Université Badji-Mokhtar. Annaba
Faculté des sciences de l'ingéniorat
Département d'Informatique

Chapitre 3

Méthodes de Compression Dictionnaires

S3- 2024/2025

Introduction

- **Compression dictionnaire :** repose sur le remplacement de séquences de données répétées par des références à un dictionnaire, un ensemble de séquences déjà rencontrées.

✓ Les données compotant beaucoup de répétitions: textes, fichiers exécutables, etc.

✓ **Exemple: Lempel ziv welch (LZW)**.

Lempel Ziv (LZ77, LZ78)

- ✓ LZ77 et LZ78 sont les deux algorithmes de compression de données sans perte.
- ✓ Développé par Abraham Lempel, Jakob Ziv en 1977 et 1978.
- ✓ Ils sont également connus sous les noms de LZ1 et LZ2.
- ✓ Ces algorithmes visent à compresser les données en identifiant et éliminant la redondance.
- ✓ LZ77 compresse les données en identifiant les séquences répétées dans une « **fenêtre glissante** » de données récemment traitées, puis encode ces séquences comme références.
- ✓ LZ78 construit un dictionnaire dynamique pendant l'encodage pour attribuer un code à chaque séquence.

LZ77 (Compression)

- **L’algorithme est le suivant :**

1. LZ77 examine un segment des données dans une fenêtre de taille fixe, qui comprend une partie des données qui ont déjà été codées (**search buffer**) et une partie qui doit encore être codée (**lookahead buffer**).
2. l'algorithme vérifie si une séquence de caractères dans **lookahead buffer** correspond à une séquence précédente dans **search buffer.**
3. Si une correspondance est trouvée, LZ77 l'encode avec un triple **« position, longueur, caractère suivant ».**
4. Si un caractère ne correspond à aucune séquence précédente, il est ajouté sous forme de « **littéral »**
5. Après avoir encodé une séquence, l'algorithme décale la fenêtre vers l'avant de la longueur de la séquence encodée et répète le processus jusqu'à ce qu'il atteigne la fin des données d'entrée.

LZ77 (Compression)

- **Exemple:**

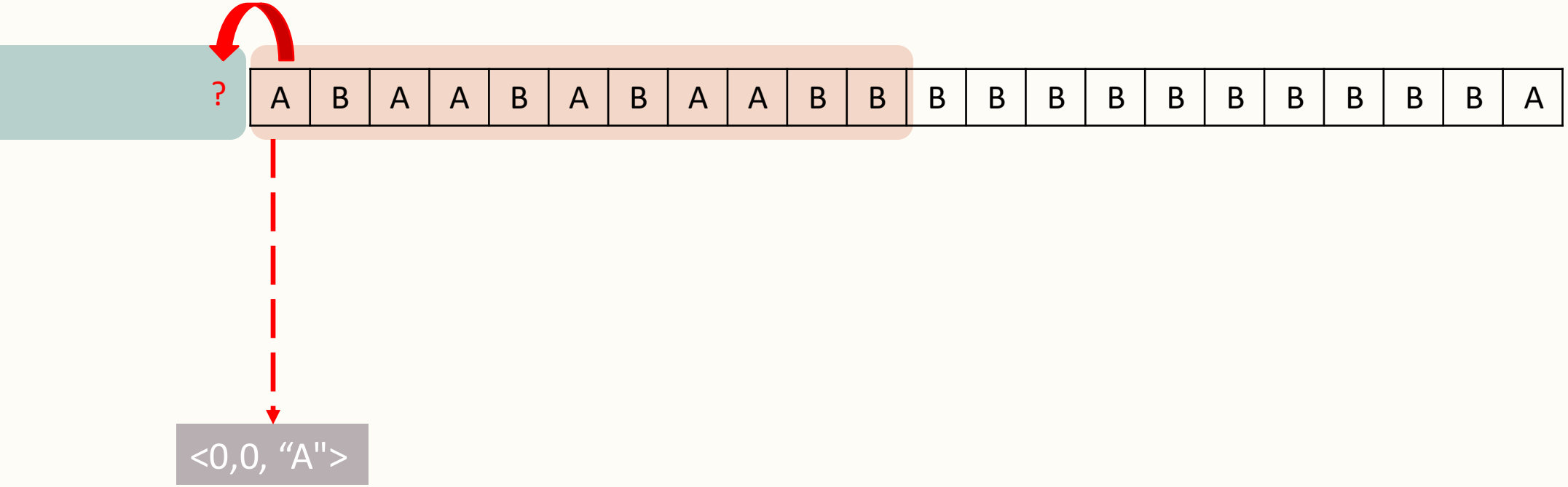
Sliding window

A	B	A	A	B	A	B	A	A	B	B	B	B	B	B	B	B	B	B

« position, longueur, caractère suivant »

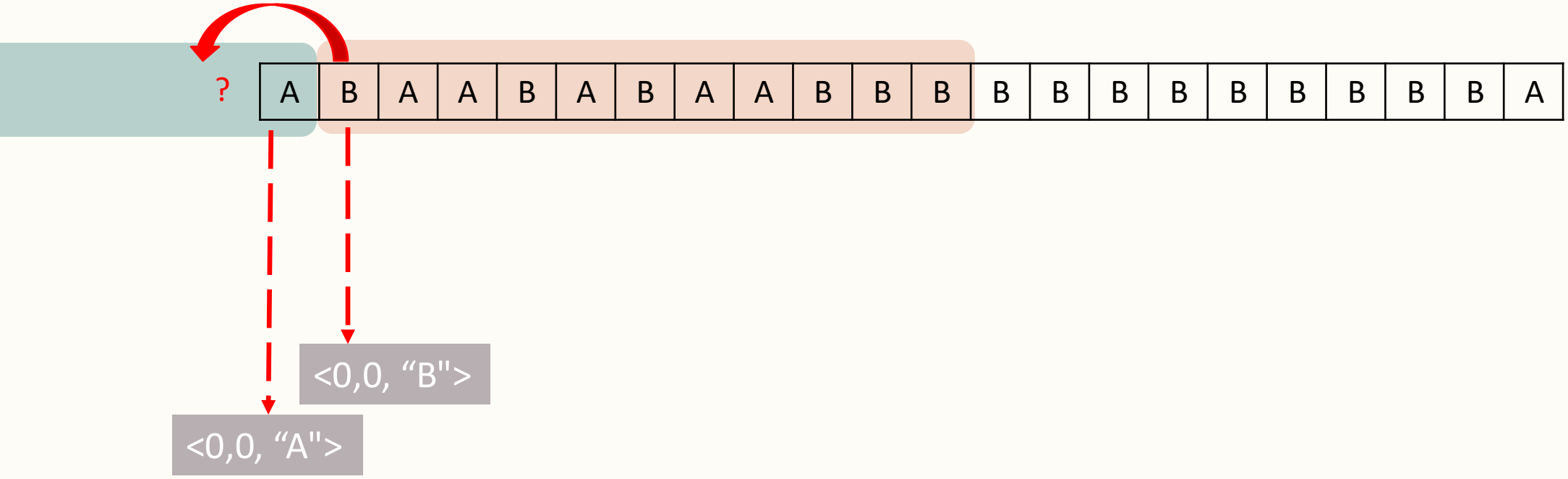
LZ77 (Compression)

- **Exemple:**



LZ77 (Compression)

- **Exemple:**

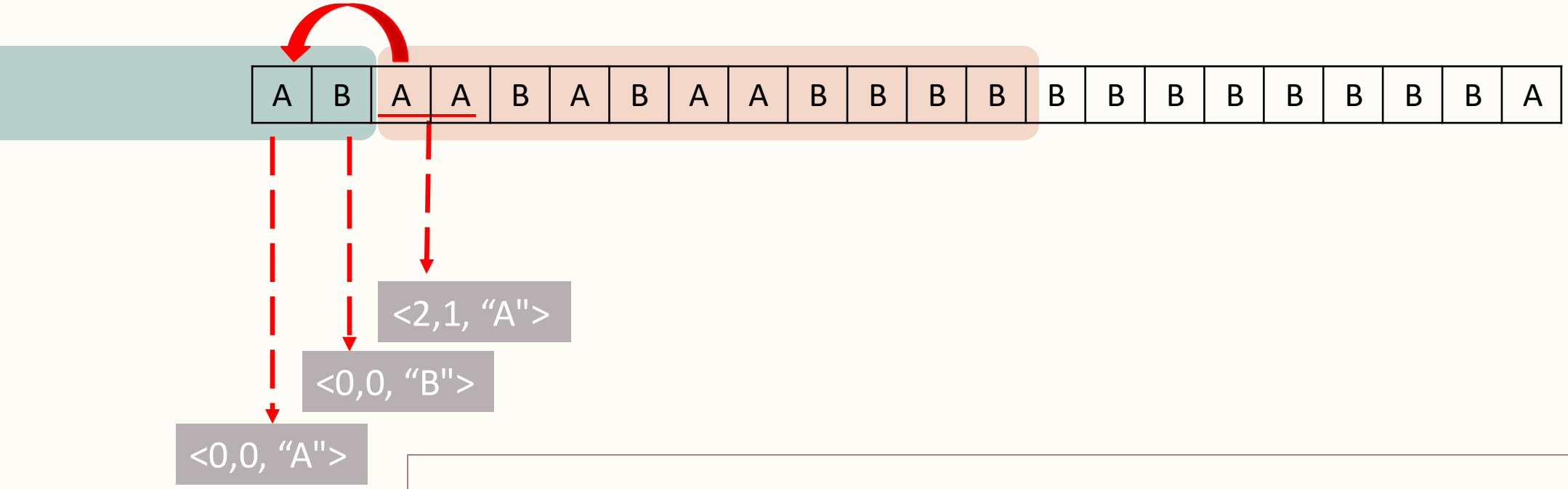


Il n'y a pas de B dans search buffer

« position=0, longueur=0, caractère suivant = «"B"»

LZ77 (Compression)

- **Exemple:**



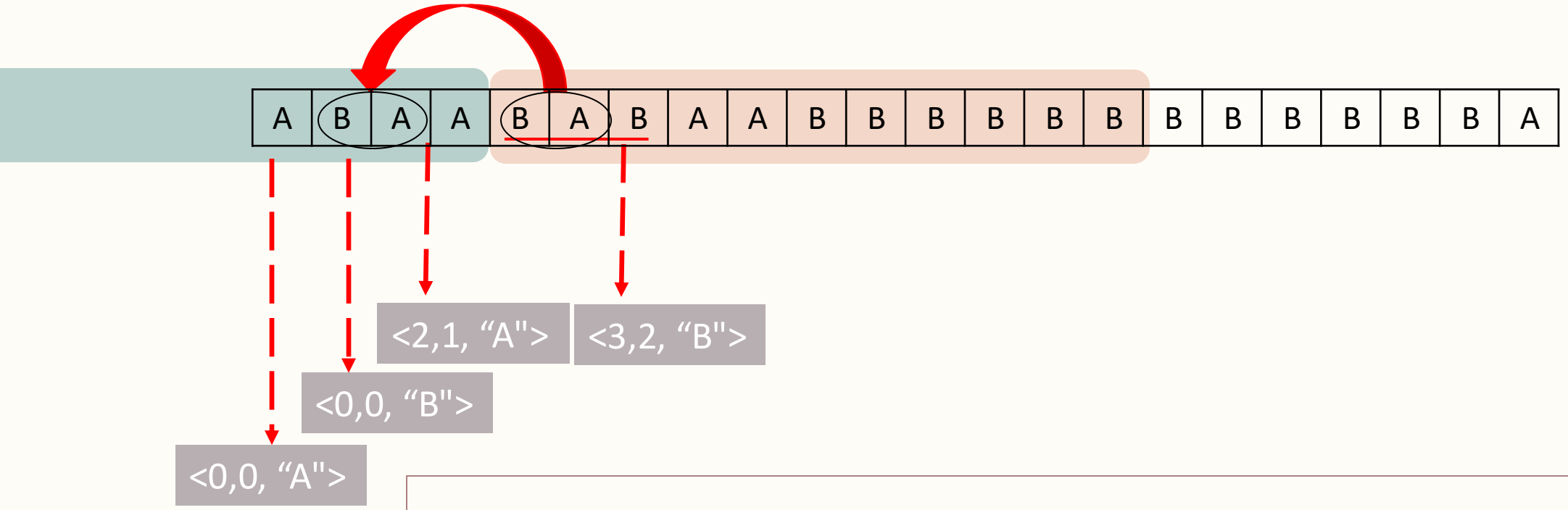
A existe dans search buffer

revenir en arrière de deux étapes, choisir un symbole

« position=2, longueur=1, caractère suivant = «"A"»

LZ77 (Compression)

- **Exemple:**



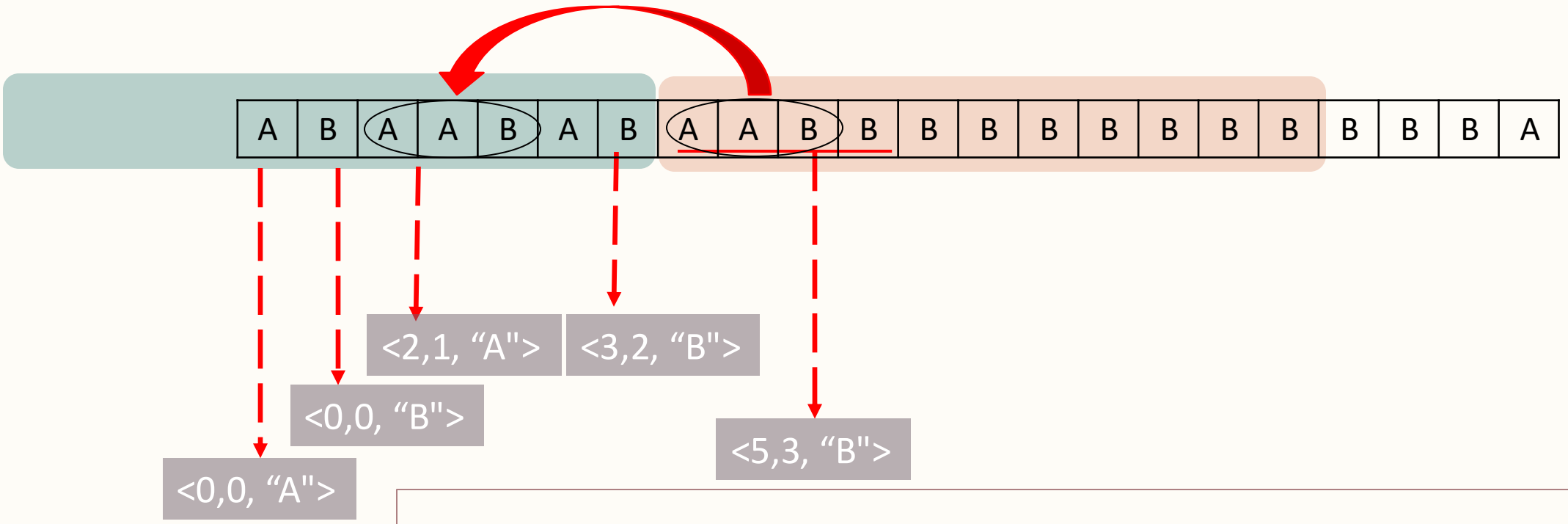
BA existe dans search buffer

revenir trois étapes en arrière, choisir deux symboles

« position=3, longueur=2, caractère suivant = «"B"»

LZ77 (Compression)

