// Cheatsheet by Qoolqas

Version 1.1

Table of Contents

[Library and Macros 2](#_Toc405227051)

[Data Structure 3](#_Toc405227052)

[Fenwick Tree (BIT) 3](#_Toc405227053)

[Contoh Pemakaian 3](#_Toc405227054)

[Graph 2](#_Toc405227055)

[Minimum Spanning Tree 2](#_Toc405227056)

[Minimum Spanning Tree – Variants 2](#_Toc405227057)

[- Partial ‘minimum’ Spanning tree 2](#_Toc405227058)

[- Minimum Spanning ‘Forest’ 3](#_Toc405227059)

[- Second Best Spanning Tree 3](#_Toc405227060)

[Depth First Search & Topological Sort 3](#_Toc405227061)

[Bipartite Graph Check 3](#_Toc405227062)

[SSSP (Dijkstra) 4](#_Toc405227063)

[Bellman-Ford 4](#_Toc405227064)

[APSP (All-Pairs Shortest Paths) 5](#_Toc405227065)

[Max Flow 5](#_Toc405227066)

[Max-Flow Variants 7](#_Toc405227067)

[Dynamic Programming 7](#_Toc405227068)

[Longest Increasing Subsequence (Full Code :D) 7](#_Toc405227069)

[Maximum Sum 8](#_Toc405227070)

[Knapsack 0-1 Problem 9](#_Toc405227071)

[Mathematics 10](#_Toc405227072)

[Java Big Integer Class 10](#_Toc405227073)

[Standard 10](#_Toc405227074)

[Greatest Common Divisor (Simplyfing Fractions) 11](#_Toc405227075)

[Modulo Arithmetic 11](#_Toc405227076)

[Base Number Conversion 12](#_Toc405227077)

[Combinatorics 12](#_Toc405227078)

[Catalan Numbers 12](#_Toc405227079)

[Number Theory 12](#_Toc405227080)

[Prime Numbers 12](#_Toc405227081)

[Greatest Common Divisor (GCD) & Least Common Multiple (LCM) 13](#_Toc405227082)

[Finding Prime Factors with Optimized Trial Divisions 13](#_Toc405227083)

[Others Functions Involving Prime Factors 13](#_Toc405227084)

[Knuth Morris-Pratt 15](#_Toc405227085)

[Geometry 15](#_Toc405227086)

[Line 2](#_Toc405227087)

[Circle 3](#_Toc405227088)

[Triangle 3](#_Toc405227089)

[POLYGON 4](#_Toc405227090)

[Convex Hull 4](#_Toc405227091)

[Rare Topics / Library 6](#_Toc405227092)

[Java Gregorian Calender 6](#_Toc405227093)

[Roman Number 6](#_Toc405227094)

[Josephus Problem 7](#_Toc405227095)

[Sliding Window 7](#_Toc405227096)

[Magic Square 8](#_Toc405227097)

[Parentheses Balancing 9](#_Toc405227098)

# Library and Macros

#include <cstdio>

#include <cmath>

#include <cstring>

#include <cstdlib>

#include <iostream>

#include <string>

#include <algorithm>

#include <vector>

#include <map>

#include <stack>

#include <queue>

#include <set>

#include <bitset>

**using** **namespace** std**;**

**typedef** long long LL**;**

**typedef** pair**<**int**,**int**>** pii**;**

**typedef** pair**<**pii**,**pii**>** ppi**;**

**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**;**

**typedef** vector**<**string**>** vstr**;**

**typedef** vector**<**char**>** vc**;**

**typedef** map **<**int**,**int**>** mii**;**

double PI **=** acos**(-**1**);**

#define REP(i,n) for(int (i) = 0; (i) < (int)(n); (i)++)

#define READ\_int(data) scanf("%d",&data);

#define RESET(c,v) memset(c, v, sizeof(c))

#define MAX(a,b) a = max(a,b)

#define MIN(a,b) a = min(a,b)

#define pb push\_back

#define mp make\_pair

#define ALL(vec) vec.begin(),vec.end()

#define sI(val) scanf("%d",&val)

#define sID(val,val2) scanf("%d %d",&val,&val2)

#define INF 2123123123

#define EPS 1e-9

inline string IntToString**(**int a**){**

char x**[**100**];**

sprintf**(**x**,**"%d"**,**a**);** string s **=** x**;**

**return** s**;**

**}**

inline int StringToInt**(**string a**){**

char x**[**100**];** int res**;**

strcpy**(**x**,**a**.**c\_str**());** sscanf**(**x**,**"%d"**,&**res**);**

**return** res**;**

**}**

inline string GetString**(**void**){**

char x**[**1000005**];**

scanf**(**"%s"**,**x**);** string s **=** x**;**

**return** s**;**

**}**

// FPB gan

int gcd**(**int a**,** int b**){**

**if(**b **==** 0**)return** a**;**

**else** **return** gcd**(**b**,**a**%**b**);**

**}**

//kpk gan

ll lcm**(**ll a**,**ll b**){**

**return** **(**a**\*(**b**/**gcd**(**a**,**b**)));**

**}**

// Flood fill gan

int dr**[]={**1**,**1**,**0**,-**1**,-**1**,-**1**,** 0**,** 1**};**

int dc**[]={**0**,**1**,**1**,** 1**,** 0**,-**1**,-**1**,-**1**};**

// Knight Moves

int a1**[**8**]** **=** **{**2**,**1**,-**1**,-**2**,-**2**,-**1**,**1**,**2**};**

int b2**[**8**]** **=** **{**1**,**2**,**2**,**1**,-**1**,-**2**,-**2**,-**1**};**

bitset**<**10000010**>** bs**; // Tambah :D**

vi primes**;**

// Prima pertama terdapat di primes[1] yah :D **(sampe 10^7 saja)**

void sieve**(**LL upperbound**){**

bs**.**set**();**

bs**[**0**]=**bs**[**1**]=**0**;**

primes**.**pb**(**0**);**

**for(**LL i**=**2**;** i**<=**upperbound**+**1**;** i**++)if(**bs**[**i**]){**

**for(**LL j**=**i**\***i**;** j**<=** upperbound**+**1**;** j**+=**i**)**bs**[**j**]** **=** 0**;**

primes**.**pb**((**int**)** i**);**

**}**

**}**

int main**()**

**{**

**return** 0**;**

**}**

# Data Structure

// User for RSQ, and more

## Fenwick Tree (BIT)

// Fenwick Tree Library

#define LSOne(S) (S & (-S))

vi ft**;**

void ft\_create**(**vi **&**t**,** int n**)**

**{**

t**.**assign**(**n**+**1**,**0**);**

**}**

// Returns RSQ(1, b)

int ft\_rsq**(**const vi **&**t**,** int b**){**

int sum **=** 0**;**

**for(;** b**;** b**-=** LSOne**(**b**))** sum **+=** t**[**b**];**

**return** sum**;**

**}**

// Returns RSQ(a,b)

int ft\_rsq**(**const vi **&**t**,** int a**,** int b**){**

**return** ft\_rsq**(**t**,**b**)** **-** **(**a **==** 1 **?** 0 **:** ft\_rsq**(**t**,**a**-**1**));**

**}**

// Adjust value of k-th element (+/-)

void ft\_adjust**(**vi **&**t**,** int k**,** int v**){**

**for(;** k **<=** **(**int**)**t**.**size**();** k**+=** LSOne**(**k**))**

t**[**k**]** **+=** v**;**

**}**

// Set the value in an index

void set\_val**(**vi **&**t**,** int k**,** int v**){**

int jum **=** ft\_rsq**(**ft**,**k**,**k**);**

//printf("jum : %d\n",jum);

