Arbres des suffixes

Nadia El-Mabrouk

Structures de données pour les séquences biologiques

Des structures de données appropriées doivent être utilisées pour traiter les séquences biologiques (ADN, protéines, ARN): accès rapide, et espace mémoire raisonnable.

Arbre de recherche: Structure de donnée appropriée pour le traitement de données numériques ordonnées. De la même façon, on a besoin de structures appropriées pour le traitement des séquences.

Les structures de données développées pour traiter les séquences (string data structures) idéales pour la recherche exacte de motifs: Étape très importante à plusieurs tâches. Par exemple, la 1ère étape de BLAST consiste en une recherche exacte de “seeds”.

Lookup Tables

Lookup Table: Structure de données simple permettant de conserver les positions des occurrences de tous les facteurs d'une certaine taille w d'une ou de plusieurs séquences: chaque entrée d'une table lookup pointe sur une liste chainée contenant les positions des occurrences du facteur correspondant.

Utilisées dans de nombreuses applications, en particulier dans BLAST pour conserver les occurrences des "seeds" (graines).

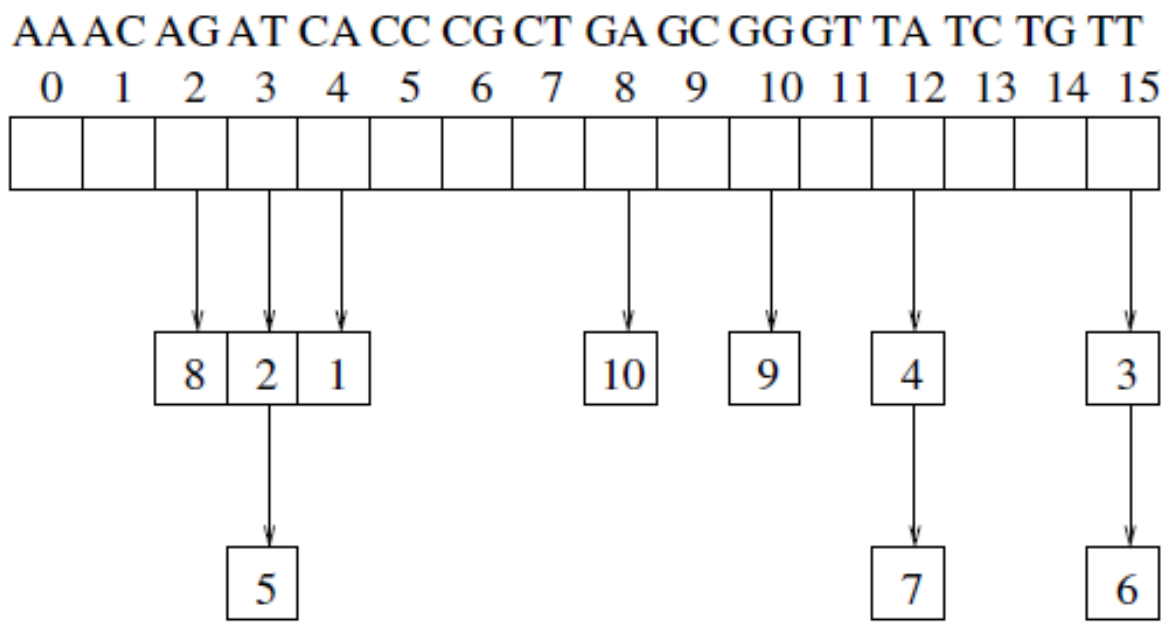
Soit w une constante (dans BLAST, taille d'une graine), et Σ l'alphabet. Une table lookup est un tableau de taille $|\Sigma|^w$, correspondant à tous les mots de taille w sur Σ. Soit $f : \Sigma \longrightarrow \{0, 1, \cdots \Sigma - 1\}$ une fonction qui associe un numéro différent à chaque caractère de l'alphabet. Alors, chaque mot de taille w peut être traité comme un nombre en base $|\Sigma|$.

Lookup Tables

Table lookup pour $S = CATTATTAGGA$, et $w = 2$. Construite en utilisant le mapping:

$$A \longrightarrow 0;\ C \longrightarrow 1; G \longrightarrow 2; T \longrightarrow 3$$

Par exemple, TA correspond à 30 en base 4, ce qui fait 12.



Lookup Tables

Une table Lookup pour une séquence S de taille n peut être créée en temps $O(|\Sigma|^w + n)$:

- Créer un tableau vide en temps $O(|\Sigma|^w)$;
- Insérer chacun des $n - w + 1$ facteurs de taille w de S: Trouver l'indice dans le tableau, puis insérer la pos. dans la liste chaînée. Peut se faire en temps constant.

Espace également en $O(|\Sigma|^w + n)$. Pour avoir un espace linéaire en la taille de la séquence on doit avoir $w \leq \log_{|\Sigma|} n$.

Une fois la table lookup construite pour S, toutes les occurrence d'une séquence requête de taille w dans S peuvent être retrouvées en temps $O(w + k)$, où k est le nombre d'occurrences.

Peut-être utilisé également pour la représentation d'un ensemble de séquences.

Le problème principal des tables lookup est leur dépendance vis-à-vis de la constante w: Pas efficace pour la recherche d'un motif de taille $l > w$.

Arbre des suffixes

Arbre des suffixes: Structure de données reflétant les caractéristiques internes des séquences.

Application naturelle: Recherche exacte dans un texte

- Phase de prétraitement: Construction de l'arbre des suffixes; $O(n)$ en temps et en espace pour un texte de taille n
- Phase de recherche: temps $O(m)$ pour un mot de taille m

Autre application: Recherche de répétitions, recherche de palindromes

Définition

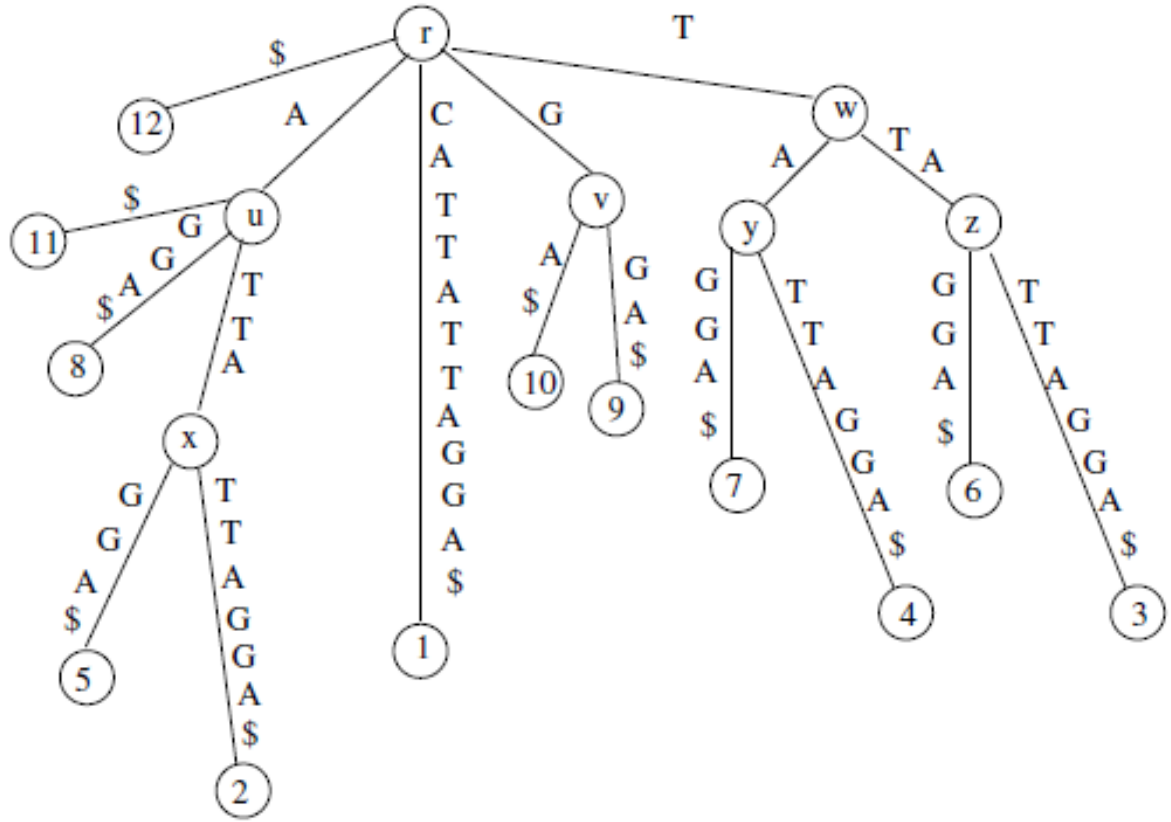
Arbre $\mathcal{S}$ des suffixes de S de taille n: arbre orienté tq

- n feuilles numérotées de 1 à n, chacune correspondant à un suffixe de S;
- Chaque nœud interne a au moins deux fils;
- Chaque arête est étiquettée par un facteur de S;
- Deux arêtes sortant d'un même nœud ont des étiquettes débutant par des caractères différents;
- Chemin de la racine à une feuille i étiquetté par le suffixe $S[i..n]$

Problème: Si un suffixe coïncide avec un facteur de S, aucune feuille ne correspondra au suffixe.

$\longrightarrow$ Rajouter un caractère "artificiel" à la fin de S. Par exemple, '#' ou '$'

Arbre des suffixes pour $S = C^1A^2T^3T^4A^5T^6T^7A^8G^9G^{10}A^{11}\$^{12}$:



Pour réduire l'espace, représenter chaque étiquette par deux positions plutôt que par un facteur

Chaque nœud interne de $\mathcal{S}$ a au moins 2 fils $\Rightarrow$ au plus $n-1$ nœudes internes $\Rightarrow$ Taille de $\mathcal{S}$ en $O(n)$

Recherche exacte de P dans S

Matcher les caractères de P en suivant un chemin unique dans $\mathcal{S}$

- Si les caractères de P ne sont pas épuisés, il n'y a aucune occurrence de P dans $S \longrightarrow O(m)$.
- Si non, les feuilles du sous-arbre déterminé par P donnent toutes les positions de P dans $S \longrightarrow$ Temps $O(m+k)$ où k nombre d'occurrences de P dans S.

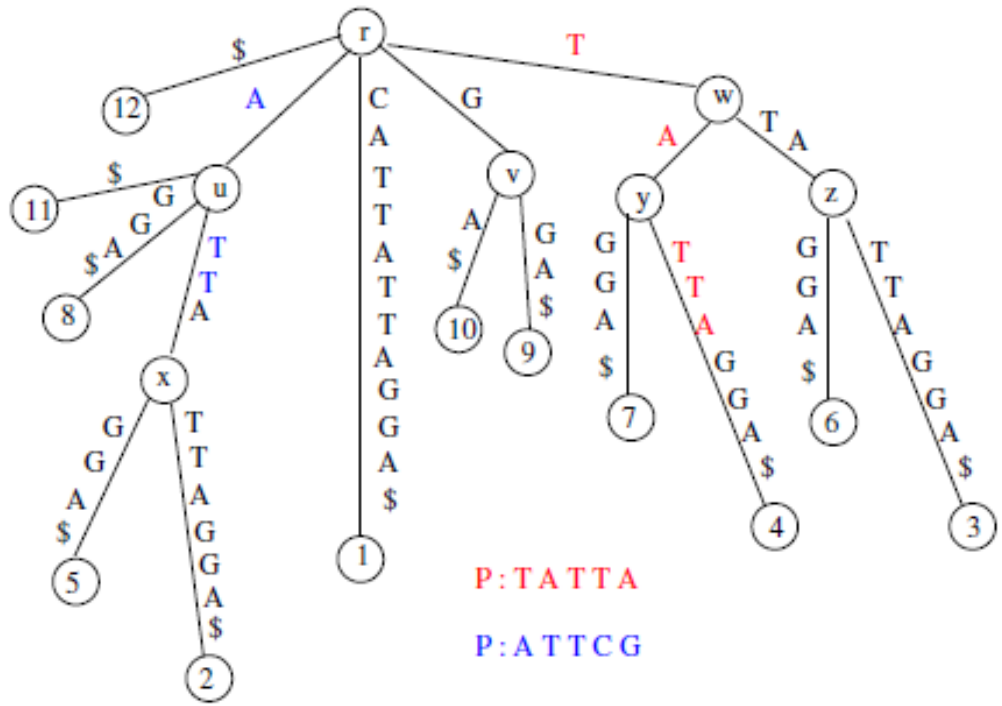


Table des suffixes

Table des suffixes (Suffix Array) pour S est une table contenant les suffixes de S en order lexicographique.

Table des suffixes pour $S = C^1A^2T^3T^4A^5T^6T^7A^8G^9G^{10}A^{11}\$^{12}$:

12	11	8	5	2	1	10	9	7	4	6	3

Les arbres et tables des suffixe peuvent-être construits en temps linéaire. Ils sont utiles pour une multitude de tâches liées à l'analyse des séquences.

L'avantage des tables des suffixes par rapport aux arbres des suffixes est une plus grande efficacité en espace.

Construction de l'arbre des suffixes

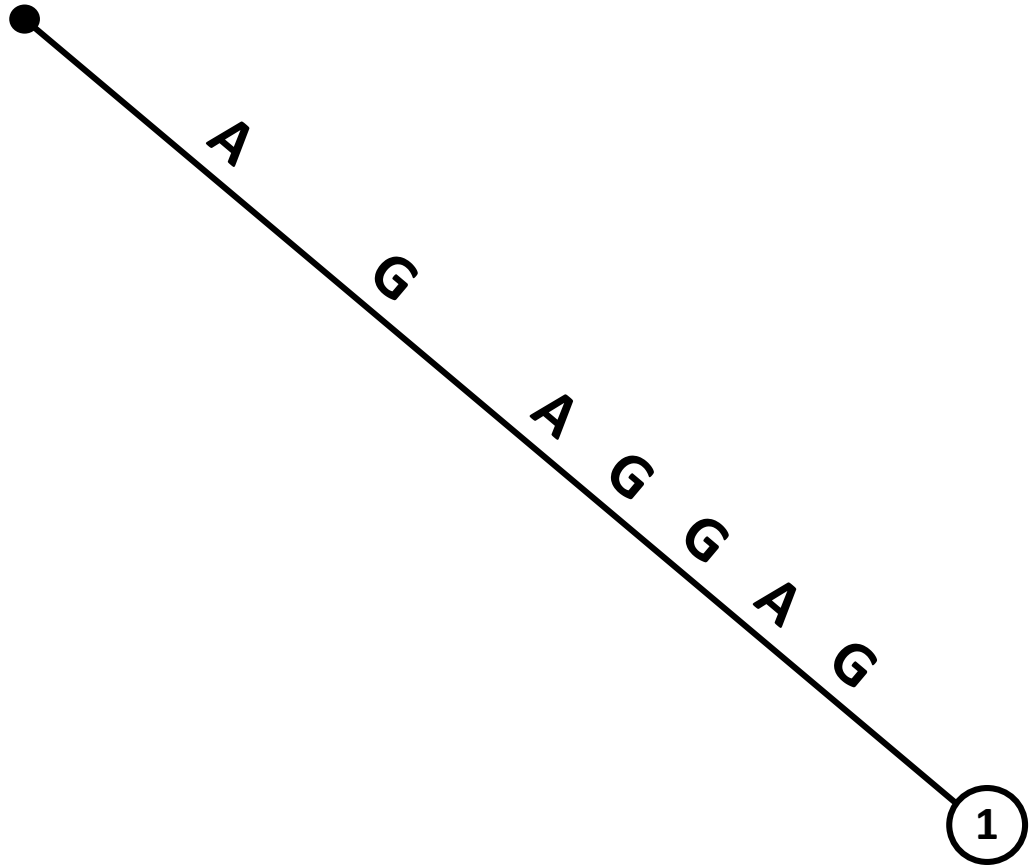
Construction naïve:

Insertion successive des suffixes, du plus long au plus court

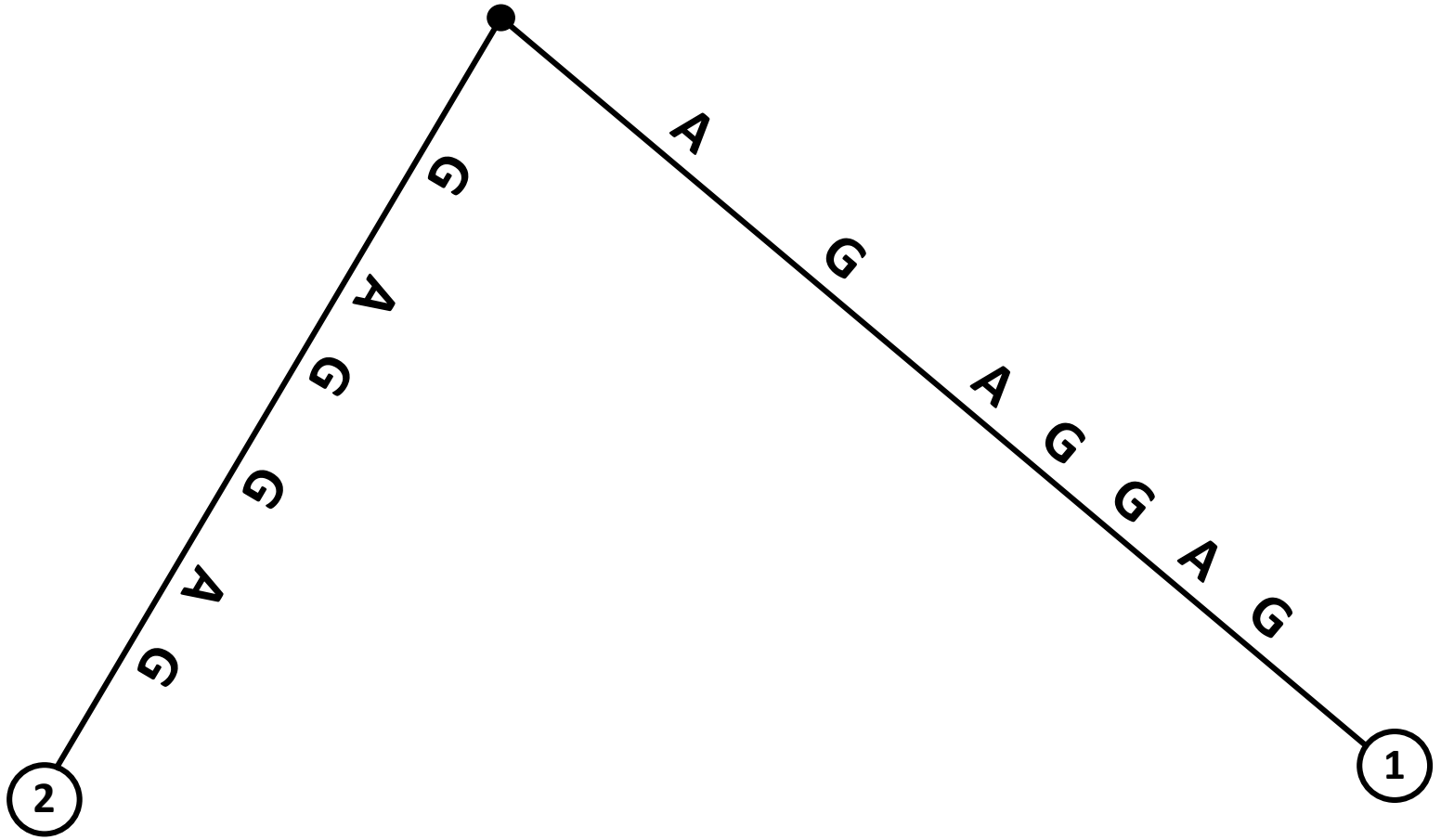
Exemple: Pour $P = aabbabb$, insérer successivement $aabbabb\$$, $abbabb\$$, $bbabb\$$, $\cdots$

