



Universidad Autónoma de Coahuila
Facultad de Sistemas

ALGORITMOS DE ORDENAMIENTO Y BUSQUEDA

Proyecto

“Problema de selección de actividades”

Alumno

Miguel Ángel Machorro García

Profesor

CHIO BENAVIDES SANTIAGO

Introducción.

El problema de selección de actividades es un problema clásico en la teoría de algoritmos y se refiere a la tarea de seleccionar el conjunto máximo de actividades no se superpuestas entre sí, dadas las restricciones de tiempo asociadas con cada actividad.

Cada actividad esta definida por un tiempo de inicio y un tiempo de finalización. El objetivo es determinar cual es el conjunto mas grande de actividades que puede realizar sin que se superpongan entre sí. En otras palabras, dado un conjunto de actividades, deseas seleccionar el máximo numero de ellas de manera que ninguna se solape en el tiempo.

Formalmente, si denotamos las actividades como pares ordenados (s_i, f_i) , donde s_i es el tiempo de inicio y f_i es el tiempo de finalización de la actividad i , el problema se plantea de la siguiente manera:

Dado un conjunto de n actividades, seleccionar el conjunto mas grande de actividades no superpuestas.

El problema de selección de actividades tiene aplicaciones en diversos campos, como la programación de tareas, la planeación de eventos, la gestión de tiempo y la optimización de recursos. La solución optima a este problema es un conjunto de actividades no superpuestas con la mayor cantidad posible.

Existen diversos enfoques algorítmicos para abordar este problema, pero los dos mas comunes y los que mejor eficacia tienen son:

- Enfoque Voraz (Greedy):
Selecciona actividades en orden creciente según sus tiempos de finalización. En cada paso, elige la actividad que finaliza primero y que no se superpone con las actividades ya seleccionadas.
- Programación Dinámica:
Utiliza la programación dinámica para construir una tabla que almacena la longitud máxima de la subsecuencia optima para cada subproblema. Luego se utiliza esta tabla para reconstruir la solución óptima.

Ambos enfoques proporcionan soluciones eficientes al problema, y la selección entre ellos puede depender de la naturaleza específica del conjunto de datos y los requisitos del problema.

Ventajas y Desventajas de ambas soluciones.

Enfoque voraz (Greedy):

- Ventajas
 - Es eficiente y fácil de implementar.
 - Tiende a proporcionar soluciones rápidas y prácticas.
 - No requiere una fase de preprocesamiento extensa.
- Desventajas
 - No siempre garantiza la solución óptima en todos los casos, aunque en el caso del problema de selección de actividades, el enfoque voraz es conocido por proporcionar la solución óptima.

Programación Dinámica:

- Ventajas
 - Garantiza la solución óptima debido a su naturaleza exhaustiva.
 - Puede manejar casos mas complejos con restricciones adicionales.
- Desventajas
 - Requiere mas tiempo y espacio de ejecución en comparación con el enfoque voraz, especialmente para conjuntos de datos grandes.
 - La implementación puede ser mas compleja.

Elección entre ellos:

- Para conjuntos de datos pequeños o casos mas simples, el enfoque voraz puede ser preferido debido a su simplicidad y eficiencia.
- Para conjuntos de datos más grandes o problemas con restricciones adicionales, la programación dinámica es una elección sólida, ya que garantiza la solución óptima y es capaz de manejar casos mas complejos.

Solución Enfoque Voraz (Greedy).

Aquí tenemos una de las soluciones al problema de selección de actividades aquí utilizamos una función tanto greedy como recursiva para explorar todas las posibles combinaciones de actividades no superpuestas.

La complejidad del programa depende de como se seleccionan las actividades y cuantas actividades hay en total.

En el mejor de los casos ocurre cuando se seleccionan las actividades en orden sin necesidad de hacer muchas comparaciones y en este caso la complejidad seria lineal $O(n)$.

En el peor de los casos ocurre cuando la función tiene que explorar todas las combinaciones posibles de actividades antes de encontrar la solución, la recursión puede llegar a explorar todas las permutaciones posibles de actividades. La complejidad en el peor de los casos es $O(2^n)$ n representa el numero de actividades.

```
ActivitySelectionProblem.py U X
Practicas > ActivitySelectionProblem.py > ...
1 def max_activities(a_start, a_end, last_selected, n):
2     next_activities = last_selected + 1
3
4     while next_activities < n and a_start[next_activities] < a_end[last_selected]:
5         next_activities = next_activities + 1
6
7     if next_activities < n:
8         return [next_activities] + max_activities(a_start, a_end, next_activities, n)
9
10    return []
11
12    start_time = [2,4,6,1,6,8,3,1,0,3]
13
14    end_time = [2,1,5,6,4,2,6,8,6,2]
15
16    print(max_activities(start_time, end_time, 0, (len(start_time))))

exe c:/Users/Miike/Desktop/Algoritmos/AlgoritmosSistemas/ago-dic-2023/MiguelMachorro/Practicas/Practicas/ActivitySelectionProblem
.py
[1, 2, 4, 5, 6]
```

