

# เอกสารประกอบการอบรม

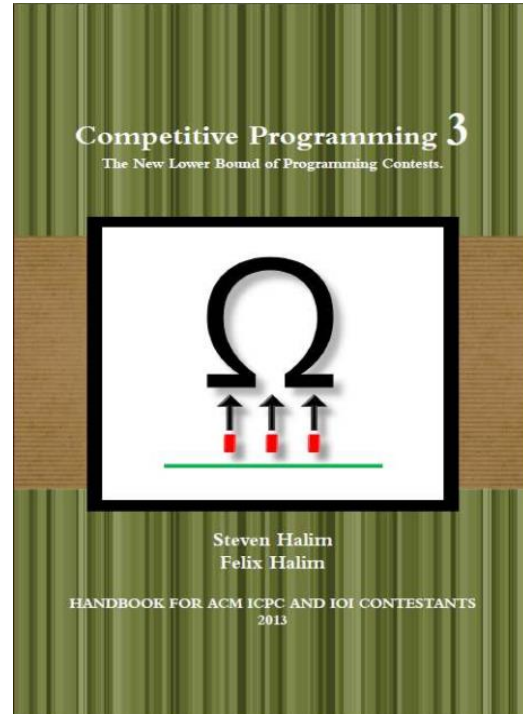
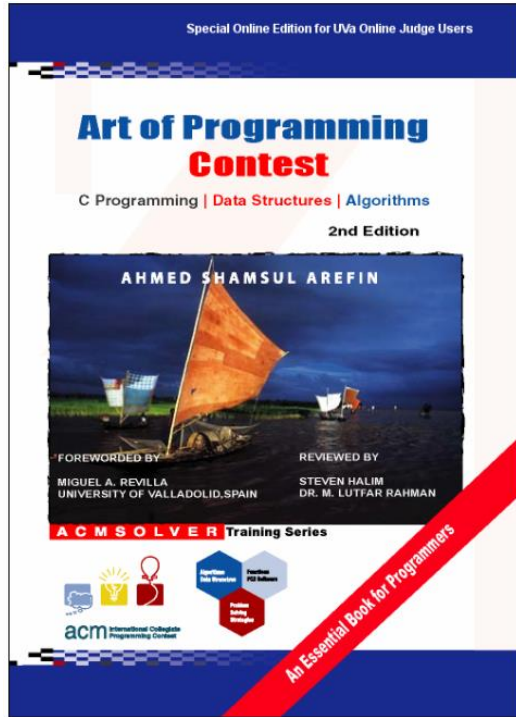
## Competitive Programming (Tips)

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

โดย  
สาขาวิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์และเทคโนโลยี  
มหาวิทยาลัยธรรมศาสตร์




# หนังสือแนะนำ



Antti Laaksonen

## Guide to Competitive Programming

Learning and Improving Algorithms Through Contests

 Springer

## Competitive Programmer's Handbook

Antti Laaksonen

Draft July 3, 2018

More ? ...

# เว็บไซต์แนะนำ

- <https://beta.programming.in.th/>
- <https://www.geeksforgeeks.org/how-to-begin-with-competitive-programming/>
- <https://www.geeksforgeeks.org/competitive-programming-a-complete-guide/>
- <https://www.youtube.com/watch?v=bVKHRtafgPc>
- [https://docs.google.com/spreadsheets/d/1iJZWP2nS\\_OB3kCTjq8L6TrJJ4o-5lhxD0yTaocSYc-k/edit?fbclid=IwAR2Csc64sQKIW6lO\\_FWeqRqF\\_3SbhwW7bT2pk-aHUW3detDkv6eQvjcg--c#gid=1696678792](https://docs.google.com/spreadsheets/d/1iJZWP2nS_OB3kCTjq8L6TrJJ4o-5lhxD0yTaocSYc-k/edit?fbclid=IwAR2Csc64sQKIW6lO_FWeqRqF_3SbhwW7bT2pk-aHUW3detDkv6eQvjcg--c#gid=1696678792)
- <https://www.codechef.com/tags/problems>

