1 C++

1.1 萬用標頭檔 / Buffer

```
/*TLE時沒更好做法可以嘗試砸砸看
#pragma GCC optimize("Ofast","inline","-ffast-math","
    no-stack-protector")
#pragma GCC optimize ("unroll-loops")
#pragma GCC target("avx,sse2,sse3,sse4,mmx")
*/
#include <bits/stdc++.h>
#define MiruSort ios::sync_with_stdio(0);cin.tie(0);
using ll = long long;
```

1.2 便利函式

1.2.1 next_permutation

is_permutation：判斷陣列 b 是否爲陣列 a 排序後的結果。
next_permutation：使用已經排序 (由小到大) 的資料, 產生下一組排列。
prev_permutation：針對已經「逆向」排序的資料, 產生上一組排序。

1.2.2 ceil, floor

ceil() 無條件進位，floor() 無條件捨去

1.2.3 nth_element()

nth_element
(contain.begin(), contain.begin() + n, contain.end());
排序一個容器，第 [n] 的元素維持不變，
第 [n] 個元素之前的所有元素都不大於該元素，而該元素後的所有元素也不小於該元素。

1.2.4 字典序比較函式

lexicographical_compare
(iter1 beg1, iter1 end1, iter2 beg2, iter2 end2);
// return true if 1 < 2

1.3 STL/資結

1.3.1 vector

```
vector<int> v; //宣告
v.emplace_back(12); //新增一元素
v[i] // 與陣列同
v.front() //回傳 vector 第一個元素的值。
v.back() //回傳 vector 最尾元素的值。
cout << "v有" << v.size() << "個元素\\n";
v.clear(); // 清除
v.pop_back() //刪除最尾端的元素。
v.insert()  //插入元素至任意位置。
v.erase() //刪除 vector 中一個或多個元素。
```

1.3.2 deque

```
deque<int> d = {1, 2, 3, 4};//宣告與賦值
用法幾乎同vector 只是多了front的操作
```

1.3.3 pair #include <utility>

```
pair<int, double> p1;//宣告
pair<int, double> p2(1, 2.4);//初始化
pair<int, double> p3(p2);//拷貝
cout << p2.first << " " << p2.second;//取值1 2.4
p1 = make_pair(1, 1.2);//賦值
//比較時先比first後比second
```

1.3.4 tuple

```
tuple<T1, T2, TN> t1;//宣告
tuple<T1, T2, TN> t2(v1, v2, ... TN);//初始化
tuple<string, double, int> t3(t2);
make_tuple(v1, v2); // make_tuple創建一個tuple對象
get<0>(t2);//取值 = v1
t2_length = tuple_size<decltype(t2)>::value
```

1.3.5 set

```
set<int> myset{1, 2, 3, 4, 5}; // 初始化
//也可使用陣列的方式初始化
int arr[] = {1, 2, 3, 4, 5};
set<int> myset(arr, arr + 5);
myset.insert(6); //插入元素
myset.find(element); // if can't find, return end()
for (auto it = myset.begin(); it != myset.end(); it++)
    scout << *it << " ";
//使用迭代器印出set內元素
myset.erase(2); myset.clear(); myset.empty();
strcut cmp // set cmp
{
    bool operator(const int &a, const int &b){
        return a > b;
    }
};
```

1.3.6 bitset

```
bitset<5> b; //初始化 00000
b[0] = 1;//賦值
b.set();//使全部都變成1
b.reset();//使全部都變成0
b.count(); // 輸出b中有幾個 1
b.flip(); // 01互換
bitset<5> a(10011);///用法同pair and tuple
string str = a.to_string();// to str
```

1.3.7 map

```
map<string, string> mapStudent;
mapStudent["r123"] = "student_first";
mapStudent["r456"] = "student_second";
auto iter = mapStudent.find("r000");
// key: iter->first value: iter->second
mapStudent.size()//size
```

1.3.8 unordered_map

```
//初始化
unordered_map<string, :string> umap =
{
    {"RED", "#FF0000"},
    {"GREEN", "#00FF00"}
};
//用 "array" 方式插入
umap["BLACK"] = "#000000";
umap.erase(umap.begin()); //清除第一個元素
for (const auto &n : umap)
    cout << "Key:[" << n.first << "] Value:[" << n.
    second << "]\\n";
//尋訪整個unordered map
umap.clear(); umap.empty();
```