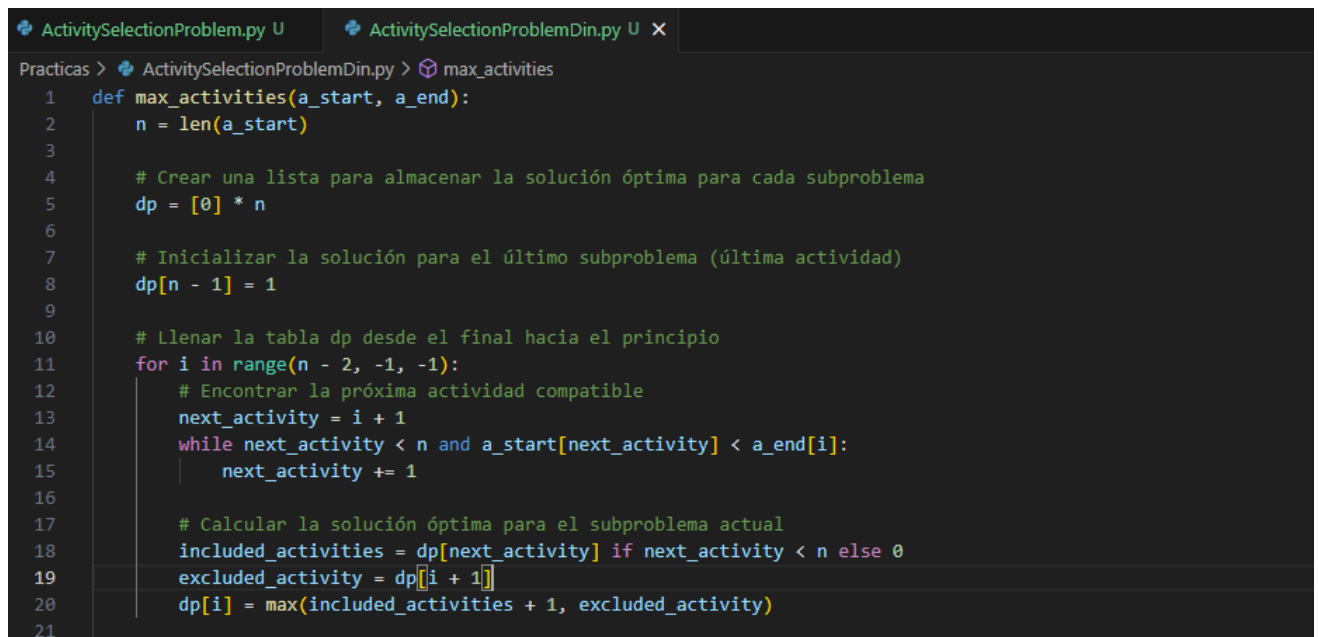
Solución con Programación Dinámica.

Aquí tenemos la segunda solución que se realizó con programación dinámica donde es algo parecido a la greedy lo que hace es que nos pide de igual manera una lista de tiempos de inicio y unas de tiempo de finalización.

En este problema se implementa una lista que en esta ocasión llamamos dp, esta se utiliza para almacenar la solución óptima para cada subproblema. Se llena de abajo hacia arriba, comenzando desde el ultimo subproblema (ultima actividad). Después, se reconstruye la solución óptima utilizando la información almacenada en nuestra lista dp.

La complejidad en el peor de los casos sería $O(n^2)$ ya que la búsqueda de la próxima actividad compatible en el bucle interno lleva tiempo lineal en el número total de actividades. Dado que este bucle interno se ejecuta en cada iteración del bucle externo que también tiene una complejidad lineal, por lo tanto, la complejidad es cuadrática.

En el mejor de los casos sería $O(n \log n)$ ya que ocurre cuando la búsqueda de la próxima actividad compatible se realiza de manera eficiente, en este caso el bucle interno tendría una complejidad de $\log n$ en lugar de n , el bucle externo sigue teniendo complejidad de n por lo tanto, la complejidad total en el mejor de los casos sería multiplicando ambas.



```
ActivitySelectionProblem.py U  ActivitySelectionProblemDin.py U X
Practicas > ActivitySelectionProblemDin.py > max_activities
1  def max_activities(a_start, a_end):
2      n = len(a_start)
3
4      # Crear una lista para almacenar la solución óptima para cada subproblema
5      dp = [0] * n
6
7      # Inicializar la solución para el último subproblema (última actividad)
8      dp[n - 1] = 1
9
10     # Llenar la tabla dp desde el final hacia el principio
11     for i in range(n - 2, -1, -1):
12         # Encontrar la próxima actividad compatible
13         next_activity = i + 1
14         while next_activity < n and a_start[next_activity] < a_end[i]:
15             next_activity += 1
16
17         # Calcular la solución óptima para el subproblema actual
18         included_activities = dp[next_activity] if next_activity < n else 0
19         excluded_activity = dp[i + 1]
20         dp[i] = max(included_activities + 1, excluded_activity)
21
```

```

21
22     # Reconstruir la solución óptima
23     result = []
24     i = 0
25     while i < n:
26         next_activity = i + 1
27         while next_activity < n and a_start[next_activity] < a_end[i]:
28             next_activity += 1
29
30         if next_activity < n and dp[i] == dp[next_activity] + 1:
31             result.append(next_activity)
32             i = next_activity
33         else:
34             i += 1
35
36     return result
37
38     # Datos de inicio y finalización de las actividades
39     start_time = [2, 4, 6, 1, 6, 8, 3, 1, 0, 3]
40     end_time = [2, 1, 5, 6, 4, 2, 6, 8, 6, 2]
41
42     result = max_activities(start_time, end_time)
43     print(result)
44

```

```

exe c:/Users/Miike/Desktop/Algoritmos/AlgoritmosSistemas/ago-dic-2023/MiguelMachorro/Practicas/Practicas/ActivitySelectionProblem
Din.py
[1, 2, 4, 5, 6]

```