**內容**

[1](#_gjdgxs) Include 2

[2](#_30j0zll) Data Structure 2

[2.1](#_1fob9te) Stack 2

[2.2](#_3znysh7) Queue 2

[2.3](#_2et92p0) Priority Queue 2

[2.4](#_tyjcwt) Set 2

[2.5](#_3dy6vkm) Union-Find Tree 2

[2.6](#_1t3h5sf) Vector 3

[2.7](#_4d34og8) Map 3

[2.8](#_2s8eyo1) BigInt 3

[2.9](#_17dp8vu) BigInt(Java) 5

[2.10](#_3rdcrjn) Treap 6

[2.11](#_26in1rg) Permutaion 6

[2.12](#_lnxbz9) 線段樹RMQ 6

[2.13](#_35nkun2) Binary Indexed Tree 6

[3](#_1ksv4uv) Arithmetic 7

[3.1](#_44sinio) Prime 7

[3.2](#_2jxsxqh) Mod 7

[3.3](#_z337ya) MatrixPower 8

[4](#_3j2qqm3) Graph 8

[4.1](#_1y810tw) 單元最短路徑(Bellman-Ford) 8

[4.2](#_4i7ojhp) 單元最短路徑(Dijkstra) 9

[4.3](#_2xcytpi) 最短路徑樹 9

[4.4](#_1ci93xb) Minimum Spanning Tree 10

[4.5](#_3whwml4) 給定樹根的有向最小生成樹 10

[4.6](#_2bn6wsx) 點著色 11

[4.7](#_qsh70q) 七橋問題 12

[4.8](#_3as4poj) Dijkstra 12

[4.9](#_1pxezwc) Dinic 13

[4.10](#_49x2ik5) Cost Flow 14

[5](#_2p2csry) String 14

[5.1](#_147n2zr) Stringstream 14

[5.2](#_3o7alnk) 最長遞增子序列 15

[5.3](#_23ckvvd) KMP 15

[6](#_ihv636) Geomerty 15

[6.1](#_32hioqz) Point operators 15

[6.2](#_1hmsyys) Intersection of two circles 15

[6.3](#_41mghml) Intersection of two lines 16

[6.4](#_2grqrue) Convex Hull 16

[7](#_vx1227) DP 16

[7.1](#_3fwokq0) 01背包 16

[7.2](#_1v1yuxt) 不限數量背包 16

[7.3](#_4f1mdlm) 分割數 16

[7.4](#_2u6wntf) 重複組合 17

[7.5](#_19c6y18) Matrix Chain Multiplication 17

1. **Include**

#include <bits/stdc++.h>

1. **Data Structure**
   1. Stack

stack<int> stk;

stk.empty(); // true

stk.push(10); // stk: [10]

stk.top(); // 10

stk.pop();

* 1. Queue

queue<int> que;

que.empty(); // true

que.push(10); // que: [10]

que.push(20); // que: [10, 20]

que.front(); // 10

que.back(); // 20;

que.pop(); // que: [20]

* 1. Priority Queue

priority\_queue<int> pq;

pq.empty(); // true

pq.push(10); // pq: {10}

pq.push(20); // pq: {10, 20}

pq.push(15); // pq: {10, 15, 20}

pq.top(); // 20

pt.pop(); // pq: {10, 15}

struct cmp {

bool operator() ( int a, int b ) { return a % 3 < b % 3; }

};

priority\_queue<int, vector<int>, cmp > pq;

pq.empty(); // true

pq.push(10); // pq: {10}

pq.push(20); // pq: {10, 20}

pq.push(15); // pq: {10, 15, 20}

pq.top(); // 20

pt.pop(); // pq: {10, 15}

pq.top(); // 10

* 1. Set

set<int> s;

for (int i = 1; i <= 5; ++i) s.insert( i \* 10 ); // set: 10 20 30 40 50

for (int i = 1; i <= 5; ++i) s.insert( i \* 20 ); // set: 10 20 30 40 50 60 80 100

set<int>::iterator it;

it = s.find(20);

s.erase(it); // set: 10 30 40 50 60 80 100

* 1. Union-Find Tree

init(n): 初始化n個元素，各自成一個獨立的集合

unite(a, b): 合併a, b兩個元素所在的集合

find(x): 得到x所在的集合

int par[MAX\_N], rank[MAX\_N]; // 父節點, 樹深度

void init(int n) {

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

par[i] = i;

rank[i] = 0;

}

}

int find(int x) {

if(par[x] == x) return x;

else return par[x] = find(par[x]);

}

void unite(int x, int y) {

x = find(x), y = find(y);

if(x == y) return;

if(rank[x] < rank[y] par[x] = y;

else {

par[y] = x;

if(rank[x] == rank[y]) rank[x]++;

}

}

bool same(int x, int y) {

return find(x) == find(y);

}

* 1. Vector

int a[5] = { 1, 2, 2, 3, 5 };

vector<int> v ( a, a + 5 );

vector<int>::iterator lb = lower\_bound( v.begin(), v.end(), 2 ); // 1

vector<int>::iterator ub = upper\_bound( v.begin(), v.end(), 3 ); // 4

// erasing from vector

#include <iostream>

#include <vector>

int main ()