1.3.9 list

其他用法同 vector，

```
prev(iterator) // 返回iterator前一個iterator
next(iterator) // 返回iterator後一個iterator
assign(count, value); // 給list count個value
erase // give range or pos
insert(pos, value) //插入一個元素到list中
merge() //合併兩個 sorted list
remove(value) //從list刪除等於value的元素
remove_if([](int n){ return n > 10; }) //按指定條件刪
    除元素
size() //返回list中的元素個數
sort() //給list排序
splice() //合併兩個list
swap() //交換兩個list
unique() //刪除list中重複的元素
/*--------   tips  --------*/
auto it = list.erase(it):
it.insert(num);
//equals to replace
/*--------   tips  --------*/
```

1.3.10 priority_queue

其他用法同 queue

```
priority_queue<int>; //大的優先取出
priority_queue<int,vector<int>,greater<int> > //小的
```

1.3.11 set_union, set_intersection, set_difference, set_symmetric_difference

將 A 和 B 兩個集合的聯集, 交集, 差集, 對稱差集複製到 C 序列上
set_union(A 集合頭, 尾, B 集合頭, 尾, C 序列頭)

```
auto i = set_union(first.begin(), first.end(), second.
    begin(), second.end(), u.begin());
u.resize(i - u.begin());
```

1.3.12 Disjoint Set

```
int lead[maxn], size[maxn];
void init(int N)
{
    for(int i = 0; i < N; ++i)
    {
        lead[i] = i;
        size[i] = 1;
    }
}
int find(int x) // 尋找父節點
{
    if(lead[x] == x)
        return x;
    else
        return lead[x] = find(lead[x]); // 路徑壓縮
}
bool same(int a, int b) // 判斷是否屬於同一集合
{
    return find(a) == find(b);
}
void merge(int a, int b) // 合併集合
{
    int fa = find(a), fb = find(b);
    if(fa < fb)
        swap(fa, fb);
    size[fa] += size[fb];
    lead[fb] = fa;
}
```

1.3.13 BIT 樹狀樹組

```
void update(int x, int d) // add d to x-th one
{
    while(x <= N)
        b[x] += d, x += x & (-x);
}
int query(int x) // sum from 1 to x
{
    int ret = 0;
    while(x) // x != 0
        ret += b[x], x -= x & (-x);
    return ret;
}
```

1.3.14 segment tree

```
ll total[100005 << 2], lazy[100005 << 2];

void pull(int x)
{
    total[x] = total[x<<1] + total[x<<1|1];
}

void push(int x, int l, int r)
{
    if (lazy[x] == 0)
        return;
    int mid = (l + r) >> 1;
    lazy[x<<1] += lazy[x];
    lazy[x<<1|1] += lazy[x];
    total[x<<1] += lazy[x] * (mid - l + 1);
    total[x<<1|1] += lazy[x] * (r - mid);
    lazy[x] = 0;
}

void build(int x, int l, int r)
{
    if (l == r)
    {
        cin >> total[x];
        return;
    }
    int mid = (l + r) >> 1;
    build(x<<1, l, mid);
    build(x<<1|1, mid+1, r);
    pull(x);
}

void update(int x, int l, int r, int ul, int ur)
{
    if (ul <= l && r <= ur)
    {
        lazy[x] += c;
        total[x] += c * (r-l+1);
        return;
    }
    push(x, l, r);
    int mid = (l + r) >> 1;
    if(ul <= mid)
        update(x<<1, l, mid, ul, ur);
    if(mid < ur)
        update(x<<1|1, mid+1, r, ul, ur);
    pull(x);
}

long long query(int x, int l, int r, int ql, int qr)
{
    if(ql <= l && r <= qr)
        return total[x];
    long long ans = 0;
    int mid = (l + r) >> 1;
    push(x, l, r);
    if(ql <= mid)
        ans += query(x<<1, l, mid, ql, qr);
    if(mid < qr)
        ans += query(x<<1|1, mid+1, r, ql, qr);
    return ans;
}
```

