Réseaux, information et communications (INFO-F303)
Partie Théorie de l'Information
4. Compression sans perte

Christophe Petit

Université libre de Bruxelles

Plan du cours

1. Notion de code
2. Source aléatoire et codes efficaces
3. Entropie et codage efficace
4. Compression sans perte
5. Canal bruité
6. Codes correcteurs d'erreurs
7. Codes linéaires
8. Quelques familles de codes linéaires

A. Rappels mathématiques (chapitre 7.1 du syllabus)

Compression sans perte

- Etant donné un message
  - Estimer la probabilité de distribution des symboles
  - Stocker la même information de façon plus courte
  - Tout en permettant la reconstruction du message initial
  - Le tout de façon efficace (rapide)

- Optimiser le **taux de compression**

$$\tau = \frac{\text{taille du message original - taille du message comprimé}}{\text{taille du message original}}$$

Compression de données *avec* perte

- Compromis acceptable pour gagner de la mémoire, surtout si l'information perdue le serait de toute façon pour des raisons physiologiques (impossibilité de percevoir certaines fréquences, etc)
- Algorithmes (non couverts dans ce cours) combinent des variantes de la transformée de Fourier et les algorithmes de compression sans perte

Compression sans perte : deux approches

- Codage par dictionnaire : algorithmes de Lempel-Ziv et Lempel-Ziv-Welch (transmissions analogiques, formats GIF, TIFF, MOD, PDF, etc)
- Codage de Huffman adaptatif : algorithme de Vitter

Codage par dictionnaire

- Idée :
  - Stocker les sous-chaînes de caractères du message dans un dictionnaire (couples indice-chaîne de caractère)
  - Si des longues sous-chaînes sont fréquentes, les substituer par leurs indices comprime le message
- Challenge : représentation du dictionnaire permettant sa construction à la volée pendant le codage et le décodage (pour éviter la transmission)

Algorithme de Lempel-Ziv

- Codage et décodage à la volée
- Dictionnaire contient toutes les sous-chaînes de caractères (de longueur $\leq b$) apparues récemment (commençant au plus w caractères plus tôt)
- Codage est une **suite** de **chaînes de caractères**, chacune représentée par un tuple (ℓ, d, s)
  - $\ell \geq 0$: longueur de la chaîne apparue récemment
  - $d \geq 0$: position de cette chaîne (distance au caractère courant)
  - s : caractère suivant dans le message
  - $\ell = 0$ et $d = \perp$ si nouveau caractère

Algorithme de Lempel-Ziv : exemple

ABRABABABRABABACACABABA, $b = 7$, $w = 8$

État															Code		
Win.								Buf.							ℓ	d	s
A	A	A	A	A	A	A	A	A	B	R	A	B	A	B	1	0	B
A	A	A	A	A	A	A	B	R	A	B	A	B	A	B	0	$\perp$	R
A	A	A	A	A	A	B	R	A	B	A	B	A	B	R	2	2	A
A	A	A	B	R	A	B	A	B	A	B	R	A	B	A	3	1	R
R	A	B	A	B	A	B	R	A	B	A	B	A	C	A	5	6	C
B	R	A	B	A	B	A	C	A	C	A	B	A	B	A	3	1	B
A	B	A	C	A	C	A	B	A	B	A	^{NUL}				3	1	^{NUL}

Crédit exemple : Y Roggeman

Lempel-Ziv : propriétés

- Pas besoin de transmettre le dictionnaire (reconstruit à la volée)
- Taux de compression
  - Négatif pour l'exemple (trop court) ci-dessus
  - Meilleur si longues chaînes proches répétées souvent
  - Pour une fenêtre et un tampon suffisamment longs, la longueur du message compressé tend vers celle obtenue avec un code de Huffman
- Codage = référence du dictionnaire + symbole suivant

Lempel-Ziv : décodage

- En procédant de la même manière, on peut reconstruire le message et le dictionnaire à partir du codé
- Exercice : reconstruire le message à partir du codé pour l'exemple ci-dessus