	1	2	3	4	5	6	7
S_1:	**A**	**G**	**A**	**G**	**G**	**A**	**G**

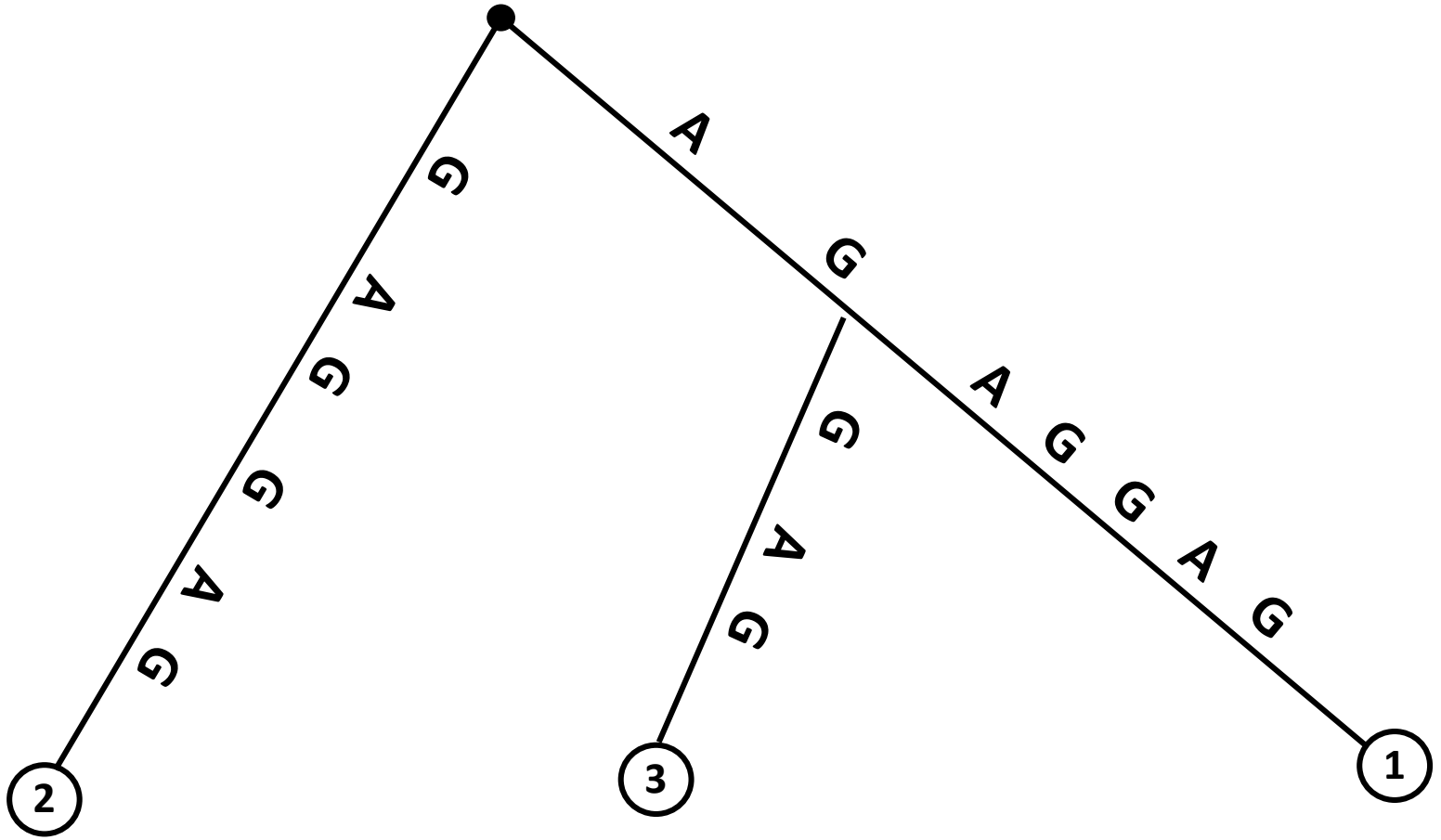
	1	2	3	4	5	6	7
S_1:	A	G	A	G	G	A	G



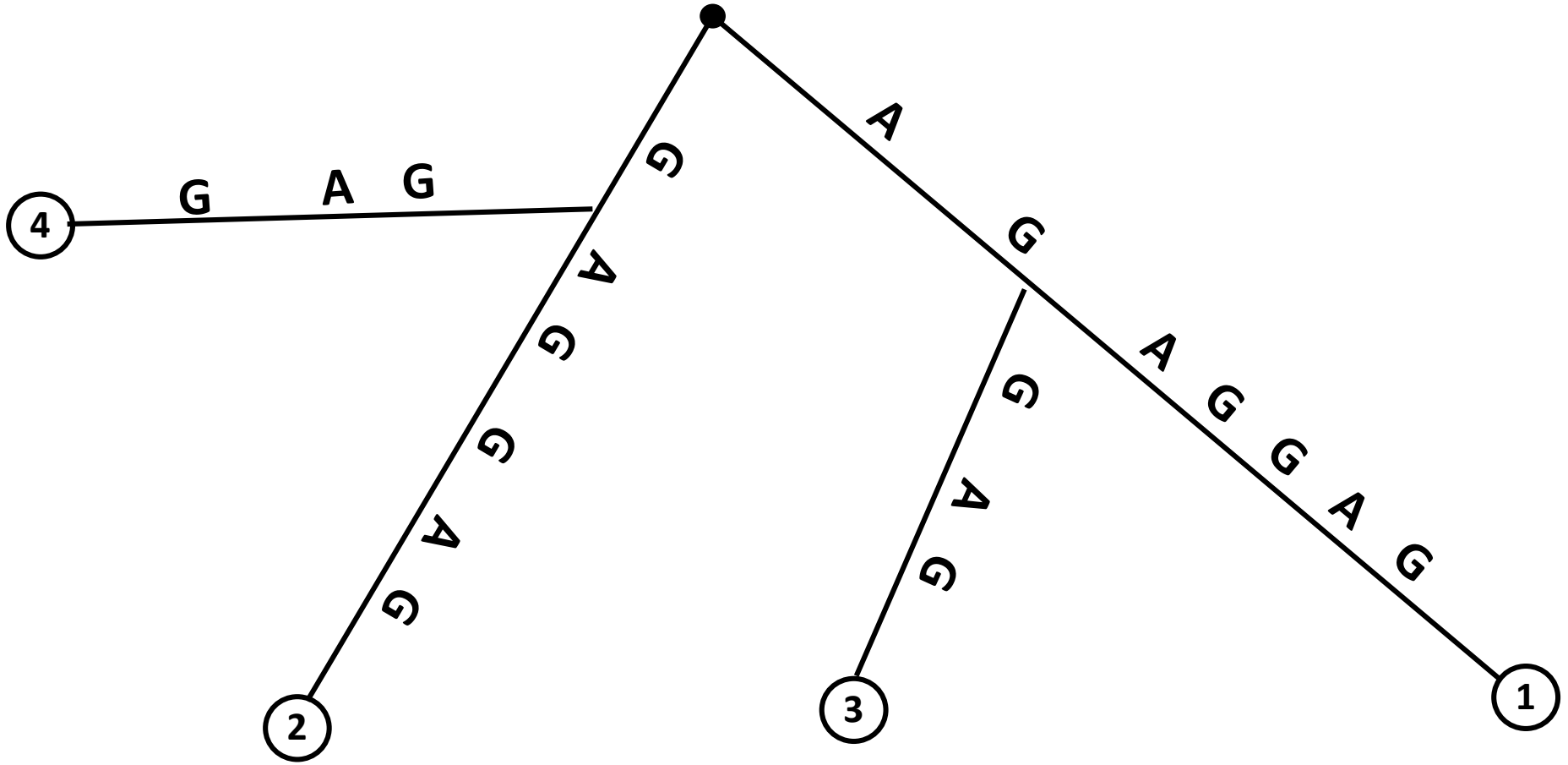
	1	2	3	4	5	6	7
S_1:	A	G	A	G	G	A	G

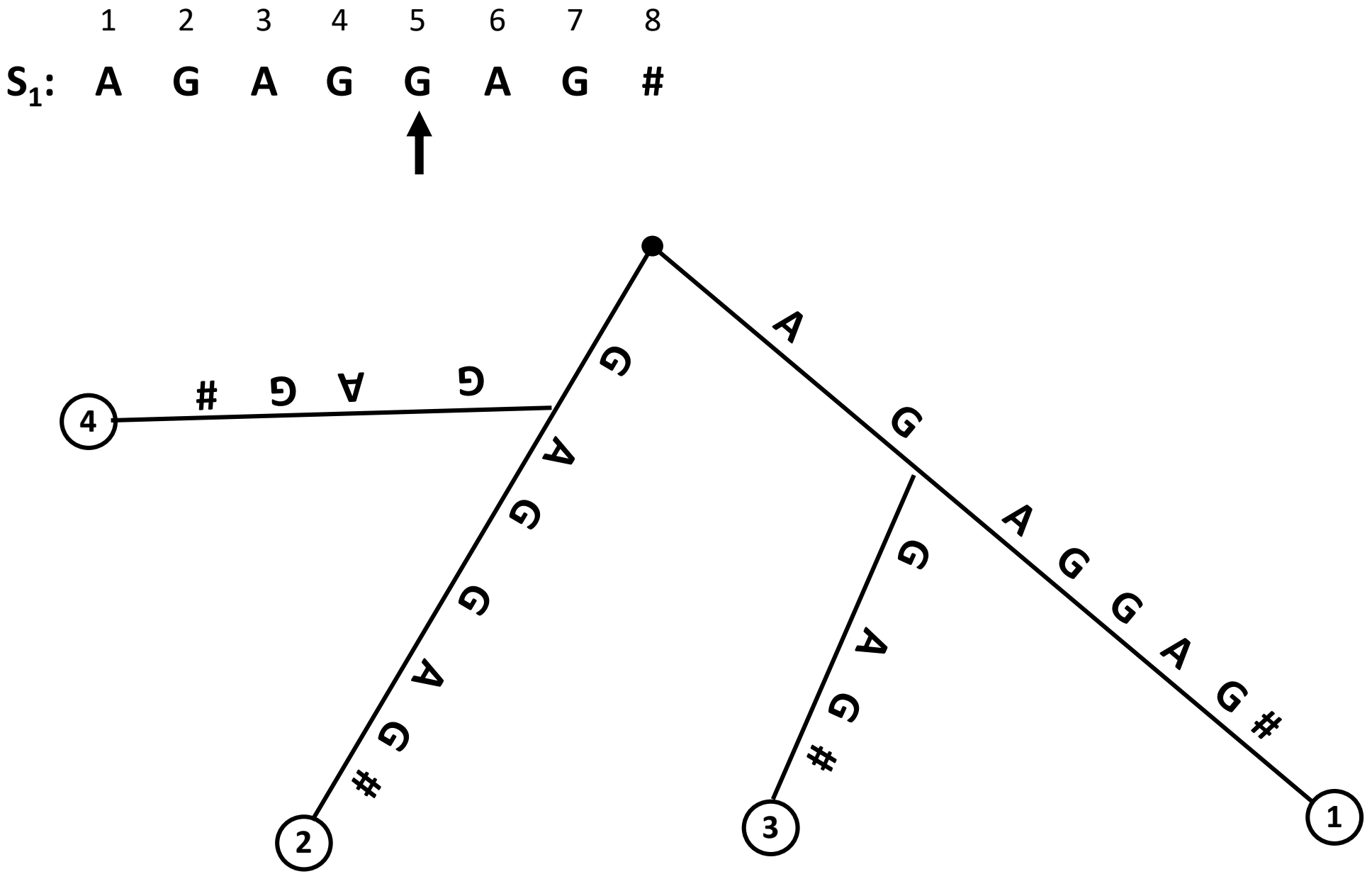


	1	2	3	4	5	6	7
S_1:	A	G	A	G	G	A	G

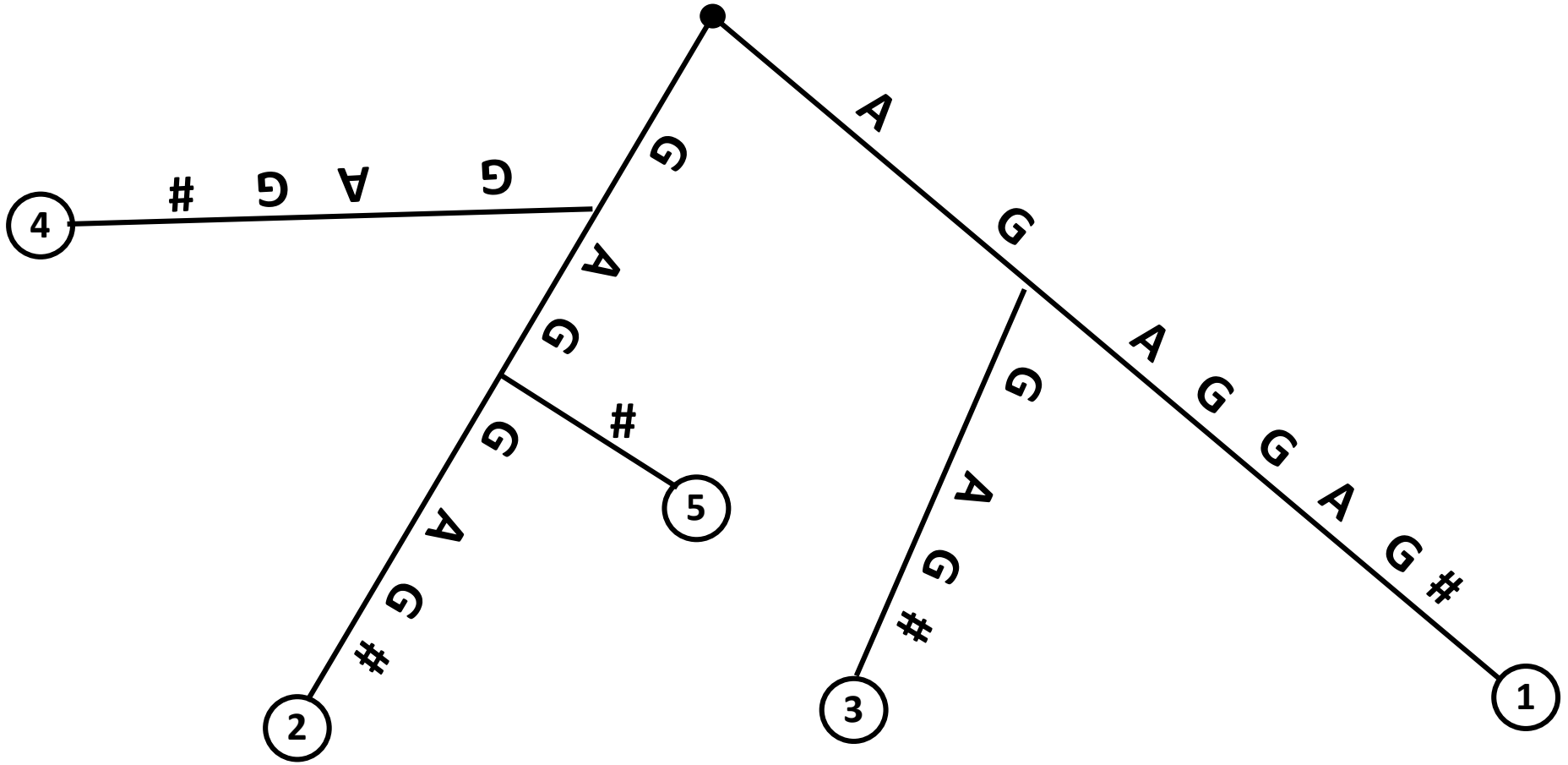


	1	2	3	4	5	6	7
S_1:	A	G	A	G	G	A	G

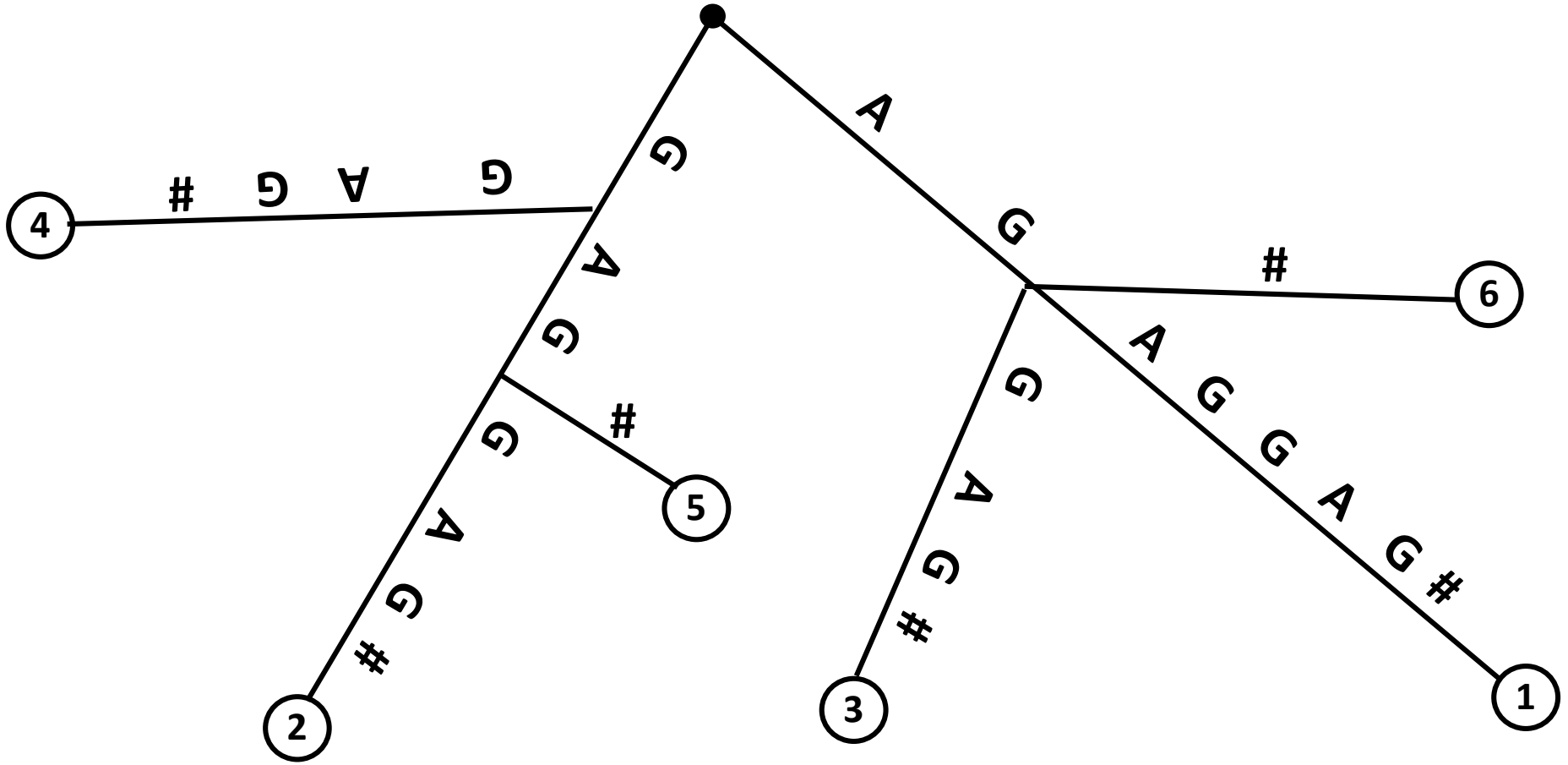



1 2 3 4 5 6 7 8
S1: A G A G G A G #
4
G A G
2
G A G G A G
3
G A G
1
A G G A G #

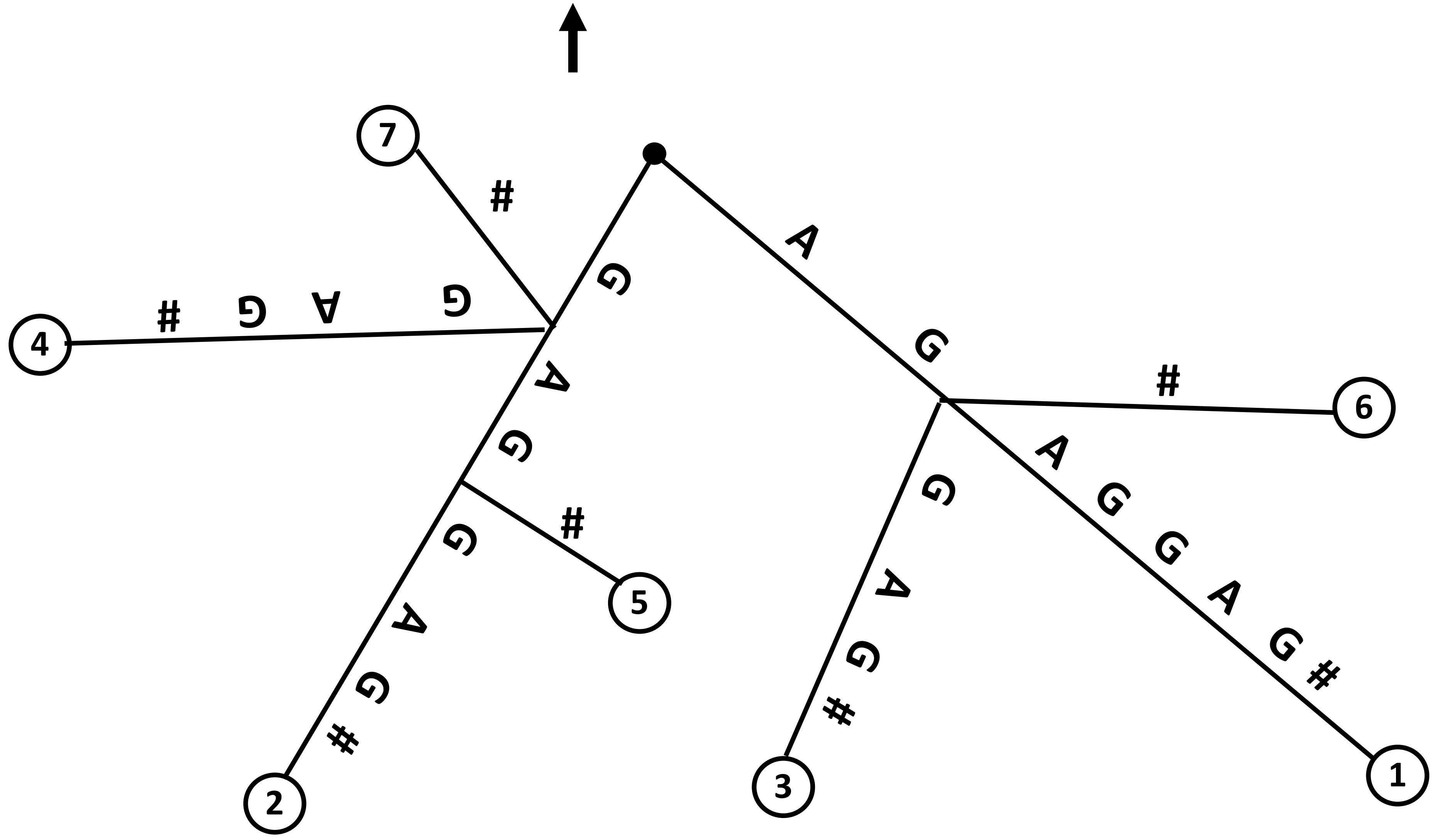
	1	2	3	4	5	6	7	8
S_1:	A	G	A	G	G	A	G	#



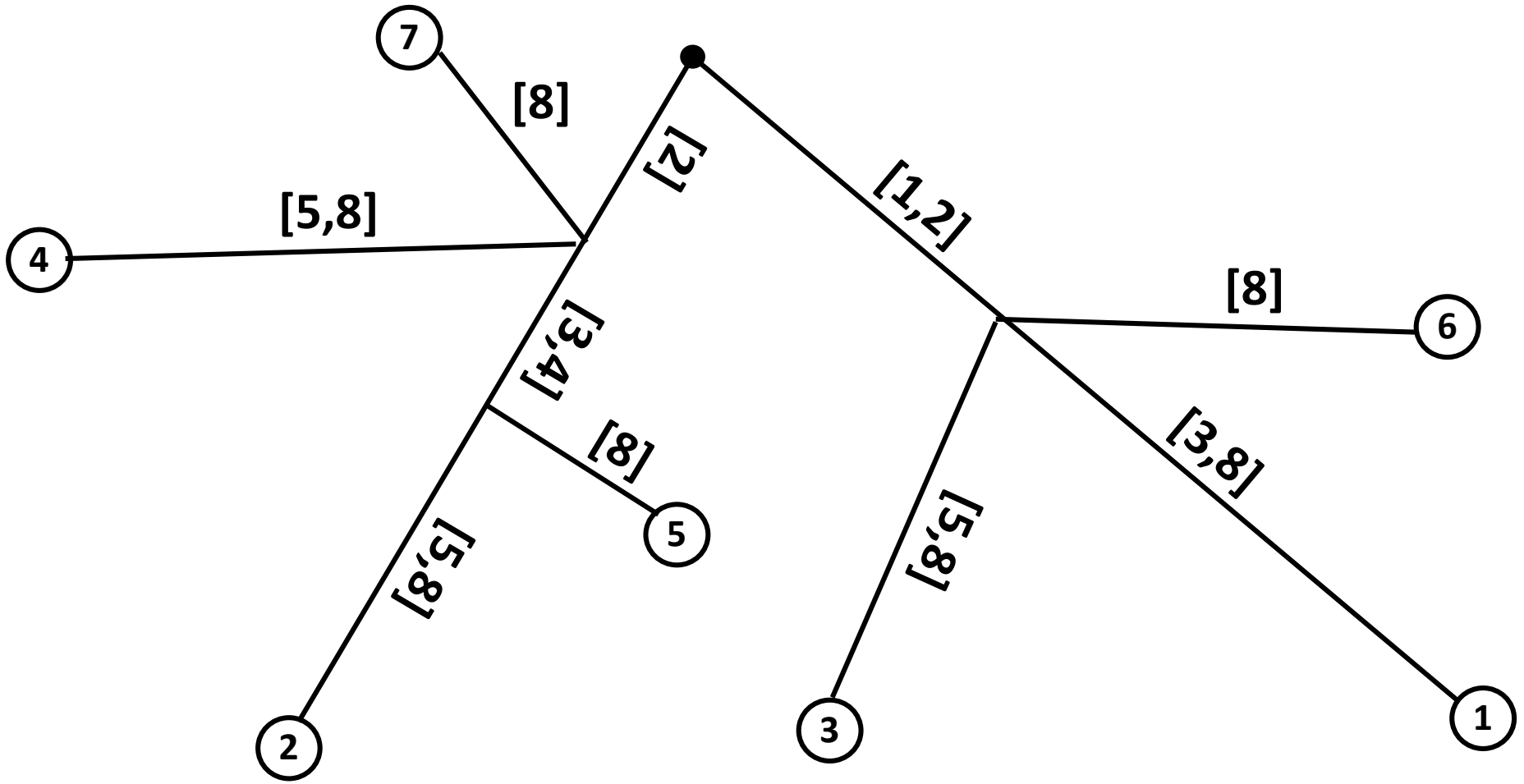
	1	2	3	4	5	6	7	8
S_1:	A	G	A	G	G	A	G	#



	1	2	3	4	5	6	7	8
S_1:	**A**	**G**	**A**	**G**	**G**	**A**	**G**	**#**



	1	2	3	4	5	6	7	8
S_1:	A	G	A	G	G	A	G	#



Construction de l'arbre des suffixes

Construction naïve:

Insertion successive des suffixes, du plus long au plus court

Exemple: Pour $P = aabbabb$, insérer successivement $aabbabb\$$, $abbabb\$$, $bbabb\$$, $\cdots$

Complexité $O(n^2)$

Construction de l'arbre des suffixes

Construction naïve:

Insertion successive des suffixes, du plus long au plus court

Exemple: Pour $P = aabbabb$, insérer successivement $aabbabb\$$, $abbabb\$$, $bbabb\$$, $\cdots$

Complexité $O(n^2)$

Constructions en temps linéaire:

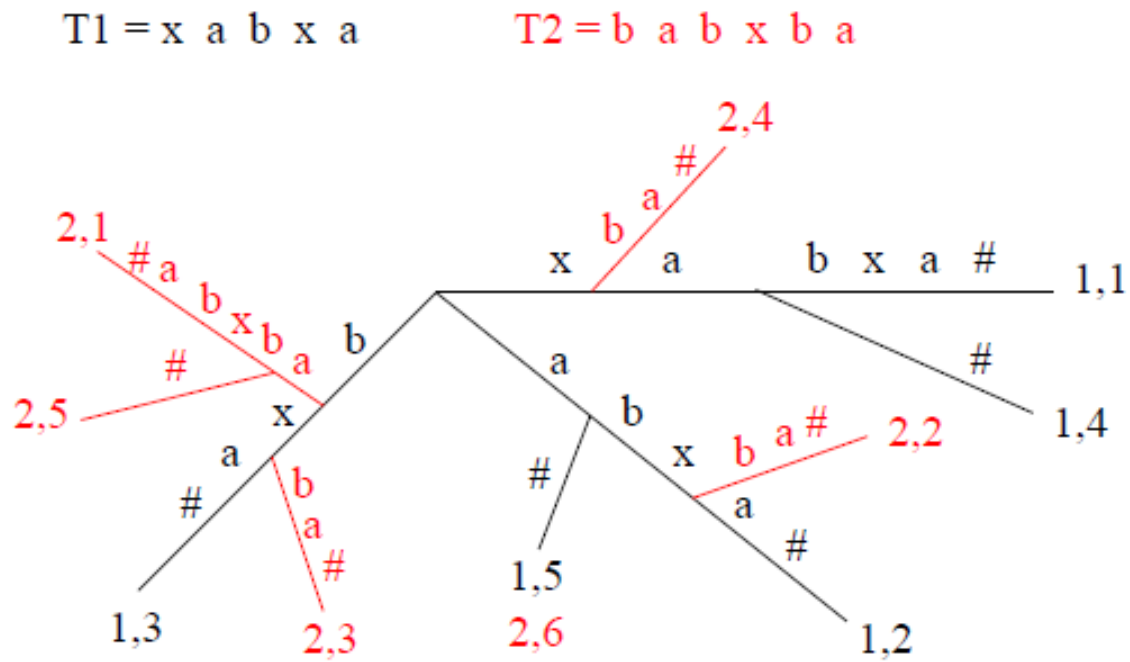
- Weiner – 1973: Premier algorithme linéaire.
- McCreight – 1976: Algorithme linéaire utilisant moins d'espace.
- Ukkonen – 1995: Algorithme linéaire, plus simple.

Arbre des suffixes généralisé

Arbre des suffixes pour un ensemble de séquences $\{S_1, S_2, \cdots, S_l\}$.
Simple extension de l'arbre des suffixes pour une séquence.

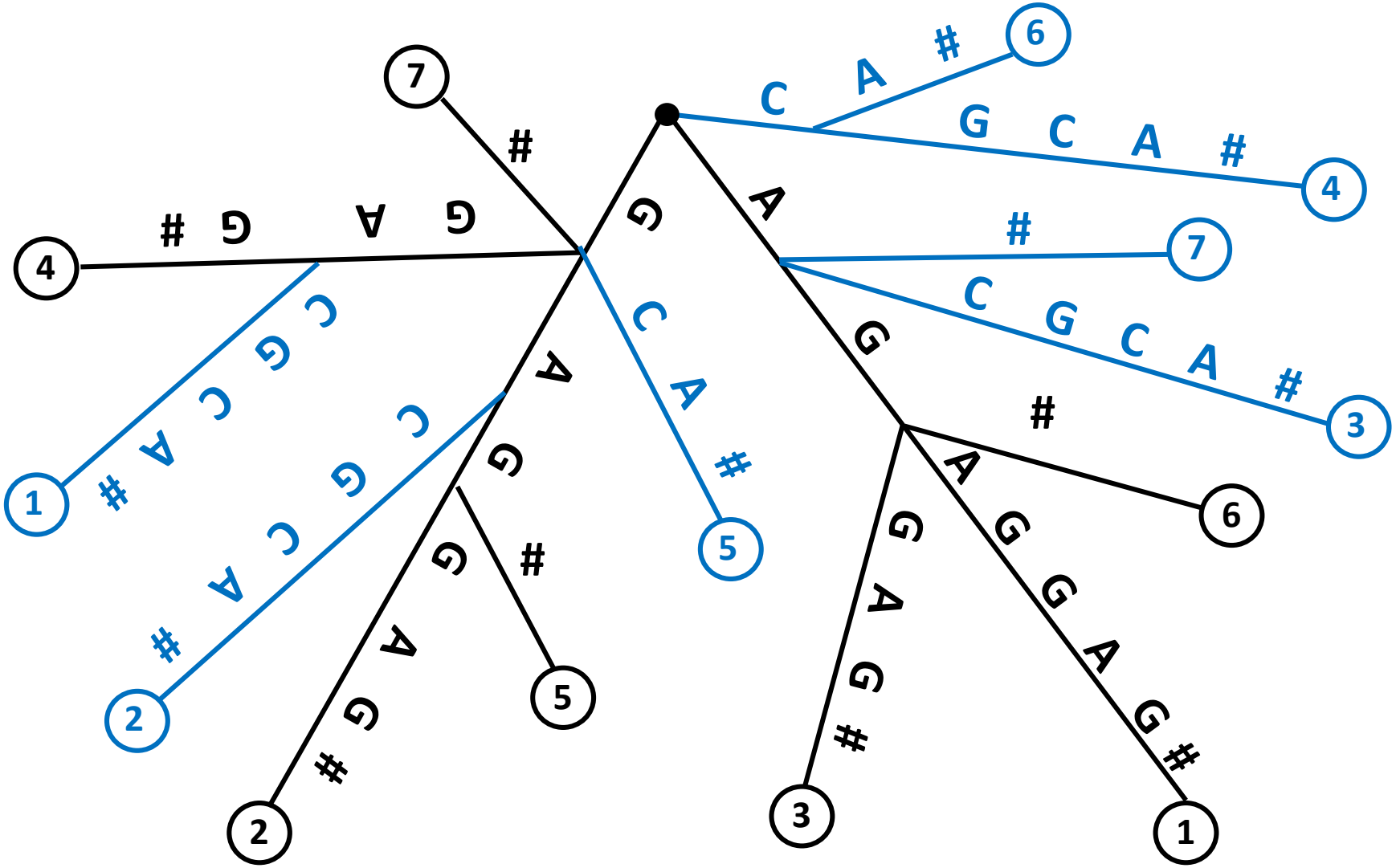
Deux subtilités:

- Étiquette d'une arête représentée par trois entiers plutôt que 2.
- Chaque feuille doit contenir les positions des suffixes dans les séquences concernées.



	1	2	3	4	5	6	7	8
S_1:	A	G	A	G	G	A	G	#

	1	2	3	4	5	6	7	8
S_2:	G	G	A	C	G	C	A	#



Différentes utilisations de l'arbre des suffixes

- Recherche exacte d'un mot dans un texte:

  Construction de l'arbre $\longrightarrow O(n)$

  Recherche de $P \longrightarrow O(m+k)$.

- Recherche multiple: $O(n+m+k)$

  Comparable à Aho-Corasick. Si les mots rentrés une fois pour toute et recherche dans des textes $\neq$, utiliser Aho-Corasick. Si texte constant et mots variables, utiliser l'arbre des suffixes.

Plus long facteur commun à deux séquences

Exemple: `ababbac` et `bbabbcab` $\longrightarrow$ `babb`

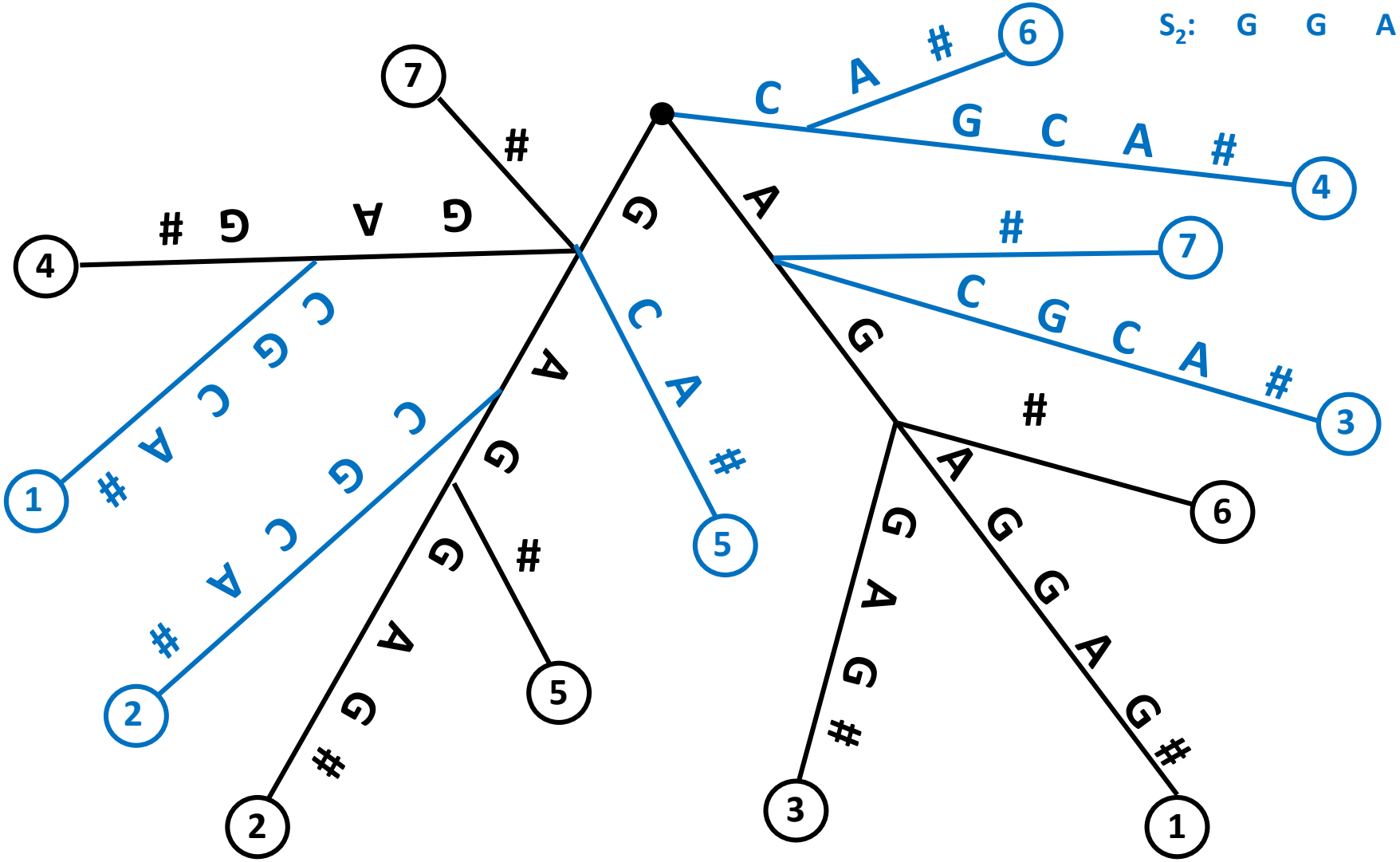
- Construire l'arbre des suffixe généralisé $\mathcal{S}$ pour $\{S_1, S_2\}$
- Marquer chaque nœud interne v: 1 (respec. 2) si il existe une feuille dans le sous-arbre de racine v qui corresponde à un suffixe de S_1 (respec. S_2), et marquer $(1, 2)$ un nœud vérifiant les deux conditions à la fois
- Trouver le nœud marqué $(1, 2)$ de profondeur en caractères maximale

Le marquage et le calcul de la profondeur en caractères peuvent être effectués par un algorithme standard de parcours d'arbre $\Longrightarrow$ Temps total en $O(n)$ où $n = |S_1| + |S_2|$.

