การแก้ปัญหาด้วยการเขียนโปรแกรมภาษาซี

อ. สุเมธ

ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

ม.มหิดล

หลักการแก้ปัญหา

1. Direct method (การแก้ปัญหาตรงๆ)
2. Divide and conquer (การย่อยปัญหาออกเป็นปัญหาที่เล็กลง แก้ปัญหาที่เล็กลงนั้น และประกอบกันเป็นคำตอบของปัญหาใหญ่)
3. Greedy (การเลือกทางเลือกที่ให้ผลตอบแทนที่มากที่สุด ในการทำซ้ำแต่ละครั้ง)
4. Dynamic programming (แก้ปัญหาย่อยขนาดที่เล็กที่สุด ใช้คำตอบที่เล็กที่สุดนั้นในการแก้ปัญหาขนาดที่ใหญ่ขึ้น ทำซ้ำไปจนกว่าจะถึงปัญหาที่ต้องการจะแก้)
5. ถ้าแก้โจทย์ไม่ได้ ลองแก้โจทย์ที่ง่ายกว่าก่อน เช่น เอาเงื่อนไขบางอันออก หรือขนาดของปัญหาเล็กกว่า

ขั้นตอนการแก้ปัญหา

1. ทำความเข้าใจโจทย์
2. พยายามตีโจทย์ให้อยู่ในรูปแบบโครงสร้างข้อมูล (data structure) เช่น อาเรย์
3. ใช้ขั้นตอนวิธี (algorithm) ที่เหมาะสมสำหรับโครงสร้างข้อมูลในข้อ 2

ข้อ 1 – หาว่าจำนวนใดๆ เป็นจำนวนเฉพาะหรือไม่

หลักการแก้ปัญหา

1. **รับจำนวน n เข้ามาในโปรแกรม**
2. **นำ n หารจำนวนตั้งแต่ 2 ไปจนถึง n/2 ถ้ามีจำนวนใด สามารถหาร n ได้ลงตัว n ไม่ใช่จำนวนเฉพาะ**
3. **ถ้าหารจนถึง n/2 แล้วยังไม่มีจำนวนใดหาร n ได้ลงตัว n เป็นจำนวนเฉพาะ**

โครงร่างโค้ดสำหรับข้อ 1

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

bool check_prime(int);

int main(){
    // main function
}

bool check_prime(int n){
    bool prime = true;
    for (                    ){
        if (     ){
            prime = false;
            break;
        }
    }
    return prime;
}
```

ข้อ 2 – หาตัวประกอบที่เป็นจำนวนเฉพาะ ของจำนวนใดๆ

- ตัวประกอบที่เป็นจำนวนเฉพาะของ n คือจำนวนที่หาร n ลงตัว และตัวมันเองเป็นจำนวนเฉพาะ ตัวอย่างเช่น
- 100 = 2x2x5x5
- 1054 = 2x17x31

ข้อ 2 – หาตัวประกอบที่เป็นจำนวนเฉพาะ ของจำนวนใดๆ

หลักการแก้ปัญหา

1. **รับจำนวน n**
2. **หาจำนวนเฉพาะทั้งหมดตั้งแต่ 2 จนถึง $\sqrt{n}$ ไปเก็บไว้ในอาเรย์ (สมมุติว่าชื่อ p)**
3. **ไล่หาร n ด้วยจำนวนใน p, ถ้าเจอตัวที่หารลงตัว นำค่านั้นไปเก็บไว้ในอาเรย์ที่ใช้เก็บตัวประกอบ (สมมุติว่าชื่อ f) และนำผลหารไปเขียนทับใน n**
4. **ทำซ้ำจนว่า n จะเหลือ 1**

ข้อ 2 – หาตัวประกอบที่เป็นจำนวนเฉพาะ ของจำนวนใดๆ

- ตัวอย่าง n=100
- p = {2,3,5,7} f = {}
- 2 หาร 100 ลงตัว นำตัวหารไปเก็บใน f และเขียนทับค่า n ด้วยผลหาร
  - f={2} n=50
- 2 หาร 50 ลงตัว นำตัวหารไปเก็บใน f และเขียนทับค่า n ด้วยผลหาร
  - f={2,2} n=25
- 5 หาร 25 ลงตัว นำตัวหารไปเก็บใน f และเขียนทับค่า n ด้วยผลหาร
  - F={2,2,5} n=5
- 5 หาร 5 ลงตัว นำตัวหารไปเก็บใน f และเขียนทับค่า n ด้วยผลหาร
  - f={2,2,5,5} n=1 จบการทำงาน

โครงร่างโค้ดสำหรับข้อ 2

```c
#define PRIME_SIZE 1000
#define FACTOR_SIZE 100

bool check_prime(int);

int main()
{
    int n;
    int primes[PRIME_SIZE];
    int factors[FACTOR_SIZE];
    int num_factor = 0, num_prime = 0;
    printf("Enter number: ");
    scanf("%d", &n);

    // build prime list up to sqrt(n)
    for (                              ){

    }

    /*
    algorithm
    1. build a list of primes up to sqrt(n)
    2. loop through the list of primes and check divisibility, if
       divisible, write that prime to the list of prime factors,
       divide n by the prime
    3. repeat until n becomes 1
    */

    // check through list of prime
    while(n > 1){

    }

    return 0;
}
```

ข้อ 3 – ปริ้นเป็นรูปแบบ

- แสดงผลภูเขาสองลูก เมื่อรับค่า n ที่เป็นความสูงของภูเขา ตามตัวอย่าง

```
Enter number: 4
___*_______*___
__***_____***__
_*****___*****_
*******_*******
```