# Interesting FAQs

- [https://docs.google.com/spreadsheets/d/1iJZWP2nS\\_OB3kCTjq8L6TrJJ4o-5lhxD0yTaocSYc-k/edit?fbclid=IwAR2Csc64sQKIW6IO\\_FWegRqF\\_3SbhwW7bT2pk-aHUW3detDkv6eQvjcg--c#gid=1696678792](https://docs.google.com/spreadsheets/d/1iJZWP2nS_OB3kCTjq8L6TrJJ4o-5lhxD0yTaocSYc-k/edit?fbclid=IwAR2Csc64sQKIW6IO_FWegRqF_3SbhwW7bT2pk-aHUW3detDkv6eQvjcg--c#gid=1696678792)
- <https://earthshakira.github.io/a2oj-clientside/server/Ladders.html>

# ประเภทของปัญหา (Problem Categorization)

- พยายามฝึกหัดแก้ปัญหาให้ได้อย่างน้อยประเภทละไม่กี่ข้อ แต่พยายามแก้ให้ได้หลาย ๆ ประเภท จะช่วยให้เราพัฒนาทักษะการแก้ปัญหา (Problem skill sets)
- <https://cses.fi/problemset/>
- <https://cpbook.net/methodstosolve>



# Convention (C++)

- Commonly used datatypes (possibly):

```
typedef long long ll;  
typedef pair<int, int> ii;  
typedef vector<ii> vii;  
typedef vector<int> vi;  
#define INF 1000000000
```

- Common memset

```
//Common memset settings  
memset(memo, -1, sizeof memo);           // initialize DP memoization table with -1  
memset(arr, 0, sizeof arr);              // to clear array of integers
```

- Others ...

```
// ans = a ? b : c; // to simplify: if (a) ans = b; else ans = c;  
// ans += val; // to simplify: ans = ans + val; and its variants  
// index = (index + 1) % n; // index++; if (index >= n) index = 0;  
// index = (index + n - 1) % n; // index--; if (index < 0) index = n - 1;  
// int ans = (int)((double)d + 0.5); // for rounding to nearest integer  
// ans = min(ans, new_computation); // min/max shortcut  
// alternative form but not used in this book: ans <?= new_computation;  
// some code use short circuit && (AND) and || (OR)
```

# Competitive Programming

การโปรแกรมมิ่งเพื่อการแข่งขันนั้นคือ:

“Given **well-known Computer Science (CS) problems**, solve them **as quickly as possible!**”

**Well-known Computer Science (CS) problems:** หมายถึงปัญหาทาง CS ที่สามารถถูกแก้ได้แล้ว ไม่ใช่ปัญหาเชิงวิจัยที่ยังไม่สามารถหาคำตอบได้

> ดังนั้นจะแก้ปัญหเหล่านี้ได้เราจำเป็นต้องฝึกและพัฒนาทักษะทาง CS ของเราไปจนถึงระดับหนึ่งที่จะสามารถ Implement code เพื่อแก้ปัญหได้

**As quickly as possible!:** จุดมุ่งหมายที่สำคัญนอกจากการแก้ปัญหได้แล้วคือความเร็วของวิธีการที่ใช้



# พิจารณาปัญหาต่อไปนี้

- UVa Online Judge [Problem number 10911]

## Abridged Problem Description:

Let  $(x, y)$  be the coordinates of a student's house on a 2D plane. There are  $2N$  students and we want to **pair** them into  $N$  groups. Let  $d_i$  be the distance between the houses of 2 students in group  $i$ . Form  $N$  groups such that  $cost = \sum_{i=1}^N d_i$  is **minimized**. Output the minimum  $cost$ . Constraints:  $1 \leq N \leq 8$  and  $0 \leq x, y \leq 1000$ .

## Sample input:

$N = 2$ ; Coordinates of the  $2N = 4$  houses are  $\{1, 1\}$ ,  $\{8, 6\}$ ,  $\{6, 8\}$ , and  $\{1, 3\}$ .

## Sample output:

$cost = 4.83$ .

