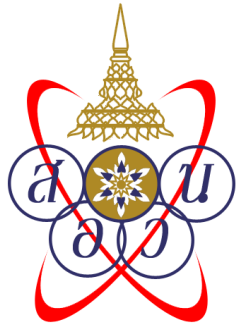


เอกสารประกอบการอบรม

ขั้นตอนวิธีแบบละโมบ

(Greedy Algorithms)

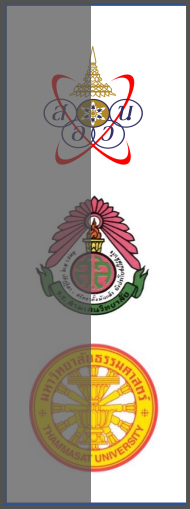
ค่ายคอมพิวเตอร์โอลิมปิก สอวน. ค่าย 2 2/2566
ศูนย์โรงเรียนสามเสนวิทยาลัย - มหาวิทยาลัยธรรมศาสตร์
ระหว่างวันที่ 18 มีนาคม – 3 เมษายน 2567

โดย
สาขาวิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์และเทคโนโลยี
มหาวิทยาลัยธรรมศาสตร์

25 มีนาคม 2567

เค้าโครงการบรรยาย

- ขั้นตอนวิธีแบบละโมบ (Greedy Algorithms)



ผศ.ดร.ฐาปนา บุญชู

25 มีนาคม 2567

Greedy Algorithms

Rod cutting

Rod Cutting

- A company buys long steel rods and cuts them into shorter rods, which it then sells.
- We assume that we know, for $i = 1,2, \ldots$, the price p_i in dollars that the company charges for a rod of length i inches.

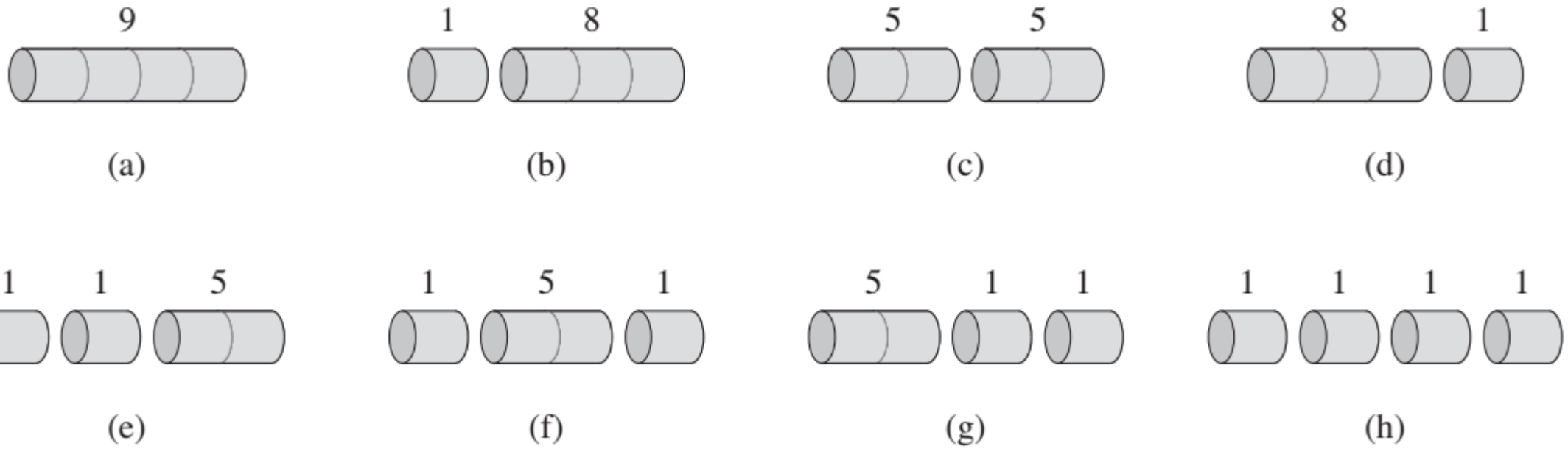
length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

Rod-cutting problem: Given a rod of length n inches and a table of prices p_i for $i = 1,2, \ldots$, determine the maximum revenue r_n obtainable by cutting up the rod and selling the pieces.

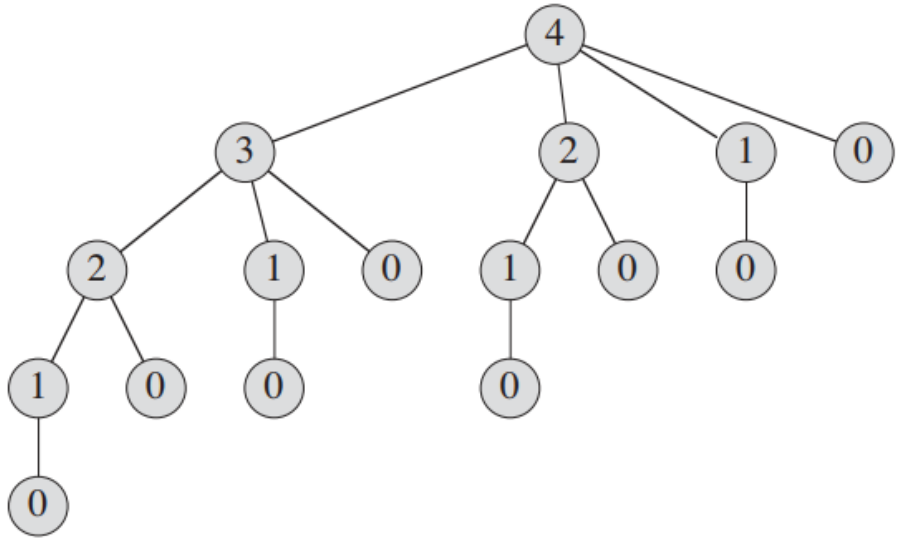
Rod Cutting Example

- Consider the case when $n = 4$,

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30



Rod Cutting Example




```
CUT-ROD(p, n)
1  if n == 0
2       return 0
3  q = −∞
4  for i = 1 to n
5       q = max(q, p[i] + CUT-ROD(p, n − i))
6  return q
```