**for(;** k **<=** **(**int**)**t**.**size**();** k**+=** LSOne**(**k**))**

**{**

t**[**k**]** **=** **(**t**[**k**]** **-** jum**)** **+** v**;**

**}**

**}**

# Contoh Pemakaian

1. ft\_create**(**ft**,**N**);**
2. ft\_adjust**(**ft**,**i**,**val**);**
3. set\_val**(**ft**,**l**,**val**);**
4. ft\_rsq**(**ft**,**l**,**r**)**

# Graph

## Minimum Spanning Tree

vector **<** pair**<**int**,** pii**>** **>** EdgeList**;**

vector **<**int**>** pSet**;** // To store the Set Parent

/// -------- Union Find Disjoint Method -------//

void initSet**(**int n**)** **{**

pSet**.**assign**(**n**,**0**);**

REP**(**i**,**n**)** pSet**[**i**]** **=** i**;**

**}**

int findSet**(**int i**)** **{**

**return** **(**pSet**[**i**]** **==** i**)** **?** i **:** pSet**[**i**]** **=** findSet**(**pSet**[**i**]);**

**}**

bool isSameSet**(**int i**,** int j**)** **{**

**return** findSet**(**i**)** **==**   
 findSet**(**j**);**

**}**

void unionSet**(**int i**,**int j**)** **{**

pSet**[**findSet**(**i**)]** **=** findSet**(**j**);**

**}**

int findST**()** **{**

int mstLength **=** 0**;**

initSet**(**n**);**

REP**(**i**,**m**)**

**{**

pair**<**int**,**pii**>** temp **=** EdgeList**[**i**];**

**if(!**isSameSet**(**temp**.**second**.**first**,** temp**.**second**.**second**))**

**{**

unionSet**(**temp**.**second**.**first**,**temp**.**second**.**second**);**

mstLength **+=** temp**.**first**;**

**}**

**}**

**return** mstLength**;**

**}**

## Minimum Spanning Tree – Variants

### Partial ‘minimum’ Spanning tree

After taking into account all the fixed edges, we continue running Kruskal’s algorithm on the remaining edges

### Minimum Spanning ‘Forest’

Run Kruskal’s algorithm as per normal, but as soon as the number of connected components equals to the desired pre-determined number, terminate the algorithm.

### Second Best Spanning Tree

Run Kruskal to find MST, the for each edge in the MST (there are at most V-1 edges in the MST), temporarily flag it so that cannot be chosen, then try to find the MST again *excluding* the flagged edge.

## Depth First Search & Topological Sort

void dfs2**(**int u**)**

**{**

dfs\_num**[**u**]** **=** 1**;**

// Set as visited

REP**(**j**,**adjList**[**u**].**size**())**

**{**

int v **=** adjList**[**u**][**j**];**

**if(**dfs\_num**[**v**]** **==** 0**)**   
 dfs2**(**v**);**

**}**

**// Use if Toposort (don’t forget to Reverse the toposort :D)**

toposort**.**pb**(**u**);**

**}**

## Bipartite Graph Check

vector **<**int**>** adjList**[**100007**];**

bool isBipartite**()**

**{**

queue**<**int**>** q**;**

q**.**push**(**0**);**

map**<**int**,**int**>** dist**;**   
 // Check has been Processed or not

dist**[**0**]** **=** 0**;**

bool isBipartite **=** **true;**

**while(!**q**.**empty**())**

**{**

int u **=** q**.**front**();**

q**.**pop**();**

REP**(**j**,**adjList**[**u**].**size**())**

**{**

int v **=** adjList**[**u**][**j**];**

**if(!**dist**.**count**(**v**))**   
 // Not Processed Yet

**{**

dist**[**v**]** **=** 1 **-** dist**[**u**];**   
 // Just record 2 colors

q**.**push**(**v**);**

**}**

**else** **if(**dist**[**v**]** **==** dist**[**u**])**

**{**

**return** **false;**

**}**

**}**

**}**

**return** **true; }**

## SSSP (Dijkstra)

vi dist**;**

void dijkstra**()**

**{**

dist**.**assign**(**n**,**INF**);**

// Jarak dari source ke source = 0

dist**[**s**]** **=** 0**;**

priority\_queue**<**pii**,**   
 vector**<**pii**>,** greater**<**pii**>** **>** pq**;**

pq**.**push**(**mp**(**0**,**s**));**

// Processing

**while(!**pq**.**empty**())**

**{**

pii temp **=** pq**.**top**();**

pq**.**pop**();**

int d **=** temp**.**first**;**

int u **=** temp**.**second**;**

//Mencegah double Relax

**if(**d **==** dist**[**u**]){**

**for(**int j **=** 0**;** j **<** **(**int**)** AdjList**[**u**].**size**();** j**++)**

**{**

pii v **=** AdjList**[**u**][**j**];**

// Semua edge yang keluar dari node

**if(**dist**[**u**]** **+** v**.**second **<** dist**[**v**.**first**])**

**{**

dist**[**v**.**first**]** **=** dist**[**u**]** **+** v**.**second**;**

pq**.**push**(**mp**(**dist**[**v**.**first**],** v**.**first**));**

**}**

**}**

**}**

**}**

**}**

## Bellman-Ford

vi dist**;**

void BellmannFord**()**

**{**

dist**.**assign**(**n**,**INT\_MAX**);**

dist**[**0**]** **=** 0**;**

**for(**int i **=** 0**;** i **<** n**-**1**;** i**++)**

// Relax all Edges V-1 Times

**{**

**for(**int u **=** 0**;** u **<** n**;** u**++)**

**{**

**for(**int j **=** 0**;** j **<** AdjList**[**u**].**size**();** j**++)**

**{**

pii v **=** AdjList**[**u**][**j**];**

dist**[**v**.**first**]** **=** min**(**dist**[**v**.**first**],** **(**dist**[**u**]** **+** v**.**second**)** **);**