En 1970 Knuth avait conjecturé: impossible de résoudre ce problème en temps linéaire.

Plus long facteur commun

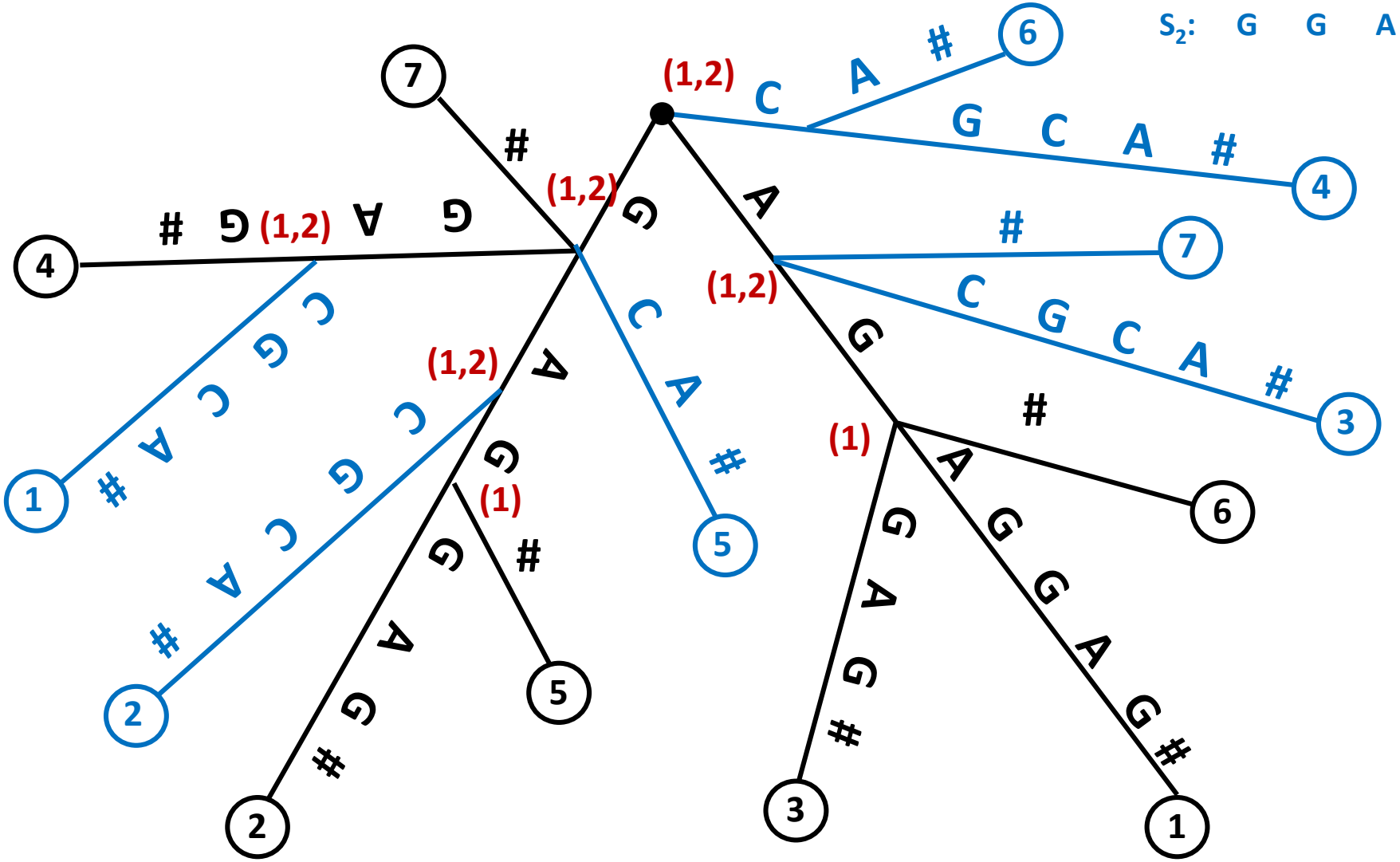
	1	2	3	4	5	6	7	8
S_1:	A	G	A	G	G	A	G	#
	1	2	3	4	5	6	7	8
S_2:	G	G	A	C	G	C	A	#



Plus long facteur commun

Marquer les nœuds internes

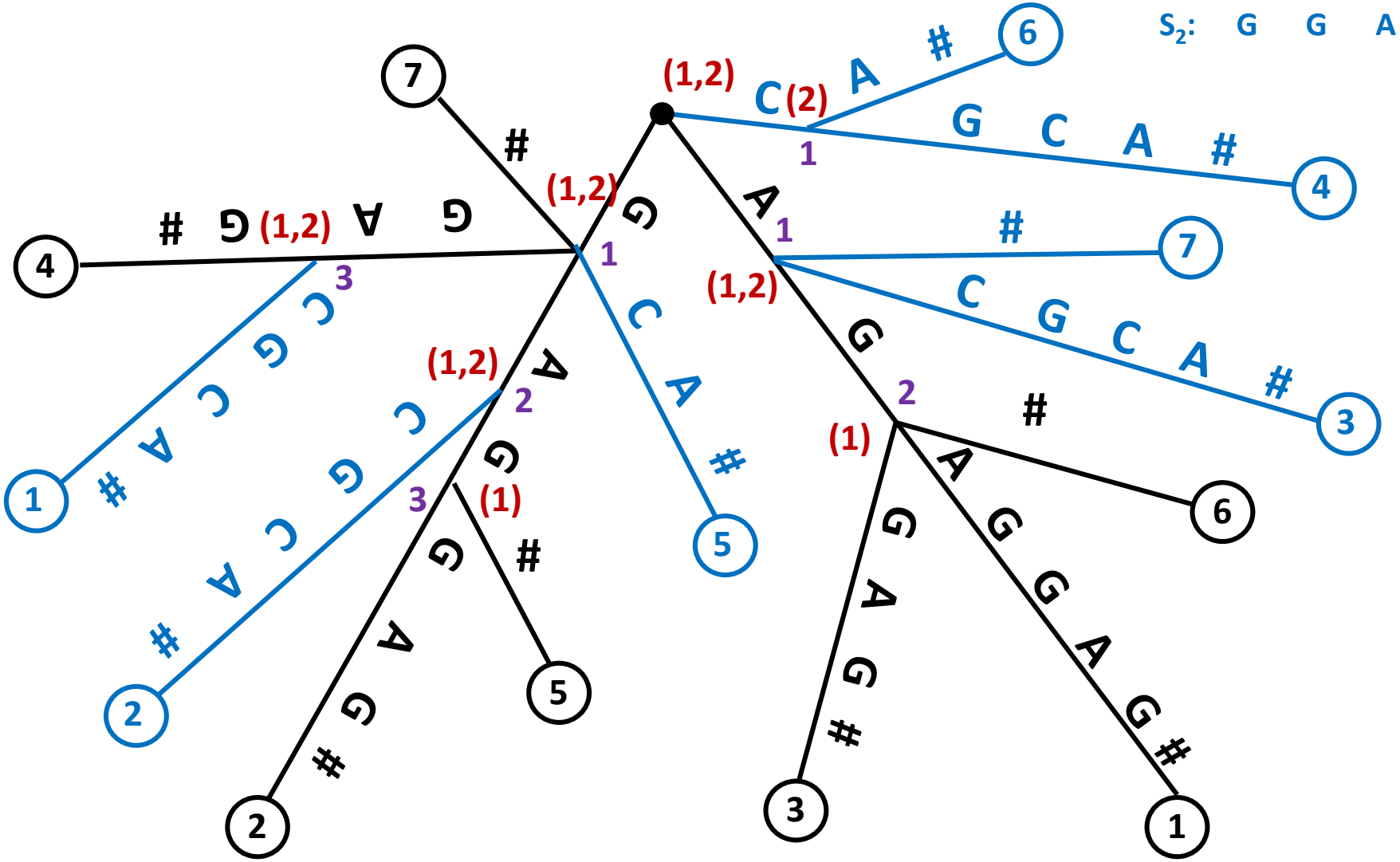
	1	2	3	4	5	6	7	8
S_1:	A	G	A	G	G	A	G	#
	1	2	3	4	5	6	7	8
S_2:	G	G	A	C	G	C	A	#



Plus long facteur commun

Marquer les nœuds internes et profondeur en caractères

	1	2	3	4	5	6	7	8
S_1:	A	G	A	G	G	A	G	#
	1	2	3	4	5	6	7	8
S_2:	G	G	A	C	G	C	A	#

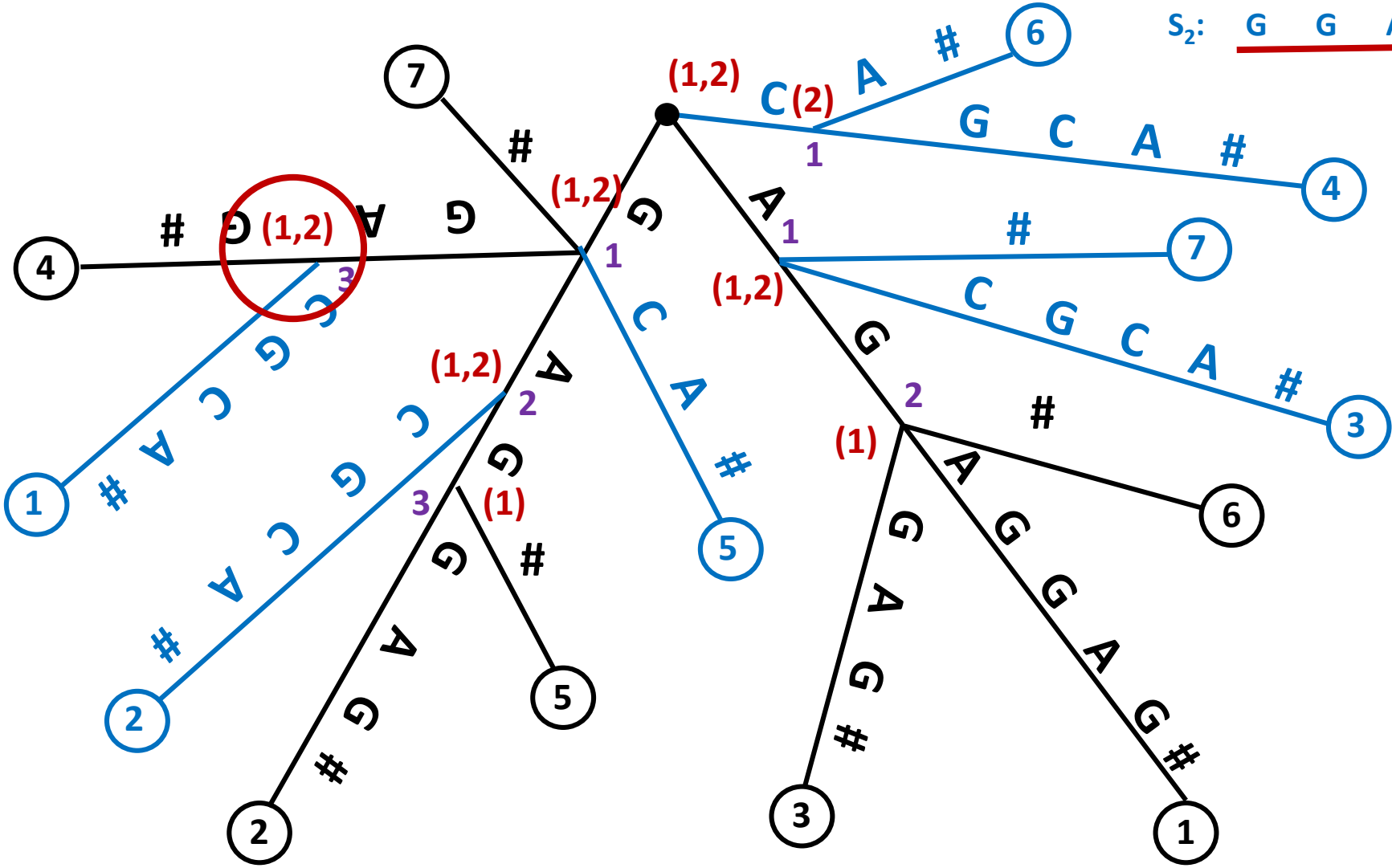


Plus long facteur commun

Marquer les nœuds internes et profondeur en caractères

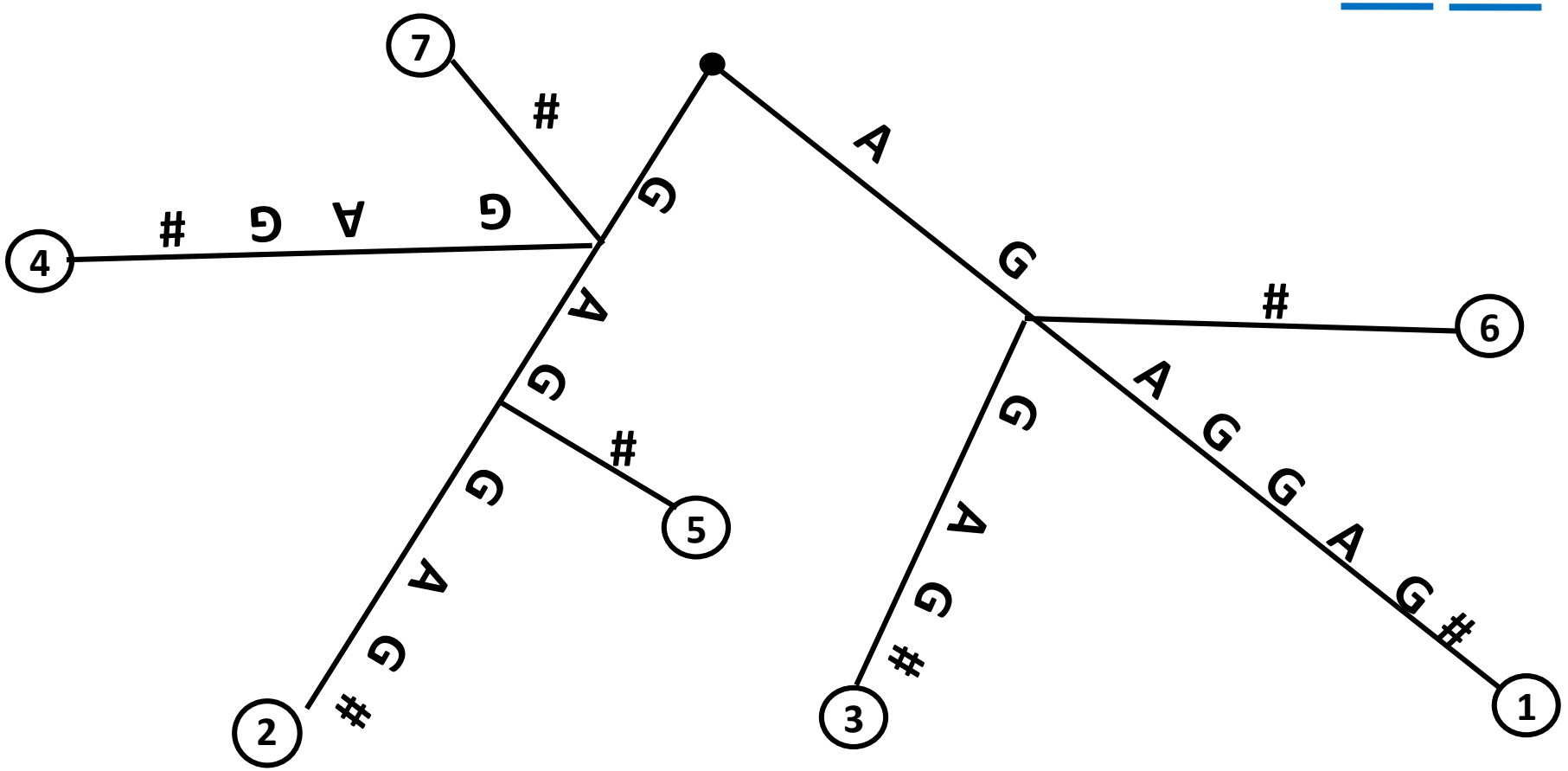
Prendre nœuds marqués (1,2) de profondeur en caractère max

	1	2	3	4	5	6	7	8
S_1:	A	G	A	G	G	A	G	#
	1	2	3	4	5	6	7	8
S_2:	G	G	A	C	G	C	A	#



Répétitions maximales et supermaximales

	1	2	3	4	5	6	7	8
S_1:	A	G	A	G	G	A	G	#



Recherche de répétitions maximales

Répétition maximale α: $\exists$ deux positions de α dans S, et à ces deux positions, si on prolonge à droite ou à gauche, plus une répétition.

Si α répétition maximale de S, alors α étiquette d'un nœud de l'arbre des suffixe de S. Mais tout nœud interne ne représente pas une répétition maximale

Au plus n répétitions maximales car au plus n nœuds internes.

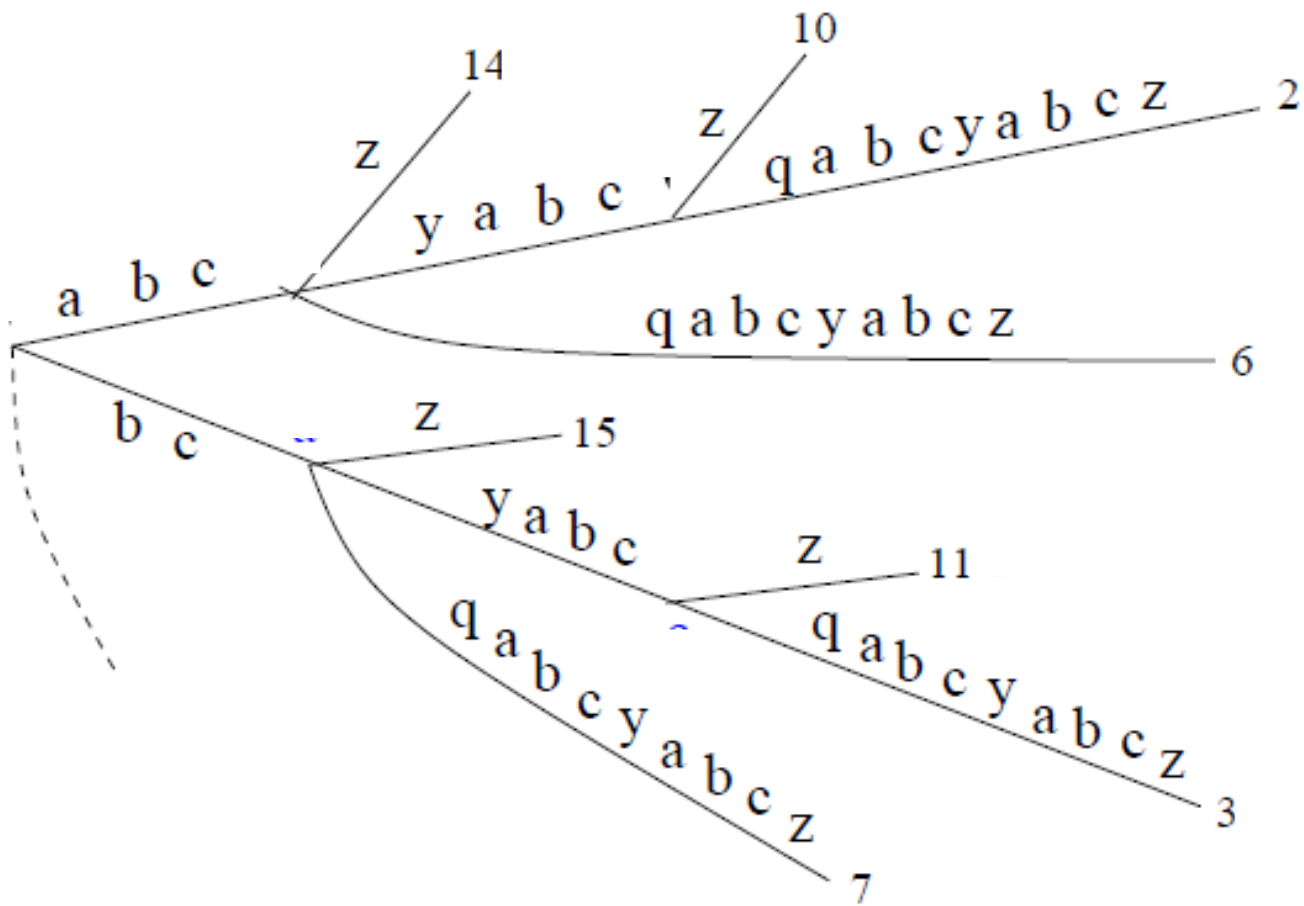
f: Feuille représentant $S[i..n]$

$S[i-1]$: Caractère gauche de f

Nœud interne v "left diverse" si au moins deux feuilles du sous-arbre de racine v ont des caractères gauches différents. Si v left diverse alors tous les ancêtres de v sont left diverse.

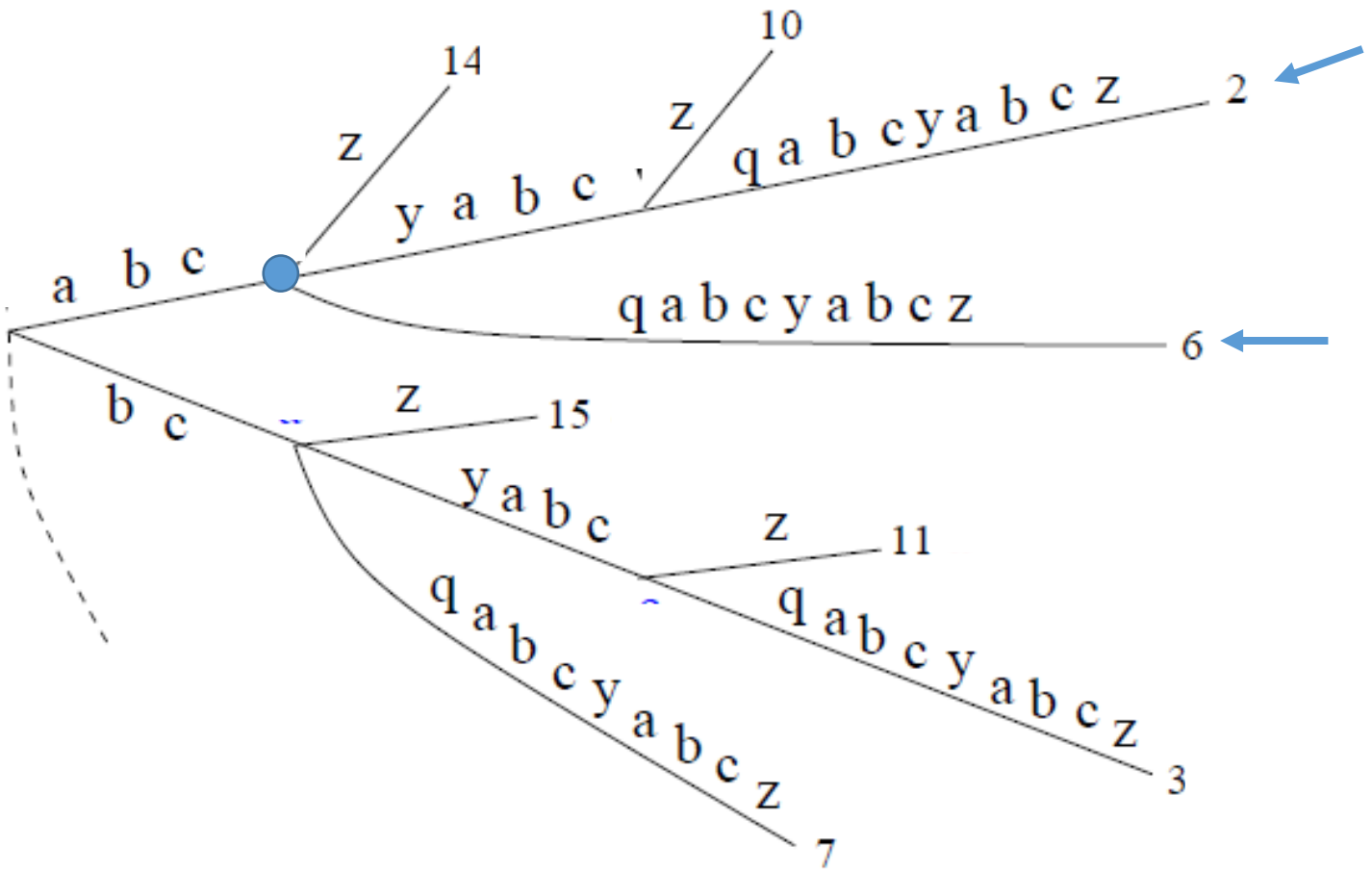
Théorème: Un mot α étiquette de v est une répétition maximale ssi v est left diverse.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
T =	x	a	b	c	y	a	b	c	q	a	b	c	y	a	b	c	z



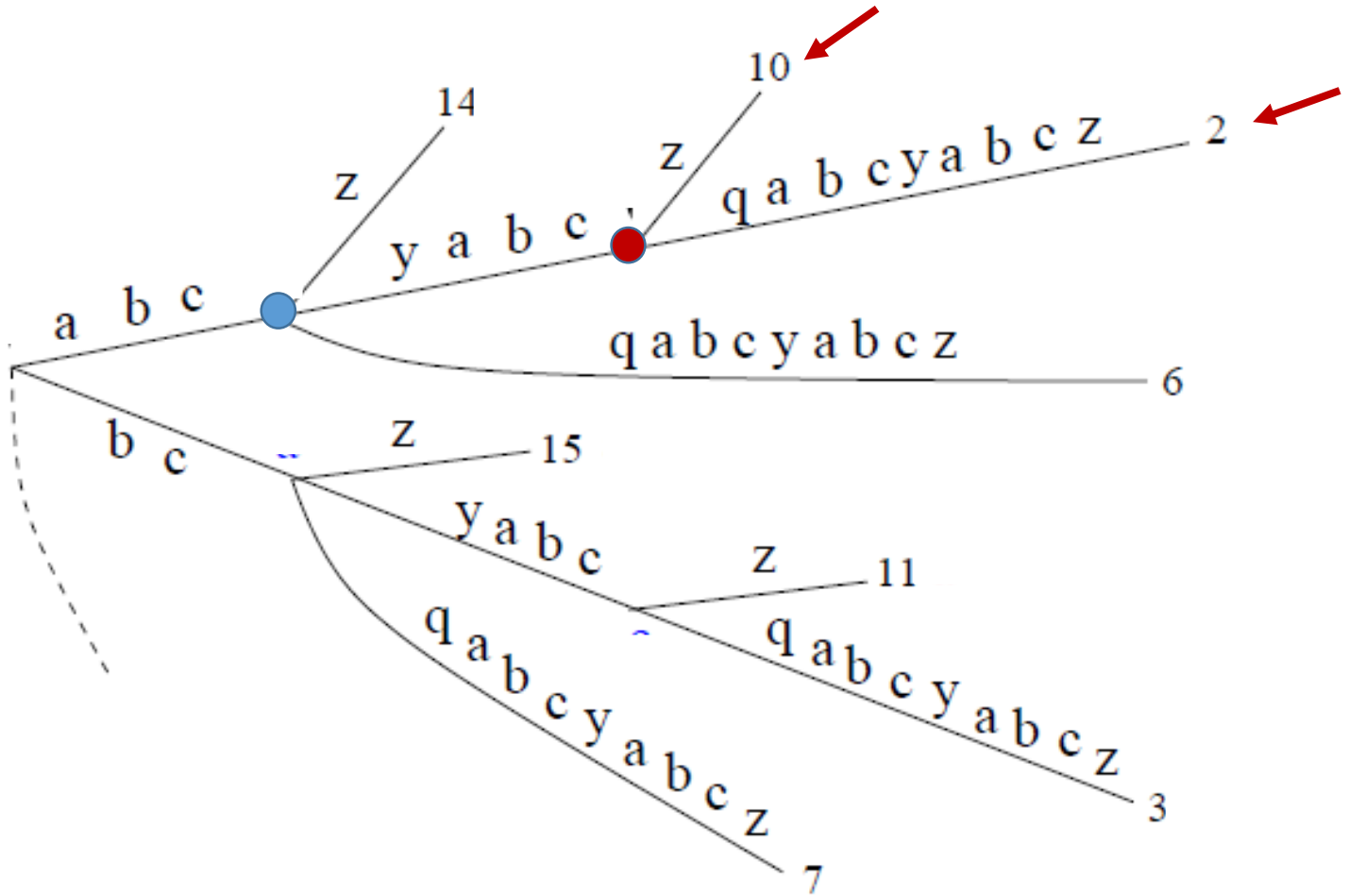
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
T =	x	a	b	c	y	a	b	c	q	a	b	c	y	a	b	c	z