Greedy Algorithms

- Algorithms สำหรับปัญหา Optimization โดยปรกติจะเป็นการเดินแต่ละ Step ใน Sequence หนึ่ง โดยการเดินแต่ละ Step จะต้องเลือกว่าจะเดินไปทางไหนดี
- สำหรับบางปัญหาที่เป็น Optimization การใช้ Dynamic Programming เพื่อหา Optimal solution จะเหมือนขี่ช้างจับตั๊กแตน (Overkill)
- Greedy algorithm (ขั้นตอนวิธีละโมบ) นั้นจะเลือกทางที่ดีที่สุด “At the moment” (Locally) หมายถึงเลือกทางเลือกที่ Optimal ที่สุดใน Step ล่าสุด เพื่อหวังว่าการนี้จะพาไปสู่ **Globally optimal solution**
- แต่การเลือกในลักษณะนี้ สุดท้ายแล้วอาจจะไม่ได้ Optimal solution ก็ได้ แต่สำหรับหลาย ๆ ปัญหามันสามารถหาได้ !

An activity-selection problem

An activity-selection problem

- ปัญหาคือการจัดการกิจกรรม (Activities) ที่มีการใช้ทรัพยากรร่วมกัน โดยที่ต้องการหาเซตของกิจกรรมที่ได้จำนวนกิจกรรมเยอะที่สุด โดยที่กิจกรรมที่เหล่านั้นต้องเป็น Compatible activities (Mutually)
- ให้ $S = \{a_1, a_2, \ldots . a_n\}$ เป็นเซตของ n กิจกรรมที่มีการใช้ทรัพยากรร่วมกัน
  - แต่ละกิจกรรม a_i จะมีเวลาเริ่มต้น s_i และเวลาสิ้นสุด f_i โดยที่ $0 \leq s_i \leq f_i \leq \infty$
  - กิจกรรม a_i และ a_j จะ Compatible เมื่อช่วงเวลาใช้ทรัพยากร $[s_i, f_i)$ และ $[s_j, f_j)$ ไม่ซ้อนทับกัน

An activity-selection problem

- สำหรับ Activity-selection problem เราต้องการหา **Maximum-size subset of mutually compatible activities**

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

*เรียงตามเวลาสิ้นสุดของกิจกรรม

- พิจารณา Sub-problems
  - สมมติว่า S_{ij} คือเซตของกิจกรรมที่เริ่มหลังจากที่กิจกรรม a_i สิ้นสุดและเสร็จสิ้นก่อน a_j เริ่ม
  - และเราต้องการหา Maximum set of mutually compatible activities A_{ij}ใน S_{ij} นี้ที่รวมบางกิจกรรม a_k เข้าไปด้วย
  - การรวม a_k เข้าไปใน Optimal solution A_{ij} นั้นทำให้เกิด 2 Sub-problems คือ S_{ik} and S_{kj}
  - โดยให้ $A_{ik} = A_{ij} \cap S_{ik}$ และ $A_{kj} = A_{ij} \cap S_{kj}$
  - จะได้ว่า $A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}$
  - ทำให้ A_{ij} ของ S_{ij} ประกอบไปด้วยจำนวนกิจกรรมทั้งหมด $|A_{ij}| = |A_{ik}| + |A_{kj}| + 1$ activities.

An activity-selection problem

- กำหนดให้ขนาดของ Optimal solution ใน S_{ij} ด้วย $c[i,j]$ และจากการวิเคราะห์ก่อนหน้าเราสามารถเขียนความสัมพันธ์ได้ว่า

$$c[i,j] = c[i,k] + c[k,j] + 1$$

- ในกรณีที่เราไม่รู้ว่า Optimal solution ใน S_{ij} นั้นมี a_k อยู่ตรงไหน เราต้องหาทุก ๆ กรณีที่เป็นไปได้ กล่าวคือ

$$c[i,j] = \begin{cases} 0 & \text{if } S_{ij} = \emptyset\,, \\ \max\limits_{a_k \in S_{ij}} \{c[i,k] + c[k,j] + 1\} & \text{if } S_{ij} \neq \emptyset\,. \end{cases}$$

An activity-selection problem

- Making a **greedy** choice
  - หากในแต่ละ Step การเลือก Activity มาใส่ใน Optimal solution โดยไม่จำเป็นต้องพิจารณาหรือไม่ต้องรู้ Optimal solution ของ Subproblems ก่อน
  - หากเป็นไปได้เราจะไม่จำเป็นต้องพิจารณาทุก ๆ Subproblems ก่อนหน้าและช่วยเราประหยัดเวลาในการประมวลผลได้
  - ใน Activity selection problem นี้เราพิจารณาเพียง Choice เดียวก็เพียงพอคือ “Greedy choice”
  - ด้วย Common sense เราน่าจะ**เลือก Activity ที่ทำให้เราเหลือเวลาสำหรับทำกิจกรรมอื่นให้เยอะมากที่สุด**
    - ด้วยแนวคิดนี้ให้เราเลือกกิจกรรมในเซต S ที่เวลา Finish time เกิดขึ้นเร็วที่สุดเพื่อจะให้เหลือเวลาทำกิจกรรมอื่นให้เยอะมากที่สุด

An activity-selection problem

- Making a **greedy** choice (ต่อ)
  - เนื่องจากกิจกรรมนั้นถูกเรียงลำดับตามเวลาสิ้นสุดมาอยู่แล้วกิจกรรมแรกที่เราจะเลือกคือ a_1
  - และหากเราพิจารณาเลือก Greedy choice นี้เราจะเหลือเพียง 1 Subproblem ที่ต้อง Solve นั่นคือ หากิจกรรม**หลัง a_1 สิ้นสุด**
  - Let $S_k = \{a_i \in S : s_i \geq f_k\}$ เป็นเซตที่มีกิจกรรมที่เกิดขึ้นหลัง**เวลาสิ้นสุดของ a_k**
  - ** หมายความว่า เมื่อเรา Make greedy choice สำหรับ Activity a_1 แล้ว S_1 จะกลายเป็นเพียง 1 Subproblem เท่านั้นที่ต้องแก้ต่อไป

An activity-selection problem

- คำถามคือ Greedy choice ที่เราเลือกมันถูกต้องแล้วหรือไม่ เพียงแค่เลือกกิจกรรมอันที่สิ้นสุดก่อนใน Subproblem นี้ และจะแน่ใจได้อย่างไรว่า a_k ที่เลือกนั้นเป็นส่วนหนึ่งของ Optimal solution

Consider any nonempty subproblem S_k, and let a_m be an activity in S_k with the earliest finish time. Then a_m is included in some maximum-size subset of mutually compatible activities of S_k.

