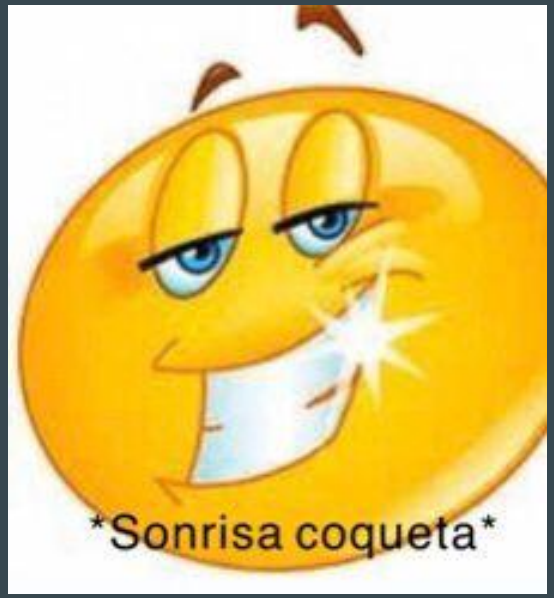
AYUDANTIA 10+1

...

Paula Grune, Agustín Gutiérrez,
Gustavo Salinas, Joaquín Viñuela

MATERIAL DE APOYO

1. Cheatsheet C (notion resumen)
2. Ejercicios de práctica C
3. Cápsulas de semestres pasados



Dónde encuentro esto?

Links en ReadMe carpeta “Ayudantías” del repo

Las 4 técnicas vistas

Para atacar los problemas complejos que se nos han presentado, contamos con las siguientes estrategias:

- Dividir para conquistar (D&C)
- Backtracking
- Algoritmos codiciosos (a.k.a. Greedy)
- Programacíon dinámica

Un breve repaso...

Dividir para conquistar

- Dividir el problema en problemas más pequeños hasta que sea trivial resolverlos
- Los problemas son **disjuntos**: Cada subproblema es independiente del resto y se resuelve aparte
- La solución del problema grande se construye a partir de las soluciones de los problemas pequeños

EJEMPLO: Ordenar un array con MergeSort

Backtracking

- Definimos variables, dominios y restricciones
- Asignamos valores a las variables de forma ordenada
- Si alguna restricción se rompe, deshacemos la última asignación y probamos nuevamente
- Iteramos hasta resolver el problema
- GRAN complejidad → $O(K^n)$

EJEMPLO: Salir de un laberinto dándose golpes contra la pared

Un breve repaso...

Greedy

- Iremos haciendo decisiones de forma secuencial
- Para cada paso, tomaremos la decisión que parezca mejor en el momento
- ¿Cuál es la mejor decisión? La que minimice/maximice cierta función
- Eficientes, pero podría llevarnos a mínimos/máximos locales

EJEMPLO: Queriendo encontrar el camino más corto en un grafo, tomar siempre la arista de menor peso

Programación Dinámica

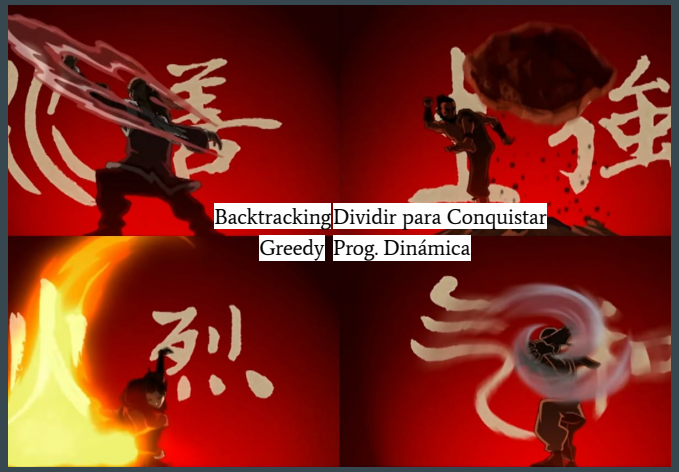
- Al igual que Dividir para Conquistar, dividir el problema en subproblemas pequeños que tienen **subestructura óptima**
- Estos subproblemas NO son disjuntos, si no que se repiten al descomponer otros problemas
- Resolvemos los problemas sencillos y guardamos sus soluciones para usarlas al componer las soluciones de problemas mayores

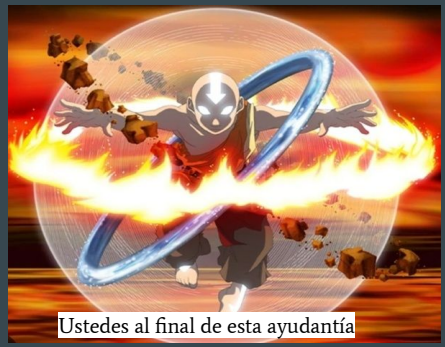
EJEMPLO: Dar vuelto con la menor cantidad de monedas

¿Cuál debemos usar?