{

std::vector<int> myvector;

// set some values (from 1 to 10)

for (int i=1; i<=10; i++) myvector.push\_back(i);

// erase the 6th element

myvector.erase (myvector.begin()+5);

// erase the first 3 elements:

myvector.erase (myvector.begin(),myvector.begin()+3);

std::cout << "myvector contains:";

for (unsigned i=0; i<myvector.size(); ++i)

std::cout << ' ' << myvector[i];

std::cout << '\n';

return 0;

}

* 1. Map

// map::lower\_bound/upper\_bound

#include <iostream>

#include <map>

int main () {

std::map<char,int> mymap;

std::map<char,int>::iterator itlow,itup;

mymap['a']=20;

mymap['b']=40;

mymap['c']=60;

mymap['d']=80;

mymap['e']=100;

itlow=mymap.lower\_bound ('b'); // itlow points to b

itup=mymap.upper\_bound ('d'); // itup points to e (not d!)

mymap.erase(itlow,itup); // erases [itlow,itup)

// print content:

for (std::map<char,int>::iterator it=mymap.begin(); it!=mymap.end(); ++it)

std::cout << it->first << " => " << it->second << '\n';

return 0;

}

* 1. BigInt

struct Bigint{

static const int LEN = 60;

static const int BIGMOD = 10000;

int s;

int vl, v[LEN];

// vector<int> v;

Bigint() : s(1) { vl = 0; }

Bigint(long long a) {

s = 1; vl = 0;

if (a < 0) { s = -1; a = -a; }

while (a) {

push\_back(a % BIGMOD);

a /= BIGMOD;

}

}

Bigint(string str) {

s = 1; vl = 0;

int stPos = 0, num = 0;

if (!str.empty() && str[0] == '-') {

stPos = 1;

s = -1;

}

for (int i=SZ(str)-1, q=1; i>=stPos; i--) {

num += (str[i] - '0') \* q;

if ((q \*= 10) >= BIGMOD) {

push\_back(num);

num = 0; q = 1;

}

}

if (num) push\_back(num);

n();

}

int len() const {

return vl;

// return SZ(v);

}

bool empty() const { return len() == 0; }

void push\_back(int x) {

v[vl++] = x;

// v.PB(x);

}

void pop\_back() {

vl--;

// v.pop\_back();

}

int back() const {

return v[vl-1];

// return v.back();

}

void n() {

while (!empty() && !back()) pop\_back();

}

void resize(int nl) {

vl = nl;

fill(v, v+vl, 0);

// v.resize(nl);

// fill(ALL(v), 0);

}

void print() const {

if (empty()) { putchar('0'); return; }

if (s == -1) putchar('-');

printf("%d", back());

for (int i=len()-2; i>=0; i--) printf("%.4d",v[i]);

}

friend std::ostream& operator << (std::ostream& out,

const Bigint &a) {

if (a.empty()) { out << "0"; return out; }

if (a.s == -1) out << "-";

out << a.back();

for (int i=a.len()-2; i>=0; i--) {

char str[10];

snprintf(str, 5, "%.4d", a.v[i]);

out << str;

}

return out;

}

int cp3(const Bigint &b)const {

if (s != b.s) return s - b.s;

if (s == -1) return -(-\*this).cp3(-b);

if (len() != b.len()) return len()-b.len();//int

for (int i=len()-1; i>=0; i--)

if (v[i]!=b.v[i]) return v[i]-b.v[i];

return 0;

}

bool operator < (const Bigint &b)const{ return cp3(b) < 0; }

bool operator <= (const Bigint &b)const{ return cp3(b) <= 0; }

bool operator == (const Bigint &b)const{ return cp3(b) == 0; }

bool operator != (const Bigint &b)const{ return cp3(b) != 0; }

bool operator > (const Bigint &b)const{ return cp3(b) > 0; }

bool operator >= (const Bigint &b)const{ return cp3(b) >= 0; }

Bigint operator - () const {

Bigint r = (\*this);

r.s = -r.s;

return r;

}

Bigint operator + (const Bigint &b) const {

if (s == -1) return -(-(\*this)+(-b));

if (b.s == -1) return (\*this)-(-b);

Bigint r;

int nl = max(len(), b.len());

r.resize(nl + 1);

for (int i=0; i<nl; i++) {

if (i < len()) r.v[i] += v[i];

if (i < b.len()) r.v[i] += b.v[i];

if(r.v[i] >= BIGMOD) {

r.v[i+1] += r.v[i] / BIGMOD;

r.v[i] %= BIGMOD;

}

}

r.n();

return r;

}

Bigint operator - (const Bigint &b) const {

if (s == -1) return -(-(\*this)-(-b));

if (b.s == -1) return (\*this)+(-b);

if ((\*this) < b) return -(b-(\*this));

Bigint r;

r.resize(len());

for (int i=0; i<len(); i++) {

r.v[i] += v[i];

if (i < b.len()) r.v[i] -= b.v[i];

if (r.v[i] < 0) {

r.v[i] += BIGMOD;

r.v[i+1]--;

}

}

r.n();

return r;

