe le plan de P, on le sépare en deux polygones Q_1 et Q_2 que l'on ajoute respectivement à L_{devant} et $L_{derriere}$.
   - Si Q est dans le même plan que P on l'ajoute à la liste de N.
6. GENERATION_BSPT(L_{devant}) devient le fils gauche de N.
7. GENERATION_BSPT($L_{derriere}$) devient le fils droit de N.
8. Renvoyer N.

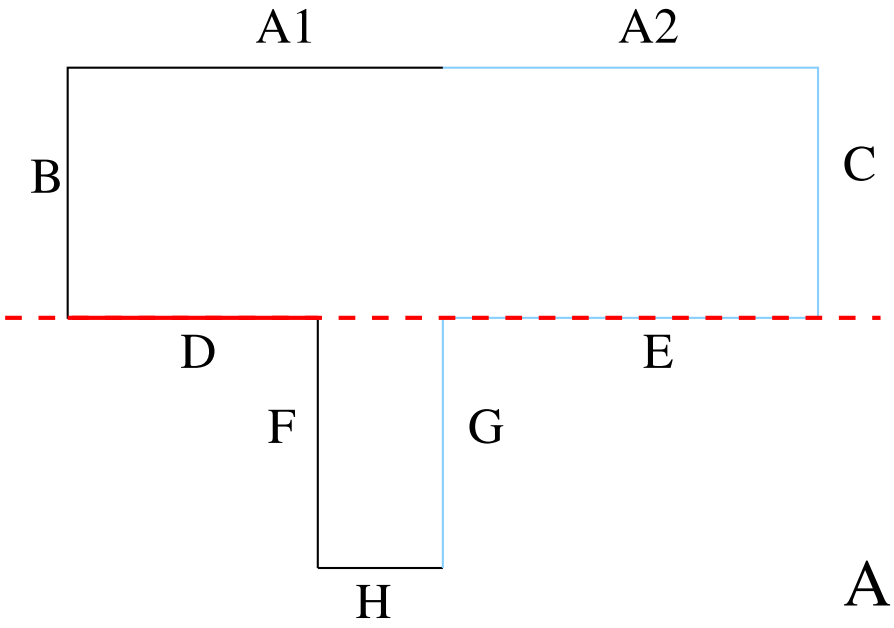
Remarque : on réalise une segmentation de l'espace avec P pour pivot.