- Dependiendo del problema, todos podrían encontrar solución, pero algunas soluciones podrían ser mejores que otras
- Hay que hacerse preguntas tales como:
  a. ¿Queremos maximizar minimizar algo?
  b. ¿Hay restricciones?
  c. ¿Hay problemas similares más pequeños? ¿Se repiten?

- Queremos encontrar la mejor estrategia para el problema





PROBLEMA 1

Supongamos que tenemos **N libros** con $\mathbf{n_i}$ páginas cada uno (almacenados en el array **PAGES**) que tenemos que asignar a **M estudiantes**. Debemos asignar los libros de forma que se minimice el máximo de páginas que deberá leer el estudiante más cargado. Indica como debemos hacer la asignación de libros a los estudiantes.

ejercicio basado en este link

PROBLEMA 1

Supongamos que tenemos **N libros** con $\mathbf{n_i}$ páginas cada uno (almacenados en el array **PAGES**) que tenemos que asignar a **M estudiantes**. Debemos asignar los libros de forma que se minimice el máximo de páginas que deberá leer el estudiante más cargado. Indica como debemos hacer la asignación de libros a los estudiantes.

Explicación

Tenemos N=9 libros, cuyos largos son [6, 1, 3, 2, 2, 4, 1, 2, 1] que queremos asignar a Paula, Gustavo y Agustín (M=3). Algunas posibles asignaciones son:

Paula: [6, 1, 2, 2]: leerá 11 páginas

Gustavo: [3, 4]: leerá 7 páginas

Agustín: [1, 2, 1]: leerá 4 páginas

El máximo de páginas leídas son 11

PROBLEMA 1

Supongamos que tenemos **N libros** con n_i páginas cada uno (almacenados en el array **PAGES**) que tenemos que asignar a **M estudiantes**. Debemos asignar los libros de forma que se minimice el máximo de páginas que deberá leer el estudiante más cargado. Indica como debemos hacer la asignación de libros a los estudiantes.

Explicación

Tenemos N=9 libros, cuyos largos son [6, 1, 3, 2, 2, 4, 1, 2, 1] que queremos asignar a Paula, Gustavo y Agustín (M=3). Algunas posibles asignaciones son:

Paula: [6, 1, 2, 2]: leerá 11 páginas

Gustavo: [3, 4]: leerá 7 páginas

Agustín: [1, 2, 1]: leerá 4 páginas

El máximo de páginas leídas son 11

Queremos encontrar una asignación que minimice este número

PROBLEMA 1: Explicación

Otra posible asignación para [6, 1, 3, 2, 2, 4, 1, 2, 1] es:

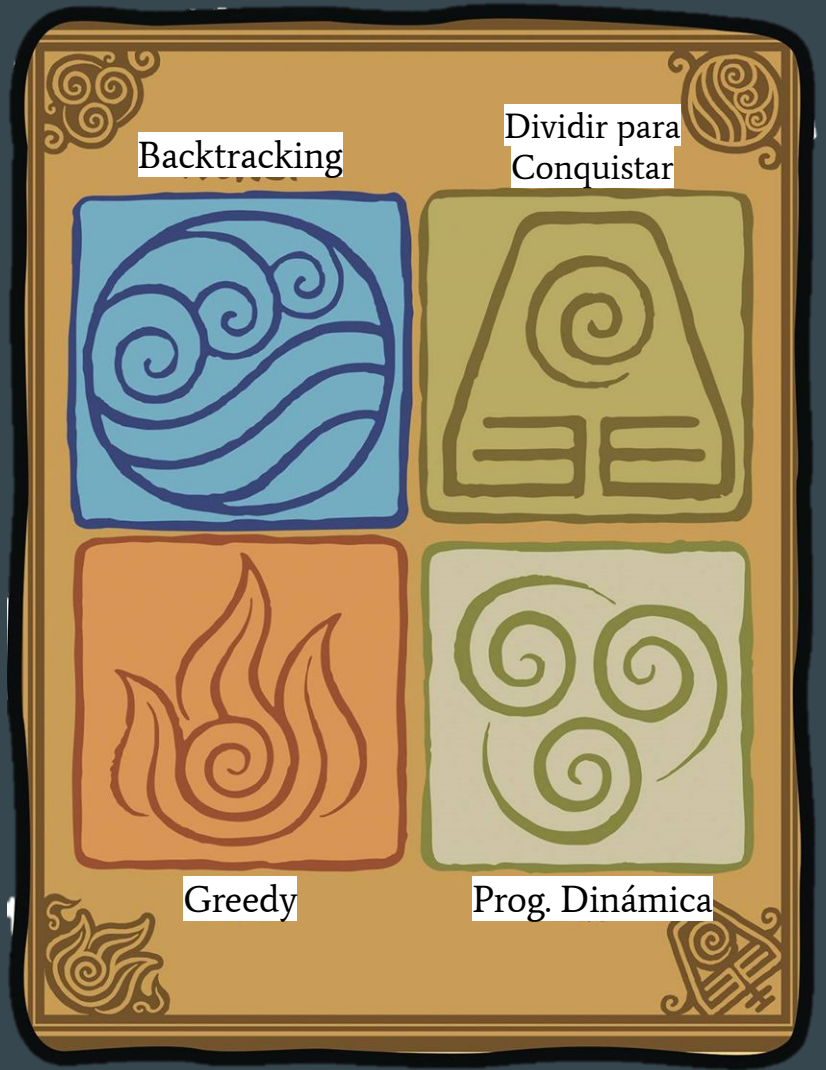
Paula: [6, 2]: leerá 8 páginas

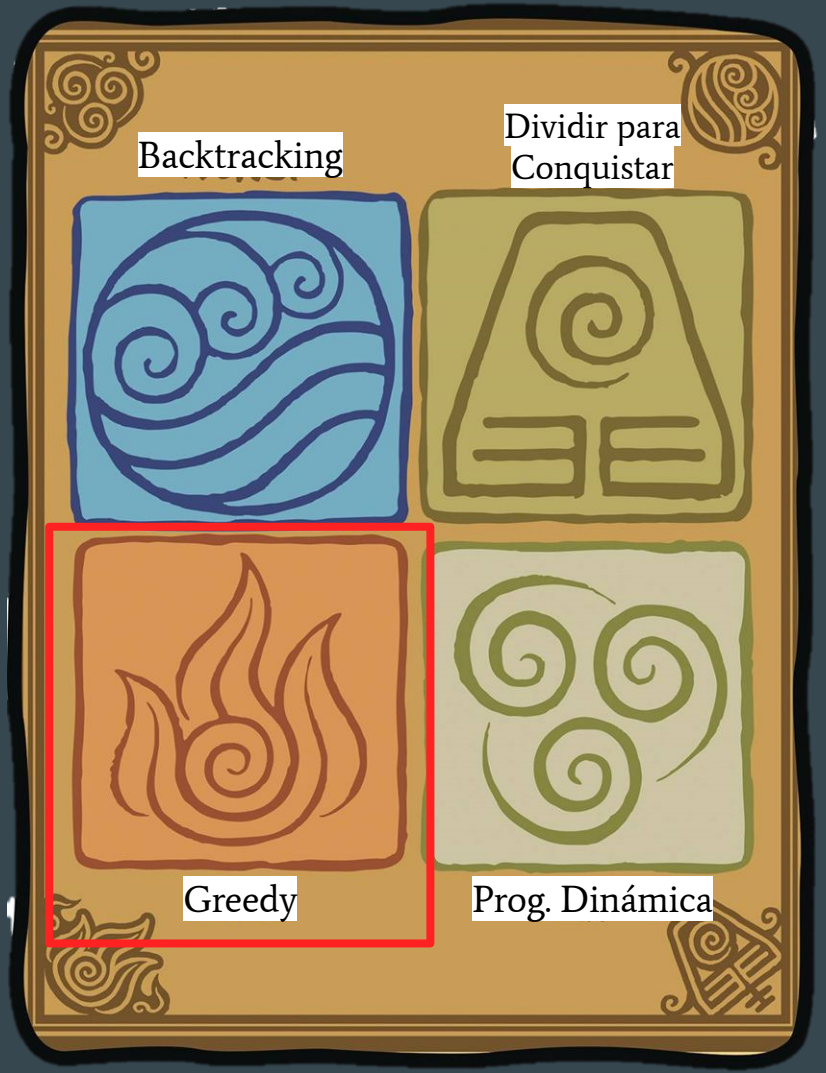
Gustavo: [1, 2, 2, 1, 1]: leerá 7 páginas

Agustín: [3, 4]: leerá 7 páginas

El máximo de páginas leídas son 8

La solución a este problema es precisamente 8. Ahora la pregunta es, ¿Cómo encontramos esta asignación óptima?