}

Bigint operator \* (const Bigint &b) {

Bigint r;

r.resize(len() + b.len() + 1);

r.s = s \* b.s;

for (int i=0; i<len(); i++) {

for (int j=0; j<b.len(); j++) {

r.v[i+j] += v[i] \* b.v[j];

if(r.v[i+j] >= BIGMOD) {

r.v[i+j+1] += r.v[i+j] / BIGMOD;

r.v[i+j] %= BIGMOD;

}

}

}

r.n();

return r;

}

Bigint operator / (const Bigint &b) {

Bigint r;

r.resize(max(1, len()-b.len()+1));

int oriS = s;

Bigint b2 = b; // b2 = abs(b)

s = b2.s = r.s = 1;

for (int i=r.len()-1; i>=0; i--) {

int d=0, u=BIGMOD-1;

while(d<u) {

int m = (d+u+1)>>1;

r.v[i] = m;

if((r\*b2) > (\*this)) u = m-1;

else d = m;

}

r.v[i] = d;

}

s = oriS;

r.s = s \* b.s;

r.n();

return r;

}

Bigint operator % (const Bigint &b) { return (\*this)-(\*this)/b\*b; }

};

* 1. BigInt(Java)

import java.util.\*;

import java.math.\*;

public class Main {

public static void main(String[] argv){

Scanner s = new Scanner(System.in);

BigInteger temp = BigInteger.ZERO;

BigInteger temp2 = BigInteger.ZERO;

temp = s.nextBigInteger();

temp2 = s.nextBigInteger();

System.out.println( temp + "+" + temp2 + "=" + temp.add(temp2));

System.out.println( temp + "-" + temp2 + "=" + temp.subtract(temp2));

System.out.println( temp + "\*" + temp2 + "=" + temp.multiply(temp2));

System.out.println( temp + "/" + temp2 + "=" + temp.divide(temp2));

}

}

* 1. Treap

pri: 二叉堆(Binary Heap)的值

key: 二元搜尋樹(Binary Search Tree)的值

struct Treap {

Treap \*l, \*r;

int pri, key;

Treap() {}

Treap( int \_key ) : l(NULL), r(NULL), pri(rand()), key(\_key) {}

};

Treap\* merge(Treap \*a, Treap \*b) {

if ( !a || !b ) return a ? a : b;

if ( a->pri > b->pri ) {

a->r = merge ( a->r, b );

return a;

} else {

b->l = merge ( a, b->l );

return b;

}

}

void split( Treap \*t, int k, Treap \*&a, Treap \*&b ) {

if ( !t ) a = b = NULL;

else if ( t->key <= k ) {

a = t;

split( t->r, k, a->r, b );

} else {

b = t;

split( t->l, k, a, b->l ) ;

}

}

Treap\* insert ( Treap \*t, int k ) {

Treap \*a, \*b;

split(t, k, a, b);

return merge( merge( a, new Treap( k ) ), b );

}

Treap remove( Treap \*t, int k ) {

Treap \*a, \*b, \*c;

split(t, k-1, a, b);

split(b, k, b, c);

return merge (a, c );

}

* 1. Permutaion

int arr[] = {1, 2, 3};

do { // 所有列舉結果 } while(next\_permutation(arr, arr + 3);

* 1. 線段樹RMQ

const int MAX\_N = 1 << 17;

int n, dat[2 \* MAX\_N - 1]; // 線段樹

void init(int m) {

n = 1;

while(n < m) n \*= 2;

for(int i = 0; i < 2 \* n - 1; i++) dat[i] = INT\_MAX;

}

void update(int k, int a) {

k += n - 1;

dat[k] = a;

while(k > 0) { // 往樹根更新

k = (k - 1) / 2;

date[k] = min(dat[k \* 2 + 1], dat[k \* 2 + 2]);

}

}

// 求[a, b]的最小值, k是節點編號, l和r表示節點對應[l,r]

// 從外面以query(a, b, 0, 0, n)呼叫

int query(int a, int b, int k, int l, int r) {

if(r <= a || b <= l) return INT\_MAX; // [a, b]和[l, r]沒交錯

if(a <= l && r <= b) return dat[k]; // [a, b]涵蓋[l, r]

else { // 傳回兩個子節點的最小值

int vl = query(a, b, k \* 2 + 1, l, (l + r) / 2);

int vr = query(a, b, k \* 2 + 2, (l + r) / 2, r);

return min(vl, vr);

}

}

* 1. Binary Indexed Tree

int bit[MAX\_N + 1], n;

int sum(int i) {

int s = 0;

while(i > 0) {

s += bit[i];

i -= i & -i;

}

return s;

}

void add(int i, int x) {

while(i <= n) {

bit[i] += x;

i += i & -i;

}

}

1. **Arithmetic**
   1. Prime

#include <bits/stdc++.h>

using namespace std;

const int maxn = 1000;

bitset<maxn> vis;

vector<int> primes;

void gen() {

primes.push\_back(2);

for(int i = 3; i < maxn; i += 2)

if(!vis.test(i)) {

primes.push\_back(i);

for(int j = i \* i; j < maxn; j += i + i) vis.set(j);

}

}

* 1. Mod

typedef long long LL;

LL add\_mod(LL a, LL b, LL n) {

return (a + b) % n;

