




Reconocimiento de problemas Greedy y diseño de soluciones





Fernando Nicolás Frassia Ferrari Leandro Javier Raffo

2do Cuatrimestre 2024, TN

1 Cómo reconocer problemas Greedy?

-  Repaso
-  Reconociendo problemas Greedy
-  PD vs Greedy

2 Resolvamos problemas Greedy

-  Suma selectiva
-  Explotación capitalista
-  Auto a mardel
-  Explotación capitalista (con ganancias)



Repaso

Qué técnicas vimos hasta ahora para resolver problemas de optimización?



Repaso

Qué técnicas vimos hasta ahora para resolver problemas de optimización?

1 Backtracking



Repaso

Qué técnicas vimos hasta ahora para resolver problemas de optimización?

- 1 Backtracking
- 2 Programación dinámica



Repaso

Qué técnicas vimos hasta ahora para resolver problemas de optimización?

- 1 Backtracking
- 2 Programación dinámica
- 3 Greedy



Repaso

Qué técnicas vimos hasta ahora para resolver problemas de optimización?

- 1 Backtracking
- 2 Programación dinámica
- 3 Greedy

Cuál es la idea principal de un algoritmo Greedy?



Repaso

Qué técnicas vimos hasta ahora para resolver problemas de optimización?

- 1 Backtracking
- 2 Programación dinámica
- 3 Greedy

Cuál es la idea principal de un algoritmo Greedy?

En cada paso elegir un óptimo **local** con la esperanza de que al terminar lleguemos a una solución óptima **global**.



Propiedades necesarias para que un problema admita una solución Greedy



Propiedades necesarias para que un problema admita una solución Greedy

- 1 Elección Greedy (greedy choice): La elección greedy que se toma en cada paso es parte de una solución óptima final.



Propiedades necesarias para que un problema admita una solución Greedy

- 1 Elección Greedy (greedy choice): La elección greedy que se toma en cada paso es parte de una solución óptima final.
- 2 Subestructura óptima: Un problema tiene subestructura óptima si una solución óptima al problema contiene dentro soluciones óptimas a sus subproblemas.



PD vs Greedy

1 En qué se parecen?



PD vs Greedy

- 1 En qué se parecen?
 - Ambos necesitan subestructura óptima.



PD vs Greedy

- 1 En qué se parecen?
 - Ambos necesitan subestructura óptima.
 - Según el Cormen: "...beneath every greedy algorithm, there is almost always a more cumbersome dynamic-programming solution."
- 2 En qué se diferencian?



PD vs Greedy

1 En qué se parecen?

- Ambos necesitan subestructura óptima.
- Según el Cormen: "...beneath every greedy algorithm, there is almost always a more cumbersome dynamic-programming solution."

2 En qué se diferencian?

- Orden de decisión: PD primero resuelve los subproblemas y en base a esas soluciones toma la decisión actual. Greedy es al revés, primero toma la decisión actual y luego resuelve los subproblemas.



Vamos con los problemas?



Figure: Siiii

1 2
3 4

Suma selectiva

Problema 1

Dado un conjunto de enteros positivos X con $|X| = n$ y un entero $0 \leq k \leq n$, queremos encontrar el máximo valor que pueden sumar los elementos de un subconjunto S de X de tamaño k . Más formalmente, queremos calcular

$$\max_{S \subseteq X, |S|=k} \sum_{s \in S} s$$

Proponer un algoritmo greedy que resuelva el problema y demostrar su correctitud.



Antes de hacerlo con greedy...

Cómo lo podríamos resolver usando programación dinámica?



Antes de hacerlo con greedy...

Cómo lo podríamos resolver usando programación dinámica?
Cuál sería la función de recurrencia?

Antes de hacerlo con greedy...

Cómo lo podríamos resolver usando programación dinámica?

Cuál sería la función de recurrencia?

$\text{maxSuma}(A, k) = \text{ms}(A, k, 0)$

Antes de hacerlo con greedy...

Cómo lo podríamos resolver usando programación dinámica?

Cuál sería la función de recurrencia?

$\text{maxSuma}(A, k) = \text{ms}(A, k, 0)$

$\text{ms}(A, k, i) =$

$$\begin{cases} 0 & \text{si } k = 0 \\ A[i] + \text{ms}(A, k - 1, i + 1) & \text{si } k > 0 \wedge i + k = n \\ \max(A[i] + \text{ms}(A, k - 1, i + 1), \\ \text{ms}(A, k, i + 1)) & \text{si } k > 0 \wedge i + k < n \end{cases}$$

Antes de hacerlo con greedy...