```
Enter number: 5
____*_________*____
___***_______***___
__*****_____*****__
_*******___*******_
*********_*********
```

ข้อ 3 – ปริ้นเป็นรูปแบบ

หลักการคิด

- พยายามหารูปแบบ ของจำนวน - และ * ในแต่ละบรรทัด
- ใช้ for loop สองชั้น ชั้นนอกนับบรรทัด ชั้นในปริ้น - และ * ตามรูปแบบ

```
Enter number: 4
___*_______*___
__***_____***__
_*****___*****_
*******_*******
```

ข้อ 4 – หาค่าน้อยสุดและมากสุด ในอาเรย์

การหาจำนวนน้อยที่สุด (และตำแหน่งของค่านั้น) หลักการแก้ปัญหา

1. ประกาศค่าน้อยสุด เป็นค่าในอาเรย์ตำแหน่งแรก เช่น min=a[0]
2. ประกาศตำแหน่งของค่าน้อยที่สุด เป็น 0 เช่น minPos = 0
3. ไล่ดูทุกตำแหน่งในอาเรย์ ถ้าเจอค่าที่น้อยกว่า min ให้เขียนทับ min ด้วยค่านั้น และเขียนทับ minPos ด้วยตำแหน่งของค่านั้น
4. ทำซ้ำจนกว่าจะสุดอาเรย์

สำหรับการหาจำนวนมากที่สุด แค่เปลี่ยนเครื่องหมาย < เป็น >

หมายเหตุ การส่งอาเรย์ไปฟังชั่น ให้ประกาศตัวแปรที่รับค่าเข้าฟังชั่นเป็น (ตัวอย่างเช่น)

int *var_name

โคร่งร่างโค้ดสำหรับข้อ 3

```c
#include <stdio.h>
#include <stdlib.h>
#define SIZE 10

int find_min(int*);
int find_max(int*);

int main()
{
    int a[SIZE] = {45,67,12,3,98,47,15,97,64,32};

    printf("Minimum is at position %d, maximum is
           at position %d", find_min(a), find_max(a));
    return 0;
}

int find_min(int *a){
    int min = a[0];
    int pos = 0;
    for (int i = 0; i < SIZE; i++){
        if (a[i] < min){
            min = a[i];
            pos = i;
        }
    }
    return pos;
}
```

ข้อ 5 – การลบค่าซ้ำออกจากอาเรย์

- ลบค่าซ้ำออกจากอาเรย์ ตัวอย่างเช่น

  a = {45,45,12,3,3,47,15,97,3,12}
  b = remove_duplicates(a) = {45,12,3,47,15,97}

- หลักการคิด
  1. ประกาศอาเรย์ b โดยไม่ต้องใส่ค่าเริ่มต้น
  2. ไล่ดูอาเรย์ a ที่ละช่อง และทำการค้นหาค่าในช่องนั้น ในอาเรย์ b
  3. ถ้าหาพบ แสดงว่าเป็นค่าซ้ำ ไปยังช่องต่อไปของ a
  4. ถ้าหาไม่พบ แสดงว่าเป็นค่าใหม่ เพิ่มค่านั้นเข้าไปใน b ก่อนจะไปยังช่องต่อไปของ a

การส่งอาเรย์ กลับมาจากฟังชั่น

- ฟังชั่น สามารถส่งค่ากลับมาเป็น ตัวแปรเดี่ยวๆเท่านั้น ไม่สามารถส่งอาเรย์กลับได้
- ถ้าต้องการส่งค่ากลับมาเป็นอาเรย์ ต้องส่งอาเรย์ที่จะใช้เก็บค่าที่ส่งกลับมา ไปที่ฟังชั่นที่ประกาศเป็นชนิด void และเขียนค่าใส่ในฟังชั่นนั้นเลย ดังตัวอย่าง

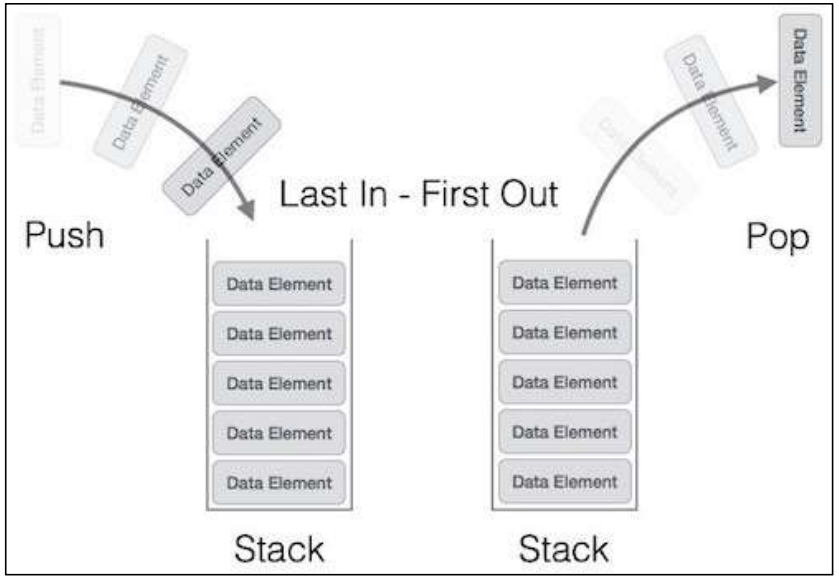
```
1 #define SIZE 10
2
3 void copy(int*, int*);
4
5 int main()
6 {
7     int a[SIZE] = {45,45,12,3,3,47,15,97,3,12};
8     int b[SIZE];
9
10    copy(a,b);
11    // now array a = array b
12
13    return 0;
14 }
15
16 void copy(int *a, int *b){
17        for (int i = 0; i < SIZE; i++)
18               b[i] = a[i];
19 }
```

โครงร่างโค้ดสำหรับข้อ 4