}

LL sub\_mod(LL a, LL b, LL n) {

return (a – b + n) % n;

}

LL mul\_mod(LL a, LL b, LL n) {

return (a \* b) % n;

}

LL pow\_mod(LL a, LL p) {

if(p == 0) return 1;

LL ans = pow\_mod(a, p / 2);

ans = (long long)ans \* ans % MOD; // Define MOD

if(p % 2) ans = (long long)ans \* a % MOD ;

return ans;

}

// ax + by = gcd(a, b) = d

void gcd(LL a, LL b, LL& d, LL& x, LL& y) {

if(!b) { d = a; x = 1; y = 0; }

else { gcd(b, a % b, d, y, x); y -= x \* (a / b); }

}

// if gcd(a, n) = 1, a^(-1) = x (mod n)

LL inv(LL a, LL n) {

LL d, x, y;

gcd(a, n, d, x, y);

return d == 1 ? ( x + n ) % n : -1 ;

}

// n : equationsx = a[i](mod m[i]) (0 <= i < n)

LL china(int n, int\* a, int\* m) {

LL M = 1, d, y, x = 0;

for(int i = 0; i < n; i++) M \*= m[i];

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

LL w = M / m[i];

gcd(m[i], w, d, d, y)

x = (x + y \* w \* a[i]) % M;

}

return (x + M) % M;

}

// a ^ x == b (mod n)

int log\_mod(int a, int b, int n) {

int m, v, e = 1, i;

m = (int) sqrt(n + 0.5);

v = inv(pow\_mod(a, m, n), n);

map<int, int> x;

x[1] = 0;

for(int i = 1; i < m; ++i) {

e = mul\_mod(e, a, n);

if(!x.count(e)) x[e] = i;

}

for(int i = 0; i < m; ++i) {

if(x.count(b)) return i \* m + x[b];

b = mul\_mod(b, v, n);

}

return -1;

}

// Wilson theorem: iff p is prime, (p-1)! ≡ -1 (mod p)

// Fermat's little theorem: if gcd(a,p)=1, p is prime, then a^(p-1) ≡ 1 (mod p)

// else if gcd(a,p)!=1, p is prime, then a^p ≡ a (mod p)

// Ex: 2^100 mod(13) = 2^(12\*8+4) (mod 13) = 1^8\*2^4 (mod 13) = 3

// Euler theorem: if gcd(a,n)=1, a^φ(n) ≡ 1(mod n)

// If n is prime, then φ(n)=n-1, we get Fermat’s little theorem

* 1. MatrixPower

#include <bits/stdc++.h>

using namespace std;

const int maxn = 110;

template<class T>

struct Matrix {

int r, c;

T a[maxn][maxn];

Matrix() { memset(a, 0, sizeof a); }

Matrix operator\*(Matrix &rhs) {

Matrix<T> ret;

ret.r = r ; ret.c = rhs.c ;

for(int k = 0; k < c; ++k)

for(int i = 0; i < ret.r; ++i)

for(int j = 0; j < ret.c; ++j)

ret.a[i][j] += a[i][k] \* rhs.a[k][j];

return ret;

}

};

template<class T>

T pow(T a, T b) {

T ret = 1, base = a ;

while(b) {

if(b & 1) ret \*= base ;

base = base \* base;

b >>= 1;

}

return ret;

}

1. **Graph**

int n //size

int point\_a //起點

int point\_b //終點

int weight //權重

//=========================adjacency matrix =======================

//以矩陣表示 graph

int graph[N][N]

void adjacency\_matrix(int n)

{

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

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

graph[i][j] = 0;

while (cin >> point\_a >> point\_b >> weight)

graph[point\_a][point\_b] = weight;

}

//============================列表 有向=========================

vector< pair<int,int> > list[5];

* 1. 單元最短路徑(Bellman-Ford)

struct edge { int from, to, cost; }; // 頂點from到頂點to且權重為cost邊

edge es[MAX\_E]; // 邊

int d[MAX\_V]; // 最短距離

int V, E; // V是頂點數, E是邊數

void shortest\_path(int s) { // 從第s個頂點到各頂點最短距離

for(int i = 0; i < V; i++) d[i] = INF;

d[s] = 0;

while(true) {

bool update = false;

for(int i = 0; i < E; i++) {

edge e = es[i];

if(d[e.from] != INF && d[e.to] > d[e.from] + e.cost) {

d[e.to] = d[e.from] + e.cost;

update = true;

}

if(!update) break;

}

}

bool find\_negative\_loop() { // 檢查負的閉路

memset(d, 0, sizeof(d));

for(int i = 0; i <V; i++) {

for(int j = 0; j < E; j++) {

edge e = es[j];

if(d[e.to] > d[e.from] + e.cost) {

d[e.to] = d[e.from] + e.cost;

if(i == V - 1) return true; // 到第n次還有更新就有負閉路

}

}

}

return false;

}

// 全點對最短路徑問題

int d[MAX\_V][MAX\_V]; // d[u][v] e=(u,v)權重(不存在INF, d[i][i] = 0)

int V;

void warshall\_floyd() {

for(int k = 0; k < V; k++)

for(int i = 0; i < V; i++)

for(int j = 0; j < V; j++) d[i][j] = min(d[i][j], d[i][k] + d[i][j];