Algorithme de Lempel-Ziv-Welch : encodage

- Initialiser un dictionnaire avec les symboles individuels
- Lire les symboles successifs à encoder, et répéter :
  - Trouver la plus longue chaîne w de symboles successifs présente dans le dictionnaire
  - Ecrire l'indice de cette chaîne dans le dictionnaire
  - Ajouter la chaîne wc dans le dictionnaire, où c est le symbole suivant en entrée

Algorithme de Lempel-Ziv-Welch : exemple

0 a
1 b

a b a b a b a b a

Algorithme de Lempel-Ziv-Welch : exemple

0 a
1 b
2 ab

a b a b a b a b a
0

Algorithme de Lempel-Ziv-Welch : exemple

0	a
1	b
2	ab
3	ba

a b a b a b a b a

0 1

Algorithme de Lempel-Ziv-Welch : exemple

0	a
1	b
2	ab
3	ba
4	aba

a	b	a	b	a	b	a	b	a
0	1		2					

Algorithme de Lempel-Ziv-Welch : exemple

0	a
1	b
2	ab
3	ba
4	aba
5	abab

a	b	a	b	a	b	a	b	a
0	1		2			4		

Algorithme de Lempel-Ziv-Welch : exemple

0	a
1	b
2	ab
3	ba
4	aba
5	abab

a	b	a	b	a	b	a	b	a
0	1		2			4		3

Algorithme de Lempel-Ziv-Welch : décodage

- Initialiser le dictionnaire
- Décoder le premier indice, en la chaîne w
- Ajouter $w?$ au dictionnaire
- Répéter :
  - Décoder le premier symbole s de la chaîne correspondant à l'indice suivant
  - Remplacer ? par s dans la chaîne précédemment ajoutée au dictionnaire
  - Décoder le reste de la chaîne w correspondant à l'indice courant
  - Ajouter $w?$ au dictionnaire

Mise en œuvre

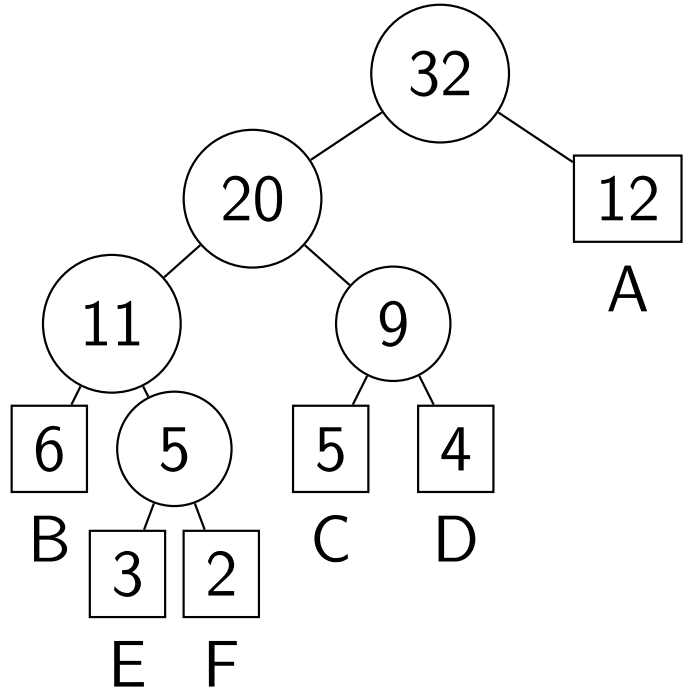
- Stockage du dictionnaire sous forme d'un arbre préfixe (trie) pour identifier rapidement la plus longue chaîne
- Quand le dictionnaire devient trop grand : étendre le nombre de bits dédiés aux indices
- Stratégie LRV : supprimer les entrées les moins visitées ("least recently visited")
- LZW utilisé pour formats GIF, TIFF, MOD, PDF, ..., transmissions analogiques (protocole modem V42bis),

Rappel : compression sans perte

- Codage par dictionnaire : algorithmes de Lempel-Ziv et Lempel-Ziv-Welch (transmissions analogiques, formats GIF, TIFF, MOD, PDF, etc)
- Codage de Huffman adaptatif : algorithme de Vitter

Rappel : code de Huffman

- Code univoque (et sans préfixe) *optimal* : pour une distribution de probabilité donnée sur les symboles, le code de Huffman a une *longueur moyenne minimum*



