Repaso I2

Hash, Greedy, Backtracking y Estrategias

Felipe Espinoza - Joaquín Viñuela - Isabella Cherubini - Paula Grune

Hashing

Tablas de Hash 2023-1

En el sistema de control de embarque de pasajeros en un aeropuerto se utiliza una tabla de hash T de tamaño M con encadenamiento para llevar un registro de los pasajeros autorizados para abordar el vuelo. Esta tabla almacena el identificador del pasajero (el RUT utilizado como llave), el código del vuelo y el 6 código de equipaje asociado a cada pasajero. Así, dado el RUT es posible recuperar de manera muy eficiente los demás datos asociados.

Parte (a)

(a) [2 ptos.] Proponga los algoritmos en pseudocódigo para los siguientes métodos:

(i) [1 pto.] `agregarPasajero(r,v,e)`: Recibe como entrada el RUT del pasajero, el código del vuelo y el código de equipaje. Agrega los datos a la tabla T correctamente.

Solución.

input : RUT r, código de vuelo v, código de equipaje e

`agregarPasajero`(r, v, e):

```
1   i ← hash(r)
2   insertarEnLista(T[i], r, v, e)
```

donde `insertarEnLista` inserta la información del pasajero en la lista ligada de colisiones de la celda $T[i]$.

Parte (a)

(ii) [1 pto.] `buscarPasajero(r)`: Recibe como entrada el RUT del pasajero. Si lo encuentra, devuelve el código del vuelo y el código de equipaje, y si no lo encuentra devuelve -1 indicando que el pasajero no está registrado.

Solución.

input : RUT r

`buscarPasajero`(r)**:**

1 $i \leftarrow$ `hash`(r)

2 $p \leftarrow$ `buscarEnLista`$(T[i], r)$

if $p \neq \varnothing$ **:**

return p

return -1

donde `BuscarEnLista` itera en la lista de la celda $T[i]$ hasta encontrar el rut r.

Parte (b)

[2 ptos.] Proponga el pseudocódigo para el método `pasajerosRegistrados()` que, a partir de la tabla T, permite obtener en tiempo lineal $\mathcal{O}(n)$ la nómina (código de vuelo, RUT) de todos los pasajeros registrados en un momento dado, ordenada por código de vuelo. Asuma que el código del vuelo es de la forma AA9999 (dos letras seguidas de 4 números). *Pista:* Puede utilizar como apoyo una estructura de datos adicional, asumiendo que el factor de carga de T es siempre menor a 1, y aplicar/modificar algoritmos vistos en clases en su solución.

Parte (b)

Solución.

```
   pasajerosRegistrados():
1      R ← arreglo de M celdas
2      for e elemento en alguna lista de la tabla de hash :
3          Guardar en R los datos e.vuelo, e.rut de e
       return RadixSort(R)
```

Observamos que R tiene el tamaño adecuado, dado que como el factor de carga es menos a 1, hay menos de M pasajeros en la tabla. Además, se asume que Radix ordena R en base al atributo código de vuelo.

Parte (c)

[2 ptos.] Un problema habitual es el equipaje perdido. Proponga un algoritmo en pseudocódigo para el método `buscarEquipaje(e)`: Recibe como entrada un código de equipaje (un número de 15 dígitos) y si lo encuentra, retorna el RUT y el código de vuelo del pasajero al que pertenece, -1 si no lo encuentra. Esta función debe operar en tiempo $\mathcal{O}(1)$. *Pista:* considere modificar primero `agregarPasajero()` de la parte (a) de modo de contar con una tabla de hash auxiliar E de tamaño M con encadenamiento que utilice como llave el código de equipaje y lo relacione con el RUT del pasajero.

Solución.