Backtracking
Dividir para Conquistar
Greedy
Prog. Dinámica


Backtracking
Dividir para Conquistar
Greedy
Prog. Dinámica

PROBLEMA 1: Greedy

Una estrategia codiciosa podría ser asignar el libro siguiente al estudiante con la menor cantidad de páginas asignadas. Con esto distribuimos mejor las páginas en todos los “buckets” para evitar que el máximo de páginas asignadas crezca

[6, 1, 3, 2, 2, 4, 1, 2, 1]

Paula: [ ]: leerá 0 páginas

Gustavo: [ ]: leerá 0 páginas

Agustín: [ ]: leerá 0 páginas

PROBLEMA 1: Greedy

Una estrategia codiciosa podría ser asignar el libro siguiente al estudiante con la menor cantidad de páginas asignadas. Con esto distribuimos mejor las páginas en todos los “buckets” para evitar que el máximo de páginas asignadas crezca

[6, 1, 3, 2, 2, 4, 1, 2, 1]

Paula: [6]: leerá 6 páginas

Gustavo: []: leerá 0 páginas

Agustín: []: leerá 0 páginas

PROBLEMA 1: Greedy

Una estrategia codiciosa podría ser asignar el libro siguiente al estudiante con la menor cantidad de páginas asignadas. Con esto distribuimos mejor las páginas en todos los “buckets” para evitar que el máximo de páginas asignadas crezca

[6, 1, 3, 2, 2, 4, 1, 2, 1]

Paula: [6]: leerá 6 páginas

Gustavo: [1]: leerá 1 página

Agustín: []: leerá 0 páginas

PROBLEMA 1: Greedy

Una estrategia codiciosa podría ser asignar el libro siguiente al estudiante con la menor cantidad de páginas asignadas. Con esto distribuimos mejor las páginas en todos los “buckets” para evitar que el máximo de páginas asignadas crezca

[6, 1, 3, 2, 2, 4, 1, 2, 1]

Paula: [6]: leerá 6 páginas

Gustavo: [1]: leerá 1 página

Agustín: [3]: leerá 3 páginas

PROBLEMA 1: Greedy

Una estrategia codiciosa podría ser asignar el libro siguiente al estudiante con la menor cantidad de páginas asignadas. Con esto distribuimos mejor las páginas en todos los “buckets” para evitar que el máximo de páginas asignadas crezca

[6, 1, 3, 2, 2, 4, 1, 2, 1]

Paula: [6]: leerá 6 páginas

Gustavo: [1, 2]: leerá 3 páginas

Agustín: [3]: leerá 3 páginas

PROBLEMA 1: Greedy

Una estrategia codiciosa podría ser asignar el libro siguiente al estudiante con la menor cantidad de páginas asignadas. Con esto distribuimos mejor las páginas en todos los “buckets” para evitar que el máximo de páginas asignadas crezca

[6, 1, 3, 2, 2, 4, 1, 2, 1]

Paula: [6]: leerá 6 páginas

Gustavo: [1, 2, 2]: leerá 5 páginas

Agustín: [3]: leerá 3 páginas

PROBLEMA 1: Greedy

Una estrategia codiciosa podría ser asignar el libro siguiente al estudiante con la menor cantidad de páginas asignadas. Con esto distribuimos mejor las páginas en todos los “buckets” para evitar que el máximo de páginas asignadas crezca

[6, 1, 3, 2, 2, 4, 1, 2, 1]

Paula: [6]: leerá 6 páginas

Gustavo: [1, 2, 2]: leerá 5 páginas

Agustín: [3, 4]: leerá 7 páginas

PROBLEMA 1: Greedy

Una estrategia codiciosa podría ser asignar el libro siguiente al estudiante con la menor cantidad de páginas asignadas. Con esto distribuimos mejor las páginas en todos los “buckets” para evitar que el máximo de páginas asignadas crezca

[6, 1, 3, 2, 2, 4, 1, 2, 1]

Paula: [6]: leerá 6 páginas

Gustavo: [1, 2, 2, 1]: leerá 6 páginas

Agustín: [3, 4]: leerá 7 páginas

PROBLEMA 1: Greedy

Una estrategia codiciosa podría ser asignar el libro siguiente al estudiante con la menor cantidad de páginas asignadas. Con esto distribuimos mejor las páginas en todos los “buckets” para evitar que el máximo de páginas asignadas crezca

[6, 1, 3, 2, 2, 4, 1, 2, 1]

Paula: [6, 2]: leerá 8 páginas

Gustavo: [1, 2, 2, 1]: leerá 6 páginas

Agustín: [3, 4]: leerá 7 páginas

PROBLEMA 1: Greedy

Una estrategia codiciosa podría ser asignar el libro siguiente al estudiante con la menor cantidad de páginas asignadas. Con esto distribuimos mejor las páginas en todos los “buckets” para evitar que el máximo de páginas asignadas crezca

[6, 1, 3, 2, 2, 4, 1, 2, 1]

Paula: [6, 2]: leerá 8 páginas

Gustavo: [1, 2, 2, 1, 1]: leerá 7 páginas

Agustín: [3, 4]: leerá 7 páginas

Llegamos al óptimo!
(pero de la forma correcta?)