}

* 1. 單元最短路徑(Dijkstra)

struct edge {int to, cost; };

typedef pair<int, int> P; // first是最短距離, second是頂點編號

int V, d[MAX\_V];

vector<edge> G[MAX\_V];

void dijkstra(int s) {

priority\_queue<P, vector<P>, greater<P> > que;

fill(d, d + V, INF);

d[s] = 0;

que.push(P(0, s));

while(!que.empty()) {

P p = que.top(); que.pop();

int v = p.second;

if(d[v] < p.first) continue;

for(int i = 0; i < G[v].size(); i++) {

edge e = G[v][i];

if(d[e.to] > d[v] + e.cost) {

d[e.to] = d[v] + e.cost;

que.push(P(d[e.to], e.to););

}

}

}

}

* 1. 最短路徑樹

一張有向圖，選定一個起點，找出起點到圖上各點的最短路徑，即是找出其中一棵最短路徑樹。

時間複雜度：O(V^2 + ElogE) W

// 要丟進Priority Queue的點

// b是點，d是可能的最短路徑長度。

// a可以提出來，不必放在Node裡。

struct Node {int b, d;};

bool operator<(const Node& n1, const Node& n2) {return n1.d > n2.d;}

int graph[9][9]; // adjacency matrix

int d[9];

int parent[9];

bool visit[9];

void dijkstra\_with\_priority\_queue(int source)

{

for (int i=0; i<9; i++) visit[i] = false;

for (int i=0; i<9; i++) d[i] = 1e9;

// C++ STL的Priority Queue

priority\_queue<Node> PQ;

d[source] = 0;

parent[source] = source;

PQ.push((Node){source, d[source]});

for (int i=0; i<9; i++)

{

// 找出下一個要加入到最短路徑樹的點。

int a = -1;

while (!PQ.empty() && visit[a = PQ.top().b])

PQ.pop(); // 最後少pop一次，不過無妨。

if (a == -1) break;

visit[a] = true;

for (int b=0; b<9; b++)

if (!visit[b] && d[a] + graph[a][b] < d[b])

{

d[b] = d[a] + graph[a][b];

parent[b] = a;

// 交由Priority Queue比較大小

PQ.push( (Node){b, d[b]} );

}

}

}

* 1. Minimum Spanning Tree

求出無向圖的其中一棵最小（大）生成樹。若是圖不連通，則是求出其中一叢最小（大）生成森林。

int graph[9][9]; // adjacency matrix

int d[9]; // 紀錄目前的MST到圖上各點的距離

int parent[9]; // 紀錄各個點在MST上的父親是誰

bool visit[9]; // 紀錄各個點是不是已在MST之中

void prim()

{

for (int i=0; i<9; i++) visit[i] = false;

for (int i=0; i<9; i++) d[i] = 1e9;

d[0] = 0; // 可以選定任何點作為樹根，這裡以第零點作為樹根。

parent[0] = 0;

for (int i=0; i<9; i++)

{

int a = -1, b = -1, min = 1e9;

for (int j=0; j<9; j++)

if (!visit[j] && d[j] < min)

{

a = j; // 記錄這一條邊

min = d[j];

}

if (a == -1) break; // 與起點相連通的MST都已找完

visit[a] = true;

// d[a] = 0; // 註解後，得到MST每條邊權重。

for (b=0; b<9; b++)

// 以下與Dijkstra's Algorithm略有不同

if (!visit[b] && graph[a][b] < d[b])

{

d[b] = graph[a][b]; // 離樹最近，不是離根最近。

parent[b] = a;

}

}

}

* 1. 給定樹根的有向最小生成樹

//用struct 來諸存兩個點和權重

int V, E;

struct Edge {int a, b, c;} edge[40000];

int d[1000], p[1000], v[1000], n[1000], m[1000];

// 每個點最小入邊的權重，每個點最小入邊的來源，

// 拜訪過，水母環，已收縮。

int MST(int r)

{

memset(m, 0, sizeof(m));

// 目前生成樹的權重，累計收縮水母環而失去的權重。

int w1 = 0, w2 = 0;

while (true) // 一旦形成生成樹就停止。最多執行V-1次。

{

/\* O(E) graph traversal.

find minimum in-edge for each vertice.

--->o

\*/

memset(d, 1, sizeof(d));

memset(p, -1, sizeof(p));

for (int i=0; i<E; ++i)

{

int& a = edge[i].a;

int& b = edge[i].b;

int& c = edge[i].c;

if (a != b && b != r && c < d[b])

d[b] = c, p[b] = a;

}

/\* O(V) jellyfish detection \*/

memset(v, -1, sizeof(v));

memset(n, -1, sizeof(n));

w1 = 0;

bool jf = false;

for (int i=0; i<V; ++i)

{

if (m[i]) continue;

if (p[i] == -1 && i != r) return 1e9;

if (p[i] >= 0) w1 += d[i];

// 找水母環

int s;

for (s = i; s != -1 && v[s] == -1; s = p[s])

v[s] = i;

// 標記水母環上的點，以及將會被收縮掉的點。

if (s != -1 && v[s] == i)