# พิจารณาปัญหาต่อไปนี

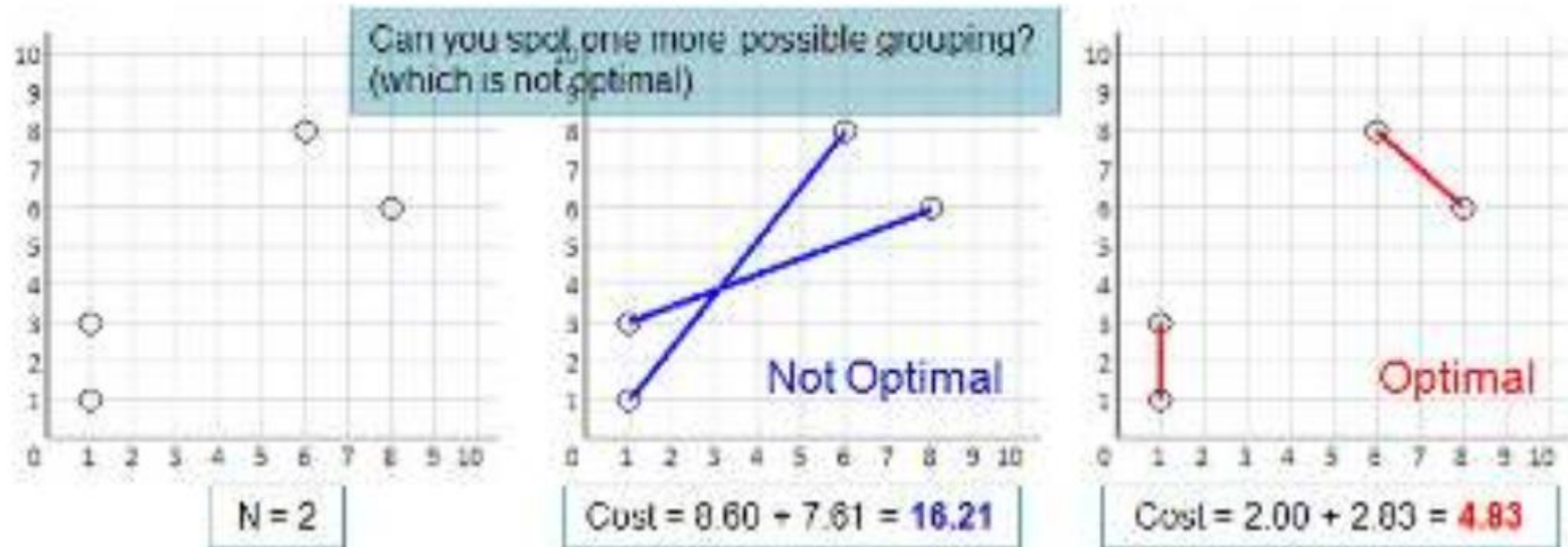


Figure 1.1: Illustration of UVa 10911 - Forming Quiz Teams

# พิจารณาปัญหาต่อไปนี้

- สิ่งที่เกิดขึ้น

- [\*] บางคนอาจจะอ่านปัญหานี้แล้วรู้สึกงง (อาจจะเพิ่งเจอปัญหาเป็นครั้งแรก)
- [\*] พยายามคิดหาคำตอบ (บางครั้งอาจจะลอง Coding) แต่พบว่า Algorithm ให้คำตอบที่ยังไม่ถูกต้อง
- [\*] บางคนอาจจะลองใช้วิธีการ Backtracking แต่พบว่ามันมีกรณีที่ต้องตรวจสอบมากเกินไปและเกิด Time Limited Exceeded (TLE) \* (แต่ถ้าใช้ Backtracking with pruning อาจจะช่วยแก้ปัญหานี้ได้)
- [\*\*] บางคนอาจจะพบว่าน่าจะเป็นปัญหาที่เคยพบมาก่อน (แต่ยังไม่เคยแก้ปัญหานี้ได้มาก่อน) จึงข้ามไปลองปัญหาอื่น
- [\*\*\*] บางคนรู้ว่านี่คือปัญหาที่ยาก (Hard problem) => “minimum weight perfect matching on a small general weighted graph” และเนื่องจาก Input ที่ให้มามันไม่ใหญ่มากจึงสามารถใช้ Dynamic Programming ในการแก้ปัญหานี้ได้
  - (ต่างคนอาจจะใช้เวลาที่ต่างกันออกไป)