PROBLEMA 1: Greedy

Pensemos que ahora tenemos esta otra configuración inicial, con otro listado de libros y ahora M = 2 (Se asigna el siguiente libro al estudiante con la menor cantidad de páginas asignadas)

[8, 15, 10, 20, 8]

Paula: [ ]: leerá 0 páginas

Gustavo: [ ]: leerá 0 páginas

PROBLEMA 1: Greedy

Pensemos que ahora tenemos esta otra configuración inicial, con otro listado de libros y ahora M = 2 (Se asigna el siguiente libro al estudiante con la menor cantidad de páginas asignadas)

[8, 15, 10, 20, 8]

Paula: [8]: leerá 8 páginas

Gustavo: [ ]: leerá 0 páginas

PROBLEMA 1: Greedy

Pensemos que ahora tenemos esta otra configuración inicial, con otro listado de libros y ahora M = 2 (Se asigna el siguiente libro al estudiante con la menor cantidad de páginas asignadas)

[8, 15, 10, 20, 8]

Paula: [8]: leerá 8 páginas

Gustavo: [15]: leerá 15 páginas

PROBLEMA 1: Greedy

Pensemos que ahora tenemos esta otra configuración inicial, con otro listado de libros y ahora M = 2 (Se asigna el siguiente libro al estudiante con la menor cantidad de páginas asignadas)

[8, 15, 10, 20, 8]

Paula: [8, 10]: leerá 18 páginas

Gustavo: [15]: leerá 15 páginas

PROBLEMA 1: Greedy

Pensemos que ahora tenemos esta otra configuración inicial, con otro listado de libros y ahora M = 2 (Se asigna el siguiente libro al estudiante con la menor cantidad de páginas asignadas)

[8, 15, 10, 20, 8]

Paula: [8, 10]: leerá 18 páginas

Gustavo: [15, 20]: leerá 35 páginas

PROBLEMA 1: Greedy

Pensemos que ahora tenemos esta otra configuración inicial, con otro listado de libros y ahora M = 2 (Se asigna el siguiente libro al estudiante con la menor cantidad de páginas asignadas)

[8, 15, 10, 20, 8]

Paula: [8, 10, 8]: leerá 26 páginas

Gustavo: [15, 20]: leerá 35 páginas

Según esta estrategia, el alumno más cargado leerá en el mejor de los casos 35 páginas

PROBLEMA 1: Greedy

Pensemos que ahora tenemos esta otra configuración inicial, con otro listado de libros y ahora M = 2 (Se asigna el siguiente libro al estudiante con la menor cantidad de páginas asignadas)

[8, 15, 10, 20, 8]

Paula: [8, 10, 8]: leerá 26 páginas

Gustavo: [15, 20]: leerá 35 páginas

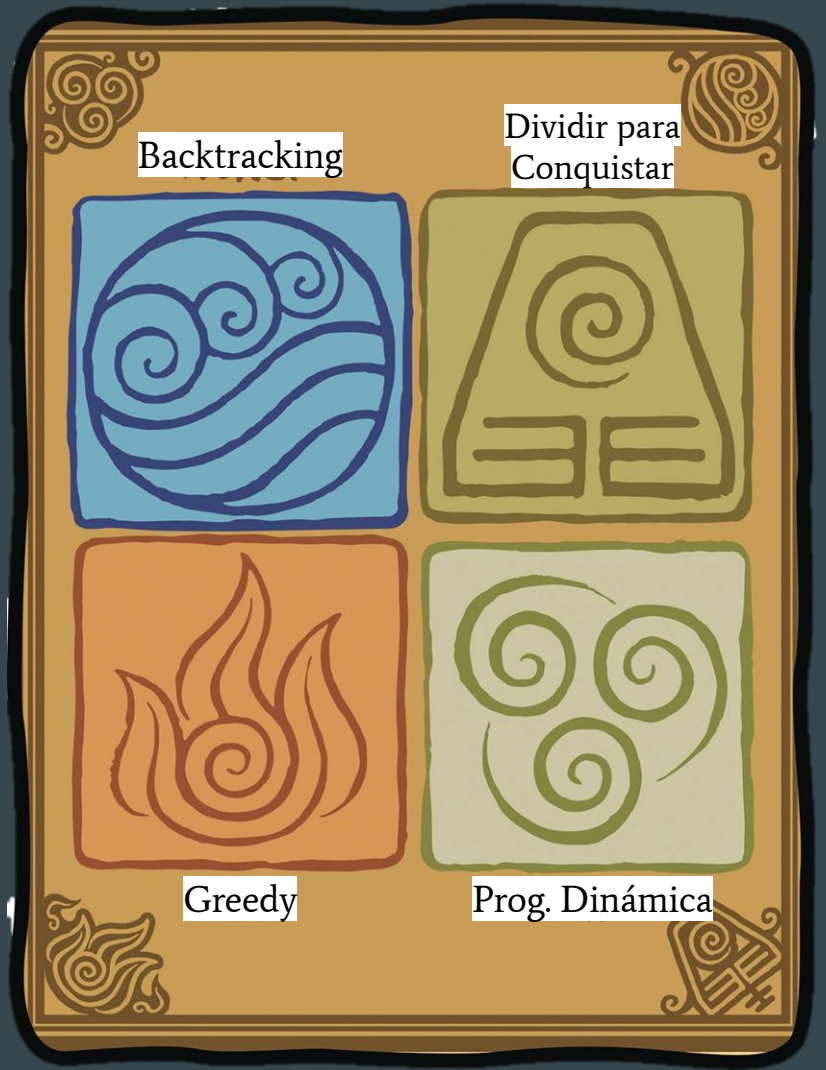
Según esta estrategia, el alumno más cargado leerá en el mejor de los casos 35 páginas