```c
#include <stdio.h>
#include <stdlib.h>
#define SIZE 10

void remove_duplicates(int*, int*);
int search(int*, int);
int items_b = 0;

int main()
{
    int a[SIZE] = {45,45,12,3,3,47,15,97,3,12};
    int b[SIZE];

    // print out array b

    remove_duplicates(a, b);
    printf("\n");

    // print out array b

    return 0;
}

int search(int *x, int key){
    // find position of value "key" in
    // array x, return -1 if cannot find
}

void remove_duplicates(int *a, int *b){
    // copy only unique items from a to b
}
```

โครงสร้างข้อมูล (data structure) – stack

- Stack คือโครงสร้างข้อมูลประเภท LIFO (last in first out)
- การใส่ข้อมูลใน stack เรียกว่า push ส่วนการนำข้อมูลออก เรียกว่า pop



Stack โดยใช้ array (และไม่มี object)

```c
#include <stdio.h>
#include <stdlib.h>
#define SIZE 1000

int TOP = 0;

void push(char*, char);
char pop(char*);

int main()
{
    char myStack[SIZE];
    push(myStack, 'h');
    push(myStack, 'e');
    push(myStack, 'l');
    push(myStack, 'l');
    push(myStack, 'o');

    printf("%c ", pop(myStack));
    printf("%c ", pop(myStack));
    printf("%c ", pop(myStack));
    printf("%c ", pop(myStack));
    printf("%c ", pop(myStack));

    return 0;
}

void push(char *s, char c){
    if (TOP == SIZE-1)
        printf("Stack is full, cannot push");
    else
        s[TOP++] = c;
}

char pop(char *s){
    if (TOP == 0){
        printf("Cannot pop from empty stack");
        return ' ';
    }
    return s[--TOP];
}
```

Global variable

ข้อควรระวังในการ push และ pop
- push ใส่ stack ที่เต็มแล้ว
- pop ออกจาก stack ที่ว่าง

ข้อ 6 – หาว่าวงเล็บครบคู่หรือไม่ (balance parentheses)

- ในการเขียนโปรแกรม จะพบว่ามีการใช้วงเล็บสามชนิด ({[ที่ต้องครบคู่กัน ไม่เช่นนั้นจะเกิด syntax error
- [(]) ไม่ครบคู่ (not balanced)
- (((ไม่ครบคู่ (not balanced)
- [()]{}{[()()]()} ครบคู่ (balanced)

ข้อ 6 – หาว่าวงเล็บครบคู่หรือไม่ (balance parentheses)

หลักการแก้ปัญหา

1. รับ input มาเป็น string (อาเรย์ที่เป็นชนิด char)
2. ไล่ดูอาเรย์ที่ละช่อง
   - ถ้าเจอวงเล็บซ้าย ([{ ให้ push ใส่ stack
   - ถ้าเจอวงเล็บขวา)]} ให้ pop วงเล็กซ้ายออกมาจาก stack 1 ตัวและดูว่าเข้าคู่กันหรือไม่
   - ถ้าเข้าคู่ ไปตัวถัดไป
   - ถ้าไม่เข้าคู่ ให้ break ออกจาก loop ผลลัพท์คือ not balanced
3. ถ้าจบ loop แล้ว ยังเหลือวงเล็บซ้ายใน stack แสดงว่า not balance

การรับและแสดงผลข้อมูลประเภท string

```c
#include <stdio.h>

int main(){
        char string[100];
        printf("Enter string: ");
        scanf("%s", &string);
        printf("%sString is: ", string);

        // find and print lenght of string
        int i = 0;
        while(string[i] != '\0')
                i++;
        printf("\nLength of string is %d", i);
        return 0;
}
```

หมายเหตุ ตัวอักษรตัวสุดท้ายของ string จะเป็น ‘\0’ (เลขศูนย์ ไม่ใช่ตัวโอ) เสมอ ซึ่งจะไม่มีการการแสดงผล แต่สามารถใช้ตรวจสอบได้ว่า ตอนนี้ string มีตัวอักษรกี่ตัว

โครงร่างโค้ดสำหรับข้อ 5

```c
int TOP = 0;

int main()
{
    char myInput[INPUT_SIZE];
    printf("Enter parentheses for check: ");
    scanf("%s", &myInput);

    // determine length of input

    /* algorithm
    left parentheses - push into stack
    right parentheses - pop stack and see if it match, break if not
    repeat until end of input
    if loop finish and stack empty - match
    if loop finish and stack not empty - not match
    */

    for (int i = 0; i < length; i++){
        if (/* if find left parentheses */){
            // push into stack
        }
        else if (/* if find ) */){
            // pop and check if the popped character is (
            // if not, break
        }
        // other cases
    }

    if (TOP == 0)
        printf("Parentheses is balanced");
    else
        printf("Parentheses is NOT balanced");

    return 0;
}
```

ขั้นตอนวิธีจัดเรียงข้อมูล (sorting algorithms)

ตัวเลขสุ่ม (random number) ในภาษาซี

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main()
{
    time_t t;
    /* Initializes random number generator */
    srand((int) time(&t));

    /* Print 10 random numbers from 0 to 100 */
    for(int i = 0 ; i < 10 ; i++ ) {
       printf("%d\n", rand() % 50);
    }
    return 0;
}
```

การจัดเรียงข้อมูลแบบ insertion sort

- มีอาเรย์ที่ไม่ได้จัดเรียงข้อมูล a
- ในการทำซ้ำครั้งที่ i สลับ a[i] กับทุกตัวในอาเรย์ที่อยู่ที่ตำแหน่ง <i ที่มีค่ามากกว่า a[i]

