



SCIENCE
SILPAKORN UNIVERSITY

ค่ายโอลิมปิก สอวน. 2

Tree Map and Unordered Map

อ.ดร.ปิญโญ แท้ประสาทสิทธิ์

ภาควิชาคอมพิวเตอร์ คณะวิทยาศาสตร์ มหาวิทยาลัย
ศิลปากร

pinyotae at gmail dot com



- แนะนำ Tree Map and Unordered Map
- การประยุกต์ใช้งาน
- เวลาที่ใช้ในการประมวลผล



- เป็นโครงสร้างข้อมูลที่ทำาการเชื่อมโยง (Mapping) ระหว่างค่าเอกลักษณ์ (Unique Key) และ ค่าใช้งาน (Value)
 - ค่าเอกลักษณ์จะมีเพียงหนึ่งเดียว ไม่มีซ้ำ (เหมือนเป็นรหัสประจำตัว)
 - ส่วนค่าใช้งานมีซ้ำได้หลายค่า
 - ค่าเอกลักษณ์กับค่าใช้งานจะผูกอยู่ด้วยกันเป็นคู่
- Tree Map เป็นสิ่งที่มีมาก่อนใน C++ STL และถูกเรียกเป็นชื่อสั้น ๆ ว่า map
 - โครงสร้างข้อมูลที่อยู่ข้างในเป็น Red-Black Tree
- Unordered Map ถูกผนวกเข้ามาใน C++ STL อย่างเป็นทางการใน C++11
 - โครงสร้างข้อมูลที่อยู่ข้างในเป็น Linear Hash
 - มี data type เป็น unordered_map (เพราะ hash ไม่มีการจัดลำดับ) แต่เพราะโครงสร้างข้อมูลภายใน บางครั้งเราจึงเรียกมันว่า Hash Map
- Map ทั้งสองประเภทมีการใช้งานที่คล้ายกัน แต่มีจุดเด่นจุดด้อยต่างกัน



- เนื่องจากโครงสร้างข้อมูลทั้งสองมีลักษณะคล้ายกัน จึงขอกล่าวถึงพื้นฐานการใช้งานไปพร้อม ๆ กัน
- จุดสำคัญคือการเชื่อมโยงค่า Key กับ Value
 - Template ของ * Map จึงต้องมีการระบุชนิดข้อมูลของ Key กับ Value
- ตัวอย่างง่าย ๆ ของการเชื่อมโยงรหัสประจำตัว (int) กับเกรดเฉลี่ย (double)

```
map<int, double> gpaTreeMap;
```

```
unordered_map<int, double> gpaHashMap;
```

- Data type ตัวแรกใน template คือ Key ซึ่งในที่นี้คือ int ส่วนตัวที่สองคือ Value ซึ่งในที่นี้คือ double

- เราสามารถอ้างอิงถึงตำแหน่งที่เก็บค่าได้ผ่าน Key
 - สามารถใช้ Key เป็นเหมือน index ของอาเรย์ได้เลย

```
gpaTreeMap[7] = 3.5;  
gpaTreeMap[2] = 3.6;  
printf("%.2f\n", gpaTreeMap[7]);  
printf("%.2f\n", gpaTreeMap[2]);
```

```
gpaHashMap[7] = 3.5;  
gpaHashMap[2] = 3.6;  
printf("%.2f\n", gpaHashMap[7]);  
printf("%.2f\n", gpaHashMap[2]);
```

- ในขณะที่มันดูง่าย แต่ที่จริงแล้วจะเข้าใจมันให้ถูกต้องถือเป็นเรื่องยาก
- เพราะเราต้องเข้าใจกลไกการทำงานของมันอยู่บ้าง ดังนี้
 - การใช้ [Key] นั้นมันจะเอา Key ไปตรวจว่ามีการผูกค่า Key นี้ไปแล้วหรือยัง
 - ถ้ายังไม่มีมีการผูกค่า Key การใช้ตัวดำเนินการ [] จะทำให้มีการสร้างช่องเก็บข้อมูลเพิ่มขึ้นมาทันที นี่เป็นเหตุที่ทำให้เราใส่ค่าใหม่เข้าไปได้
 - ถ้าเราคิดที่จะแค่ทำการตรวจว่าเราผูกค่า Key ไปแล้วหรือยัง
→ ใช้คำสั่ง `find(Key)` แทน
 - เพราะแค่เรียก [Key] มันจะผูก Key กับค่า Default ของ Value ให้ทันที
 - มันเป็นผลข้างเคียงของการใช้ตัวดำเนินการ [] ที่ต้องเข้าใจและระวัง

- สังเกตดูให้ดีว่าในตอนแรกที่เราทดสอบด้วย find เราจะไม่เจอการผูก Key
- เราใช้ .end() เพื่อดูว่าผลการ find ถ้าผลเป็น .end แสดงว่าไม่เคยผูก Key

```
if(gpaTreeMap.find(99) == gpaTreeMap.end())  
    printf("Key 99 is not mapped yet.\n");
```