Répétitions maximales



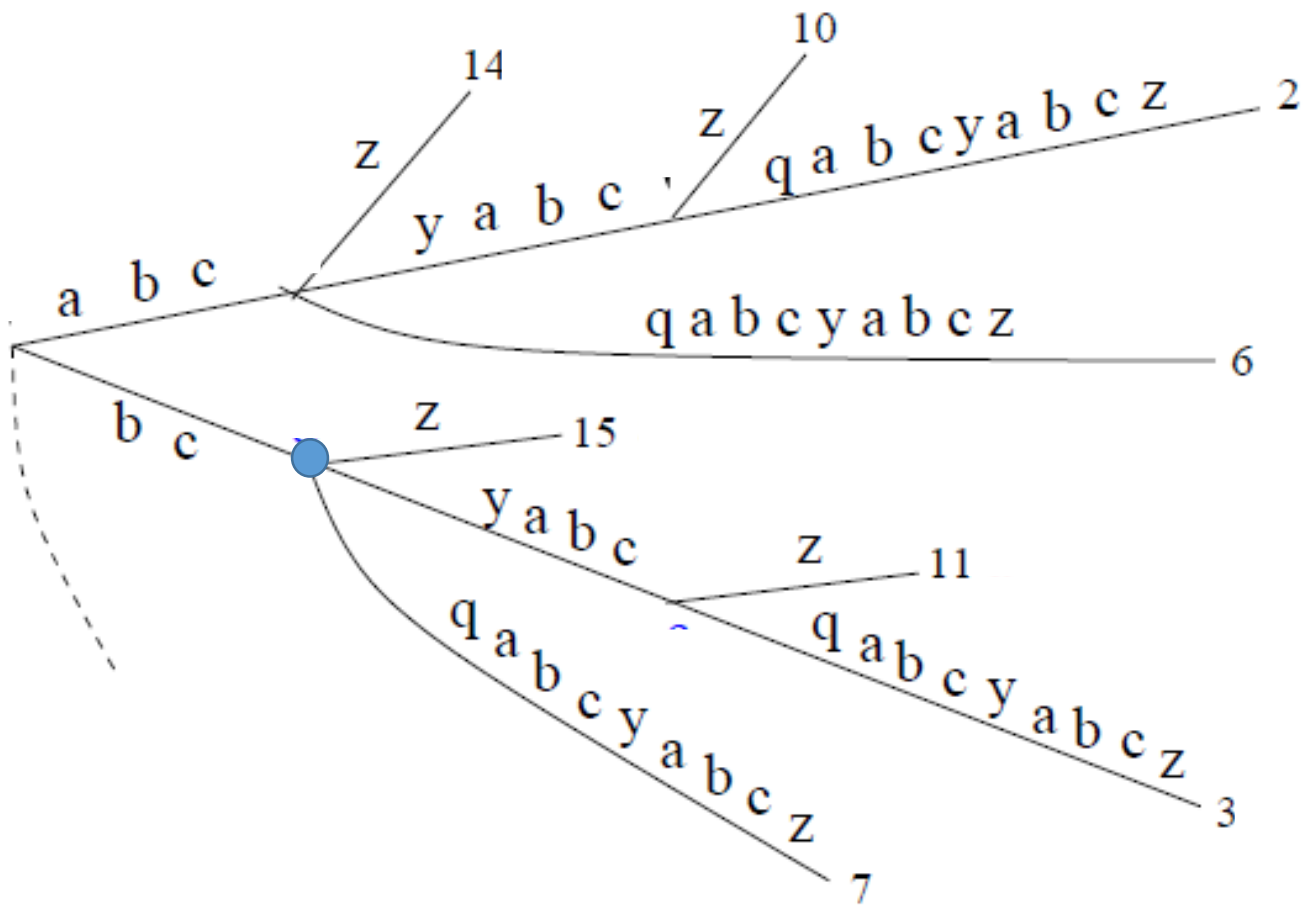
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
T =	x	a	b	c	y	a	b	c	q	a	b	c	y	a	b	c	z

Répétitions maximales



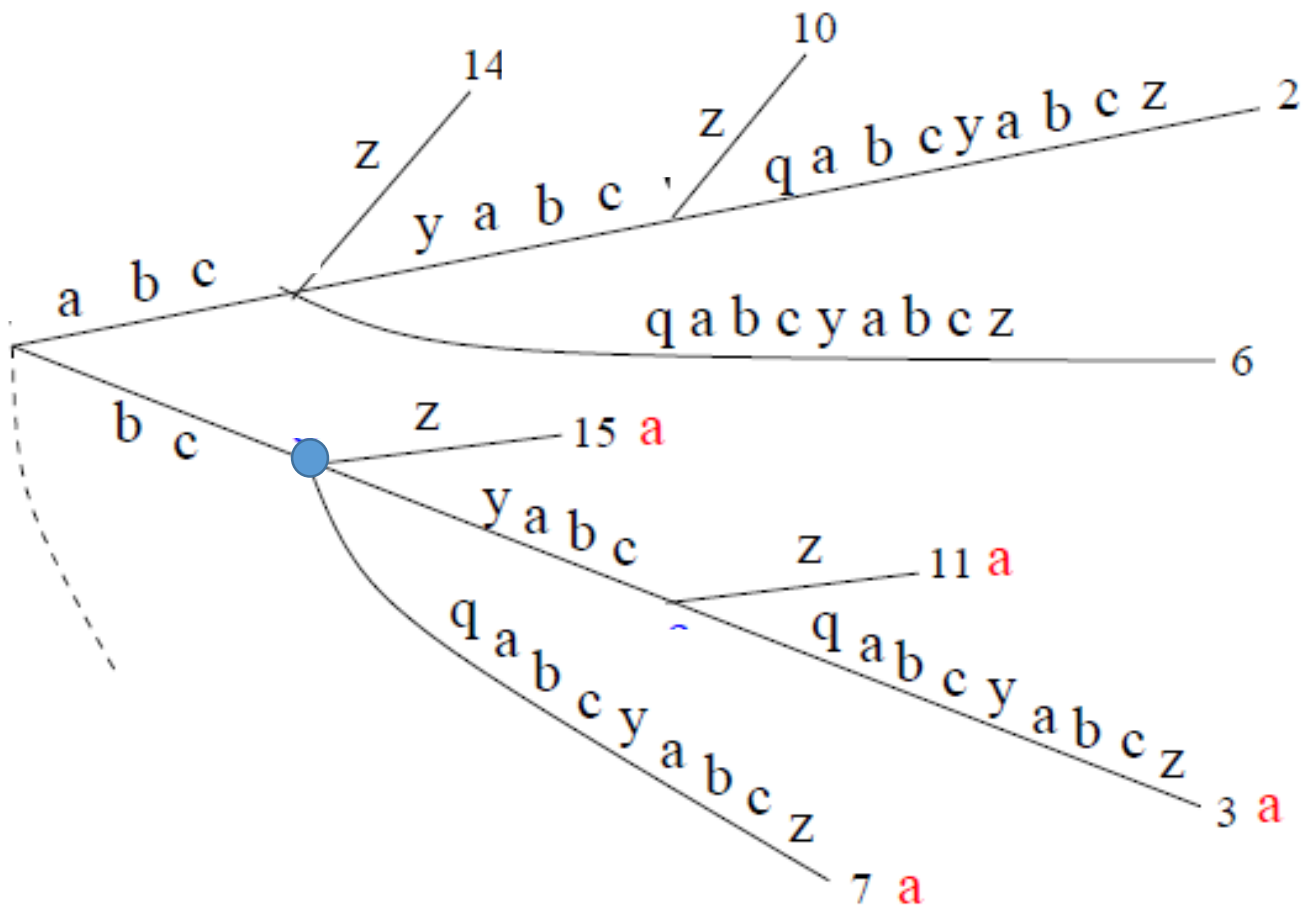
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
T =	x	a	b	c	y	a	b	c	q	a	b	c	y	a	b	c	z

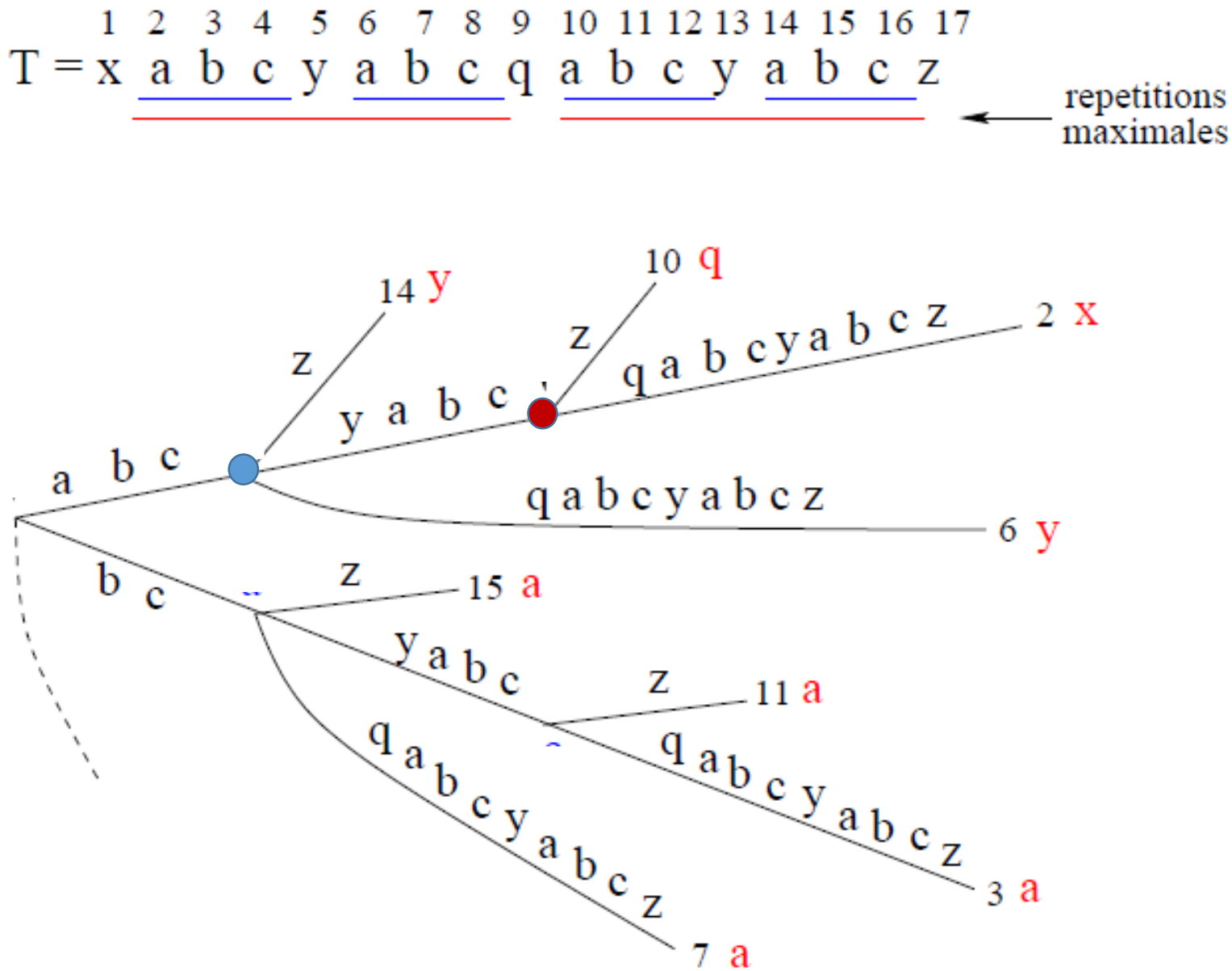
Pas répétition maximale



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
T =	x	a	b	c	y	a	b	c	q	a	b	c	y	a	b	c	z

Pas répétition maximale

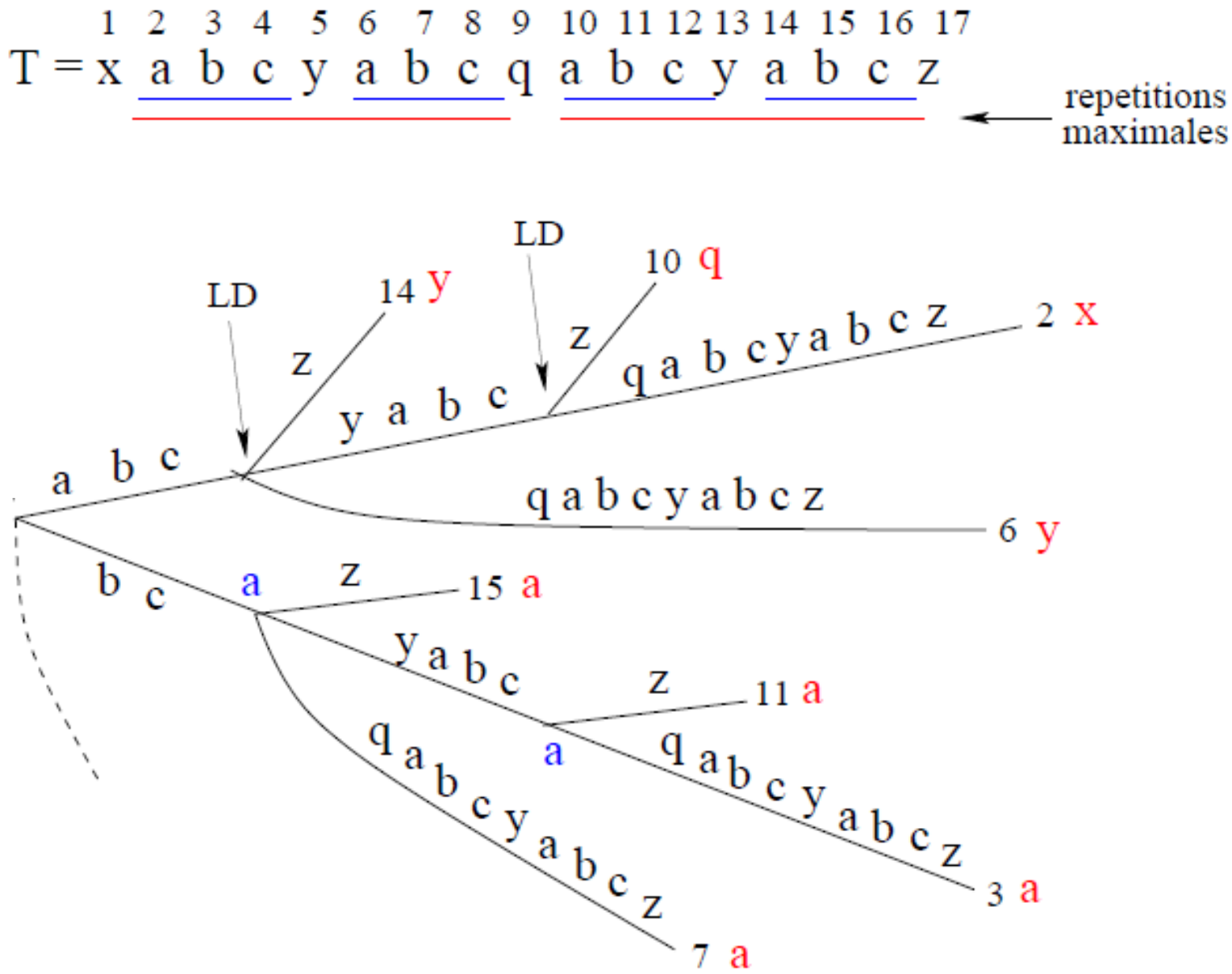



1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
T = x a b c y a b c q a b c y a b c z
repetitions maximales
14 y
10 q
z
z
y a b c
q a b c y a b c z
2 x
a b c
q a b c y a b c z
6 y
b c
z
15 a
y a b c
z
11 a
q a b c y a b c z
3 a
q a b c y a b c z
7 a

Recherche des nœuds left diverse:

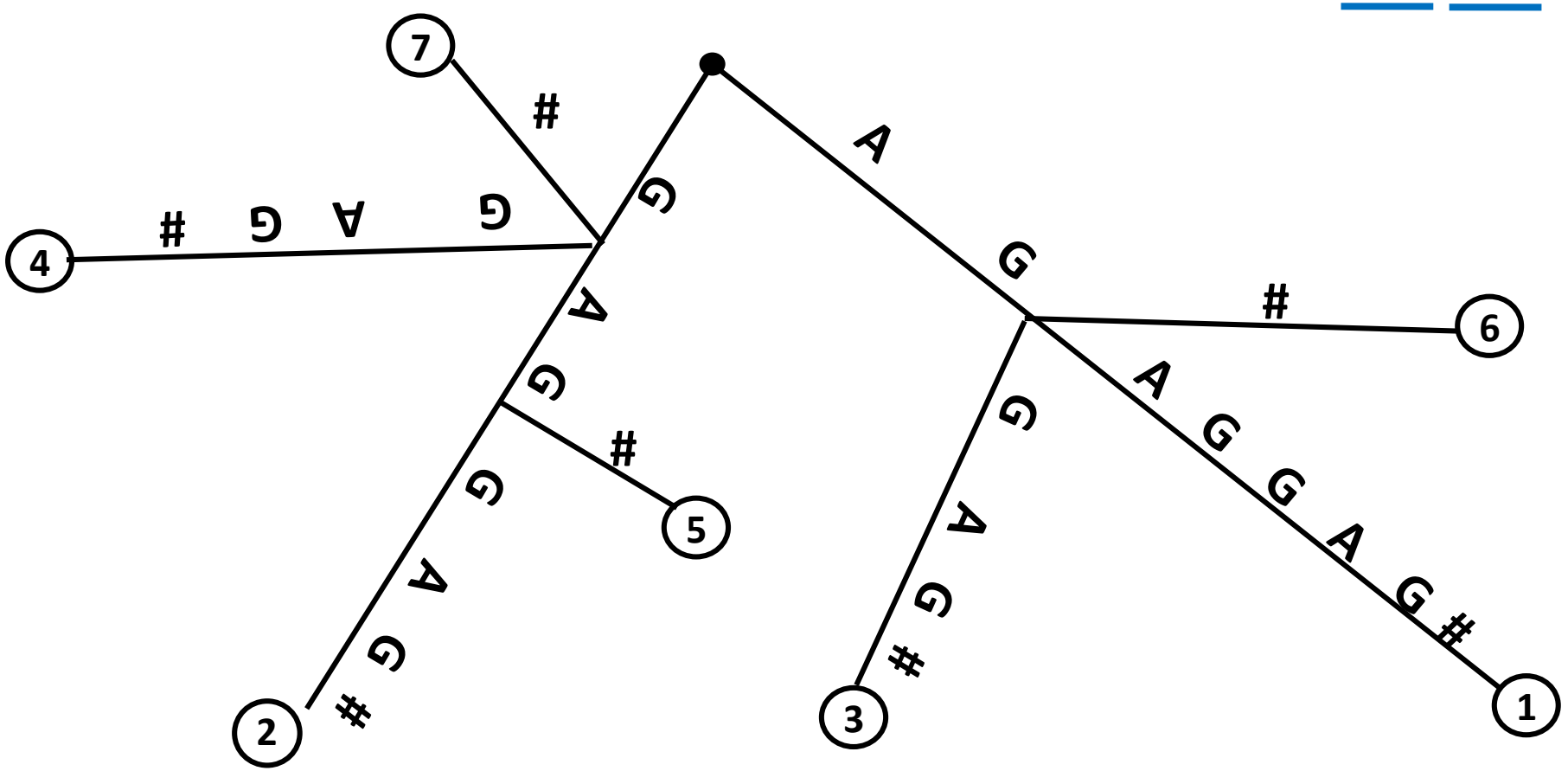
- Pour chaque feuille, conserver son caractère gauche
- Examiner les nœuds internes de bas en haut (des feuilles à la racine)
  - Si au moins un fils de v left diverse alors v left-diverse
  - Sinon, considérer les caractères gauches des fils de v
    - Si un même caractère x, alors x caractère gauche de v
    - Si au moins deux caractères différents, alors v est left diverse

Théorème: Les répétitions maximales d'une séquence S peuvent être trouvées en temps $O(n)$


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
T = x a b c y a b c q a b c y a b c z
repetitions maximales
LD
LD
14 y
10 q
z
z
2 x
y a b c
q a b c y a b c z
a b c
q a b c y a b c z
6 y
b c
a
z
15 a
y a b c
z
11 a
a
q a b c y a b c z
3 a
q a b c y a b c z
7 a

Répétitions maximales

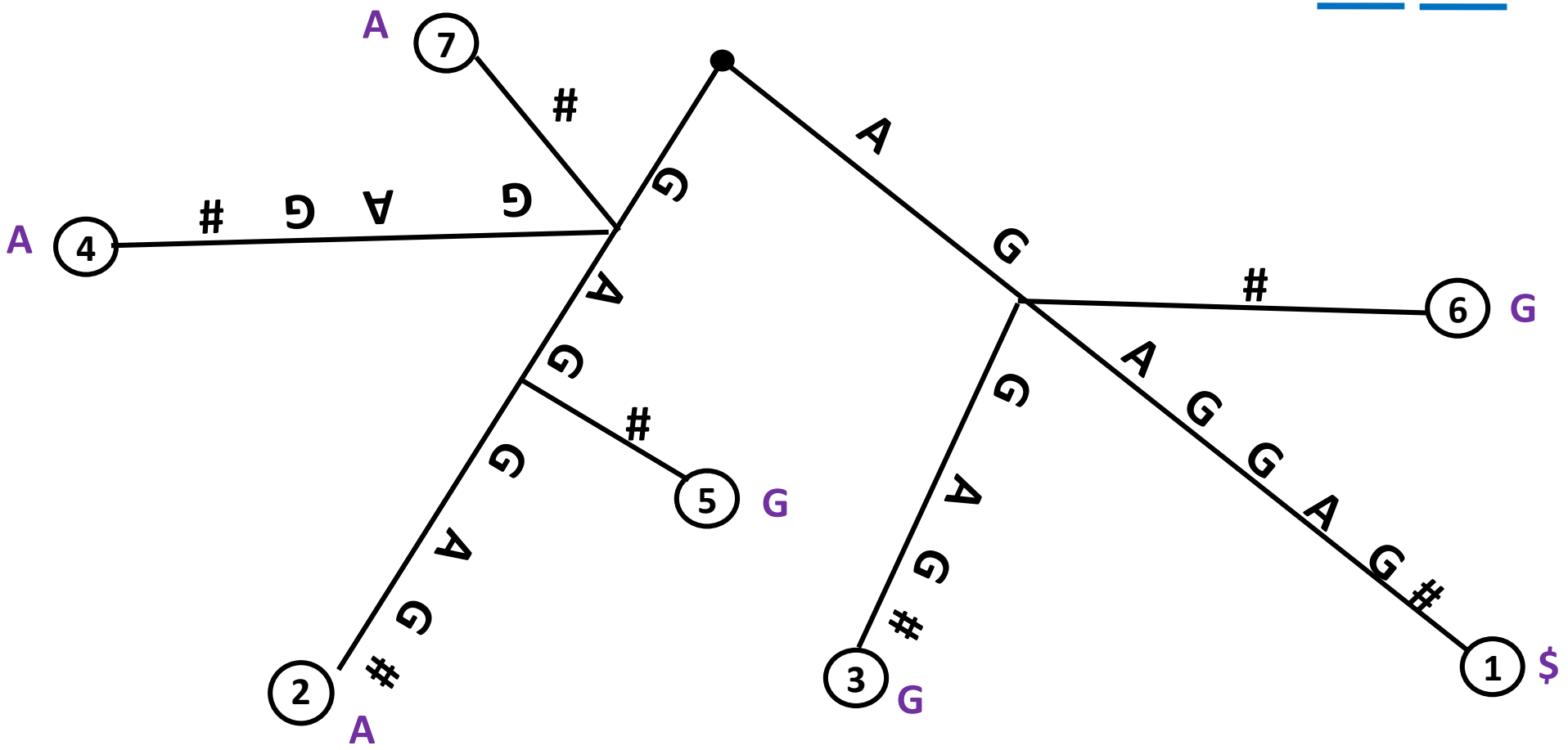
	1	2	3	4	5	6	7	8
S_1:	A	G	A	G	G	A	G	#



Répétitions maximales

1. Étiqueter les feuilles par le caractère gauche

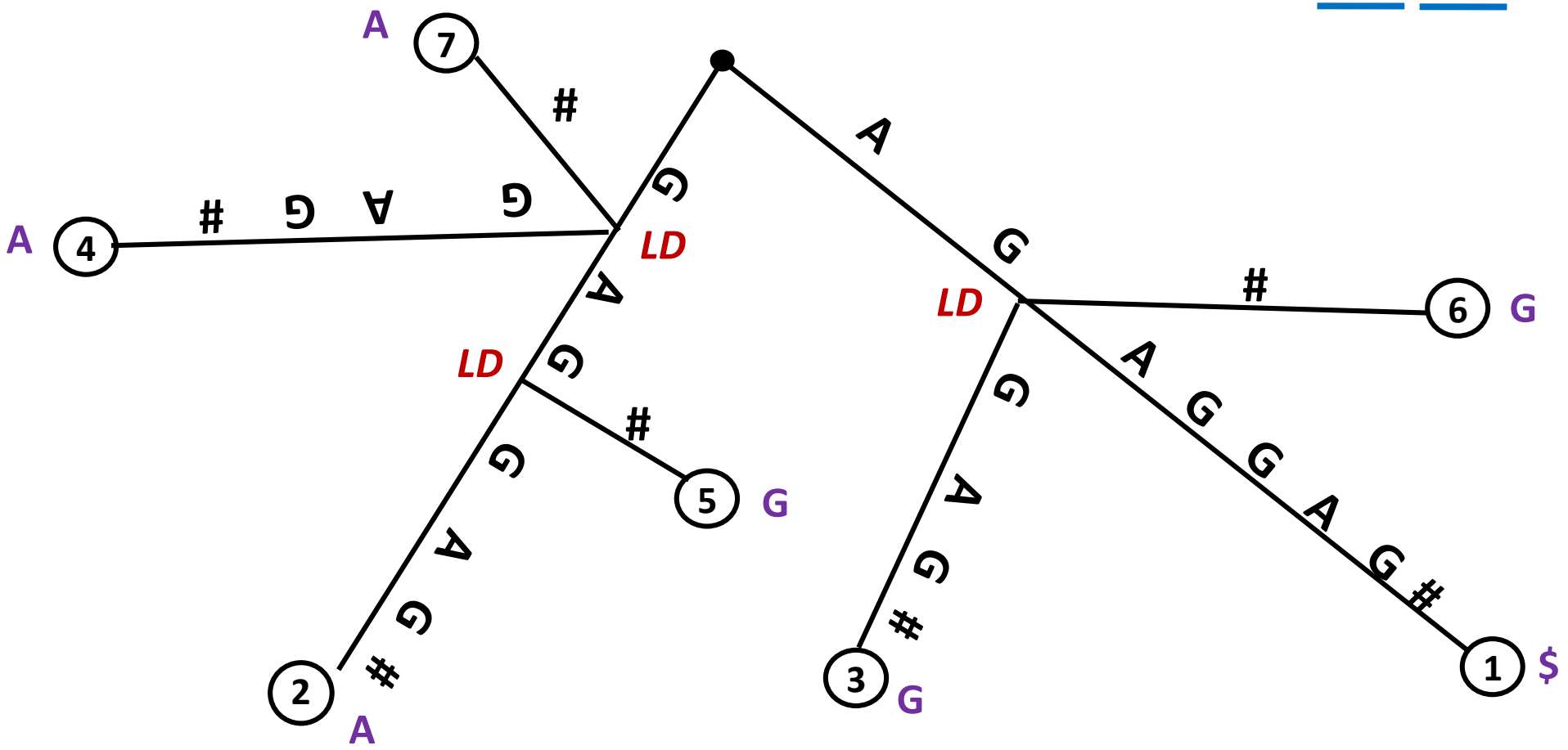
	1	2	3	4	5	6	7	8
S_1:	A	G	A	G	G	A	G	#



Répétitions maximales

1. Étiqueter les feuilles par le caractère gauche
2. ***Étiqueter les nœuds internes***

