UNIVERSIDAD PRIVADA BOLIVIANA DIRECCIÓN DE PREGRADO

FACULTAD DE INGENIERÍAS Y ARQUITECTURA

INVESTIGACIÓN SEGUNDO PARCIAL

Algoritmos Extra – Algorítmica II

JUAN CLAUDIO CARRASCO TAPIA

LA PAZ – BOLIVIA

2022

Algoritmo Edit Distance

Se tiene 3 operaciones realizables:

- Remove
- Replace
- Insert

El algoritmo determina las diferencias entre 2 cadenas, basándose en el proceso de cambio de cadena carácter por carácter que tome la menor cantidad de operaciones.

Comenzando desde la izquierda o la derecha de ambas cadenas, hay dos posibilidades por cada posición de caracteres que se revise

Teniendo 2 cadenas de largos m y n, puede que sea innecesario un cambio en alguna de las cadenas si ambos caracteres en la posición actual son iguales, como también puede que se realice un cambio que sea una de las 3 operaciones mencionadas. El proceso se repite a lo largo de las cadenas hasta definir el camino mas corto para tener 2 cadenas idénticas.

Código del algoritmo:

```
int editDistance(string a, string b, int m, int n){
    int dp[m + 1][n + 1];
    for (int i = 0; i <= m; i++) {
        for (int j = 0; j <= n; j++) {
            if (i == 0)
                dp[i][j] = j;
            else if (j == 0)
                dp[i][j] = i;
            else if (a[i - 1] == b[j - 1])
                dp[i][j] = dp[i - 1][j - 1];
             else
                dp[i][j] = 1 + min(dp[i][j - 1],min(dp[i - 1][j],dp[i - 1][j - 1]));
       }
    }
    return dp[m][n];
}
```

Complejidad O(n*m) → (for i to n) y dentro (for j to m)

Algoritmo de Floyd Warshall

Es útil para encontrar la ruta más corta entre dos nodos en un grafo ponderado, aún cuando este tiene bordes de valor negativo. Hace esto comparando todos los posibles caminos entre dos nodos. Aunque el algoritmo solo provee el peso total de los caminos entre nodos, se puede modificar para reconstruir el camino.
Código del algoritmo:

```
void floydWarshall() {
    for(int k = 0; k<V; k++)
        for(int i = 0; i<V; i++)
            for(int j = 0; j<V; j++)
                dist[i][j] = min(GRAFO[i][j], GRAFO[i][k] + GRAFO[k][j]);
```

Complejidad O(V^3) → donde V es la cantidad de vértices en el grafo

Algoritmo Rod Cutting

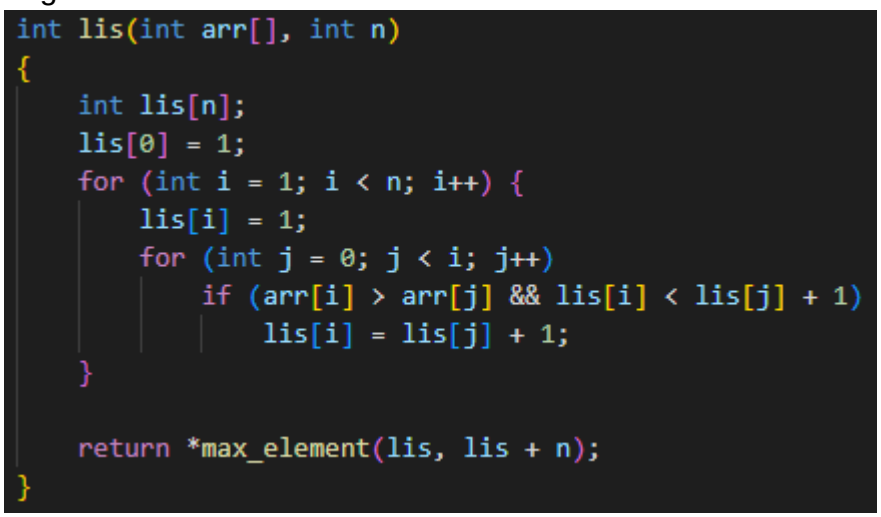
Dada una barra de longitud n y una tabla con los precios de todas sus piezas de longitud menor, el algoritmo determina la ganancia máxima que se puede obtener cortandola y vendiendo las piezas.
Para evitar tener que generar todas las posibles formas de cortar la barra, se usa la programación dinámica, armando un arreglo parecido al que se usa para el problema de la mochila (knappsack).
Código del algoritmo:

```
int cutRod(int price[], int n)
{
   int val[n+1];
   val[0] = 0;
   int i, j;
   for (i = 1; i<=n; i++)
   {
       int max_val = INT_MIN;
       for (j = 0; j < i; j++)
         max_val = max(max_val, price[j] + val[i-j-1]);
       val[i] = max_val;
   }
   return val[n];
}
```

Complejidad O(n^2) → (for I to n) y (for j to i)

Algoritmo Longest Increasing Subsequence

Sirve para determinar la subsecuencia mas larga posible desde un arreglo inicial sin cambiar el orden de sus miembros, si no solo usando elementos específicos del arreglo.
Por ejemplo:
[10,9,2,5,3,7,101,18]
LIS:
[2,3,7,101]
Largo LIS es 4
Con un arreglo auxiliar se puede tener guardada la subsecuencia en si, no solo su largo.



```
int lis(int arr[], int n)
{
    int lis[n];
    lis[0] = 1;
    for (int i = 1; i < n; i++) {
        lis[i] = 1;
        for (int j = 0; j < i; j++)
            if (arr[i] > arr[j] && lis[i] < lis[j] + 1)
                lis[i] = lis[j] + 1;
    }

    return *max_element(lis, lis + n);
}
```

Complejidad O(n^2) → (for I to n) y (for j to i)