Primero modificamos `agregarPasajero` según

input : RUT r, código de vuelo v, código de equipaje e

`agregarPasajero`(r, v, e)**:**

1 $i \leftarrow$ `hash`(r)
2 $j \leftarrow$ `hash`$'(e)$
3 $p \leftarrow$ `insertarEnLista`$(T[i], r, v, e)$
4 `insertarEnLista`$(E[j], e, p)$

donde E es una segunda tabla de hash para los equipajes como llave (con función de hash `hash`$'$), y donde cada elemento almacenado en las listas de colisiones contiene un puntero p a la información del pasajero en la primera tabla, el cual puede ser fácilmente retornado por el método que inserta en la tabla T.

input : Código de equipaje e

`buscarEquipaje`(e)**:**

1 $i \leftarrow$ `hash`$'(e)$
2 $p \leftarrow$ `buscarEnLista`$(E[i], e)$
3 **return** `obtenerInfo`(p)

donde `obtenerInfo` retorna la información contenida en el objeto apuntado por el puntero p.

Grafos

Kosaraju y CFC

Se tiene un grafo direccional $G(V, E)$. Se define el grafo $G'(V, E')$ como un grafo que tiene el mismo grafo de componentes que G y cuya cantidad de aristas $|E'|$ es mínima. Escribe un algoritmo eficiente que calcule

$\Delta E = |E| - |E'|$.

Podemos separar las aristas E0 en dos grupos:

1. Las aristas que están dentro de una CFC, E’0

2. Las aristas que están fuera de todas las CFC, E’x

En el primer caso, tenemos que por cada CFC G con n > 1 nodos, E’0 tiene n aristas, ya que es necesario preservar un solo ciclo que pase por todos los nodos de la CFC para que siga siendo una CFC en G’.

Es decir, si definimos

$$C = \{c \mid c \text{ es una CFC de } G\}$$

Entonces

$$|E'_o| = \sum_{c \in C,\ |c| > 1} |c|$$

En el segundo caso, tenemos que las aristas que no están dentro de ninguna CFC son las aristas que van de una CFC a otra. En G’ no pueden haber múltiples aristas que conectan un mismo par de CFCs, ya que en el grafo de componentes se reducen a 1, por lo que sobrarían aristas.

Es decir, si H(C, F) es el grafo de componentes de G

$$|E'_x| = |F|$$

Teniendo esto, podemos calcular |E’| sin necesidad de computar un G’ . Pero sí va a ser necesario calcular el grafo de componentes. El algoritmo es como sigue:

Primero, para identificar las CFCs, usamos el algoritmo de Kosaraju como se vió en clases, pero le hacemos una pequeña modificación a la funci´on que asigna representantes para calcular el tamaño de la CFC:

```
assign(u, rep):
    total ← 0
    if u.rep = ∅:
        u.rep ← rep
        foreach v in α[u]:
            total ← total + assign(v, rep)
    return total
```

Teniendo eso podemos calcular |E'0| sumando la cantidad de nodos de cada CFC con más de un nodo, mientras que para calcular |E'x| simplemente computamos F:

$|E'|(G(V, E))$:

$|E'_o| \leftarrow 0$

Crear lista L vacía

Ejecutar $dfs(G)$ con tiempos

Insertar vértices en L en orden decreciente de tiempos f

for each u *in* L:

$n \leftarrow assign(u, u)$

if $n > 1$:

$|E'_o| \leftarrow |E'_o| + n$

$F \leftarrow \emptyset$

for each $(u, v) \in E$:

if $u.\text{rep} \neq v.\text{rep}$:

if $(u.\text{rep}, v.\text{rep}) \notin F$:

Agregar $(u.\text{rep}, v.\text{rep})$ a F

return $|E'_o| + |F|$

$|E'|(G(V, E))$:

$|E'_o| \leftarrow 0$

Crear lista L vacía

Ejecutar $dfs(G)$ con tiempos

Insertar vértices en L en orden decreciente de tiempos f

for each u *in* L:

$n \leftarrow assign(u, u)$

if $n > 1$:

$|E'_o| \leftarrow |E'_o| + n$

$F \leftarrow \emptyset$

for each $(u, v) \in E$:

if $u.\text{rep} \neq v.\text{rep}$:

if $(u.\text{rep}, v.\text{rep}) \notin F$:

Agregar $(u.\text{rep}, v.\text{rep})$ a F

return $|E'_o| + |F|$

Y una vez obtenido |E'|, podemos fácilmente retornar $\Delta E = |E| - |E'|$

Este algoritmo es O(V + E), ya que Kosaraju es O(V + E) y computar F es O(E)

Backtracking
vs
Greedy & DP

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 (Greedy)
- Programación dinámica (DP)

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

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

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

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

Queremos encontrar una asignación que minimice este número

Problema: Explicación

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

Paula: [6, 2]: leerá 8 páginas

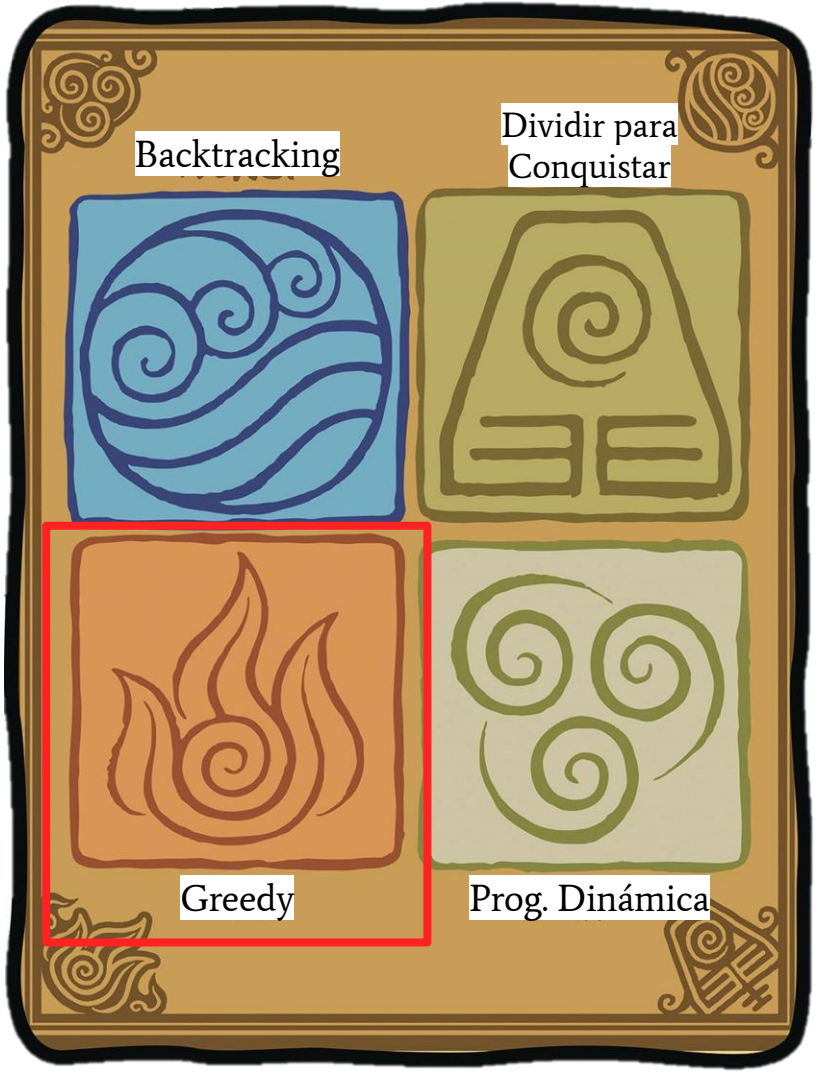
Gustavo: [1, 2, 2, 1, 1]: leerá 7 páginas

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

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

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

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

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

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

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

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

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

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

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?)