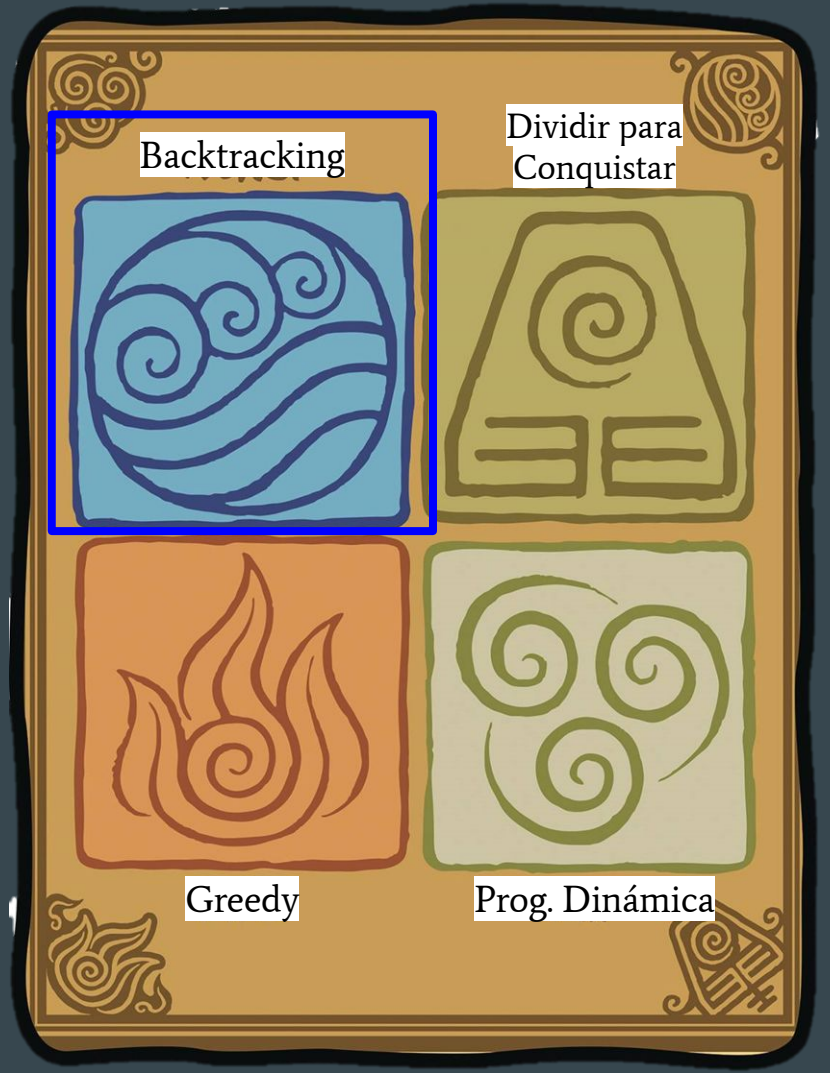
Sin embargo, tenemos esta otra asignación que sí da el mínimo. Por lo tanto, esta estrategia greedy falla

[8, 15, 10, 20, 8]