**}**

**}**

**}**

**}**

void hasNegativeCycle**()**

**{**

bool hNC **=** **false;**

**for(**int u **=** 0**;** u **<** n**;** u**++)**

**{**

**for(**int j **=** 0**;** j **<** AdjList**[**u**].**size**();** j**++)**

**{**

pii v **=** AdjList**[**u**][**j**];**

**if(**dist**[**v**.**first**]** **>** dist**[**u**]** **+** v**.**second**)**

**{**

hNC **=** **true;**

**}**

**}**

**}**

printf**(**"%s\n"**,**hNC **?** "possible" **:** "not possible"**);**

**}**

## APSP (All-Pairs Shortest Paths)

void floydWarshall**()**

**{**

**for(**int k **=** 0**;** k **<** n**;** k**++)**

**for(**int i **=** 0**;** i **<** n**;** i**++)**

**for(**int j **=** 0**;** j **<** n**;** j**++)**

AdjMat**[**i**][**j**]** **=** min**(**AdjMat**[**i**][**j**],** **(**AdjMat**[**i**][**k**]** **+** AdjMat**[**k**][**j**])** **);**

**}**

**// minimax**

d**[**i**][**j**]** **=** min**(**d**[**i**][**j**],** max**(**d**[**i**][**k**],**d**[**k**][**j**])** **);**

## Max Flow

// The flow, initially 0

int cap**[**1000**][**1000**];**

int mf**,**f**;**

//Stores the parent, find path s->t

vi p**;**

void addFlow**(**int n1**,** int n2**,** int w**)** **{**

// Tambahkan flow

cap**[**n1**][**n2**]** **+=** w**;**

cap**[**n2**][**n1**]** **+=** w**;**

adjList**[**n1**].**pb**(**n2**);**

adjList**[**n2**].**pb**(**n1**);**

**}**

// CP's Library

void augment**(**int v**,**int minEdge**) {**

/Traverse spanning tree from s -> t

//printf("masuk4\n");

**if(**v**==**s**)** **{**

//printf("masuk5\n");

f **=** minEdge**;**

**return;**

**}**

**else** **if(**p**[**v**]** **!=** **-**1**) {**

//printf("masuk6\n");

augment**(**p**[**v**],** min**(**minEdge**,** cap**[**p**[**v**]][**v**]));**

cap**[**p**[**v**]][**v**]** **-=** f**;** cap**[**v**][**p**[**v**]]** **+=** f**;**

**}**

**}**

int maxFlow**(){**

mf **=** 0**;**

**while(**1**){**

queue **<**int**>** q**;**

f **=** 0**;**

p**.**assign**((**N **+** 5**),-**1**);**

q**.**push**(**s**);**   
 // Cari Dari source

p**[**s**]** **=** s**;**

**while(!**q**.**empty**()){**

int u **=** q**.**front**();**

//printf("u : %d\n",u);

q**.**pop**();**

//Setelah dapat sink, break

**if(**u **==** t**)** **break;**

REP**(**v**,**adjList**[**u**].**size**()){**

int node **=** adjList**[**u**][**v**];**

**if(**cap**[**u**][**node**]** **>** 0 **&&** p**[**node**]==** **-**1**)** **{**

q**.**push**(**node**);**

p**[**node**]** **=** u**;**

**}**

**}**

**}**

// Cari lagi minimum edge weight

augment**(**t**,**INF**);**   
**if(**f **==** 0**)** **break;**  
mf **+=** f**;**  
//printf("mf saat ini : %d\n",mf);

**}**

**return** mf**;**

**}**

Max-Flow Variants  
- Min Cut  
 **Definition** : If all edges in the cut-set of C are removed, the Max Flow from *s* to *t* is 0 (i.e ***s* and *t* disconnected**).

**Step :**

1. After Max Flow Algorithm stops, run graph traversal (DFS,BFS) from source *s* again.
2. Vertices that are still reachable from source *s* belong to **S-Component.**
3. All other unreachable vertices belong to the **T-Component**.
4. All other connecting S-Component and T component belong to the cut-set.

# Dynamic Programming

## Longest Increasing Subsequence (Full Code :D)

vi data**;**

// Menyimpan index element yang masuk ke dalam LIS.

vi lisIdx**;**

// Menyimpan parent (Untuk ngeprint path)

vi lisPar**;**

int BS**(**int x**)** **{**

int l**,**r**,**mid**;**

l **=** 0**;**

r **=** lisIdx**.**size**();**

**while(**l **<** r**)** **{**

mid **=** **(**l **+** r**)** **/** 2**;**

**if(**data**[**lisIdx**[**mid**]]** **<** x**)** l **=** mid **+** 1**;**

**else** r **=** mid**;**

**}**

**return** r**;**

**}**

int main**()** **{**

int bil**;**

**while(**sI**(**bil**)** **==** 1**)** data**.**pb**(**bil**);**

// Set parentnya -1 dlu.

REP**(**i**,** data**.**size**())** lisPar**.**pb**(-**1**);**

// Mulai pengecekan

int idx**;**

REP**(**i**,** data**.**size**())**

**{**

idx **=** BS**(**data**[**i**]);**

// Bisa di-append ke belakang

**if(**idx **==** lisIdx**.**size**())** lisIdx**.**pb**(**i**);**

**else** lisIdx**[**idx**]** **=** i**;**

// Masukkan ke list parent, exception untuk elemen pertama (no have parent)

**if(**idx **!=** 0**)** lisPar**[**i**]** **=** lisIdx**[**idx**-**1**];**

**}**

vi ans**;**

int curr **=** lisIdx**.**back**();** // Traverse dari belakang, Generate Path

**while(**curr **!=** **-**1**)** **{**

ans**.**pb**(**curr**);**

curr **=** lisPar**[**curr**];**

**}**

reverse**(**ALL**(**ans**));**

// Print hasil

printf**(**"%d\n-\n"**,**lisIdx**.**size**());**

// Print path

REP**(**a**,**ans**.**size**())** printf**(**"%d\n"**,**data**[**ans**[**a**]]);**

