

# Repaso I2

Clase 24

IIC 2133 - Sección 2

Prof. Mario Droguett

# Sumario

**Introducción**

Tablas de hash

Estrategias algorítmicas

DFS

Un ejemplo de prueba

Cierre

# Los grandes temas

En esta interrogación abarcamos los siguientes macrotemas

1. Diccionarios
  - Tablas de hash
2. Estrategias algorítmicas
  - Backtracking
  - Algoritmos codiciosos (greedy)
  - Programación dinámica
3. Intro a algoritmos en grafos
  - DFS (detección de ciclos, orden topológico, Kosaraju)

Más de un tema podría corresponderse con una misma pregunta en la I2

# Interrogación 2

Objetivos a evaluar en la I2

- ☐ Comprender implementación y uso de tablas de hash
- ☐ Diseñar algoritmos usando técnicas e ideas estudiadas
- ☐ Aplicar y modificar algoritmos basados en DFS para resolver problemas en grafos

Varios objetivos pueden incluirse en cada pregunta

# Interrogación 2

## Formato de la prueba

- 2 horas de tiempo
- Pool de 4 preguntas para elegir 3
- Cada pregunta incluye un título que describe sus temas
- ¡**SOLO** se entregan 3 preguntas respondidas!

Nota de la I2: promedio de las 3 preguntas entregadas

# Interrogación 2

## Material adicional

- Pueden usar un formulario/apuntes durante la prueba
- Debe estar escrito a mano (puede ser impreso de tablet)
- Una hoja (por ambos lados)
- Sugerencia: incluyan los pseudocódigos vistos

No se aceptarán diapositivas impresas

# Objetivos de la clase

- ☐ Recordar elementos esenciales de los contenidos a evaluar
- ☐ Identificar diferencias de contenidos que aplican a un mismo escenario
- ☐ Aplicar algunos de los contenidos a ejemplos concretos
- ☐ Conocer enfoque esperado en las respuestas de algunos ejemplos

# Sumario

Introducción

**Tablas de hash**

Estrategias algorítmicas

DFS

Un ejemplo de prueba

Cierre



# Diccionarios

## Definición

Un **diccionario** es una estructura de datos con las siguientes operaciones

- **Asociar** un valor a una llave
- **Actualizar** el valor asociado a una llave
- **Obtener** el valor asociado a una llave
- En ciertos casos, **eliminar** de la estructura una asociación llave-valor

Objetivo central: búsqueda eficiente

# Diccionarios: dos enfoques

Vimos dos instancias de diccionarios

1. Árboles de búsqueda

- Binarios AVL
- 2-3
- Binarios rojo-negro

2. Tablas de Hash

# Tablas de Hash

## Aspectos esenciales de las tablas de hash

- La búsqueda se basa en el resultado de una **función de hash**
- El valor de hash se usa como índice en un arreglo (la tabla)
- Cada colisión se almacena mediante algún procedimiento
  - Direccionamiento abierto: dónde se pueda
  - Encadenamiento: usar listas ligadas de colisiones

También se podrían usar árboles para manejar colisiones.  
Beneficio: la búsqueda es más rápida que en las listas

# Tablas de Hash

## Operaciones

- Búsqueda en dos fases

1. Identificación de celda:  $\mathcal{O}(1)$  si la función de hash es adecuada
2. Búsqueda en colisiones

- Inserción también en dos fases

Note que la complejidad depende de cómo se almacenan colisiones

- Listas:  $\mathcal{O}(n)$  en el peor caso

- Árboles:  $\mathcal{O}(\log(n))$  en el peor caso, manteniendo balance

Se pueden usar otras formas de almacenamiento de colisiones:  
tablas de hash por ej

# Tablas de hash

## Orientaciones para el estudio

- ☐ Comprender funciones de hash y propiedades deseables (impacto en las colisiones)
- ☐ Comparar desempeño de técnicas de resolución de colisiones
- ☐ Uso de tablas para almacenar y consultar valores