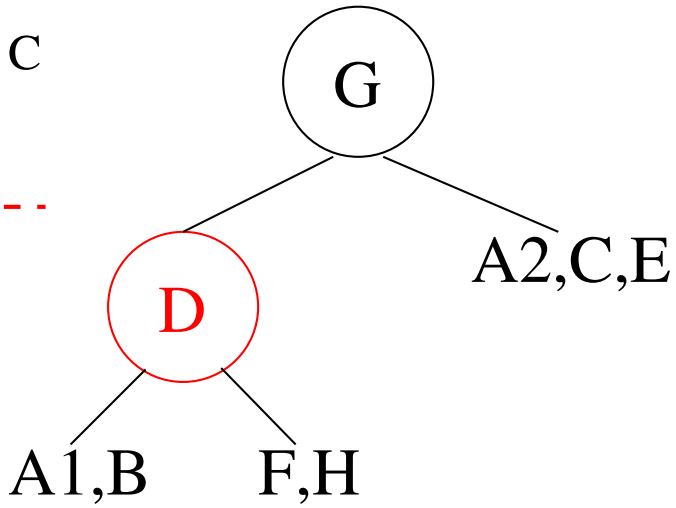
Exemple



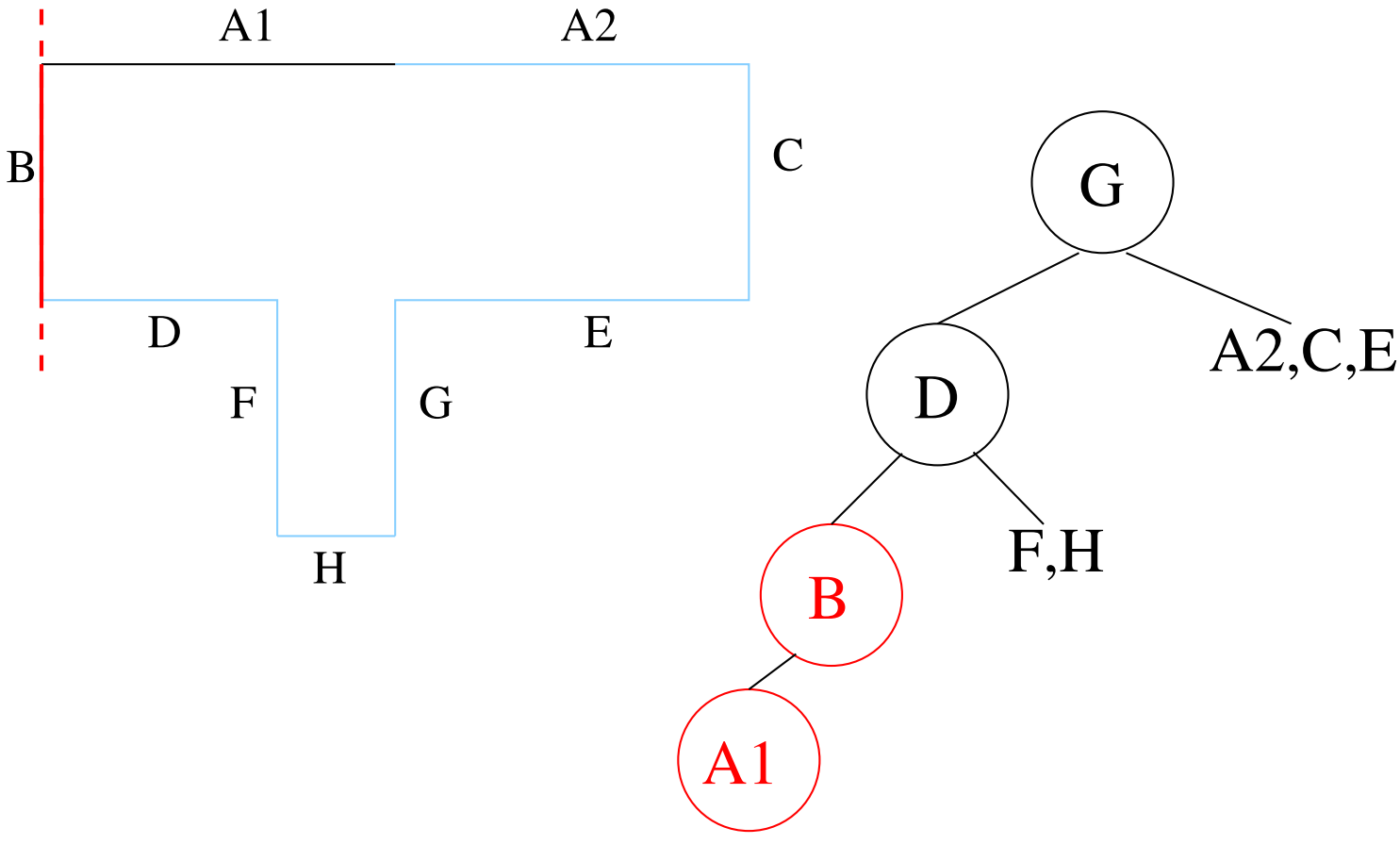


Exemple

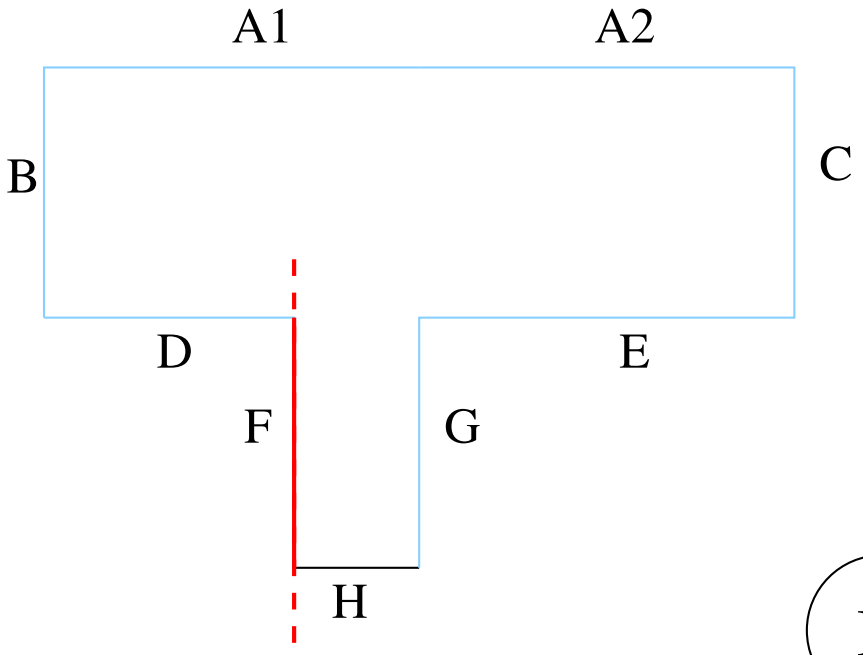


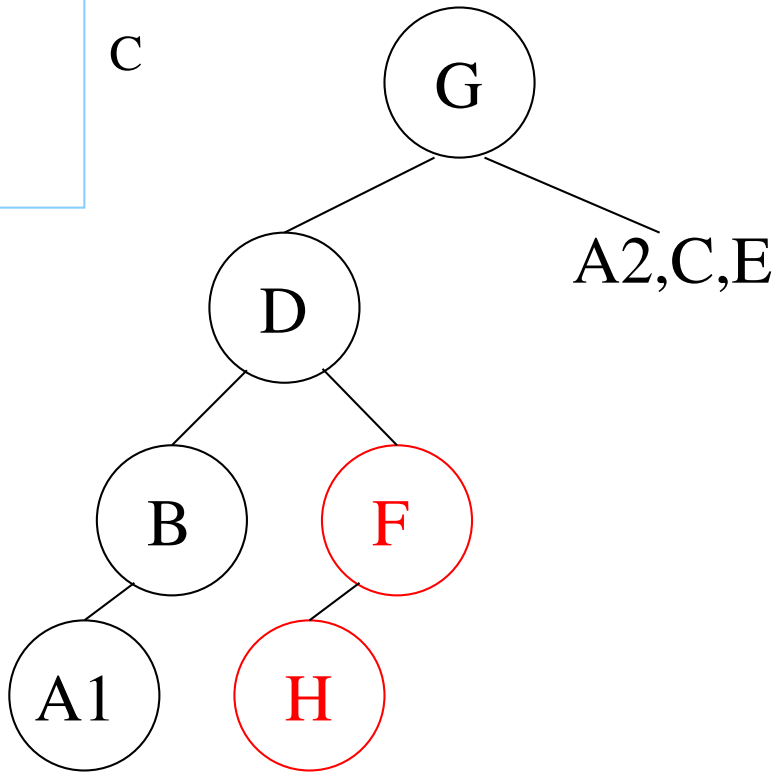


Exemple

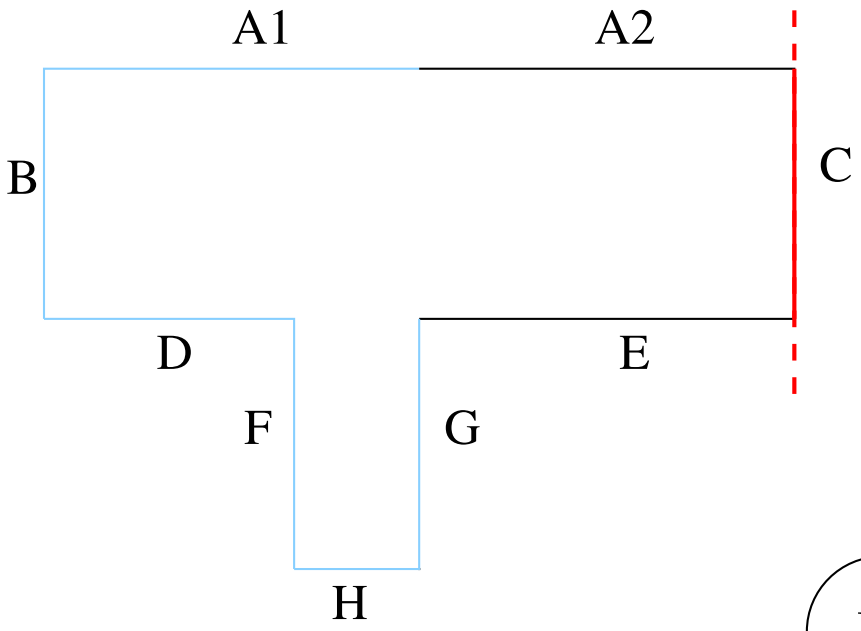


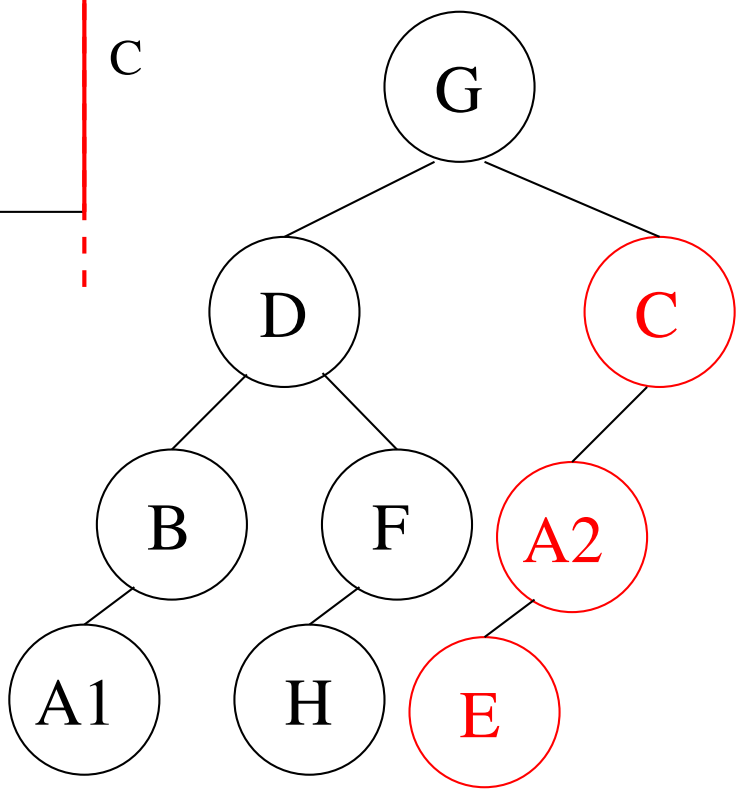
Exemple





Exemple





Parcours du BSPT

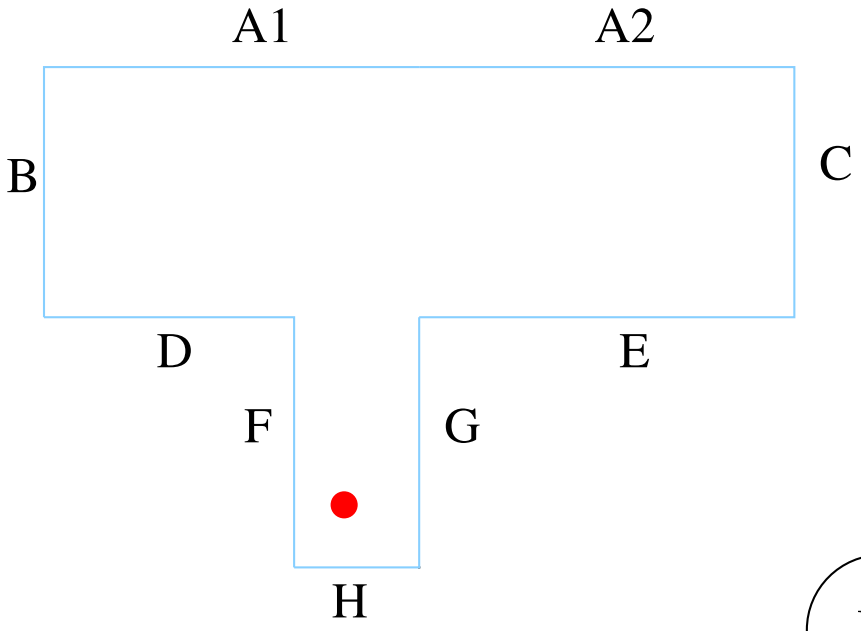
On fixe un point de vue V.
PARCOURS_BSPT(a, V)
Données : un BSPT a, un point de vue V

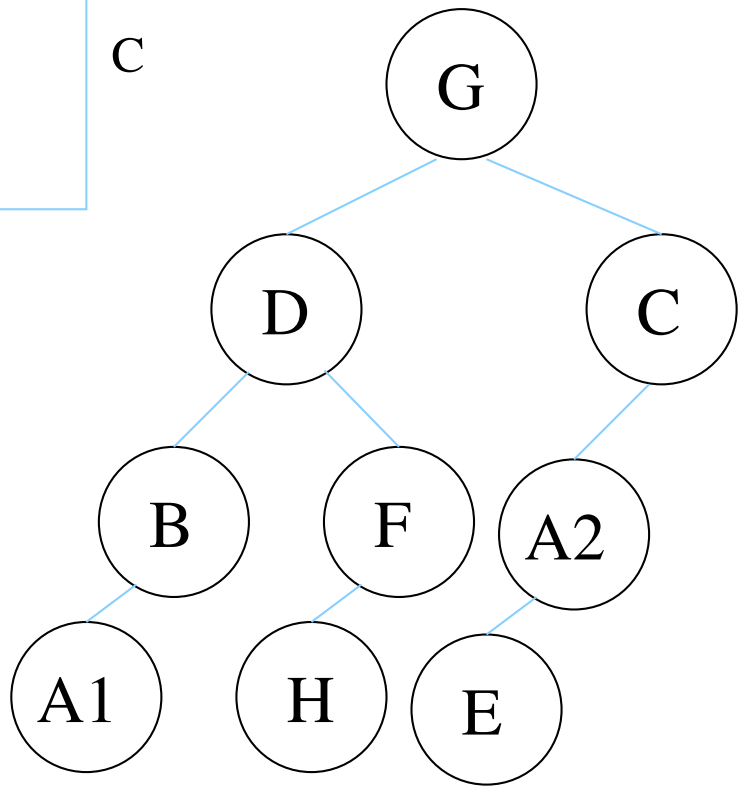