Proof Let A_k be a maximum-size subset of mutually compatible activities in S_k, and let a_j be the activity in A_k with the earliest finish time. If $a_j = a_m$, we are done, since we have shown that a_m is in some maximum-size subset of mutually compatible activities of S_k. If $a_j \neq a_m$, let the set $A'_k = A_k - \{a_j\} \cup \{a_m\}$ be A_k but substituting a_m for a_j. The activities in A'_k are disjoint, which follows because the activities in A_k are disjoint, a_j is the first activity in A_k to finish, and $f_m \leq f_j$. Since $|A'_k| = |A_k|$, we conclude that A'_k is a maximum-size subset of mutually compatible activities of S_k, and it includes a_m. ■

An activity-selection problem

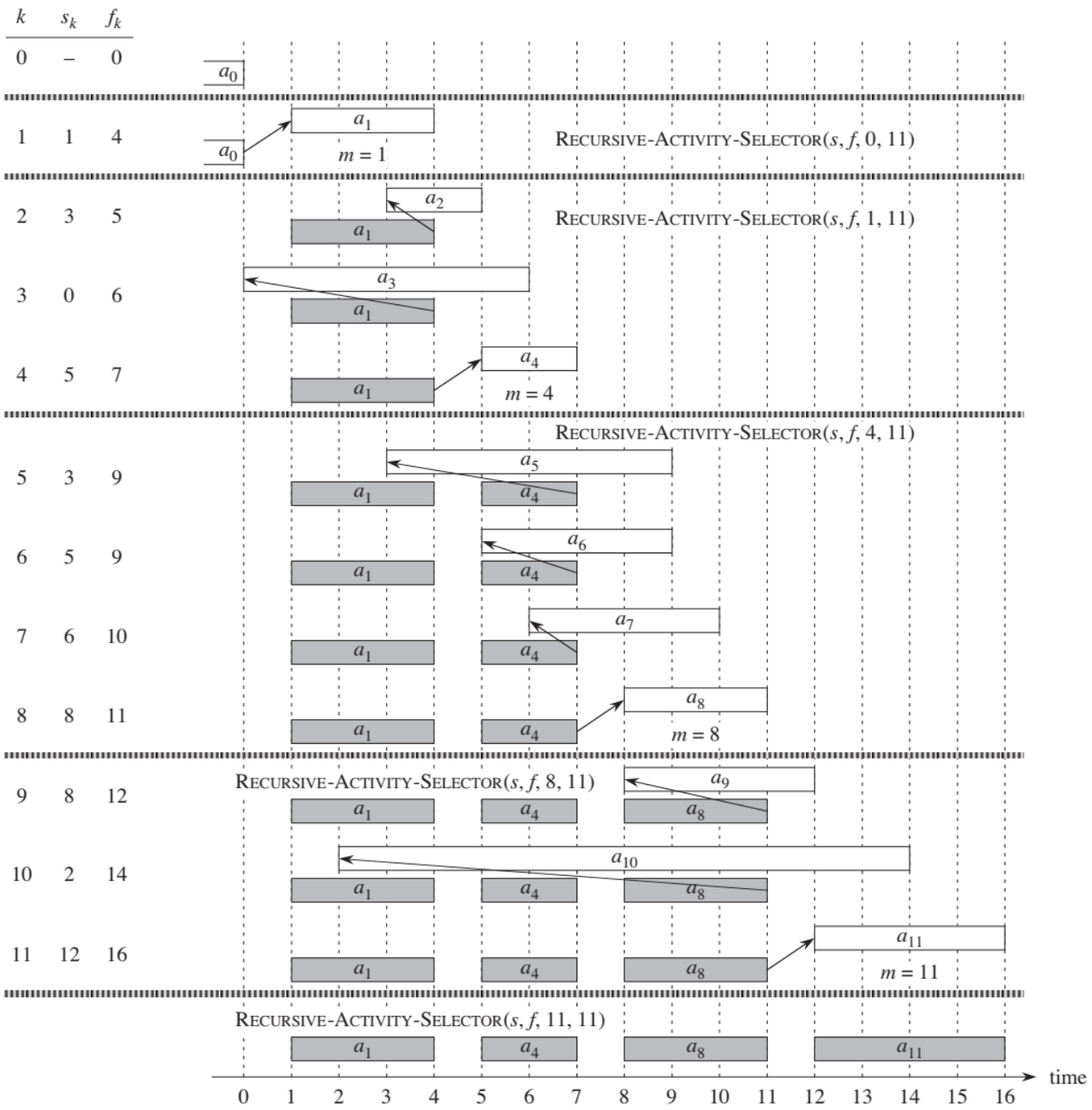
- Greedy algorithm ที่ใช้ในการแก้ปัญหา Activity selection problem นั้นมีการใช้ Top-down approach กล่าวคือ มันเลือก Activity แล้วใส่เข้าไปในเซตคำตอบที่เป็น Optimal solution แล้วก็ทำการเลือก Activity อื่น ๆ ที่ Compatible กับ Activity ที่เคยเลือกไปแล้วต่อ
- ซึ่ง Greedy algorithms ส่วนใหญ่จะมีการทำงานในลักษณะนี้คือ Make choice แล้ว Solve subproblem

A recursive greedy algorithm

- RECURSIVEACTIVITY-SELECTOR รับพารามิเตอร์
  - s: start times
  - f: finish times
  - k: ใช้ในการกำหนด sub problem S_k
  - n: size ของ Original problem
- RECURSIVEACTIVITY-SELECTOR คืนค่า maximum-size set of mutually compatible activities in S_k
- Assumption: สมมติว่าทุก ๆ Activities ถูกเรียงลำดับด้วย Finish times มาเรียบร้อยแล้ว

