Lab 10 : Hashing

วัตถุประสงค์ ฝึกเรื่อง hashing : hash table, hashing function, collision policy : separate chaining & open addressing-linear probing, rehashing by resizing, load factor, collission

ทฤษฎี hashing เป็นการเก็บข้อมูลในตาราง เพื่อให้การสืบค้นข้อมูลไม่ต้องไล่หาข้อมูลไปทีละตัว ดังนั้นในการเก็บข้อมูลจึง map key ของข้อมูลเข้ากับ index ของ array (ตาราง) เพื่อนำข้อมูลไปเก็บที่ index นั้น เมื่อจะหาข้อมูล ก็ map key ของข้อมูลกับ index ของ array ด้วยวิธีเดิมเพื่อหาว่าเก็บข้อมูลไว้ที่ index ใด แล้วไปหาข้อมูลนั้นได้ทันที ไม่ต้องไล่หาดังวิธีอื่นที่เคยเรียนมาในเรื่อง searching ฟังก์ชั่นที่ใช้ map key ของข้อมูลเข้ากับ index ของ array เรียกว่า **hashing function** เช่นถ้า key คือ รหัสนักศึกษา ถ้าทุกคน id นำหน้าด้วย 54011 เราอาจใช้ hasing function เป็น

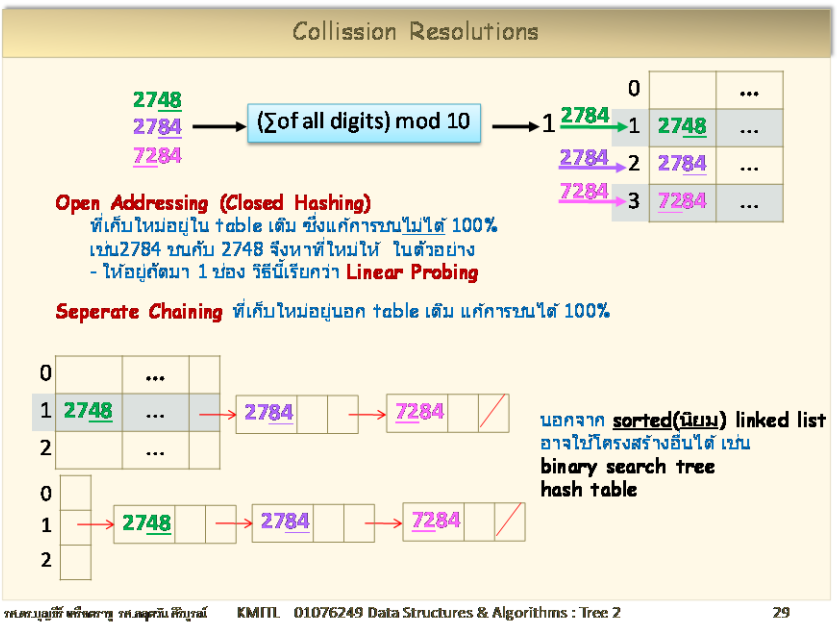
hf(key) = (key-54011000) mod ARRAY_SIZE

hf(54 01 1037) = 37

hf(54 01 1576) = 576

การ mod ด้วย ARRAY_SIZE ทำให้ได้ index ที่อยู่ใน range ของ array algorithm ที่ hash key ให้เป็น index เรียก **hasing algorithm** array (ตาราง) เรียก **hashing table** ในการ hash ข้อมูลอาจได้ index ที่ซ้ำกัน เรียกว่าเกิดการชนกัน collision หาก index นั้นมีข้อมูลอื่นเก็บอยู่แล้ว จึงต้องหา index ใหม่ให้ข้อมูลที่ไปชนกับเขา เรียกว่าการแก้ collision (collision resolution) ซึ่งแบ่งเป็น 2 วิธีใหญ่ๆ คือ

1. **Separate Chaining** : สร้าง linked list ณ index นั้น เพื่อเก็บ data ที่ hash แล้วได้ index นี้ทั้งหมด ซึ่งอาจเป็น linear list หรือ binary search tree ก็ได้ วิธีนี้แก้ collision ได้ 100%
2. **Open Addressing (close hashing)** : หา index ใหม่ในตาราง hash เดิม เรียกว่า **rehashing** คือ hash อีกครั้งให้ได้ index ใหม่ ซึ่งมีหลายวิธีเช่น **Linear Probing** (ลองช่องถัดไป) และ **Quadratic Probing** เป็นต้น วิธีนี้ไม่สามารถแก้ collision ได้ 100% อาจเกิดการชนขึ้นอีก



การชนกันจะเกิดขึ้นบ่อยถ้าตารางแน่น ความแน่นของตาราง **Load factor** λ คือจำนวน element ใน hash table / table size ในวิธี Open Addressing แก้โดยขยายขนาดของตาราง แล้วนำข้อมูลมา hash อีกครั้งเพื่อใส่ตารางใหม่ ดังนั้นจึงเรียกว่า **rehash** เหมือนกัน การ resize ตารางอาจดูจากค่า λ ที่มากเกินต้องการ หรือ วัดจากจำนวน probe (จำนวนครั้งที่หยั่งลงไปในแต่ละช่องของตาราง) ที่มากเกินต้องการ แล้วแต่นโยบาย

การทดลอง เขียนโปรแกรม spell checker เพื่อค้นหาคำศัพท์ว่าถูกต้องหรือไม่เช่น

```
Enter your word : bird
“bird” is correctly spelled

Enter your word : wird
“wird” is not in the dictionary
Possible corrections are : bird gird ward word wild wind wire wiry (optional ไม่ทำก็ได้)
```