- **Exemple:**



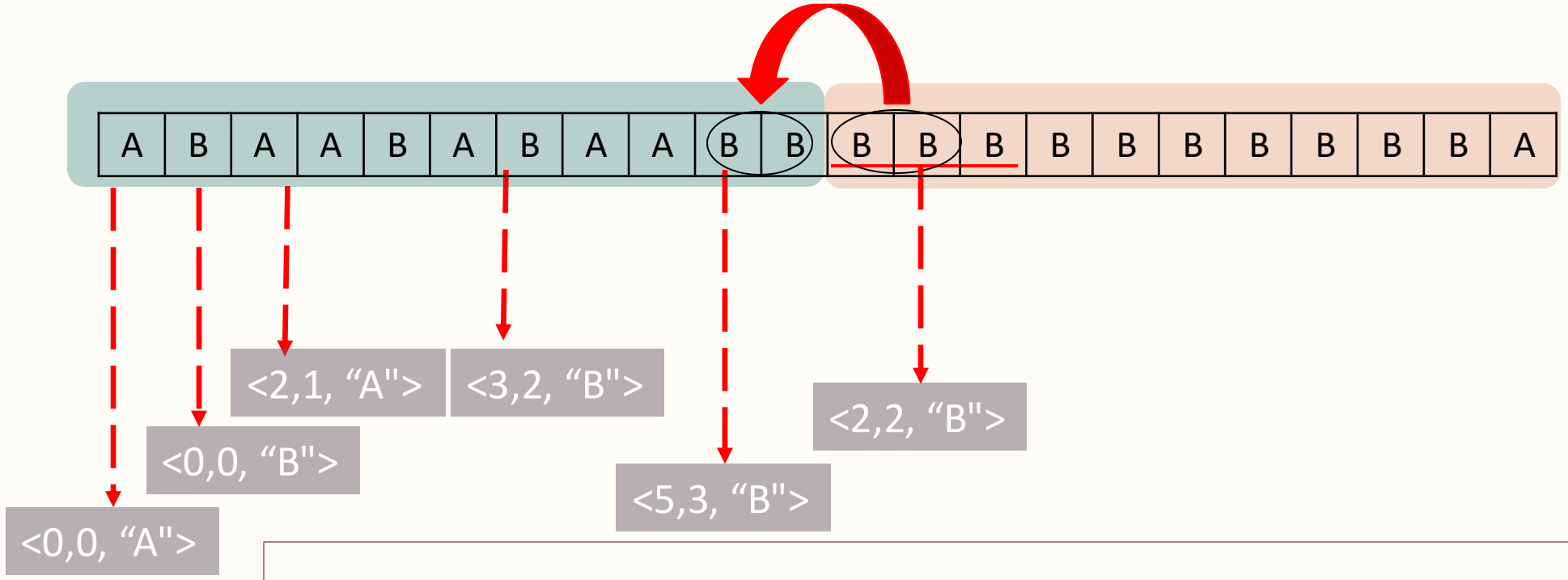
AAB existe dans search buffer

revenir cinq étapes en arrière, choisir trois symboles

« position=5, longueur=3, caractère suivant = «"B"»

LZ77 (Compression)

- **Exemple:**



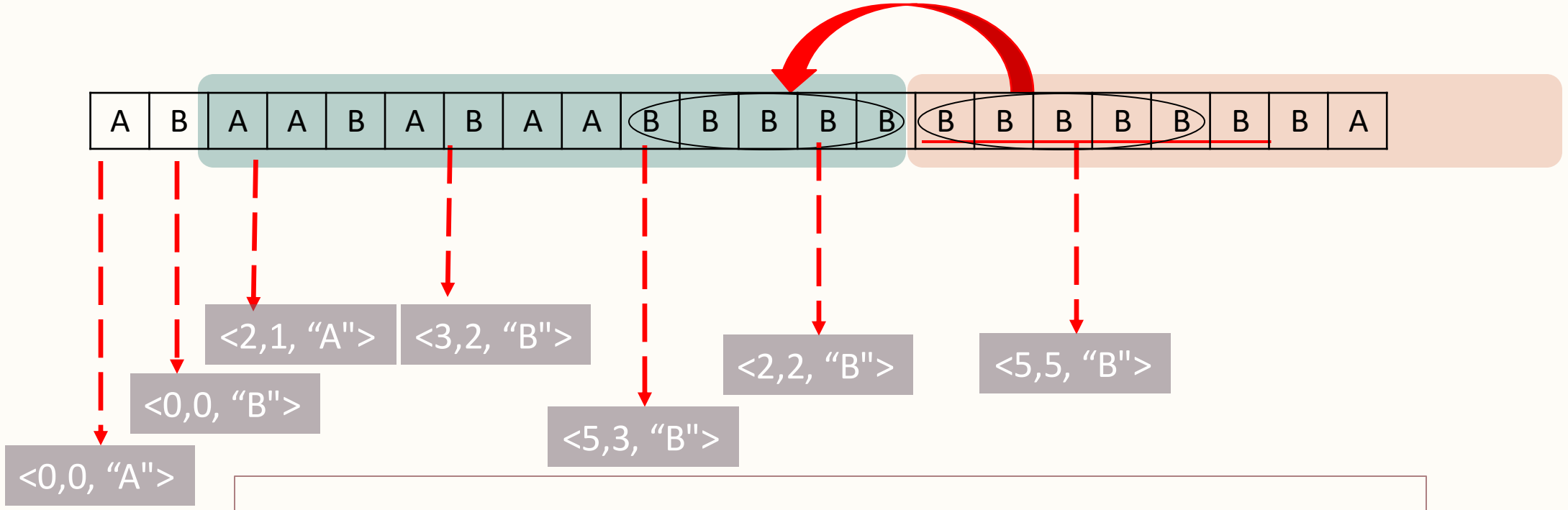
BB existe dans search buffer

revenir en arrière de deux étapes, choisir deux symboles

« position=2, longueur=2, caractère suivant = «"B"»

LZ77 (Compression)

- **Exemple:**



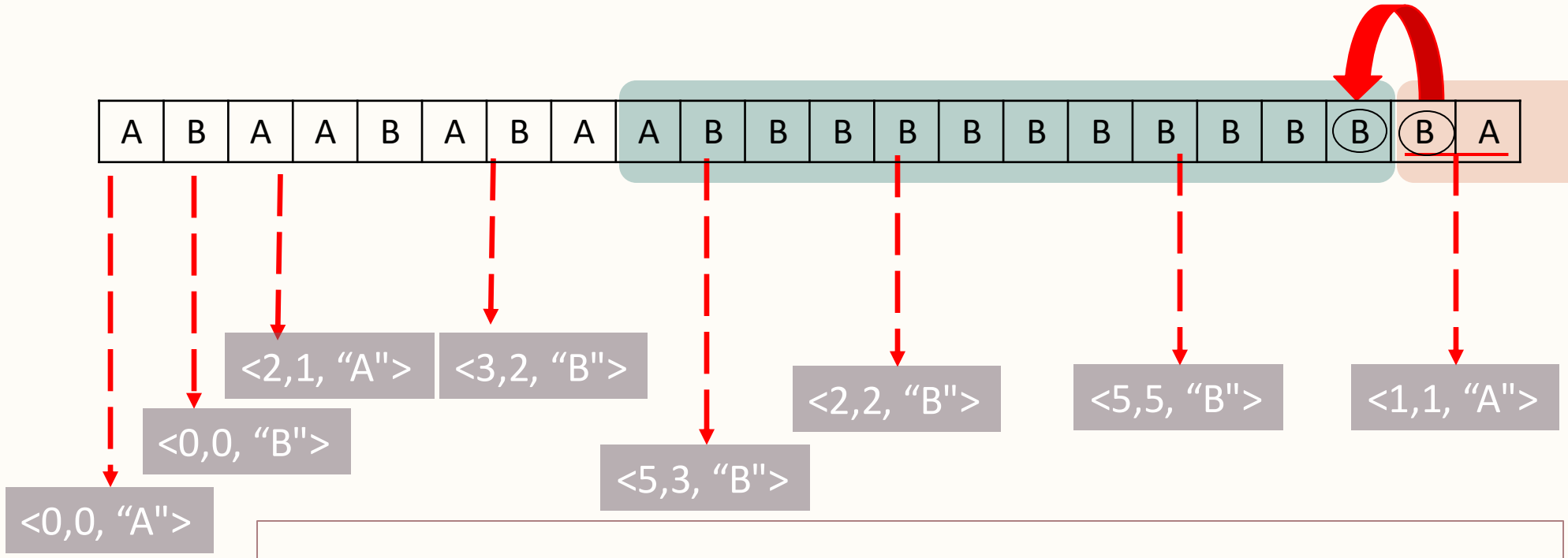
BBBBB existe dans search buffer

revenir cinq étapes en arrière, choisir cinq symboles

« position=5, longueur=5, caractère suivant = «"B"»

LZ77 (Compression)

- **Exemple:**



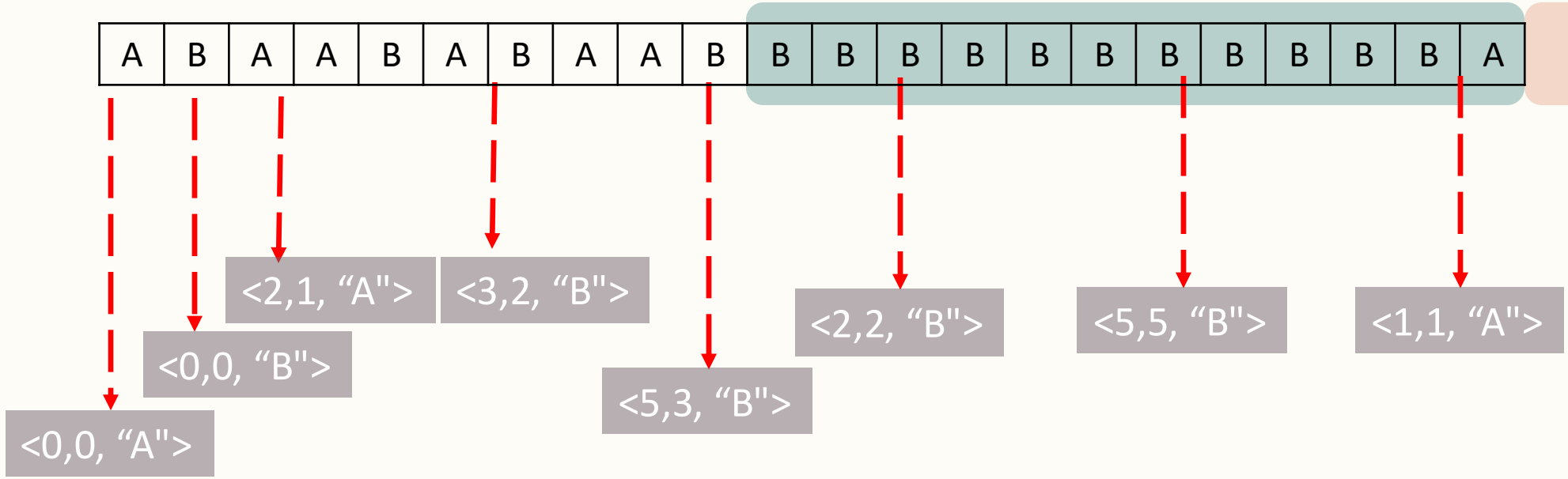
BBBBB existe dans search buffer

revenir en arrière, choisir un symbole

« position=1, longueur=1, caractère suivant = «"A"»

LZ77 (Compression)

- **Exemple:**



position maximale: 5 (**3 bits**)

longueur maximale : 5 (**3 bits**)

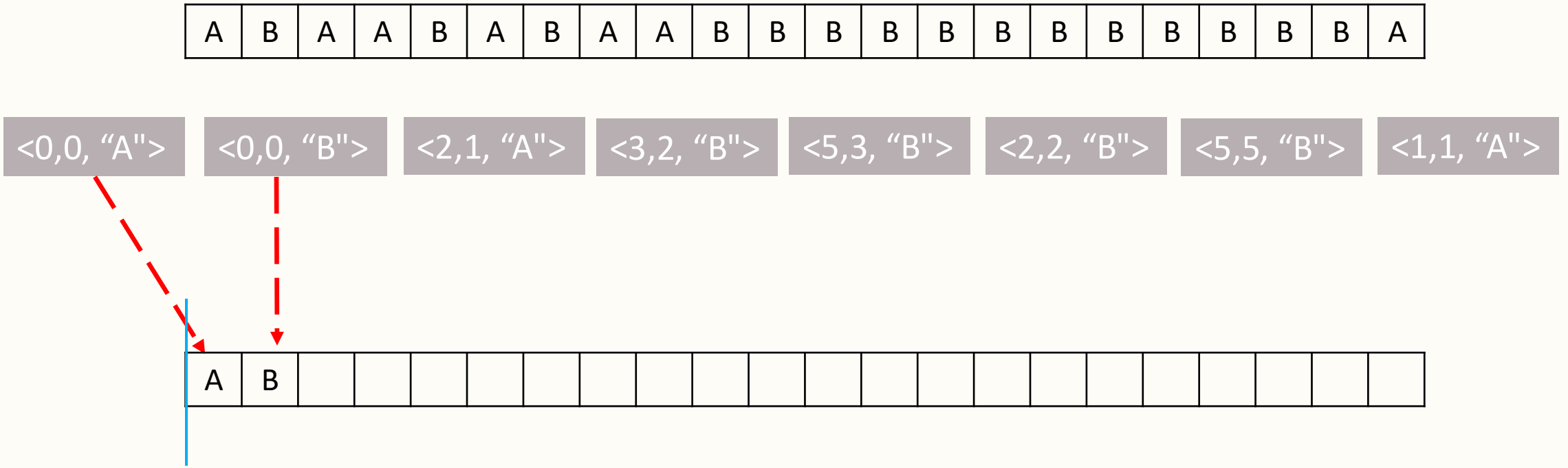
Symboles: 8 bits

3+3+8=14 bits

Taille compressée: 8*14= 112 bits , T= 0.63, G= 36%

LZ77 (Décompression)

- **Exemple:**



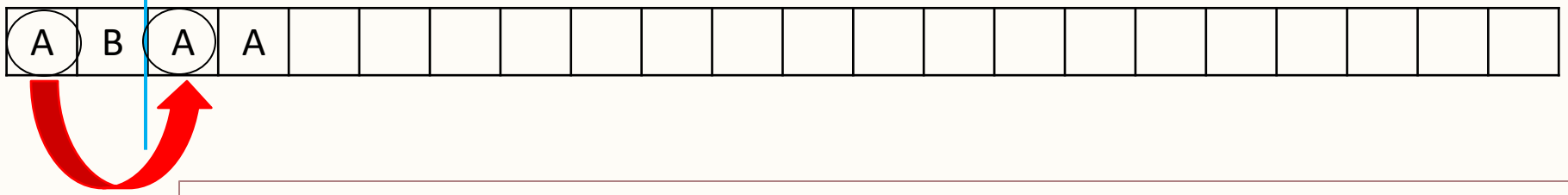
Ajouter les symboles A et B

LZ77 (Décompression)