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1  m = k + 1
2  while m ≤ n and s[m] < f[k]        // find the first activity in S_k to finish
3      m = m + 1
4  if m ≤ n
5      return {a_m} ∪ RECURSIVE-ACTIVITY-SELECTOR(s, f, m, n)
6  else return ∅
```

Time complexity: $\boldsymbol{\Theta(n)}$; *Assuming that the activities were already sorted initially by their finish times.*


k
s_k
f_k
0 – 0
1 1 4
2 3 5
3 0 6
4 5 7
5 3 9
6 5 9
7 6 10
8 8 11
9 8 12
10 2 14
11 12 16
a_0
a_1
$m = 1$
RECURSIVE-ACTIVITY-SELECTOR(s, f, 0, 11)
a_2
RECURSIVE-ACTIVITY-SELECTOR(s, f, 1, 11)
a_3
a_4
$m = 4$
RECURSIVE-ACTIVITY-SELECTOR(s, f, 4, 11)
a_5
a_6
a_7
a_8
$m = 8$
RECURSIVE-ACTIVITY-SELECTOR(s, f, 8, 11)
a_9
a_{10}
a_{11}
$m = 11$
RECURSIVE-ACTIVITY-SELECTOR(s, f, 11, 11)
time
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

An iterative greedy algorithm

```
GREEDY-ACTIVITY-SELECTOR(s, f)
1  n = s.length
2  A = {a_1}
3  k = 1
4  for m = 2 to n
5      if s[m] ≥ f[k]
6          A = A ∪ {a_m}
7          k = m
8  return A
```

f[k] จะเป็นเวลาที่ a_k *สิ้นสุดเสมอ*

Time complexity: $\boldsymbol{\Theta(n)}$; *Assuming that the activities were already sorted initially by their finish times.*

6 Steps

- 1. ระบุ Optimal substructure ของปัญหา
- 2. พัฒนา Recursive solution
- 3. พิจารณาว่าหากเรา Make the greedy choice จะมีเพียง 1 Subproblem เหลืออยู่
- 4. พิสูจน์ความถูกต้องในการ Make the greedy choice
- 5. พัฒนา Recursive algorithm
- 6. แปลง Recursive algorithm ไปเป็น Iterative algorithm

Sorting using STL (1)

```
#include <bits/stdc++.h>
using namespace std;

// An interval has a start
// time and end time
struct Interval {
    int start, end;
};

// Compares two intervals
// according to starting times.
bool compareInterval(Interval i1, Interval i2)
{
    return (i1.start < i2.start);
}

int main()
{
    Interval arr[]
        = { { 6, 8 }, { 1, 9 }, { 2, 4 }, { 4, 7 } };
    int n = sizeof(arr) / sizeof(arr[0]);

    // sort the intervals in increasing order of
    // start time
    sort(arr, arr + n, compareInterval);

    cout << "Intervals sorted by start time : \n";
    for (int i = 0; i < n; i++)
        cout << "[" << arr[i].start << "," << arr[i].end
             << "] ";

    return 0;
}
```

Sorting using STL (2)

```
#include<bits/stdc++.h>
using namespace std;

bool sortbysec(const pair<int,int> &a,
               const pair<int,int> &b)
{
    return (a.second < b.second);
}

int main()
{
    // declaring vector of pairs
    vector< pair <int, int> > vect;

    // Initialising 1st and 2nd element of pairs with array
    int arr[] = {10, 20, 5, 40 };
    int arr1[] = {30, 60, 20, 50};
    int n = sizeof(arr)/sizeof(arr[0]);

    // Entering values in vector of pairs
    for (int i=0; i<n; i++)
        vect.push_back( make_pair(arr[i],arr1[i]) );

    // Printing the original vector(before sort())
    cout << "The vector before sort operation is:\n" ;
    for (int i=0; i<n; i++)
    {
        // "first" and "second" are used to access 1st and 2nd
element of pair respectively
        cout << vect[i].first << " "
             << vect[i].second << endl;

    }

    sort(vect.begin(), vect.end(), sortbysec);

    // Printing the sorted vector(after using sort())
    cout << "The vector after sort operation is:\n" ;
    for (int i=0; i<n; i++)
    {
        // "first" and "second" are used to access 1st and 2nd
element of pair respectively
        cout << vect[i].first << " " << vect[i].second << endl;
    }
    return 0;
}
```

Exercise

- ให้นักเรียนเขียนโปรแกรมเพื่อแก้ปัญหาการเลือกทำกิจกรรม (Activity Selection Problem) โดยใช้ Greedy method.
- ให้ลองทดสอบกับ Inputs ต่อไปนี้

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

i	1	2	3	4	5	6
s_i	5	1	3	0	5	8
f_i	9	2	4	6	7	9

Policemen catch thieves

Policemen catch thieves

- Each element in the array contains either a policeman or a thief.
- Each policeman can catch only one thief.
- A policeman cannot catch a thief who is more than K units away from the policeman (จากทั้งซ้ายและขวา).

- **We need to find the maximum number of thieves that can be caught.**

Example

- Input:
  P T T P T
  1
- Output:
  2

- Input
  P T P T T P
  3
  Output
  3

Solution

- Greedy Property:
  - Get the lowest index of policeman p and thief t. Make an allotment if |p-t| <= k and increment to the next p and t found.
  - Otherwise increment min(p, t) to the next p or t found.
  - Repeat above two steps until next p and t are found.
  - Return the number of allotments made.