Code de Huffman adaptatif

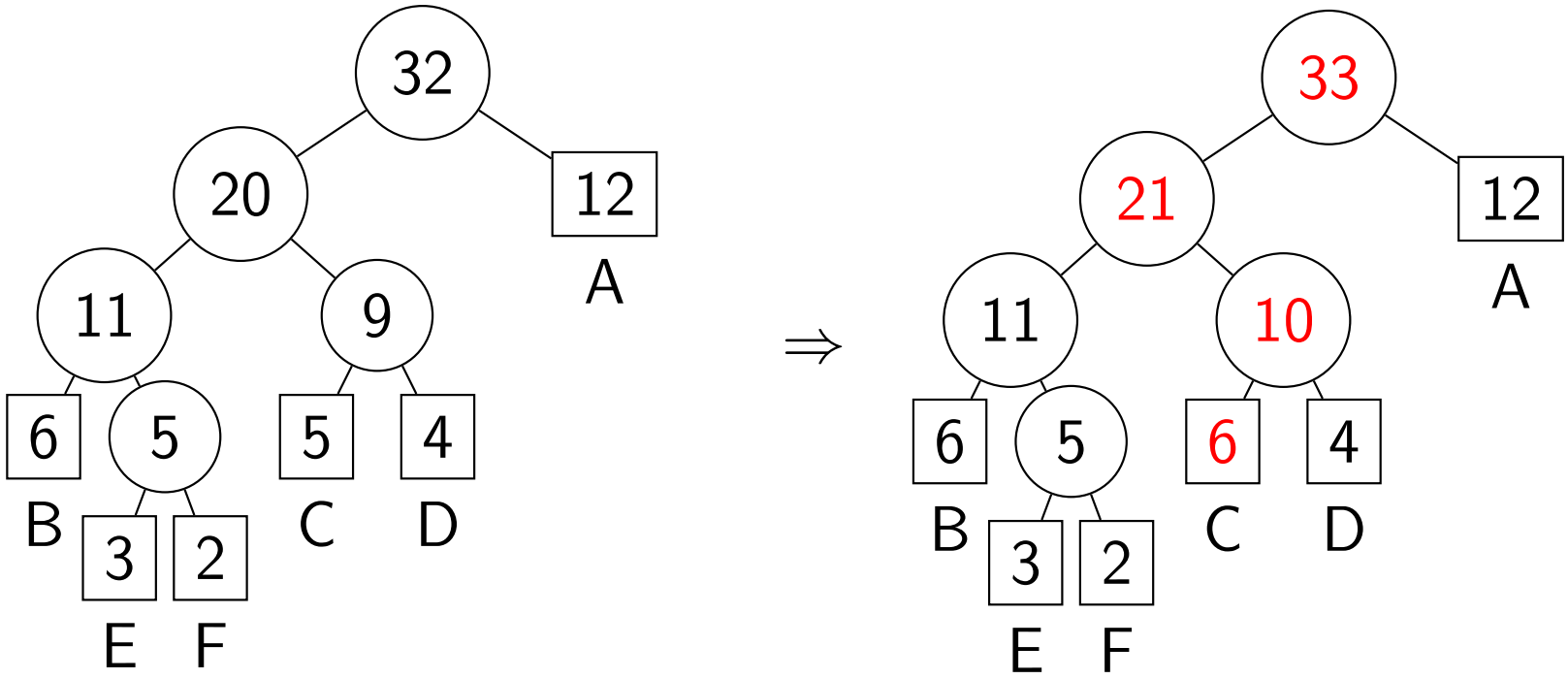
- Code de Huffman est optimal pour une distribution de probabilité donnée, mais son calcul nécessite de connaître ces probabilités
- Idée du codage adaptatif : probabilités estimées à la volée (nombre d'occurences), et code/arbre correspondant mis à jour au fur et à mesure

- Algorithmes de Faller-Gallager-Knuth et Vitter
- Moins utilisés que le codage par dictionnaire en pratique (manipulations des arbres coûteuses)

Code de Huffman adaptatif : squelette

- Démarrer d'un code de Huffman initial trivial "void"
- Lire le message caractère à caractère et répéter :
  - Emission du code pour le symbole courant
  - Incrément de la fréquence du symbole dans l'arbre, ou ajout du symbole
  - Modification de l'arbre pour retrouver un code de Huffman

Mise à jour des fréquences et de l'arbre



- Le résultat est-il encore un code de Huffman ?
- Si pas, peut-on obtenir à nouveau un code de Huffman en effectuant de petites mises à jour sur l'arbre en cours ?