- **Exemple:**

A	B	A	A	B	A	B	A	A	B	B	B	B	B	B	B	B	B	B	B	A

<0,0, “A"> <0,0, “B"> <2,1, “A"> <3,2, “B"> <5,3, “B"> <2,2, “B"> <5,5, “B"> <1,1, “A">



Revenir en arrière de 2 positions dans la fenêtre de recherche, Choisir

1 symbole,

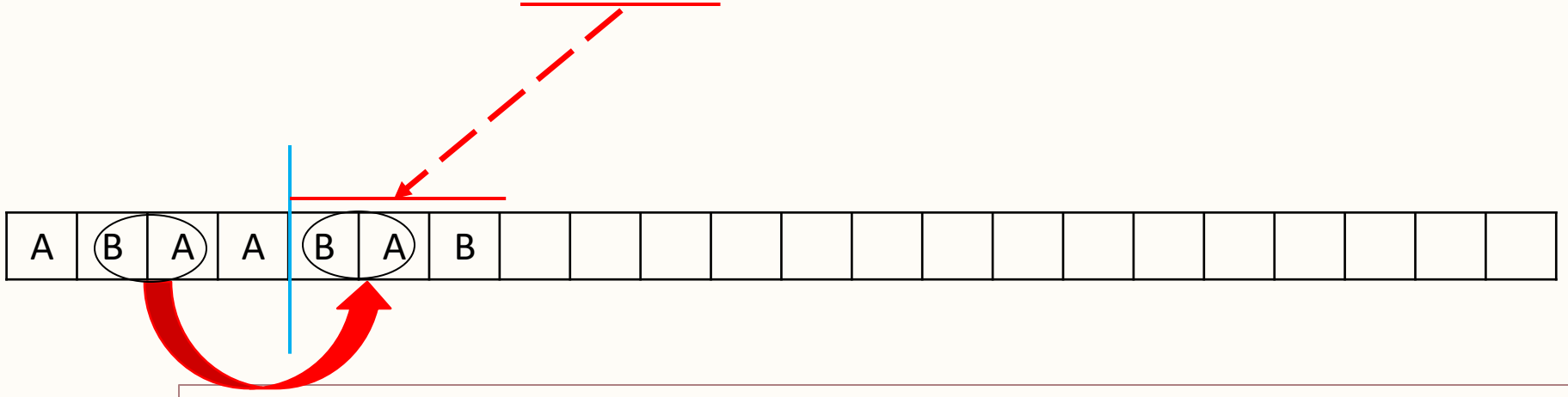
ajouter A

LZ77 (Décompression)

- **Exemple:**

A	B	A	A	B	A	B	A	A	B	B	B	B	B	B	B	B	B	B	B	A

<0,0, “A"> <0,0, “B"> <2,1, “A"> <3,2, “B"> <5,3, “B"> <2,2, “B"> <5,5, “B"> <1,1, “A">



Revenir 3 positions en arrière dans la fenêtre de recherche,

choisir 2 symboles,

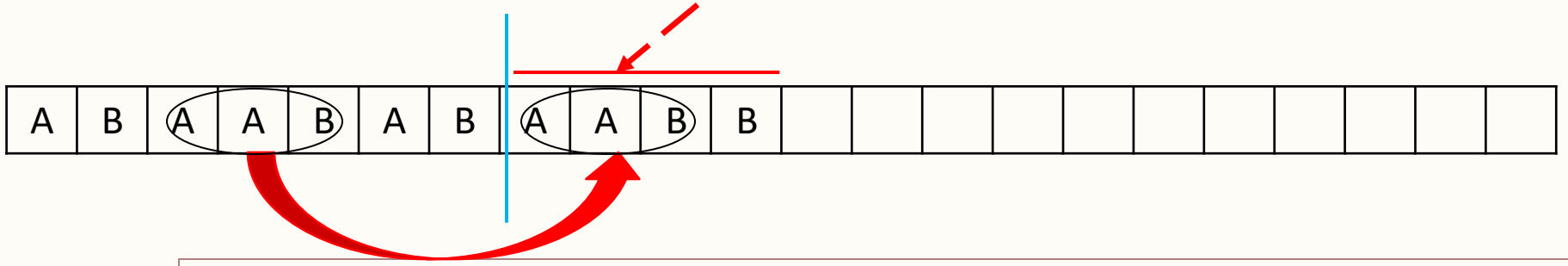
ajouter B

LZ77 (Décompression)

- **Exemple:**

A	B	A	A	B	A	B	A	A	B	B	B	B	B	B	B	B	B	B	B	A

<0,0, “A"> <0,0, “B"> <2,1, “A"> <3,2, “B"> <5,3, “B"> <2,2, “B"> <5,5, “B"> <1,1, “A">



Revenir 5 positions en arrière dans la fenêtre de recherche,

choisir 3 symboles,

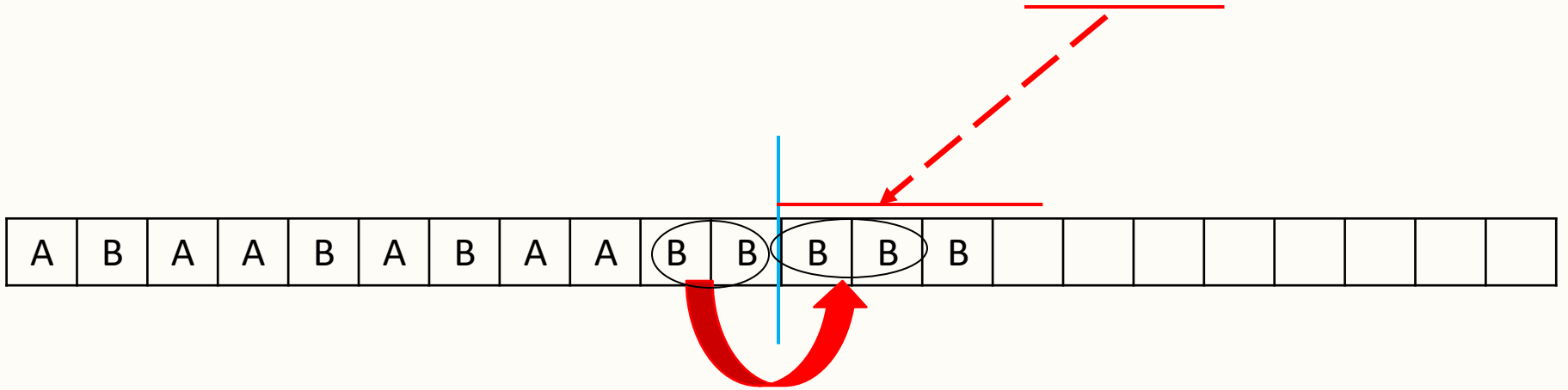
ajouter B

LZ77 (Décompression)

- **Exemple:**

A	B	A	A	B	A	B	A	A	B	B	B	B	B	B	B	B	B	B	B	A

<0,0, “A"> <0,0, “B"> <2,1, “A"> <3,2, “B"> <5,3, “B"> <2,2, “B"> <5,5, “B"> <1,1, “A">



Revenir 2 positions en arrière dans la fenêtre de recherche,

choisir 2 symboles,

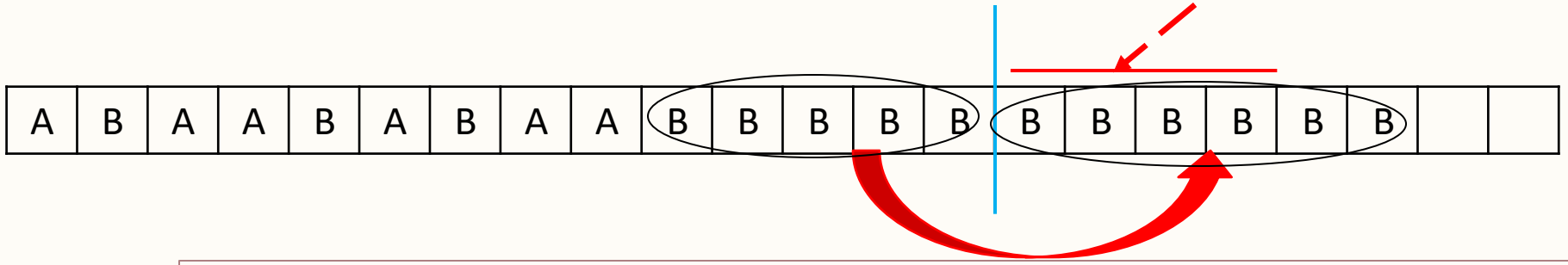
ajouter B

LZ77 (Décompression)

- **Exemple:**

A	B	A	A	B	A	B	A	A	B	B	B	B	B	B	B	B	B	B	B	A

<0,0, “A"> <0,0, “B"> <2,1, “A"> <3,2, “B"> <5,3, “B"> <2,2, “B"> <5,5, “B"> <1,1, “A">



Revenir 5 positions en arrière dans la fenêtre de recherche,

choisir 5 symboles,

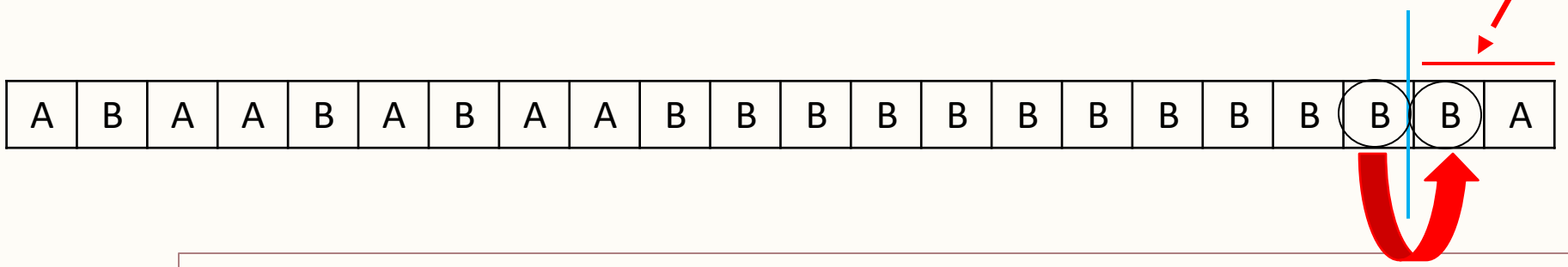
ajouter B

LZ77 (Décompression)

- **Exemple:**

A	B	A	A	B	A	B	A	A	B	B	B	B	B	B	B	B	B	B	B	A

<0,0, “A"> <0,0, “B"> <2,1, “A"> <3,2, “B"> <5,3, “B"> <2,2, “B"> <5,5, “B"> <1,1, “A">



Revenir 1 position en arrière dans la fenêtre de recherche,

choisir 1 symbole,

ajouter A

LZ77

- **Avantages:**

✓ Adapté à la compression en temps réel (il n'est pas nécessaire de connaître les probabilités des symboles).

✓ Plus la taille de la fenêtre glissante est longue, meilleur est le taux de compression des données.

✓ Aucune table de codage requise pour la décompression

- **Inconvénient:**

✓ La complexité de la comparaison est très grande

- **Application**: ZIP, PNG

LZ78

- **L’algorithme est le suivant :**

✓ LZ78 crée un dictionnaire vide.

✓ Il recherche la plus longue séquence de caractères correspondant à une entrée du dictionnaire.

✓ Si une nouvelle séquence est trouvée, l'algorithme génère une paire (**index, caractère**)

- **Index** : index de la plus longue séquence correspondante trouvée dans le dictionnaire

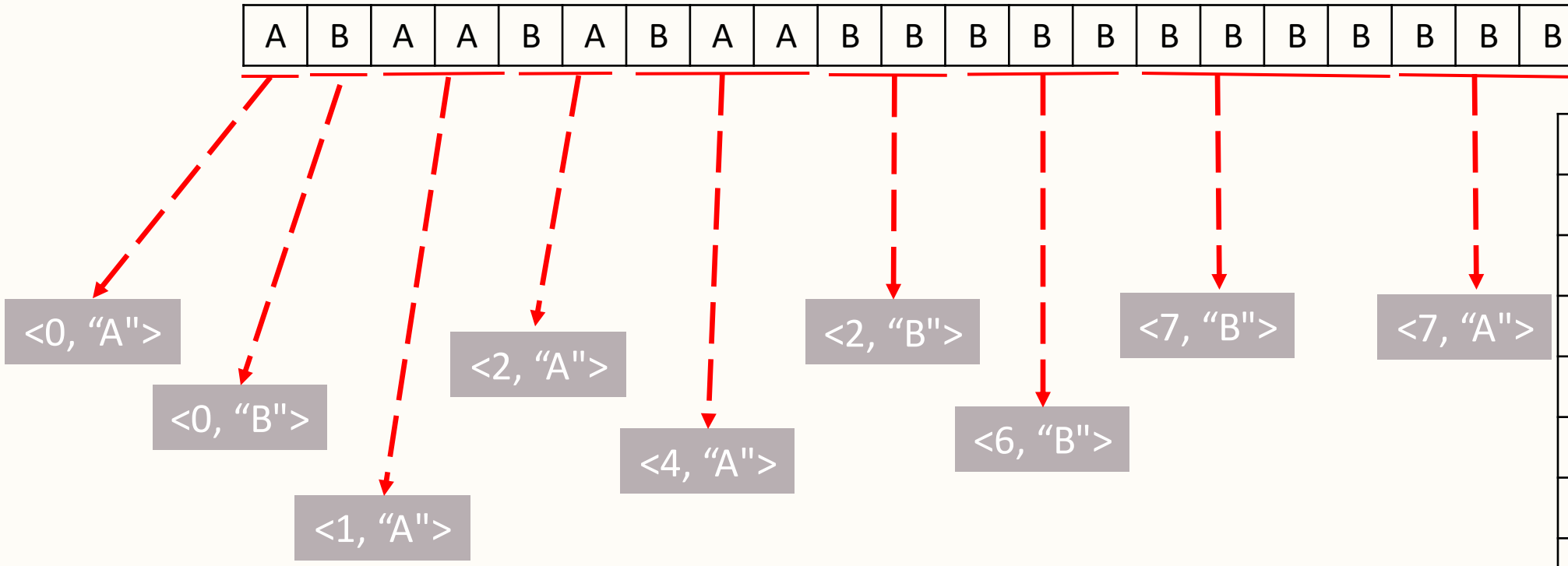
- **Caractère**: est le caractère suivant cette séquence dans les données d'entrée.

✓ Si aucune correspondance n'est trouvée dans le dictionnaire, l'index est défini sur 0.

✓ Ensuite, la nouvelle séquence (correspondance précédente + nouveau caractère) est ajoutée au dictionnaire avec un index unique.

