Estructuras de Datos #2

Eficiencia
Complejidad

1

Estructuras de Datos

- Son un total de diez estructuras de datos a revisar:
  - Arreglos, pilas (stacks), colas (queues), listas enlazadas (linked lists), arboles binarios (binary trees), arboles AVL (AVL trees), B* trees (Arboles B*), tablas de hashing, heaps, y grafos
  - Cada una de ellas es una forma de almacenar un grupo de datos
- Para cada una de ellas se analizará:
  - Su motivación
  - Sus operaciones
  - Su eficiencia
  - Su implementación en Java

2

Definiciones

- Una *estructura de datos* es la disposición de los datos en la memoria de un computador (o disco).
- Un *algoritmo* proporciona un conjunto de instrucciones para la manipulación de datos en las estructuras.

3

Operaciones sobre las estructuras de datos

- Crear la estructura
- Recorrer la estructura
- Buscar un elemento
- Insertar un elemento
- Eliminar un elemento
- Y otras específicas a la estructura de datos en uso

4

Dilemas de las Estructuras de Datos

- Estos son los costos que típicamente se deben considerar:
  - Acceso a un fragmento de datos
  - Búsqueda de un fragmento de datos
  - Inserción de datos en la estructura
  - Eliminación de datos de la estructura
  - La implementación de la estructura
  - Memoria (en realidad, el almacenamiento de los datos)

5

Costos generales de las Estructuras a estudiar

Estructura	Acceso	Búsqueda	Inserción	Eliminación	Impl.	Memoria
Arreglo	Muy Bajo	Alto	Med	Alto	Bajo	Bajo
Lista Enlazada	Alto	Alto	Bajo	Bajo	Med	Med
Pila	Med	Alto	Med	Med	Med	Med
Cola	Med	Alto	Med	Med	Med	Med
Árbol Bin.	Med	Bajo	Bajo	Bajo	Alto	Alto
Árbol R-N	Med	Muy Bajo	Bajo	Bajo	Muy Alto	Alto
Tabla Hash	Med	Med	Bajo	Alto	Bajo	Alto
Heap	Med	Med	Muy Bajo	Bajo	Alto	Alto
Grafo	Alto	Alto	Med	Med	Med	Med

6

Importante: No hay una "mejor" estrucutura, depende del problema a resolver

Eficiencia

- Un algoritmo es eficiente cuando logra llegar a sus objetivos planteados utilizando la menor cantidad de recursos posibles, es decir, minimizando el uso memoria, de pasos y de esfuerzo humano.
- Un algoritmo es eficaz cuando alcanza el objetivo primordial, el análisis y resolución del problema es lo prioritario.
- Puede darse el caso de que exista un algoritmo eficaz pero no eficiente, en lo posible debemos de manejar estos dos conceptos conjuntamente.

7

Eficiencia

- La eficiencia de un programa tiene dos ingredientes fundamentales: espacio y tiempo.
  - La eficiencia en espacio es una medida de la cantidad de memoria requerida por un programa. EFICIENCIA ESPACIAL
  - La eficiencia en tiempo se mide en términos de la cantidad de tiempo de ejecución del programa. EFICIENCIA TEMPORAL

8

Análisis de Algoritmos

- Es una herramienta para hacer la evaluación de un algoritmo.
- Permite establecer la calidad de un programa y compararlo con otros programas que resuelven un mismo problema.
- Suponga que existen dos programas P1 y P2 para resolver el mismo problema.
- ***¿Cuál de los dos es mejor?.***

9

Análisis de Algoritmos

Solución:

Desarrollar (Implementar) ambos programas y medir el tiempo que cada uno de ellos consume para resolver el problema. Modificar los datos de entrada, promediar al final su desempeño para establecer su comportamiento en el caso promedio.

10

Metodologías para el análisis de algoritmos: Experimentación

- El tiempo de ejecución es prioritario cuando se analiza un algoritmo.
- El tiempo de ejecución de un algoritmo depende de varios factores relativos al hardware (procesador, reloj, memoria, disco, etc) y el software (sistema operativo, lenguaje, compilador, etc.).