05DemoInsertionSort.pdf

ข้อ 7 – เขียน Insertion sort

1. สุ่มตัวเลขจำนวนเต็มใส่อาเรย์
2. เรียกฟังชั่น insertionSort
3. ในฟังชั่น insertionSort มี for loop สองชั้น
    1. ชั้นนอก นับจาก 1 จนสุดอาเรย์
    2. ชั้นใน นับลงจาก i ถึง 1 (สมมุติว่าใช้ j)
    3. ถ้าค่าที่ตำแหน่ง j-1 มากกว่า j ให้สลับตำแหน่งกัน

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define SIZE 20
#define MAX 100

void insertionSort(int*);

int main()
{
    time_t t;
    /* Initializes random number generator */
    srand((int) time(&t));

    int num[SIZE];
    int x;
    /* generate SIZE random numbers and put in array */
    for(int i = 0 ; i < SIZE ; i++ ) {
       x = rand() % MAX;
       num[i] = x;
       printf("%d ", num[i]);
    }

    insertionSort(num);
    // print array again

    return 0;
}

void insertionSort(int *a){
    int tmp;
    for (int i = 1; i < SIZE; i++){
        for (/*j counts down from i to 1*/){
            if ( /*a[j] is more than the one on the left */){
                // swap a[j] and a[j-1]
            }
            else
                break;
        }
    }
}
```

การจัดเรียงข้อมูลแบบ merge sort

- Insertion sort มีข้อเสียคือ จำนวนการเปรียบเทียบ (a[j-1] กับ a[j]) ที่จำเป็นต้องใช้ อยู่ที่ประมาณ $N^2/2$ สำหรับอาเรย์ขนาด N
- สำหรับอาเรย์ขนาดใหญ่ (N>1,000,000) จะใช้เวลานานมาก
- มีวิธีการจัดเรียงที่เร็วกว่าคือ merge sort

Merge (การรวมเป็นหนึ่ง)

- มีอาเรย์ย่อยสองอัน a[lo] ถึง a[mid] และ a[mid+1] ถึง a[hi] ที่จัดเรียงแล้ว
- รวมอาเรย์ย่อยสองอัน เป็นอันเดียว a[lo] ถึง a[hi] ที่จัดเรียง

06DemoMerge.pdf

Merge

```
1 void merge(int *a, int *aux, int lo, int mid, int hi){
2     // copy to aux array
3     for (int k = lo; k <= hi; k++)
4         aux[k] = a[k];
5
6     // i = begin of left array
7     // j = begin of right array
8     int i = lo, j = mid + 1;
9     for (int k = lo; k <= hi; k++){
10        if (i > mid) // left array finished
11            a[k] = aux[j++];
12        else if (j > hi) // right array finished
13            a[k] = aux[i++];
14        else if (aux[j] < aux[i]) // take from right
15            a[k] = aux[j++];
16        else // take from left
17            a[k] = aux[i++];
18    }
19 }
```

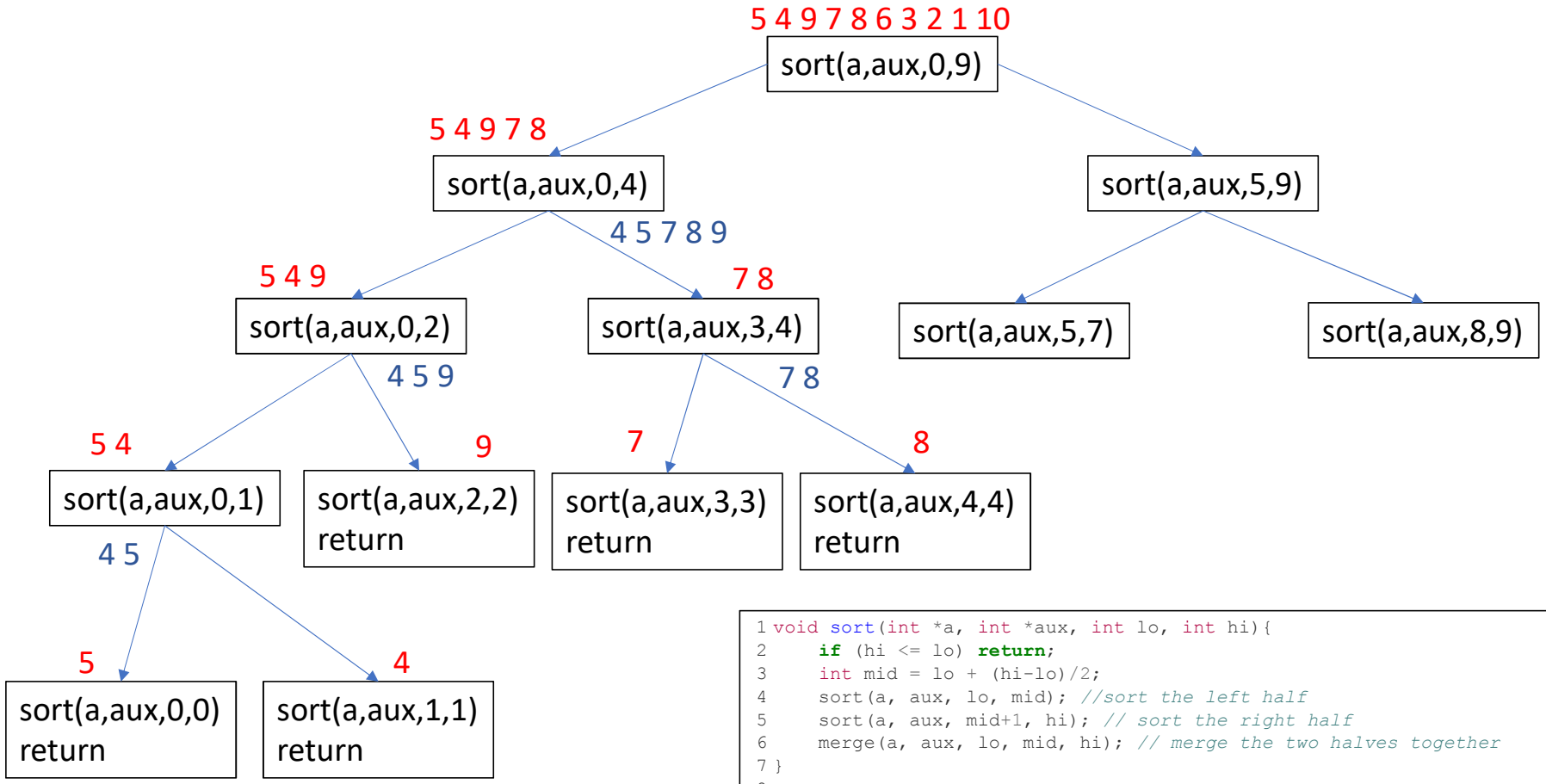
Merge sort (divide and conquer)

การนำฟังชั่น merge มาประกอบเป็น sort มีหลักการต่อไปนี้

1. ฟังชั่น sort รับตัวแปรสี่ตัวคือ อาเรย์ a อาเรย์สำหรับคัดลอกค่า aux และตำแหน่งในอาเรย์สองที่ lo hi
2. ถ้า hi <= lo หมายความว่าตอนนี้อาเรย์ย่อยมีขนาดเหลือแค่ 1 ให้ return
3. แบ่งอาเรย์ออกเป็นส่วน โดยประกาศตัวแปร mid ที่อยู่ตรงกลางระหว่าง lo และ hi
4. เรียกฟังชั่น sort โดยระบุขอบเขตเป็นครึ่งซ้ายของอาเรย์
5. เรียกฟังชั่น sort โดยระบุขอบเขตเป็นครึ่งขวาของอาเรย์
6. เรียกฟังชั่น merge

Merge sort