LZ78 (Compression)

- **Exemple:**

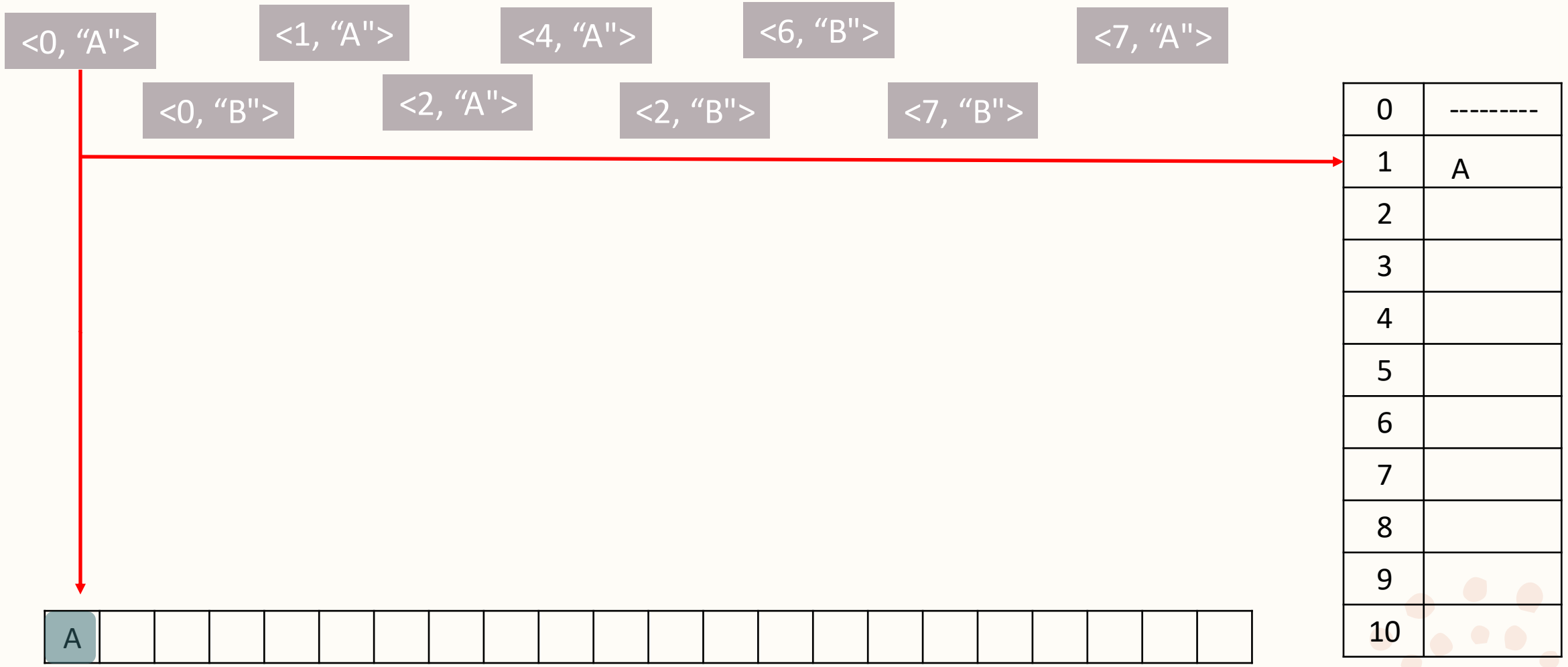


0	---------
1	A
2	B
3	AA
4	BA
5	BAA
6	BB
7	BBB
8	BBBB
9	BBBA
10	

Tag = <Index dans le dictionnaire, symbole suivant>

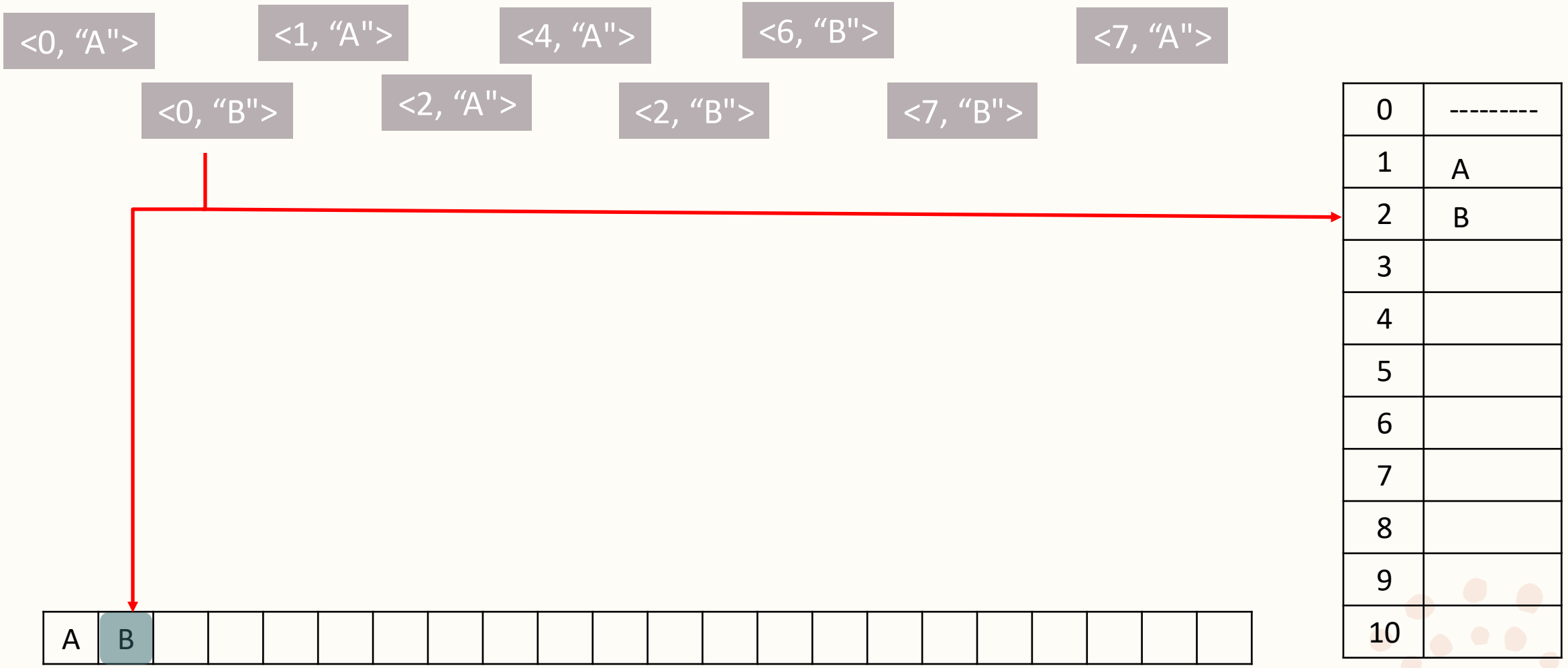
LZ78 (Décompression)

- **Exemple:**



LZ78 (Décompression)

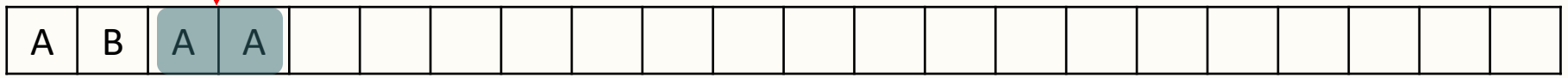
- **Exemple:**



LZ78 (Décompression)

- **Exemple:**

<0, “A">
<0, “B">
<1, “A">
<2, “A">
<4, “A">
<2, “B">
<6, “B">
<7, “B">
<7, “A">

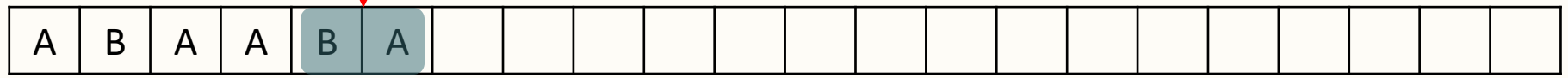


0	---------
1	A
2	B
3	AA
4	
5	
6	
7	
8	
9	
10	

LZ78 (Décompression)

- **Exemple:**

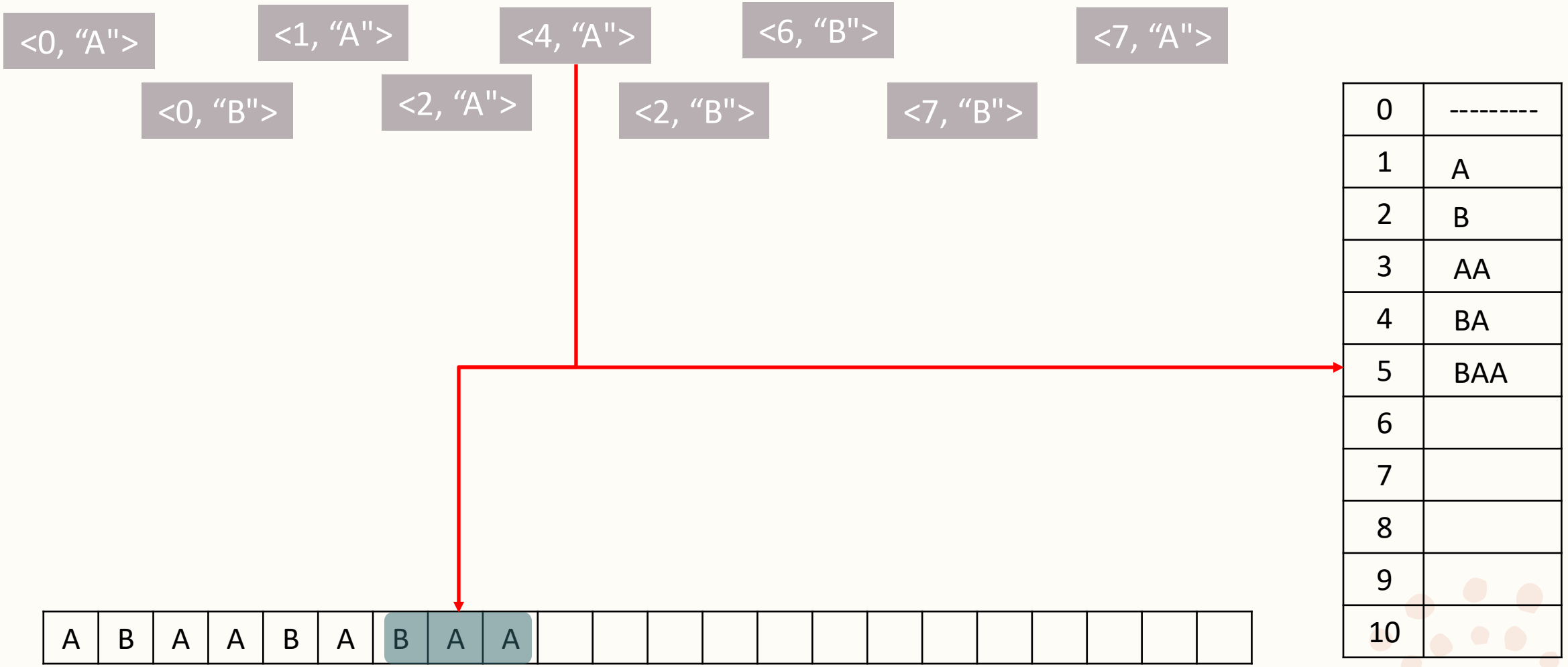
<0, “A">
<1, “A">
<4, “A">
<6, “B">
<7, “A">
<0, “B">
<2, “A">
<2, “B">
<7, “B">



0	---------
1	A
2	B
3	AA
4	BA
5	
6	
7	
8	
9	
10	

LZ78 (Décompression)

- **Exemple:**



LZ78 (Décompression)

- **Exemple:**

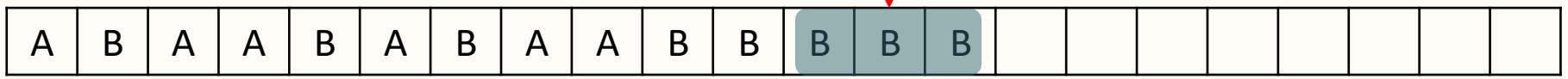
<0, “A">
<1, “A">
<4, “A">
<6, “B">
<7, “A">
<0, “B">
<2, “A">
<2, “B">
<7, “B">

0	---------
1	A
2	B
3	AA
4	BA
5	BAA
6	BB
7	
8	
9	
10	

LZ78 (Décompression)

- **Exemple:**

<0, “A">
<1, “A">
<4, “A">
<6, “B">
<7, “A">
<0, “B">
<2, “A">
<2, “B">
<7, “B">

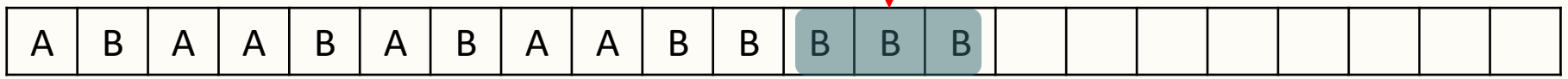


0	---------
1	A
2	B
3	AA
4	BA
5	BAA
6	BB
7	BBB
8	
9	
10	

LZ78 (Décompression)

- **Exemple:**

<0, “A">
<1, “A">
<4, “A">
<6, “B">
<7, “A">
<0, “B">
<2, “A">
<2, “B">
<7, “B">

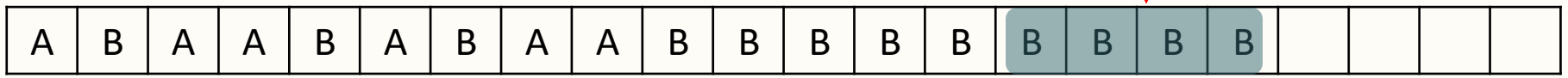


0	---------
1	A
2	B
3	AA
4	BA
5	BAA
6	BB
7	BBB
8	
9	
10	

LZ78 (Décompression)

- **Exemple:**

<0, “A">
<1, “A">
<4, “A">
<6, “B">
<7, “A">
<0, “B">
<2, “A">
<2, “B">
<7, “B">



0	---------
1	A
2	B
3	AA
4	BA
5	BAA
6	BB
7	BBB
8	BBBB
9	
10	

LZ78 (Décompression)

- **Exemple:**

<0, “A">
<0, “B">
<1, “A">
<2, “A">
<4, “A">
<2, “B">
<6, “B">
<7, “B">
<7, “A">

A	B	A	A	B	A	B	A	A	B	B	B	B	B	B	B	B	B	B	B	A

0	---------
1	A
2	B
3	AA
4	BA
5	BAA
6	BB
7	BBB
8	BBBB
9	BBBA
10	

LZ78 (Décompression)

- **Exemple:**

<0, “A">
<0, “B">
<1, “A">
<2, “A">
<4, “A">
<2, “B">
<6, “B">
<7, “B">
<7, “A">

A	B	A	A	B	A	B	A	A	B	B	B	B	B	B	B	B	B	B	B	A

Index maximale: 7 (**3 bits**)

Symboles: 8 bits

3+8=11 bits

Taille compressée: 9*11= 99 bits , T= 0.56, G= 44%

0	---------
1	A
2	B
3	AA
4	BA
5	BAA
6	BB
7	BBB
8	BBBB
9	BBBA
10	

LZ78

- **Avantages:**

✓ LZ78 est un algorithme de compression de données sans perte.

✓ Fonctionne efficacement avec des données plus répétitives.

✓ Rapide et simple à mettre en œuvre dans les phases d'encodage et de décodage.

✓ LZ78 est la base de LZW.

- **Inconvénients:**

✓ LZ78 est moins efficace avec les données non répétitives

✓ Le dictionnaire peut devenir volumineux, ce qui peut entraîner des inefficacités potentielles, et doit être réinitialisé périodiquement pour éviter qu'il ne consomme trop de mémoire.