1. Si a est une feuille, dessiner les polygones de a.
2. Sinon, on appelle
   - a_{proche} le fils de a situé du même côté que V par rapport au polygone racine de a
   - a_{loin} le fils de a situé de l'autre côté
3. PARCOURS_BSPT(a_{loin}, V)
4. Dessiner les polygones de a.
5. PARCOURS_BSPT(a_{proche}, V)

Dans le cas (rare !) où V est dans le plan de P :

- l'ordre de parcours n'a pas d'importance ;
- on peut s'abstenir de dessiner les polygones de a.

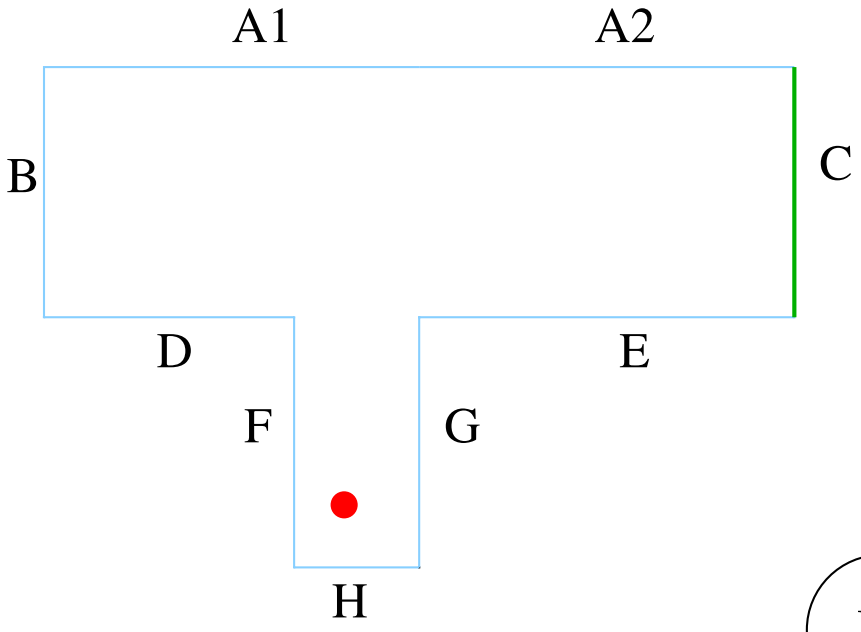
Exemple (cont'd)

