Searching

โครงสร้างข้อมูลและอัลกอริทึมเบื้องต้น
305214 / 235012

Topic

- Sequential Search
- Searching Algorithm Analysis
- Big-O notation
- Binary Search
- Hashing

Searching

การค้นหาข้อมูล

- เป็นการเข้าไปถึงตัวข้อมูล
- เพื่อไปทำ operation กับตัวข้อมูล
- มีหลายวิธีการในการค้นหาและขึ้นอยู่กับโครงสร้างข้อมูล
  - Sequential Search
  - Binary Search
  - Hashing

Sequential Search

การค้นหาแบบลำดับ หรือ แบบเชิงเส้น

- ทำการ traverse ตามลำดับของโครงสร้างข้อมูล
- ทำการเปรียบเทียบค่าที่ต้องการค้นหากับค่าข้อมูลที่ traverse เรียงกันไป
- จำแนกได้เป็น
  - ข้อมูลเรียงลำดับ (sorted list)
  - ข้อมูลไม่เรียงลำดับ (unsorted list)

Sequential Search

ตัวอย่าง Search for “G” (unsorted list)

- Begin with “E”
- next B , next D , next A
- next G
- Founded “G”
- Finish

Sequential Search

ตัวอย่าง Search for “C” (unsorted list)

- Begin with “E”
- next B , next D , next A , …
- until finish list
- return “Not Founded”
- Finish

Sequential Search

ตัวอย่าง Search for “C” (sorted list)

- Begin with “A”
- next B
- next D , more than C → finish
- return “Not Founded”
- Finish

Search Algorithm Analysis

การวิเคราะห์ประสิทธิภาพการค้นหา

- บ่งบอกและเปรียบเทียบประสิทธิภาพการค้นหาแบบต่างๆ
- พิจารณาจำนวนรอบในการเปรียบเทียบค่าค้น
- 3 cases analysis
  - Best Case
  - Average Case
  - Worse Case

Search Algorithm Analysis

ตัวอย่าง

N → จำนวนข้อมูล

- Best case - Search for “A” → รอบ = 1
- Worse case - Search for “I” → รอบ = 6 -- N
- Average case – (1+2+…+6)/6

$$\frac{\frac{N(N+1)}{2}}{N} = \frac{(N+1)}{2}$$

Big-O Notation

Order of Magnitude

- เป็นการเปรียบเทียบค่าโดยประมาณ (Approximation) ทางคณิตศาสตร์
- ดูพจน์ที่มีผลกระทบมากที่สุดเพียงค่าเดียว
- มีประโยชน์ในการวิเคราะห์จำนวนข้อมูลมาก ๆ
- เขียนแทนด้วยสัญลักษณ์ O (เรียกว่า Big O)

O(f(N)) → หาค่า Big O ของ function f(N)

Big-O Notation

ตัวอย่าง

$f(N) = 4N^3 – 3N^2 + 2$

พจน์ที่มีผลกระทบต่อการเปลี่ยนแปลงค่า f(N)

$4N^3$, $3N^2$ และ 2

แต่ที่มีผลมากที่สุดคือ N^3 (ไม่คิดสัมประสิทธิ์)