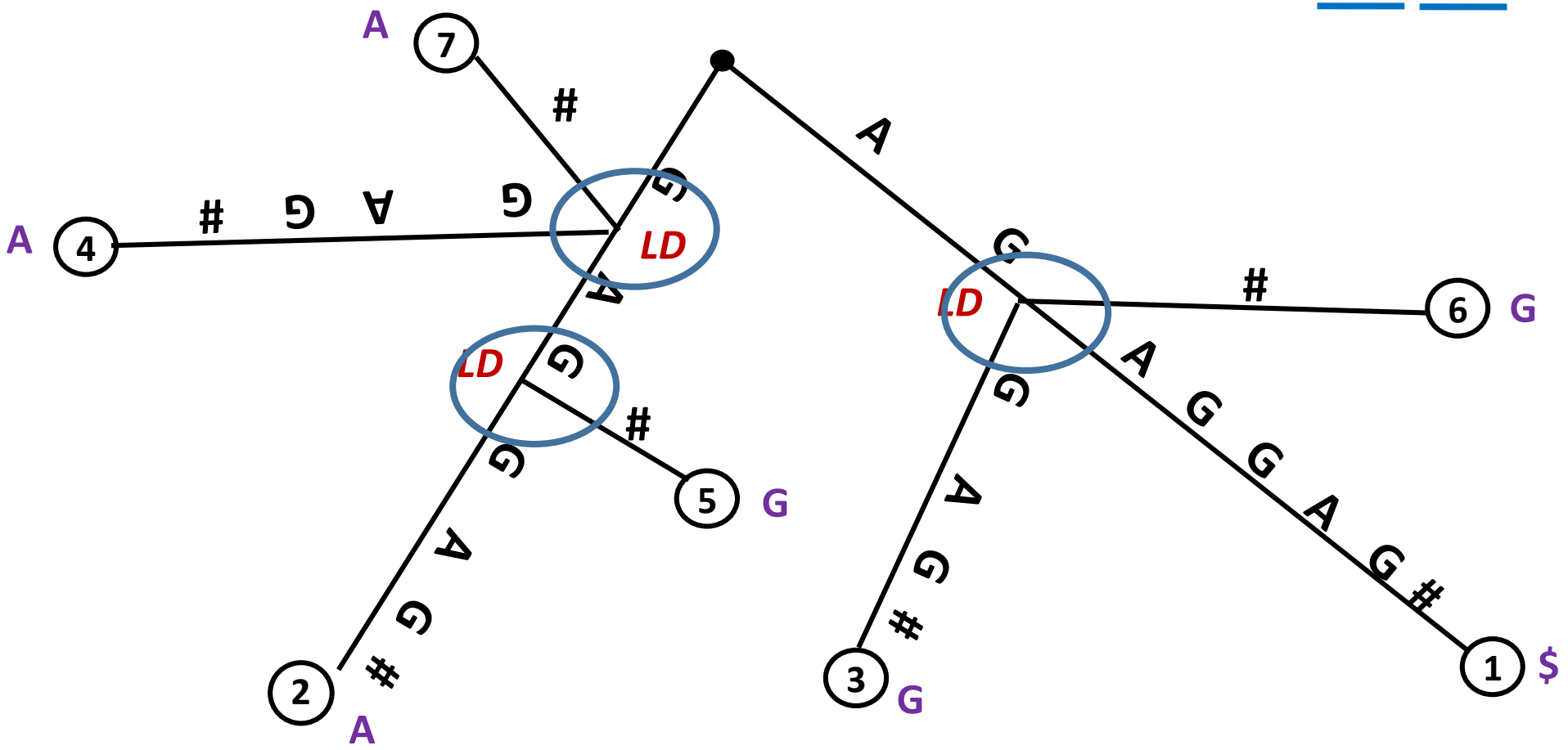
	1	2	3	4	5	6	7	8
S_1:	A	G	A	G	G	A	G	#



Répétitions maximales

1. Étiqueter les feuilles par le caractère gauche
2. ***Étiqueter les nœuds internes***
3. **Les nœuds LD correspondent aux répétitions max**

	1	2	3	4	5	6	7	8
S_1:	A	G	A	G	G	A	G	#

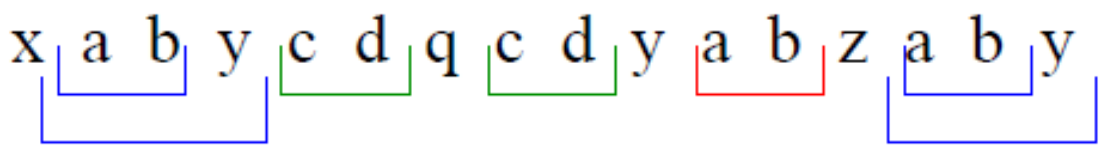


Recherche de répétitions supermaximales

Répétition supermaximale: Répétition maximale qui n'est facteur d'aucune autre répétition maximale

Répétition presque supermaximale: Répétition maximale qui apparaît à au moins une position dans S où elle n'est pas facteur d'une autre répétition maximale: position témoin.

Exemple



`c d`: Répétition supermaximale; `a b`: Répétition presque sm.
Position témoin pour `a b`: 2^{eme} position.

Répétitions presque-supermaximales

	1	2	3	4	5	6	7	8
S_1:	A	G	A	G	G	A	G	#

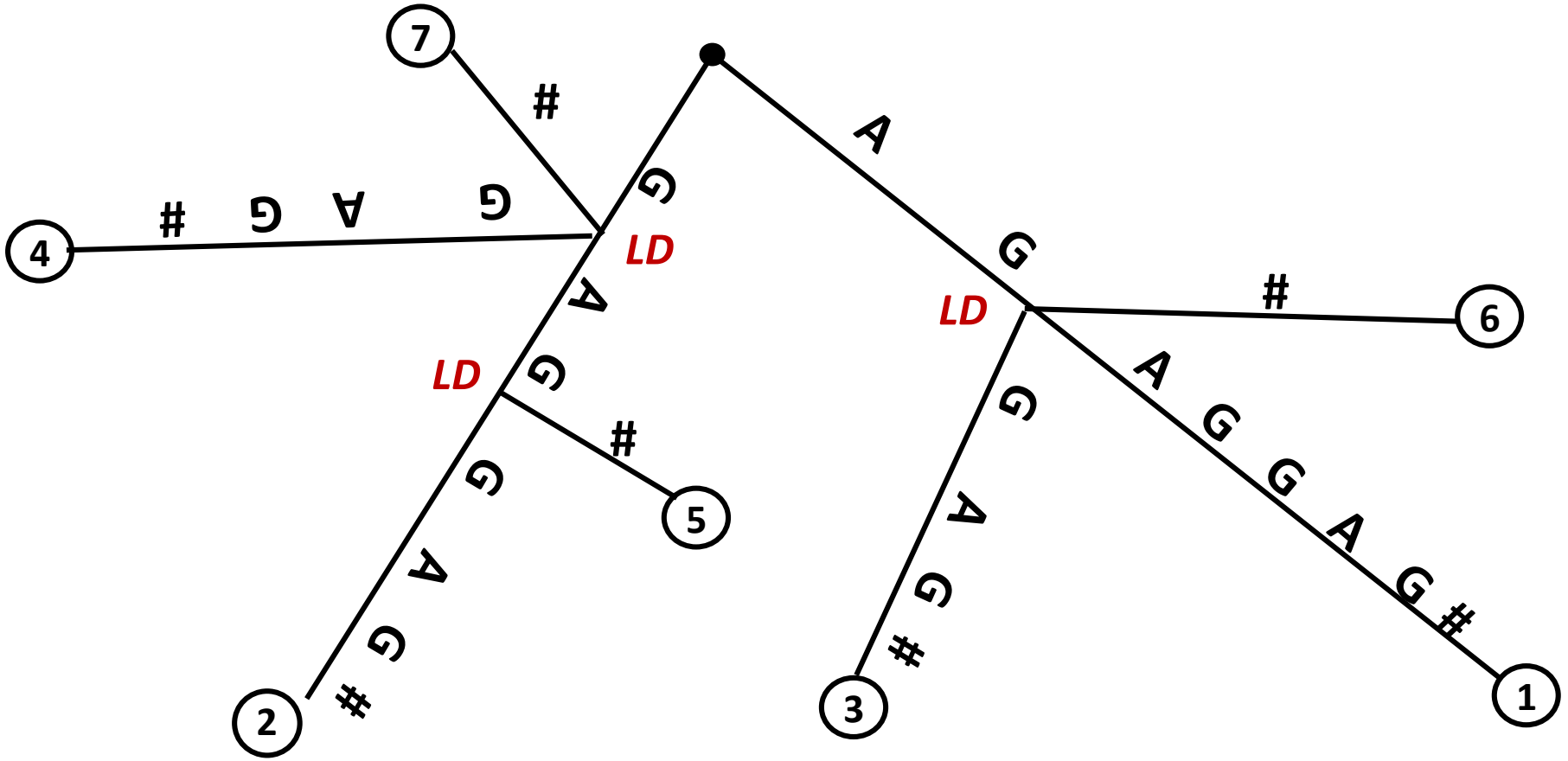
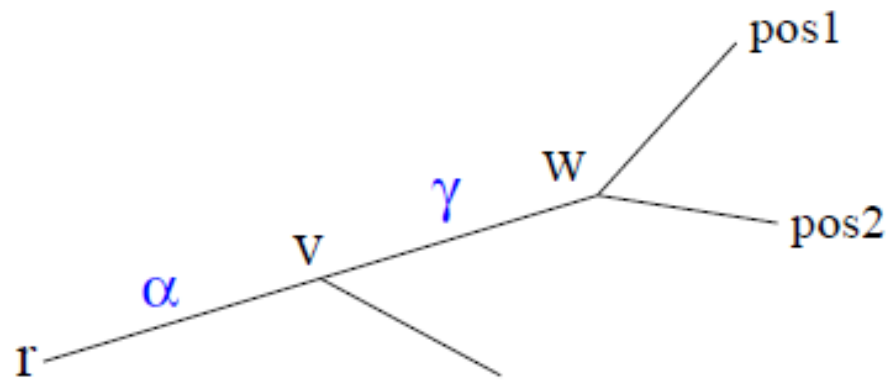


Figure suivante: v nœud left diverse, i.e. α répétition max. Soit w fils de v qui n'est pas une feuille.



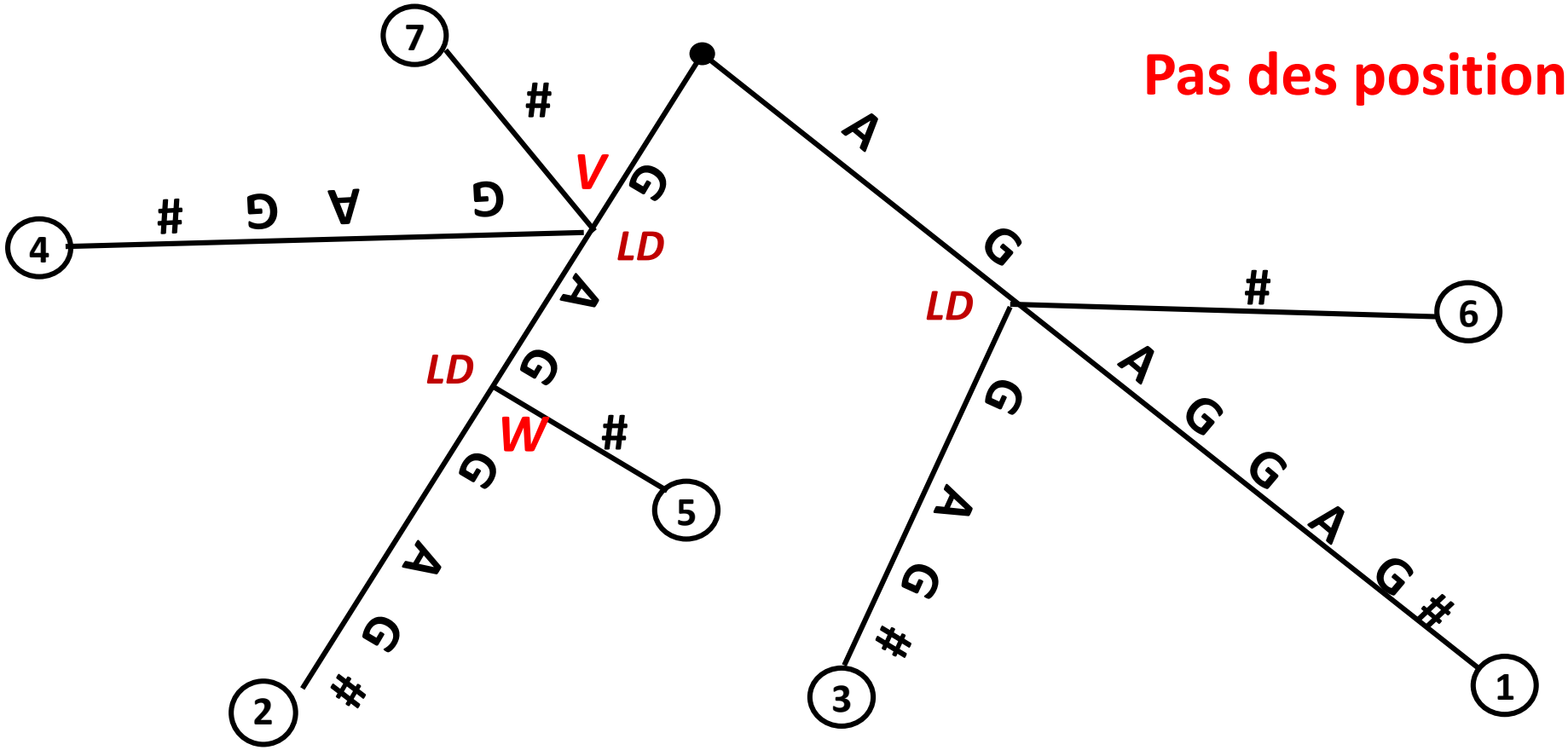
$L(w)$: Ensemble des positions de α dans S identifiées par le sous-arbre de racine w.

Lemme: Aucune position de $L(w)$ n'est une position témoin pour v.

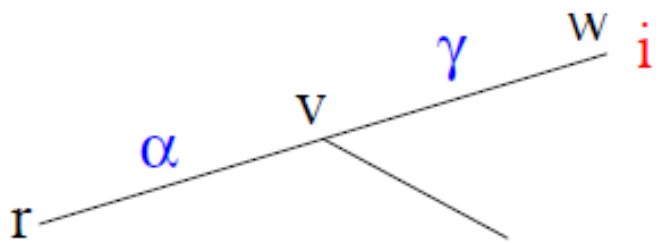
Répétitions presque-supermaximales

	1	2	3	4	5	6	7	8
S_1:	A	G	A	G	G	A	G	#

Pas des positions témoin



Si w feuille, alors étiquette de w: $\beta = \alpha\gamma$. Soit i position de w dans S, et x caractère gauche de w

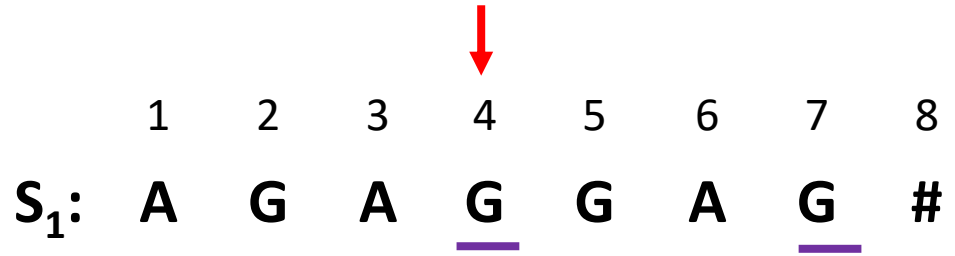


Lemme: i position témoin pour α ssi x n'est le caractère gauche d'aucune autre feuille du sous-arbre de v.

Preuve: Si $\exists$ une autre occurrence de α précédée de x, alors $x\alpha$ apparaît deux fois, et i ne peut pas être une position témoin.

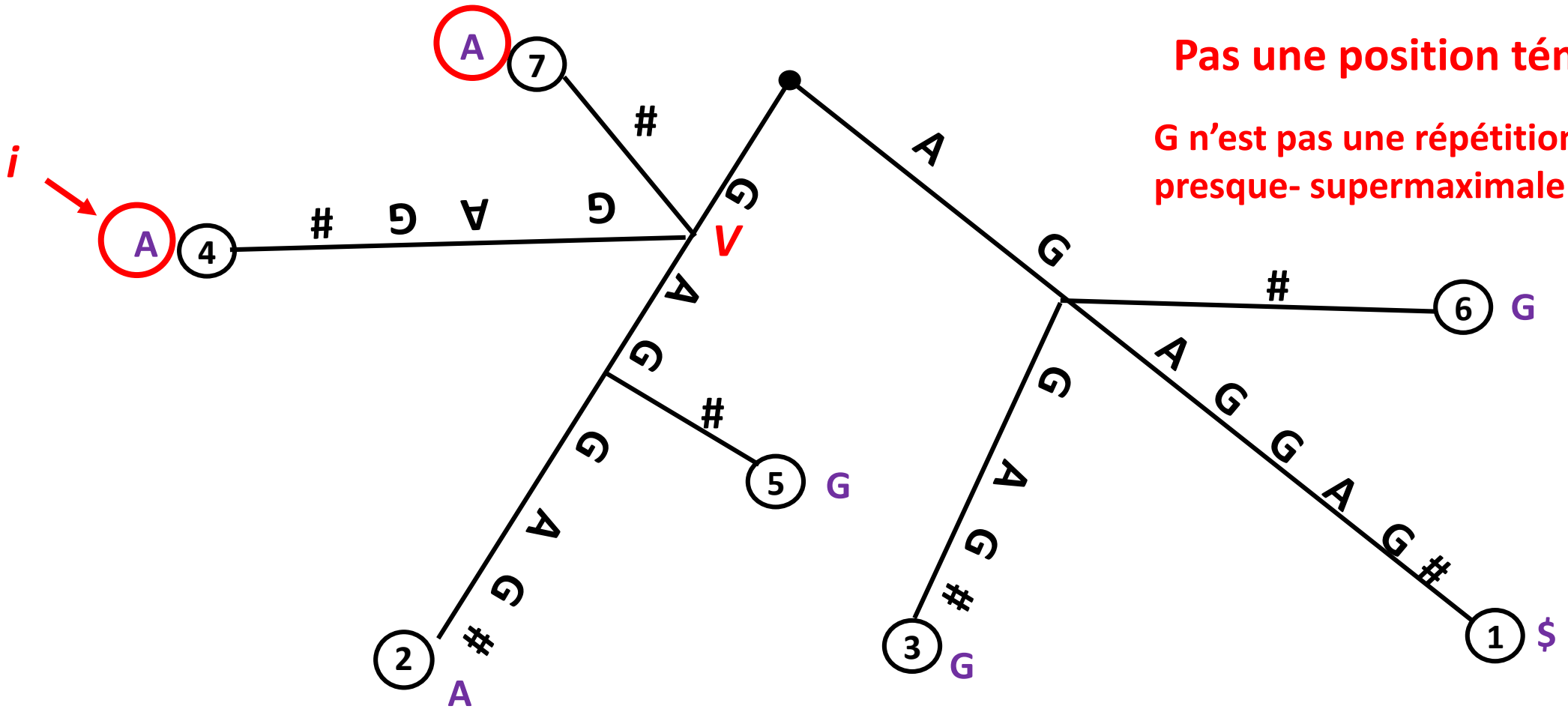
Sinon, aucune autre auccurrence de α n'est précédée de $x \Longrightarrow$ l'occurrence de α à la pos. i n'est contenue dans aucune répétition max.

Répétitions presque-supermaximales



Pas une position témoin

G n'est pas une répétition presque- supermaximale

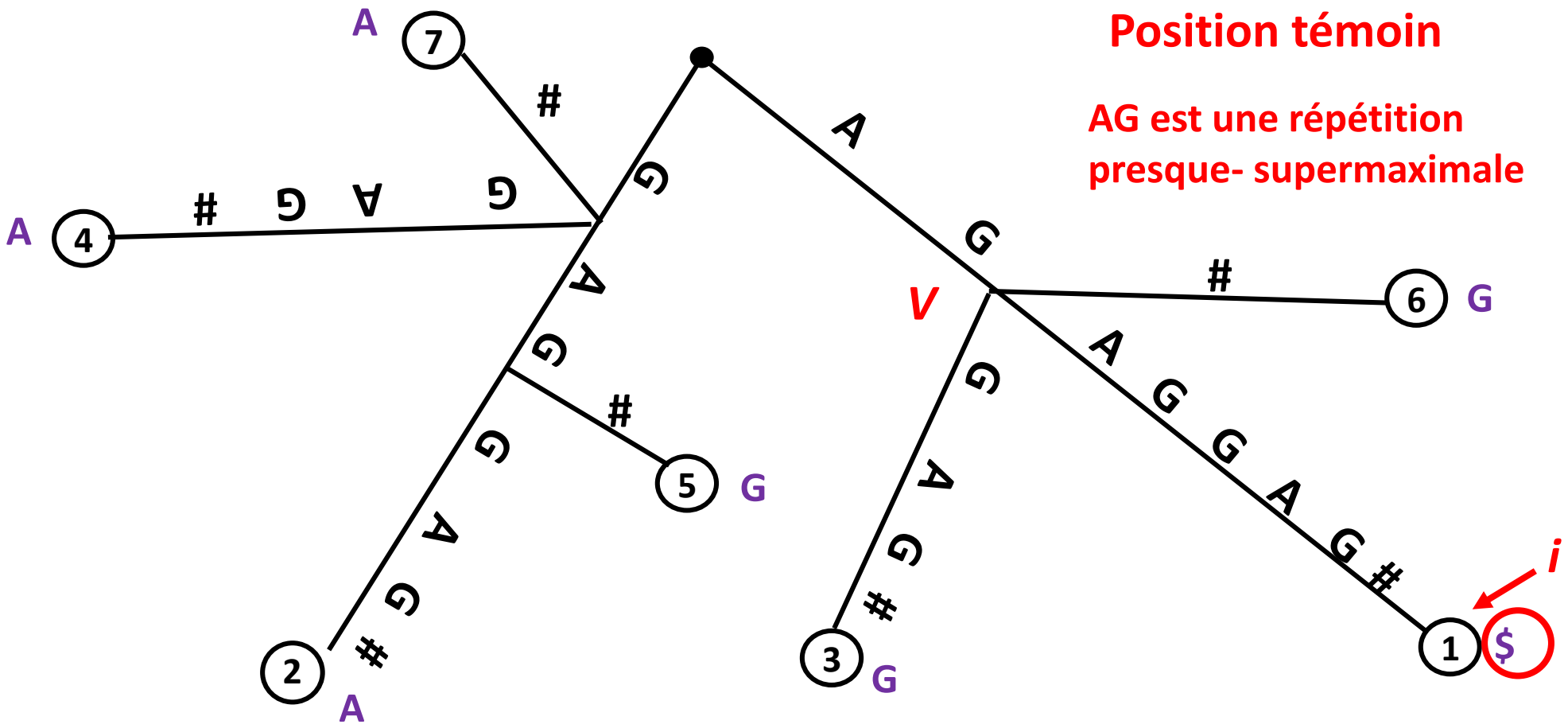


Répétitions presque-supermaximales

	1	2	3	4	5	6	7	8
S_1:	A	G	A	G	G	A	G	#

Position témoin

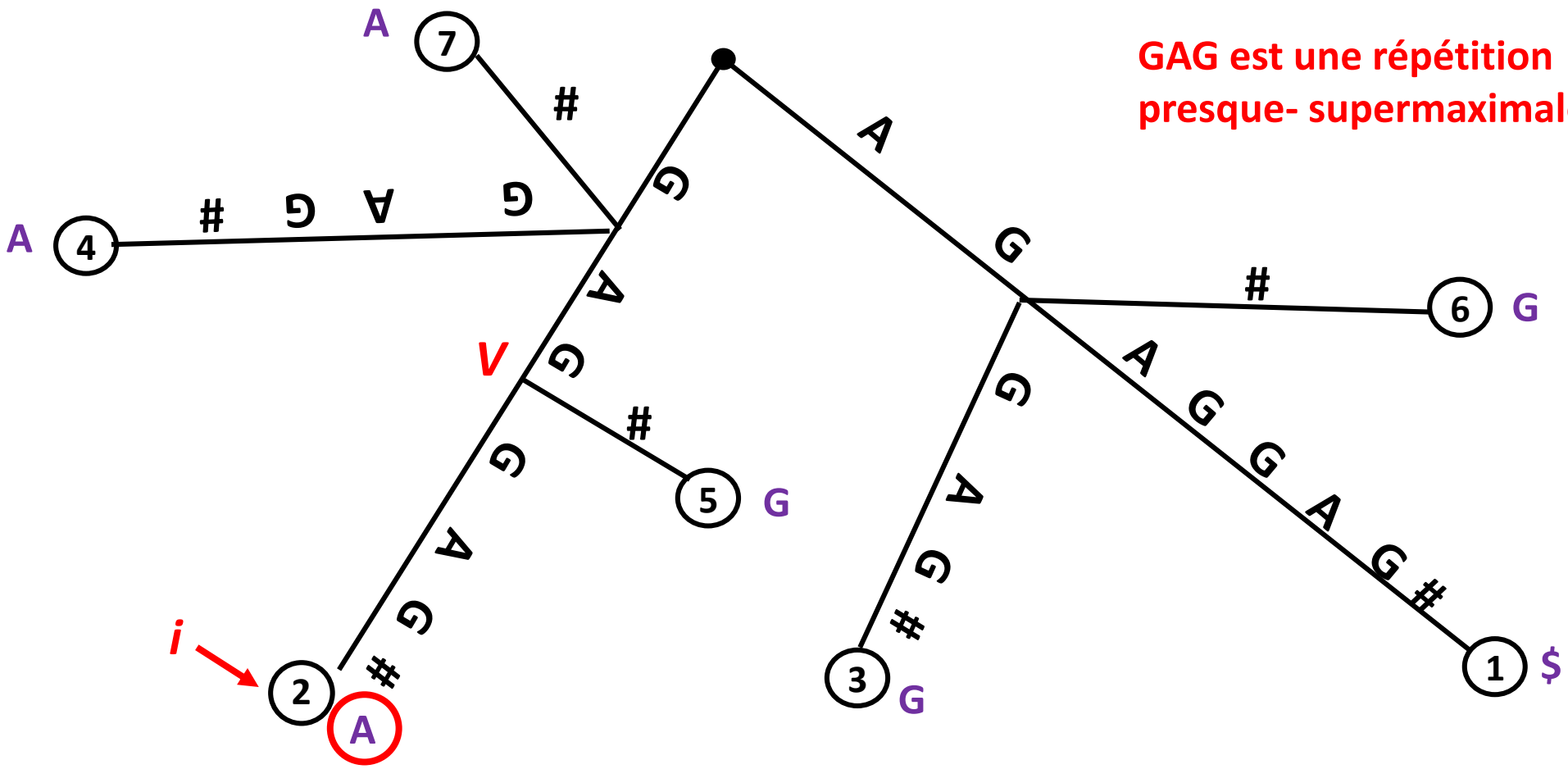
AG est une répétition presque- supermaximale



Répétitions presque-supermaximales

	1	2	3	4	5	6	7	8
S_1:	A	G	A	G	G	A	G	#

GAG est une répétition presque- supermaximale



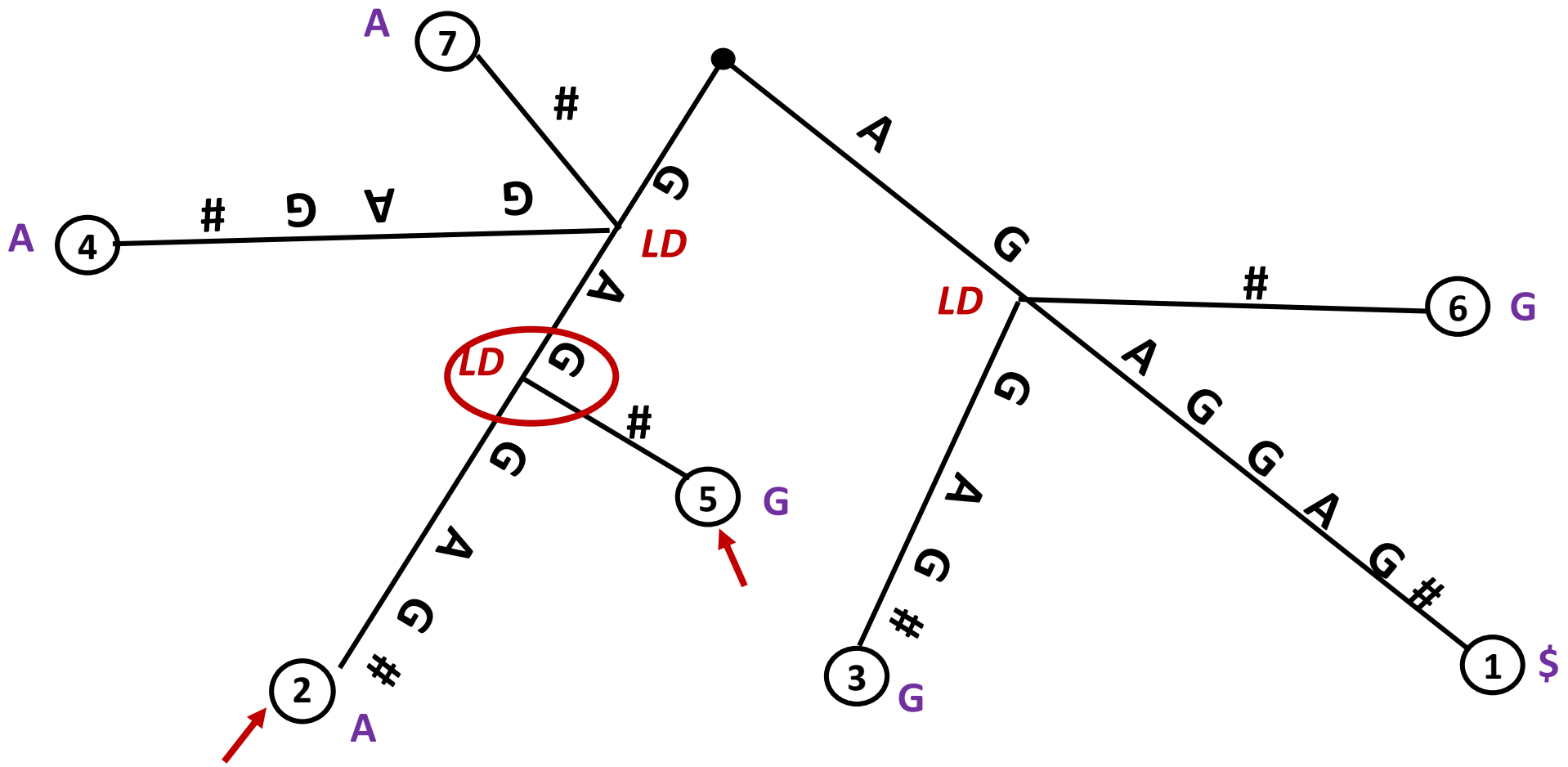
Théorème: Un nœud v left diverse représente une répétition presque supermaximale α ssi l'un des fils de v est une feuille i et $S[i-1]$ n'est le caractère gauche d'aucune autre feuille du sous-arbre de v

Un nœud left diverse représente une répétition supermaximale ssi tous les fils de v sont des feuilles, et tous les caractères gauches de ces feuilles sont différents

Toutes les répétitions presque supermaximales et supermaximales peuvent être trouvées en un temps linéaire.