Invariants de Gallager et Vitter

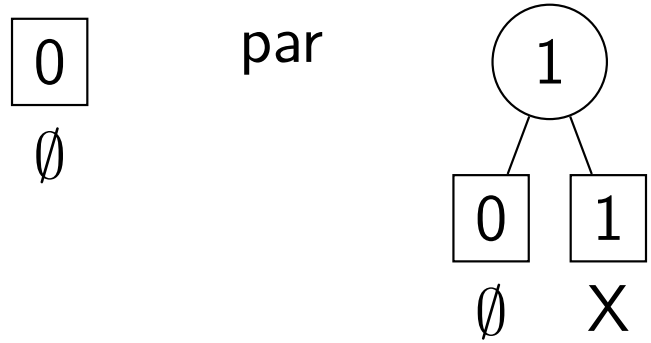
- **Gallager** (1978) : un arbre de code est de Huffman si et seulement s'il satisfait la **propriété de fratrie** : tous les sommets ont un frère (sauf la racine), et on peut les ordonner totalement par probabilités non décroissantes en plaçant les frères de manière contiguë
- **Vitter** (1987) : si dans le parcours par niveau d'un arbre de Huffman, de bas en haut et de gauche à droite, on rencontre, parmi les sommets de même probabilité, toutes les feuilles avant tous les nœuds, alors cet arbre est de *hauteur minimale*

Algorithme de Vitter : définitions

- Ordre implicite : de bas en haut, puis de gauche à droite
- Bloc : ensemble de tous les noeuds de même *poids* et de même *type* (feuille vs sommet interne)
- Les noeuds d'un bloc sont ordonnés implicitement
- Leader : le plus grand noeud d'un bloc (le plus haut, ou le plus à droite en cas d'égalité)
- Invariant de Vitter à conserver : une feuille précède ($<$) toujours un noeud de même poids
- Feuille *symbole inconnu* $\emptyset$: symbole pas encore rencontré (l'arbre est initialisé avec ce sommet en racine)

Algorithme de Vitter : émission du code

- Le symbole à coder X est cherché dans l'arbre
- S'il y est, on émet le mot de code associé à X ($\sim$ chemin de l'arbre menant à X)
- Sinon, on émet le mot de code associé à $\emptyset$ suivi de X, et on remplace

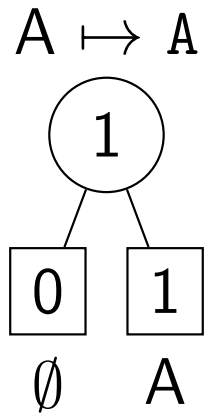


Algorithme de Vitter : mise à jour de l'arbre

- On remonte du sommet codé vers la racine
  - On incrémente le sommet en cours
  - On le place au bout (ordre minimal, en bas à gauche) de son nouveau bloc (poids $+1$, même type)
  - Ceci nécessite de permuter des sommets du même bloc entre eux (ainsi que leurs sous-arbres)

