รายงานวิชา 2110327 Algorithm Design

ภาคการศึกษา 2023/3

Day Cover

a66_q1a_day_cover

พีรณัฐ กิตติวิทยากุล

6330374121

แนวทางการแก้ปัญหา

Version 1

Points	40.0/100
Comment	[PPP]PPP-P--P---------

day_cover_v1(vector< pair <int, int> > &staff_and_numfree, vector< vector <bool> > &availability, vector<bool> &covered_day_vector, int covered_day, int len, int staff_count, int n, int m) จะเป็น function ที่รับข้อมูล input ที่ function main ส่งไปแล้ว return จำนวนนิสิตน้อยที่สุดที่จะได้มาช่วยงานในทุก ๆ วัน โดยข้อมูลที่ main จะรับแล้วเตรียมก่อนส่งไปคำนวนได้แก่

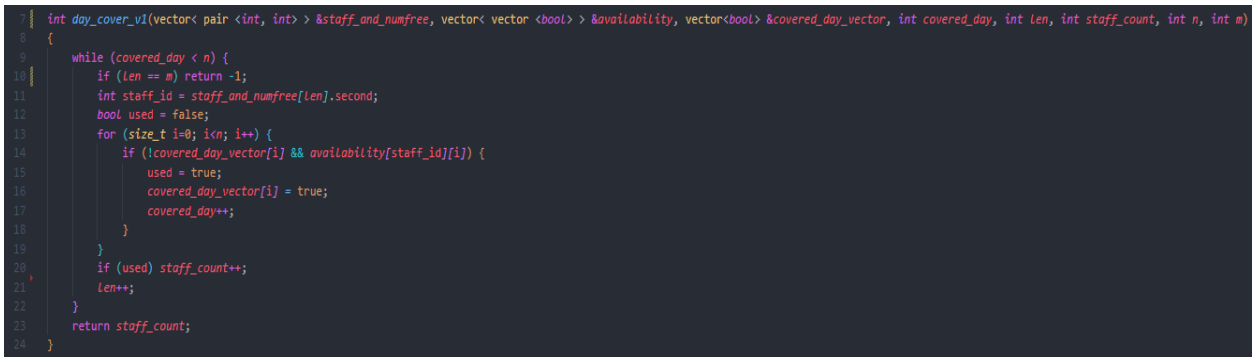
1. n, m จำนวนวันงานและจำนวนนิสิตที่รับมาตามโจทย์
2. staff_and_numfree เป็น vector ของ pair<int,int> ซึ่งจะเก็บ pair ของจำนวนวันที่ว่างของนิสิตคนที่ i จาก input และ index ค่า i ของนิสิตคนนั้น (เหตุผลที่เก็บ index ไว้ second จะระบุทีหลัง)
3. availability เป็น vector ขนาด m ของ vector<bool> ขนาด n โดยใน index ที่ i จะเก็บค่า vector ของวันที่นิสิตคนที่ i+1 ยินดีมาช่วยงาน เช่น นิสิตคนที่ 1 ว่าง 3 วันคือวันที่ 1 3 5 (3 1 3 5) available index ที่ 0 จะเก็บ vector {true, false, true, false, true}
4. covered_day_vector เป็น vector<bool> ขนาด n แทนวันนิทรรศการแต่ละวัน โดยจะ initialized เป็นค่า false ทั้งหมดแล้วส่งไปในฟังก์ชั่นเพื่อเปลี่ยนค่าระหว่างการคำนวนว่าวันไหนมีนิสิตมาช่วยงานแล้วบ้างก็เปลี่ยนวันนั้นเป็น true
5. staff_count จำนวนนิสิตที่มาช่วยงาน โดยจะเป็นคำตอบของโจทย์โดยจะเป็นค่า return กลับมาโดยครั้งแรกที่เรียนจาก main จะใส่ค่า 0
6. len เป็นค่า index ไล่การทำงานเริ่มจาก 0 ไปหยุดเมื่อ len == n

