Selección de actividades

Jessica Tatiana Naizaque Guevara[1]

[1]Departamento de Ingeniería de Sistemas, Pontificia Universidad Javeriana
Bogotá, Colombia
{j.naizaque}@javeriana.edu.co

6 de octubre de 2022

Resumen

En este documento se presenta la comparación de los algoritmos implementados en python para la solución del problema de selección de actividades.

Índice

1. Introducción

La selección de actividades es un problema bastante recurrente que debe ser solucionado por un algoritmo que permita maximizar la cantidad de actividades en un rango de horas establecido. Por lo tanto, en este documento se presenta la comparación de las implementaciones de programación dinámica y voraz de las soluciones presentadas para dicho problema. Este informe se divide en varias secciones: descripción de los algoritmos 2, análisis de complejidad de los algoritmos 3, análisis de calidad de las soluciones 4 y experimentación 5.

2. Descripción de los algoritmos

2.1. Algoritmo de programación dinámica

La idea de este algoritmo es inicialmente, organizar las actividades teniendo en cuenta su hora de finalización. Luego, recorrer estas actividades y almacenarlas en una nueva estructura, de modo que se encuentren aquellas donde se cumple que la hora de fin de la primera sea anterior a la hora de inicio de la segunda y la hora de finalización de esta última sea anterior al fin de la última actividad. Luego de tener esta estructura creada, se recorre iterativamente con un índice x y se divide el problema, comenzando con la primera actividad hasta la que lleva el índice y desde la que lleva el índice hasta la última actividad. Al final, sumando ambos valores de la división del problema, se debe evaluar con una nueva variable para ir guardando el máximo valor que es aquel que maximiza la cantidad de actividades en el rango de horas y es el que se guardará en la matriz de memoización. Esta última es de dos dimensiones y almacena los valores máximos encontrados en cada iteración, al final debe contener la respuesta (máximo) en la primera fila y la última columna.

2.2. Algoritmo voraz

La idea de este algoritmo es ordenar inicialmente las actividades teniendo en cuenta su hora de finalización. Luego de esto, se recorren iterativamente y teniendo en cuenta la comparación entre la actividad actual y la siguiente que me cumpla la condición de comenzar después de que la actual termine, se irán almacenando en una estructura aparte

3. Análisis de complejidad

3.1. Algoritmo de programación dinámica

Por inspección de código, es posible encontrar que inicialmente para el llenado de la tabla de memoización, se tiene una complejidad de $O(n^2)$. Sin embargo, a esta se le añade la complejidad que tiene el recorrer el arreglo que guarda las actividades como tal, que tendría complejidad de $O(n)$.

3.2. Algoritmo voraz

Por inspección de código, es posible observar que no requiere más de un ciclo para resolver el problema, lo cual quiere decir que tiene una complejidad de $O(n)$.

4. Análisis de calidad

4.1. Algoritmo de programación dinámica

Aunque es bueno en ocasiones que se calculen todas las posibilidades para conocer la más óptima, muchas veces se requiere una que sea solamente aceptable y no del todo óptima. Este algoritmo permite hallar todos los caminos posibles hasta el objetivo planteado, por lo que se puede decir que tiene una alta calidad, no obstante, hay un factor que afecta bastante en la calidad y es la cuestión del tiempo, pues es muy estricto en su orden y sus pasos. Adicional a esto, el algoritmo pierde ventaja al hacer uso de infinito o menos infinito, pues después de cierta cantidad de datos deja de responder correctamente al problema planteado.

4.2. Algoritmo voraz

Es posible decir que este algoritmo tiene una calidad media a baja, esto se debe que aunque regresa un resultado lo más pronto posible, este es aceptable y no del todo óptimo. Por lo cual, podrían estarse perdiendo datos importantes a tener en cuenta en las soluciones presentadas.

5. Experimentación

Para este caso, se tomó un archivo .txt que se lee desde cada una de las implementaciones, este contiene inicialmente 10 actividades, luego 25, después 50 y, por último, 100. A continuación, es posible observar los resultados que obtuvo cada algoritmo en estos casos.

5.1. Algoritmo programación dinámica

- 10 actividades:

```
5
Tiempo algoritmo programación dinámica: 0.0003356439992785454 segundos
```

Figura 1: Resultado para 10 actividades en el algoritmo de programación dinámica

- 25 actividades:

```
8
Tiempo algoritmo programación dinámica: 0.0009142049821093678 segundos
```

Figura 2: Resultado para 25 actividades en el algoritmo de programación dinámica

- 50 actividades:

```
    if M[i][j] == -math.inf:
RecursionError: maximum recursion depth exceeded in comparison
```

Figura 3: Resultado para 50 actividades en el algoritmo de programación dinámica

- 100 actividades:

```
    if M[i][j] == -math.inf:
RecursionError: maximum recursion depth exceeded in comparison
```

Figura 4: Resultado para 100 actividades en el algoritmo de programación dinámica

5.2. Algoritmo voraz

- 10 actividades:

```
5
Tiempo algoritmo voraz: 0.00016213301569223404 segundos
```

Figura 5: Resultado para 10 actividades en el algoritmo voraz

- 25 actividades:

```
8
Tiempo algoritmo voraz: 0.0011212798999622464 segundos
```

Figura 6: Resultado para 25 actividades en el algoritmo voraz

- 50 actividades:

```
12
Tiempo algoritmo voraz: 0.0001780620077624917 segundos
```

Figura 7: Resultado para 50 actividades en el algoritmo voraz

- 100 actividades:

Figura 8: Resultado para 100 actividades en el algoritmo voraz