เราพยายามพิมพ์ค่าออกมา ทั้งที่ไม่เคยผูก Key มาก่อน
ในลักษณะนี้เราจะได้ค่า default ของ double ซึ่งก็คือ 0
และการผูก Key 99 กับเลข 0 จะเกิดขึ้นทันทีด้วย!

```
printf("%.2f\n", gpaTreeMap[99]);  
if(gpaTreeMap.find(99) != gpaTreeMap.end())  
    printf("Key 99 is already mapped.\n");
```

- พฤติกรรมนี้เกิดขึ้นกับ unordered_map เหมือนกัน

ที่จริงแล้ว * Map เก็บ pair<Key, Value>



- ดังนั้นเวลาที่เรา iterate ไปบน map ที่เราใช้เป็นตัวอย่าง เราจะได้ pair<int, double> ออกมา
- เราสามารถวนเรียกดูรายการทั้งหมดที่ map เก็บไว้ออกมาได้ผ่าน iterator
- เราอย่ามัวไปกังวลกับ data type ของ iterator ที่มันซับซ้อน เช่นในที่นี้คือ std::map<int, double>::iterator แต่เราใช้ auto& ก็ได้เลย

```
for(auto& keyValue : gpaTreeMap)
    printf("(%d, %.2f)\n",
           keyValue.first, keyValue.second);
```

- อย่างไรก็ตาม ลำดับค่าที่จะออกมานี้จะมีความแตกต่างกันระหว่าง Tree Map กับ Hash Map
 - Tree Map จะพิมพ์ออกมาจากค่า Key น้อยไปมาก ส่วน Hash Map จะไม่มีลำดับที่เราคาดการณ์ได้ง่าย ๆ (เพราะมันเป็น unordered)



- ลบออกได้ด้วยคำสั่ง `erase`
 - ถ้ามี `iterator` ของช่องที่จะลบอยู่แล้วก็ใช้ได้เลย เช่น `gpaTreeMap.erase(it);` ซึ่งสำหรับ `Tree Map` วิธีนี้จะเร็วมาก
 - ถ้ารู้ `Key` ก็ลบด้วย `Key` ได้ เช่น `gpaTreeMap.erase(99);`
 - ถ้ารู้พิสัย `range` ของ `iterator` ที่เกี่ยวข้องกับค่าที่จะลบก็ใช้ได้ เช่น `gpaTreeMap.erase(gpaTreeMap.find(7), gpaTreeMap.end());`
- การลบคือเป็นสิ่งที่ซ้ำใน `unordered_map`
- ถ้าจะลบ `Value` ที่ต้องการ จะทำไม่ได้โดยตรง แต่เราอาจจะต้อง `iterate` ไปบน `Map` แล้วค่อยดูว่ามันมีค่า `value` ที่ต้องการหรือไม่
 - ถ้าใช่จึงทำการลบผ่าน `iterator` นั้น

- ขึ้นอยู่กับหลายปัจจัย แต่โดยมากถ้าเราต้องค้นหาค่าบ่อย ๆ เมื่อเทียบกับปริมาณการใส่ Key ตัวใหม่เข้าไป
 - unordered_map มีแนวโน้มที่จะเร็วกว่า
 - ทั้งนี้ขึ้นอยู่กับว่าอัตราการแย่งและชนกันของช่อง hash ต้องน้อยด้วย
- แต่ต่อให้ unordered_map จะเร็วกว่า
 - เรามักต้องเตรียมพื้นที่หน่วยความจำไว้ให้มาก ยกเว้นเราจะออกแบบการ hash มาเป็นอย่างดี
 - มันไม่มีความสามารถพหุหาค่าที่น้อยที่สุด ทำให้ประโยชน์ใช้สอยลดลงไป
 - ถ้ามีการลบ Key ออกบ่อย ๆ unordered_map จะช้ามาก

- ลองทำโจทย์ UniqueCount เพื่อนับว่ามีเลขทั้งหมดกี่ค่าที่แตกต่างกัน
 - เช่น ถ้ามีเลข 7 8 5 7 5 3 7 เราก็จะตอบว่า 4 เพราะมีเลขที่แตกต่างกัน 4 ตัวคือ 3, 5, 7, 8
 - ข้อนี้ให้หน่วยความจำมาก ถ้าจะให้ Hash Map ลองประกาศเป็น `unordered_map<int, int> numMap(50000000);`
 - ลองเปลี่ยนขนาดเริ่มต้นของ Hash Map ให้น้อยลงแล้วดูว่าเกิด time out หรือไม่
 - ลองทำโจทย์ CountVote โดยใช้ * Map ทั้งสองประเภท แทนที่จะใช้อาเรย์ธรรมดา
 - ทำโจทย์ PhoneVote แล้วดูว่าระหว่าง Tree Map กับ Hash Map อันไหนจะประสบความสำเร็จ
 - สำหรับคนที่ต้องการความท้าทายลองประยุกต์ใช้ * Map กับโจทย์โอลิมปิก ระดับประเทศ Wizard (ต้องใช้ไหวพริบเพื่อดูให้ออกกว่าจะใช้ Map อย่างไร)
- ** Key ของ Map จะเป็น `std::pair` ก็ได้ หรือเป็นคลาสอื่น ๆ ก็ได้