11

Experimentación

- Medida del tiempo de ejecución:.
  - Escribir un programa que implemente el algoritmo.
  - Ejecutar el programa con un conjunto de datos que varían en tamaño y composición (objetivo: determinar el peor caso, mejor caso y caso promedio) .
  - Usar un método o instrucción del lenguaje de programación para obtener una medida precisa del tiempo de ejecución.

12

Condiciones de la Experimentación

- Interesa hallar la dependencia del tiempo de ejecución en función del tamaño de la entrada.
- la ***experimentación*** tiene limitaciones:
  - Los experimentos se pueden hacer sobre un conjunto limitado de entradas de prueba.
  - Es necesario realizar los experimentos con el mismo hardware y software.
  - Es necesario implementar y ejecutar el algoritmo.

13

Análisis de Algoritmos

Problemas:

- Pueden existir muchos algoritmos para resolver el problema.
- Costoso y casi imposible implementar todos los programas.
- Modificación de los datos puede ser una labor sin sentido.

14

Análisis de Algoritmos

Objetivo:

"Establecer una medida de la calidad de los algoritmos, que permita compararlos sin la necesidad de implementarlos"

- Esto implica asociar a cada algoritmo una función matemática que mida su eficiencia.
- Además del tiempo de ejecución, para medir la eficiencia se debe considerar la cantidad de memoria utilizada por el programa.

15

Para tener una medida del tiempo de ejecución de un programa, se debe pensar en los factores que tienen influencia en dicho valor.

- Velocidad de procesamiento.
- El compilador utilizado (calidad del código generado).
- La estructura del algoritmo.

¿Cuáles de estos factores no son inherentes a la solución?.

16

Además de la estructura del algoritmo, se debe tener en cuenta que el numero de datos con los cuales trabaja un programa influye en su tiempo de ejecución.

Un programa de ordenamiento de un arreglo, se demora menos en ordenar 100 elementos que 500.

"El tiempo de ejecución de un algoritmo debe medirse en función del tamaño de los datos de entrada que debe procesar".

17

$T_A(n)$ Se define como el tiempo empleado por el algoritmo A en procesar una entrada de tamaño n y producir una solución al problema.

El ideal es encontrar una función matemática que describa de manera exacta $\boldsymbol{T_A(n)}$.

18

Metodologías para el análisis de algoritmos: Teórica

- Adicionalmente a la experimentación conviene disponer de un enfoque analítico que:
  - Tome en consideración todas las posibles entradas.
  - Permita evaluar la eficiencia de dos algoritmos de forma independiente del hardware y software.
  - Se pueda realizar estudiando una representación de alto nivel del algoritmo sin necesidad de implementarlo.

19

Aunque se conozca el tamaño de los datos de entrada, es imposible para muchos problemas determinar el tiempo de ejecución para cada una de las posibles entradas.

Por esta razón se debe trabajar con el tiempo utilizado por el algoritmo en el peor de los casos ya que es mucho más fácil definir este peor caso.

Con este antecedente se redefine $T_A(n)$.

$T_A(n)$ = Tiempo que se demora el algoritmo A en el peor de los casos, para encontrar una solución a un problema de tamaño n.

20

Concepto de Complejidad

- La complejidad (o costo) de un algoritmo es una medida de la cantidad de recursos (tiempo, memoria) que el algoritmo necesita. La complejidad de un algoritmo se expresa en función del tamaño (o talla) del problema.
- Tipos de análisis(1):
  - Análisis A Priori (o teórico)
  - Prueba A Posteriori (experimental o empírica)
- Tipos de análisis(2):
  - Peor caso
  - Mejor caso
  - Caso promedio

21

Notación Orden (Big-Oh)

- Provee una métrica para evaluar la eficiencia de un algoritmo
- Esto es la eficiencia asintótica del algoritmo. Se denomina "asintótica" porque analiza el comportamiento de las funciones en el límite, es decir, su tasa de crecimiento.
- La notación asintótica captura el comportamiento de la función para valores grandes de N.
- La notación O se utiliza para comparar funciones. Resulta particularmente útil cuando se quiere analizar la complejidad de un algoritmo, en otras palabras, la cantidad de tiempo que le toma a un computador ejecutar un programa.