```
int policeThief(char arr[], int n, int k)
{
    int res = 0;
    vector<int> thi;
    vector<int> pol;

    // store indices in the vector
    for (int i = 0; i < n; i++) {
        if (arr[i] == 'P')
            pol.push_back(i);
        else if (arr[i] == 'T')
            thi.push_back(i);
    }

    // track lowest current indices of
    // thief: thi[l], police: pol[r]
    int l = 0, r = 0;
    while (l < thi.size() && r < pol.size()) {
        // can be caught
        if (abs(thi[l] - pol[r]) <= k) {
            l++;
            r++;
            res++;
        }
        // increment the minimum index
        else if (thi[l] < pol[r]) {
            l++;
        }
        else {
            r++;
        }
    }
    return res;
}
```

Exercise

- ให้นักเรียนเขียนโปรแกรมเพื่อแก้ปัญหาตำรวจจับโจร
  - กำหนดให้ Array มาให้เพื่อแสดงตำแหน่งโจร (T) และตำแหน่งของตำรวจ (P) เช่น
    - Positions [] = {'P', 'T', 'P', 'T', 'T', 'P'} และกำหนดค่าความสามารถของการจับโจรของตำรวจ (K) โดยค่า K คือระยะห่างที่ตำรวจจะจับโจรได้
  - **<u>โจทย์</u>**ให้นักเรียนเขียนโปรแกรมเพื่อช่วยหาว่าจะมีตำรวจกี่คนที่สามารถจับโจรได้
  - ตัวอย่าง [1]Input: {'P', 'T', 'P', 'T', 'T', 'P'}, K = 3

    Output: 3

    [2] Input: {'P', 'T', 'T', 'P', 'T'}, K=1

    Output: 1
  - Greedy Choice: ให้พิจารณาตำแหน่งของตำรวจและโจรแบบ Relative เรื่อยๆ หาก index ใครน้อยกว่า ให้เพิ่ม index ของคนนั้น ในกรณีที่ตำรวจจับโจรได้ให้ขยับ index ของทั้งคู่

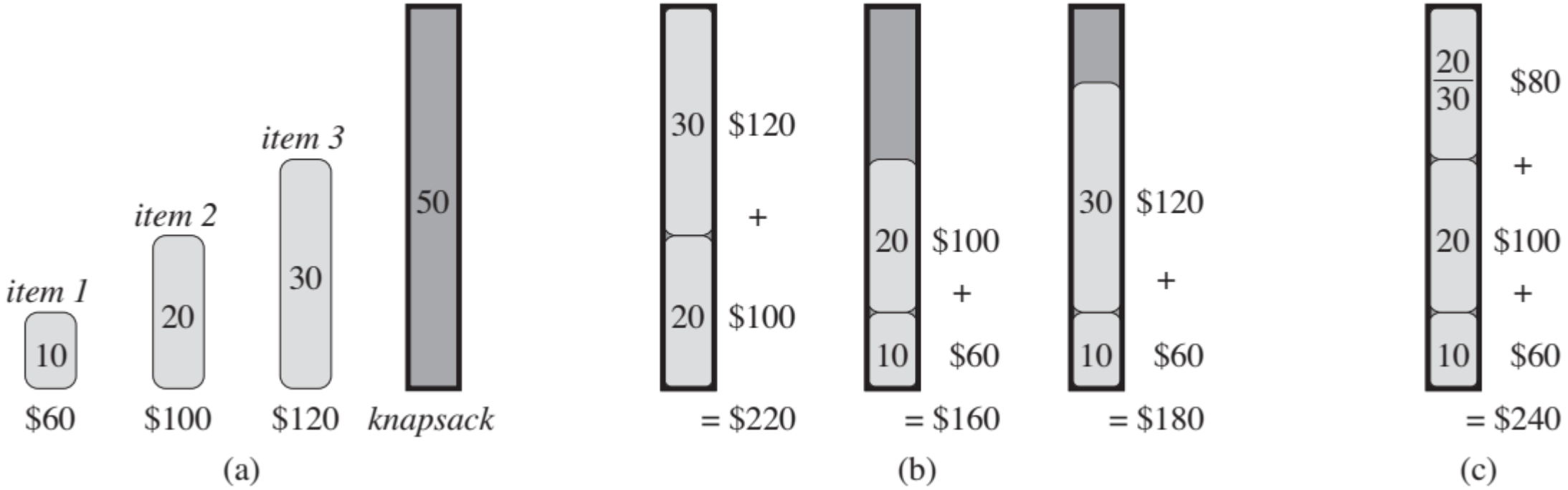
0-1 knapsack problem

0-1 knapsack problem

- A thief robbing a store finds n items.
- The ith item is worth v_i dollars and weighs w_i pounds, where v_i and w_i are integers.
- The thief wants to take as valuable a load as possible, but he can carry at most W pounds in his knapsack, for some integer W.

Which items should he take?

*In the **fractional knapsack** problem, the setup is the same, but the thief can take fractions of items, rather than having to make a binary (0-1) choice for each item.*



An example showing that the greedy strategy does not work for the 0-1 knapsack problem. **(a)** The thief must select a subset of the three items shown whose weight must not exceed 50 pounds. **(b)** The optimal subset includes items 2 and 3. Any solution with item 1 is suboptimal, even though item 1 has the greatest value per pound. **(c)** For the fractional knapsack problem, taking the items in order of greatest value per pound yields an optimal solution.

Coin Change

Coin Change