$O(f(N)) = N^3$

Big-O Notation

ตัวอย่าง

$f(N) = 4N + 2$

$O(f(N)) = N$

$f(N) = 20$

$O(f(N)) = 1$

$f(N) = 3^N + 2^N$

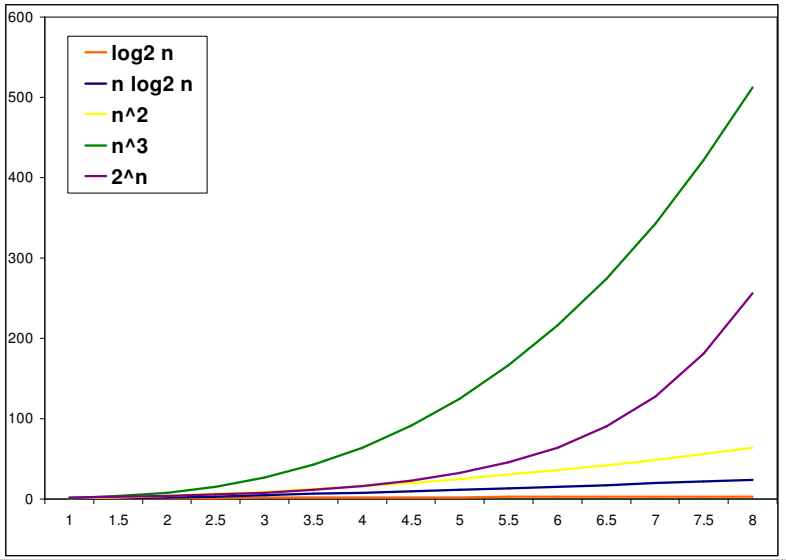
$O(f(N)) = 3^N$

Big-O Notation

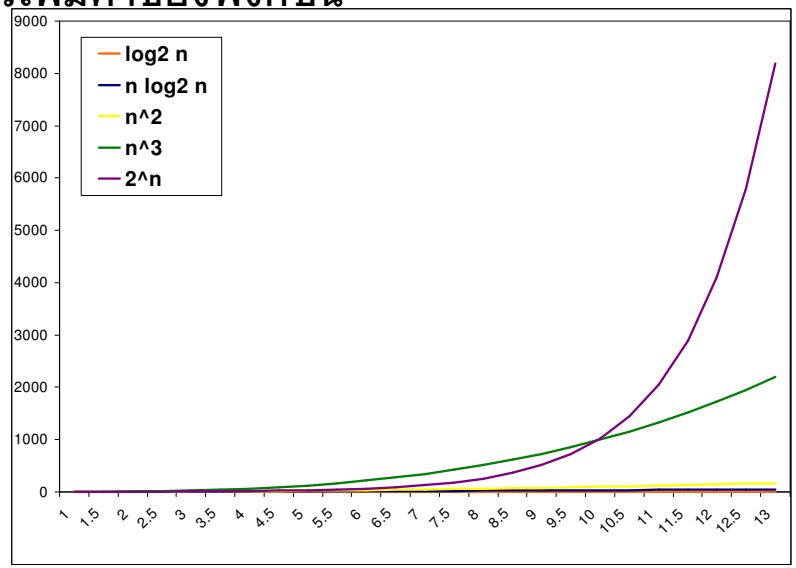
การเพิ่มค่า N

N	$Log_2 N$	$N Log_2 N$	N^2	N^3	2^N
1	0	0	1	1	2
2	1	2	4	8	4
4	2	8	16	64	16
8	3	24	64	512	256
16	4	64	256	496	65536
32	5	160	1024	32768	2147483648
64	6	384	4096	262144	1.8447E+19
128	7	896	16384	2097152	3.4028E+38
256	8	2048	65536	16777216	1.1579E+77

การเพิ่มค่าของฟังก์ชั่น



การเพิ่มค่าของฟังก์ชั่น



Big-O Notation

ลำดับการเลือก - ดูที่การเพิ่มค่าของฟังก์ชั่นที่ทำให้กระทบมากที่สุด

- $O(1)$
- $O(log_2 N)$
- $O(N \log_2 N)$
- $O(N^2)$
- $O(2^N)$

Big-O Notation

Big-O ของ sequential search

- Best Case
  - O(1)
- Worse Case
  - O(N)
- Average Case
  - O((N+1)/2) = O(N)

Binary Search

การค้นหาแบบทวิภาค

- เป็นการแบ่งข้อมูลเป็นสองส่วน
- ดูค่ากลางถ้าน้อยกว่าให้ไปค้นกลุ่มมาก และในทางตรงข้ามถ้าน้อยกว่า
- แบ่งครึ่งค้นหาไปเรื่อยๆ จนกว่าจะพบหรือหมดข้อมูล
- ทำได้กับข้อมูลแบบเรียงลำดับเท่านั้น

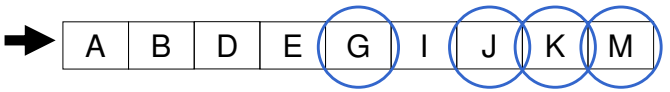
Binary Search

ตัวอย่าง Search “B”