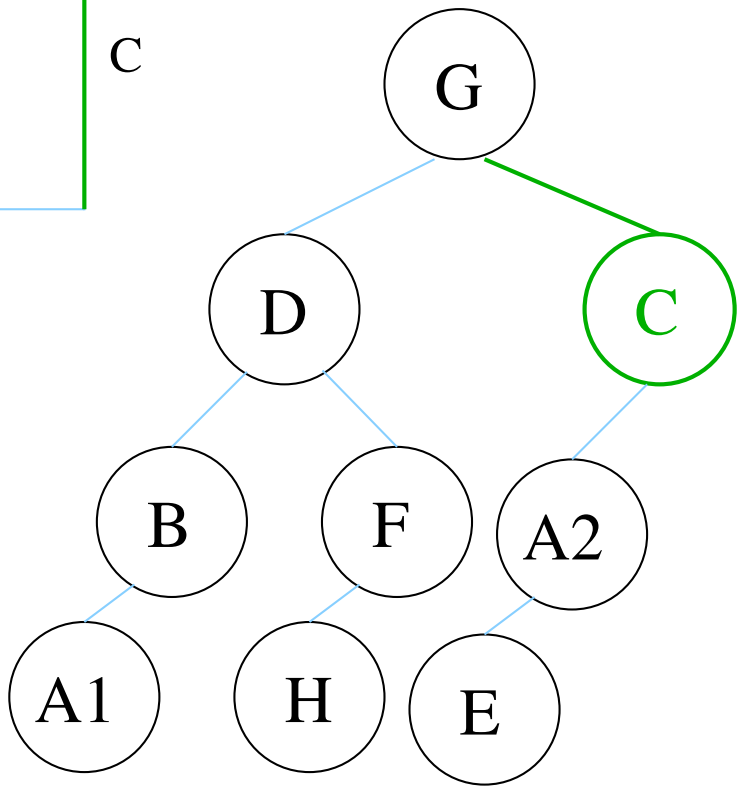




Ordre d'affichage :

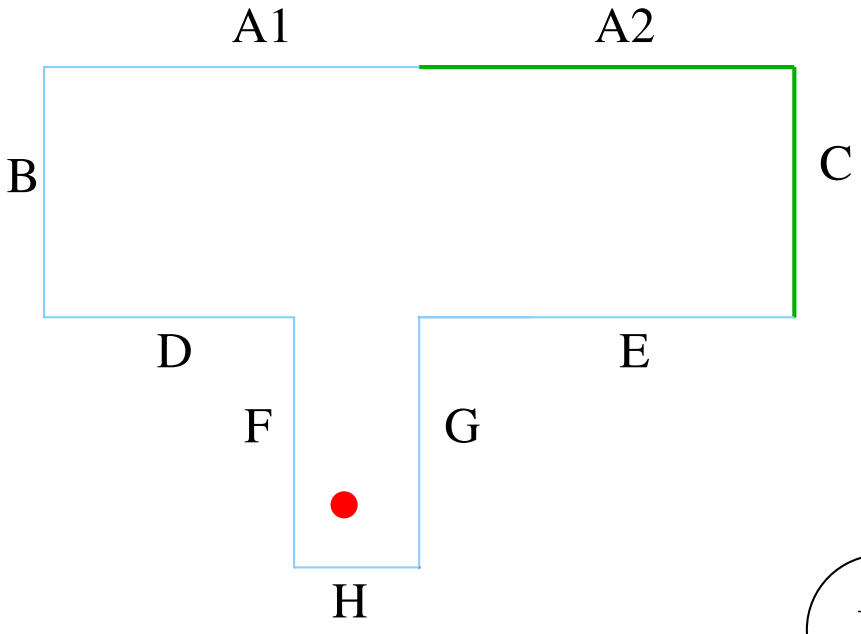
Exemple (cont'd)