Lempel Ziv Welch (LZW)

- ✓ LZW est un algorithme de compression de données sans perte « basé sur un dictionnaire ».
- ✓ Développé par Abraham Lempel, Jakob Ziv et Terry Welch.
- ✓ Construire un dictionnaire dans lequel chaque séquence à une adresse fixe (code)
- ✓ Initialement le dictionnaire contient les caractères avec les codes ASCII correspondants (255 caractères sur 8 bits)
- ✓ Chaque séquence ne se trouvant pas dans le dictionnaire doit être ajoutée
- ✓ Les séquences ont donc des adresses > 255

Lempel Ziv Welch (Compression)

Input: Dictionnaire Dico initial avec 256 chaînes de caractères uniques (et leurs codes ASCII) et le fichier à compresser

```
Prefix = first input character;
CodeWord = 256;
while(not end of character stream){
    Char = next input character;
    if(Prefix + Char exists in the Dictionary)
        Prefix = Prefix + Char;
    else{
        Output: the code for Prefix;
        insertInDictionary(  (CodeWord , Prefix + Char) ) ;
        CodeWord++;
        Prefix = Char;
    } }
Output: the code for Prefix;
```

Lempel Ziv Welch (Compression)

- ✓ **Exemple 1 : BABAABAAA**

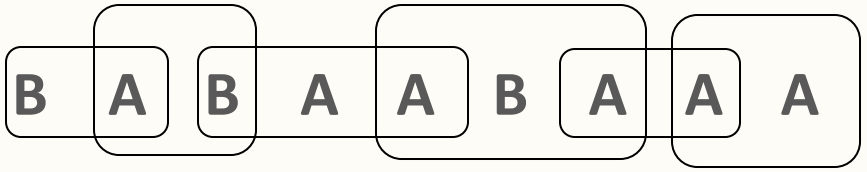
B A B A A B A A A

Dico	Output

le message compressé est :

Lempel Ziv Welch (Compression)

✓ **Exemple 1 : BABAABAAA**



Dico	Output
BA (256)	66
AB (257)	65
BAA (258)	256
ABA (259)	257
AA (260)	65
	260

le message compressé est : <66><65><256><257><65><260>

Lempel Ziv Welch (Compression)

✓ **Exemple 2 : BABAABRRRA**

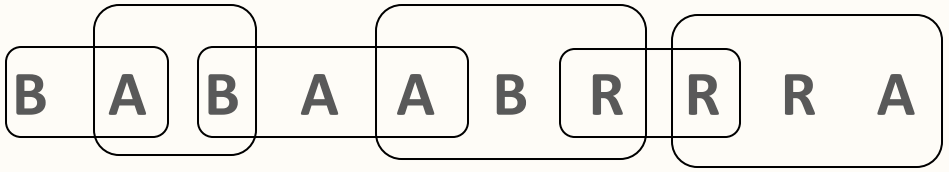
B A B A A B R R R A

Dico	Output

le message compressé est :

Lempel Ziv Welch (Compression)

- ✓ **Exemple 2 : BABAABRRRA**



Dico	Output
BA (256)	66
AB (257)	65
BAA (258)	256
ABR (259)	257
RR (260)	82
RRA (261)	260
	65

le message compressé est : <66><65><256><257><82><260> <65>

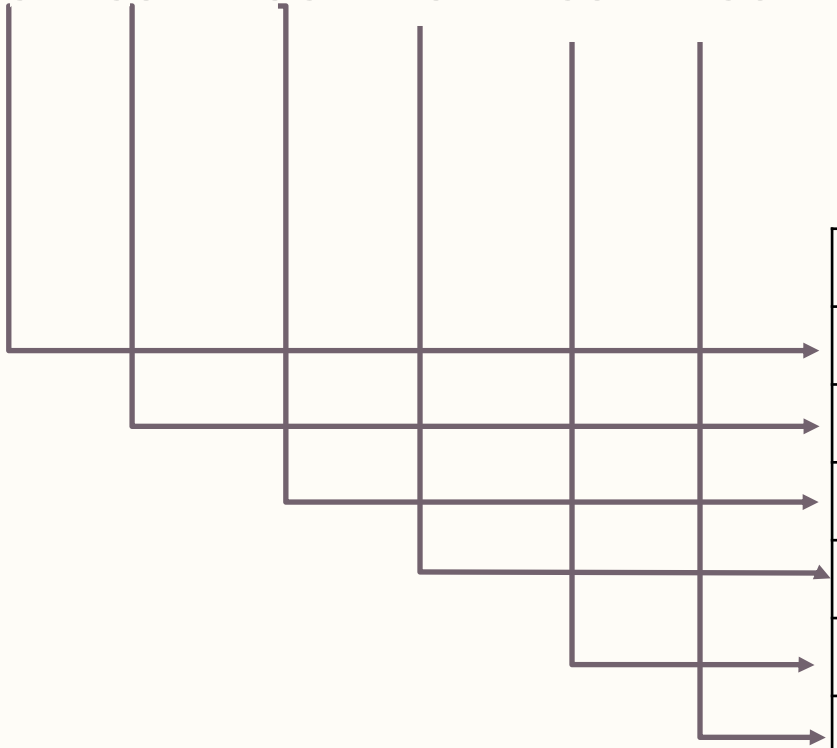
Lempel Ziv Welch (Décompression)

Input: Dictionnaire Dico initial avec 256 chaînes de caractères uniques et leurs codes ASCII)

```
PreviousCodeWord = first input code;
Output: string(PreviousCodeWord) ;
Char = character(first input code);
CodeWord = 256;
while(not end of code stream){
        CurrentCodeWord = next input code ;
        if(CurrentCodeWord exists in the Dictionary)
            String = string(CurrentCodeWord) ;
        else
            String = string(PreviousCodeWord) + Char ;
        Output: String;
        Char =  first character of String ;
        insertInDictionary(  (CodeWord , string(PreviousCodeWord) + Char )   );
        PreviousCodeWord = CurrentCodeWord ;
        CodeWord++ ;}
```

Lempel Ziv Welch (Décompression)

✓ Exemple: <66><65><256><257><65><260>



	Dico	
Output	index	string
B		
A	256	BA
BA	257	AB
AB	258	BAA
A	259	ABA
AA	260	AA

✓ le message décompressé est : **BABAABAAA**

T = 6*9/9*8 = 54/72 = 0.75 | G= 1-T = 0.25

Avantages et Inconvénients

- **Avantages:**

✓ Meilleure technique pour réduire la taille des fichiers contenant des données plus répétitives.

✓ Technique de compression simple et rapide avec un taux élevé.

✓ L'algorithme de décompression suit toujours l'algorithme de compression.

✓ LZW est efficace (il n'est pas nécessaire de passer la table de chaînes au code de décompression, la table peut être recréée telle qu'elle était lors de la compression).

Inconvénient:

✓ Crée des entrées dans le dictionnaire qui ne seront peut-être jamais utilisées.

Applications de LZW

- LZW peut être utilisé dans différents formats de fichiers :

✓ TIFF: tagged image file format

✓ GIF: graphic iterchange format

✓ PDF: portable document format

✓ gzip

✓ Fichiers texte

Images GIF (Graphic Iterchange Format)

- La compression GIF est basée sur la combinaison de l'optimisation de la palette de couleurs et de LZW pour réduire la taille du fichier sans perdre la qualité de l'image.
- Le processus de compression GIF est basé sur les étapes suivantes :
  - ✓ **L’optimisation de la palette de couleurs**
  - ✓ **Compression LZW**
  - ✓ **Structure d'un fichier GIF**

Images GIF

- **L’optimisation de la palette de couleurs:**

✓ Le GIF utilise une palette de couleurs (table de couleurs) qui comporte 256 couleurs (8 bits). Si une image comporte plus de 256 couleurs, le GIF réduit la palette de couleurs à 256 à l'aide d'algorithmes de quantification (ex: Median Cut), qui regroupent les couleurs similaires et choisissent des couleurs représentatives pour chaque groupe.

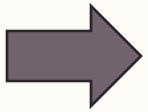
✓ Ensuite, l'image est décomposée en pixels, puis chaque couleur de pixel est mappée à un index dans la palette de couleurs. Cela permet aux données d'image d'être stockées sous forme d'une séquence d'index de couleurs plutôt que de valeurs de couleurs complètes.

Images GIF

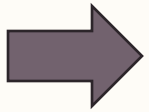
- **Compression LZW**

✓ Exemple:

Red	Blue	Red
Blue	Red	Green
Red	Red	Green



0	1	0
1	0	2
0	0	2



{0 1 0 1 0 2 0 0 2}

✓ A table des couleurs peut mapper : - `Rouge` à l'index `0` - `Bleu` à l'index `1` - `Vert` à l'index `2`.

✓ Initialiser le dictionnaire avec ‘0’, ‘1’, ‘2’, représentant chaque couleur de la palette de couleurs

✓ Image compressée: : <0><1><3><0><2><0><6>

✓ T= 7*3/9*8= 0.29, G= 71%

Dico	Output
01 (3)	0
10 (4)	1
010 (5)	3
02 (6)	0
20(7)	2
00(8)	0
	6

Images GIF

- **Structure d'un fichier GIF:**

✓ Le format de fichier GIF organise les données en blocs :

- ***En-tête :*** définit le type et la version du fichier***.***
- ***En-tête et descripteur d'écran logique*** : inclut des informations telles que la largeur et la hauteur, la résolution des couleurs et la couleur d'arrière-plan.
- ***Table des couleurs globales*** : stocke la palette de couleurs.
- ***Bloc de données d'image*** : contient les données compressées LZW pour chaque *frame* du GIF.
- ***Trailer*** : marque la fin du fichier GIF.

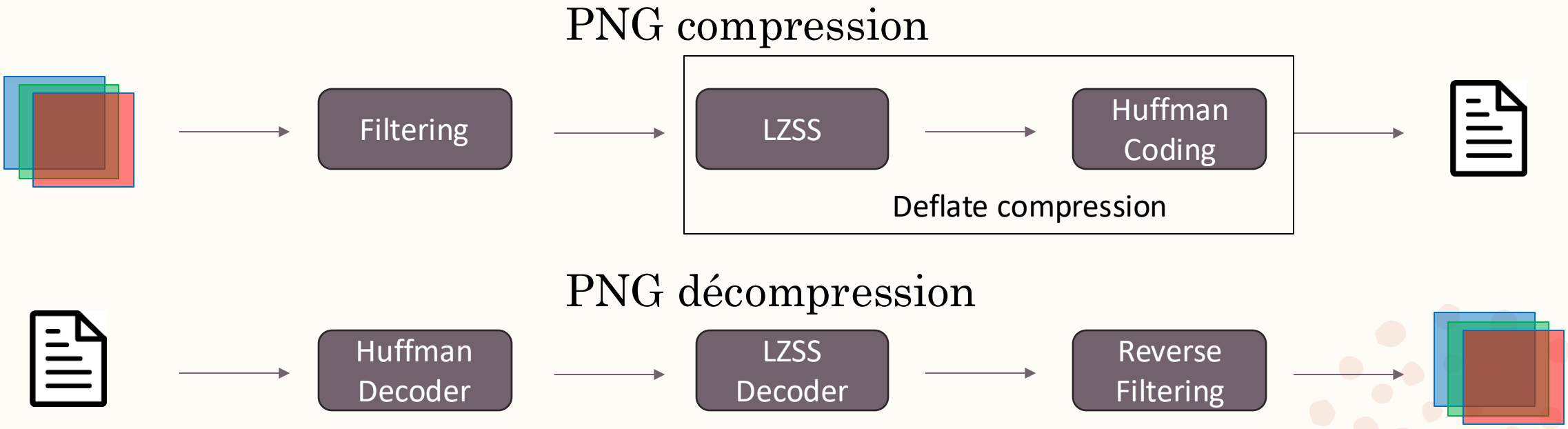
Images PNG (Portable Network Graphics)

- Un PNG est compressé en trois étapes :

✓ **Pré-filtrage**

✓ **Codage basé sur un dictionnaire via LZSS**

✓ **Codage d'entropie à l'aide de l'algorithme de Huffman.**



Pré-filtrage

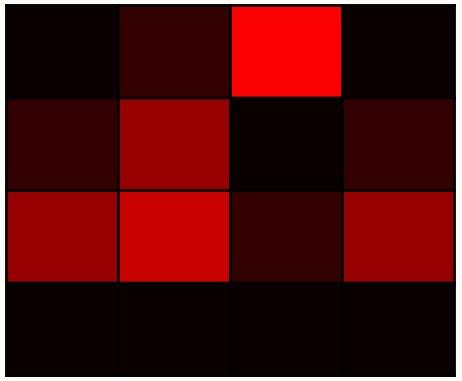
- **Pré-filtrage:**

✓ PNG applique un processus appelé filtrage, qui tente de simplifier les données. Le filtrage fonctionne en prédisant la couleur de chaque pixel en fonction de ses voisins et en stockant la différence entre la prédiction et la valeur de couleur réelle, qui a tendance à être plus petite et plus cohérente que les valeurs de pixels brutes.

✓ PNG dispose de cinq filtres différents (par exemple, Sub, Up, Average, Paeth, None) qui sont choisis de manière adaptative pour chaque ligne de pixels afin d'obtenir la meilleure compression

Pré-filtrage

- ✓ Exemple: NONE, SUB

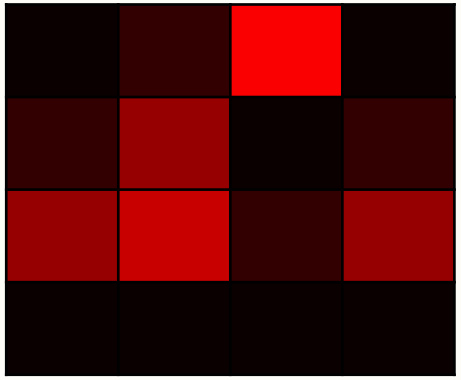


10	50	250	10
50	150	10	50
150	200	50	150
10	10	10	10

NONE

$x_{out} = \mathrm{X}$

10	50	250	10



10	50	250	10
50	150	10	50
150	200	50	150
10	10	10	10

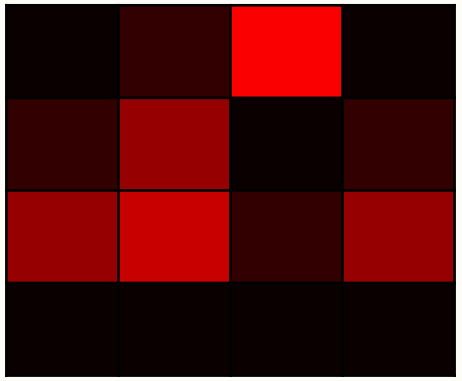
SUB

$x_{out} = \mathrm{X} - \mathrm{L}$

10	40	200	-240
50	100	-140	40

Pré-filtrage

- ✓ Exemple: NONE, SUB
- ✓ Le filtre png prend la valeur de l'opération mathématique et s'assure qu'elle se situe dans la plage de 8 bits représentables. Ceci est fait en: (X-L) mod 256

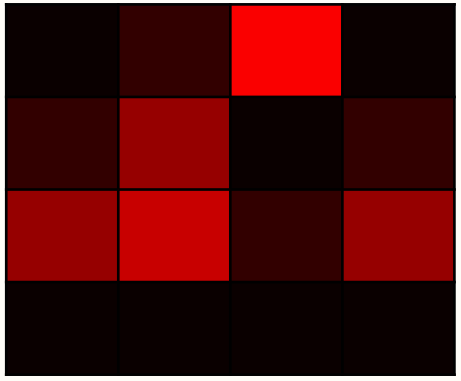


10	50	250	10
50	150	10	50
150	200	50	150
10	10	10	10

NONE

$x_{out} = X$

10	50	250	10



10	50	250	10
50	150	10	50
150	200	50	150
10	10	10	10

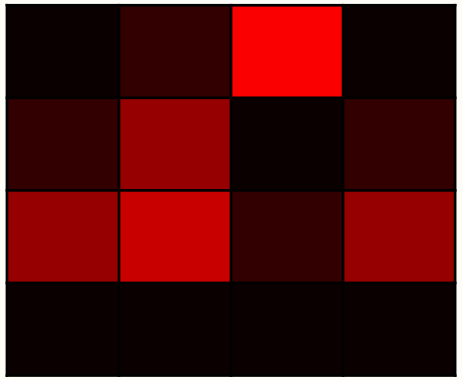
SUB

$x_{out} = X - L$

10	40	200	-240
50	100	116	40

Pré-filtrage

- ✓ Exemple: UP, AVG
- ✓ (X-(U+L)/2) mod 256

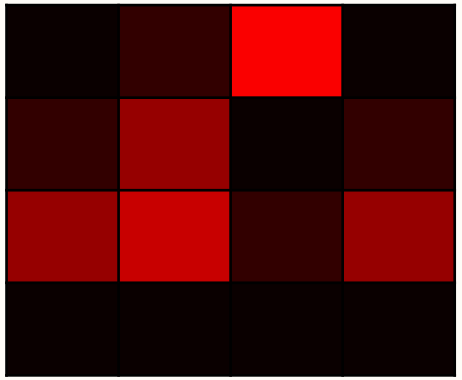


10	50	250	10
50	150	10	50
150	200	50	150
10	10	10	10

UP

$$x_{out} = X - U$$

10	50	250	10
50	100	116	40
100	50	40	100



10	50	250	10
50	150	10	50
150	200	50	150
10	10	10	10

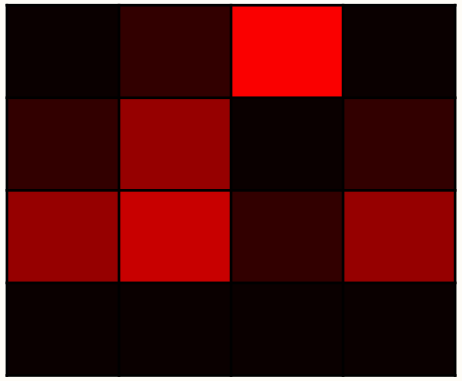
AVG

$$x_{out} = \mathrm{X} - \frac{U + L}{2}$$

10	50	250	10
50	100	116	40
100	50	40	100
10	161	236	186

Pré-filtrage

- ✓ **Exemple: AVG, PAETH**



10	50	250	10
50	150	10	50
150	200	50	150
10	10	10	10

PAETH

V=U+L-UL

$$x_{out} = X - MIN$$

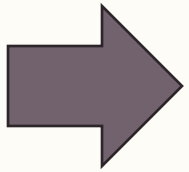
10	50	250	10
50	100	116	40
100	50	40	100
10	116	66	116

$$\left.\begin{aligned} V_L &= V - L \\ V_U &= V - U \\ V_{UL} &= V - UL \end{aligned}\right\} \text{MIN}$$

Algorithme DEFLATE

- ✓ **Pré-filtrage: NONE**

10	50	250	10
50	150	10	50
150	200	50	150
10	10	10	10



{10 50 250 10 50 150 10 50 150 200 50 150 10 10 10 10}

Algorithme DEFLATE

- **Algorithme DEFLATE:**

✓ DEFLATE réduit efficacement la taille des fichiers en combinant l'algorithme LZSS avec le codage Huffman.

