Indication pour les Algorithmes de compression:

Notes:

Tout les algorithmes sont lossless (pas de perte de données)
Chaque algorithme est implémenté dans Splitter.py

Informations:

Chaque donnée doit être compressée par rapport à ses pairs, et non par rapport aux données autour.

Exemple (voir sortie de SimuScintillateur.c):

Temps: Incrémentation continue → Delta compression
Type d'événement: Valeurs 0, 1 ou 2 → Normalized size
Énergie de la particule: Valeurs ayant une répartition aléatoire → les algorithmes peuvent varier selon la fréquence des valeurs (Huffmann, Unary, etc ...)
Etc ...

Les valeurs inutiles doivent être supprimées, exemple:

- Si le type d'événement est 2 (bruit de fond), la ligne correspondante peut être «ignorée».
- Si le statut de l'ADC indique 0 (sous le seuil de détection) ou 2 (saturation) il est inutile de compresser et d'envoyer la valeur lue de l'ADC car celle-ci sera respectivement «0» et «3fff»

Adaptive Huffman coding:

Cet algorithme calcule l'entropie de l'alphabet pour compresser les données.

Exemple:

Texte:

-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 2 2 2 -1 -1 -1

Fréquences:

-1 = 13
0 = 0
1 = 0
2 = 3

Alphabet:

-1 = 1
2 = 01
0 = 001
1 = 000

Texte compressé en binaire:

Alphabet + 1111111111010101111

Problèmes:

-Si la répartition de l'alphabet n'est pas «déséquilibré», ie: les fréquences de chaque lettres sont proches les unes des autres, cet algorithme à un rendu très mauvais.

Exemple:

Texte:

-1 0 1 2 -1 0 1 2

Fréquences:

-1 = 1

0 = 1

1 = 1

2 = 1

Alphabet:

-1 = 1

0 = 01

1 = 001

2 = 000

Texte compressé en binaire:

Alphabet + 101001000101001000

-Il faut encoder l'alphabet et l'envoyer pour pouvoir décompresser celui-ci plus tard, si l'alphabet est très grand cela peut nuire au taux de compression.

Unary compression:

Idem que Huffmann, mais l'alphabet n'est pas fait de la même façon:

Alphabet:

-1 = 1

2 = 01

0 = 001

1 = 0001

Texte compressé en binaire:

Alphabet + 1111111111010101111

Problèmes:

Texte compressé en binaire:

Alphabet + 10100100011010010001

Delta compression:

Cet algorithme est très efficace pour compresser des données qui s'incrémentent rapidement (comme le temps), car Il suffit de coder les différences entre chaque élément

au lieu de coder chaque élément.

Exemple:

Texte:

1 2 3 4 5 ... 1000000 ... 5 4 3 2 1

Texte compressé:

1 +1 +1 +1 +1 ... +1 ... -1 -1 -1 -1 -1

Problèmes:

La taille de chaque valeur encodée peux varier entre la longueur maximale (ici 1000000) et la valeur minimale (ici 1), il faut donc prendre cela en compte et envoyer d'une façon claire où couper les bits pour pouvoir récupérer les valeurs lors de la décompression, il faut aussi noter le signe (+ ou -).

Exemple:

1 10 100 1000 100 10 1

1 +9 +90 +900 -900 -90 -9

Premier bit:

0 = +

1 = -

En binaire:

01 01001 01011010 01110000100 11110000100 11011010 11001

Normalized size:

Chaque valeur est encodée sur un nombre défini de bits.

Exemple:

Texte:

-1 0 1 2 -1 0 1 2

Alphabet:

-1 = 00

0 = 01

1 = 10

2 = 11

Texte compressé en binaire:

Alphabet + 0001101100011011

Problèmes:

Si le texte a une disparité trop grande au niveau des valeurs, l'alphabet de compression sera très mauvais.

Exemple:

0 1 2 3 ... 1000000 ... 3 2 1 0

Alphabet:

0 = 00000000000000000000

1 = 00000000000000000001

2 = 00000000000000000010
3 = 00000000000000000011
...
1000000 = 11110100001001000000

Texte compressé en binaire:

00000000000000000000000000000000000000010000000000000000010
00000000000000000011 ... 11110100001001000000 ...
00000000000000000011000000000000000000010000000000000000010
00000000000000000000

Améliorations:

Il est possible de compresser une «seconde fois» les données si les valeurs sont très disparates.

Exemple avec normalized size:

00000000000000000000 = 20 zéro
00000000000000000001 = 19 zéro, 1 un
00000000000000000010 = 18 zéro, 1 un, 1 zéro
....

Ici il est possible de faire un mantisse du type:

11111 10100 0
11111 = Ceci est une valeur «spéciale» (valeur max 1000000 commence par 11110 ce qui serait déclaré lors de l'envoi)
10100 = Il y a 20 fois la valeur suivante
0 = La valeur est zéro

Ou encore:

11111 00001 01
11111 = Valeur «spéciale»
00001 = Puissance
01 = valeur

Etc ...

Problèmes:

La longueur des valeurs encodées peux devenir variable.
Il faudra aussi envoyer un alphabet.