1.3.15 持久化線段樹

```
struct node
{
    int val;
    node* left, *right;
    node() {}
    node(node* l, node* r, int v)
    {
        left = l;
        right = r;
        val = v;
    }
};
int arr[MAXN]; // input
node* version[MAXN]; // root pointers for all versions

void build(node* n, int low, int high)
{
    if (low == high)
    {
        n->val = arr[low];
        return;
    }
    int mid = (low+high) / 2;
    n->left = new node(NULL, NULL, 0);
    n->right = new node(NULL, NULL, 0);
    build(n->left, low, mid);
    build(n->right, mid + 1, high);
    n->val = n->left->val + n->right->val;
}

void upgrade(node* prev, node* cur, int low, int high,
                                    int idx, int value)
{
    if(idx > high or idx < low or low > high)
        return;
    if(low == high)
    {
        cur->val = value;
        return;
    }
    int mid = (low+high) / 2;
    if(idx <= mid)
    {
        cur->right = prev->right;
        // create new node in current version
        cur->left = new node(NULL, NULL, 0);
        upgrade(prev->left, cur->left, low, mid, idx,
    value);
    }
    else
    {
        cur->left = prev->left;
        cur->right = new node(NULL, NULL, 0);
        upgrade(prev->right, cur->right, mid + 1, high
    , idx, value);
    }
    // calculating data for current version by
    combining previous version and current
    cur->val = cur->left->val + cur->right->val;
}

int query(node* n, int low, int high, int l, int r)
{
    if (l > high or r < low or low > high)
       return 0;
    if (l <= low and high <= r)
       return n->val;
    int mid = (low+high) / 2;
    int p1 = query(n->left, low, mid, l, r);
    int p2 = query(n->right, mid + 1, high, l, r);
    return p1 + p2;
}

int main()
{
    int A[] = {1, 2, 3, 4, 5};
    int n = sizeof(A) / sizeof(int);
    for (int i = 0; i < n; i++)
        arr[i] = A[i];
    node* root = new node(NULL, NULL, 0);
    build(root, 0, n - 1);
    version[0] = root;
    version[1] = new node(NULL, NULL, 0);
    upgrade(version[0], version[1], 0, n - 1, 4, 1);
    version[2] = new node(NULL, NULL, 0);
    upgrade(version[1], version[2], 0, n - 1, 2, 10);
    cout << "In version 1 , query(0,4) : ";
    cout << query(version[1], 0, n - 1, 0, 4) << endl;
    return 0;
}
```

1.3.16 Preficx sum

```
int prefixSum1D[10000] = {0};
for(int i = 1; i < 10000; ++i)
{
    cin >> num;
    prefixSum1D[i] = prefixSum1D[i - 1] + num;
}
// 輸入區間範圍(l,r)，[r] - [l-1]的結果就是所求區間的
    和
int prefixSum2D[10000][10000] = {0};
fo(int i = 1; i <= n; i++)
{
    for(int j = 1; j <= n; j++)
    {
        cin >> prefixSum2D[i][j];
        prefixSum2D[i][j] += prefixSum2D[i][j-1];
    }
}
```

1.4 模板

1.4.1 自訂義 cmp

```
template <typename T>
bool cmp(T &x, T &y) { return x > y; /* reverse sort
    */ }
```

1.4.2 二分搜模板

```
int lower_bound(vector<int> nums, int target)
{
    int l = 0, r = nums.size(), m = 0;
    while(l < r)
    {
        m = (l + r) >> 1;
        if(nums[m] < target) l = m + 1;
        // add = means upper bound
        else r = m;
    }
    return l;
    //if return is nums.size() means no lower_bound
}
```

1.4.3 fast pow

```
long long binpow(long long a, long long b)
{
    long long res = 1;
    while (b > 0)
    {
        if (b & 1)
            res = res * a;
        a = a * a;
        b >>= 1;
    }
    return res;
}
```

1.4.4 矩陣快速冪

```
struct Matrix A
{
    int a, b, c, d;
    // a b
    // c d
    Matrix operator*(Matrix const& r)const;
    Matrix(int a, int b,int c,int d);
};
Matrix pow(Matrix& A, int e)
{
    if(e & 1)
        return pow(A, e - 1) * A;
    else
        return 乃pow(A * A, e / 2);
}
int F(int n)
{
    Matrix A(1, 1, 1, 0);
    return pow(A, n).b;
}
```

1.4.5 數據離散化

```
vector<int> solve(vector <int> a)
{
    vector<int> b = a;
    sort(b.begin(), b.end());
    b.resize(unique(b.begin(), b.end()) - b.begin());
    for(int i=0;i<a.size();i++)
    {
        a[i] = lower_bound(b.begin(), b.end(), a[i])
            - b.begin() + 1;
    }
    return a;
}
```