{

jf = true;

int j = s;

do

{

n[j] = s; m[j] = 1;

w2 += d[j]; j = p[j];

} while (j != s);

m[s] = 0;

}

}

if (!jf) break;

/\* O(E) edge reweighting and cycle contraction

\_\_\_

/ \ <-

\\_\_\_/

\*/

for (int i=0; i<E; ++i)

{

int& a = edge[i].a;

int& b = edge[i].b;

int& c = edge[i].c;

if (n[b] >= 0) c -= d[b];

if (n[a] >= 0) a = n[a];

if (n[b] >= 0) b = n[b];

if (a == b) edge[i--] = edge[--E];

}

}

return w1 + w2;

}

* 1. 點著色

無向圖相鄰點不同色

vector<int> G[MAX\_V]; // 圖

int V; // 頂點數

int color[MAX\_V]; // 頂點i的顏色(1 or -1)

bool dfs(int v, int c) { // 用1或-1逐步填滿頂點

color[v] = c;

for(int i = 0; i < G[v].size(); i++) {

if(color[G[v][i]] == c) return false; // 鄰點顏色相同

if(color[G[v][i]] == 0 && !dfs(G[v][i], -c)) return false;

}

return true;

}

void solve() {

for(int i = 0; i < V; i++) {

if(color[i] == 0) {

if(!dfs(i, 1)) {

printf(“No\n”);

return;

}

}

}

printf(“Yes\n”);

}

* 1. 七橋問題

int graph[10][10]; // adjacency matrix，紀錄邊數。

int degree[10]; // 紀錄每個點的邊數

deque< pair<int,int> > edges; // Euler Circuit 經過的邊

// deque<int> vertices; // Euler Circuit 經過的點

bool EulerCircuit()

{

memset(degree, 0, sizeof(degree));

int E = 0; // 總邊數

for (int x=0; x<10; ++x)

for (int y=x; y<10; ++y)

{

// 累計連到 x 點的邊數

degree[x] += graph[x][y];

// 累計連到 y 點的邊數

degree[y] += graph[x][y];

// 累計邊數

E += graph[x][y];

}

for (int i=0; i<10; ++i) // 檢查每個點

if (degree[i] % 2 == 1) // 如果某個點擁有奇數條邊

return false;

int s = -1; // 設定起點

for (int i=0; i<10; ++i)

if (degree[i] > 0) // 有邊的點當作起點

{

s = i;

break;

}

if (s == -1) return false; // 此圖無點

edges.clear();

// vertices.clear();

FindEulerCircuit(s); // 遞迴亂繞圈圈

// 圖可能不連通，檢查是否走完所有邊。

if (edges.size() != E) return false;

// if (vertices.size() != E+1) return false;

return true;

}

void FindEulerCircuit(int x)

{

for (int y=0; y<10; ++y) // 不斷找可以走的邊

if (graph[x][y] > 0)

{

adj[x][y]--;

if (x != y) graph[y][x]--;

FindEulerCircuit(y);

edges.push\_front( make\_pair(x,y) ); // Combine階段才做

// break消失不見

}

// vertices.push\_front(x); // Combine階段才做

}

* 1. Dijkstra

#include <cstdio>

#include <cstring>

#include <queue>

using namespace std;

const int INF = 1000000000;

const int maxn = 80000 + 10;

struct Edge {

int from, to, dist;

};

struct HeapNode {

int d, u;

bool operator < (const HeapNode& rhs) const {

return d > rhs.d;

}

};

struct Dijkstra {

int n, m;

vector<Edge> edges;

vector<int> G[maxn];

bool done[maxn]; // 是否已永久標記

int d[maxn]; // s到各點距離

int p[maxn]; // 最短路中的上一條

void init(int n) {

this->n = n;

for(int i = 0; i < n; i++) G[i].clear();

edges.clear();

}

void AddEdge(int from, int to, int dist) {

edges.push\_back((Edge) {from, to, dist} );

m = edges.size();

G[from].push\_back(m – 1);

}

void dijkstra(int s) {

priority\_queue<HeapNode> Q;

for(int i = 0; i < n; i++) d[i] = INF;

d[s] = 0;

memset(done, 0, sizeof(done));

Q.push((HeapNode) { 0, s } );

while(!Q.empty()) {

HeapNode x = Q.top(); Q.pop();

int u = x.u;

if(done[u]) continue;

done[u] = true;

for(int i = 0; i < G[u].size(); i++) {

Edge& e = edges[G[u][i]];

if(d[e.to] > d[u] + e.dist) {

d[e.to] = d[u] + e.dist;

p[e.to] = G[u][i];

Q.push((HeapNode) { d[e.to], e.to});

}

}

}

}

};

* 1. Dinic

struct Dinic{

static const int MXN = 10000;

struct Edge{ int v,f,re; };

int n,s,t,level[MXN];

vector<Edge> E[MXN];

void init(int \_n, int \_s, int \_t){

n = \_n; s = \_s; t = \_t;

for (int i=0; i<n; i++) E[i].clear();

}

void add\_edge(int u, int v, int f){

E[u].PB({v,f,SZ(E[v])});

E[v].PB({u,0,SZ(E[u])-1});