```
 1 void sort(int *a, int *aux, int lo, int hi){
 2     if (hi <= lo) return;
 3     int mid = lo + (hi-lo)/2;
 4     sort(a, aux, lo, mid); //sort the left half
 5     sort(a, aux, mid+1, hi); // sort the right half
 6     merge(a, aux, lo, mid, hi); // merge the two halves together
 7 }
 8
 9 void mergeSort(int *a){
10     int aux[SIZE];
11     sort(a, aux, 0, SIZE-1);
12 }
```




```
1 void sort(int *a, int *aux, int lo, int hi){
2     if (hi <= lo) return;
3     int mid = lo + (hi-lo)/2;
4     sort(a, aux, lo, mid); //sort the left half
5     sort(a, aux, mid+1, hi); // sort the right half
6     merge(a, aux, lo, mid, hi); // merge the two halves together
7 }
8
9 void mergeSort(int *a){
10    int aux[SIZE];
11    sort(a, aux, 0, SIZE-1);
12 }
```

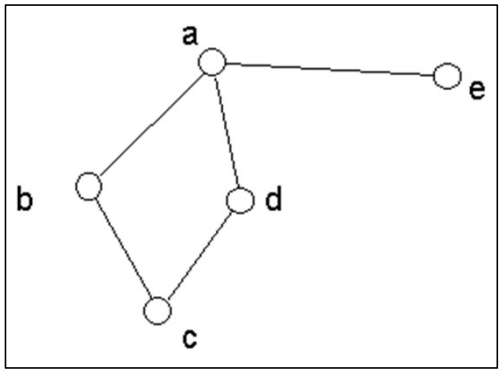
ข้อ 8 – เขียน Merge sort

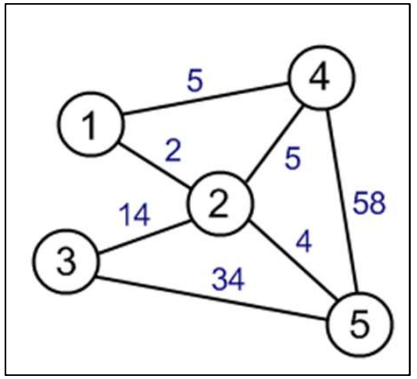
จงเขียนโปรแกรม merge sort ที่สมบูรณ์

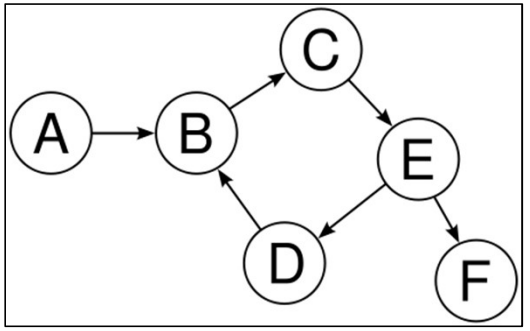
1. สร้างอาเรย์โดยการสุ่มเลขจำนวนเต็มใน main
2. แสดงผลอาเรย์
3. เรียกฟังชั่น mergeSort โดยส่งอาเรย์ไป
4. แสดงผลอาเรย์อีกครั้ง

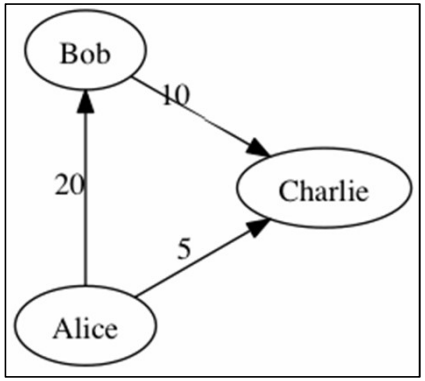
กราฟ (Graphs)

- กราฟ คือแผนภูมิที่แสดงถึงความเชื่อมโยงระหว่างวัตถุ เช่น เมืองที่เชื่อมต่อกันด้วยถนน หรือการเป็นเพื่อนกันบน social media
- กราฟ ประกอบด้วยโหนด (node) และ แอจ (edge) โดยโหนดแต่ละโหนดมีชื่อ ใช้แสดงถึงวัตถุ ส่วนแอจ แสดงถึงความเชื่อมโยงระหว่างโหนดสองโหนด โดยแอจ มีแบบระบุทิศทาง (มีลูกศร) กับไม่มีทิศทาง (ไม่มีลูกศร) และมีแบบมีน้ำหนัก (แต่ละแอจมีตัวเลขกำกับ) กับแบบไม่มีน้ำหนัก (แอจไม่มีตัวเลขกำกับ)