- Begin with position (1+9)/2 = 5 → “G”
- B < G → left group
- position (1+4)/2 = 2 → “B”
- founded
- Finish

Binary Search

ตัวอย่าง Search “L”



- Begin with position (1+9)/2 = 5 → “G”
- L > G → right group
- position (6+9)/2 = 7 → “J” L > J → right group
- position (8+9)/2 = 8 → “K” L > K → right group
- “M” → not matched → Not founded
- Finish

Binary Search

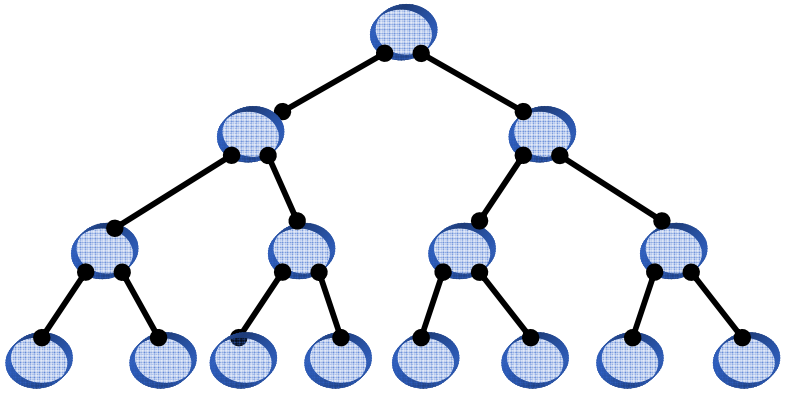
Algorithm Analysis

- หาจำนวนครั้งโดยมองว่าข้อมูลเป็น BST
- จำนวนครั้งเท่ากับระดับ (level) ของโหนด
- นำมาทำการคำนวณหาจำนวนครั้ง และ Big-O ของ

Best, Worse และ Average case

Binary Search

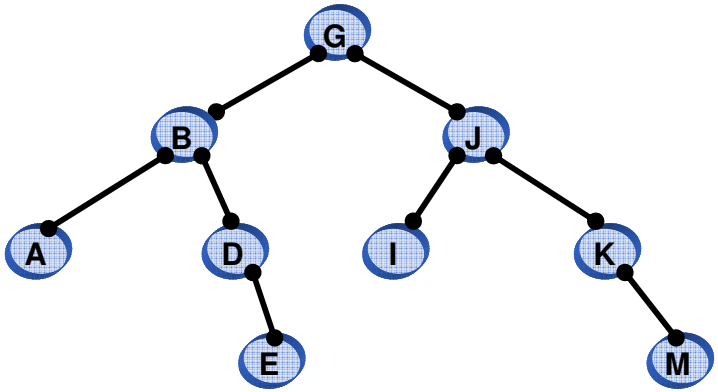
Algorithm Analysis



จำนวนโหนดของ level = 2^m , m เป็น level

Binary Search

Algorithm Analysis



Search “L” เปรียบเทียบ 4 ครั้ง m = 4

Binary Search

Worse Case

- หาจำนวนครั้งที่เปรียบเทียบมากที่สุดจากข้อมูล N ตัว
- หาได้จาก level ของ BST

จากจำนวนโหนดของ level m = 2^m

ผลรวมโหนดของต้นไม้ m level เป็น

$$2^0+2^1+2^2+ \ldots + 2^m$$

$$N = 2^0+2^1+2^2+ \ldots + 2^m$$

Binary Search

Worse Case (ต่อ)

$$N = 2^0+2^1+2^2+ \ldots + 2^m$$

$$2N = (2^0+2^1+2^2+ \ldots + 2^m) \times 2$$

$$= 2^1+2^2+ \ldots + 2^{m+1}$$

$$2N - N = 2^{m+1} - 2^0$$

$$\boxed{N = 2^{m+1} - 1}$$

Binary Search

Worse Case (หา m)

$$N = 2^{m+1} - 1$$

$$2^{m+1} = N + 1$$

$$\log_2(2^{m+1}) = \log_2(N+1)$$

$$(m + 1)\log_2 2 = \log_2(N+1)$$

$$\boxed{m = \log_2(N+1) - 1}$$

Binary Search

Worse Case (หา Big-O m)