Summary on main()

```cpp
int main(int argc, char const *argv[])
{
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    int n, m;
    cin >> n >> m;

    vector< pair <int, int> > staff_and_numfree;
    vector< vector<bool> > availabilty(m);
    for (int i = 0; i < m; i++)
    {
        int days_free;
        cin >> days_free;

        if (days_free >= n)
        {
            cout << "1";
            return 0;
        }

        staff_and_numfree.push_back(make_pair(days_free, i));
        vector<bool> tmp(n);
        availabilty[i] = tmp;

        for (int j = 0; j < days_free; j++)
        {
            int day;
            cin >> day;
            availabilty[i][day - 1] = true;
        }
    }

    sort(staff_and_numfree.begin(), staff_and_numfree.end());
    reverse(staff_and_numfree.begin(), staff_and_numfree.end());


    vector<bool> covered_day_vector(n);
    cout << day_cover_v1(staff_and_numfree, availabilty, covered_day_vector, 0, 0, 0, n, m);
    return 0;
}
```

นอกจากการเตรียมของเพื่อส่งไปฟังก์ชั่น day_cover ที่ line 83,84 มีการเรียงของใน staff_and_numfree แล้ว reverse เพื่อให้ได้ vector ที่เรียง pair จากนิสิตที่วันสะดวกช่วยงานเยอะขึ้นมาก่อนจนวันสะดวกน้อย และใน line 65 มีการดักกรณีถ้ามีนิสิตคนไหนสะดวกช่วยงานมากกว่าหรือเท่ากับจำนวนวันนิทรรศการแล้ว ก็ใช้นิสิตคนนั้นคนเดียวได้เลยก็ตอบ 1 เลย

แนวคิดของ day_cover_v1()



```
int day_cover_v1(vector< pair <int, int> > &staff_and_numfree, vector< vector <bool> > &availability, vector<bool> &covered_day_vector, int covered_day, int len, int staff_count, int n, int m)
{
    while (covered_day < n) {
        if (len == m) return -1;
        int staff_id = staff_and_numfree[len].second;
        bool used = false;
        for (size_t i=0; i<n; i++) {
            if (!covered_day_vector[i] && availability[staff_id][i]) {
                used = true;
                covered_day_vector[i] = true;
                covered_day++;
            }
        }
        if (used) staff_count++;
        len++;
    }
    return staff_count;
}
```

แนวคิดของฟังก์ชั่นนี้คือเน้น brute force แบบไล่วน loop ธรรมดาโดยจะวน while loop จนกว่าจำนวน covered_day หรือวันที่มีนิสิตมาช่วยงานครบ n วันที่ต้องการ ถ้า len แทนจำนวนนิสิตไล่ถึง m แต่ covered_day ยังไม่ครบก็จะสรุปว่าได้วันไม่ครบแม้ไล่นิสิตทุกคนแล้ว โดยแต่ละ while iteration จะทำนิสิตคนที่ i ที่เรียงจากจำนวนวันว่างมากไปน้อยแล้ว เอา index มาไล่ถ้าวันนั้นยังไม่มีนิสิตมาช่วย(ใน covered_day_vector ยังเป็น false) และวันนั้นใน availability[staff_id] เป็น true ให้ปรับวันนั้นเป็น true แล้วเพิ่มจำนวนวันที่ covered แล้วเพิ่มจำนวน staff_count ว่าเอา staff_count คนนี้ ทำจนได้วันครบก็ return จำนวนนิสิตกลับมา

ปัญหาของ day_cover_v1()

ปัญหาของ v1 คือมี testcases ที่ถูกและผิด แปลว่าอัลกอริทึ่มนี้ไม่ได้ให้คำตอบที่ถูกต้องในทุก instances เนื่องจากแนวคิดของวิธีนี้คือไล่เอานิสิตทีละคนมาลงวันว่างตั้งแต่หัวไปท้าย แต่ปัญหานี้เป็นปัญหา constraint optimization problem นั่นคือมี optimization function คือต้องได้ staff_count ที่น้อยที่สุด จึงมีคำตอบได้ชุดเดียว ถึงวิธี v1 เรียงจำนวนวันว่างจากมากไปน้อยแต่ไม่ได้หมายความว่านิสิตลำดับที่ 0 1 2 ... จะสามารถเติมวันจนเต็มได้ดีที่สุด

ตัวอย่าง instance ที่ v1 ให้คำตอบผิดซึ่งไม่ minimum คือ instance ที่คนที่จำนวนวันว่างน้อย ๆ ซึ่งอยู่ลำดับท้าย ๆ ของ vector staff_and_numfree ว่างเติมวันที่คนลำดับต้น ๆ ไม่ว่างพอดี แต่ถ้าไล่ตามจำนวนวันว่างอาจจะต้องใช้จำนวนคนเติมวันที่ขาดมากกว่าใช้คนลำดับหลัง ๆ เช่น