Cómo lo podríamos resolver usando programación dinámica?

Cuál sería la función de recurrencia?

$\text{maxSuma}(A, k) = \text{ms}(A, k, 0)$

$\text{ms}(A, k, i) =$

$$\begin{cases} 0 & \text{si } k = 0 \\ A[i] + \text{ms}(A, k - 1, i + 1) & \text{si } k > 0 \wedge i + k = n \\ \max(A[i] + \text{ms}(A, k - 1, i + 1), \\ \text{ms}(A, k, i + 1)) & \text{si } k > 0 \wedge i + k < n \end{cases}$$

Complejidad:

Antes de hacerlo con greedy...

Cómo lo podríamos resolver usando programación dinámica?

Cuál sería la función de recurrencia?

$\text{maxSuma}(A, k) = \text{ms}(A, k, 0)$

$\text{ms}(A, k, i) =$

$$\begin{cases} 0 & \text{si } k = 0 \\ A[i] + \text{ms}(A, k - 1, i + 1) & \text{si } k > 0 \wedge i + k = n \\ \max(A[i] + \text{ms}(A, k - 1, i + 1), \\ \text{ms}(A, k, i + 1)) & \text{si } k > 0 \wedge i + k < n \end{cases}$$

Complejidad: $\mathcal{O}(nk)$ (con memo)



Ahora sí, cómo sería con Greedy?

Qué les dice la intuición?



Ahora sí, cómo sería con Greedy?

Qué les dice la intuición? Tomemos el máximo cada vez.



Ahora sí, cómo sería con Greedy?

Qué les dice la intuición? Tomemos el máximo cada vez.
Primero, esta idea cumple las dos propiedades que vimos?



Ahora sí, cómo sería con Greedy?

Qué les dice la intuición? Tomemos el máximo cada vez.
Primero, esta idea cumple las dos propiedades que vimos?

- Greedy choice:

Ahora sí, cómo sería con Greedy?

Qué les dice la intuición? Tomemos el máximo cada vez.
Primero, esta idea cumple las dos propiedades que vimos?

- Greedy choice: Sí (demo en la siguiente slide)
- Subestructura óptima:

Ahora sí, cómo sería con Greedy?

Qué les dice la intuición? Tomemos el máximo cada vez.
Primero, esta idea cumple las dos propiedades que vimos?

- Greedy choice: Sí (demo en la siguiente slide)
- Subestructura óptima: Sí (se los dejo de tarea).

Ahora sí, cómo sería con Greedy?

Qué les dice la intuición? Tomemos el máximo cada vez.
Primero, esta idea cumple las dos propiedades que vimos?

- Greedy choice: Sí (demo en la siguiente slide)
- Subestructura óptima: Sí (se los dejo de tarea).

Ahora que sabemos que sale con Greedy, hagámoslo.

Demostración de Greedy Choice

Teorema: Sea un subproblema S_i y sea e un número máximo de ese subproblema, entonces e pertenece a alguna solución óptima de S_i .

Demostración de Greedy Choice

Teorema: Sea un subproblema S_j y sea e un número máximo de ese subproblema, entonces e pertenece a alguna solución óptima de S_j .

Demo: Sea S_j un subproblema y sea e un número máximo de S_j .

Sea A_j una solución óptima para S_j y sea e' un número máximo de A_k . Si e es e' terminé, porque e pertenece a A_j y A_j es una solución óptima.

Por otro lado, si e no es e' , puedo armar la solución $A'_j = A_j - e' + e$ y será óptima porque tanto e como e' son máximos de S_j , por ende $e = e'$ y $A'_j = A_j$. Entonces e pertenece a alguna solución óptima de S_j .



Pseudocódigo

Pseudocódigo

Algorithm 2 $\text{maxSuma}(X: [\text{Int}], k: \text{Int}) \rightarrow \text{Int}$

1: $X.\text{sortAscendente}()$	$\triangleright \mathcal{O}(n \log n)$
2: $val \leftarrow 0$	$\triangleright \mathcal{O}(1)$
3: while $k > 0$ do	$\triangleright \mathcal{O}(k)$
4: $val \leftarrow val + X[X.\text{length}() - k]$	$\triangleright \mathcal{O}(1)$
5: $k --$	$\triangleright \mathcal{O}(1)$
6: end while	
7: return val	$\triangleright \mathcal{O}(1)$

Complejidad:

Pseudocódigo

Algorithm 3 $\text{maxSuma}(X: [\text{Int}], k: \text{Int}) \rightarrow \text{Int}$

1: $X.\text{sortAscendente}()$	$\triangleright \mathcal{O}(n \log n)$
2: $val \leftarrow 0$	$\triangleright \mathcal{O}(1)$
3: while $k > 0$ do	$\triangleright \mathcal{O}(k)$
4: $val \leftarrow val + X[X.\text{length}() - k]$	$\triangleright \mathcal{O}(1)$
5: $k --$	$\triangleright \mathcal{O}(1)$
6: end while	
7: return val	$\triangleright \mathcal{O}(1)$

Complejidad: $\triangleright \mathcal{O}(n \log n) + \mathcal{O}(k) = \mathcal{O}(n \log n)$

Correctitud

Por Algo I: Nuestro algoritmo agarra los k elementos más grandes y devuelve la suma de ellos como solución.
Ahora demostremos que hacer eso da la solución óptima.

Correctitud

Por Algo I: Nuestro algoritmo agarra los k elementos más grandes y devuelve la suma de ellos como solución.

Ahora demostremos que hacer eso da la solución óptima.

Definamos otra solución S' óptima diferente a la nuestra S .

Ambas tienen k elementos que se suman. Ordenemos ambas de mayor a menor. Como son diferentes en algún momento difieren.

En ese momento hay dos opciones, o mi valor es mejor, o el otro valor es mejor. El otro valor no puede ser mejor: porque elegí este valor tomando el máximo. Y si mi valor es mejor: puedo cambiar el mío por el de él. Como S' era óptima y la mejoré llegamos a un absurdo.

Sale un break?



Figure: A descansar las neuronas



Explotación capitalista

Problema 2 (versión fácil)

Fer vive en una sociedad capitalista que lo explota y tiene muchos trabajos que hacer.

Cada trabajo tiene un deadline positivo asociado (o tiempo límite) y otorga una ganancia fija de 1 unidad si se hace antes de su deadline. Asimismo, cada trabajo le lleva 1 unidad de tiempo y sólo puede hacer un trabajo a la vez. Como Fer vive en una sociedad capitalista, necesita maximizar su ganancia.

Dar un algoritmo eficiente que tome una secuencia de trabajos a hacer y devuelva la secuencia que maximiza la ganancia posible, demostrar su correctitud.

Pseudocódigo

Algorithm 4 maxGanancia(deadlines: [Int]) \rightarrow [Int]

```
1: deadlines.sortAscendente()  $\triangleright \mathcal{O}(n \log n)$ 
2:  $B \leftarrow []$   $\triangleright \mathcal{O}(1)$ 
3: for deadline  $\in$  deadlines do  $\triangleright \mathcal{O}(n)$ 
4:   if deadline  $>$  B.length() then  $\triangleright \mathcal{O}(1)$ 
5:     B.append(deadline)  $\triangleright \mathcal{O}(1)$  amortizado
6:   end if
7: end for
8: return B  $\triangleright \mathcal{O}(1)$ 
```

Complejidad:

Pseudocódigo

Algorithm 5 maxGanancia(deadlines: [Int]) \rightarrow [Int]

```
1: deadlines.sortAscendente()  $\triangleright \mathcal{O}(n \log n)$ 
2:  $B \leftarrow []$   $\triangleright \mathcal{O}(1)$ 
3: for deadline  $\in$  deadlines do  $\triangleright \mathcal{O}(n)$ 
4:   if deadline  $>$  B.length() then  $\triangleright \mathcal{O}(1)$ 
5:     B.append(deadline)  $\triangleright \mathcal{O}(1)$  amortizado
6:   end if
7: end for
8: return B  $\triangleright \mathcal{O}(1)$ 
```

Complejidad: $\triangleright \mathcal{O}(n \log n) + \mathcal{O}(n) = \mathcal{O}(n \log n)$



Explotación capitalista

Problema 2 (versión difícil)

Mismo problema que antes pero ahora los trabajos pueden dar ganancias diferentes (como en la vida real).



Explotación capitalista

Problema 2 (versión difícil)

Mismo problema que antes pero ahora los trabajos pueden dar ganancias diferentes (como en la vida real).

Pregunta: El algoritmo anterior sigue funcionando?



Explotación capitalista

Problema 2 (versión difícil)

Mismo problema que antes pero ahora los trabajos pueden dar ganancias diferentes (como en la vida real).