- กำหนดจำนวนเงิน V บาท และ List ของมูลค่าเหรียญที่สามารถแลกเป็นเงินทอนได้ (n ชนิด) กล่าวคือ $coinValue[i]$ บาท สำหรับเหรียญประเภท $i \in [0, \ldots, n-1]$
- **โจทย์**: หาจำนวนเหรียญที่น้อยที่สุดที่จะใช้ในการทอน
  **สมมติว่าเรามีเหรียญทุกชนิดไม่จำกัดจำนวน*
- ตัวอย่าง $n = 4, coinValue = \{25, 10, 5, 1\}$ และเราต้องการ $V = 42$ บาท
- Greedy algorithm (Greedy choice): เลือกเหรียญที่มีขนาดใหญ่สุดมาทอนก่อนเสมอ
- **42**-25 = 17 ⟶ **17**-10 = 7 ⟶ **7**-5 = **2** ⟶ 2-1 = **1** ⟶ 1-1 = 0, a total of 5 coins.

Coin Change (ต่อ)

- วิเคราะห์ปัญหา Coin changer:
  - Optimal substructure:
    - เราพบว่า 42 บาทนั้นเราใช้ 25 + 10 + 5 + 1 + 1
    - **ซึ่งเป็น Optimal 5-coin solution สำหรับ Original problem
    - **Optimal solution สำหรับ Subproblem ก็เป็นส่วนหนึ่งของคำตอบใน Optimal 5-coin solution
      - เช่น ถ้าเราต้องการทอน 17 บาท เราจะใช้เหรียญ 10 + 5 + 1 + 1 (เป็นส่วนหนึ่งของคำตอบ 42 บาท)
      - หรือ 7 เราจะใช้เหรียญ 5 + 1 +1 (เป็นส่วนหนึ่งของคำตอบ 42 บาท เช่นกัน)
  - Greedy property:
    - สำหรับทุก ๆ Amount V เราสามารถ เราสามารถเลือกเหรียญที่มีมูลค่ามากที่สุดแล้วหักลบกับ V ได้
- แต่! Greedy algorithm อาจจะไม่สามารถหา Optimal solution ให้กับทุก ๆ เซตของเหรียญ
  - เช่น {4, 3, 1} สำหรับ 6 บาท หากเราใช้ Greedy approach เราจะได้เงินเทอน 3 เหรียญคือ 4, 1, 1 แต่ Optimal solution คือ 3, 3
  - General version => Brute force, Dynamic programing

Huffman codes

Huffman codes

- Huffman codes compress data very effectively: savings of 20% to 90% are typical, depending on the characteristics of the data being compressed.
- We consider the data to be a sequence of characters.
- Huffman's greedy algorithm uses a table giving how often each character occurs (i.e., its frequency) to build up an optimal way of representing each character as a **binary string**.
- Suppose we have a 100,000-character data file that we wish to store compactly.

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

Huffman codes

- Here, we consider the problem of designing a **binary character code** (or **code** for short) in which each character is represented by a unique binary string, which we call a **codeword**.
- If we use a *fixed-length code*, we need 3 bits to represent 6 characters (A data file of 100,000 characters):
  - a = 000, b = 001, …, f=101.
  - This method requires 300,000 bits to code the entire file. Can we do better?
- A **variable-length code** can do considerably better than a fixed-length code, by giving frequent characters short codewords and infrequent characters long codewords.