**return** 0**;**

**}**

## Maximum Sum

int main**()**

**{**

int n**;**

sI**(**n**);**

int temp**;**

int data**[**105**][**105**];**

// Generate Sum Table

REP**(**a**,**n**)**

**{**

REP**(**b**,**n**)**

**{**

sI**(**data**[**a**][**b**]);**

// Inclusion - Exclusion Principle

**if(**a **>** 0**)** data**[**a**][**b**]** **+=** data**[**a**-**1**][**b**];**

**if(**b **>** 0**)** data**[**a**][**b**]** **+=** data**[**a**][**b**-**1**];**

**if(**a **>** 0 **&&** b **>** 0 **)** data**[**a**][**b**]** **-=** data**[**a**-**1**][**b**-**1**];**

**}**

**}**

// 100 is N constraint

int MaxSum **=** **-**127 **\*** 100 **\*** 100**;**

// Hitung Max Sum

int subRect**;** // Variabel penampung sum sub rectangular

REP**(**a**,**n**)**

**{**

REP**(**b**,**n**)**

**{**

**for(**int c **=** 0**;** c **<=** a**;** c**++)**

**{**

**for(**int d **=** 0**;** d **<=** b**;** d**++)**

**{**

subRect **=** data**[**a**][**b**];**

// Inclusi - Exclusi again

**if(**c **>** 0**)** subRect **-=** data**[**c**-**1**][**b**];**

**if(**d **>** 0**)** subRect **-=** data**[**a**][**d**-**1**];**

**if(**c **>** 0 **&&** d **>** 0**)** subRect **+=** data**[**c**-**1**][**d**-**1**];**

MaxSum **=** max**(**subRect**,** MaxSum**);**

**}**

**}**

**}**

**}**

printf**(**"%d\n"**,**MaxSum**);**

**return** 0**;**

**}**

## Knapsack 0-1 Problem

// Processing -- Knapsack Table & KeepTable

int knTable**[**nItem **+** 5**][**money **+** 5**];**

int keepTable**[**nItem**+**5**][**money **+** 5**];**

RESET**(**knTable**,**0**);**

RESET**(**keepTable**,**0**);**

int stat**;**

**for(**int i **=** 1**;** i **<=** t**;** i**++)** **{**

**for(**int j **=** 1**;** j **<=** oxygen**;** j**++){**

int temp**;**

// Cost

// Jika bisa make barang yang di baris ke - i

**if(**j **–** data[i] **>=** 0**)** **{**

temp **=** knTable**[**i**-**1**][**j**-data[1]];**

stat **=** 1**;**

**}**

**else** stat **=** 0**;**

**if(**stat **==** 0**)** **{**

// Item tidak bisa dipakai, ambil yang dari knapsack table sebelumnya

knTable**[**i**][**j**]** **=** knTable**[**i**-**1**][**j**];** keepTable**[**i**][**j**]** **=** 0**;** // Item tidak di-keep , set keep table 0

**}**

**else {**

knTable**[**i**][**j**]** **=** max**(**knTable**[**i**-**1**][**j**],** **(**temp **+** data**[**i**].**second**)** **);**

// Ambil dari yang atas, item ga dikeep

**if(**knTable**[**i**][**j**]** **==** knTable**[**i**-**1**][**j**])** **{**

keepTable**[**i**][**j**]** **=** 0**;**

**}**

// Item included

**else** **{**

keepTable**[**i**][**j**]** **=** 1**;**

**}**

**}**

**}**

**}**

// Ambil Optimal Solusion di kolom terakhir. :D

**if** **(** test**++** **)** puts**(**""**);**

printf**(**"%d\n"**,**knTable**[**nItem**][**money**]);**

// Print Path

int berat **=** money**;**

vi idxGet**;**

**for(**int i **=** t**;** i **>=** 1**;** i**--)**

**{**

**if(**keepTable**[**i**][**money**]** **==** 1**)**

**{**

idxGet**.**pb**(**i**);**

berat **=** berat **-** **data[i];**   
 **}**

**}**

printf**(**"%d\n"**,**idxGet**.**size**());**

**for(**int i **=** idxGet**.**size**()-**1**;** i**>=**0**;** i**--)**   
 printf**(**"%d\n"**,**idxGet**[**i**])**

**}**

**return** 0**;**

**}**

# Mathematics

## Java Big Integer Class

### Standard

**import** java**.**math**.**BigInteger**;**

**import** java**.**util**.**Scanner**;**

class Main **{**

public static void main**(**String**[]** args**)** **{**

BigInteger angka **=** BigInteger**.**valueOf**(**1**);**

// TODO code application logic here

Scanner scanner **=** **new** Scanner**(**System**.**in**);**

**while(**scanner**.**hasNext**())** **{**

angka **=** scanner**.**nextBigInteger**();**

**if(**angka**.**equals**(**BigInteger**.**ZERO**))** **break;**

**if(**angka**.**mod**(**BigInteger**.**valueOf**(**17**)).**equals**(**BigInteger**.**ZERO**))** **{**

System**.**out**.**println**(**"1"**);**

**}**

**else{**

System**.**out**.**println**(**"0"**);**

**}**

**}**

**}**

**}**

### Greatest Common Divisor (Simplyfing Fractions)

**// Find simplest p/q**

**import** java**.**util**.**Scanner**;**

**import** java**.**math**.**BigInteger**;**

class Main**{**

public static void main**(**String**[]** args**)** **{**

Scanner sc **=** **new** Scanner**(**System**.**in**);**

int N **=** sc**.**nextInt**();**

**while(**N**--** **>** 0**){**

BigInteger p **=** sc**.**nextBigInteger**();**

String ch **=** sc**.**next**();**

BigInteger q **=** sc**.**nextBigInteger**();**

// Find the GCD

BigInteger gcd\_pq **=** p**.**gcd**(**q**);**

System**.**out**.**println**(**p**.**divide**(**gcd\_pq**)** **+** " / " **+** q**.**divide**(**gcd\_pq**));**

**}**

**}**

**}**

### Modulo Arithmetic

**// find x^y(mod n)**

**import** java**.**util**.**Scanner**;**

**import** java**.**math**.**BigInteger**;**

class Main**{**

public static void main**(**String**[]** args**){**

Scanner sc **=** **new** Scanner**(**System**.**in**);**

int N **=** sc**.**nextInt**();**

**while(**N**--** **>** 0**){**

BigInteger x **=** BigInteger**.**valueOf**(**sc**.**nextInt**());**

BigInteger y **=** BigInteger**.**valueOf**(**sc**.**nextInt**());**

BigInteger n **=** BigInteger**.**valueOf**(**sc**.**nextInt**());**

System**.**out**.**println**(**x**.**modPow**(**y**,**n**));**

**}**

**}**

**}**

### Base Number Conversion

**// Print in any Base**

**import** java**.**util**.**Scanner**;**

**import** java**.**math**.**BigInteger**;**

class Main**{**

public static void main**(**String**[]** args**){**

Scanner sc **=** **new** Scanner**(**System**.**in**);**

**while(true){**

int b **=** sc**.**nextInt**();**

**if(**b **==** 0**)** **break;**

String p\_str **=** sc**.**next**();**

// Special class's Constructor

BigInteger p **=** **new** BigInteger**(**p\_str**,**b**);**

String m\_str **=** sc**.**next**();**

// 2nd Parameter is the base;

BigInteger m **=** **new** BigInteger**(**m\_str**,**b**);**

// Can output in any base :D

System**.**out**.**println**((**p**.**mod**()).**toString**(**b**));**

**}**

**}**

**}**

## Combinatorics

### Catalan Numbers

Cat(n) = 2n!/ n! x n! x (n+1)   
Cat(n + 1) = ( (2n+2) x (2n+1) / (n+2) x (n+1) ) x Cat(n)