Input:

n = 6 m = 4

4 1 3 5 6

3 1 2 5

3 1 4 5

2 2 4

T	F	T	F	T	T
T	T	F	F	T	F
T	F	F	T	T	F
F	T	F	T	F	F

Output = 3 Expect = 2

ตารางด้านขวาไล่จากบนลงล่างคือเรียงตามจำนวนวันว่างแล้ว โดนคนแรกว่าง 4 วันแต่ขาดวันที่ 2 และ 4 หากไล่ตาม v1 จะได้คนที่ 2 เติมวันที่ 2 และคนที่ 3 เติมวันที่ 4 ทั้งที่จริง ๆ แล้วคนที่ 4 ที่ว่างน้อยสุด 2 วันสามารถเติมทั้งวันที่ 2 และวันที่ 4 ได้ในคนเดียวทำให้ได้ค่า minimum กว่าที่ 2 คน v1 จึงยังไม่สามารถตอบคำถามนี้ได้อย่างถูกต้องทุก instances

Version 2

Points	65.0/100
Comment	[PPP]PPPPPPPPPPTTTTTTT

day_cover_v2(vector< vector <bool> > &availability, vector<bool> covered_day_vector, int len, int m) จะ return จำนวนนิสิตน้อยที่สุดที่จะได้มาช่วยงานในทุก ๆ วันเหมือนกันโดยจะรับพารามิเตอร์แค่ 4 ตัวคือ availability, covered_day_vector, len, m ซึ่งรับข้อมูลและความหมายเดิมกับ v1

Summary on main()

```cpp
int main(int argc, char const *argv[])
{
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    int n, m;
    cin >> n >> m;

    vector< vector<bool> > availabilty(m);
    for (int i = 0; i < m; i++)
    {
        int days_free;
        cin >> days_free;

        if (days_free >= n)
        {
            cout << "1";
            return 0;
        }

        vector<bool> tmp(n);
        availabilty[i] = tmp;

        for (int j = 0; j < days_free; j++)
        {
            int day;
            cin >> day;
            availabilty[i][day - 1] = true;
        }
    }

    vector<bool> covered_day_vector(n);
    cout << day_cover_v2(availabilty, covered_day_vector, 0, m);
    return 0;
}
```

พอไม่ต้องใช้ staff_and_numfree แล้วก็สามารถลบ line ที่เกี่ยวข้องทั้งหมดได้รวมถึง line การ sort และ reverse ด้วย

แนวคิดของ day_cover_v2()

```
26  // use combination
27  int day_cover_v2(vector< vector <bool> > &availability, vector<bool> covered_day_vector, int len, int m)
28  {
29      // All days are covered, end of this path.
30      if (!count(covered_day_vector.begin(), covered_day_vector.end(), false)) {
31          return 0;
32      }
33
34      // Iterate until last one but all days are not covered -> This path cannot be the candidate solution.
35      if (len == m) {
36          return m+1;
37      }
38
39      // exclude current staff (staff count is not incremented.)
40      int exclude = day_cover_v2(availability, covered_day_vector, len+1, m);
41
42      // include current staff (staff count++)
43      for (size_t i=0; i<covered_day_vector.size(); i++) {
44          covered_day_vector[i] = covered_day_vector[i] || availability[len][i];
45      }
46      int include = day_cover_v2(availability, covered_day_vector, len+1, m) + 1;
47      return min(exclude, include);
48  }
```

แนวคิดของฟังก์ชั่นนี้คือ brute force โดยใช้หลักการ combination และ recursion ที่เรียนในคลาส โดยจะแยกเป็นสองขา exclude คือไม่เอา staff คนที่ len ส่วน include คือเอาคนที่ len termination condition มี 2 กรณี

1. line 30: node นี้ถ้าได้นิสิตมาช่วยงานทุกวันแล้วจะ return 0 เพื่อให้ฟังก์ชั่นที่เรียก include นิสิตคนนี้ ได้ค่า 0 มา +1 ที่ line 46 เป็นนับ 1 ที่ความลึกนั้น
2. line 35: ถ้าวน recursively จนถึง len ที่ m หรือครบนิสิตทุกคนแล้วยังได้วันไม่ครบแปลว่า path ที่สิ้นสุด node นี้ใช้ไม่ได้ และเมื่อสุดท้ายเราเอาสองทางที่แตก recursion มา min กัน เพื่อการันตีว่าถ้าเทียบกับคำตอบที่ถูกต้อง คำตอบนั้นจะต้องผ่าน function min ออกไปเสมอ จึงให้ node ที่ไม่มีทางเป็นคำตอบ return m+1 เพราะไม่มีทางที่คำตอบที่ถูกต้องจะเกินจำนวนนิสิต