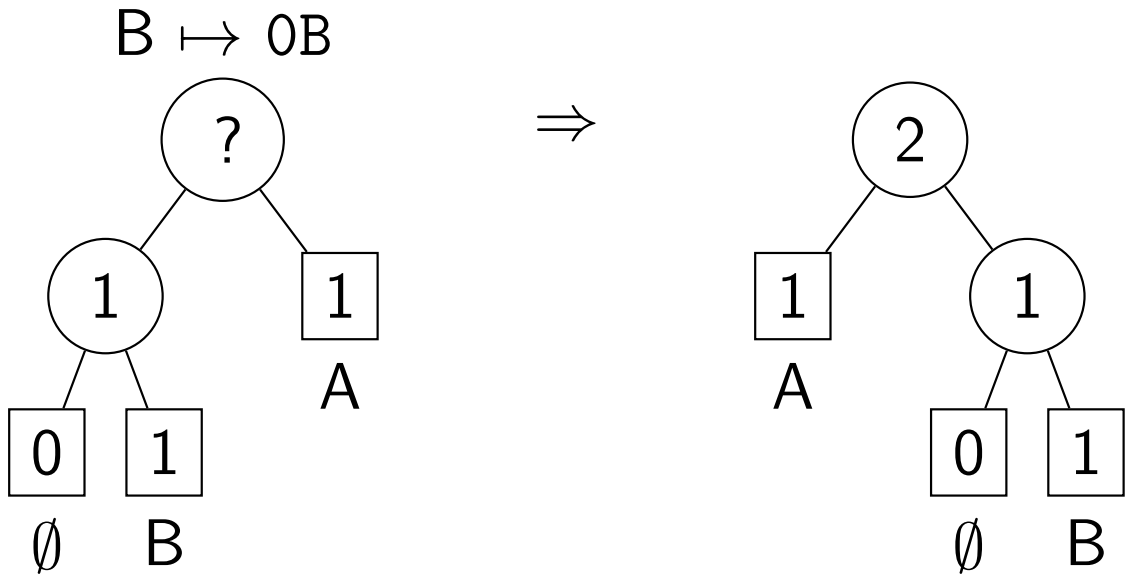
Algorithme de Vitter : exemple

"ABRABABABRABABACACABABA"



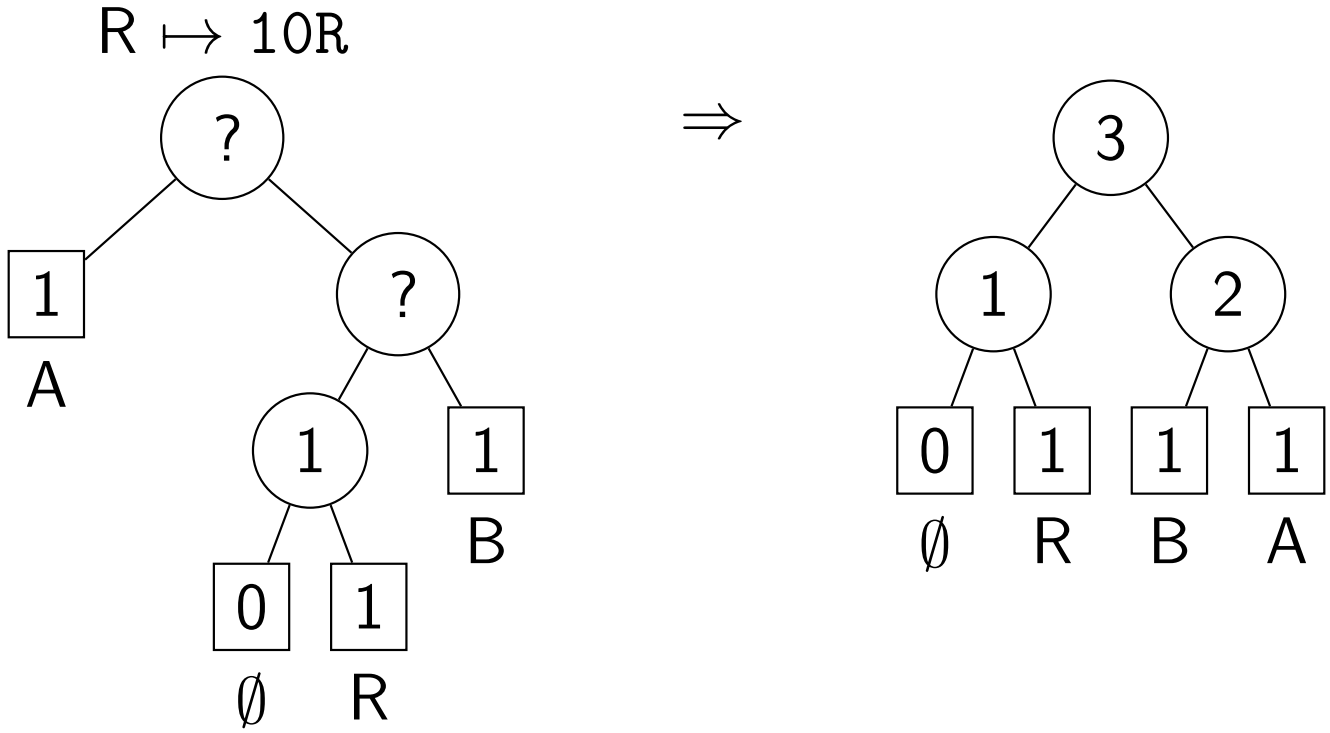
Algorithme de Vitter : exemple

"ABRABABABRABABACACABABA"