## Number Theory

### Prime Numbers

LL \_sieve\_size**;**

bitset**<**10000010**>** bs**;**  
vi primes**;**

// Prima pertama terdapat di primes[1] yah :D

void sieve**(**LL upperbound**){**

bs**.**set**();**

bs**[**0**]=**bs**[**1**]=**0**;**

primes**.**pb**(**0**);**

**for(**LL i**=**2**;** i**<=**upperbound**+**1**;** i**++)if(**bs**[**i**]){**

**for(**LL j**=**i**\***i**;** j**<=** upperbound**+**1**;** j**+=**i**)**bs**[**j**]** **=** 0**;**

primes**.**pb**((**int**)** i**);**

**}**

**}**

// A good Deterministic Prime Factor

bool isPrime**(**LL N**)**

**{**

// O(1) for small primes

**if(**N **<=** \_sieve\_size**)** **return** bs**[**N**];**

**for(**int i **=** 0**;** i **<** primes**.**size**();** i**++)**

**{**

**if(**N **%** primes**[**i**]** **==** 0**)** **return** **false;**

**return** **true;**

**}**

**}**

### Greatest Common Divisor (GCD) & Least Common Multiple (LCM)

// gcd(4,8) = 4, gcd(10,5) = 5, gcd(20,12) = 4

int gcd**(**int a**,** int b**){**

**if(**b **==** 0**)return** a**;**

**else** **return** gcd**(**b**,**a**%**b**);**

**}**

// lcm(4,8) = 8 lcm(10,5) = 10, lcm(20,12) = 60

int lcm**(**int a**,**int b**){**

**return** **(**a **\*** **(**b **/** gcd**(**a**,**b**)));**

**}**

### Finding Prime Factors with Optimized Trial Divisions

Contoh Penggunaan :  
- Mencari Faktor Prima terbesar dari suatu bilangan

vl primeFactors**(**LL N**){**

vl factors**;**

LL PF\_idx **=** 0**;**

LL PF **=** primes**[**PF\_idx**];**

// Make abs(N) in case negative value

**while(**N **!=** 1 **&&** **(**PF **\*** PF **<=** N**)){**

**while(**N **%** PF **==** 0**){**

factors**.**pb**(**PF**);**

N **/=** PF**;**

**}**

PF **=** primes**[++**PF\_idx**];**

**}**

**if(**N **!=** 1**)** factors**.**pb**(**N**);**

**return** factors**;**

**}**

### Others Functions Involving Prime Factors

* numDiv(n) : **Count the number of divisor of n**

LL numDiv**(**LL N**){**

LL PF\_idx **=** 0**;**

PF **=** primes**[**PF\_idx**];**

ans **=** 1**;**

**while(**N **!=** 1 **&&** **(**PF **\*** PF**)** **<=** N**){**

LL power **=** 0**;**

**while(**N **%** PF **==** 0**)** **{**

N **/=** PF**;**

power**++;**

**}**

ans **\*=** **(**power**+**1**);**

PF **=** primes**[++**PF\_idx**];**

**}**

**if(**N **!=** 1**)** ans **\*=** 2**;**

**return** ans**;**

**}**

* sumDiv(n) : **Sum the divisors of N**

// Untuk menghitung Sum dari semua Divisors dari N

LL sumDiv**(**LL N**){**

LL PF\_idx **=** 0**;**

LL PF **=** primes**[**PF\_idx**];**

LL ans **=** 1**;**

**while(**N **!=** 1 **&&** **(**PF **\*** PF **<=** N**)){**

LL power **=** 0**;**

**while(**N **%** PF **==** 0**){**

N **/=** PF**;**

power**++;**

**}**

ans **\*=** **((**LL**)** pow**(** **(**double**)**PF**,** power **+** 1.0**)** **-**1**)** **/** **(**PF**-**1**);**

PF **=** primes**[++**PF\_idx**];**

**}**

**if(**N **!=** 1**)** ans **\*=** **((**LL**)** pow**(** **(**double**)**N**,** 2.0**)** **-**1**)** **/** **(**N**-**1**);**

**return** ans**;**

**}**

* EulerPhi(n) : **Count the number of positive integers < N that are relatively Prime to N, gcd(a,b) = 1.**

LL EulerPhi**(**LL N**){**

LL PF\_idx **=** 0**;**

LL PF **=** primes**[**PF\_idx**];**

LL ans **=** N**;**

**while(**N **!=** 1 **&&** **(**PF **\*** PF **<=** N**)) {**

**if(**N **%** PF **==** 0**)** ans **-=** ans **/** PF**;**

**while(**N **%** PF **==** 0**)** N **/=** PF**;**

PF **=** primes**[++**PF\_idx**];**

**}**

**if(**N **!=** 1**)** ans **-=** ans**/**N**;**

**return** ans**;**

**}**

## Knuth Morris-Pratt

Penggunaan KMP : Ambil 2 string, String pertama data kata String kedua kata yang mau dicari

Misal String 1 = "aku makan roti bakar"; String 2 = "aka";

Lalu 2 integer untuk menampung length masing-masing string

Trus ada 2 method, kmpPreprocess(); dan kmpSearch();

Setelah data tertampung, panggil kmpPreprocess(); terlebih

dahulu, lalu panggil kmpSearch();

char T**[**MAX\_N**],** P**[**MAX\_N**];** // T = text, P = pattern

int b**[**MAX\_N**],** n**,** m**;** // b = back table, n = length of T, m = length of P

void kmpPreprocess**()** **{** // call this before calling kmpSearch()

int i **=** 0**,** j **=** **-**1**;** b**[**0**]** **=** **-**1**;** // starting values

**while** **(**i **<** m**)** **{** // pre-process the pattern string P

**while** **(**j **>=** 0 **&&** P**[**i**]** **!=** P**[**j**])** j **=** b**[**j**];** // if different, reset j using b

i**++;** j**++;** // if same, advance both pointers

b**[**i**]** **=** j**;** // observe i = 8, 9, 10, 11, 12 with j = 0, 1, 2, 3, 4

**}**

**}** // in the example of P = "SEVENTY SEVEN" above

void kmpSearch**()** **{** // this is similar as kmpPreprocess(), but on string T

int i **=** 0**,** j **=** 0**;** // starting values

**while** **(**i **<** n**)** **{** // search through string T

**while** **(**j **>=** 0 **&&** T**[**i**]** **!=** P**[**j**])** j **=** b**[**j**];** // if different, reset j using b

i**++;** j**++;** // if same, advance both pointers

**if** **(**j **==** m**)** **{** // a match found when j == m

printf**(**"P is found at index %d in T\n"**,** i **-** j**);**

j **=** b**[**j**];** // prepare j for the next possible match

**}**

**}**

**}**

# Geometry

struct point\_i **{** int x**,** y **};** // use this whenever possible

struct point **{** double x**,** y **};** // but I will use this form now

struct point **{**

double x**,** y**;**

point**(**double \_x**,** double \_y**)** **{** x **=** \_x**,** y **=** \_y**;** **}**

bool operator **<** **(**point other**)** **{**

**if** **(**fabs**(**x **-** other**.**x**)** **>** EPS**)** // useful for sorting

**return** x **<** other**.**x**;** // first criteria , by x-axis

**return** y **<** other**.**y**;** // second criteria, by y-axis

**}**

**};**

sort**(**P**.**begin**(),**P**.**end**());** //Sort the point

bool areSame**(**point\_i p1**,**point\_i p2**){**

**return** p1**.**x **==** p2**.**x **&&** p1**.**y **==** p2**.**y**;**

**}**

bool areSame**(**point p1**,** point p2**)** **{** // floating point version

// use EPS when testing equality of two floating points

**return** fabs**(**p1**.**x **-** p2**.**x**)** **<** EPS **&&** fabs**(**p1**.**y **-** p2**.**y**)** **<** EPS**;** **}**

double dist**(**point p1**,** point p2**)** **{**

double dist**(**point p1**,** point p2**)** **{** // Euclidean distance // Euclidean distance