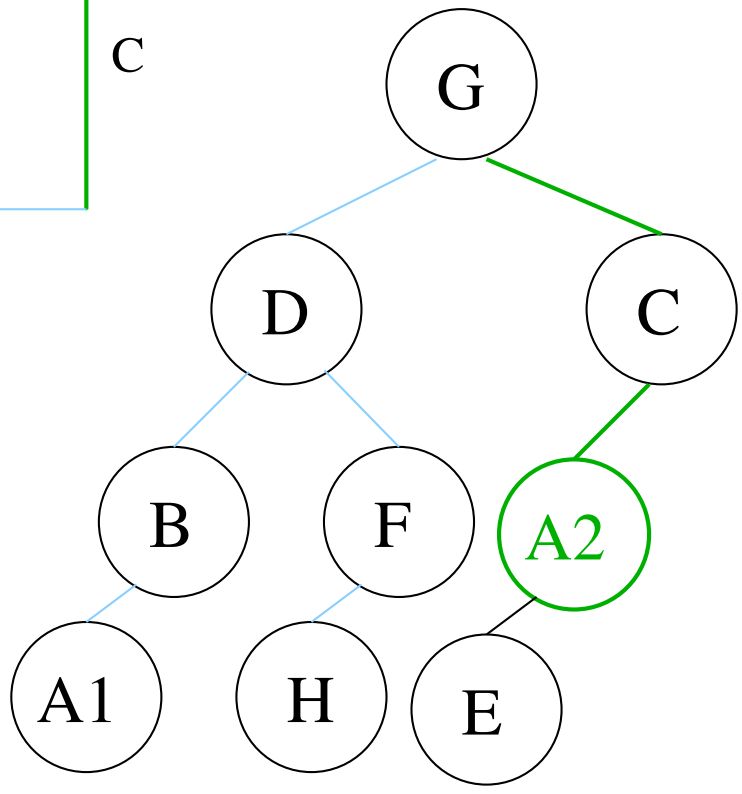




Ordre d'affichage : C,

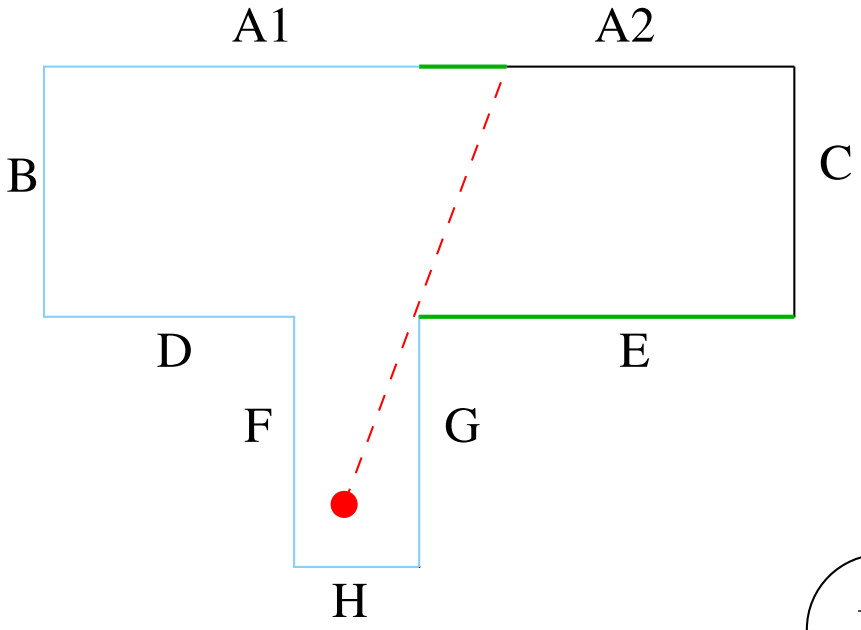
Exemple (cont'd)

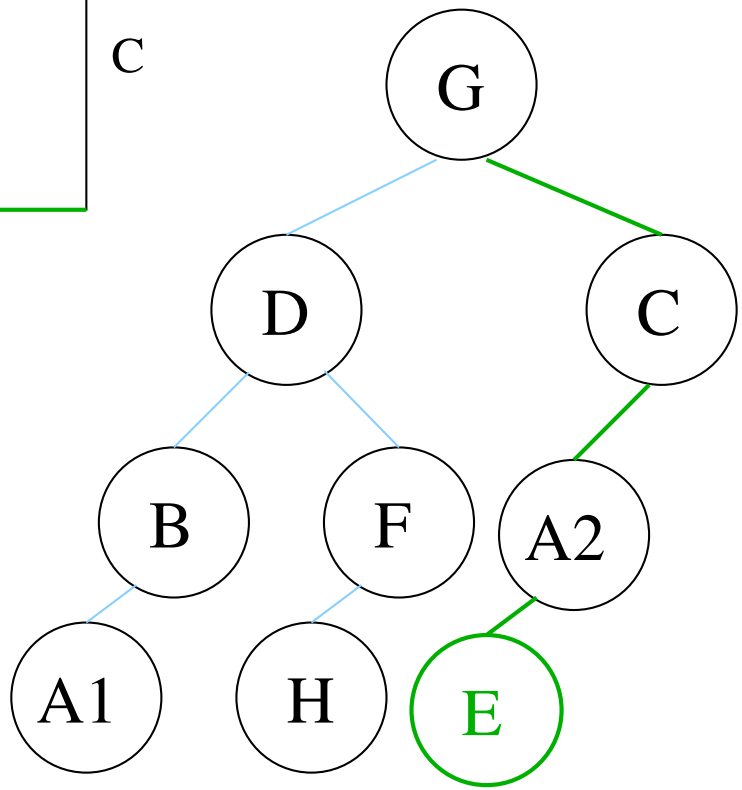




Ordre d'affichage : C, A2,

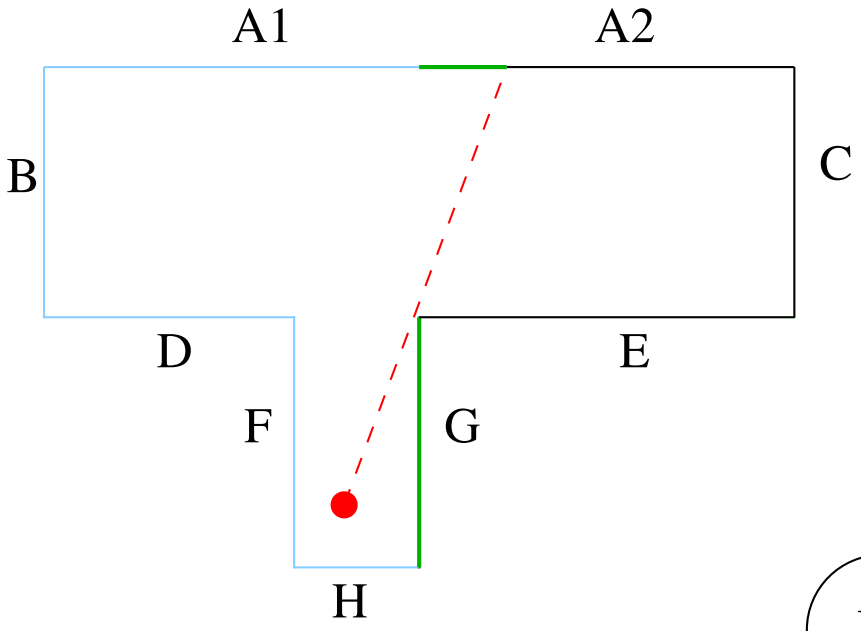
Exemple (cont'd)