Paula: [8, 15, 8]: leerá 31 páginas

Gustavo: [10, 20]: leerá 30 páginas


Backtracking
Dividir para Conquistar
Greedy
Prog. Dinámica


Backtracking
Dividir para Conquistar
Greedy
Prog. Dinámica

PROBLEMA 1: Backtracking

Vamos a ir asignando ordenadamente todos los libros a los estudiantes y calcularemos el máximo de la asignación. Luego compararemos ese máximo con el mínimo actual, y actualizamos en caso de que el nuevo valor sea menor

Libro	8	15	10	20	8
Se asigna al estudiante...	-	-	-	-	-

Mínimo actual : ∞

Paula: [ ]: leerá 0 páginas

Gustavo: [ ]: leerá 0 páginas

PROBLEMA 1: Backtracking

Vamos a ir asignando ordenadamente todos los libros a los estudiantes y calcularemos el máximo de la asignación. Luego compararemos ese máximo con el mínimo actual, y actualizamos en caso de que el nuevo valor sea menor

Libro	8	15	10	20	8
Se asigna al estudiante...	P	P	P	P	G

Mínimo actual : 53

Paula: [8, 15, 10, 20]: leerá 53 páginas

53 < ∞: Actualizamos

Gustavo: [8]: leerá 8 páginas

PROBLEMA 1: Backtracking

Vamos a ir asignando ordenadamente todos los libros a los estudiantes y calcularemos el máximo de la asignación. Luego compararemos ese máximo con el mínimo actual, y actualizamos en caso de que el nuevo valor sea menor

Libro	8	15	10	20	8
Se asigna al estudiante...	P	P	P	G	P

Mínimo actual : 41

Paula: [8, 15, 10, 8]: leerá 41 páginas

41 < 53: Actualizamos

Gustavo: [20]: leerá 20 páginas

PROBLEMA 1: Backtracking

Vamos a ir asignando ordenadamente todos los libros a los estudiantes y calcularemos el máximo de la asignación. Luego compararemos ese máximo con el mínimo actual, y actualizamos en caso de que el nuevo valor sea menor

Libro	8	15	10	20	8
Se asigna al estudiante...	P	P	P	G	G

Mínimo actual : 33

Paula: [8, 15, 10, 8]: leerá 33 páginas

33 < 41: Actualizamos

Gustavo: [20]: leerá 28 páginas

PROBLEMA 1: Backtracking

Vamos a ir asignando ordenadamente todos los libros a los estudiantes y calcularemos el máximo de la asignación. Luego compararemos ese máximo con el mínimo actual, y actualizamos en caso de que el nuevo valor sea menor

Libro	8	15	10	20	8
Se asigna al estudiante...	P	P	G	P	P

Mínimo actual : 33

Paula: [8, 15, 20, 8]: leerá 51 páginas

Gustavo: [10]: leerá 10 páginas

51 > 33: Seguimos como estamos

PROBLEMA 1: Backtracking

Vamos a ir asignando ordenadamente todos los libros a los estudiantes y calcularemos el máximo de la asignación. Luego compararemos ese máximo con el mínimo actual, y actualizamos en caso de que el nuevo valor sea menor

Libro	8	15	10	20	8
Se asigna al estudiante...	P	P	G	P	G

Mínimo actual : 33

Paula: [8, 15, 20]: leerá 43 páginas

Gustavo: [10, 8]: leerá 18 páginas

43 > 33: Seguimos como estamos

PROBLEMA 1: Backtracking

Vamos a ir asignando ordenadamente todos los libros a los estudiantes y calcularemos el máximo de la asignación. Luego compararemos ese máximo con el mínimo actual, y actualizamos en caso de que el nuevo valor sea menor

Libro	8	15	10	20	8
Se asigna al estudiante...	P	P	G	G	P

Mínimo actual : 31

Paula: [8, 15, 8]: leerá 31 páginas

31 < 33: Actualizamos

Gustavo: [10, 20]: leerá 30 páginas

PROBLEMA 1: Backtracking

Vamos a ir asignando ordenadamente todos los libros a los estudiantes y calcularemos el máximo de la asignación. Luego compararemos ese máximo con el mínimo actual, y actualizamos en caso de que el nuevo valor sea menor

Libro	8	15	10	20	8
Se asigna al estudiante...	P	P	G	G	G

Mínimo actual : 31

Paula: [8, 15]: leerá 23 páginas

Gustavo: [10, 20, 8]: leerá 38 páginas

38 > 31: Seguimos como estamos

PROBLEMA 1: Backtracking

Vamos a ir asignando ordenadamente todos los libros a los estudiantes y calcularemos el máximo de la asignación. Luego compararemos ese máximo con el mínimo actual, y actualizamos en caso de que el nuevo valor sea menor

Libro	8	15	10	20	8
Se asigna al estudiante...	P	G	P	P	P

Mínimo actual : 31

Paula: [8, 10, 20, 8]: leerá 46 páginas

Gustavo: [15]: leerá 15 páginas

46 > 31: Seguimos como estamos

and so on ...