// hypot(dx, dy) returns sqrt(dx \* dx + dy \* dy)

**return** hypot**(**p1**.**x **-** p2**.**x**,** p1**.**y **-** p2**.**y**);**

**}}** // return

## Line

**-** Poor line equation**,** y **=** mx **+** c **(**vertical line**,**special **case)**

**-** Better line equation**,** ax **+** by **+** c **=** 0

struct line **{** double a**,** b**,** c**;** **};** // a way to represent a line

// the answer is stored in the third parameter (pass byref)

void pointsToLine**(**point p1**,** point p2**,** line **\***l**)** **{**

**if** **(**p1**.**x **==** p2**.**x**)** **{** // vertical line is handled nicely here

l**->**a **=** 1.0**;** l**->**b **=** 0.0**;** l**->**c **=** **-**p1**.**x**;**

**}** **else** **{**

l**->**a **=** **-(**double**)(**p1**.**y **-** p2**.**y**)** **/** **(**p1**.**x **-** p2**.**x**);**

l**->**b **=** 1.0**;** // fix the value of b to 1.0

l**->**c **=** **-(**double**)(**l**->**a **\*** p1**.**x**)** **-** **(**l**->**b **\*** p1**.**y**);**

**}**

**}**

//Interaction Between 2 LINES

bool areParallel**(**line l1**,** line l2**)** **{** // check coefficient a + b

**return** **(**fabs**(**l1**.**a**-**l2**.**a**)** **<** EPS**)** **&&** **(**fabs**(**l1**.**b**-**l2**.**b**)** **<** EPS**);** **}**

bool areSame**(**line l1**,** line l2**)** **{** // also check coefficient c

**return** areParallel**(**l1**,** l2**)** **&&** **(**fabs**(**l1**.**c **-** l2**.**c**)** **<** EPS**);** **}**

// returns true (+ intersection point) if two lines are intersect

bool areIntersect**(**line l1**,** line l2**,** point **\***p**)** **{**

**if** **(**areSame**(**l1 l2**))** t f l **(**l1**,** l2**))** **return** false**;** // ll i t i t t // all points intersect

**if** **(**areParallel**(**l1**,** l2**))** **return** false**;** // no intersection

// solve system of 2 linear algebraic equations with 2 unknowns

p**->**x **=** **(**d bl **)(**l2 b **\*** l1 **(**double**)(**l2**.**b **\*** l1**.**c **-** l1 b **\*** l2 **)** **/** l1**.**b **\*** l2**.**c**)** **/** **(**l2**.**a **\*** l1**.**b **-** l1**.**a **\*** l2**.**b**);**

**if** **(**fabs**(**l1**.**b**)** **>** EPS**)** // test for vertical line

p**->**y **=** **-** **(**l1**.**a **\*** p**->**x **+** l1**.**c**)** **/** l1**.**b**;** // avoid div by zero

**else** // this is another special case in geometry problem...

p**->**y **=** **-** **(**l2**.**a **\*** p**->**x **+** l2**.**c**)** **/** l2**.**b**;**

**return** true**;**

**}**

## Circle

//Chek if a point inside a circle

int inCircle**(**point\_i p**,** point\_i c**,** int r**){** // All integer version

int dx **=** p**.**x **-** c**.**x**,** dy**=**p**.**y**-**c**.**y**;**

int Euc **=** dx**\***dx**+**dy**\***dy**,** rSq**=**r**\***r**;**

**return** Euc **<** rSq**?**0**:**Euc**==**rSq **?**1**:**2**;**

**}**

pi **=** 2 **\*** acos**(**0.0**)** //pi

d **=** 2 **\*** r // diameter

c **=**p **\*** d //keliling

A **=** p **\*** r **\*** r //Luas

arc**=** x **/** 360.0 **\*** c**;** //ARC adalah bagian dari keliling sesuai dengan sudut

sector **=** x**/** 360.0 **\*** A**;** //Luas daerah yang ada sudutnya itu

bool circle2PtsRad**(**point p1**,** point p2**,** double r**,** point **\***c**){** //ANSWER AT \*C

double d2**=(**p1**.**x**-**p2**.**x**)\*(**p1**.**x**-**p2**.**x**)+(**p1**.**y**-**p2**.**y**)\*(**p1**.**y**-**p2**.**y**);**

double det **=** r**\***r **/**d2**-**0.25**;**

**if(**det**<**0.0**)** **return** false**;**

double h **=** sqrt**(**det**);**

c**->**x**=(**p1**.**x**+**p2**.**x**)** **\*** 0.5 **+** **(**p1**.**y **-** p2**.**y**)\***h**;**

c**->**y**=(**p1**.**y**+**p2**.**y**)** **\*** 0.5 **+** **(**p2**.**x**-**p1**.**x**)** **\***h**;**

**return** true**;** //to get The other center, reverse p1 and p2

**}**

## Triangle

/\*

Equilateral = sama sisi

isosceles, sama kaki

scalene = sembarang

right = segitiga siku2

\*/

A **=** 0.5 **\*** b x h**;** //Luas

p **=** a **+** b **+** c //Keliling

A **=** sqrt**(**s **\*** **(**s**-** a**)** **\*** **(**s**-** b**)** **\*** **(**s**-** c**))** //Luas Cara 2 (Heron Formula)

s **=** 0.5 **\*** p //Semi parameter

r **=** area **/** s**;** // jari jari dalam segitiga

R **=** .25 **\*** **(**a **\*** b **\*** c**)** **/** area // jari2 luar segitiga

//Check if 3 lines can make a triangle

**(**a**+**b**>**c**)** **&&** **(**a**+**c**>**b**)&&** **(**b**+**c**>**a**);**

//Trigonometry/Law of Cosines

c2 **=** a2 **+** b2 **-** 2 **\*** a **\*** b **\*** cos**(?)**

**?** **=** acos**(((**a**\***a**)+(**b**\***b**)-(**c**\***c**))/**2**\***a**\***b**);**

//T i t /L Trigonometry/Law of Sines

a **/** sin**(?)** **=** b **/** sin**(?)** **=** c **/** sin**(?)**

//Trigonometry/Phytagorean Theorem

c2 **=** a2 **+** b2

## POLYGON

vector**<**point**>** P**;**

P**.**push\_back**(**point**(**1**,** 1**));**

P**.**push\_back**(**point**(**3**,** 3**));**

P**.**push\_back**(**point**(**9**,** 1**));**

P**.**push\_back**(**point**(**12**,** 4**));**

P**.**push\_back**(**point**(**9**,** 7**));**

P**.**push\_back**(**point**(**1**,** 7**));**

P**.**push\_back**(**P**[**0**]);**

double perimeter**(**vector**<**point**>** P**)** **{**

double result **=** 0.0**;**

**for** **(**int i **=** 0**;** i **<** **(**int**)**P**.**size**();** i**++)**

result **+=** dist**(**P**[**i**],** P**[(**i **+** 1**)** **%** P**.**size**()]);**