- exclude ไม่เอาคนนี้ ไม่ต้องทำอะไร เรียก recursion แต่ส่ง len+1 ไปเพื่อไปที่นิสิตคนต่อไป

- include เอาคนนี้ ก่อนที่จะเรียก recursion จะต้องปรับ covered_day_vector ก่อน โดยไล่วนตามจำนวนวันนิทรรศการ(n รอบ) แล้วเอาค่า Boolean ปัจจุบันไป OR กับ availability วันนั้นของนิสิตคนที่ len

day_cover_v2() Time Complexity

- termination condition 1 เป็น std::count() => O(n)
- termination condition ที่ 2 เป็น if else ธรรมดา => O(1)
- for loop ปรับ covered_day_vector วน n รอบ => O(n)
- min(exclude, include) => O(1)

จะได้ T(n) = 2T(m-1) + O(2n) เมื่อคำนวน big O notation ถ้าหากเป็น T(n) = 2T(n-1) + O(2n) จะได้ O(2^n) แต่เนื่องจาก m กับ n ไม่เท่ากันจึงเป็นการวัดกันระหว่างการเรียก recursion ของ 2T(m-1) กับ การทำงานที่ต้องทำในทุกการเรียกซ้ำ O(2n) ว่าก้อนไหนจะมากกว่าและ dominate อีกก้อนได้ big O จะยึดตามก้อนนั้น

ปัญหาของ day_cover_v2()

จากผลลัพธ์คะแนน วิธีนี้ให้คำตอบ testcase 13 cases แรกได้ถูกต้อง แปลว่าสามารถให้คำตอบที่ถูกต้องได้มากกว่า v1 แน่นอนแต่ตั้งแต่ case 14 เป็นต้นไปใช้เวลาเกิน เมื่อไปสำรวจเปรียบเทียบ testcase ที่ผ่านและไม่ผ่านพบว่า testcase 14 n=900 และ 15 ถึง 20 n=1000 ส่วนก่อนหน้านี้ไม่มี case ไหน n ถึง 900 จึงสันนิษฐานว่าถ้าหากเราสามารถทำให้ส่วนของฟังก์ชั่นที่ทำงานกับ n ทำงานเร็วขึ้นจะสามารถทำให้ program เร็วขึ้นจนทันเวลาใน testcases ที่เหลือหรือไม่เพราะดูจากจำนวน n กับ m ในเคสที่ไม่ผ่าน n ค่อนข้างมากเทียบกับ m (900 19, 1000 20, 1000 20, 1000 20, 1000 20, 1000 20, 1000 20)

Version 3

Points	100.0/100
Comment	[PPP]PPPPPPPPPPPPPPPP

Objective ของ version นี้คือการทำให้ code ส่วนที่ทำงานกับ n ทำงานเร็วขึ้น นั่นคือ termination condition 1 std::count() => O(n) และ for loop ปรับ covered_day_vector => O(n)

หากลองพิจารณาแล้ว ที่ for loop ปรับ vector ของ boolean ใช้เวลาค่อนข้างนานในการไล่ทีละ index n รอบแล้วเอาค่า boolean มา OR กัน มีวิธีที่ทำให้เร็วกว่านี้หรือไม่ จะมี data structure ไหนที่สามารถทำให้ข้อมูล boolean หลาย ๆ จำนวนติดกันทำงานได้เร็วกว่า vector เพราะปกติถ้าเป็นข้อมูลประเภทอื่นที่ต้องไล่ loop ทำงานต่าง ๆ อาจจะต้องใช้ vector แต่การทำงานของอัลกอริทึ่มเราใช้แค่ OR ข้อมูลทั้งแผงอย่างเดียว จึงไปศึกษา std::bitset ซึ่งเป็น class ที่ represents a fixed-size sequence of N bits

A bitset is an array of bools but each boolean value is not stored in a separate byte instead, bitset optimizes the space such that each boolean value takes 1-bit space only, so space taken by bitset is less than that of an array of bool or vector of bool.