1.4.6 Quick Sort

```
int a[maxn];
int partition(int l, int r)
{
    int p = a[r], ls = l;
    // p := pivot, ls := less equal
    for (int i = l; i < r; i++)
        if (a[i] <= p)
            swap(a[i], a[ls++]);
    swap(a[r], a[ls]);
    return ls;
}

void quicksort(int l, int r)
{ // 左閉右閉區間 [l, r]
    if (l >= r)
        return;
    int s = partition(l, r);
    quicksort(l, s - 1);
    quicksort(s + 1, r);
}

int main()
{
    quicksort(0, n - 1);
    return 0;
}
```

1.4.7 Merge Sort

```
#include <stdio.h>

typedef long long LL;
LL inv(int a[], int le, int ri)
{
    if (le + 1 >= ri)
        return 0;
    int m = (le + ri) / 2;
    LL w = inv(a, le, m) + inv(a, m, ri), cross = 0;
    int temp[ri - le], j = m, k = 0;
    for(int i = le; i < m; i++)
    {
        while (j < ri && a[j] < a[i])
            temp[k++] = a[j++];
        temp[k++] = a[i];
        cross += j - m; // num <a[i]
    }
    for(k = le; k < j; k++)
        a[k] = temp[k - le];
    return w + cross;
}
int main()
{
    int i, n, ar[100010];
    scanf("%d", &n);
    for (i = 0; i < n; i++)
        scanf("%d", ar + i);
    printf("%lld\n", inv(ar, 0, n));
    return 0;
}
```

1.4.8 Topological Sort

拓樸排序是一個排序，使得對於任意有向邊 (u, v)，u 都在 v 之前；
一個圖有拓樸排序若且唯若它是一個 DAG。

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

#define pb push_back
int N; // Number of nodes
vector<int> graph[100000], top_sort;
// Assume that this graph is a DAG
bool visited[100000];

void dfs(int node)
{
    for(int i : graph[node])
    {
        if(!visited[i])
        {
            visited[i] = true;
            dfs(i);
        }
    }
    top_sort.pb(node);
}
void compute()
{
    for(int i = 0; i < N; i++)
    {
        if(!visited[i])
        {
            visited[i] = true;
            dfs(i);
        }
    }
    reverse(begin(top_sort),end(top_sort));
}
int main()
{
    int M; cin >> N >> M;
    for(int i = 0; i < M; ++i)
    {
        int a, b; cin >> a >> b;
        graph[a - 1].pb(b - 1);
```

```
    }
    compute();
    vector<int> ind(N);
    for(int i = 0; i < N; i++)
        ind[top_sort[i]] = i;
    for(int i = 0; i < N; i++)
    {
        for(int j: graph[i])
        {
            if(ind[j] <= ind[i])
            {
                cout << "IMPOSSIBLE\n";
                exit(0);
            }
        }
    }
    for(int i : top_sort)
        cout << i + 1 << " ";
    cout << "\n";
}
```

1.4.9 KMP 尋找子字串

```
bool kmpSearch(string s, string key)
{
    int pat[50] = {0}; // build patten
    for(int i = 1; i < (int)s.length(); i++)
    {
        int pos = pat[i-1];
        while(s[pos] != s[i] && pos)
            pos = pat[pos-1];
        if(s[pos] == s[i]) pat[i] = pos+1;
        else pat[i] = 0;
    }
    int si = 0, keyi = 0;
    while(si < s.length() && keyi < key.length())
    {
        if(s[si] == key[keyi])
        {
            si++;
            keyi++;
        }
        else if(keyi <= 0) si++;
        else keyi = pat[keyi - 1];
    }
    if(keyi == key.length()) return true;
    else return false;
}
```

1.4.10 sstream

```
#include <sstream>
#include <iostream>
using namespace std;
int main(){
    stringstream s1;
    int N, i1;
    while(cin >> N){
        cin.ignore();
        string line;
        for(int i = 0; i < N; i++){
            getline(cin, line);
            int sum = 0;
            s1.clear();
            s1 << line; // or s1.str(line);
            while(true){
                s1 >> i1;
                if(s1.fail())
                    break;
                sum += i1;
            }
            cout << sum << endl;
        }
    }
    return 0;
}
```

1.5 Brute Force

1.5.1 DFS Subset 爆搜

```
void DFS(long long count, long long now)
{
    if(count == n)
    {
        if(now == k)
            flag = true;
        return;
    }
    visit[count] = false;
    DFS(count + 1, now);
    visit[count] = true;
    DFS(count + 1, now + arr[count]);
}
```

1.5.2 DFS Permutation