# Sumario

Introducción

Tablas de hash

**Estrategias algorítmicas**

DFS

Un ejemplo de prueba

Cierre

# Estrategias algorítmicas

Estudiamos tres de ellas: Backtracking, greedy y programación dinámica

Aspectos esenciales

- Cada una se centra en una forma de tomar decisiones y avanzar
  - Backtracking: se puede arrepentir y cambiar su decisión
  - Codiciosos: toman la mejor decisión aparente en el momento y no se arrepienten
  - Dinámica: comparan soluciones de subproblemas, recordando
- Las tres se pueden usar para optimizar, pero greedy y dinámica son las más indicadas para esto

# Estrategias algorítmicas

## Backtracking

- Requiere considerar **satisfacción de restricciones**
- Al evaluar una decisión posible, se verifica si es factible *en este momento*
- Se hace un llamado recursivo para determinar si dicha decisión puede ser exitosa en última instancia
- Si no lo es, entonces se revierte y se toma otra
- Cuando no quedan opciones, se da por fracasado el llamado actual



# Estrategias algorítmicas

## Algoritmos codiciosos

- Exclusivos para problemas de optimización
- Se basan en una **estrategia codiciosa**
- Esta define qué elemento/decisión tomar a continuación
- Existen problemas sin una estrategia codiciosa exitosa

# Estrategias algorítmicas

## Programación dinámica

- Muy usados para problemas de optimización, pero no de forma exclusiva
- Se basan en una **ecuación de recurrencia**
- Esta compara decisiones posibles (recursivas)
- La recurrencia sugiere la forma del algoritmo diseñado
- **Se deben guardar** los resultados de subproblemas para reciclarlos

# Orden lineal

## Orientaciones para el estudio

- ☐ Comprender casos de uso de las tres estrategias
- ☐ Conocer la idea general del funcionamiento de cada estrategia
- ☐ Diseñar algoritmos usando estas estrategias

# Sumario

Introducción

Tablas de hash

Estrategias algorítmicas

**DFS**

Un ejemplo de prueba

Cierre

# Intro a algoritmos en grafos

## Algoritmos basados en DFS

- **Búsqueda en profundidad** que recorre aristas hasta agotarlas
- Permite resolver variados problemas
  - Detección de ciclos: aristas hacia atrás en grafos dirigidos
  - Orden topológico
  - Kosaraju: componentes fuertemente conectadas

# Sumario

Introducción

Tablas de hash

Estrategias algorítmicas

DFS

**Un ejemplo de prueba**

Cierre

## Ejemplo: Dinámica

### Ejercicio (I2 P2 - 2023-2)

Dado un conjunto de enteros positivos  $A = \{a_1, \dots, a_n\}$ , el problema `DivSum` consiste en responder si es posible construir dos conjuntos  $B_1, B_2$  disjuntos, no vacíos, tales que  $B_1 \cup B_2 = A$  y tales que tienen la misma suma de sus elementos. `DivSum` puede resolverse como problema de decisión, definiendo

$$p(k, S) := \begin{array}{l} 1 \text{ si y solo si es posible construir un conjunto con} \\ \text{elementos de } a_k, \dots, a_n \text{ con suma exactamente } S. \end{array}$$

y su relación recursiva

$$p(k, S) = p(k+1, S) \text{ OR } p(k+1, S - a_k)$$

donde `OR` es el operador de disyunción que evalúa primero el lado izquierdo. Si dicho lado es 1, entrega 1 sin evaluar el lado derecho.

Esto significa que es posible que ciertos llamados de  $p(k, S)$  no se realicen.

# Ejemplo: Dinámica

## Ejercicio

- (a) Identifique los casos base para  $p(k, S)$ . Defina el valor adecuado de  $p(k, S)$  en tales casos.