# Tips

- Tip 1: Type Faster
- Tip 2: พยายามดูให้ออกกว่าปัญหาที่เรา กำลังจะแก้เป็นปัญหาประเภทไหน
  - ตัวอย่างของปัญหา (Recursive อยู่ใน DP หรือ Backtracking)

No	Category	In This Book	Frequency
1.	Ad Hoc	Section 1.4	1-2
2.	Complete Search (Iterative/Recursive)	Section 3.2	1-2
3.	Divide and Conquer	Section 3.3	0-1
4.	Greedy (usually the original ones)	Section 3.4	0-1
5.	Dynamic Programming (usually the original ones)	Section 3.5	1-3
6.	Graph	Chapter 4	1-2
7.	Mathematics	Chapter 5	1-2
8.	String Processing	Chapter 6	1
9.	Computational Geometry	Chapter 7	1
10.	Some Harder/Rare Problems	Chapter 8-9	1-2
		Total in Set	8-17 ( $\approx \leq 12$ )

# Tips (ต่อ)

- จากการที่เราได้เรียน Algorithms ต่าง ๆ มาแล้วเราอาจจะจำแนกปัญหาออกได้เป็น 3 ประเภทดังนี้

No	Category	Confidence and Expected Solving Speed
A.	I have solved this type before	I am sure that I can re-solve it again (and fast)
B.	I have seen this type before	But that time I know I cannot solve it yet
C.	I have not seen this type before	See discussion below

- การฝึกฝนเพื่อไปแข่งขันเราต้องพยายามเพิ่ม Type A และลด Type B
  - กล่าวคือ เราต้อง**เรียนรู้หรือรู้จัก Algorithms** ให้ได้มากที่สุด และ พัฒนา**ฝีมือการโปรแกรมมิ่ง** ให้สามารถเขียนโปรแกรมเพื่อแก้ปัญหา Classic ต่าง ๆ ได้อย่างง่ายดาย
  - และเหนือไปกว่านั้นการที่จะชนะการแข่งขัน เราต้องพัฒนา Problem solving skills
    - การลดรูปปัญหาที่ไม่เคยพบมาก่อนไปเป็นปัญหาที่เราแก้ได้ (Known problem)
    - พยายามวิเคราะห์คำใบ้ที่ให้มา และ คุณสมบัติเฉพาะของโจทย์
    - \*\* พยายามแก้ปัญหาจากวิธีที่ไม่ Obvious
    - จึงทำให้เราสามารถเอาชนะโจทย์ในการแข่งขันได้

# ทดสอบการจำแนกปัญหา/ บางปัญหาไหนเราจะหยาบมา Discuss กัน

UVa	Title	Problem Type	Hint
10360	Rat Attack	Complete Search or DP	Section 3.2
10341	Solve It		Section 3.3
11292	Dragon of Loowater		Section 3.4
11450	Wedding Shopping		Section 3.5
10911	Forming Quiz Teams	DP with bitmask	Section 8.3.1
11635	Hotel Booking		Section 8.4
11506	Angry Programmer		Section 4.6
10243	Fire! Fire!! Fire!!!		Section 4.7.1
10717	Mint		Section 8.4
11512	GATTACA		Section 6.6
10065	Useless Tile Packers		Section 7.3.7

Table 1.3: Exercise: Classify These UVa Problems

# Tips (ต่อ)

- Tip 3: Do Algorithm Analysis

- เมื่อโจทย์กำหนด Maximum input bound มาให้ มี Algorithms ไหนบ้างสามารถรันได้/รันไม่ได้ภายในเวลาที่กำหนด
- หรือเราอาจจะพบว่าอาจจะมีหลาย Algorithms ที่สามารถแก้ปัญหาได้
  - บาง Algorithm ไม่สามารถให้ผลลัพธ์ที่ถูกต้อง
  - บาง Algorithm ช้าเกินไป
  - วิธีการที่ดีวิธีการหนึ่งคือ List หลาย ๆ Algorithms ที่เป็นไปได้แล้วเลือกอันที่ง่ายที่สุด (เร็วมากพอที่จะผ่าน Time และ Memory limits แต่ยังคง Produce ผลลัพธ์ที่ถูกต้องได้)