Répétitions supermaximales

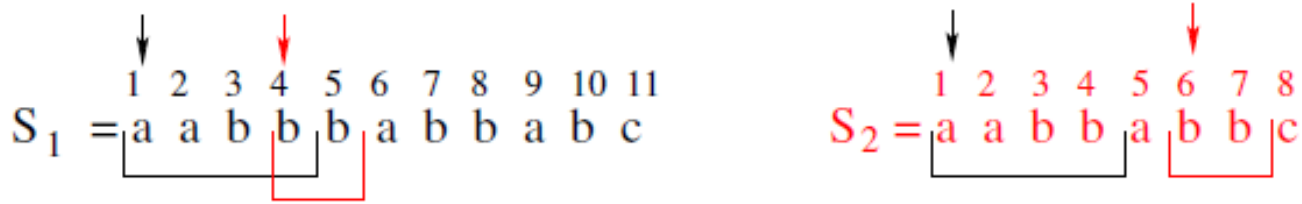
	1	2	3	4	5	6	7	8
S_1:	A	G	A	G	G	A	G	#



Plus longue extension commune

S_1, S_2: Deux séquences

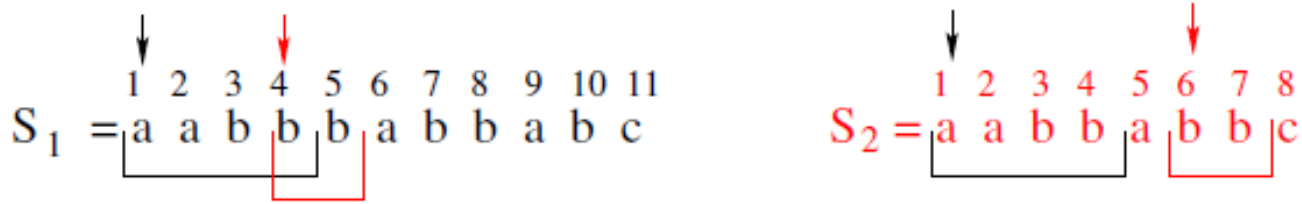
Lowest common extension (lce) pour (i, j): Plus long préfixe commun de $S[i..n_1]$ et $S_2[j..n_2]$



Plus longue extension commune

S_1, S_2: Deux séquences

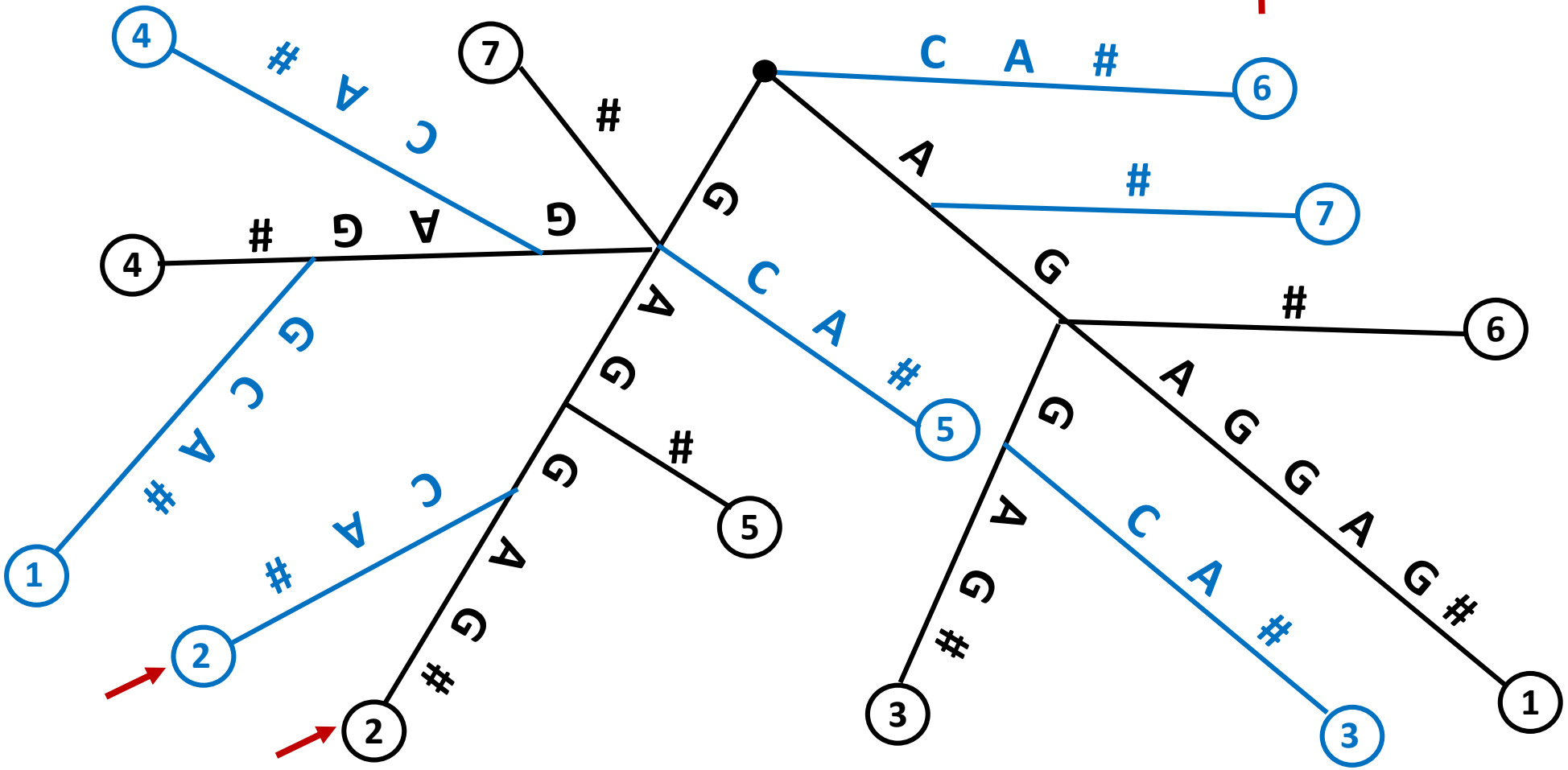
Lowest common extension (lce) pour (i, j): Plus long préfixe commun de $S[i..n_1]$ et $S_2[j..n_2]$



- Construire l'arbre des suffixes généralisé pour S_1, S_2. Conserver la profondeur en caractère de chaque nœud.

Plus longue extension commune

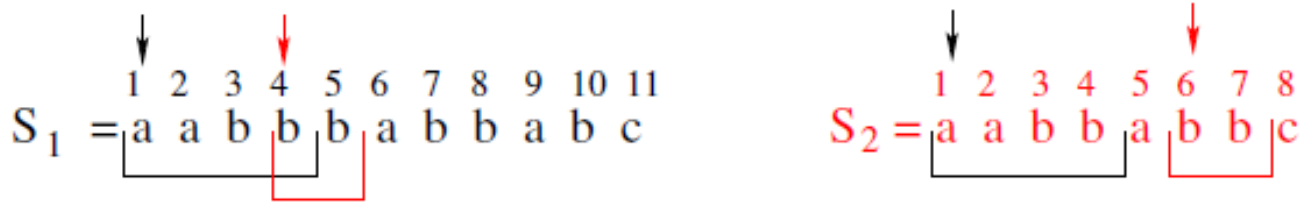
	1	2	3	4	5	6	7	8
S_1:	A	G	A	G	G	A	G	#
	1	2	3	4	5	6	7	8
S_2:	G	G	A	G	G	C	A	#



Plus longue extension commune

S_1, S_2: Deux séquences

Lowest common extension (lce) pour (i, j): Plus long préfixe commun de $S[i..n_1]$ et $S_2[j..n_2]$

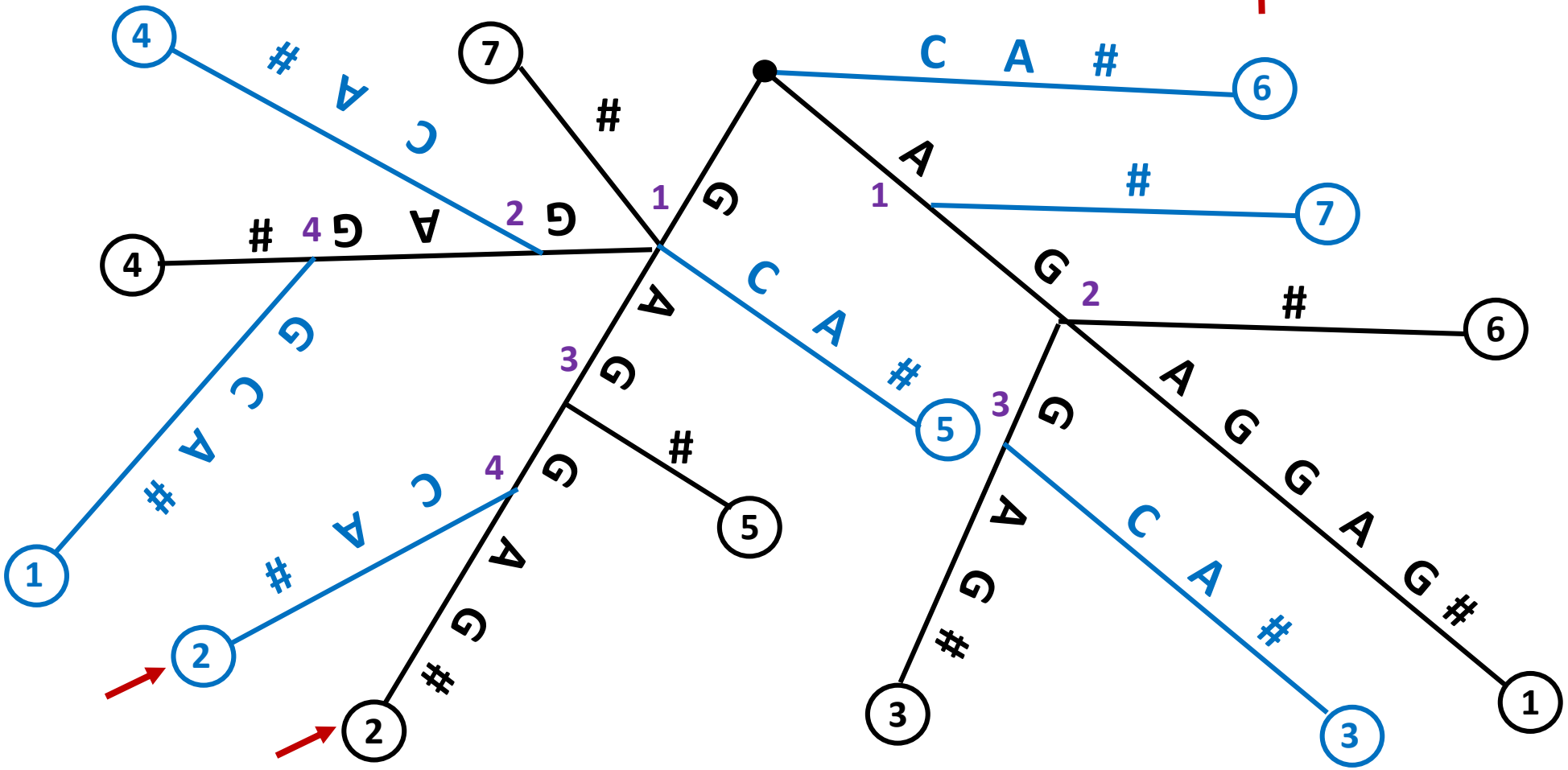


- Construire l'arbre des suffixes généralisé pour S_1, S_2. Conserver la profondeur en caractère de chaque nœud.
- Pour tout (i, j), trouver le dernier ancêtre commun (lowest common ancestor: lca) v des feuilles i, j. Profondeur en caractère de v = lce pour (i, j)

Plus longue extension commune

Profondeur en caractère du dernier ancêtre commun

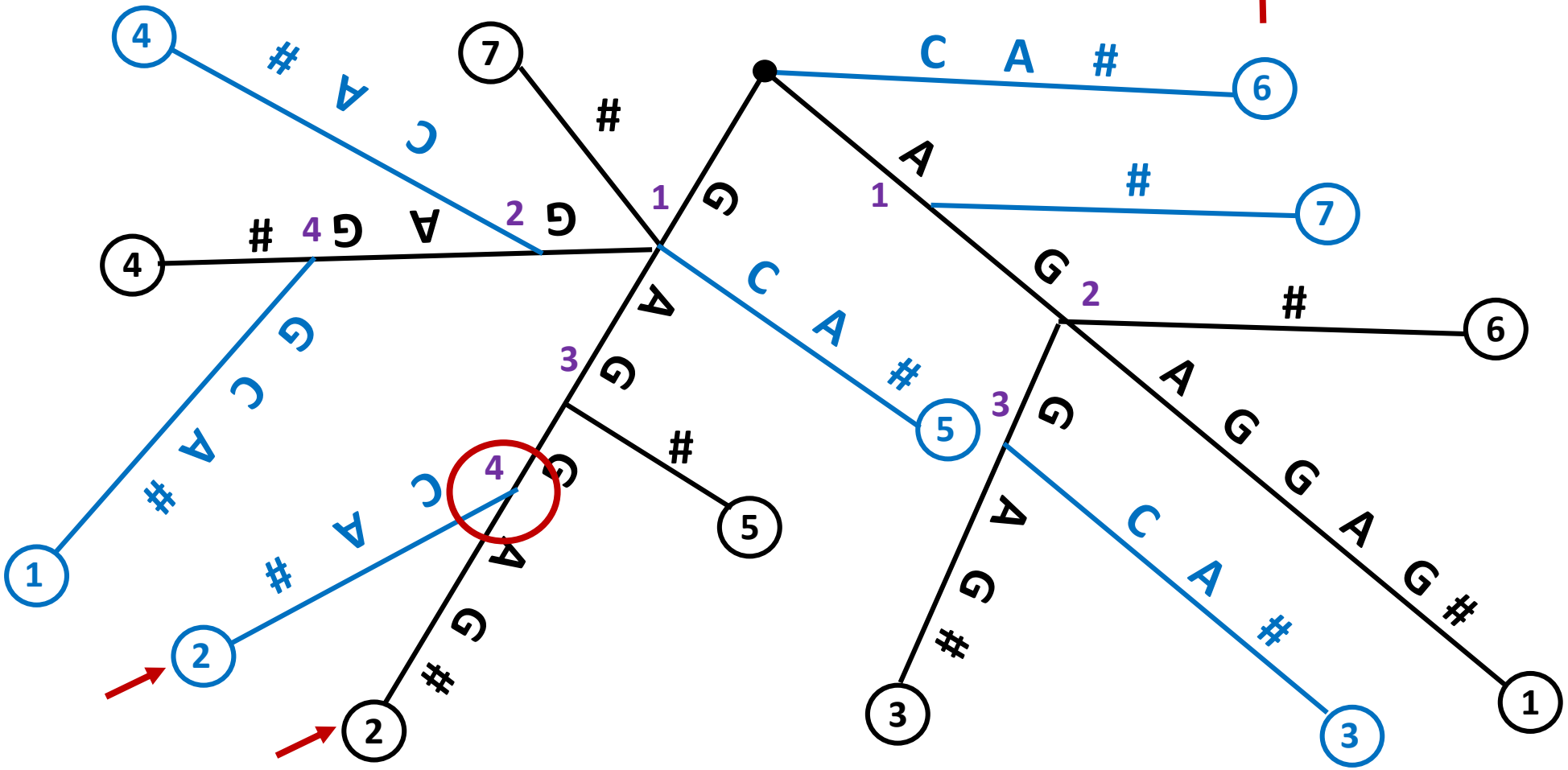
	1	2	3	4	5	6	7	8
S_1:	A	G	A	G	G	A	G	#
	1	2	3	4	5	6	7	8
S_2:	G	G	A	G	G	C	A	#



Plus longue extension commune

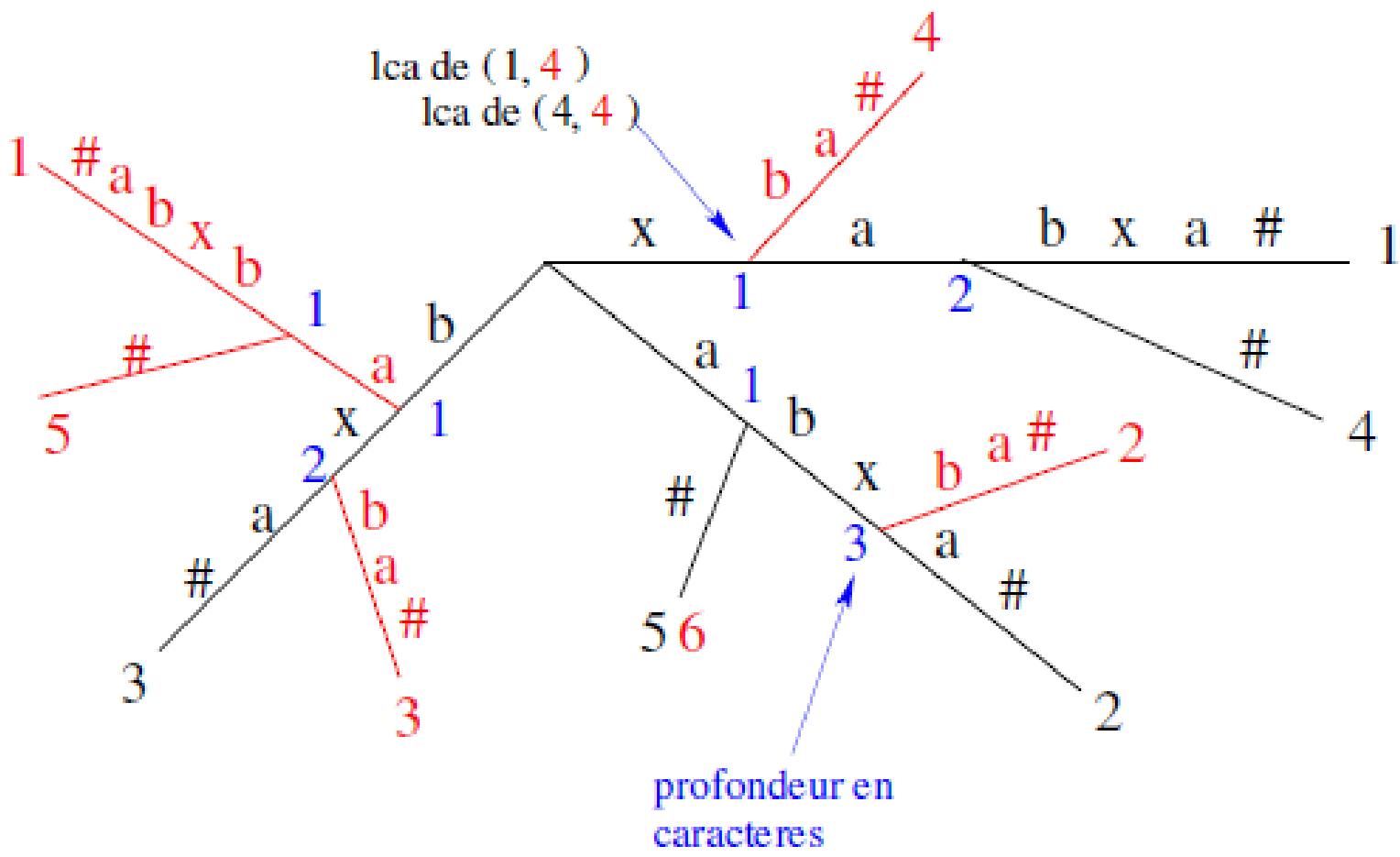
Profondeur en caractère du dernier ancêtre commun

	1	2	3	4	5	6	7	8
S_1:	A	G	A	G	G	A	G	#
	1	2	3	4	5	6	7	8
S_2:	G	G	A	G	G	C	A	#



T1 = x a b x a (1 2 3 4 5)

T2 = b a b x b a (1 2 3 4 5 6)



Dernier ancêtre commun: Après un prétraitement de l'arbre des suffixes, le lca de deux nœuds quelconques peut être trouvé en temps constant.

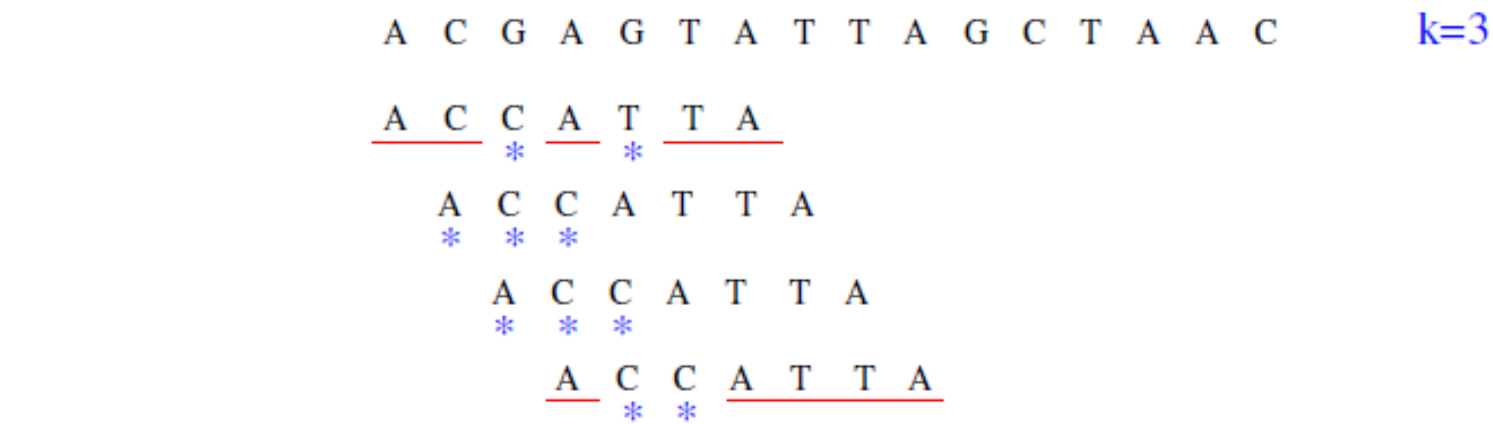
Résultat obtenu par *Harel & Tarjan, 1984* et simplifié par *Schierber & Vishkin, 1988.*

Idée: Numéroter les nœuds par des nombres représentés sur $\log(n)$ bits $\longrightarrow$ nb constant de comparaisons, additions et opérations logiques AND, OR, XOR.

Arbre des suffixes pour la recherche avec mismatches

Recherche de P de taille m dans S de taille n à k mismatches près

Idée: À chaque position j dans S, exécuter au plus k recherches de lce



Algorithme:

(1) Pour tout j faire

(2) $\quad i \leftarrow 1$; $j' \leftarrow j$; $count \leftarrow 0$;

(3) $\quad$ Tant que $count \leq k$ et $i \leq m$ faire

(3) $\quad\quad l \leftarrow$ taille de la plus longue extension des positions i dans P et j' dans S;

(4) $\quad\quad i \leftarrow i + l$;

(5) $\quad\quad$ Si $i > m$, occurrence de P à la position i;

(6) $\quad\quad$ Si non $count \leftarrow count + 1$; $j' \leftarrow j' + l + 1$;

Complexité en temps: Pour chaque position j dans S, $O(k)$ recherches de plus longues extensions $\Longrightarrow O(kn)$

Algorithme hybride

(a) G. M. Landau et U. Vishkin, *Fast parallel and serial approximate string matching*, J. Algorothms, vol. 10, pp 137 - 169, 1989

(b) Z. Galil and K. Park, *An improved algorithm for approximate string matching*, SIAM J. Comput., Vol 19, num. 6, pp. 989 - 999, 1990

(c) E. Ukkonen, D. Wood, *Approximate String Matching with Suffix Automata*, Algorithmica, vol. 10, pp 353 - 364, 1993

Recherche de P de taille m dans T de taille n à k erreurs près.

- Utilise la table des distances D par un calcul de diagonales et un prétraitement (de P ou de P ET T)
- Différence entre les algos (a), (b) et (c) est dans le prétraitement utilisé. Tous ont la même complexité.

Définitions:

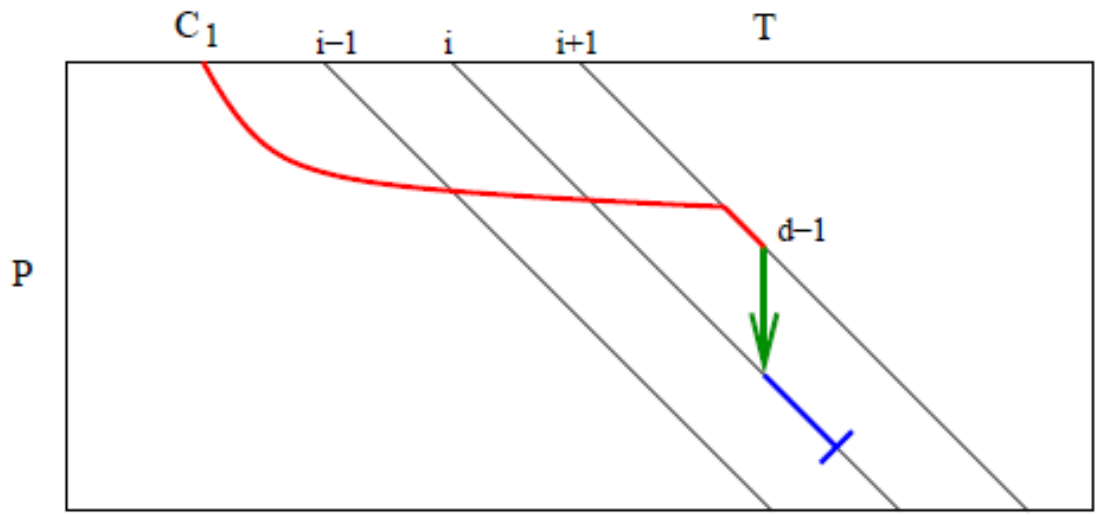
- Un chemin-d, $d \leq k$, est un chemin commençant à la ligne 0 et décrivant un alignement contenant exactement d erreurs.
- Un chemin-d est extrémal à la diagonale i, s'il est le chemin-d atteignant la case la plus éloignée sur cette diagonale.