✓ L'algorithme de compression DEFLATE est appliqué bloc par bloc dans chaque image. La taille du bloc est variable, avec un maximum de 64 Ko par bloc.

✓ La taille de lookahead window est 258 octets et la taille search window est généralement définie sur 32 Ko.

Algorithme DEFLATE

- **Algorithme DEFLATE:**

✓ Deflate utilise le codage Huffman pour réduire la longueur globale des bits des données compressées.

✓ Il existe des arbres Huffman dynamiques et statiques :

- **Codage Huffman statique** utilise un arbre Huffman prédéfini qui est le même pour tous les fichiers.
- **Codage Huffman dynamique** : DEFLATE calcule la fréquence de chaque symbole (littéral, longueur et distance) dans le bloc de données. Il construit ensuite deux arbres Huffman : arbre littéral/longueur et arbre distance. Il nécessite également de stocker la structure de l'arbre dans le fichier compressé, ce qui ajoute une légère taille.

Algorithme DEFLATE

- **LZSS (Lempel Ziv Storer Szymanski):**

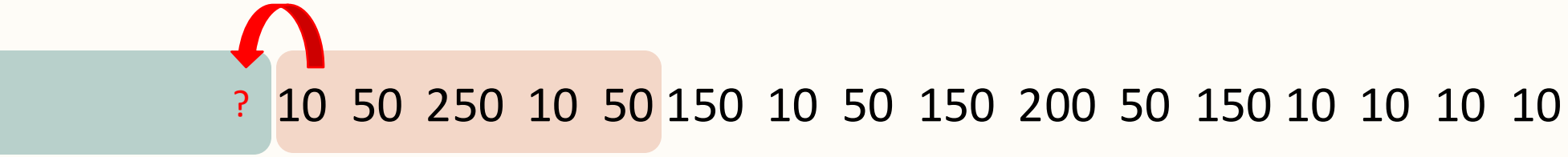
✓ Développé par James storer and Thomas Szymanski en 1982.

✓ Dérivé de LZ77

1. l'algorithme vérifie si une séquence de caractères dans **lookahead buffer** correspond à une séquence précédente dans **search buffer.**
2. Si une correspondance est trouvée, LZSS l'encode avec un triple **« flag, position, longueur».**
3. Si un caractère ne correspond à aucune séquence précédente, il est ajouté sous forme de « **flag**, **littéral »**

Algorithme DEFLATE

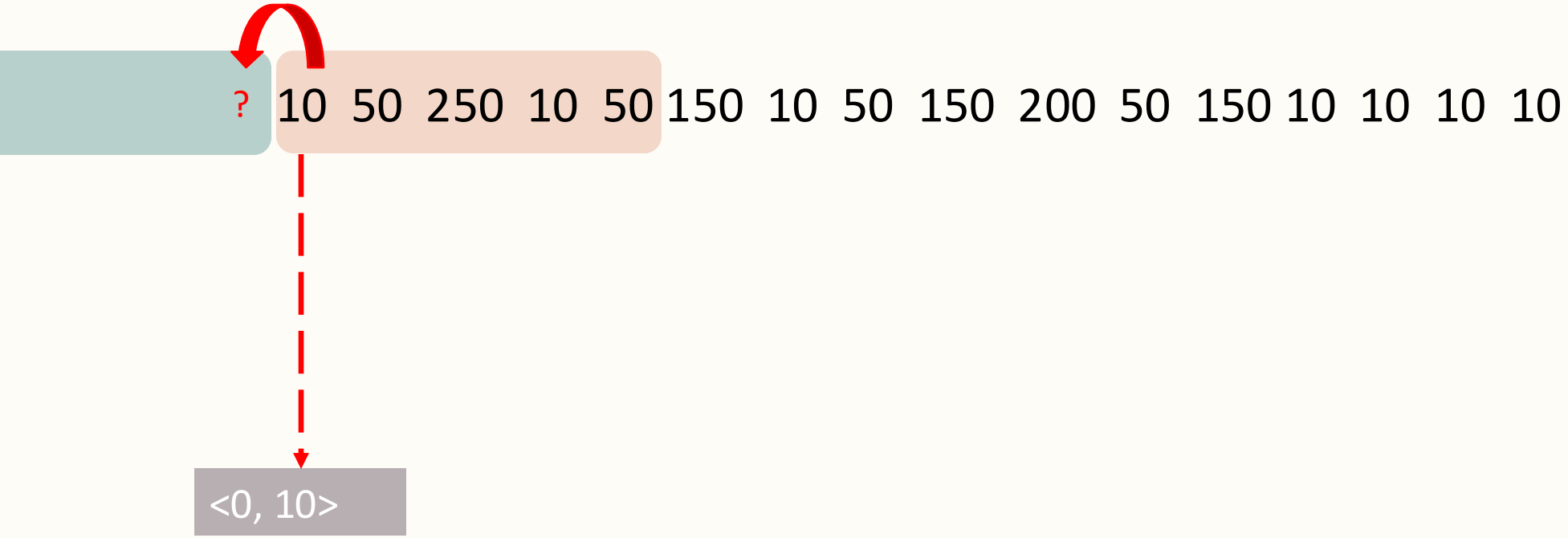
- **Exemple : LZSS**



- ✓ L'encode avec une paire (flag, littéral) ou un triple « Flag, position, longueur».
- ✓ **Flag** : 0 signifie aucune correspondance, 1 correspondance
- ✓ **Search window: 8 bytes, lookahead window: 5 bytes**

Algorithme DEFLATE

- Exemple : LZSS

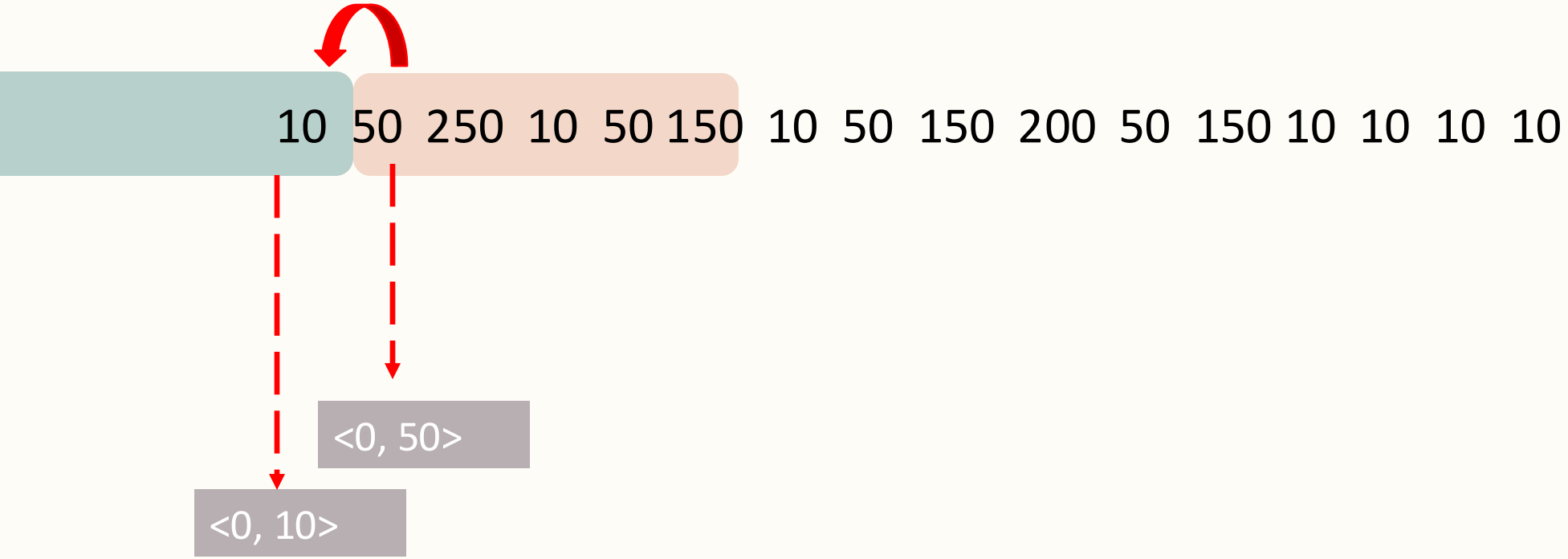


Il n'y a pas de 10 dans search buffer

« flags=0, littéral = 10»

Algorithme DEFLATE

- Exemple : LZSS

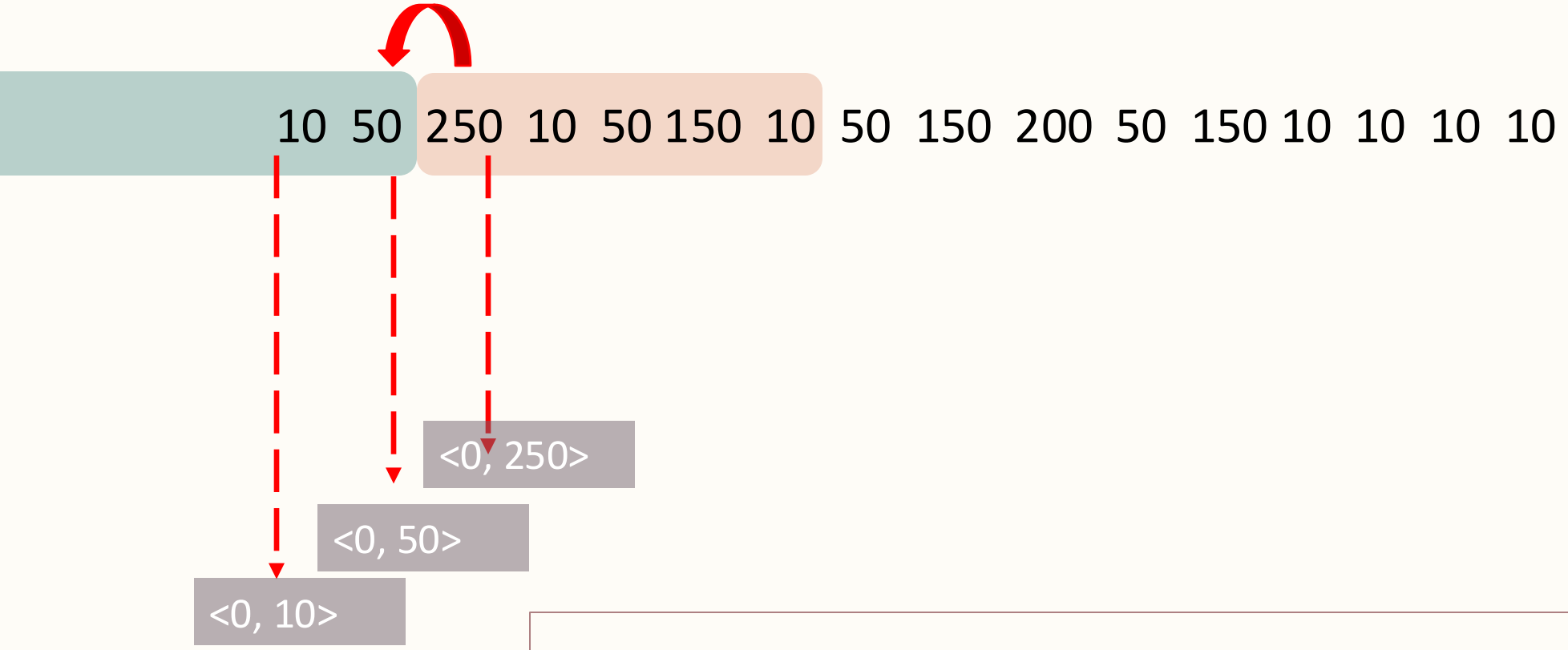


Il n'y a pas de 50 dans search buffer

« flags=0, littéral = 50»

Algorithme DEFLATE

- Exemple : LZSS

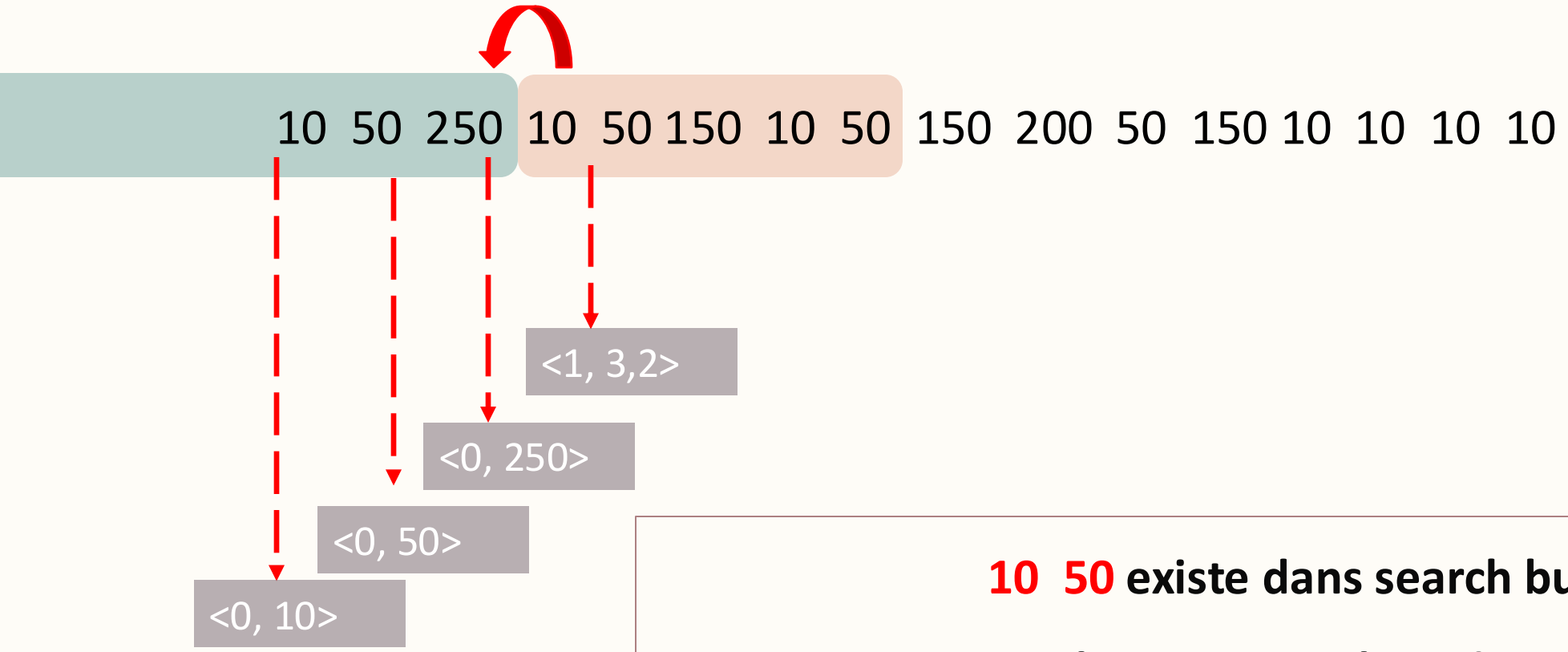


Il n'y a pas de 250 dans search buffer

« flags=0, littéral = 250»