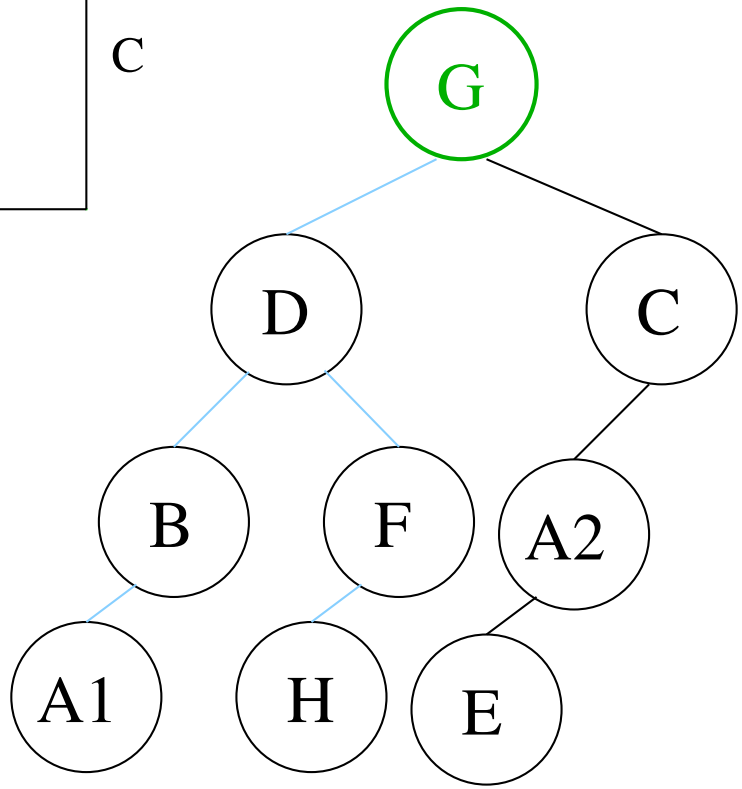




Ordre d'affichage : C, A2, E,

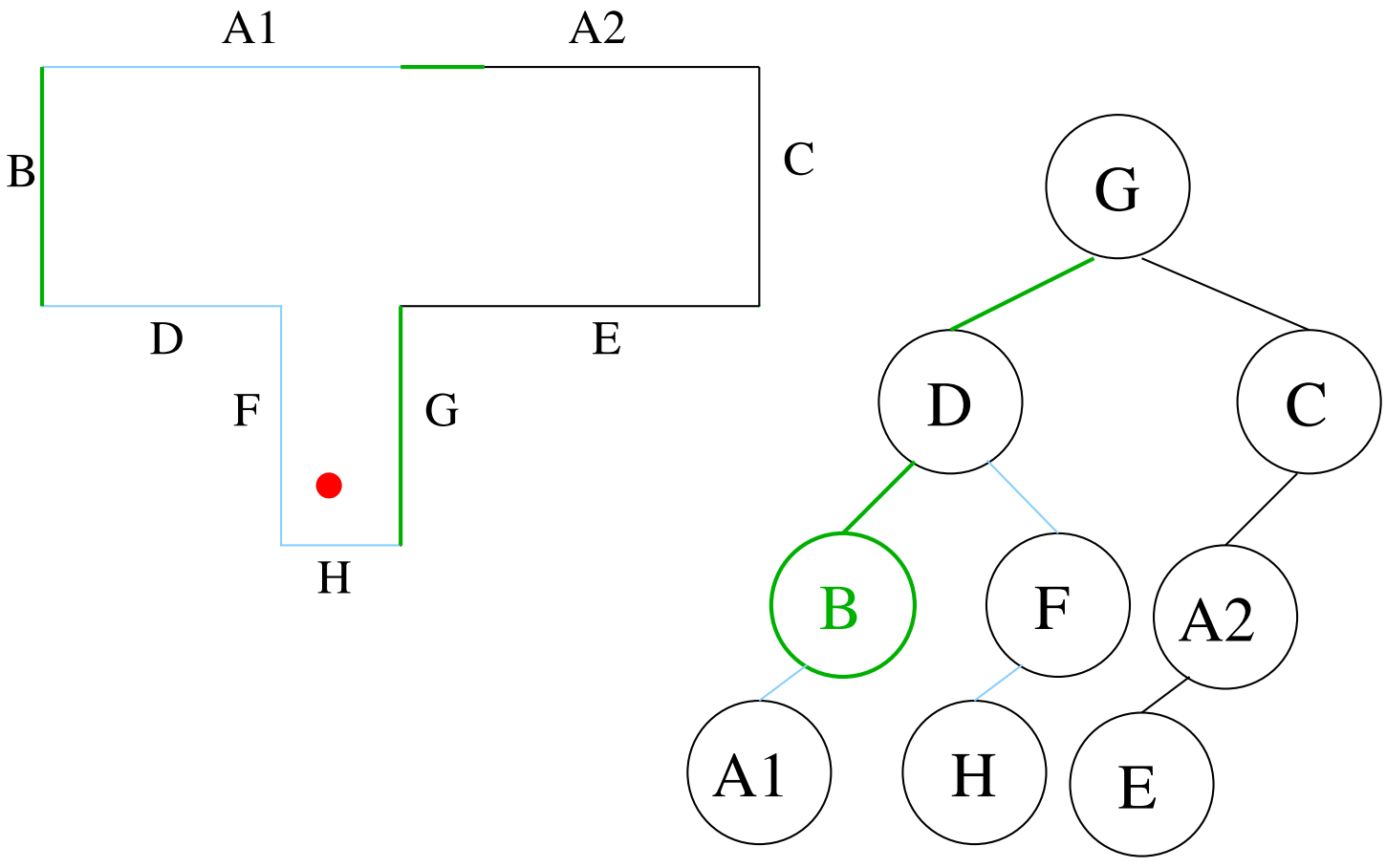
Exemple (cont'd)