จาก $m = \log_2(N+1) - 1$

หาค่า Big-O ของ m

$$\boxed{O(m) = \log_2(N)}$$

Binary Search

Average Case

- หาจำนวนครั้งเฉลี่ยจากข้อมูล N ตัว
- หาได้จากแต่ละ level ของ BST

จากจำนวนโหนดของ level m = 2^m

จำนวนครั้งของ level m เป็น $(2^m) \times (m+1)$

จะได้ว่า จำนวนครั้งทั้งหมด $S = (2^{m+1})m + 1$

Binary Search

Average Case (ต่อ)

จาก $2^{m+1} = N + 1$

และ $m = \log_2(N+1) - 1$

$(2^{m+1})m + 1 = (N+1)(\log_2(N+1))-N$

หาค่าเฉลี่ย (หารด้วย N)

Average = (N+1)/N(log2(N+1))-1

Binary Search

Average Case (หาค่า Big O)

O(Average) = O[(N+1)/N(log2(N+1))-1]

= $O(\log_2 N)$

Binary Search

Best Case

- หาตัวแรกแล้วเจอเลย
- ใช้จำนวนครั้งในการเปรียบเทียบ 1 ครั้ง

หาค่า Big-O

O(Best) = O(1)

Binary Search

Quiz

- หา Big-O ของ f(n) ต่อไปนี้

1. $(n+3)(n-4) + \log_2 n$
2. $3(2^n) + 5n^2 + 6n$
3. $n(n+3)(n-5) + 2(n^2)$
4. $8(2^n) + 3^n - 9(n^8)$

Hashing

การค้นหาด้วยวิธี Hashing

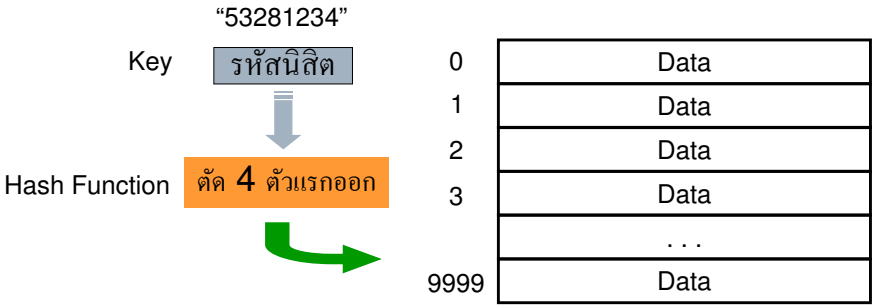
- มุ่งให้ได้ประสิทธิภาพเป็น O(1)
- ค่าที่ค้นหาสามารถสื่อถึงตำแหน่งที่เก็บข้อมูลได้
- เข้าถึงข้อมูลได้ด้วยการใช้ฟังชั่นแฮช (Hash Function)
- โดยทั่วไปเก็บข้อมูลในรูป ตารางแฮช (Hash Table)