22

Notación Asintótica

- La eficiencia de un algoritmo se mide mediante las operaciones elementales, más específicamente del número de operaciones elementales que se deben ejecutar.
- **Análisis del Peor Caso**: se define como el máximo costo (operaciones elementales) de aplicar el algoritmo a un problema de tamaño n.
- Este análisis se suele aplicar para casos extremos en los que interesa saber cuanto, como máximo, va a costar la ejecución del algoritmo.

23

Notación asintótica (terminología)

- Clases especiales de algoritmos:

  logarítmico: **O(log n)**

  lineal: **O(n)**

  cuadrático: **$O(n^2)$**

  polinómico: **$O(n^k)$, $k \geq 1$**

  exponencial: **$O(a^n)$, $n > 1$**

- "Alternativos" de Big-Oh
  - Ω (f(n)): Big Omega-- cota *inferior* asintótica
  - Θ (f(n)): Big Theta-- cota *promedio* asintótica

24

Notación Asintótica

- La ***notación asintótica*** nos permite realizar simplificaciones sustanciales aun cuando estemos interesados en medir algo más tangible que el tiempo de ejecución, tal como es el número de veces que se ejecuta una instrucción dentro del programa.

- Se denomina notación asintótica porque trata acerca del comportamiento de funciones en el límite, esto quiere decir, para valores suficientemente grandes de su parámetro. Esto hace que los valores pequeños de las entradas no sean interesantes.

- Dicho de otra manera, estamos interesados en las tasas de crecimientos en lugar de los valores concretos.

25

Notación Asintótica

- La notación (o grande) o cota superior es la encargada de dar una cota para el peor caso y determinar las acotaciones superiores lo más exactamente posible para valores crecientes de la entrada.

- Por lo tanto se puede asegurar que conociendo la cota superior, ningún tiempo empleado en resolver el problema dado será de un orden superior al de la cota. Se conoce como el orden del peor caso.

- La notación asintótica clasifica las funciones de tiempo de los algoritmos para que puedan ser comparadas.

26

Tiempo de ejecución

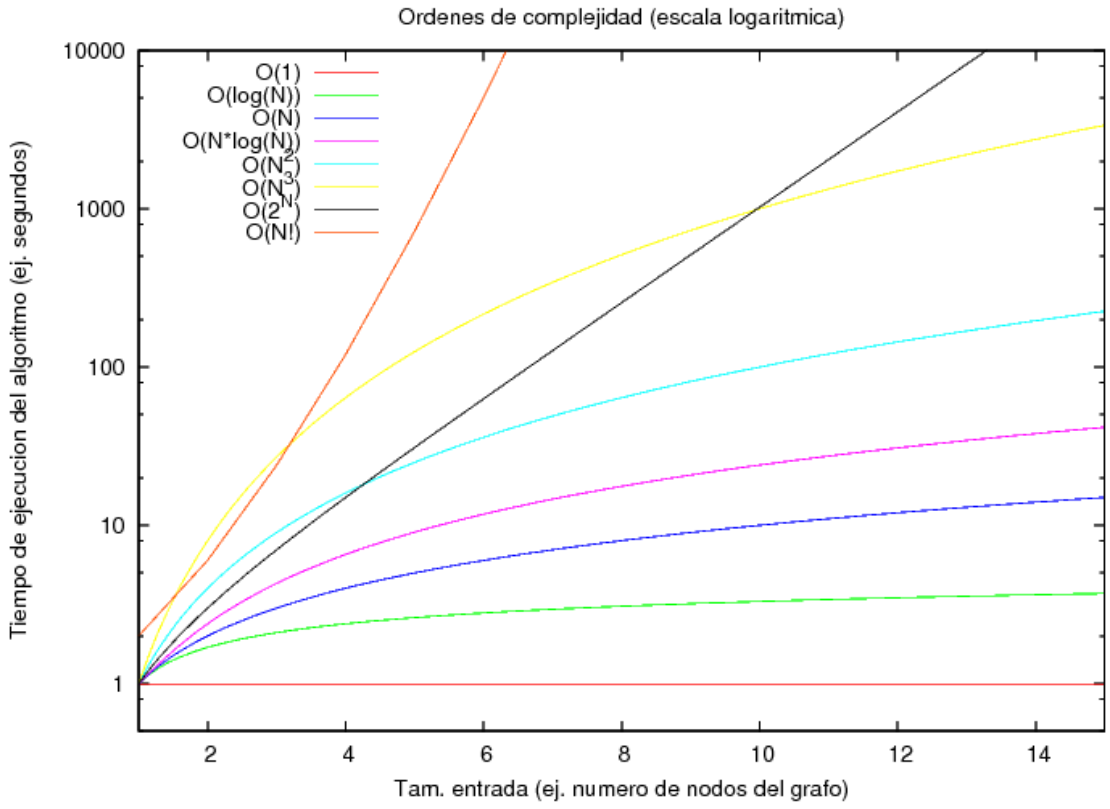
- Usar la notación Big-Oh para expresar el número de operaciones primitivas ejecutadas como función del tamaño de entrada.
- Ejemplo: decimos que un algoritmo se ejecuta en tiempo $O(n)$.
- Comparación de tiempos de ejecución asintóticos
  - un algoritmo que corre en tiempo $O(n)$ es mejor que uno que corre en tiempo $O(n^2)$
  - de forma similar, $O(\log n)$ es mejor que $O(n)$
  - jerarquía de funciones: $\log n << n << n^2 << n^3 << 2^n$
- Cuidado! Con los factores constantes muy grandes. Un algoritmo que corre en tiempo 1,000,000 n todavía es $O(n)$ pero puede ser menor eficiente para un conjunto de datos que uno que corre en tiempo $2n^2$, que es $O(n^2)$