# Tips (ต่อ)

- คอมพิวเตอร์ทั่วไปสามารถรันประมาณ 100M ( $10^8$ ) คำสั่งได้ในไม่กี่วินาที
- เราสามารถใช้ข้อมูลนี้คาดการณ์ว่า Algorithm ที่เราเลือกมันจะสามารถรันได้ภายในเวลาที่กำหนดหรือไม่ จาก Maximum bound ของ Input
  - เช่น หาก Input Size 100K ( $10^5$ ) Algorithm ที่ใช้  $O(n^2) = 10^{10}$  ไม่น่าจะรันได้ทันเวลา (อาจจะใช้หลายร้อยวินาที) => เราอาจจะต้องเลือก Algorithm ที่มีประสิทธิภาพมากกว่านั้น
  - ในขณะที่ถ้าสมมติว่าคุณใช้  $O(n \log n)$  Algorithm จะใช้ประมาณ  $10^5 \log_2 10^5$  ซึ่งน่าจะต่ำกว่า 1 วินาที จะทำให้เราสามารถแก้ปัญหาได้ทันเวลาได้
- การพิจารณา Maximum bound ของ Input จึงเป็นสิ่งสำคัญมากที่จะช่วยให้คุณตัดสินใจว่า Solution ของคุณเหมาะสมหรือไม่
  - สมมติว่าคุณมี Algorithm ที่รันที่  $O(n^4)$  แต่ในโจทย์กำหนด Maximum bound  $n \leq 50$  Algorithm นี้ก็อาจจะแก้ปัญหาได้เพราะ ( $50^4 = 6.25M$ ) ซึ่งใช้เวลารันประมาณในหน่วยวินาทีเท่านั้น
  - นอกจากโจทย์แล้ว ควรทำความเข้าใจกับ Maximum bound ของ Input ด้วยจะทำให้การตัดสินใจเลือกปัญหาเร็วขึ้น



# Tips (ข้อ)

Familiarity with these bounds:

- $2^{10} = 1,024 \approx 10^3$ ,  $2^{20} = 1,048,576 \approx 10^6$ .
- 32-bit signed integers (`int`) and 64-bit signed integers (`long long`) have upper limits of  $2^{31} - 1 \approx 2 \times 10^9$  (safe for up to  $\approx 9$  decimal digits) and  $2^{63} - 1 \approx 9 \times 10^{18}$  (safe for up to  $\approx 18$  decimal digits) respectively.
- Unsigned integers can be used if only non-negative numbers are required. 32-bit unsigned integers (`unsigned int`) and 64-bit unsigned integers (`unsigned long long`) have upper limits of  $2^{32} - 1 \approx 4 \times 10^9$  and  $2^{64} - 1 \approx 1.8 \times 10^{19}$  respectively.
- If you need to store integers  $\geq 2^{64}$ , use the Big Integer technique (Section 5.3).
- There are  $n!$  permutations and  $2^n$  subsets (or combinations) of  $n$  elements.
- The best time complexity of a comparison-based sorting algorithm is  $\Omega(n \log_2 n)$ .
- Usually,  $O(n \log_2 n)$  algorithms are sufficient to solve most contest problems.
- The largest input size for typical programming contest problems must be  $< 1M$ . Beyond that, the time needed to read the input (the Input/Output routine) will be the bottleneck.
- A typical year 2013 CPU can process  $100M = 10^8$  operations in a few seconds.

2022??

# Tips (ต่อ)

- หลาย ๆ คนอาจคิดว่าเราน่าจะเริ่มจากการ Coding เลย ?
- แต่คนเขียนหนังสือเล่มนี้แนะนำว่า **ไม่ควรจะ Code** จนกระทั่งคุณมั่นใจว่า Algorithm ของคุณจะถูกและเร็วมากพอ