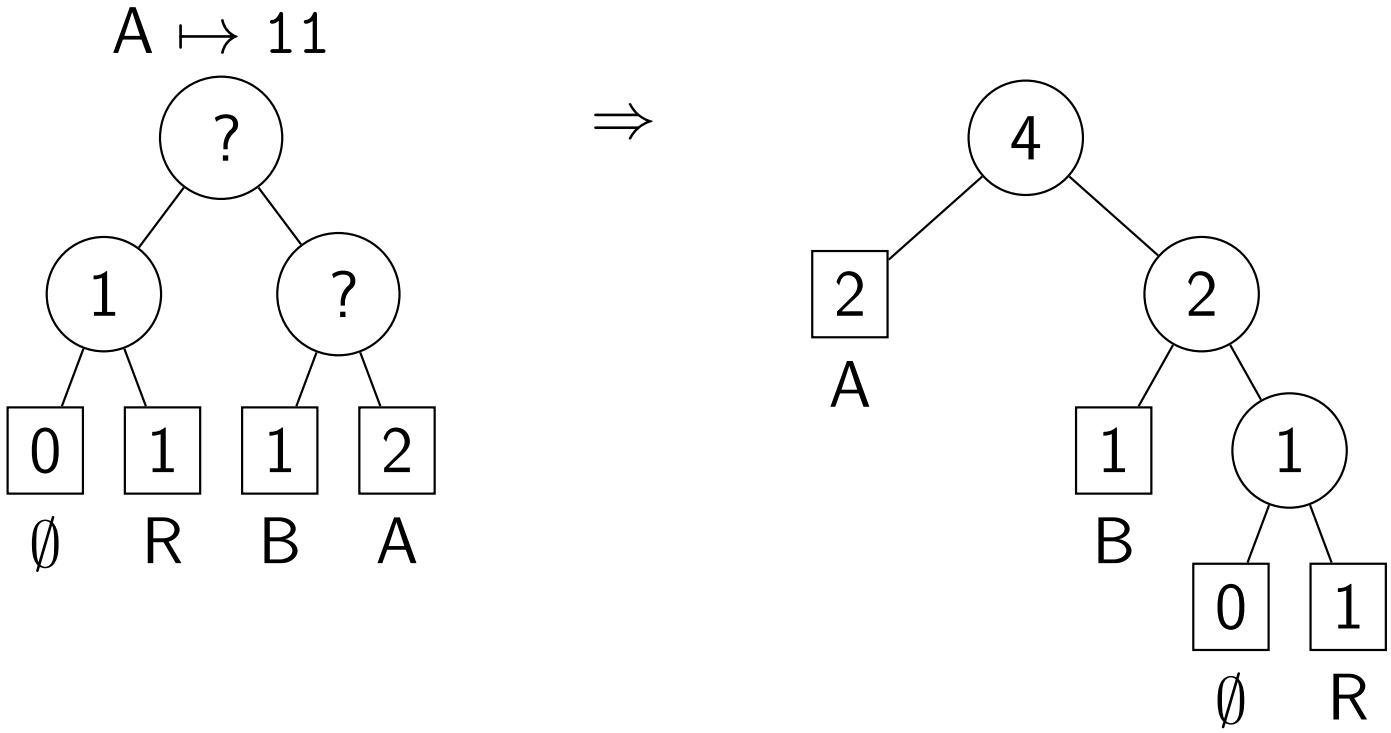
Algorithme de Vitter : exemple

"ABRABABABRABABACACABABA"