PROBLEMA 1: Backtracking

De esta forma llegamos a que la estrategia de backtracking soluciona el problema de forma óptima. Recordemos que backtracking siempre encontrará una solución, pero a costa de una gran complejidad, y dependiendo del caso puede que no encuentre lo que buscamos (por ejemplo, no podemos saber a priori si lo encontrado fue el óptimo)

Tips para saber qué estrategia usar:

- Teniendo una estrategia en mente, tratar de buscar un contraejemplo
- Si podemos descomponer en subproblemas, quizás D&C o Prog. Dinámica sean buen acercamiento
- Greedy se ve bien a priori, pero puede caer en mínimos locales
- Si todo falla, backtracking es la vieja confiable

Problema 2: Knapsack problem (problema de la mochila)

Dado un set de objetos que tienen un valor y un peso, encuentre el valor máximo obtenido en la mochila según una restricción de peso.

Cree una solución con backtracking, una con PD y una con un algoritmo greedy

Ej:

weights = [2, 3, 4, 5]

values = [3, 4, 5, 6]

W = 5

Resolver en la pizarra

Backtracking

Un approach posible:

Para todos los elementos: probar insertarlos en la mochila, revisar si al insertarlo viola la restricción de peso, si es así ver si es la mejor solución encontrada, en caso contrario llamar a la función con el siguiente elemento. Una vez revisado el caso en donde sí se inserta, hacer el mismo procedimiento, pero NO utilizar el elemento en la solución.

Complejidad: O(2^n)

```python
def knapsack_backtracking(weights, values, W):
    n = len(weights)
    max_value = 0

    def backtrack(i, current_weight, current_value):
        nonlocal max_value
        if current_weight > W:
            return
        if current_value > max_value:
            max_value = current_value
        if i == n:
            return
        backtrack(i + 1, current_weight + weights[i], current_value + values[i])
        backtrack(i + 1, current_weight, current_value)

    backtrack(0, 0, 0)
    return max_value

weights = [2, 3, 4, 5]
values = [3, 4, 5, 6]
W = 5
print(knapsack_backtracking(weights, values, W))
```

Programación dinámica

Posible approach:

Calcular el máximo valor para cada posible w en [0, W] e ir construyéndola con los resultados anteriores

Complejidad: O(n*W)

```python
def knapsack_dynamic_programming(weights, values, W):
    n = len(weights)
    # Create a 2D array dp where dp[i][j] represents the maximum value
    dp = [[0] * (W + 1) for _ in range(n + 1)]

    # Filling the dp table
    for i in range(1, n + 1):
        for w in range(W + 1):
            if weights[i - 1] <= w:
                dp[i][w] = max(dp[i - 1][w], dp[i - 1][w - weights[i - 1]] + values[i - 1])
            else:
                dp[i][w] = dp[i - 1][w]

    # Finding the items included in the optimal solution
    solution = []
    w = W
    for i in range(n, 0, -1):
        if dp[i][w] != dp[i - 1][w]:
            solution.append(i - 1)
            w -= weights[i - 1]

    solution.reverse()  # Optional: to print items in the order they were added

    # Print the dp table and the solution
    print("DP Table:")
    for row in dp:···

    print("\nItems included in the optimal solution (0-indexed):")
    for item in solution:···

    return dp[n][W]

# Example usage
weights = [2, 3, 4, 5]
values = [3, 4, 5, 6]
W = 6
max_value = knapsack_dynamic_programming(weights, values, W)
print(f"\nMaximum value obtainable: {max_value}")
```

Greedy

Probar insertar el elemento con el valor más denso (valor/peso) y repetir hasta no cumplir con la restricción de carga.

Complejidad: O(n*log(n))

```python
def knapsack_greedy(weights, values, W):
    n = len(weights)
    value_density = [(values[i] / weights[i], weights[i], values[i]) for i in range(n)]
    value_density.sort(reverse=True, key=lambda x: x[0])

    total_value = 0
    total_weight = 0

    for density, weight, value in value_density:
        if total_weight + weight <= W:
            total_weight += weight
            total_value += value
        else:
            remain = W - total_weight
            total_value += density * remain
            break

    return total_value

weights = [2, 3, 4, 5]
values = [3, 4, 5, 6]
W = 5
print(knapsack_greedy(weights, values, W))
```