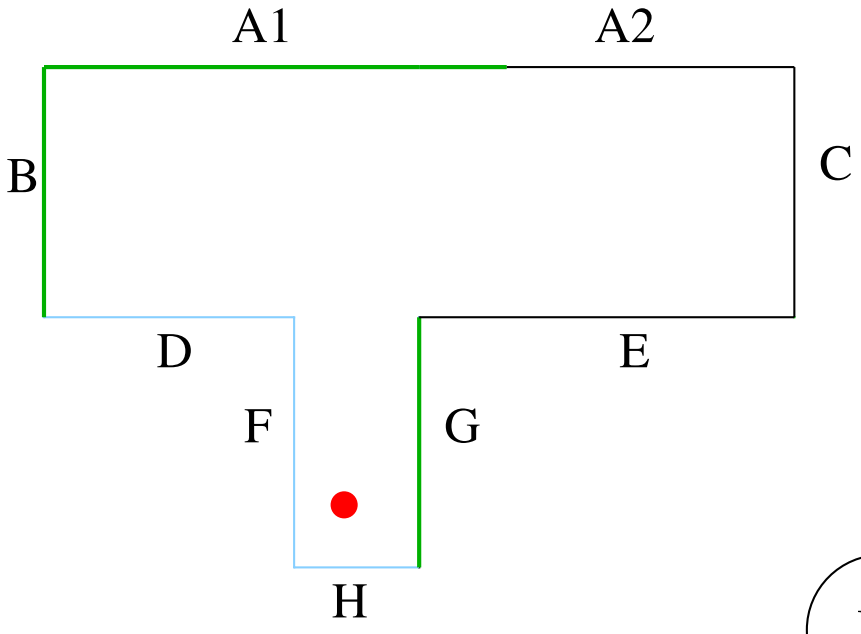
Ordre d'affichage : C, A2, E, G,

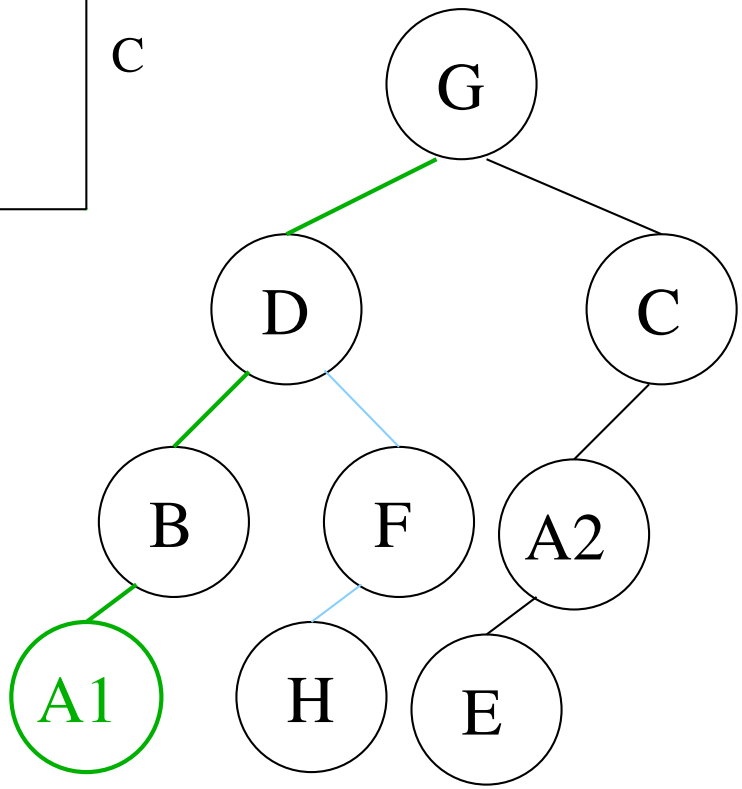
Exemple (cont'd)



Ordre d'affichage : C, A2, E, G, B,

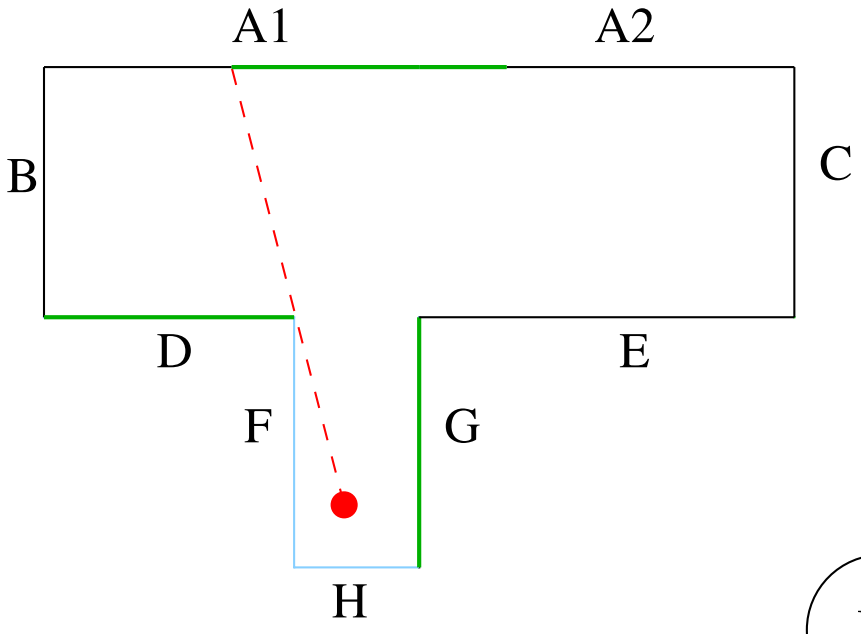
Exemple (cont'd)

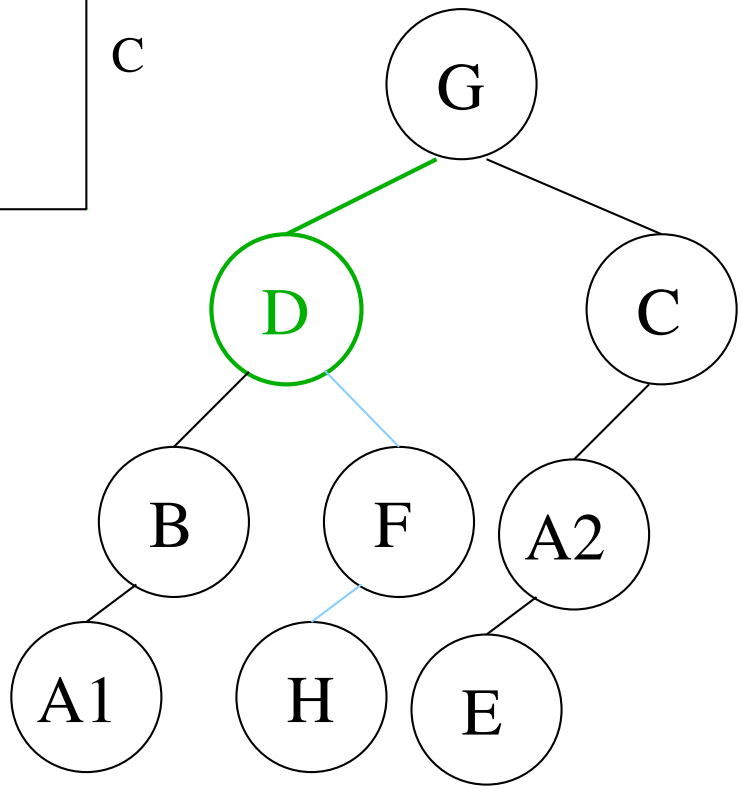




Ordre d'affichage : C, A2, E, G, B, A1,

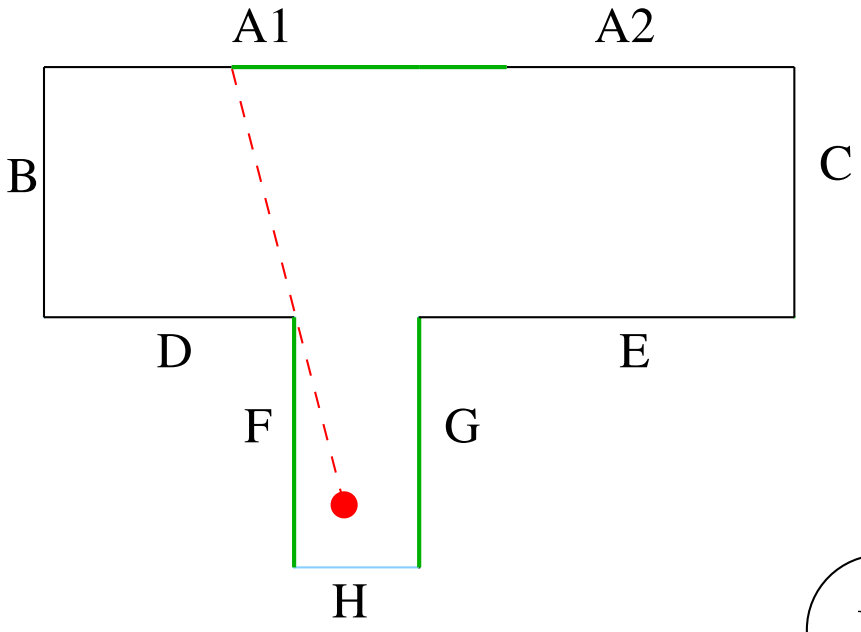
Exemple (cont'd)