  - It takes 224, 000 bits to repr

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

).

Prefix codes

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

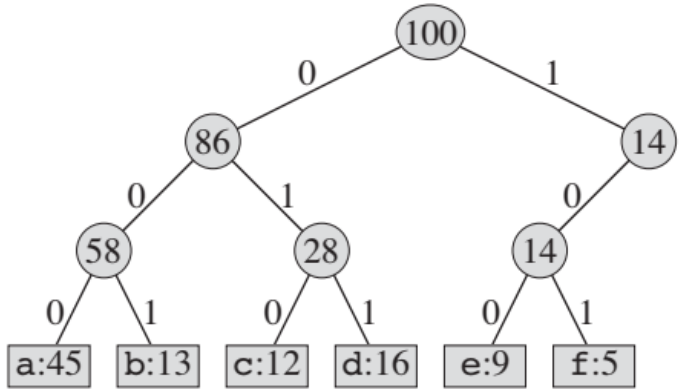
- **prefix codes**: codes in which no codeword is also a prefix of some other codeword.
- We code the 3-character file **"abc"** as 0•101•100, • denotes concatenation.
- Since no codeword is a prefix of any other, the codeword that begins an encoded file is unambiguous.
- We can simply identify the initial codeword, translate it back to the original character, and repeat the decoding process on the remainder of the encoded file.
- According to the table, the string **001011101** parses uniquely as 0•0•101•1101, which decodes to **aabe**.
- The decoding process needs a convenient representation for the prefix code so that we can easily pick off the initial codeword.
  - A **binary tree** whose leaves are the given characters provides one such representation.

Prefix codes

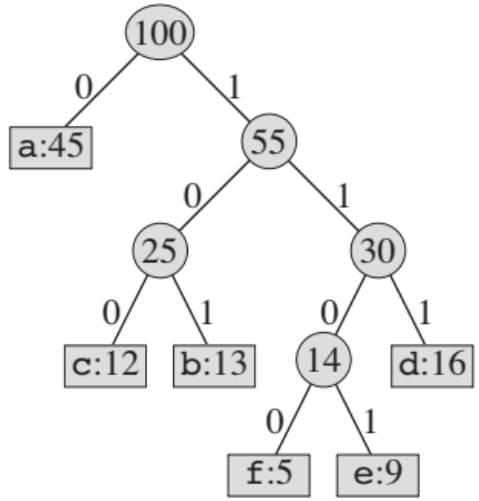
	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

- We interpret the binary codeword for a character as *the simple path from the root to that character*, where 0 means “go to the left child” and 1 means “go to the right child.”
- An **optimal code** for a file is always represented by a full binary tree, in which every nonleaf node has two children.

Fixed-length codes



Variable-length codes



Prefix codes

- We can say that if C is the alphabet from which the characters are drawn and all character frequencies are positive, then the tree for an optimal prefix code has exactly $|C|$ **leaves**.
  - One for each letter of the alphabet, and exactly $|C| - 1$ **internal nodes.**
- Given a tree T corresponding to a prefix code, we can easily compute the *number of bits* required to encode a file.
- For each character c in the alphabet C, let the attribute $c.freq$ denote the frequency of c in the file and let $d_T(c)$ denote the depth of c's leaf in the tree. Note that $d_T(c)$ is also the length of the codeword for character c.
- The number of bits required to encode a file is thus

$$B(T) = \sum_{c \in C} c.freq \cdot d_T(c)$$

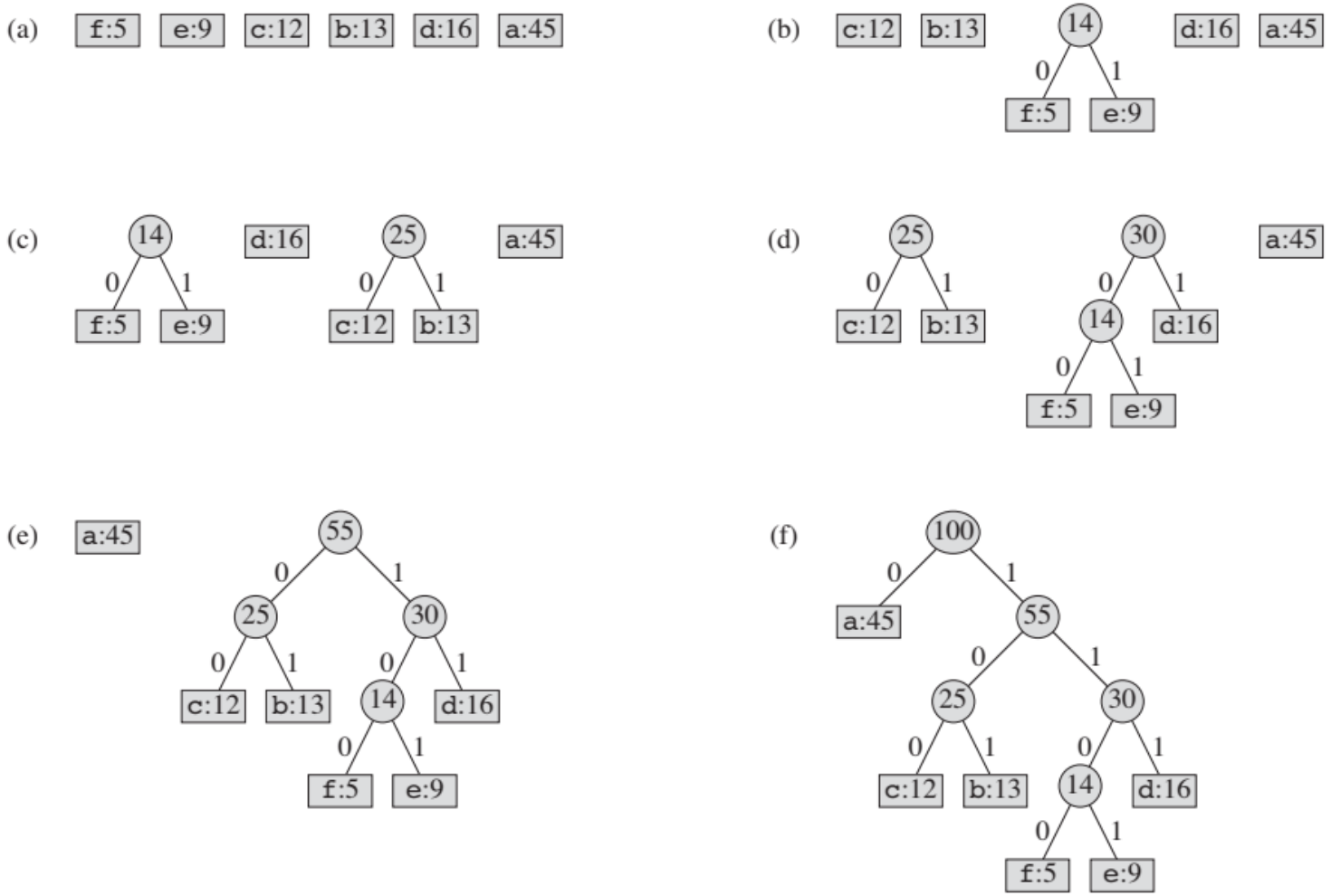
which we define as the cost of the tree T.

Constructing a Huffman code

- Huffman invented a **greedy algorithm** that constructs an **optimal prefix code** called a **Huffman code**.
- C is a set of n characters.