**return** result**;** **}**

double area**(**vector**<**point**>** P**)** **{**

double result **=** 0.0**,** x1**,** y1**,** x2**,** y2**;**

**for** **(**int i **=** 0**;** i **<** **(**int**)**P**.**size**();** i**++)** **{**

x1 **=** P**[**i**].**x**;** x2 **=** P**[(**i **+** 1**)** **%** P**.**size**()].**x**;**

y1 **=** P**[**i**].**y**;** y2 **=** P**[(**i **+** 1**)** **%** P**.**size**()].**y**;**

result **+=** **(**x1 **\*** y2 **-** x2 **\*** y1**);** 7 y

**}**

**return** fabs**(**result**)** **/** 2.0**;** **}**

## Convex Hull

point pivot**(**0**,**0**);**

double lenSqr**(**point a**,** point b**)** // function to compute distance between 2 points

**{**

double dx **=** a**.**x **-** b**.**x**,** dy **=** a**.**y **-** b**.**y**;**

**return** dx **\*** dx **+** dy **\*** dy**;**

**}**

int turn**(**point p**,** point q**,** point r**)**

**{**

int result **=** **(**r**.**x **-** q**.**x**)** **\*** **(**p**.**y **-** q**.**y**)** **-** **(**r**.**y **-** q**.**y**)** **\*** **(**p**.**x **-** q**.**x**);**

**if** **(**result **<** 0**)** **return** **-**1**;** // P->Q->R is a right turn

**if** **(**result **>** 0**)** **return** 1**;** // P->Q->R is a left turn

**return** 0**;** // P->Q->R is a straight line, i.e. P, Q, R are collinear

**}**

bool collinear**(**point **&**a**,**point **&**b**,**point **&**c**)**

**{**

**return** **(**turn**(**a**,**b**,**c**)==**0**);**

**}**

bool ccw**(**point p**,** point q**,** point r**)**

**{**

**return** **(**turn**(**p**,** q**,** r**)** **>** 0**);**

**}**

bool angleCmp**(**point a**,**point b**)**

**{**

**if(**collinear**(**pivot**,**a**,**b**))**

**{**

**return** lenSqr**(**pivot**,**a**)** **<** lenSqr**(**pivot**,**b**);**

**}**

int d1x **=** a**.**x **-** pivot**.**x**,** d1y **=** a**.**y **-** pivot**.**y**;**

int d2x **=** b**.**x **-** pivot**.**x**,** d2y **=** b**.**y **-** pivot**.**y**;**

**return** **(**atan2**((**double**)**d1y**,** **(**double**)**d1x**)** **-** atan2**((**double**)**d2y**,** **(**double**)**d2x**))** **<** 0**;**

**}**

vector**<**point**>** CH**(**vector**<**point**>** P**){**

int i**,**N**=(**int**)**P**.**size**();**

// return the vector

**if(**N **<=**3**)** //special case, the CH is P itself

**{**

P**.**push\_back**(**P**[**0**]);**

**return** P**;**

**}**

//first, find po = point with lowest Y and if tie: rightmost X

int po**=**0**;**

**for** **(** i**=**0**;** i**<**N**;** i**++)**

**if(**P**[**i**].**y **<** P**[**po**].**y **||** **(**P**[**i**].**y**==** P**[**po**].**y **&&** P**[**i**].**x **>** P**[**po**].**x**))**

po**=**i**;**

//swap selected vertex with P[0]

point temp **=** P**[**0**];**P**[**0**]** **=** P**[**po**];**P**[**po**]=**temp**;**

//second, sort points by angle w.r.. pivot po

pivot **=** P**[**0**];** //use this global variable as referenc

sort**(++**P**.**begin**(),**P**.**end**(),**angleCmp**);** //notice that we does not sort P[0]

//third, the ccw tests

point prev**(**0**,**0**),**now**(**0**,**0**);**

stack**<**point**>** S**;** //initial content of stack S

S**.**push**(**P**[**N**-**1**]);**

S**.**push**(**P**[**0**]);**

i**=**1**;**// then, we check the rest

**while(**i**<**N**)** //note: N must be >=3 for this method to work

**{**

now **=** S**.**top**();**

S**.**pop**();** //get 2nd from top

prev**=**S**.**top**();**

S**.**push**(**now**);**

**if(**ccw**(**prev**,**now**,**P**[**i**]))** //left turn , accept

S**.**push**(**P**[**i**++]);**

**else** S**.**pop**();** //otherwise, pop the top of stack S until we have a left turn

**}**

vector**<**point**>** ConvexHull**;** //from stack back to vector

**while(!**S**.**empty**()){**

ConvexHull**.**push\_back**(**S**.**top**());**

S**.**pop**();**

**}**

**return** ConvexHull**;** //return Result

**}**

# Rare Topics / Library

## Java Gregorian Calender

**import** java**.**util**.**GregorianCalendar**;**

**import** java**.**util**.**Calendar**;**

**import** java**.**util**.**Scanner**;**

class Main**{**

public static void main**(**String**[]** args**){**

Scanner scanner **=** **new** Scanner**(**System**.**in**);**

int n **=** scanner**.**nextInt**();**

int a**,**tanggal**,**bulan**;**

String**[]** names **=** **new** String**[]**

**{**""**,**"Sun"**,**"Mon"**,**"Tue"**,**"Wed"**,**"Thu"**,**"Fri"**,**"Sat"**};**

**for(**a **=** 1**;** a **<=** n**;** a**++){**

bulan **=** scanner**.**nextInt**();**

tanggal **=** scanner**.**nextInt**();**

Calendar calendar **=** **new** GregorianCalendar**(**2011**,(**bulan**-**1**),**tanggal**);**

System**.**out**.**println**(**names**[**calendar**.**get**(**Calendar**.**DAY\_OF\_WEEK**)]);**

**}**

**}**

**}**

### Roman Number

int th**,**h**,**t**,**o**;**

string thousands**[]** **=** **{**""**,** "M"**,** "MM"**,** "MMM"**,** "MMMM"**};**

string hundreds**[]** **=** **{**""**,** "C"**,** "CC"**,** "CCC"**,** "CD"**,** "D"**,** "DC"**,** "DCC"**,** "DCCC"**,** "CM"**};**

string tens**[]** **=** **{**""**,** "X"**,** "XX"**,** "XXX"**,** "XL"**,** "L"**,** "LX"**,** "LXX"**,** "LXXX"**,** "XC"**};**

string ones**[]** **=** **{**""**,** "I"**,** "II"**,** "III"**,** "IV"**,** "V"**,** "VI"**,** "VII"**,** "VIII"**,** "IX"**};**