Pregunta: El algoritmo anterior sigue funcionando? **No.**



Explotación capitalista

Problema 2 (versión difícil)

Mismo problema que antes pero ahora los trabajos pueden dar ganancias diferentes (como en la vida real).

Pregunta: El algoritmo anterior sigue funcionando? **No.**

Idea: Ordenemos primero por ganancia descendente y después por deadline ascendente con un sort estable.

Pseudocódigo

Algorithm 6 maxGanancia(trabajos: $[(\text{Int}, \text{Int})]$) $\rightarrow [(\text{Int}, \text{Int})]$

```
1: trabajos.sortPorGananciaDescendente()           ▷  $\mathcal{O}(n \log n)$ 
2: trabajos.sortPorDeadlineAscendente()             ▷  $\mathcal{O}(n \log n)$ 
3:  $B \leftarrow []$                                    ▷  $\mathcal{O}(1)$ 
4: for trabajo  $\in$  trabajos do                       ▷  $\mathcal{O}(n)$ 
5:     if trabajo.deadline > B.length() then         ▷  $\mathcal{O}(1)$ 
6:         B.append(trabajo)                          ▷  $\mathcal{O}(1)$  amortizado
7:     end if
8: end for
9: return B                                           ▷  $\mathcal{O}(1)$ 
```

Pseudocódigo

Algorithm 7 maxGanancia(trabajos: $[(\text{Int}, \text{Int})]$) $\rightarrow [(\text{Int}, \text{Int})]$

```
1: trabajos.sortPorGananciaDescendente()           ▷  $\mathcal{O}(n \log n)$ 
2: trabajos.sortPorDeadlineAscendente()             ▷  $\mathcal{O}(n \log n)$ 
3:  $B \leftarrow []$                                    ▷  $\mathcal{O}(1)$ 
4: for trabajo  $\in$  trabajos do                       ▷  $\mathcal{O}(n)$ 
5:     if trabajo.deadline > B.length() then         ▷  $\mathcal{O}(1)$ 
6:         B.append(trabajo)                          ▷  $\mathcal{O}(1)$  amortizado
7:     end if
8: end for
9: return B                                           ▷  $\mathcal{O}(1)$ 
```

Ojo, esto **no** funciona,

Pseudocódigo

Algorithm 8 maxGanancia(trabajos: $[(\text{Int}, \text{Int})]$) $\rightarrow [(\text{Int}, \text{Int})]$

```
1: trabajos.sortPorGananciaDescendente()           ▷  $\mathcal{O}(n \log n)$ 
2: trabajos.sortPorDeadlineAscendente()             ▷  $\mathcal{O}(n \log n)$ 
3:  $B \leftarrow []$                                    ▷  $\mathcal{O}(1)$ 
4: for trabajo  $\in$  trabajos do                       ▷  $\mathcal{O}(n)$ 
5:   if trabajo.deadline > B.length() then           ▷  $\mathcal{O}(1)$ 
6:     B.append(trabajo)                             ▷  $\mathcal{O}(1)$  amortizado
7:   end if
8: end for
9: return B                                           ▷  $\mathcal{O}(1)$ 
```

Ojo, esto **no** funciona, contraejemplo: $[(1,1), (2,2), (2,2)]$



Auto a mardel

Tomás quiere viajar de Buenos Aires a Mar del Plata en su flamante Renault 12. Como está preocupado por la autonomía de su vehículo, se tomó el tiempo de anotar las distintas estaciones de servicio que se encuentran en el camino. Modeló el mismo como un segmento de 0 a M , donde Buenos aires está en el kilómetro 0, Mar del Plata en el M , y las distintas estaciones de servicio están ubicadas en los kilómetros $0 = x_1 \leq x_2 \leq \dots x_n \leq M$.

Razonablemente, Tomás quiere minimizar la cantidad de paradas para cargar nafta. Él sabe que su auto es capaz de hacer hasta C kilómetros con el tanque lleno, y que al comenzar el viaje este está vacío.



Auto a mardel II

- Proponer un algoritmo *greedy* que indique cuál es la cantidad mínima de paradas para cargar nafta que debe hacer Tomás, y que aparte devuelva el conjunto de estaciones en las que hay que detenerse. Probar su correctitud.
- Dar una implementación de complejidad temporal $O(n)$ del algoritmo.

Explotación capitalista (con ganancias)

Cambiamos el problema de los trabajos. Ahora cada trabajo además de un deadline tiene una ganancia asociada. Es decir, la diferencia con la versión anterior es que antes las ganancias estaban fijas y eran todas 1, ahora pueden ser diferentes.



Gracias!