```
HUFFMAN(C)
1  n = |C|
2  Q = C
3  for i = 1 to n − 1
4      allocate a new node z
5      z.left = x = EXTRACT-MIN(Q)
6      z.right = y = EXTRACT-MIN(Q)
7      z.freq = x.freq + y.freq
8      INSERT(Q, z)
9  return EXTRACT-MIN(Q)      // return the root of the tree
```

Constructing a Huffman code



Complexity Analysis

- Q is implemented as a binary min-heap. For a set of C of n characters.
- Initializing Q takes $O(n)$
- In the loop starting at Line 3, it executes n-1 times, and each time requires $O(\lg n)$, so the loop contributes $O(n \lg n)$ to the running time.
- Thus, the total running time of HUFFMAN on a set of n characters is $O(n \lg n)$.

Internal covering problem

Internal covering problem

- กำหนด Array A ที่ประกอบด้วย N Intervals (N ช่วง) และ Target interval X
- **โจทย์** ให้หาจำนวนของ Intervals ที่น้อยที่สุดที่จาก A ที่สามารถครอบคลุมทั้งช่วง X ได้
- เช่น

  [1] Input: A[] = {{1, 3}, {2, 4}, {2, 10}, {2, 3}, {1, 1}}, X = {1, 10}

  Output: 2

  {1, 3} and {2, 10} can be selected. Therefore, the points in the range [1, 3] are covered by the interval {1, 3} and the points in the range [4, 10] are covered by the interval {2, 10}.

-

  [2] Input: A[] = {{2, 6}, {7, 9}, {3, 5}, {6, 10}} , X = {1, 4}

  Output: -1

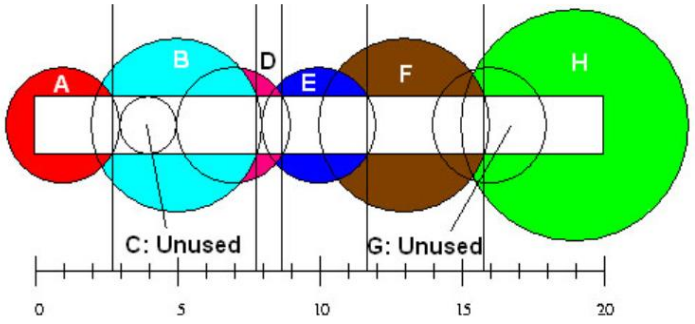
- Greedy approach: หากพิจารณาจะพบว่า Greedy choice คือ {u, v} ที่เป็นไปได้คือ u <= p และ v นั้นมากที่สุดเท่าที่เป็นไปได้

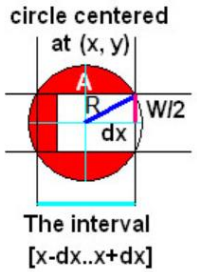
Internal covering problem

- Algorithm: (กำหนดให้ช่วง X={M, N})
  - Sort A จากน้อยไปมากด้วยจุดเริ่ม
  - กำหนดให้ตัวแปร Start = M และ End = Start-1
  - สร้างตัวแปร cnt แทนการนับจำนวนของช่วงที่เลือก
  - วนลูปใน A; A[0]…A[i]…A[n-1]
  - ถ้า Start >= A[i].Start ให้อัปเดตค่า End ด้วย max(End, A[i].End)
  - ถ้า Start < A[i].Start ให้อัปเดตค่า End ด้วย A[i].End และ Start = A[i].Start พร้อมกับเพิ่มค่า cnt ด้วยจำนวน 1
  - ออกจากลูปเมื่อ A[i].Start > End หรือ A[i].End > End
  - คืนค่า -1 เมื่อวนลูปครอบแล้ว End < A[i].End มิเช่นนั้นคือ cnt
- Time Complexity: O(N*log N)
- Auxiliary Space: O(1)
- More efficient approach: Minimum number of jumps to reach end

Watering Grass (UVa 10382)

- กำหนดให้มี n Sprinklers ติดตั้งตรงกลางสนามหญ้าที่มีความยาว L เมตร กว้าง W เมตร
- โดย Sprinklers แต่ละตัวจะถูกระบุตำแหน่งจากระยะห่างจากซ้ายสุดของสนามหญ้า และ กำหนดระยะเหวี่ยงน้ำของ Sprinkler แต่ละตัว
- **โจทย์:** จำนวน Sprinklers ที่น้อยที่สุดที่ใช้เพื่อจะรดน้ำให้ครอบคลุมทั้งสนามหญ้าเป็นคือกี่ตัว
- ตัวอย่าง จากรูปด้านล่างมี Sprinklers ทั้งหมด 6 ตัวคือ {A,B,C,D,E,F} มี 2 Sprinkler ที่ไม่ได้ใช้งานคือ {C, G}
- Brute force คือ ทดลองทุก ๆ Subset ที่เป็นไปได้ของการใช้ Sprinklers (=> Time limit exceeded??)





$$dx = \mathrm{sqrt}(R^2 - (W/2)^2)$$

เราสามารถใช้ความสัมพันธ์นี้แปลงเป็นช่วงได้ ... [x-dx, x+dx]

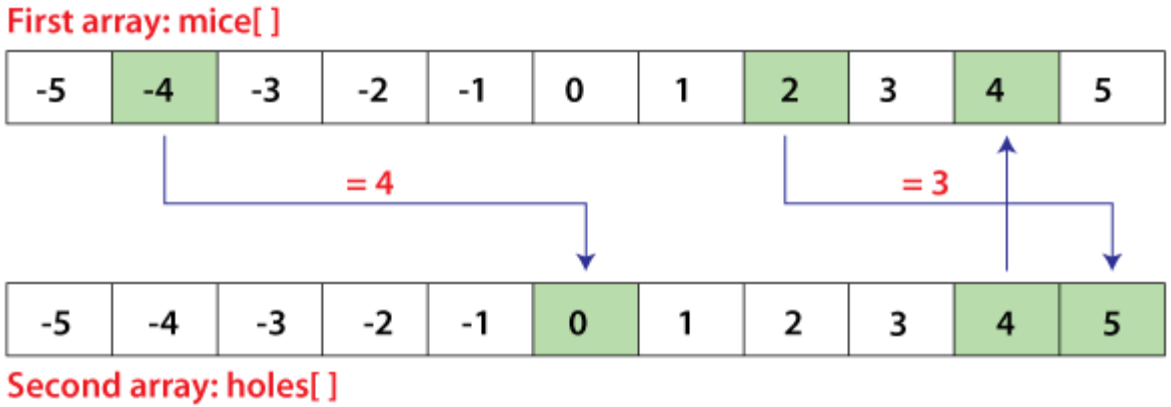
แบบฝึกหัด (ฝึกด้วยตนเอง)

- 10020 - Minimal Coverage
- 10340 - All in All
- 10440 - Ferry Loading (II)
- Sorting a Three-Valued Sequence (IOI'96)
  - https://ioinformatics.org/files/ioi1996problem4.pdf

Assign Mice to Holes

Assign Mice to Holes

- There are N Mice and N holes are placed in a straight line.
- Each hole can accommodate only 1 mouse.
- A mouse can stay at his position, move one step right from x to x + 1, or move one step left from x to x -1.
- Any of these moves consumes 1 minute.
- Assign mice to holes so that the time when the last mouse gets inside a hole is minimized.



https://www.javatpoint.com/java-programming-challenges

Algorithm

- Sort the positions of the mice in ascending order preferably.
- Sort the positions of the holes
- Loop i = 1 to N: Update answers according to the value of |mice(i) - hole(i)|. It should be maximum of all differences.
- Let i1 < i2 be the positions of two mice and let j1 < j2 be the positions of two holes.

แบบฝึกหัด

- เขียนโปรแกรม
  - Input
    - 1. ตำแหน่งของหนู (Mice) เป็นตัวเลขจำนวนเต็ม N ตัว
    - 2. ตำแหน่งของรูที่หนูอยู่ได้ เป็นตัวเลขจำนวนเต็ม N ตัว
  - Output:
    - หาเวลาที่น้อยที่สุดที่จะให้หนูตัวสุดท้ายลงรู
- ตัวอย่าง
  - Input
    4 -4 2
    4 0 5
  - Output
    4
- ตัวอย่าง
  - Input
    10, -79, -79, 67, 93, -85, -28, -94
    -2, 9, 69, 25, -31, 23, 50, 78
  - Output
    102

แบบฝึกหัด

- ทำโจทย์ข้อ เหรียญโอลิมปิก Medal (TOI'15)
- https://programming.in.th/tasks/toi15_medal

References

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. Introduction to Algorithms, Third Edition (3rd. ed.). The MIT Press.
- https://www.otfried.org/courses/cs500/slides-approx.pdf
- Competitive Programming 3
- https://www.geeksforgeeks.org/greedy-algorithms/?ref=lbp