

BACKTRACKING



30 de abril de 2018

Gema rico pozas

UO238096

TRABAJO PEDIDO

Se tratará de optimizar al máximo los tiempos de ejecución de cada una de las condiciones y por tanto del algoritmo en general.

Es necesario calcular el tiempo del algoritmo propuesto para claves de 3 caracteres hasta el tamaño que invierta un tiempo que se considere demasiado prolongado.

*public void backtracking(int nivel) {*

*Random random = new Random(); //para seleccionar una letra de aux aleatoria*

*if (nivel == totalSize) //si es el último nivel*

*meta = true;*

*else {*

*Character [] aux;*

*//condición que indica si ponemos carácter o terminal*

*if (nivel < totalSize - totalSizeSimbolo) {*

*aux = copiar((Character[]) abcList.toArray());*

*}*

*else {*

*aux = copiar((Character[]) NoAbcList.toArray());*

*}*

*do {*

*//seleccionamos carácter aleatorio*

*char letra = aux[random.nextInt(aux.length)];*

*if (check(password, nivel, letra) == true) { //validamos las condiciones*

*password[nivel] = letra;*

*backtracking(nivel + 1); //llamamos con sig nivel*

*}*

*} while (meta != true);*

*}}*

Mediciones de tiempos:

|  |  |
| --- | --- |
| **Caracteres** | **tiempo en micros** |
| 3 | 34 |
| 6 | 42 |
| 12 | 64 |
| 24 | 119 |
| 48 | 221 |
| 96 | 438 |
| 192 | 890 |
| 384 | 1787 |
| 768 | 3596 |
| 1536 | 7233 |
| 3072 | 14443 |
| 6144 | 29131 |

La complejidad del algoritmo es lineal. El algoritmo no llega a ser exponencial puesto que antes se produce overflow de la pila.