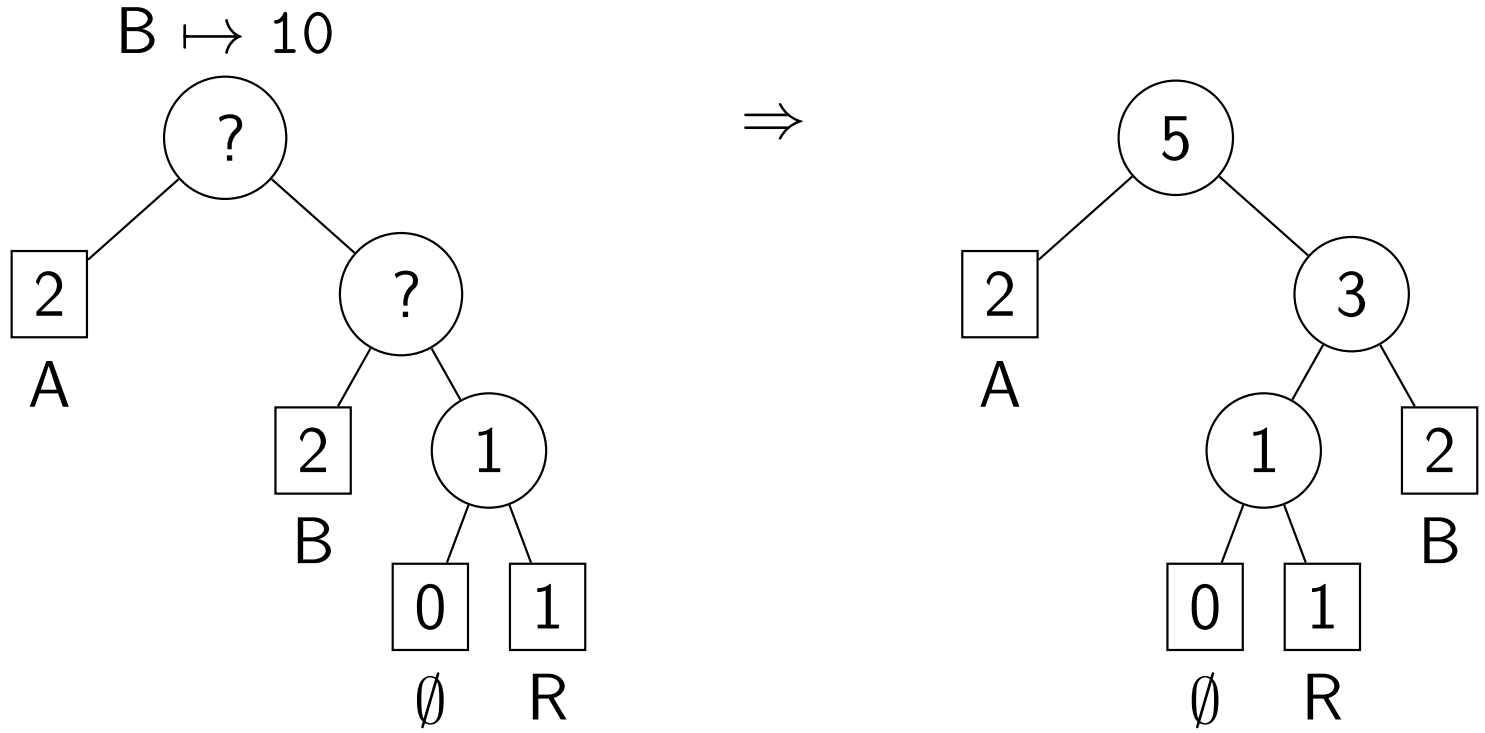
Algorithme de Vitter : exemple

"ABRABABABRABABACACABABA"



Algorithme de Vitter : exemple

"ABRABABABRABABACACABABA"



Questions ?

?

Crédits et remerciements

- Mes transparents suivent fortement les notes de cours développées par le Professeur Yves Roggeman pour le cours INFO-F303 à l'Université libre de Bruxelles
- Une partie des transparents et des exercices ont été repris ou adaptés des transparents développés par le Professeur Jean Cardinal pour ce même cours
- Je remercie chaleureusement Yves et Jean pour la mise à disposition de ce matériel pédagogique, et de manière plus large pour toute l'aide apportée pour la reprise de ce cours
- Les typos et erreurs sont exclusivement miennes (merci de les signaler !)