Hashing

Hashing

ตัวอย่าง

เก็บข้อมูลคะแนนโดยใช้รหัสนิสิตของวิชานี้



การเข้าถึงทำกับตำแหน่ง array คำนวณจากรหัสนิสิตโดย Hash Function

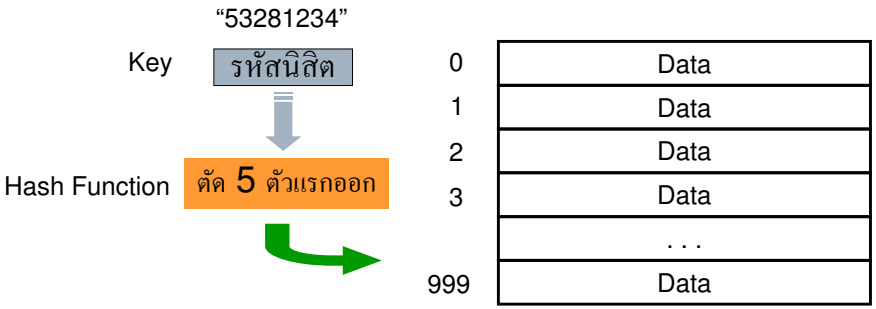
Hashing

การค้นหาด้วยวิธี Hashing

- การค้นหาใช้การหาค่าจากฟังก์ชั่นได้เพียงรอบเดียว
  - Big–O ของทุกกรณีเป็น O(1)
- ต้องมีการเตรียมที่สำหรับเก็บข้อมูลโดยที่มีบางอันไม่ได้ใช้
- ฟังชั่นแฮช (Hash Function) จะต้องให้ค่าที่ไม่ซ้ำกัน
  - ค่าไม่ซ้ำกันเลย → ต้องใช้เนื้อที่มาก
  - ยอมให้มีการซ้ำกันได้บ้าง ให้ใช้เนื้อที่น้อยลง

Hashing
ตัวอย่าง

ใช้รหัสแค่ 3 ตัวท้ายเมื่อทราบว่ารหัส 5328 มีประมาณพันคน



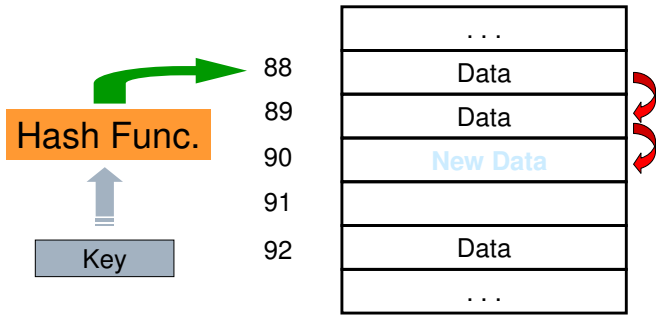
ต้องมีการแก้ไขปัญหาในกรณีที่เกิดการซ้ำกันของตำแหน่ง !!!

Hashing
Keys Collisions ปัญหาการซ้ำกัน

- เกิดจากการที่ key ให้ค่า hash ที่ซ้ำกัน
  - ทำให้การอ้างอิงตำแหน่งในตาราง hash ซ้ำกัน
- การแก้ไขปัญหา
  - Linear Probing
  - Buckets
  - Chaining

Hashing
Linear Probing

- ตรวจสอบตำแหน่งที่ว่างถัดไปจากตำแหน่งที่ได้จากฟังก์ชั่นแฮช



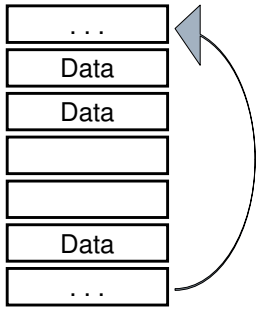
Hashing
Linear Probing

- การค้นหา
  - หาตำแหน่งผ่านฟังก์ชั่นแฮชก่อน
  - เทียบค่าถ้าไม่ใช่ให้วนหาค่าต่อไปแล้วเปรียบเทียบเรื่อยๆ
- ปัญหา
  - worse case เป็นกรณีที่ค่าแรกซ้ำ แล้วต้องทำการเปรียบเทียบ n (ขนาด hash table)
  - ค่าข้อมูลที่ถูกตำแหน่งถูกใช้ไปก่อนหน้า

Hashing

Linear Probing

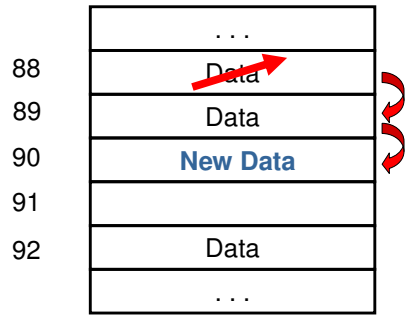
- การแทรกข้อมูลไม่ได้
  - มีกรณีที่ hash แล้วได้ค่าท้ายๆ อาจไม่เจอที่ว่างทั้งๆ ที่มีที่ว่างก่อนหน้า
  - แก้ปัญหาโดยใช้ list แบบวน



