Logo, company name

Description automatically generated

S3.01 - Développement d’application et Gestion de Projet

Enseignant tuteur : Nathalie VALLES-PARLANGEAU

Groupe n°1

**SAE du Troisième Semestre**

- Pôle Développement -

Spécifications Externes de notre problème algorithmique

Logo

Description automatically generated

Sport Track

Application Web Responsive de gestion de clubs sportifs amateurs

Matis Chabanat | Titouan Cocheril

Arthur Le Menn | Ivan Salle

- - -

TD1 | TP1

BUT Informatique

Semestre 3

- 2022 | 2023 -

**Piste de recherche :**

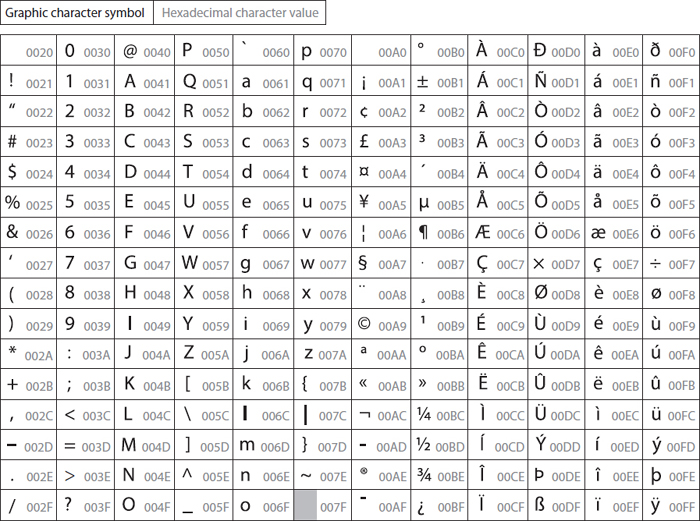
Nous cherchons à encoder une chaine de caractère en code barre (modèle Spotify)

On prend pour l’exemple en donnée le numéro de licence suivant : « VT0410532 »

On créé un page web avec dans un conteneur le logo de l’application ainsi que 30 barres dont on ne définit pas la hauteur. Notre algorithme sera chargé de définir la hauteur des barres en fonction de la chaine de caractère donnée.

On cherche une manière d’encoder

On choisit le système de conversion Unicode :



En appliquant l’algorithme on obtient :

Text

Description automatically generated

Algorithme graphique GitHub : SportTrack/Algorithme/algoUnicode

Code GitHub : SportTrack/src/convertionUnicode

Dans notre exemple le logo n’est pas utilisé comme ration il est juste présent à titre indicatif. La première barre prend la taille la plus petite possible.

Problème Unicode : il comprend beaucoup trop de caractère et n’est pas optimisé à notre exemple.

On va donc utiliser la base 36 qui comprend les 26 lettres de l'alphabet (a-z) et les 10 chiffres (0-9).

On créé notre propre table d’encodage.

Chaque caractère se verra attribuer deux barres la tables de cryptage défini la taille de ces barres pour chaque caractère.

Table d’encodage :

|  |  |
| --- | --- |
| a | 1,1 |
| b | 1,2 |
| c | 1,3 |
| d | 1,4 |
| e | 1,5 |
| f | 1,6 |
| g | 1,7 |
| h | 1,8 |
| i | 1,9 |
| j | 2,0 |
| k | 2,1 |
| l | 2,2 |
| m | 2,3 |
| n | 2,4 |
| o | 2,5 |
| p | 2,6 |
| q | 2,7 |
| r | 2,8 |
| s | 2,9 |
| t | 3,0 |

|  |  |
| --- | --- |
| u | 3,1 |
| v | 3,2 |
| w | 3,3 |
| x | 3,4 |
| y | 3,5 |
| z | 3,6 |
| 0 | 3,7 |
| 1 | 3,8 |
| 2 | 3,9 |
| 3 | 4,0 |
| 4 | 4,1 |
| 5 | 4,2 |
| 6 | 4,3 |
| 7 | 4,4 |
| 8 | 4,5 |
| 9 | 4,6 |

En appliquant l’algorithme graphique on obtient :

\*\*\*

<https://levelup.gitconnected.com/spotify-codes-and-how-they-work-664f4e4b8489>

<https://boonepeter.github.io/posts/2020-11-10-spotify-codes/>

Skimage

regionprops

Gestion et correction des erreurs

2 types de correction d’erreur:

Codes convolutionnels et blocs de codes linéaires

|  |  |
| --- | --- |
| Codes convolutionnels | Codes de blocs linéaires |
| exemples:  Code Turbo  Diagrammes Trellis | exemples:  codes de Hamming  codes cycliques |

Dans notre cas nous utiliserons un code convolutionnels car la longueur du mot encodé nous importe (pour réduire le nombre de barres)

GRAY CODE sur 6 bits (62 mots différents donc 2^6)

Gray code : comme le binaire mais 1 seul bit change entre chaque caractère suivant pour faciliter la correction des erreurs.

|  |  |
| --- | --- |
| a | 000000 |
| b | 000001 |
| c | 000011 |
| d | 000111 |
| e | 001111 |
| f | 011111 |
| g | 111111 |
| h | 111110 |
| i | 111100 |
| j | 111000 |
| k | 110000 |
| l | 100000 |
| m | 100001 |
| n | 100011 |
| o | 100111 |
| p | 101111 |
| q | 101110 |
| r | 101100 |
| s | 101000 |
| t | 001000 |

|  |  |
| --- | --- |
| u | 001100 |
| v | 011100 |
| w | 011110 |
| x | 010110 |
| y | 010100 |
| z | 010101 |
| 0 | 010101 |
| 1 | 011101 |
| 2 | 001101 |
| 3 | 001001 |
| 4 | 000001 |
| 5 | 101001 |
| 6 | 101011 |
| 7 | 111011 |
| 8 | 111010 |
| 9 | 101010 |

Schéma des étapes de conversion en « ST code »

Une image contenant texte

Description générée automatiquement