Idée générale:

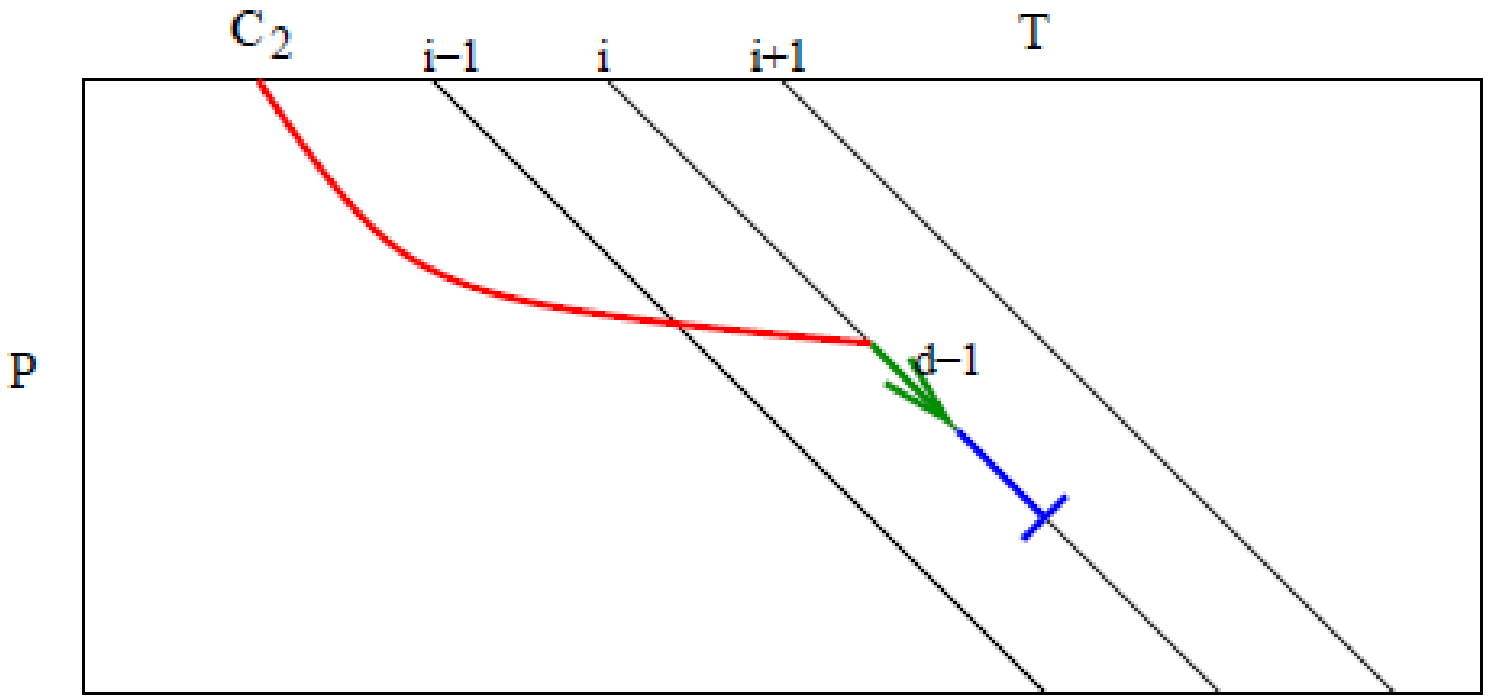
- Procéder en k itérations.
- Pour chaque itération $d \leq k$, et pour chaque diagonale i, trouver la case la plus éloignée sur la diagonale i contenant la valeur d.
- Le chemin-d extrémal à la diagonale i est déduit à partir des chemin-(d-1) extrémaux aux diagonales $(i-1)$, i et $(i+1)$.
- Un chemin-d se terminant à la ligne m spécifie une occurrence de P dans T contenant d erreurs.

Détails:

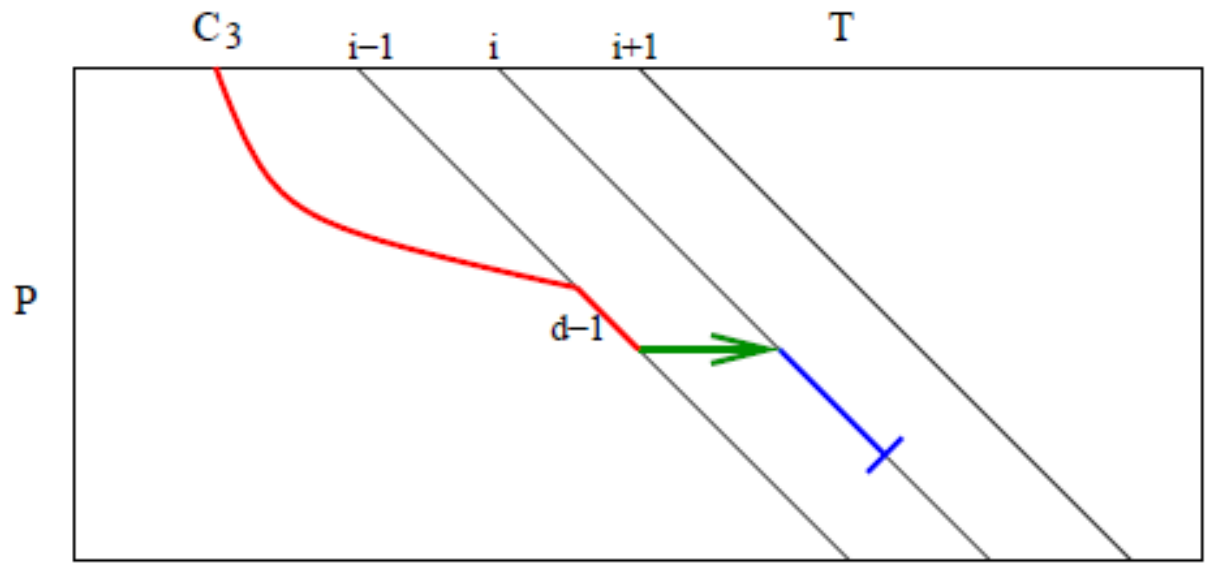
- Pour $d = 0$, le chemin-0 extrémal à une diagonale i est déterminé par le plus long facteur commun entre P et $T[i..n]$.
- Pour $d > 0$, le chemin-d extrémal à la diagonale i est obtenu à partir des trois chemins suivants:
  - C_1: Chemin-(d-1) extrémal à la diagonale $(i+1)$, suivi d'une suppression (arc vertical), suivi de l'extension maximale sur la diagonale i (plus long facteur commun entre P et T)



– C_2: Chemin-(d-1) extrémal à la diagonale i, suivi d'une substitution, suivi de la plus longue extension à la diagonale i.



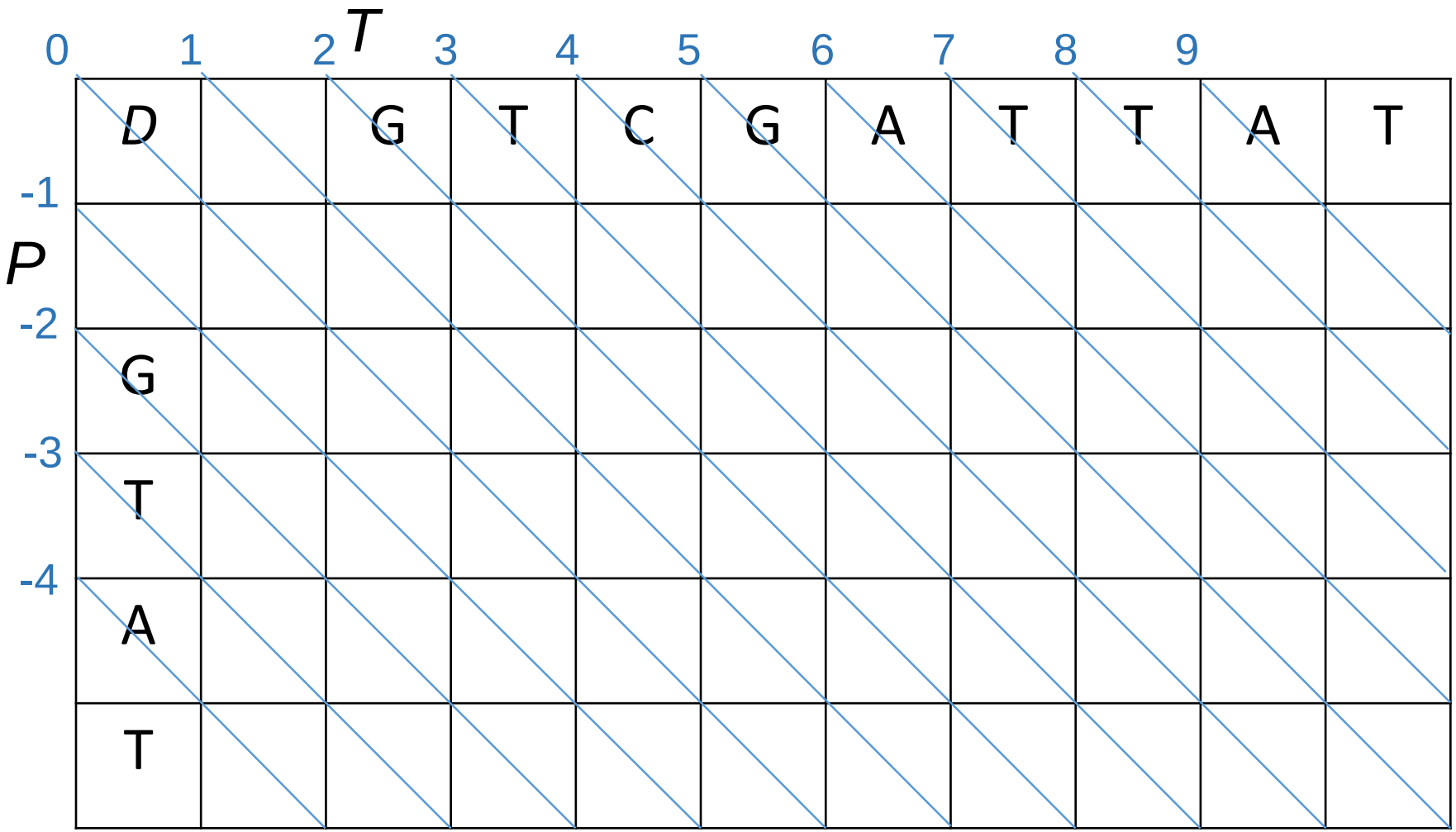
– C_3: Chemin-(d-1) extrémal à la diagonale $(i-1)$, suivi d'une insertion (arc horizontal), suivi d'une extension maximale sur la diagonale i.



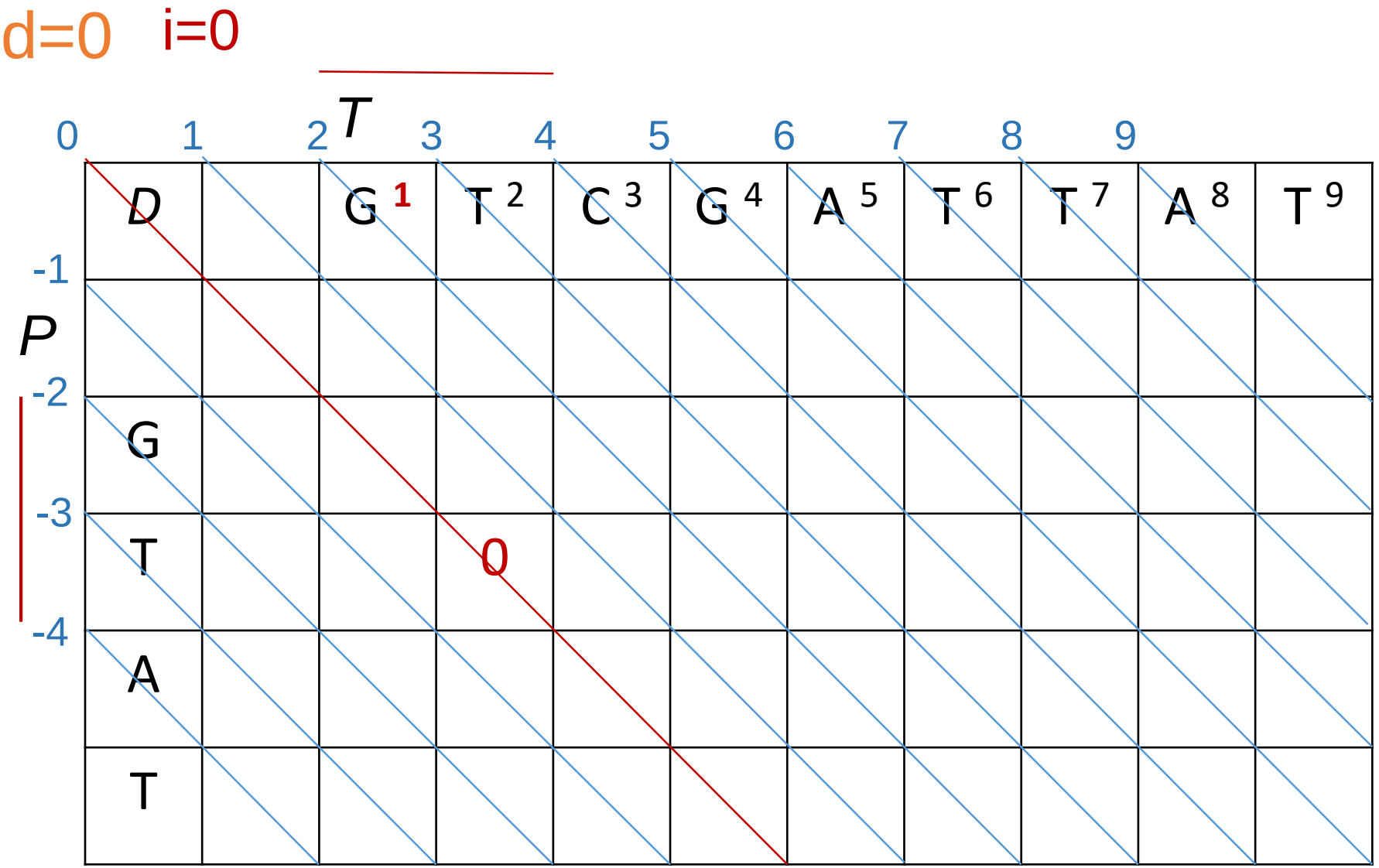
Théorème: Le chemin-d extrémal à la diagonale i est celui des trois chemins C_1, C_2 ou C_3 qui atteind la case la plus éloignée à la diagonale i.

		T								
	D		G	T	C	G	A	T	T	A	T
P											
	G										
	T										
	A										
	T										

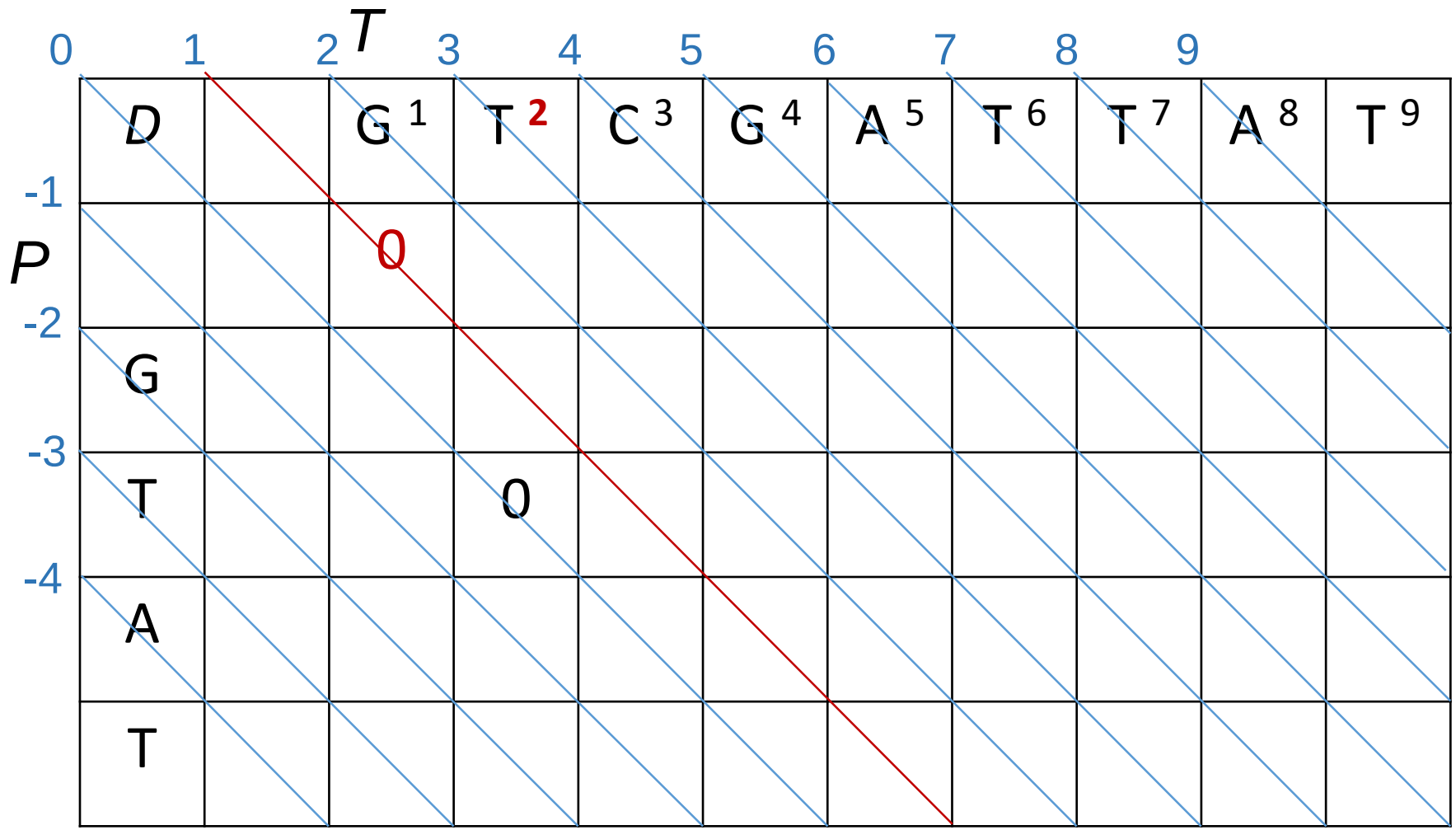
Recherche de P dans T à 1 erreur près.



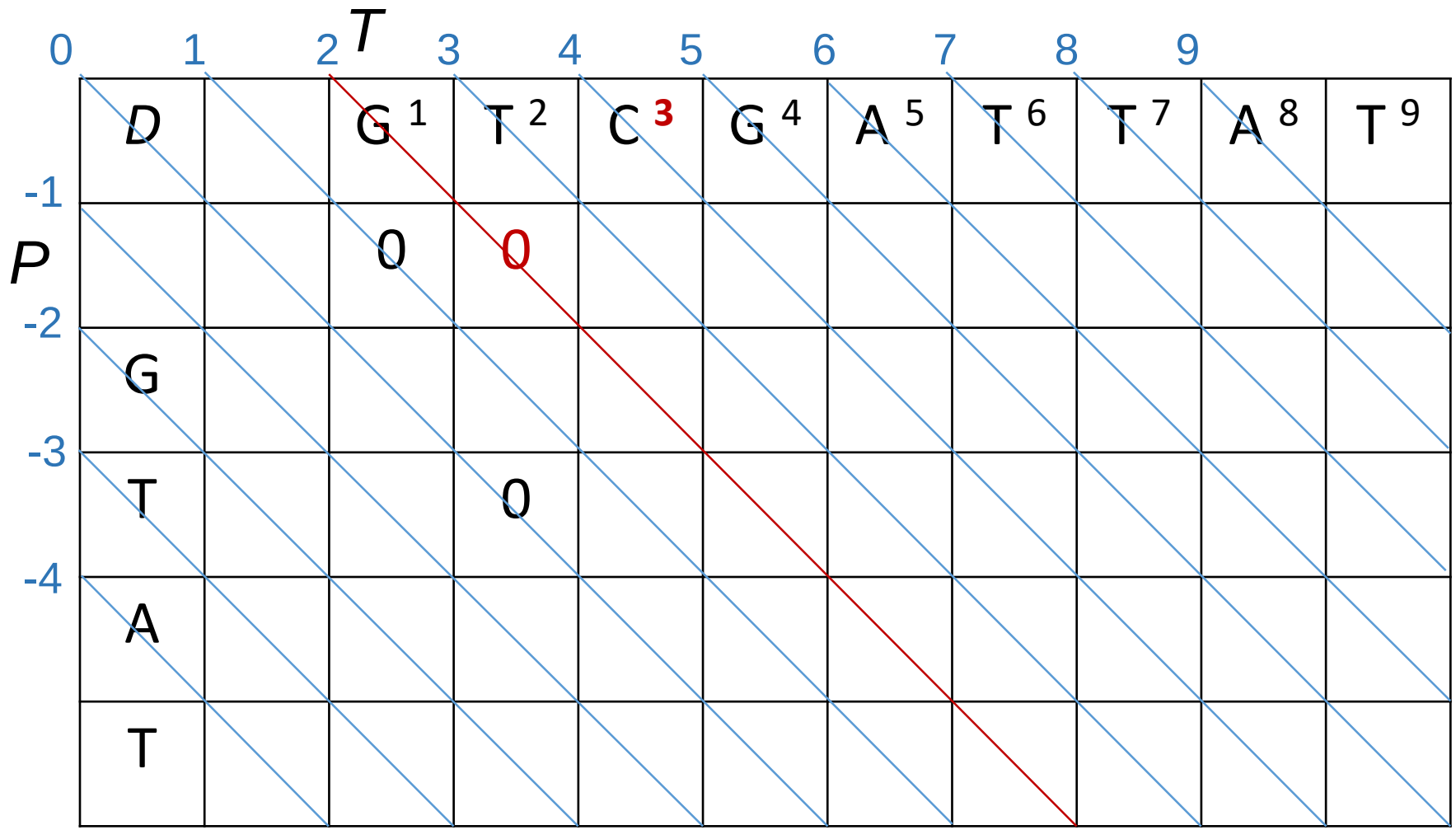
Recherche de P dans T à 1 erreur près.


