Análisis del algoritmo de ordenamiento de Timsort

Samy Felipe Cuestas Merchan, Camilo Hernández Guerrero, Juan Camilo Mendieta Hernández

17 de febrero de 2022

Resumen

En este documento se presenta el análisis del ordenamiento de TimSort. Los elementos a presentar son análisis, diseño y algoritmo de Timsort.

1. Parte I: Análisis y diseño del algoritmo

1.1. Análisis

El problema parte de la necesidad de ordenar una secuencia dada mediante el uso del algoritmo de ordenamiento conocido como Timsort, dicha secuencia está dada por

$$S = \langle s_1, s_2, \cdots, s_n \rangle = \langle s_i \in \mathbb{Z} \wedge 1 \leq i \leq n \rangle$$

donde S es la secuencia entrante, n es el tamaño de la secuencia, i es el índice de cada elemento y $\mathbb{Z}$ es el conjunto de los números enteros.

1.2. Diseño

1.2.1. Entradas

Como entradas tenemos la secuencia $S = \langle s_i \in \mathbb{T} \wedge 1 \leq i \leq n \rangle$ de $n \in \mathbb{N}$ elementos que pertenecen a un conjunto $\mathbb{T}$.

1.2.2. Salidas

Como salidas tenemos la secuencia $S = \langle s_i \in \mathbb{T} \wedge 1 \leq i \leq n \rangle$ pero con sus elementos cumpliendo la relacion de orden parcial $\leq$ para cada elemento adyacente.

2. Parte II: Algoritmos

2.1. Timsort

Algorithm 1 Ordenamiento TimSort

```
procedure TimSort(S)
    minRun ← calcMinRun(|S|)
    for i ← 0 to |S| of tam minRun do
        end ← min(i + minRun - 1, n - 1)
        InsertionSort(S, i, end)
    end for
    size ← minRun
    while size < n do
        for left ← 0 to |S| of tam 2 * size do
            mid ←min(n - 1, left + size - 1)
            right ←min(left + 2 * size - 1, n - 1)
            if mid < right then
                merge(S, left, mid, right)
            end if
        end for
        size ← 2 * size
    end while
end procedure
```

2.2. Calculo del run minimo

Algorithm 2 calcMinRun

```
procedure calcMinRun(|S|)
    r ← 0
    while n >= 32 do
        r| ← n & 1
        r>> ← 1
    end while
    return n + r
end procedure
```

2.3. Mezcla de subsecuencias

Algorithm 3 Función merge

```
procedure MERGE(S, l, m, r)
    len1 ← m − l + 1
    len2 ← r − m
    for i ← 0 to len1 do
        left append S[l + i]
    end for
    for i ← 0 to len2 do
        right append S[m + 1 + i]
    end for
    i ← 0
    j ← 0
    k ← l
    while i < len1 and j < len2 do
        if left[i] ≤ right[j] then
            S[k] ← left[i]
            i ← i + 1
        else
            S[k] ← right[j]
            j ← j + 1
        end if
        k ← k + 1
    end while
    while i < len1 do
        S[k] ← left[i]
        k ← k + 1
        i ← i + 1
    end while
    while j < len2 do
        S[k] = right[j]
        k ← k + 1
        j ← j + 1
    end while
end procedure
```

2.4. Ordenamiento por inserción

Algorithm 4 Ordenamiento por inserción

```
procedure INSERTIONSORT(S, left, right)
    for i ← left + 1 to right + 1 do
        j ← i
        while j > left and S[j] < S[j − 1] do
            S[j] ← S[j − 1]
            S[j − 1] ← S[j]
            j ← j − 1
        end while
    end for
end procedure
```

2.5. Complejidad

Por investigación se encontro que el algoritmo TimSort tiene orden de complejidad $O\left(|S|\,log(|S|)\right)$ en su caso promedio y en su peor caso, mientras que en su mejor caso tiene orden de complejidad

$\Omega\ (\ n)$. Esto gracias a que aprovecha las secciones d ela secuencias que ya están ordenadas para copiarlas directamente en una sebsecuencia sin tener que ordenarlas de nuevo.

2.6. Invariante

- **TimSort**Por cada iteración cada subsecuencia cuyo tamaño se determina por el tamaño de la minima subsecuencia se encuentra organizada, luego se mezcla dando como resultado una secuencia que cumple con la relación de orden parcial $<=$.

  Inicio: La secuencia S esta dividida en i subsecuencias las cuales pueden estar ordenadas y cuyo tamaño es determinado por minrun

  Avance: En la i-esima permutación la i-esima subsecuencia está ordenada mientras que en cada iteración del while se mezclan estas subsecuencias

  Terminación: S' es una secuencia ordenada

- **Merge** Las banderas de control $len1$ y $len2$ indican si las variables i y j siguen la relación de orden parcial $<$.

  Inicio: Se inicia con el resultado de CALCMINRUN, dicha función determina la mínima longitud de un run entre 32 y 64 para que el tamaño del arreglo sea menor o igual a una potencia de 2.

  Avance: Se valida que la variable $size$ sea menor a n, de ser así, se elije un punto en el subarreglo para después fusionar el arreglo a su izquierda, el siguiente a este y después se multiplica en 2.

  Terminación: La variable $size$ es mayor que el tamaño del arreglo.

- **InsertionSort**Por cada iteración los primeros i elementos están ordenados

  Inicio: El elemento i está ordenado.

  Avance: Por cada iteración los elementos desde el inicio de arreglo hasta i están ordenados.

  Terminación: S' es una secuencia ordenada.

Referencias

[1] Kumar,A.(26 de junio de 2021) GeeksforGeeks.Recuperado de https://www.geeksforgeeks.org/timsort/.

[2] Thomas, H., Charles, E., Ronald, L., Clifford, S.(2009).Introduction to Algorithms.Massachusetts,USA: The MIT Press

[3] Hoque,T.(4 de junio de 2019) Youtube.Recuperado de https://www.youtube.com/watch?v=_dlzWEJoU7I.