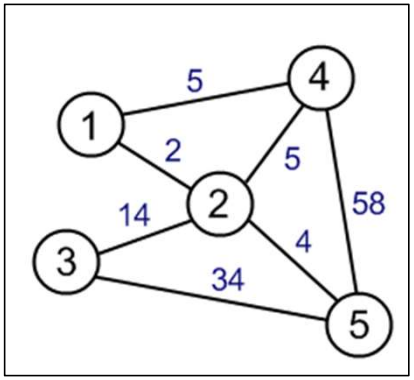




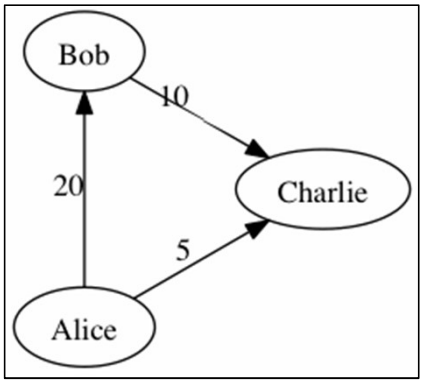
การแสดงกราฟในคอมพิวเตอร์

- การแสดงกราฟในคอมพิวเตอร์ ทำโดยใช้เมททริก V $(n \times n)$ ตัวเลขที่ตำแหน่ง V_{ij} แสดงถึงแอจจาก โหนด j ไปยัง โหนด i
- $V_{ij} = 0$ ในกรณีที่ไม่มีการเชื่อมต่อ ระหว่างโหนด j และโหนด i
- $V_{ij} = 1$ ในกรณีที่มีการเชื่อมต่อ ระหว่างโหนด j และโหนด i (กราฟที่ไม่มีน้ำหนัก)
- $V_{ij} = w$ ในกรณีที่มีการเชื่อมต่อ ระหว่างโหนด j และโหนด i (กราฟมีน้ำหนัก)
- กราฟที่ไม่มีทิศทาง $V_{ij} = V_{ji}$
- กราฟที่มีทิศทาง $V_{ij} \neq V_{ji}$

การแสดงกราฟในคอมพิวเตอร์



$$\begin{bmatrix} 0 & 2 & 0 & 5 & 0 \\ 2 & 0 & 14 & 5 & 4 \\ 0 & 14 & 0 & 0 & 34 \\ 5 & 5 & 0 & 0 & 58 \\ 0 & 4 & 34 & 58 & 0 \end{bmatrix}$$



$$\begin{bmatrix} 0 & 0 & 0 \\ 20 & 0 & 0 \\ 5 & 10 & 0 \end{bmatrix}$$

Alice = 1, Bob = 2, Charlie = 3

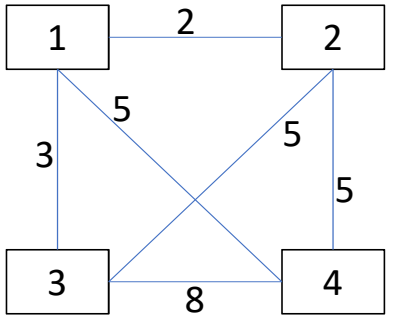
Weeping Fig (ACM-ICPC 2016)

- บ้านของคุณมีต้นไม้พิเศษอยู่ต้นหนึ่ง ที่มีหลายราก แต่ละราก เชื่อมต่อกับรากอื่นๆ
- การเชื่อมต่อแต่ละอัน มีความหนาไม่เท่ากัน ดังนั้นต้องใช้เวลาในการตัดไม่เท่ากัน
- เพื่อนของคุณอยากได้ต้นไม้ต้นนี้ไปปลูก การจะนำไปได้ ต้องแยกราก 1 ราก ออกจากรากอื่นๆที่เหลือทั้งหมด
- โจทย์คือ ถ้ามีราก n อัน และทราบการเชื่อมต่อระหว่างราก และเวลาที่ต้องใช้ในการตัดแต่ละอัน ให้หาเวลาน้อยที่สุด ที่จำเป็นต้องใช้ในการแยกรากออกมา 1 รากจากต้นไม้
- Input บรรทัดแรก มีเลขสองตัว $N\ M$ บอกจำนวนราก และจำนวนการเชื่อมต่อ
- อีก M บรรทัดต่อมา ประกอบด้วยชุดตัวเลขสามตัว a, b, w แสดงถึงการเชื่อมต่อระหว่าง ราก a และ ราก b ที่ต้องใช้เวลา w ในการตัด

Weeping Fig

- ตัวอย่าง input