// Romawi to Decimal

int roman2decimal**(**string roman**)** //assume roman is uppercase**{**

int result **=** 0**;**

int LEN **=** roman**.**length**();**

**for** **(**int i **=** 0**;** i **<** LEN**;** **++**i**){**

**if** **(**roman**[**i**]** **==** 'I' **&&** i **!=** LEN**-**1**){**

**if** **(**roman**[**i**+**1**]** **==** 'V'**)** **{** result **+=** 4**;** **++**i**;** **}** //IV leap V

**else** **if** **(**roman**[**i**+**1**]** **==** 'X'**)** **{** result **+=** 9**;** **++**i**;** **}** //IX leap X

**else** **{** result **+=** 1**;** **}**

**}**

**else** **if** **(**roman**[**i**]** **==** 'X' **&&** i **!=** LEN**-**1**){**

**if** **(**roman**[**i**+**1**]** **==** 'L'**)** **{** result **+=** 40**;** **++**i**;** **}** //XL leap L

**else** **if** **(**roman**[**i**+**1**]** **==** 'C'**)** **{** result **+=** 90**;** **++**i**;** **}** //XC leap C

**else** **{** result **+=** 10**;** **}**

**}**

**else** **if** **(**roman**[**i**]** **==** 'C' **&&** i **!=** LEN**-**1**){**

**if** **(**roman**[**i**+**1**]** **==** 'D'**)** **{** result **+=** 400**;** **++**i**;** **}** //CD leap D

**else** **if** **(**roman**[**i**+**1**]** **==** 'M'**)** **{** result **+=** 900**;** **++**i**;** **}** //CM leap M

**else** **{** result **+=** 100**;** **}**

**}**

**else** **if** **(**roman**[**i**]** **==** 'I'**)** **{** result **+=** 1**;** **}**

**else** **if** **(**roman**[**i**]** **==** 'V'**)** **{** result **+=** 5**;** **}**

**else** **if** **(**roman**[**i**]** **==** 'X'**)** **{** result **+=** 10**;** **}**

**else** **if** **(**roman**[**i**]** **==** 'L'**)** **{** result **+=** 50**;** **}**

**else** **if** **(**roman**[**i**]** **==** 'C'**)** **{** result **+=** 100**;** **}**

**else** **if** **(**roman**[**i**]** **==** 'D'**)** **{** result **+=** 500**;** **}**

**else** **if** **(**roman**[**i**]** **==** 'M'**)** **{** result **+=** 1000**;** **}**

**}**

**return** result**;**

**}**

// Int To Roman

string intToRoman**(**int j**){**

int tempJ **=** j**;**

string romanNumber **=** ""**;**

th **=** tempJ **/** 1000**;**

tempJ **=** tempJ **%** 1000**;**

h **=** tempJ **/** 100**;**

tempJ **=** tempJ **%** 100**;**

t **=** tempJ **/** 10**;**

tempJ **=** tempJ **%** 10**;**

o **=** tempJ **%** 10**;**

romanNumber **=** romanNumber **+** thousands**[**th**]** **+** hundreds**[**h**]** **+** tens**[**t**]** **+** ones**[**o**];**

//printf("%s\n",romanNumber.c\_str());

**return** romanNumber**;**

**}**

## Josephus Problem

int josephus**(**int n**,** int k**){**

int r **=** 0**;**

int i **=** 1**;**

**while(**i **<=** n**)**

**{**

r **=** **(**r **+** k**)** **%** i**;**

i **+=** 1**;**

**}**

**return** r**+**1**;**

**}**

# Sliding Window

// Cari minimum length yang lebih besar / sama dengan k

int slidingWindow**(**int n**,**int k**){**

int l **=** 0**,** r **=** 0**,** minx **=** 100005**;**

LL sum **=**0**;**

**while(**l **<** n**)**

**{**

**while(**r **<** n **&&** **(**sum **+** data**[**r**])** **<** k**)**

**{**

sum **+=** data**[**r**];**

r**++;**

**}**

// Sum >= k

**if(**sum **+** data**[**r**]** **>=** k **&&** r **<** n**)**

**{**

// printf("sum : %lld data[%d] : %d\n",sum,r,data[r]);

// printf("l : %d (R - L : %d) \n",l,(r+1)-l);

minx **=** min**(**minx**,** **(** **(**r**+**1**)** **-** l**)** **);**

**}**

sum **=** sum **-** data**[**l**];**

l**++;**

**}**

**return** minx**;**

**}**

# Magic Square

data**[**1**][(**n**/**2**)** **+** 1**]** **=** val**;**

int x**,**y**;** // Posisi awal

x **=** 1**;**

y **=** **(**n**/**2**)** **+** 1**;**

val**++;**

int maks**[**n**+**15**];**

RESET**(**maks**,**0**);**

**while(**val **<=** n**\***n**){**

int tempX **=** x**;**

int tempY **=** y**;**

x**--;**

y**++;**

**if(**x **==** 0**)** x **=** n**;**

**if(**y **>** n**)** y **=** 1**;**

// Penentuan Lokasi

**if(**data**[**x**][**y**]** **==** 0**)** data**[**x**][**y**]** **=** val**;**

**else{**

x **=** tempX**;**

y **=** tempY**;**

data**[**x**+**1**][**y**]** **=** val**;**

x**++;**

**}**

val**++;**

**}**

# Parentheses Balancing

stack **<**char**>** S**;**

int stat **=** 0**;**

int len **=** strlen**(**data**);**

REP**(**b**,**len**)**

**{**

**if(**data**[**b**]** **==** '(' **||** data**[**b**]** **==** '['**)** S**.**push**(**data**[**b**]);**

**else**

**{**

**if(**data**[**b**]** **==** ')'**){**

// Validasi

**if(**S**.**size**()** **==** 0 )**{**

stat **=** 1**;**

printf**(**"No\n"**);**

**break;**

**}**

//printf("1. S.top() : %c\n",S.top());

**if(**S**.**top**()** **==** '('**){**

S**.**pop**();**

//printf("Masuk1 di b : %d\n",b);

**}**

**else** **{**

stat **=** 1**;**

printf**(**"No\n"**);**

**break;**

**}**

**}**

**else** **if(**data**[**b**]** **==** ']')**{**

// Validasi

**if(**S**.**size**()** **==** 0 **){**

stat **=** 1**;**

printf**(**"No\n"**);**

**break;**

**}**

//printf("2. S.top() : %c\n",S.top());

**if(**S**.**top**()** **==** '['**)** **{**

//printf("Masuk2 di b : %d\n",b);

S**.**pop**();**

**}**

**else{**

stat **=** 1**;**

printf**(**"No\n"**);**

**break;**

**}**

**}**

**}**

**}**

**if(!**stat**){**

**if(**S**.**size**()** **==** 0**)** printf**(**"Yes\n"**);**

**else** printf**(**"No\n"**);**

**}**