Reference: https://www.geeksforgeeks.org/cpp-bitset-and-its-application/

แนวคิดของ day_cover_v3()

เมื่อลองพิจารณาแล้ว ตัว bitset เป็น data structure ที่เหมาะจะใช้แทน vector<bool> มาก ๆ ในกรณีของโจทย์ข้อนี้เพราะใน v2 operation ที่เราเอามาใช้มีแค่การ OR ทั้ง vector และนับจำนวน element ที่เป็น true ใน vector เมื่อใช้ bitset ดังนั้นเราจึงเปลี่ยนประเภท data structure ของ covered_day_vector และ availability เป็นใช้ bitset แทน vector<bool>

การประกาศฟังก์ชั่นและพารามิเตอร์จึงเปลี่ยนเป็น day_cover_v3(vector< bitset<1000> > &availability, bitset<1000> covered_day_vector, int len, int m, int n)

Summary on main()

```cpp
#include <bitset>
using namespace std;
```

```cpp
int main(int argc, char const *argv[])
{
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    int n, m;
    cin >> n >> m;

    vector< bitset<1000> > availabilty(m);
    for (int i = 0; i < m; i++)
    {
        int days_free;
        cin >> days_free;

        if (days_free >= n)
        {
            cout << "1";
            return 0;
        }

        for (int j = 0; j < days_free; j++)
        {
            int day;
            cin >> day;
            availabilty[i].set(day - 1); // Using 0-based indexing
        }
    }

    bitset<1000>covered_day_vector;
    cout << day_cover_v3(availabilty, covered_day_vector, 0, m, n);
    return 0;
}
```

เนื่องจาก A limitation of the bitset is that size must be known at compile time i.e. size of the bitset is fixed. ดังนั้นเราต้องกำหนดขนาดของ bitset ใน code เลยจึงเป็นข้อเสียจุดหนึ่งของอัลกอริทึ่มนี้ เนื่องจาก input n ที่จะรับมากสุดคือ 1000 จึงกำหนดขนาด bitset เป็น 1000

แนวคิดของ day_cover_v3() (ต่อ)

```
51  // use combination + std::bitset
52  int day_cover_v3(vector< bitset<1000> > &availability, bitset<1000> covered_day_vector, int len, int m, int n)
53  {
54      // All days are covered, end of this path.
55      if (covered_day_vector.count() == n) {
56          return 0;
57      }
58
59      // Iterate until last one but all days are not covered -> This path cannot be the candidate solution.
60      if (len == m) {
61          return m+1;
62      }
63
64      // exclude current staff (staff count is not incremented.)
65      int exclude = day_cover_v3(availability, covered_day_vector, len+1, m, n);
66
67      // include current staff (staff count++)
68      covered_day_vector |= availability[len];
69
70      int include = day_cover_v3(availability, covered_day_vector, len+1, m, n) + 1;
71      return min(exclude, include);
72  }
```

จากที่กล่าวมาข้างต้น version นี้จะปรับจาก v2 2ที่ คือ termination condition 1 และ for loop ปรับ covered_day_vector

1. line 55 : จากเดิมที่ใช้ std::count() เปลี่ยนเป็น bitset.count() แทน
2. lint 68 : จากเดิมที่เป็นการวน for loop ตามจำนวน n แล้ว OR ทีละ index เป็นการใช้ bitwise OR operator ระหว่าง bitset กับ bitset เลย

day_cover_v3() Time Complexity

- bitset.count() time complexity คือ O(number of bits in the bitset) => O(1000)
- termination condition ที่ 2 => O(1)
- bitwise OR => O(1)
- min(exclude, include) => O(1)

จะได้ว่า time complexity โดยภาพรวมจะกลายเป็น T(n) = 2T(m-1) + O(1000) ถ้าหากเขียนตามหลักจริง ๆ 1000 คือค่า n ที่มากที่สุดที่รับได้ หากพิจารณาภาพรวม งานที่ทำจริงของแต่ละ node ในก้อนหลัง ก็เป็น O(n) เหมือนกัน แต่เนื้อในจริง ๆ เราสามารถลดงานส่วนที่ทำ for loop ได้จาก n

รอบจนเหลือแค่ O(1) และจากลักษณะ input ที่ n มีค่าเยอะเทียบกับ m การลดเพียงส่วนนี้ก็เพียงพอที่ทำให้ code day_cover_v3() สามารถทำงานเร็วพอที่จะคำนวน testcases ที่เหลือภายในเวลาที่กำหนดได้

Finalized code: https://github.com/Peeranut-Kit/algorithm-design-coding/blob/main/problem/day_cover/day_cover.cpp

Reference: https://www.geeksforgeeks.org/cpp-bitset-and-its-application/