$n$	Worst AC Algorithm	Comment
$\leq [10..11]$	$O(n!), O(n^6)$	e.g. Enumerating permutations (Section 3.2)
$\leq [15..18]$	$O(2^n \times n^2)$	e.g. DP TSP (Section 3.5.2)
$\leq [18..22]$	$O(2^n \times n)$	e.g. DP with bitmask technique (Section 8.3.1)
$\leq 100$	$O(n^4)$	e.g. DP with 3 dimensions + $O(n)$ loop, ${}_nC_{k=4}$
$\leq 400$	$O(n^3)$	e.g. Floyd Warshall's (Section 4.5)
$\leq 2K$	$O(n^2 \log_2 n)$	e.g. 2-nested loops + a tree-related DS (Section 2.3)
$\leq 10K$	$O(n^2)$	e.g. Bubble/Selection/Insertion Sort (Section 2.2)
$\leq 1M$	$O(n \log_2 n)$	e.g. Merge Sort, building Segment Tree (Section 2.3)
$\leq 100M$	$O(n), O(\log_2 n), O(1)$	Most contest problem has $n \leq 1M$ (I/O bottleneck)

Table 1.4: Rule of thumb time complexities for the 'Worst AC Algorithm' for various single-test-case input sizes  $n$ , assuming that your CPU can compute  $100M$  items in 3s.

**\*\*Algorithm ยิ่งเร็ว ยิ่งแรง การ Implement อาจจะยากและไม่ตรงไปตรงมา**

# Tips (ต่อ)



- Tip 4: Practice and More Practice
- Competitive programmers เหมือนนักกีฬา ต้องซ้อมบ่อย ๆ และ “Keep programming fit”
- The University of Valladolid Online Judge (<https://onlinejudge.org/>) มีโจทย์มากมาย  
ACM Contest problems (Locals, Regionals, and up to World Finals)
  - <http://livearchive.onlinejudge.org/>
  - <http://train.usaco.org/usacogate>
  - <http://www.spoj.pl/>

# Tips (ต่อ)

- ฟีก code ให้รวดเร็วโดยใช้ Template:

```
#include <bits/stdc++.h> // Include every standard library
using namespace std;
```

```
typedef long long LL;
typedef pair<int, int> pii;
typedef pair<LL, LL> pll;
typedef pair<string, string> pss;
typedef vector<int> vi;
typedef vector<vi> vvi;
typedef vector<pii> vii;
typedef vector<LL> vl;
typedef vector<vl> vvl;
```

```
double EPS = 1e-9;
int INF = 1000000005;
long long INFF = 1000000000000000005LL;
double PI = acos(-1);
int dirx[8] = { -1, 0, 0, 1, -1, -1, 1, 1 };
int diry[8] = { 0, 1, -1, 0, -1, 1, -1, 1 };
```

<https://www.geeksforgeeks.org/c-methods-of-code-shortening-in-competitive-programming/?ref=rp>

**Macro:** คือการที่ String บางส่วนของ Code จะถูกเปลี่ยนแปลงก่อนเกิดการ Compiled

**\*\*ใช้ #define**

```
#define FOR(a, b, c) for (int(a) = (b); (a) < (c); ++(a))
#define FORN(a, b, c) for (int(a) = (b); (a) <= (c); ++(a))
#define FORD(a, b, c) for (int(a) = (b); (a) >= (c); --(a))
#define FORSQ(a, b, c) for (int(a) = (b); (a) * (a) <= (c); ++(a))
#define FORC(a, b, c) for (char(a) = (b); (a) <= (c); ++(a))
#define FOREACH(a, b) for (auto&(a) : (b))
#define REP(i, n) FOR(i, 0, n)
#define REPN(i, n) FORN(i, 1, n)
#define MAX(a, b) a = max(a, b)
#define MIN(a, b) a = min(a, b)
#define SQR(x) ((LL)(x) * (x))
#define RESET(a, b) memset(a, b, sizeof(a))
#define fi first
#define se second
#define mp make_pair
#define pb push_back
#define ALL(v) v.begin(), v.end()
#define ALLA(arr, sz) arr, arr + sz
#define SIZE(v) (int)v.size()
#define SORT(v) sort(ALL(v))
#define REVERSE(v) reverse(ALL(v))
#define SORTA(arr, sz) sort(ALLA(arr, sz))
#define REVERSEA(arr, sz) reverse(ALLA(arr, sz))
#define PERMUTE next_permutation
#define TC(t) while (t--)
```

# Adhoc Problems

- ปัญหา Ad hoc เป็นปัญหาที่ไม่ได้จัดอยู่ในปัญหา Type อื่น ๆ เนื่องจากว่าคำอธิบายปัญหา และ Solution นั้น “Unique”