### Propuesta de solución.

Como nos interesa que la recursión no persista, debemos considerar:

1.  $k > n$  que representa el caso en que no quedan elementos o  $S < 0$  que representa suma negativa
2.  $S = 0$  que representa la condición de éxito (agotamos la suma buscada)

Con esto, los casos se resumen en

$$p(k, S) = \begin{cases} 0 & k > n \vee S < 0 \\ 1 & S = 0 \end{cases}$$



## Ejemplo: Dinámica

### Ejercicio

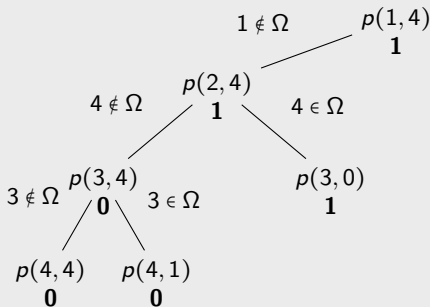
- (b) Resuelva el problema `DivSum` para  $A = \{1, 4, 3\}$  mediante la definición de  $p(k, S)$ , usando un llamado inicial adecuado. Muestre el árbol recursivo de llamados que se genera, indicando qué elementos se incluyen/descartan en cada paso y mostrando cuándo la recursión llega a un caso base.

# Ejemplo: Dinámica

## Ejercicio

### Propuesta de solución.

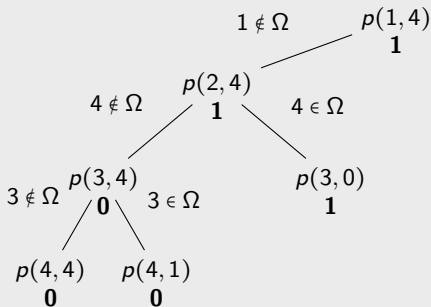
El llamado que responde al problema DivSum para  $A = \{1, 4, 3\}$  es  $p(1, 4)$ , donde el segundo argumento es la mitad de la suma de todos los elementos de  $A$ . Si  $p(1, 4) = 1$ , significa que es posible construir conjuntos con igual suma. Sea  $\Omega$  uno de los subconjuntos. A continuación mostramos los llamados recursivos que indican cómo construir  $\Omega$ .



# Ejemplo: Dinámica

## Ejercicio

### Propuesta de solución.



Observamos que todo el subárbol derecho no se genera porque se llegó a un 1 en el lado izquierdo. Del árbol, deducimos que sí el problema DivSum tiene respuesta positiva para  $A$ . Más aún, el diagrama muestra  $\Omega = \{4\}$  como uno de los conjuntos.

## Ejemplo: Dinámica

### Ejercicio

- (c) Proponga el pseudocódigo de un algoritmo recursivo que para cualquier conjunto  $A$  de enteros positivos, responda el problema `DivSum` para  $A$ .

## Ejemplo: Dinámica

### Propuesta de solución.

Se asume  $M$  un arreglo bidimensional con celdas vacías. Primero proponemos el algoritmo de programación dinámica que calcula  $p(k, S)$ .

Dynamic( $k, S$ ):

```
1   if  $k > n \vee S < 0$  :  
2       return 0  
3   if  $S = 0$  :  
4       return 1  
5   else:  
6       if  $M[k][S] \neq \emptyset$  :  
7           return  $M[k][S]$   
8       else:  
9            $M[k][S] \leftarrow \text{Dynamic}(k + 1, S) \text{ OR } \text{Dynamic}(k + 1, S - a_k)$   
10          return  $M[k][S]$ 
```

## Ejemplo: Dinámica

### Propuesta de solución.

Luego, el algoritmo que resuelve el problema DivSum es

DivSum( $A$ ):

```
1    $S \leftarrow \text{SUM}(A)$ 
2   if  $S$  es impar :
3       return 0
4   return Dynamic(1,  $S/2$ )
```

# Sumario

Introducción

Tablas de hash

Estrategias algorítmicas

DFS

Un ejemplo de prueba

**Cierre**

# Recomendaciones finales

Para los ejemplos vistos en clase

- Replicarlos comprendiendo los pasos de su resolución
- Asegurarse de poder motivar las decisiones

Pautas anteriores

- Hay hart material resuelto en el repo!
- No se aprendan pautas... seleccionen y aprovéchenlas
- Planifiquen su solución antes de verla, y luego consulten la pauta



# Objetivos de la clase

- ☐ Recordar elementos esenciales de los contenidos a evaluar
- ☐ Identificar diferencias de contenidos que aplican a un mismo escenario
- ☐ Aplicar algunos de los contenidos a ejemplos concretos
- ☐ Conocer enfoque esperado en las respuestas de algunos ejemplos