```
void DFS(int u)
{
    if(u == n)
    {
        cout << permutation << '\n';
        return;
    }
    for(int i = 0; i < n; i++)
    {
        if(used[i])
            continue; //i已被放過
        permutation.push_back(input[i]);
        used[i] = true; //把i標記爲放過
        DFS(u + 1);
        permutation.pop_back();
        used[i] = false;
    }
}
```

1.6 Dynamic Programming

1.6.1 0/1 knapsack

```
const int N = 100, W = 100000;
// 物品的價值與重量，合併成一個物件，讓人容易理解。
struct Item {int cost, weight;} items[N];
int c[W + 1];

void knapsack(int n, int w)
{
    memset(c, 0, sizeof(c));

    for (int i = 0; i < n; ++i)
    {
        int weight = items[i].weight;
        int cost = items[i].cost;
        for (int j = w; j - weight >= 0; --j)
            c[j] = max(c[j], c[j - weight] + cost);
    }

    cout << "最高的價值爲" << c[w];
}
```

1.6.2 LIS

```
int num;
vector<int> a;
while(cin >> num)
    a.push_back(num);
int N = (int)a.size();
int dp[N+1];
vector<int> v;
dp[0] = 1;
v.push_back(a[0]);
int L = 1; //LIS length
```

```
for (int i=1; i<N; i++)
{
    if (a[i] > v.back())
    {
        v.push_back(a[i]);
        L++;
        dp[i] = L;
    }
    else
    {
        auto it = lower_bound(v.begin(),v.end(),a[i]);
        *it = a[i];
        dp[i] = (int) (it - v.begin() + 1);
    }
}
cout << L << "\n-\n";
vector<int> ans;
for (int i=N-1; i>=0; i--)
{
    if (dp[i] == L)
    {
        ans.push_back(a[i]);
        L--;
    }
}
reverse(ans.begin(), ans.end());
for (auto i: ans)
    cout << i << '\n';
```

1.6.3 LCS

```
int l1, l2;
while(cin >> s1 >> s2){
    l1 = (int)s1.length();
    l2 = (int)s2.length();
    memset(dp, 0, sizeof(dp));
    for(int i = 1; i <= l1; i++){
        for(int j = 1; j <= l2; j++){
            if(s1[i-1] == s2[j-1])
                dp[i][j] = dp[i-1][j-1] + 1;
            else
                dp[i][j] = max(dp[i][j-1], dp[i-1][j])
    ;
        }
    }
    cout << dp[l1][l2] << endl;
}
```

1.6.4 最長迴文子字串

```
string longestPalindrome(string s)
{
    int ansi = 0, ansj = 0, l = s.length();
    if (l == 0)
        return "";
    int a[l][l]; // 用來表示從
    memset(a, 0, sizeof(a));
    for(int i = 0; i < l; ++i)
        a[i][i] = 1; // 自己一個字元必爲迴文
    for(int i = 0; i < l - 1; ++i){
        if (s[i] == s[i + 1]){
            a[i][i + 1] = 1;
            ansi = i;
            ansj = i + 1;
        }
    }
    for(int j = 2; j < l; j++){
        for(int i = 0; i < l-j; i++){
            if(a[i+1][i+j-1] == 1 && s[i] == s[i+j])
            {
                a[i][i+j] = 1;
                ansi = i;
                ansj = i + j;
            }
        }
    }
    return s.substr(ansi, ansj - ansi + 1);
}
```

1.7 圖論

1.7.1 DFS

```
const int vertex = 1000;
int inStamp[vertex] = {0}, outStamp[vertex] = {0};
int stamp = 1;
vector<int> Graph[vertex];
void dfs(int u)
{
    inStamp[u] = stamp++;
    for(int v: Graph[u])
        if(inStamp[v] == 0)
            dfs(v);
    outStamp[u] = stamp++;
}
```

1.7.2 BFS

```
const int vertex = 1000;
vector<int> Graph[vertex];
int level[vertex];
void init()
{
    memset(level, -1, sizeof(level));
}
void bfs(int s)
{
    queue<int> q;
    q.push(s); level[s] = 0;
    while(q.size())
    {
        int u = q.front();q.pop();
        for(int v: Graph[u])
        {
            if(level[v] != -1)
                continue;
            level[v] = level[u] + 1;
            q.push(v);
        }
    }
}
```

1.7.3 子樹大小