Algorithme DEFLATE

- Exemple : LZSS



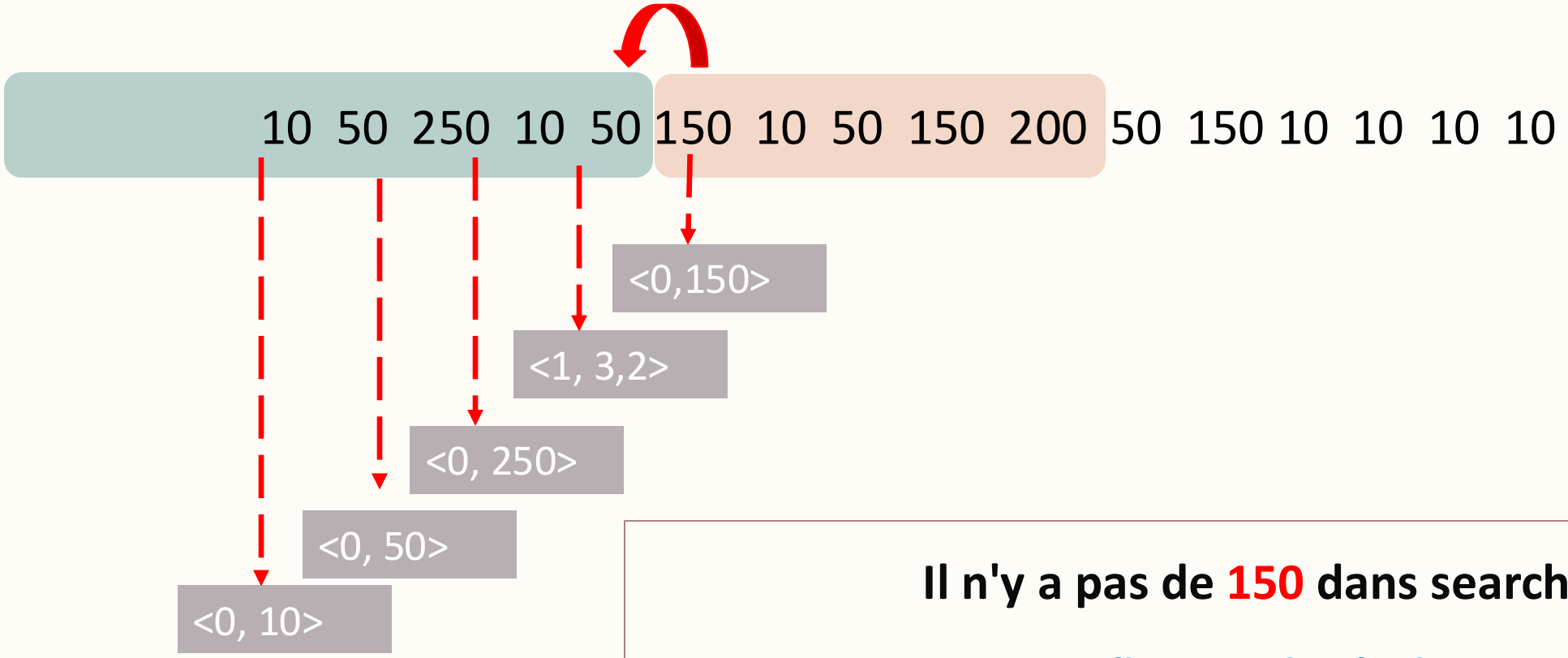
10 50 existe dans search buffer

revenir trois étapes en arrière, choisir un symbole

« flags=1, position=3, longueur=2»

Algorithme DEFLATE

- Exemple : LZSS

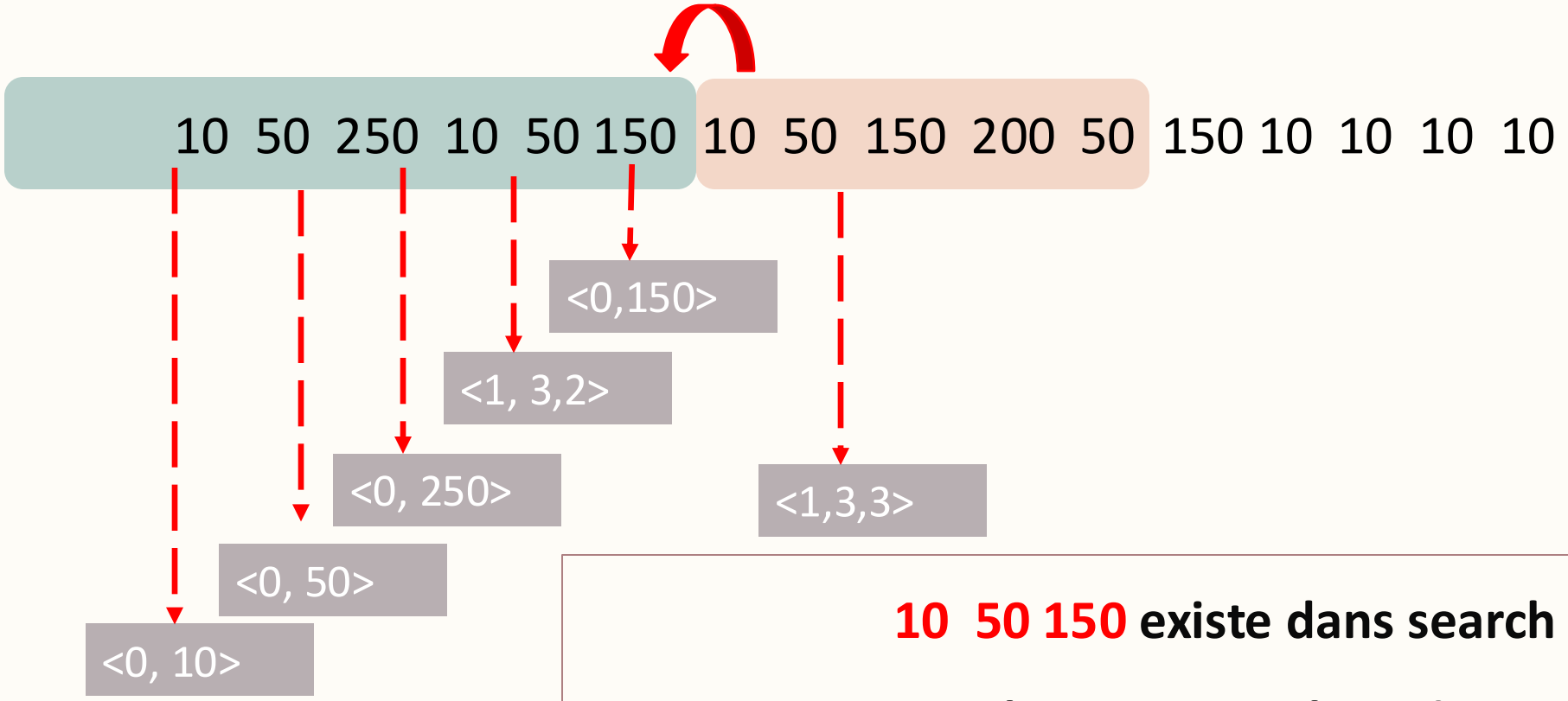


Il n'y a pas de 150 dans search buffer

« flags=0, littéral =150»

Algorithme DEFLATE

- Exemple : LZSS



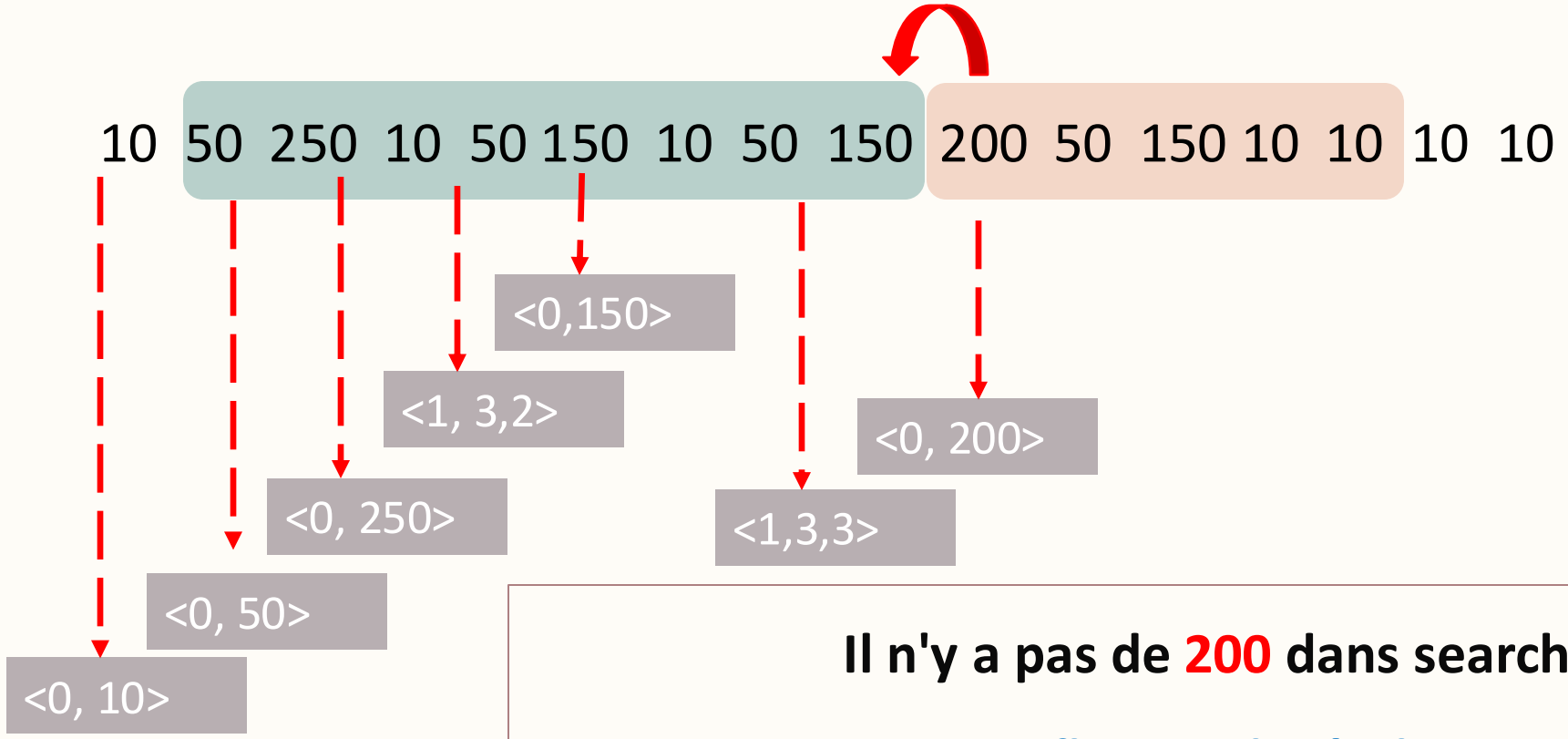
10 50 150 existe dans search buffer

revenir trois étapes en arrière, choisir trois symboles

« flags=1, position=3, longueur=3»

Algorithme DEFLATE

- Exemple : LZSS

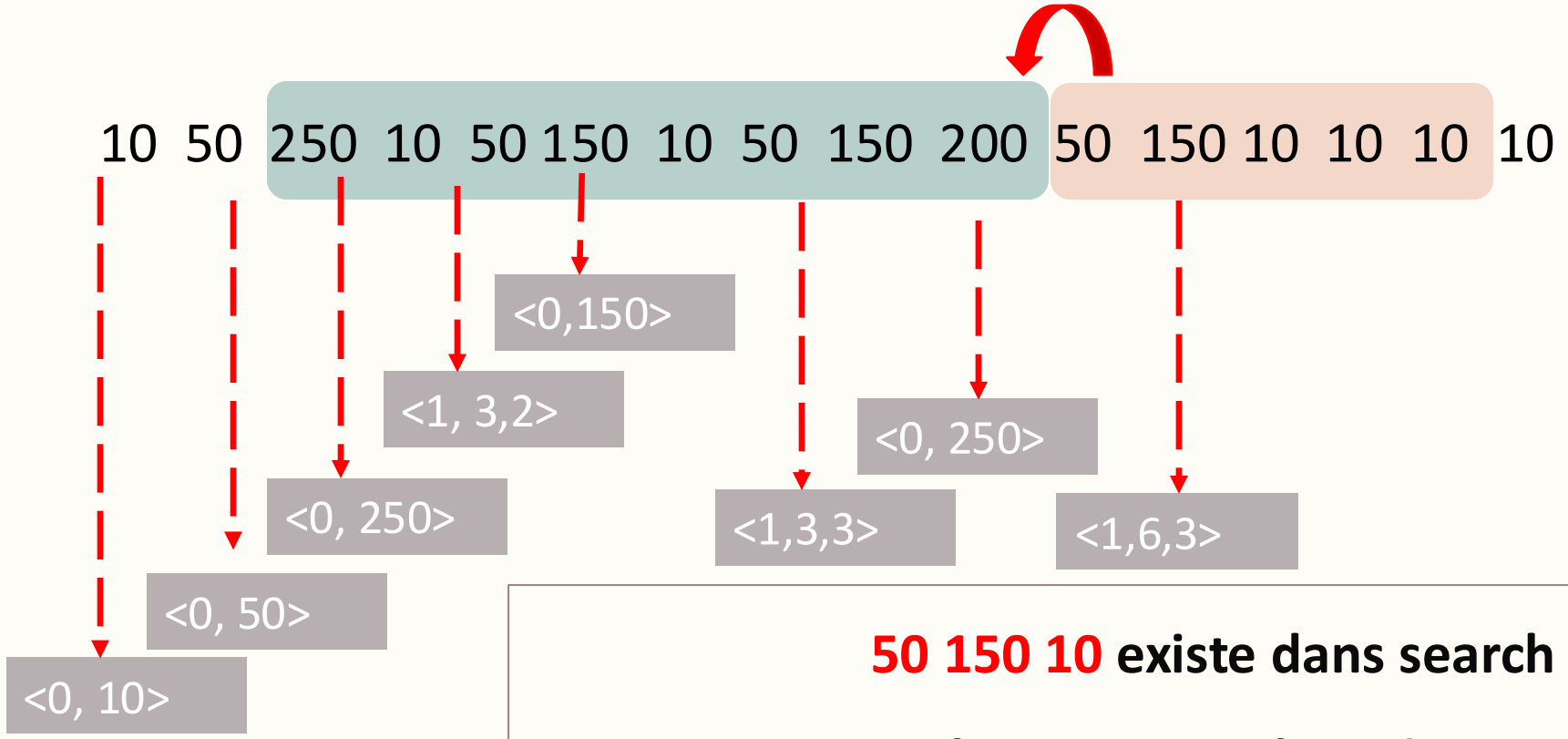


Il n'y a pas de 200 dans search buffer

« flags=0, littéral =200»