}

bool BFS() {

FMO(level);

queue<int> que;

que.push(s);

level[s] = 0;

while (!que.empty()){

int u = que.front(); que.pop();

for (auto it : E[u]){

if (it.f > 0 && level[it.v] == -1){

level[it.v] = level[u]+1;

que.push(it.v);

}

}

}

return level[t] != -1;

}

int DFS(int u, int nf){

if (u == t) return nf;

int res = 0;

for (auto &it : E[u]){

if (it.f > 0 && level[it.v] == level[u]+1){

int tf = DFS(it.v, min(nf,it.f));

res += tf; nf -= tf; it.f -= tf;

E[it.v][it.re].f += tf;

if (nf == 0) return res;

}

}

if (!res) level[u] = -1;

return res;

}

int flow(int res=0){

while ( BFS() )

res += DFS(s,2147483647);

return res;

}

} flow;

* 1. Cost Flow

typedef pair<long long, long long> pll;

struct CostFlow {

static const int MXN = 205;

static const long long INF = 102938475610293847LL;

struct Edge {

int v, r;

long long f, c;

};

int n, s, t, prv[MXN], prvL[MXN], inq[MXN];

long long dis[MXN], fl, cost;

vector<Edge> E[MXN];

void init(int \_n, int \_s, int \_t) {

n = \_n; s = \_s; t = \_t;

for (int i=0; i<n; i++) E[i].clear();

fl = cost = 0;

}

void add\_edge(int u, int v, long long f, long long c) {

E[u].PB({v, SZ(E[v]) , f, c});

E[v].PB({u, SZ(E[u])-1, 0, -c});

}

pll flow() {

while (true) {

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

dis[i] = INF;

inq[i] = 0;

}

dis[s] = 0;

queue<int> que;

que.push(s);

while (!que.empty()) {

int u = que.front(); que.pop();

inq[u] = 0;

for (int i=0; i<SZ(E[u]); i++) {

int v = E[u][i].v;

long long w = E[u][i].c;

if (E[u][i].f > 0 && dis[v] > dis[u] + w) {

prv[v] = u; prvL[v] = i;

dis[v] = dis[u] + w;

if (!inq[v]) {

inq[v] = 1;

que.push(v);

}

}

}

}

if (dis[t] == INF) break;

long long tf = INF;

for (int v=t, u, l; v!=s; v=u) {

u=prv[v]; l=prvL[v];

tf = min(tf, E[u][l].f);

}

for (int v=t, u, l; v!=s; v=u) {

u=prv[v]; l=prvL[v];

E[u][l].f -= tf;

E[v][E[u][l].r].f += tf;

}

cost += tf \* dis[t];

fl += tf;

}

return {fl, cost};

}

} flow;

1. **String**
   1. Stringstream

#include <cstdio>

#include <string> // std::string

#include <iostream> // std::cout

#include <sstream> // std::stringstream

int main () {

std::stringstream ss;

ss << 100 << ' ' << 200;

int foo,bar;

ss >> foo >> bar;

std::cout << "foo: " << foo << '\n'; // foo:100

std::cout << "bar: " << bar << '\n'; // bar:200

return 0;

}

* 1. 最長遞增子序列

int n, a[MAX\_N], dp[MAX\_N];

void solve() {

int res = 0;

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

dp[i] = 1;

for(int j = 0; j < i; j++) if(a[j] < a[i])

dp[i] = max(dp[i], dp[j] + 1);

res = max(res, dp[i]);

}

printf(“%d\n”, res);

}

int dp[MAX\_N];

void solve() {

fill(dp, dp + n, INF);

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

\*lower\_bound(dp, dp + n, a[i]) = a[i];

printf(“%d\n”, lower\_bound(dp, dp + n, INF) - dp);

}

* 1. KMP

// 次長的相同前綴後綴

void getFail(char \*P, int \*f) {

int m = strlen(P);

f[0] = f[1] = 0;

for(int i = 1; i < m; ++i) {

int j = f[i];

while(j && P[i] != P[j]) j = f[j];

f[i + 1] = P[i] == P[j] ? j + 1 : 0;

}

}

void find(char \*T, char \*P, int \*f) {

int n = strlen(T), m = strlen(P);

getFail(P,f);

for(int i = 0, j = 0; i < n; ++i) {

while(j && P[j] != T[i]) j = f[j];

if(P[j] == T[i]) ++j;

if(j == m) printf(“%d\n”, i – m + 1);

}

}

1. **Geomerty**
   1. Point operators

#define x first

#define y second

#define cpdd const pdd

struct pdd : pair<double, double> {

using pair<double, double>::pair;

pdd operator + (cpdd &p) const {

return {x+p.x, y+p.y};

}

pdd operator - () const {

return {-x, -y};

}

pdd operator - (cpdd &p) const {

return (\*this) + (-p);

}

pdd operator \* (double f) const {

return {f\*x, f\*y};

}

double operator \* (cpdd &p) const {

return x\*p.x + y\*p.y;