```
4 6
1 2 2
1 3 3
1 4 5
2 3 5
2 4 5
3 4 8
```



$$V=\begin{bmatrix} 0 & 2 & 3 & 5 \\ 2 & 0 & 5 & 5 \\ 3 & 5 & 0 & 8 \\ 5 & 5 & 8 & 0 \end{bmatrix}$$

Weeping Fig

หลักการคิด

- **เวลาที่ต้องใช้ในการตัดราก i ออกจากต้นไม้ คือผลรวมของการเชื่อมต่อทุกอัน ที่มีโหนด i อยู่ด้วย**
- **หาผลรวมของแต่ละแถวของเมททริก V**
- **ผลรวมน้อยที่สุด คือคำตอบ**

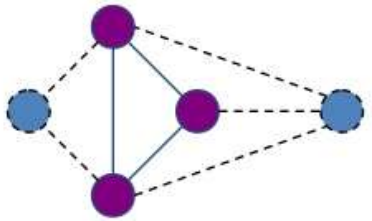
ข้อ 9 – Weeping Fig

- จงเขียนโปรแกรมแก้โจทย์ Weeping Fig ให้สมบูรณ์

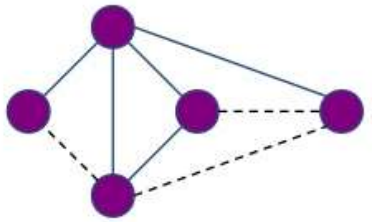
1. รับ input บรรทัดแรก (N, M)
2. สร้างอาเรย์สองมิติ ขนาด $N \times N$ เพื่อใช้เก็บกราฟ
3. รับ input M บรรทัดถัดมา โดยใช้ลูป เมื่อรับ input แต่ละบรรทัดมาแล้ว นำตัวเลขที่รับเข้ามาไปกรอกใส่อาเรย์ในตำแหน่งที่ถูกต้อง
   - เช่น 1 1 2 ให้ใส่ค่า 2 ไปที่อาเรย์ตำแหน่งที่ 0,0
4. หาผลรวมของแต่ละแถวในอาเรย์
5. หาผลรวมน้อยที่สุด

Subgraph

- กราฟย่อย (subgraph) ของกราฟ V คือกราฟที่มีบางส่วน (หรือทั้งหมด) ของโหนดและแอจของ V
- กราฟย่อยคลุม (spanning subgraph) ของกราฟ V คือกราฟย่อยที่มีโหนดทั้งหมดของ V (แต่อาจจะไม่มีแอจทั้งหมด)



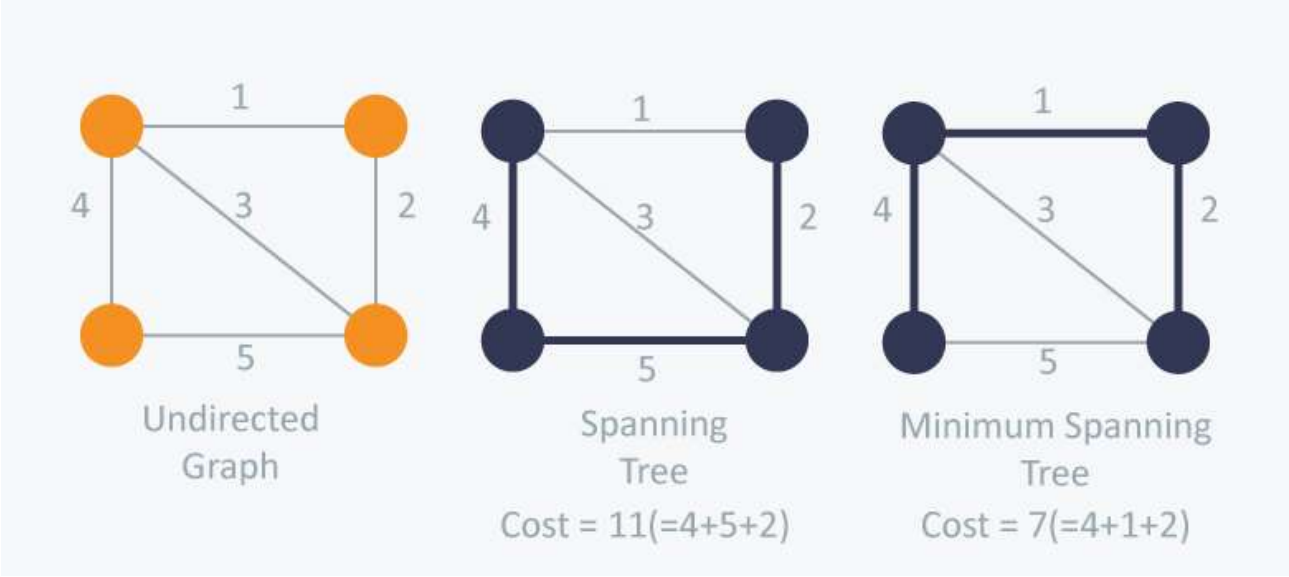
Subgraph



Spanning subgraph

Minimum Spanning Tree

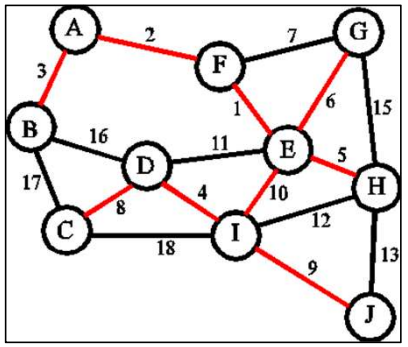
- Minimum Spanning Tree (MST) ของกราฟ V คือ spanning subgraph ที่มีผลรวมน้ำหนักแอจน้อยที่สุด



การหา MST (greedy)

1. เริ่มจากโหนดใดก็ได้ (เรียกว่า v1)
   1. ตั้งเซ็ตของแอจของ MST เป็นเซ็ตว่าง E = {}
   2. ตั้งเซ็ตของโหนดของ MST เป็นเซ็ตว่าง V = {}
2. ใส่ v1 เข้าไปใน V
3. เลือกแอจที่ค่าน้ำหนักน้อยที่สุด ที่เชื่อมต่อกับโหนดใดก็ได้ใน v1 เพิ่มเข้ามาใน E พร้อมกับเพิ่มโหนดที่แอจนั้น วิ่งไปหา เข้าไปใน V
4. ทำซ้ำจนกว่า V จะมีสมาชิกครบทุกโหนดในกราฟ คำตอบอยู่ใน E

การหา MST (greedy)



- เริ่มจากโหนด A V = {A}, E = {}
- เพิ่มแอจ AF เข้ามาใน E และโหนด F เข้ามาใน V V={A,F}, E={AF}
- เพิ่มแอจ FE เข้ามาใน E และโหนด E เข้ามาใน V V={A,F,E}, E={AF, FE}