Algorithme DEFLATE

- Exemple : LZSS



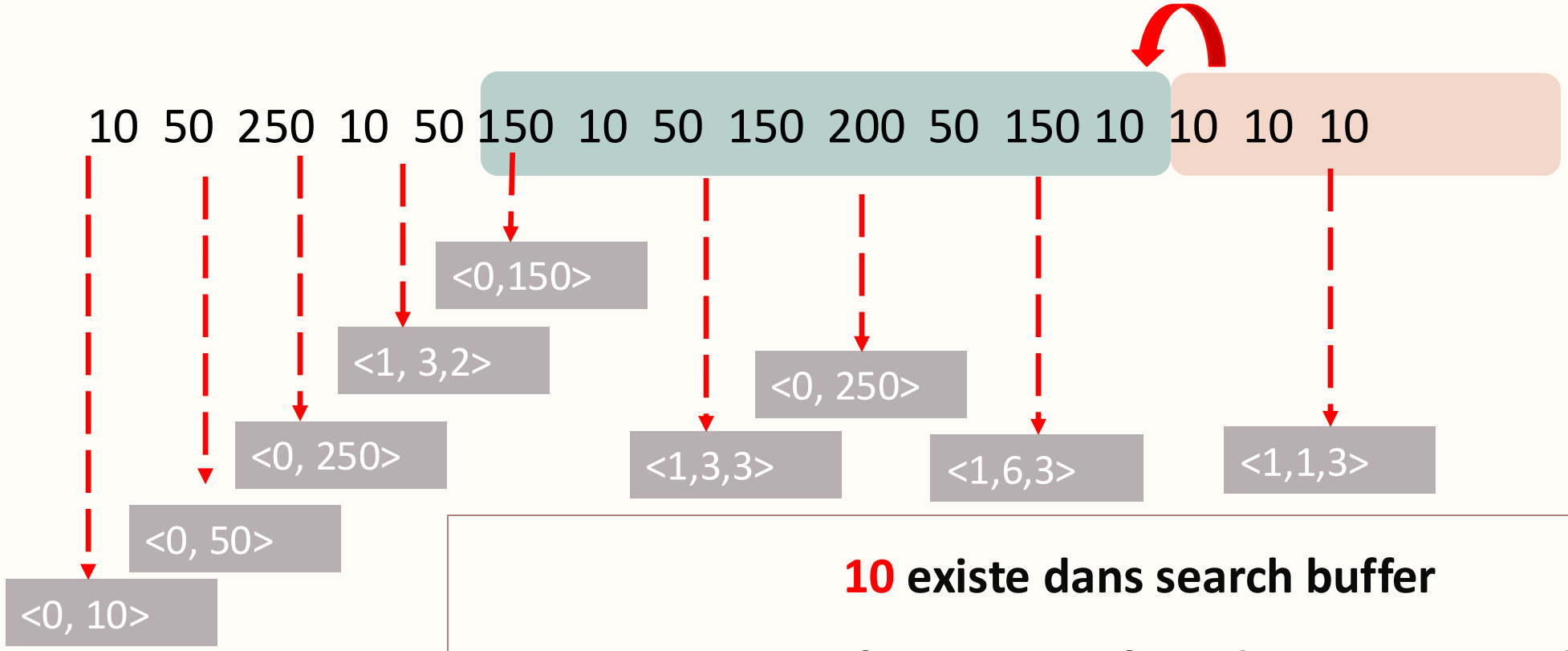
50 150 10 existe dans search buffer

revenir six étapes en arrière, choisir trois symboles

« flags=1, position=6, longueur=3»

Algorithme DEFLATE

- Exemple : LZSS



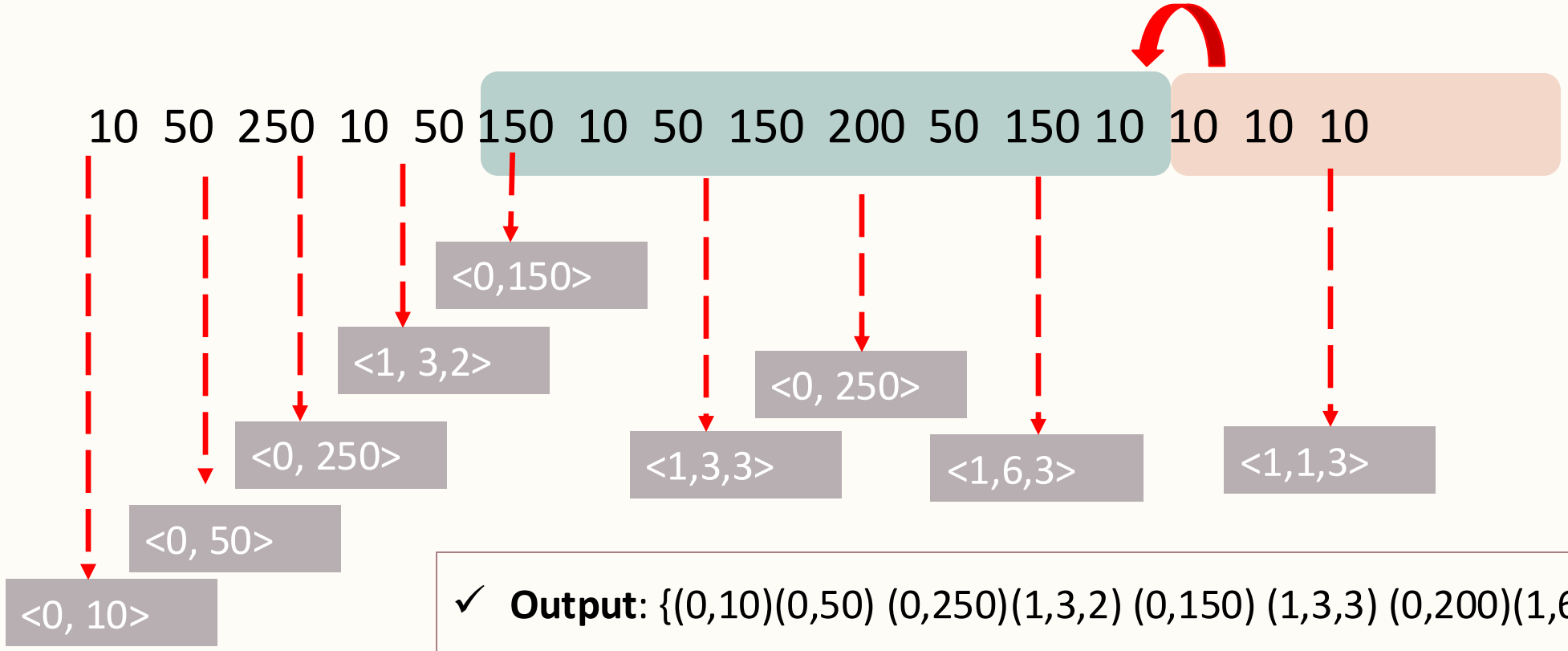
10 existe dans search buffer

revenir une étape en arrière, choisir trois symboles

« flags=1, position=1, longueur=3»

Algorithme DEFLATE

- **Exemple : LZSS**

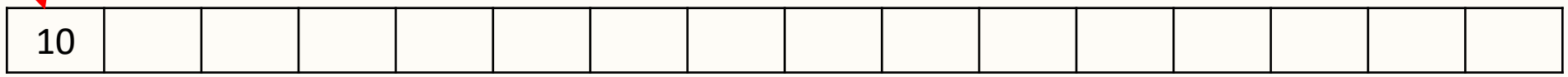


- ✓ **Output**: {(0,10)(0,50) (0,250)(1,3,2) (0,150) (1,3,3) (0,200)(1,6,3) (1,1,3)}
- ✓ L'image compressée dans LZSS: 5*8bits+(3+2bits)*4+9bits (flags 0 ou 1)=69
- ✓ **T=0.54, G=46%**

Algorithme DEFLATE

- **Exemple : LZSS Décompression**

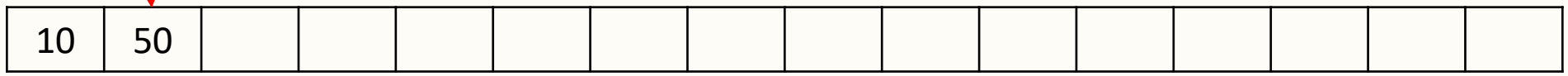
<0, 10> | <0, 50> | <0, 250> | <1, 3,2> | <0,150> | <1,3,3> | <0, 250> | <1,6,3> | <1,1,3>



Algorithme DEFLATE

- **Exemple : LZSS Décompression**

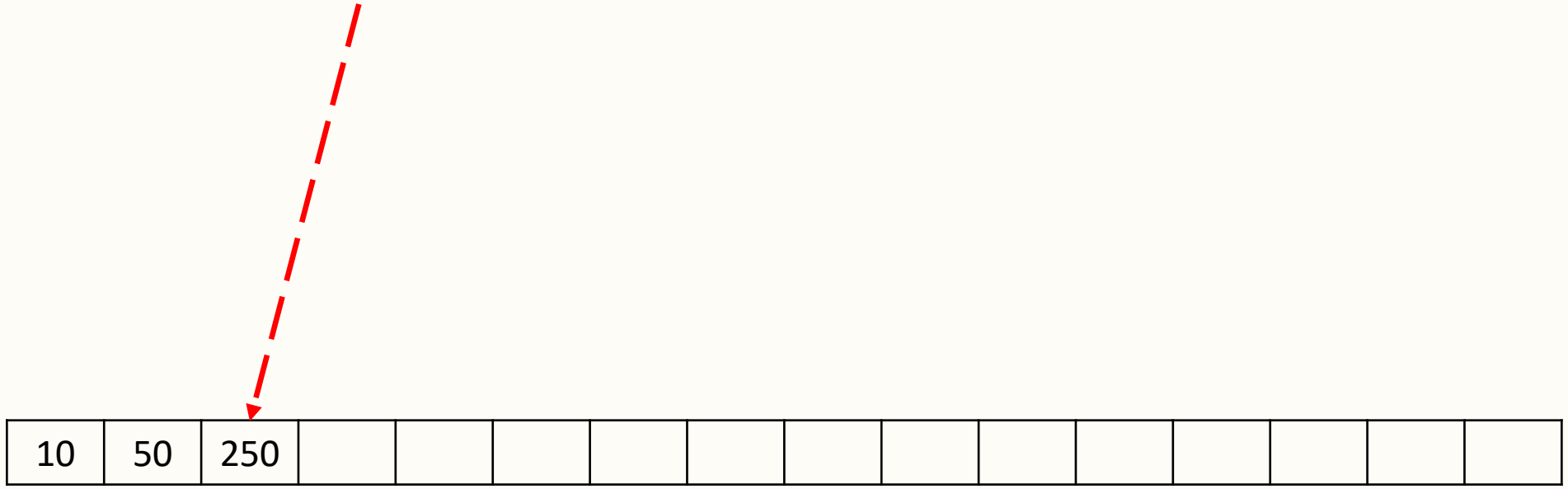
<0, 10> <0, 50> <0, 250> <1, 3,2> <0,150> <1,3,3> <0, 250> <1,6,3> <1,1,3>



Algorithme DEFLATE

- **Exemple : LZSS Décompression**

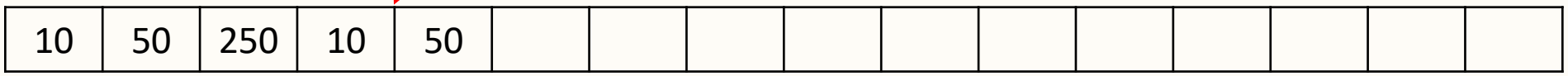
<0, 10> <0, 50> <0, 250> <1, 3,2> <0,150> <1,3,3> <0, 250> <1,6,3> <1,1,3>



Algorithme DEFLATE

- **Exemple : LZSS Décompression**

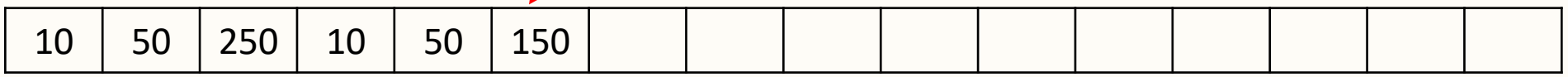
<0, 10> <0, 50> <0, 250> <1, 3,2> <0,150> <1,3,3> <0, 250> <1,6,3> <1,1,3>



Algorithme DEFLATE

- **Exemple : LZSS Décompression**

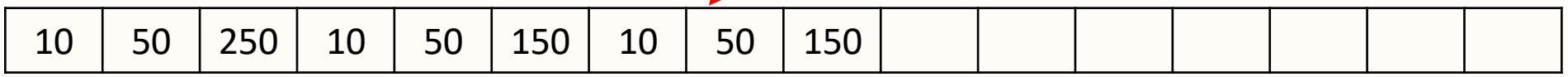
<0, 10> <0, 50> <0, 250> <1, 3,2> <0,150> <1,3,3> <0, 250> <1,6,3> <1,1,3>



Algorithme DEFLATE

- **Exemple : LZSS Décompression**

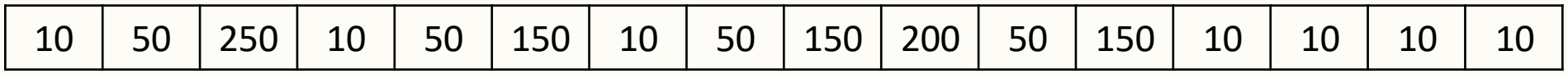
<0, 10> <0, 50> <0, 250> <1, 3,2> <0,150> <1,3,3> <0, 250> <1,6,3> <1,1,3>



Algorithme DEFLATE

- **Exemple : LZSS Décompression**

<0, 10> <0, 50> <0, 250> <1, 3,2> <0,150> <1,3,3> <0, 250> <1,6,3> <1,1,3>



Algorithme DEFLATE

- Exemple : Codage de Huffman

✓ Créer l'arbre de Huffman pour le littéral/longueur

✓ **Output : (10, 50, 250, 2, 150, 3, 200, 3, 3)**

3	2	10	50	150	200	250
3	1	1	1	1	1	1
00	010	011	100	101	110	111

✓ Créer l'arbre de Huffman pour la distance

✓ **Output : (3, 3, 6, 1)**

3	6	1
2	1	1
0	10	11

✓ **L'image compressée dans LZSS+Huffman:**

✓ **Taille = 30bits**

✓ **T=0.23, G=76%**

Avantages et Inconvénients

- **Avantages:**

✓ Bon équilibre entre temps d'exécution et taux de compression.

✓ utilisé dans PNG ZIP GZIP

- **Inconvénient:**

✓ Plus complexe .

✓ Peut conduire à une taille plus grande que les données d'origine pour les petits fichiers.