}

};

double abs(cpdd &p) { return hypot(p.x, p.y); }

double arg(cpdd &p) { return atan2(p.y, p.x); }

double cross(cpdd &p, cpdd &q) { return p.x\*q.y - p.y\*q.x; }

double cross(cpdd &p, cpdd &q, cpdd &o) { return cross(p-o, q-o); }

pdd operator \* (double f, cpdd &p) { return p\*f; } // Not f\*q

* 1. Intersection of two circles

using ld = double;

vector<pdd> interCircle(pdd o1, double r1, pdd o2, double r2) {

ld d2 = (o1 - o2) \* (o1 - o2);

ld d = sqrt(d2);

if (d > r1+r2) return {};

pdd u = 0.5\*(o1+o2) + ((r2\*r2-r1\*r1) / (2\*d2)) \* (o1-o2);

double A = sqrt((r1+r2+d) \* (r1-r2+d) \* (r1+r2-d) \* (-r1+r2+d));

pdd v = A / (2\*d2) \* pdd(o1.S-o2.S, -o1.F+o2.F);

return {u+v, u-v};

}

* 1. Intersection of two lines

const double EPS = 1e-9;

pdd interPnt(pdd p1, pdd p2, pdd q1, pdd q2, bool &res) {

double f1 = cross(p2, q1, p1);

double f2 = -cross(p2, q2, p1);

double f = (f1 + f2);

if(fabs(f) < EPS) {

res = false;

return {};

}

res = true;

return (f2 / f) \* q1 + (f1 / f) \* q2;

}

* 1. Convex Hull

vector<pdd> convex\_hull(vector<pdd> pt) {

sort(pt.begin(),pt.end());

int top=0;

vector<pdd> stk(2\*pt.size());

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

while (top >= 2 && cross(stk[top-2],stk[top-1],pt[i]) <= 0) top--;

stk[top++] = pt[i];

}

for (int i=pt.size()-2, t=top+1; i>=0; i--) {

while (top >= t && cross(stk[top-2],stk[top-1],pt[i]) <= 0)

top--;

stk[top++] = pt[i];

}

stk.resize(top-1);

return stk;

}

1. **DP**
   1. 01背包

int dp[MAX\_N + 1][MAX\_W + 1];

void solve() {

for(int i = n - 1; i >= 0; i--) {

if(j < w[i]) dp[i][j] = dp[i + 1][j];

else dp[i][j] = max(dp[i+1][j], dp[i+1][j - w[i]] + v[i]);

}

printf(“%d\n”, dp[0][W]);

}

//重量大 價值小

int dp[MAX\_N + 1][MAX\_N \* MAX\_V + 1];

void solve() {

fill(dp[0], dp[0] + MAX\_N \* MAX\_V + 1, INF);

dp[0][0] = 0;

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

for(int j = 0; j <= MAX\_N \* MAX\_V; j++) {

if(j < v[i]) dp[i + 1][j] = dp[i][j];

else dp[i + 1][j] = min(dp[i][j], dp[i][j – v[i]] + w[i]);

}

}

int res = 0;

for(inti = 0; i <= MAX\_N \* MAX\_V; i++) if(dp[n][i] <= W) res = i;

printf(“%d\n”, res);

}

* 1. 不限數量背包

void solve() {

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

for(int j = 0; j <= W; j++) {

if(j < w[i]) dp[i + 1][j] = dp[i][j];

else

dp[i+1][j]=max(dp[i][j],dp[i+1][j-w[i]]+v[i]);

}

}

printf(“%d\n”, dp[n][W]);

}

* 1. 分割數

將n個無法互相區別的物品分割成m個以下的方法總數

// 輸入n, m

int n, m, dp[MAX\_N + 1][MAX\_M + 1];

void solve() {

dp[0][0] = 1;

for(inti = 1; i <= m; i++) {

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

if(j - i >= 0) dp[i][j] = (dp[i - 1][j] + dp[i][j - 1]) % M;

else dp[i][j] = dp[i - 1][j];

}

}

printf(“%d\n”, dp[m][n]);

}

* 1. 重複組合

n種物品，i號物品有a*i*個不同種類物品可區分，同種類不能

// 輸入n, m

int n, m, a[MAX\_N + 1][MAX\_M + 1];

void solve() {

for(inti = 0; i <= n; i++) dp[i][0] = 1; // 總會有一種都沒選的方法

for(inti = 0; i < n; i++) {

for(int j = 1; j <= m; j++) {

if(j - 1 - a[i] >= 0)

dp[i + 1][j] = (dp[i + 1][j - 1] + dp[i][j] - dp[i][j - 1- a[i]] + M) % M;

else dp[i + 1][j] = (dp[i + 1][j - 1] + dp[i][j]) % M;

}

}

printf(“%d\n”, dp[n][m]);

}

* 1. Matrix Chain Multiplication

int dp[100][100];, r[100], c[100];

void solve() {

fill(dp, dp + 100 \* 100, INF);

for(int i = 0; i < N; i++) dp[i][i] = 0;

for(int k = 1; k < N; k++)

for(int i = k - 1; i >= 0; i--)

for(int j = k - 1; j >= 0; j--)

dp[i][k] = min(dp[i][k], dp[i][j] + dp[j + 1][k] + r[i] \* c[j] \* c[k]);

Common Sums

|  |
| --- |
|  |

Combinations

|  |
| --- |
|  |