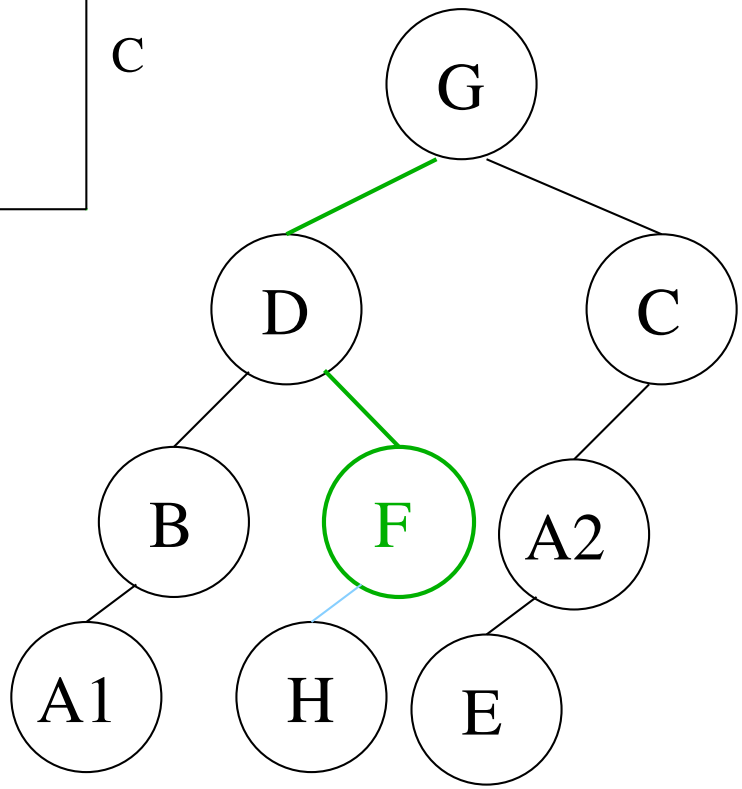




Ordre d'affichage : C, A2, E, G, B, A1, D,

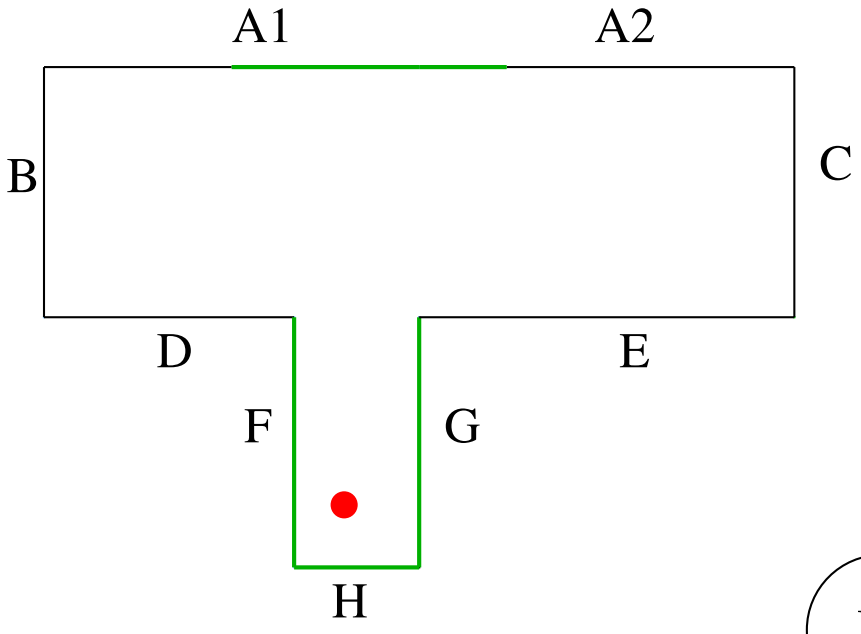
Exemple (cont'd)

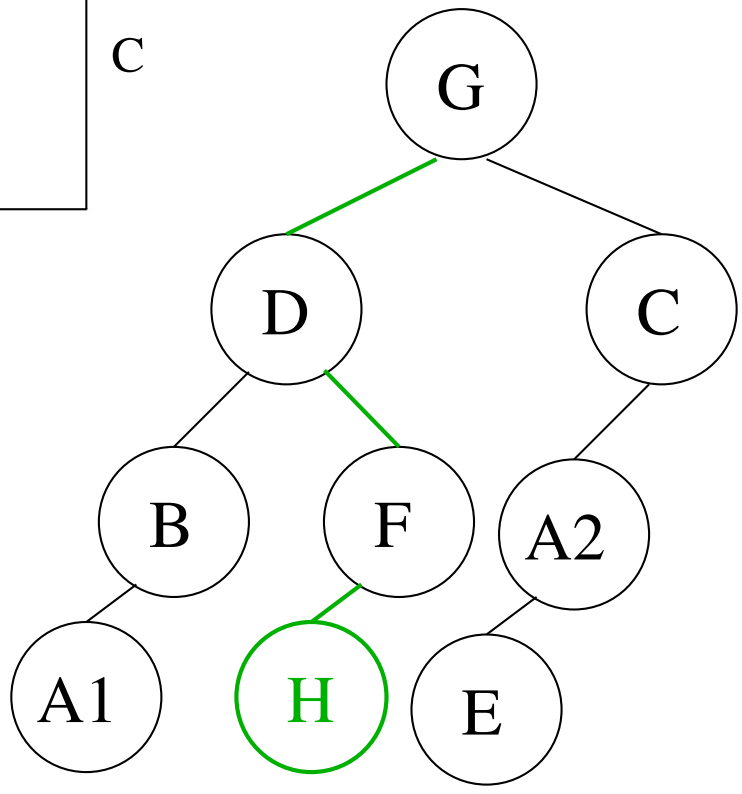




Ordre d'affichage : C, A2, E, G, B, A1, D, F,

Exemple (cont'd)





Ordre d'affichage : C, A2, E, G, B, A1, D, F, H

Gains et limites

Une fois le BSPT généré, pour un point de vue donné, le parcours de l'arbre :

- fournit un ordre compatible avec l'algorithme du peintre
- s'exécute en temps **linéaire** en le nombre de nœuds (mais certains polygones sont dupliqués).

Un tri systématique de tous les polygones à chaque nouveau point de vue demanderait un temps $\mathcal{O}(n \log n)$.

Limites

- Pas adapté aux espaces dynamiques (recalcul de l'arbre trop coûteux)
- La discrimination « devant/derrière » est sujette aux erreurs d'approximation sur les réels.
- La description en polygones devient obsolète dans les environnements virtuels récents.

En résumé

Aujourd'hui

- Les structures arborescentes permettent d'**organiser l'information**
- Le Type Abstrait Arbre offre une interface minimale mais il est facile à adapter à des **besoins variés**
- Une Partition Binaire de l'Espace permet un **parcours ordonné efficace**

La prochaine fois

- Structure de données dynamique
- Arbre Binaire de Recherche