<b>Optimization</b> <b>Common:</b> <ul style="list-style-type: none"> <li>- Fake/dfs</li> <li>- DP/greedy/bf</li> <li>- Binary Search/TS</li> <li>- Branch &amp; Bound</li> <li>- RMQ/LCA</li> <li>- Line sweep</li> <li>- AlgoX</li> </ul> <b>Minimization</b> <ul style="list-style-type: none"> <li>- MCMF</li> <li>- Min cut / vertex</li> <li>- MST / Dijkstra</li> <li>- Chull / mec</li> </ul> <b>Maximization</b> <ul style="list-style-type: none"> <li>- Max flow / MCMF</li> <li>- Max Independent Set</li> <li>- Kruskal Reverse</li> <li>- LIS/GCD</li> </ul>	<b>DP</b> <b>General</b> <ul style="list-style-type: none"> <li>- State representation(s)</li> <li>- Diff sub-states calls?</li> <li>-- move to state</li> <li>- Cycles?</li> <li>-- Depth?</li> <li>-- Dijkstra / Bfs</li> <li>-- Dec(rement)-inc-dec</li> </ul> <b>Types</b> <ul style="list-style-type: none"> <li>- Restricted / Range</li> <li>- Counting</li> <li>- Tree / Partitioning</li> <li>- Extending table</li> </ul> <b>Concerns</b> <ul style="list-style-type: none"> <li>- Base case order</li> <li>- Search space?</li> <li>-- Constrained pars</li> <li>- Redundant pars</li> </ul> <b>States</b> <ul style="list-style-type: none"> <li>- Canonical states?</li> <li>- Local Minima</li> <li>- Small substates cnt?</li> <li>- Large pars</li> <li>- Reduces fast? (e.g. /)</li> </ul> <b>Counting Problems</b> <ul style="list-style-type: none"> <li>- DP</li> <li>- Combinations / Perms</li> <li>- Inclusion-exclusion</li> <li>- Graph Power</li> </ul>	<b>Data Structures</b> <ul style="list-style-type: none"> <li>- Set/Heap /DisjointSets</li> <li>- BIT</li> <li>- Segmentation Tree</li> <li>- Treab, KDT</li> <li>- LCA/RMQ</li> <li>- Hashing</li> <li>- Interval Compression</li> <li>- Quad Tree</li> </ul> <b>Graph Algorithms</b> <ul style="list-style-type: none"> <li>- MST: Kruskal / Prime</li> <li>- Dijkstra / Topological</li> <li>- Convex Hull / Floyd</li> <li>- Max Flow/Min Cut</li> <li>- Max Matching</li> <li>- Max Indep Set</li> <li>- Min path/vertex cover</li> <li>- Bellman / DConsts</li> <li>- Euler/Postman</li> </ul> <b>String Algorithms</b> <ul style="list-style-type: none"> <li>- Trie</li> <li>- Permutation Cycles</li> <li>- LIS / LCS</li> <li>- Polynomial Hashing</li> <li>- KMP / Aho Corasick</li> <li>- Suffix tree/array</li> </ul>	<b>Mathematics</b> <ul style="list-style-type: none"> <li>- GCD/LCM/Phi/Mob</li> <li>- NIM/Grundy/Chinese</li> <li>- Seive/Factorization</li> <li>- System of Linear Eqs</li> <li>- Determinant</li> <li>- Simplex/ Pick's Theo</li> <li>- Numerical Integration</li> <li>- Matrix Power</li> <li>- Closed Form</li> <li>- Pigeon Hole</li> <li>- Triangle inequality</li> <li>- Voronoi diagram</li> </ul> <b>Adhock Algorithms</b> <ul style="list-style-type: none"> <li>- Greedy</li> <li>- Line Sweep</li> <li>- Sliding Window</li> <li>- Canonical Form</li> <li>- Grid Compression</li> <li>- Constructive algos</li> <li>- Test cases driven</li> <li>- Randomization</li> <li>- Time cut-off</li> <li>- Stress Test &amp; Observe</li> </ul> <b>Decision Algorithms</b> <ul style="list-style-type: none"> <li>- 2SAT</li> <li>- Difference constraints</li> <li>- Grundy</li> <li>- Bipartite?</li> </ul>
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

# 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