Hashing

Linear Probing

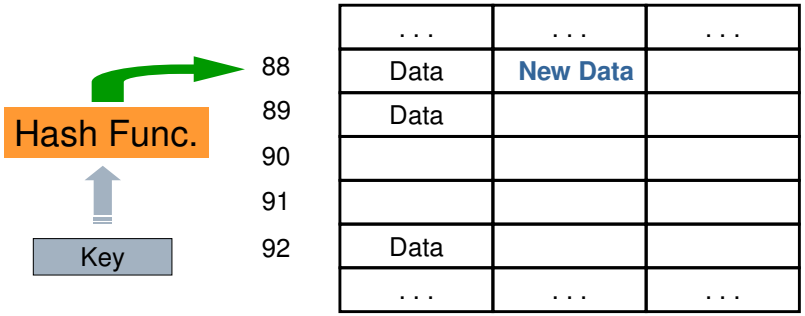
- ปัญหาการลบข้อมูล
  - ลบข้อมูลมีค่าซ้ำกันตัวแรก
  - หาข้อมูลตัวที่ตามมาไม่พบ
  - เนื่องจากตำแหน่งว่าง



Hashing

Buckets

- เตรียมที่ว่างไว้เก็บข้อมูลมากกว่า 1 ที่ในตำแหน่งเดียวกัน



เช่นการใช้ array 2 มิติช่วยในการเก็บ

Hashing

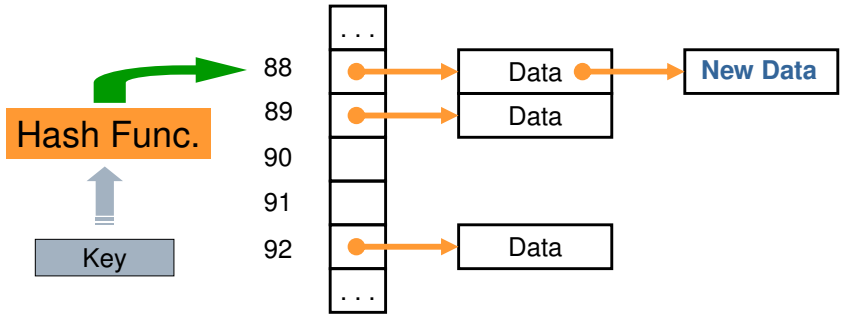
Buckets

- ข้อดี
  - มีที่เพิ่มเติมได้ในตำแหน่งเดียวกัน
  - แก้ปัญหาการลบเช่นที่เกิดใน linear probing
- ข้อด้อย
  - ต้องเตรียมเนื้อที่ไว้รอซึ่งอาจไม่ได้ใช้
  - มีข้อจำกัดเรื่องเนื้อที่ต่อตำแหน่ง (ด้อยกว่า linear probing)

Hashing

Chaining

- ใช้ linked list เข้ามาเก็บข้อมูลในแต่ละตำแหน่ง hash



แก้ปัญหาจำนวนข้อมูลที่จำกัดของแต่ละตำแหน่ง

Hashing

Chaining

- การลบข้อมูล
  - แก้ปัญหาการลบที่เกิดใน linear probing
  - เป็นการลบใน linked list ปกติ
- เนื้อที่การเก็บข้อมูล
  - ใช้เนื้อที่เท่าจำนวนข้อมูลจริง ๆ
  - ไม่ติดปัญหาเรื่องจำนวนข้อมูลต่อตำแหน่ง hash ที่จำกัด

Hashing

Algorithm Analysis

- Best case
  - O(1)
- Worse case
  - Linear Probing → O(n)
  - Buckets / Chaining ขึ้นอยู่กับขนาดของข้อมูลซ้ำ O(n)
- Average case
  - โดยประมาณ O(1)

Hashing

ข้อเสียของ Hashing

- ไม่สามารถเข้าถึงข้อมูลแบบเรียงลำดับ (Inorder) ได้
  - เนื่องจากค่าไม่ได้เรียงกันตามข้อมูล
  - ค่าที่เรียงเป็นการเรียงกันตามผลลัพธ์ฟังก์ชั่นแฮช

คำถามข้อสงสัย