- เพิ่มแอจ AB เข้ามาใน E และโหนด B เข้ามาใน V V={A,F,E,B}, E={AF, FE, AB}
- เพิ่มแอจ EH เข้ามาใน E และโหนด H เข้ามาใน V V={A,F,E,B,H}, E={AF, FE, AB, EH}
- เพิ่มแอจ EG เข้ามาใน E และโหนด G เข้ามาใน V V={A,F,E,B,H,G}, E={AF, FE, AB, EH, EG}
- เพิ่มแอจ EI ...
- เพิ่มแอจ ID ...
- เพิ่มแอจ DC ...
- เพิ่มแอจ IJ ...

โครงสร้างข้อมูลสำหรับการหา MST

- V สำหรับเก็บกราฟเต็ม ($N \times N$)
- อาเรย์ p สำหรับเก็บว่า ตอนนี้ใน MST มีโหนดอะไรบ้างแล้ว
- จำนวนเต็ม k สำหรับเก็บว่าตอนนี้ใน p มีสมาชิกกี่ตัว
- อาเรย์ e1 สำหรับเก็บต้นทางของแอจใน MST
- อาเรย์ e2 สำหรับเก็บปลางทางของแอจใน MST
- จำนวนเต็ม l สำหรับเก็บว่าตอนนี้ใน e1,e2 มีสมาชิกกี่ตัว

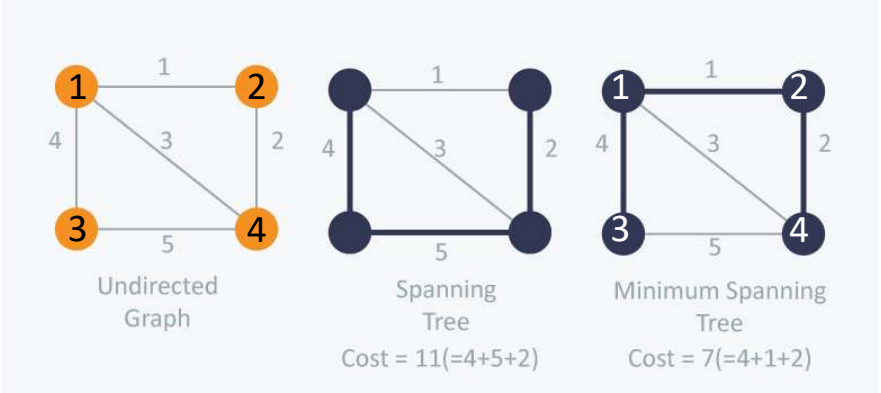
ตัวอย่างการหา MST

$$V=\begin{bmatrix}0 & 1 & 4 & 3\\ 1 & 0 & 0 & 2\\ 4 & 0 & 0 & 5\\ 3 & 2 & 5 & 0\end{bmatrix} \quad p=\{1\}\ e_1=\{\ \ \}\ e_2=\{\ \ \}$$

$$V=\begin{bmatrix}0 & 0 & 4 & 3\\ 0 & 0 & 0 & 2\\ 4 & 0 & 0 & 5\\ 3 & 2 & 5 & 0\end{bmatrix} \quad p=\{1,2\}\ e_1=\{2\}\ e_2=\{1\}$$

$$V=\begin{bmatrix}0 & 0 & 4 & 3\\ 0 & 0 & 0 & 0\\ 4 & 0 & 0 & 5\\ 3 & 0 & 5 & 0\end{bmatrix} \quad p=\{1,2,4\}\ e_1=\{2,4\}\ e_2=\{1,2\}$$

$$V=\begin{bmatrix}0 & 0 & 0 & 3\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 5\\ 3 & 0 & 5 & 0\end{bmatrix} \quad p=\{1,2,4,3\}\ e_1=\{2,4,3\}\ e_2=\{1,2,1\}$$



ข้อ 10 – เขียน MST

- V สำหรับเก็บกราฟเต็ม ($N \times N$)
- อาเรย์ p สำหรับเก็บว่า ตอนนี้ใน MST มีโหนดอะไรบ้างแล้ว
- จำนวนเต็ม k สำหรับเก็บว่าตอนนี้ใน p มีสมาชิกกี่ตัว
- อาเรย์ e1 สำหรับเก็บต้นทางของแอจใน MST
- อาเรย์ e2 สำหรับเก็บปลางทางของแอจใน MST
- จำนวนเต็ม l สำหรับเก็บว่าตอนนี้ใน e1,e2 มีสมาชิกกี่ตัว

1. หาค่าที่น้อยที่สุด (แต่ไม่ใช่ 0) ของทุกๆแถวของ V ที่เก็บไว้ใน p แล้ว
2. นำตำแหน่ง j ของค่าที่หาได้จากข้อ 1 เพิ่มเข้าไปใน p และ e1
3. นำตำแหน่ง i ของค่าที่หาได้จากข้อ 1 เพิ่มเข้าไปใน p และ e1
4. เปลี่ยน V[i][j] เป็น 0
5. ทำซ้ำจนกว่าจำนวนสมาชิกใน p จะเท่ากับ N