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

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

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

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

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 5 páginas

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 5 páginas

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 5 páginas

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

Greedy

[8, 15, 10, 20, 8]

Paula: [8,10,8]: leerá 26 páginas

Gustavo: [15,20]: leerá 5 páginas

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

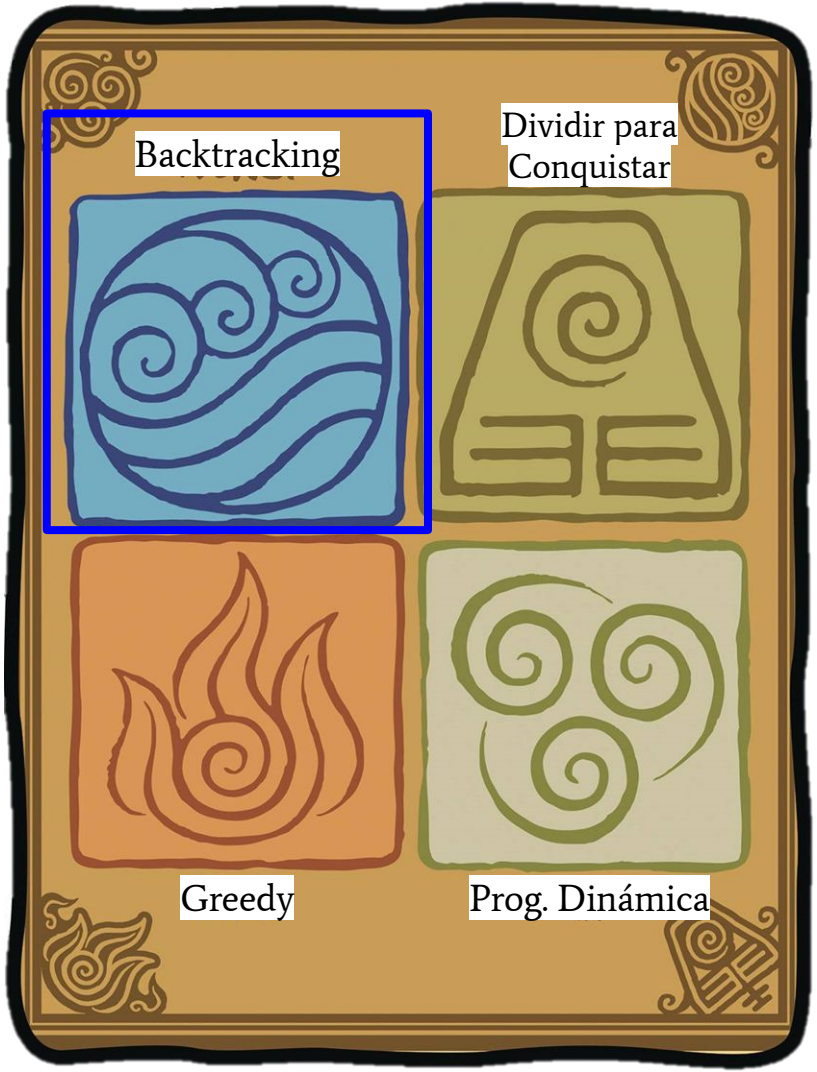
Sin embargo, tenemos esta otra asignación que sí da el mínimo. Por lo tanto, esta *estrategia greedy falla*

[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

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

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

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 : 31

Paula: [8,15,10,8]: leerá 41 páginas

41 < 53 : Actualizamos

Gustavo: [20]: leerá 20 páginas

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]: leerá 33 páginas

33 < 41 : Actualizamos

Gustavo: [20,8]: leerá 28 páginas

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

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: [18]: leerá 18 páginas

43 > 33 : Seguimos como estamos

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

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 : 33

Paula: [8,10,20,8]: leerá 46 páginas

Gustavo: [15]: leerá 15 páginas

46 > 31 : Seguimos como estamos

Y así podemos seguir hasta ver todos los casos …

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

Éxito en la I2 :3