1. ในข้อต่อไปนี้มีส่วนของ code ให้มาเพื่อให้เริ่มต้นได้ นศ.ไม่จำเป็นต้องดู code หากทำได้เอง หาก นศ.ลืมเรื่องใด เช่น ลืมเรื่อง file ก็สามารถ search หาและเขียนเองได้
2. สร้าง dictionary เป็น hash table ที่เก็บข้อมูลคำศัพท์จาก input text file (มีให้ 2 files dic1 มีคำศัพท์ 341 คำ และ dic2 มีคำศัพท์ 34,829 คำ ให้ลองทำโดยใช้ file เล็กก่อน รูปแบบ file เป็นดังแสดง

```
a about above absolutely
acceptable add adjacent after
algorithm all along also an
analyses and any anyone are
argument as assignment assume
at available be been below
bird body but by c can cannot
capitalization ...
```

```
FILE *fp;
char *str = (char*)malloc(1024);
//open file dic1
fp = fopen("d:/dic1.txt","r");
if (fp == NULL){
     printf("Cannot open file.\n");
     return 0;
}
while (fscanf(fp,"%s",str) != EOF){//read each word from dic1
     //your code to insert str in your dictionary
}
```

3. hash table จะเก็บข้อมูลคำศัพท์ซึ่งยาวไม่เท่ากัน ให้ใช้ pointer และ allocate memory มาเท่ากับความยาวของแต่ละคำ ส่วนขนาดของ table อาจมีการ resize จึงต้องกำหนดเป็น pointer to array ไม่ใช่กำหนดเป็น array ที่มีขนาดคงที่

```
int TABLE_SIZE = 11; //start table size
char **dic;//dictionary and newDic for resizing the table
dic = (char**)malloc(TABLE_SIZE * sizeof(char*));
for(int i = 0; i<TABLE_SIZE; i++)
     dic[i] = NULL; //initialize all to NULL before inserting
```

4. ขนาดของ hash table ให้เริ่มที่ prime ตัวแรกที่ 11 สำหรับ dic1 เพื่อทดสอบความถูกต้องได้ง่าย เมื่อถูกต้องดีแล้ว เมื่อใช้ file dic2 อาจเริ่มต้นที่ขนาด >= 20% ของจำนวน data ทั้งหมด ใช้ isPrime(i) เพื่อหาว่า i เป็น prime หรือไม่ เพื่อวน loop หา prime ตัวแรกที่ใหญ่กว่าค่า int ที่ต้องการ

```
#include <cmath>
bool isPrime(int xx) {//test wheather xx is prime
     int max;

     max = (int)sqrt((double)xx) + 1;
     for(int ii = 2; ii <= max; ii++)
          if(xx % ii == 0)
               return false;
          return true;
}
```

5. การจัดการการชน collission เลือกทำได้ 2 วิธีคือ

5.1. Separate Chaining โดยนศ.อาจ #include linear linked list หรือ binary search tree ที่เคยทำมาแล้วในการทดลองครั้งก่อน นำมาใช้เพื่อประหยัดเวลาในการเขียน ทั้งยังทำให้ได้ลอง #include module ที่ทำขึ้นเองในกรณีนี้จะไม่ต้องแก้การชนกัน เพราะเก็บข้อมูลไว้ใน linked list และไม่ต้องขยายขนาดของ table ก็ได้ แต่ใช้ data structure ที่สองแก้ปัญหา เช่นใช้ binary search tree เพราะการ rehash โดย resize table แพง เพราะต้องนำข้อมูลจาก data structure ที่สองออกมาและสร้างใหม่ กรณีนี้ให้ใช้ table ขนาด 11

5.2. Open Addressing วิธี Linear Probing คือ rehash โดยลองในช่องถัดไป วิธีนี้เมื่อ table แน่น ต้อง resize table แล้ว rehash data มาเข้า table ใหม่ โดยใช้ prime ตัวแรกที่ใหญ่ขึ้นเป็น 2 เท่าของขนาดเดิม นโยบายคือจะ resize เมื่อ load factor > 0.5 เมื่อสร้าง table ใหม่แล้ว อย่าลืม free memory ใน table เดิมด้วย

6. Hash function ใช้ตาม text คือ Horner's rule : Polynomial of 32 ดังนี้

```
int hash(const char *key, int TABLE_SIZE){
      unsigned int hashVal = 0;
      while(*key != '\0')
            hashVal = (hashVal<<5) + *key++;
      return hashVal % TABLE_SIZE;
}
```

7. การเก็บสถิติ

6.1 จำนวนครั้งที่ขยาย table 6.2 load factor 6.3 จำนวนครั้งที่เกิดการชนกันทั้งหมด

6.4 ความยาวของ collision chain ที่ยาวที่สุด (ในกรณีใช้ linear probing การชนต่อเนื่องกันในการเก็บข้อมูลแต่ละตัว เอาตัวที่ชนต่อเนื่องยาวที่สุด)

หมายเหตุ สถิติลำดับที่ 6.2-6.4 ต้องมีการเก็บใหม่ทุกครั้งที่มีการขยาย table

เมื่อสร้าง dictionary เสร็จเรียบร้อยแล้ว ให้แสดงสถิติในรูปแบบดังนี้

```
Collission resolution : [linear probing or chaining]
Total words : x
Table size : n (in case of linear probing)
n expansions, load factor f, n collisions, longest chain n
```

8. ในการ search dictionary

8.1. เวลา search จะต้อง rehash ไปจนเมื่อใด? จึงจะทราบว่าเป็น unsuccessful search นศ.ต้องกำหนด algorithm ในการหยุด search ด้วย

8.2. ฟังก์ชั่น `strcmp(a,b)` return 0 เมื่อ string a และ b มีค่าเท่ากัน ต้อง `#include <string.h>`