27

Órdenes de Complejidad

- Se dice que O(f(n)) define un "orden de complejidad".
- Algunos ejemplos:
  - O(1) orden constante
  - O(log n) orden logarítmico
  - O(n) orden lineal
  - $O(n^2)$ orden cuadrático
  - $O(n^a)$ orden polinomial (a > 2)
  - $O(a^n)$ orden exponencial (a > 2)
  - O(n!) orden factorial

28


Ordenes de complejidad (escala logaritmica)
Tiempo de ejecucion del algoritmo (ej. segundos)
Tam. entrada (ej. numero de nodos del grafo)
10000
1000
100
10
1
2
4
6
8
10
12
14
O(1)
O(log(N))
O(N)
O(N*log(N))
O(N2)
O(N3)
O(2N)
O(N!)

29

Eficiencia

Estructura de Datos	Búsqueda	Inserción	Eliminación	Recorrido
Arreglos	O(N)	O(1)	O(N)	--
Arreglo Ordenado	O(logN)	O(N)	O(N)	O(N)
Lista Enlazada	O(N)	O(1)	O(N)	--
Lista Enlazada Ordenada	O(N)	O(N)	O(N)	O(N)
Árbol Binario (promedio)	O(logN)	O(logN)	O(logN)	O(N)
Árbol Binario (Peor caso)	O(N)	O(N)	O(N)	O(N)
Árbol Balanceado (peor y caso promedio)	O(logN)	O(logN)	O(logN)	O(N)
Tabla de Hash	O(1)	O(1)	O(1)	--

30

¿Por qué esto es útil?

- Útil para evaluar como un algoritmo escala con el tamaño de la entrada n. Por ejemplo:
  - O(1) escala mejor que…
  - O(log n),el cual escala mejor que…
  - O(n), el cual escala mejor que…
  - O(n log n), el cual escala mejor que…
  - $O(n^2)$, etc.
- Cada uno de estos crece sucesivamente más rápido con n.

31

En términos generales...

- Para una entrada de tamaño n y una función T (n), para calcular el valor del orden, usted toma el término principal y coloca el coeficiente.
  Ejemplos - calcular los valores de orden de los siguientes tiempos de ejecución:
  - $T(n) = 100*n^2 + n + 70000$
  - $T(n) = (n*\log n) / n$
  - $T(n) = n^3 + 754.000*n^2 + 1$
  - $T(n) = (n + 2) * (\log n)$

32

Pero, esas grandes constantes deben tener algún significado...

- $T(n) = n^3 + 754.000*n^2 + 1$

- Esa gran constante en el termino n^2, ¿tiene algún efecto?
- La respuesta es si y no.

- Si, si la entrada es _____.
- Pero para grandes valores de n, n^3 sobrepasa este término, aun cuando la constante sea grande.

33