d=0
i=0
T
0
1
2
3
4
5
6
7
8
9
D
G 1
T 2
C 3
G 4
A 5
T 6
T 7
A 8
T 9
-1
P
-2
G
-3
T
0
-4
A
T

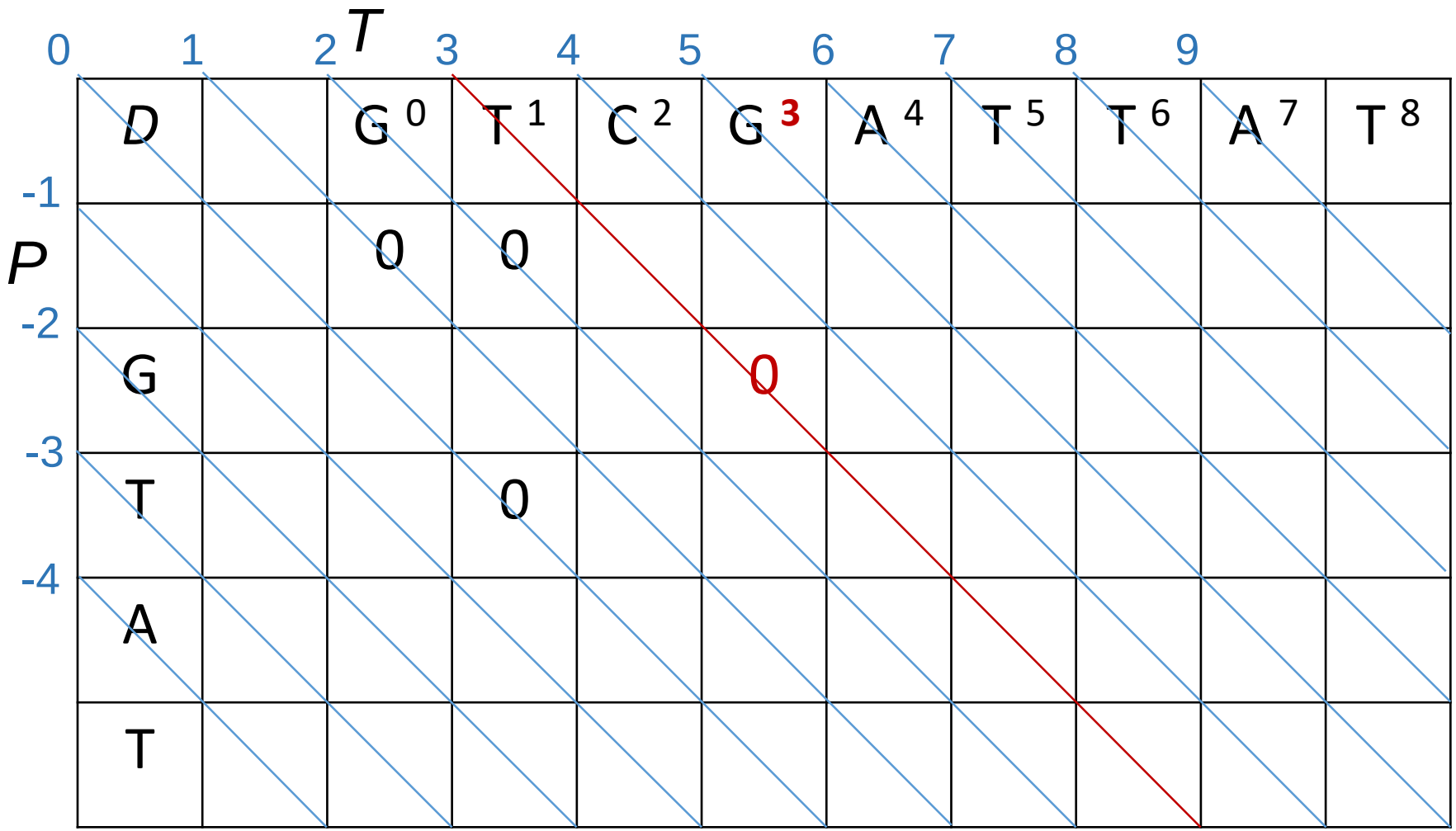
d=0 i=1



d=0 i=2

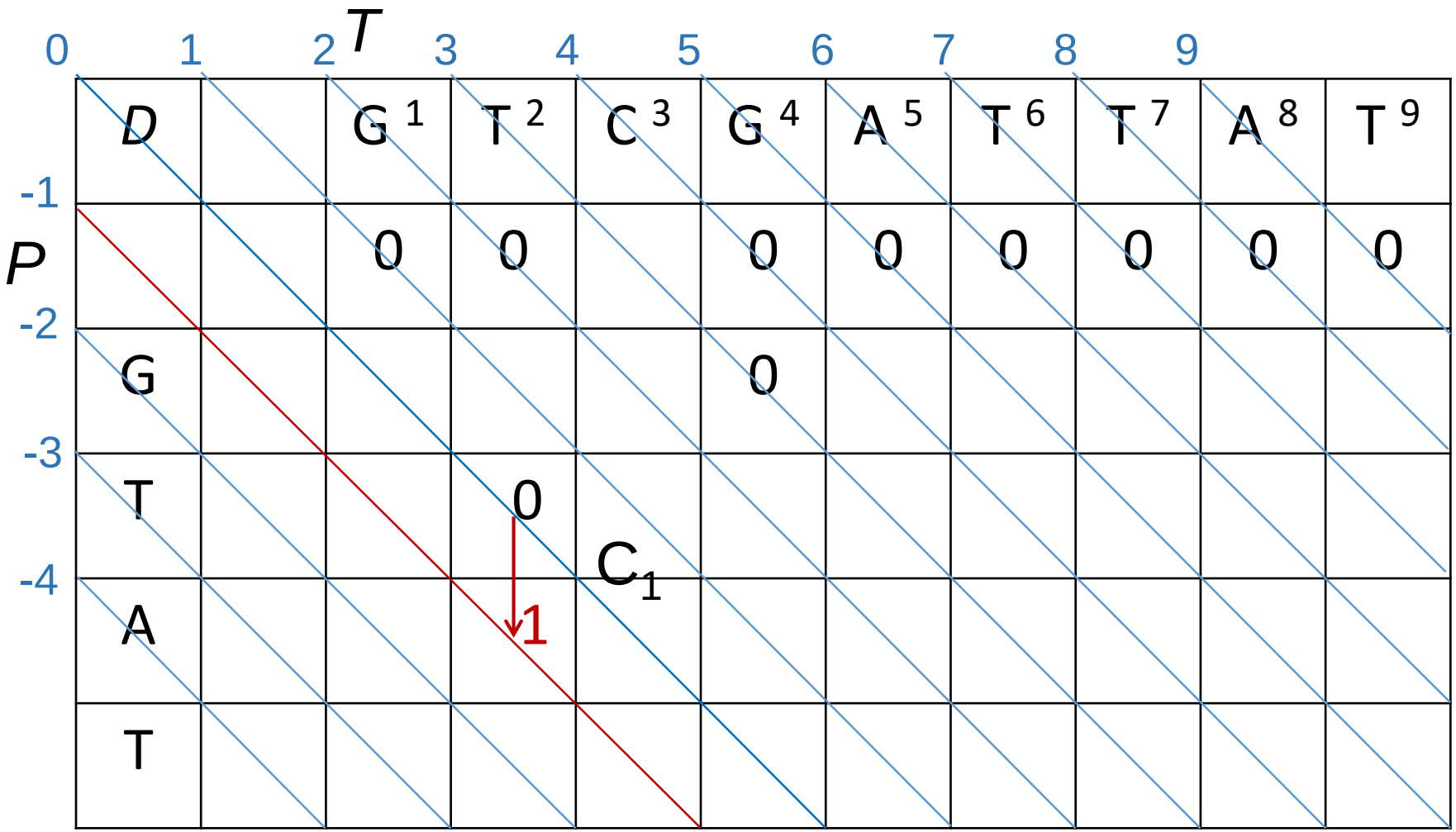


d=0 i=3

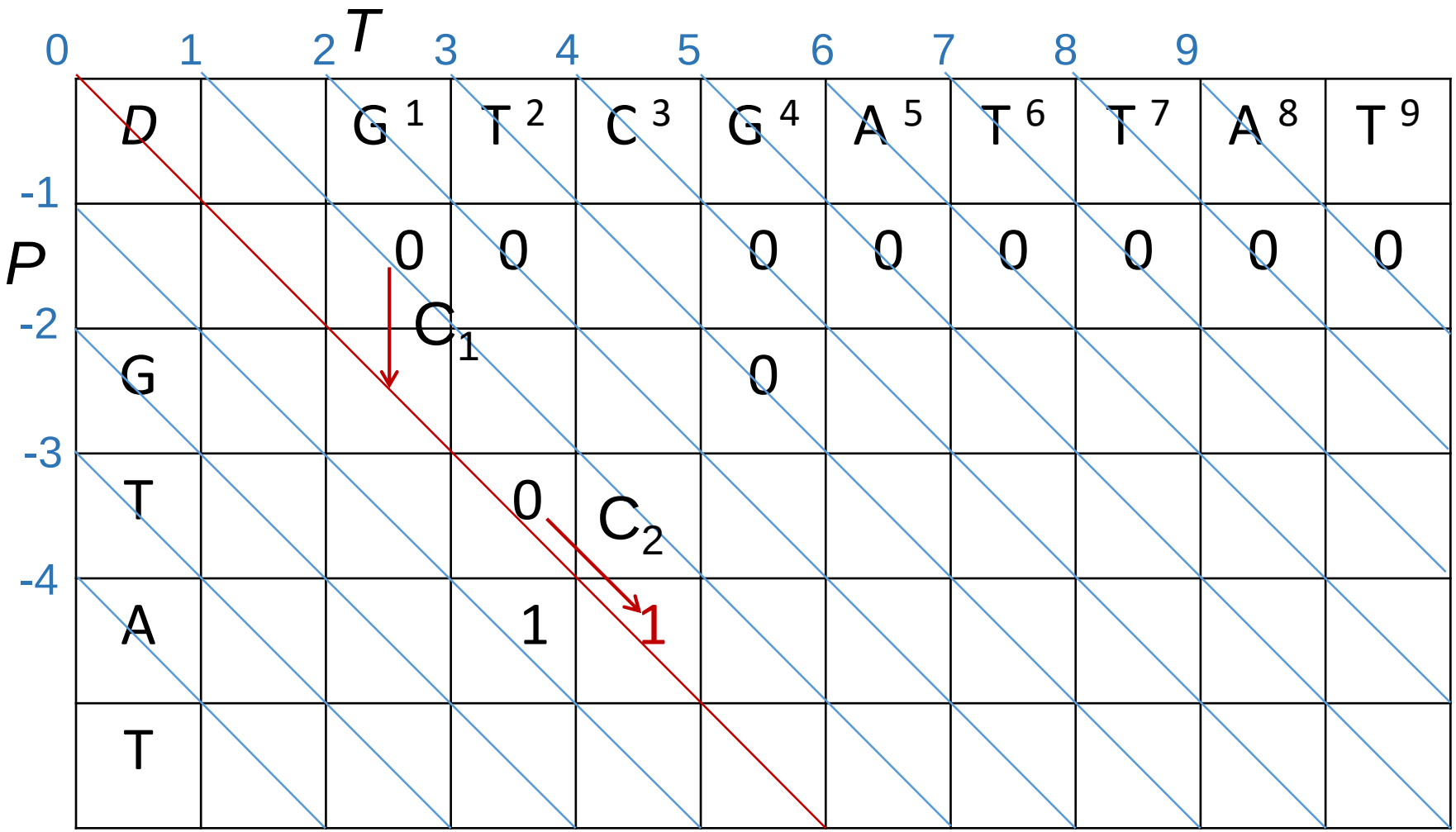


d=0

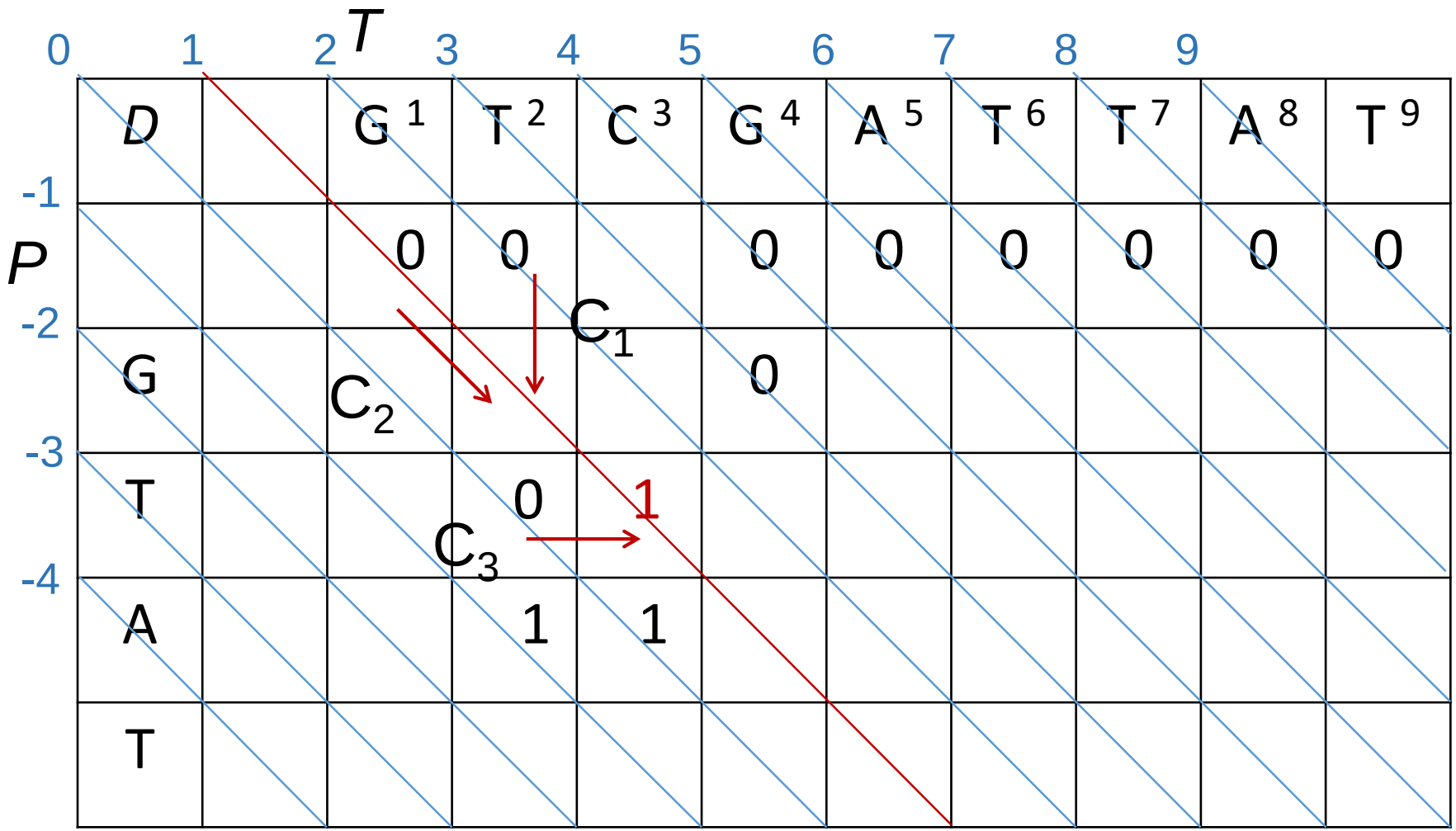
d=1 i=-1


T
0 1 2 3 4 5 6 7 8 9
P
-1 -2 -3 -4
D G 1 T 2 C 3 G 4 A 5 T 6 T 7 A 8 T 9
0 0 0 0 0 0 0 0
G 0
T 0
A
T
C1
1

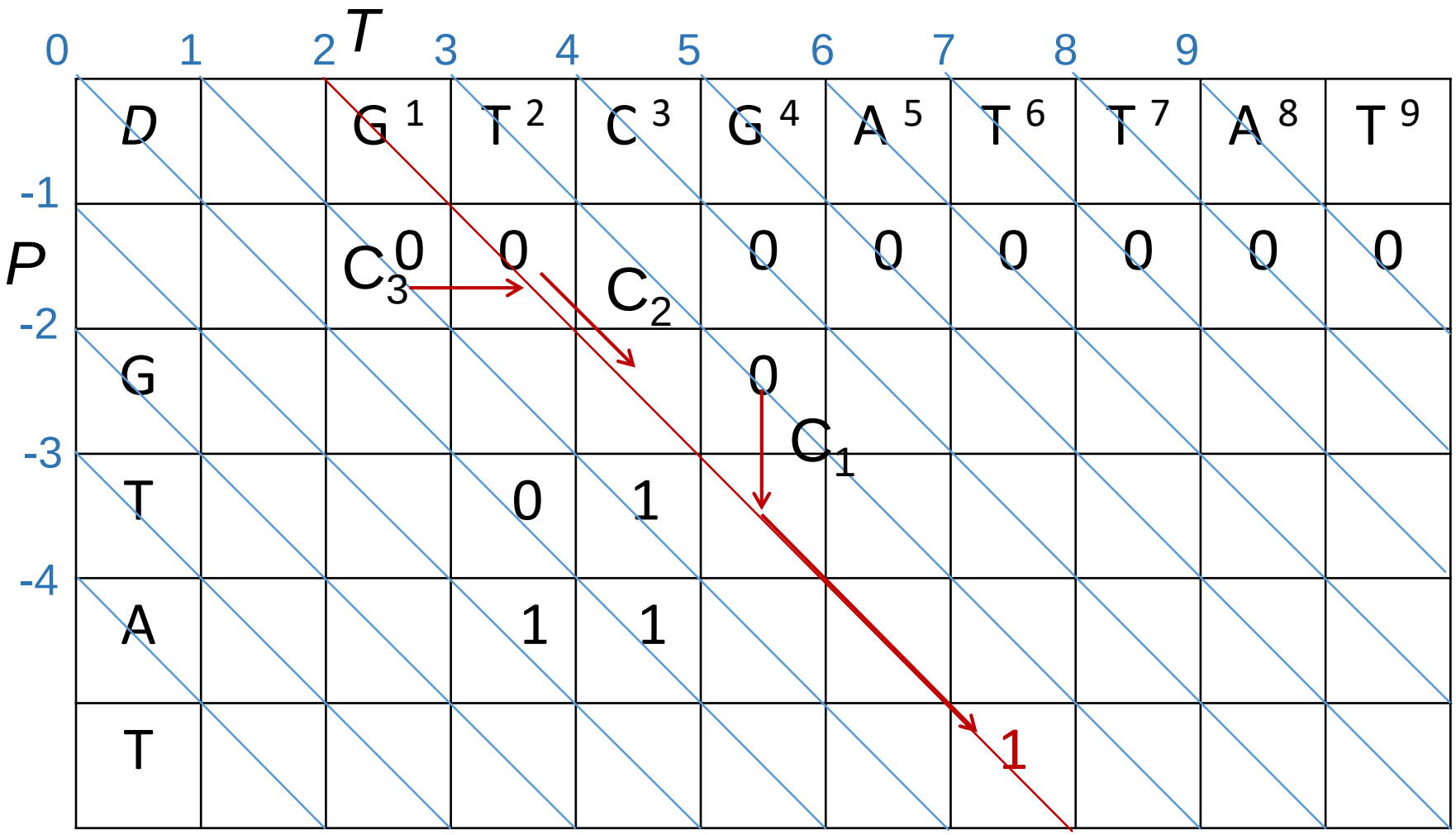
d=1 i=0


T
0 1 2 3 4 5 6 7 8 9
P
-1 -2 -3 -4
D G 1 T 2 C 3 G 4 A 5 T 6 T 7 A 8 T 9
0 0 0 0 0 0 0 0
C1
G 0
T 0
C2
A 1 1
T

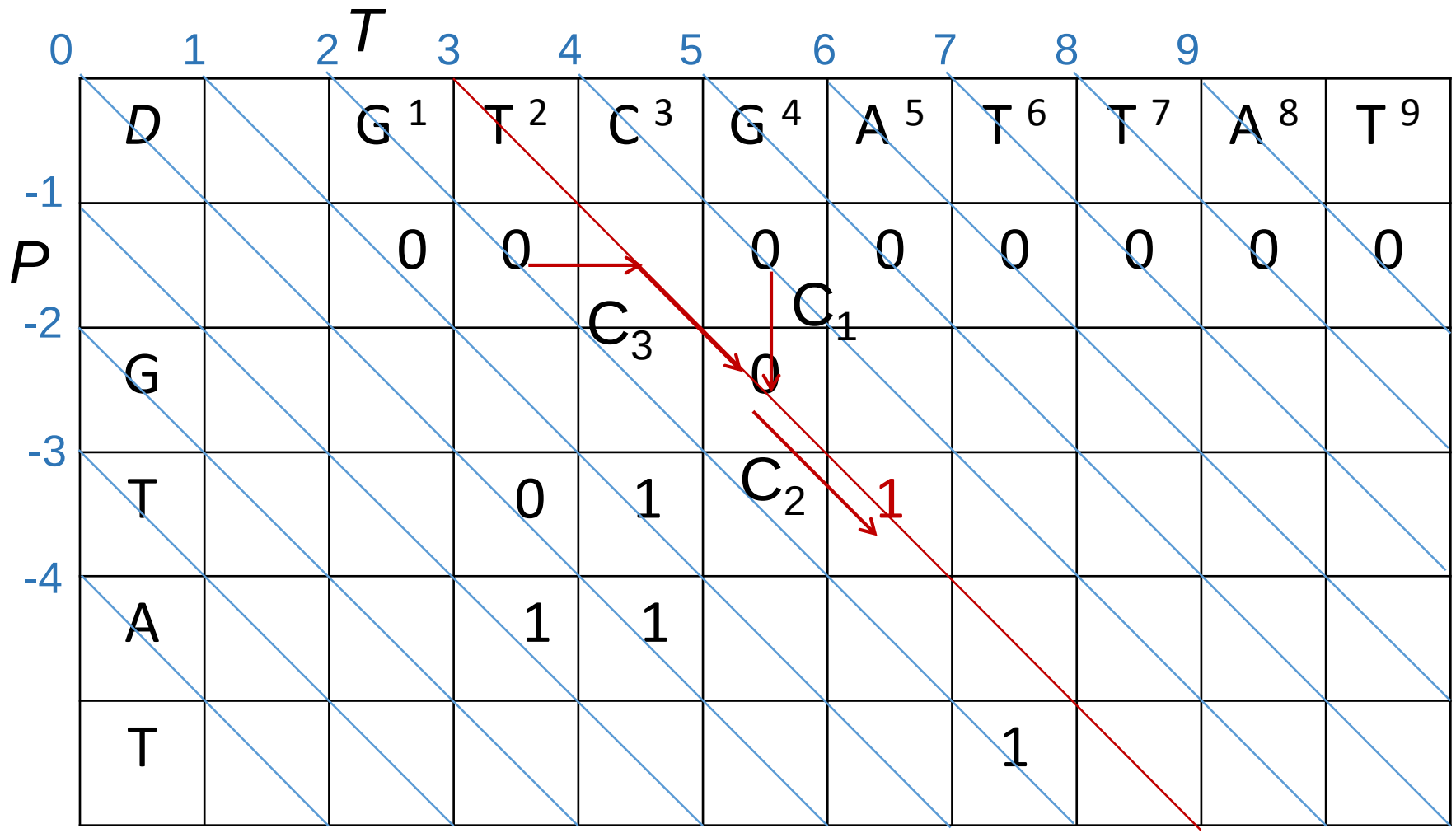
d=1 i=1


T
0 1 2 3 4 5 6 7 8 9
P
-1 -2 -3 -4
D G 1 T 2 C 3 G 4 A 5 T 6 T 7 A 8 T 9
0 0 0 0 0 0 0 0
C1
G
C2
0
T
0 1
C3
A
1 1
T

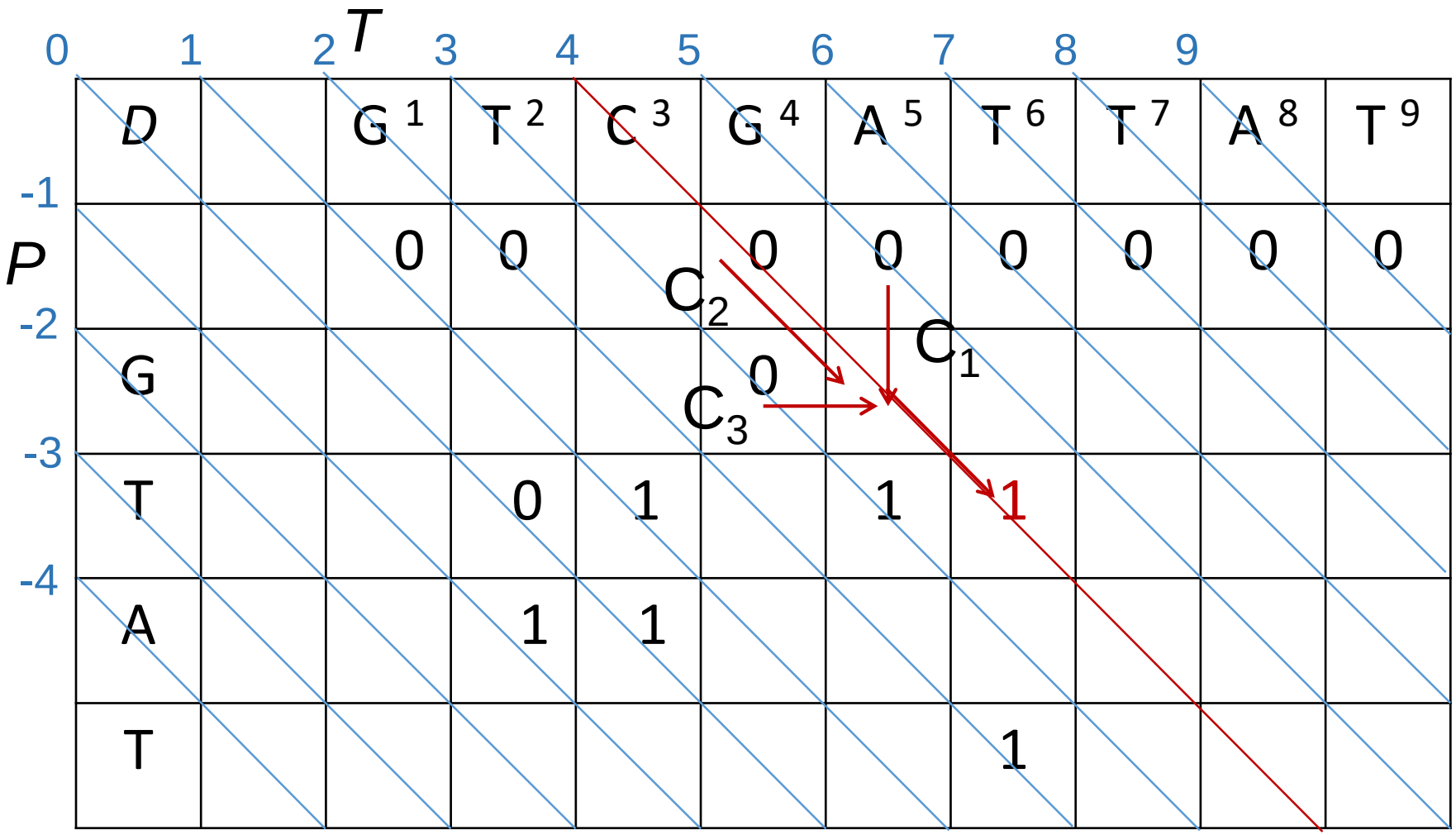
d=1 i=1


T
0 1 2 3 4 5 6 7 8 9
P
-1 -2 -3 -4
D G 1 T 2 C 3 G 4 A 5 T 6 T 7 A 8 T 9
0 0 0 0 0 0 0 0
C3
C2
G
0
C1
T
0 1
A
1 1
T
1

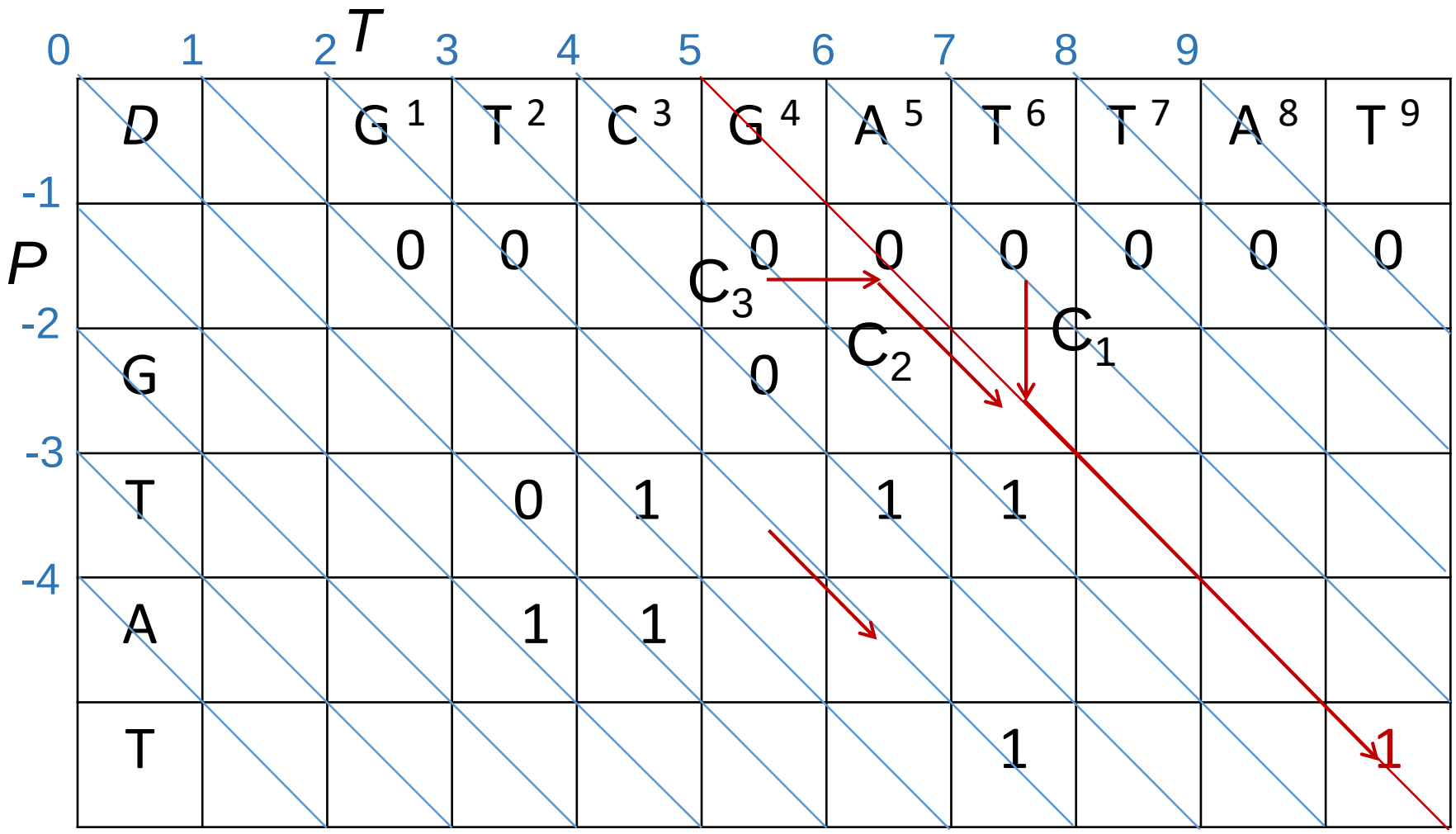
d=1 i=1


T
0 1 2 3 4 5 6 7 8 9
P
-1 -2 -3 -4
D G 1 T 2 C 3 G 4 A 5 T 6 T 7 A 8 T 9
0 0 0 0 0 0 0 0
G
T 0 1
A 1 1
T 1
C3
C1
C2
1

d=1 i=1


T
0 1 2 3 4 5 6 7 8 9
P
-1 -2 -3 -4
D G 1 T 2 C 3 G 4 A 5 T 6 T 7 A 8 T 9
0 0 0 0 0 0 0 0
G
0
T 0 1 1 1
A 1 1
T 1
C2
C1
C3

d=1 i=1


T
0 1 2 3 4 5 6 7 8 9
P
-1 -2 -3 -4
D G 1 T 2 C 3 G 4 A 5 T 6 T 7 A 8 T 9
0 0 0 0 0 0 0
G 0
T 0 1 1 1
A 1 1
T 1
1
C3
C2
C1

d=1 i=1

		T								
	D		G ^{1}	T ^{2}	C ^{3}	G ^{4}	A ^{5}	T ^{6}	T ^{7}	A ^{8}	T ^{9}
P			0	0		0	0	0	0	0	0
	G					0					
	T			0	1		1	1			
	A			1	1						
	T							1			1

Algorithme hybride:

$d = 0$;

Pour $i = 1$ à $m + 1$ faire

Trouver le plus long préfixe commun entre $P[1..m]$ et $T[i..n]$. Spécifie la dernière case du chemin-0 extrémal à la diagonale $i - 1$.

Pour $d = 1$ à k faire

Pour $i = -m$ à n faire

Trouver les cases terminales des chemins-d C_1, C_2 et C_3. Le chemin-d extrémal est l'un des trois chemins précédent qui atteint la case la plus éloignée.

Pour $j = 1$ à n

Si $D(m, j) \leq k$, alors il existe une occurrence de P dans T se terminant à la position j. Pour retrouver l'alignement, suivre les pointeurs jusqu'à la ligne 0.

Complexité:

- En espace: Pour chaque d, $0 \leq d \leq k$ et chaque diagonale i, $-m \leq i \leq n$, conserver une case (+ un pointeur).

$$\Longrightarrow O(k(m+n)) \text{ ou } O(kn)$$

- En temps: Pour chaque d et chaque i, retrouver la plus longue extension le long de la diagonale i. Correspond à retrouver un préfixe commun entre un suffixe de P et un suffixe de T, commençant à des positions connues. Peut se faire en temps constant en utilisant l'arbre des suffixes.

  Il suffit de construire l'arbre des suffixes pour P

$$\Longrightarrow O(m+kn) = O(kn)$$

Références

- *Algorithms on Strings, Trees and Sequences – Computer science and Computational biology,* Dan Gusfield, Cambridge University Press, 1997. Partie II
- *Handbook of Computational Molecular Biology,* Srinivas Aluru ed., Chapman & Hall/CRC Computer and Information Science Series, 2005. Partie II. (Voir également chapitre 35)