```
const int vertex = 1000;
vector<int> Graph[vertex];
int sz[vertex];
int dfs(int u, int parent)
{
    sz[u] = 1;
    for(int v: Graph[u])
        if(v != parent)
            sz[u] += dfs(v, u);
    return sz[u];
}
```

1.7.4 MST Kruskal's algorithm

```
struct Edge{
    int u, v, w;
};
bool cmp(Edge a, Edga b){
    return a.w < b.w;
}
vector<Edge> edgeList;
int kruskal()
{
    init();
    int weight = 0;
    sort(edgeList.begin(), edgeList.end(), cmp);
    for(auto k: edgeList)
        if(merge(k.u, k.v)) // 是否無環，無環加入DSU
            weight += k.w;
    return weight;
}
```

1.7.5 樹直徑

```
const int MAXN = 1e3 + 5;
vector<int> Graph[MAXN];
int level[MAXN];
void init(); // code in BFS
void bfs(int s) // code in BFS
int getDiameter()
{
    init();BFS(0);
    int maxLevel = -1, maxIndex = -1;
    for(int i = 0; i < MAXN; ++i)
    {
        if(maxLevel < level[i])
        {
            maxLevel = level[i];
            maxIndex = i;
        }
    }
    init();BFS(maxIndex);
    maxLevel = -1, maxIndex = -1;
    for(int i = 0; i < MAXN; ++i)
    {
        if(maxLevel < level[i])
        {
            maxLevel = level[i];
            maxIndex = i;
        }
    }
    return maxLevel;
}
```

1.7.6 Dijakstra 單源最短路徑

```
#define INF 0x3F3F3F3F3F3F3F3F
#define pairlli pair<long long, int>
const int maxn = 20000;
vector<pair<int, int>> G[maxn]; //adj list
long long dist[maxn];

void addEdge(int u, int v, int w)
{
    G[u].push_back({v, w});
    G[v].push_back({u, w});
}
void Dijkstra(int S, int T)
{
    memset(dist, 0x3F, sizeof(dist));
    priority_queue<pairlli,
        vector<pairlli>, greater<pairlli> > pq;
    dist[S] = 0;
    pq.push({0, S});
    while(!pq.empty())
    {
        pairlli p = pq.top();
        pq.pop();
        if (dist[p.second] < p.first) continue;
        for(auto nxt: G[p.second])
        {
            if(dist[nxt.first] >
                dist[p.second] + nxt.second)
            {
                dist[nxt.first] =
                    dist[p.second] + nxt.second;
                pq.push({dist[nxt.first], nxt.first});
            }
        }
    }
    if(dist[T] == INF)
        cout << "-1 ";
    else
        cout << dist[T] << ' ';
}
int main()
{
    int n, m, S, T;
    int u, v, w;
    for (int i = 0; i < maxn; i++)
        G[i].clear(); // init
    cin >> n >> m >> S >> T; // n nodes and m edges,
    find shortest path from S to T
    for(int i = 0; i < m; i++) // constructor
    {
        cin >> u >> v >> w;
        addEdge(u, v, w);
    }
    Dijkstra(S, T);
    return 0;
}
```

1.7.7 Bellman-Ford

```
const ll INF = 1e18;
int n, dist[maxn];
vector<Edge>edgeList;
void bellmanFord(int s)
{
    for(int i = 0; i < maxn; ++i)
        dist[i] = INF;
    dist[s] = 0;
    for(int i = 0; i < n - 1; ++i)
        for(auto e: edgeList)
            dist[e.v] = min(dist[e.v], dist[e.u] + e.w
    );
}
```

1.7.8 Bellman-Ford 偵測負環

```
const ll INF = 1e18;
int n, dist[maxn];
vector<Edge>edgeList;
bool negative_cycle()
{
    for(int i = 0; i < maxn; ++i)
        dist[i] = INF;
    for(int i = 0; i < n; ++i)
    {
        for(auto e: edgeList)
        {
            if(dist[e.v] > dist[e.u] + e.w)
            {
                dist[e.v] = dist[e.u] + e.w;
                if(i == n - 1)
                    return true;
            }
        }
    }
    return false;
}
```

1.7.9 Floyd Warshall 一般圖全點對最短路徑

```
int vertex = 100;
int dist[vertex][vertex];
for(int k = 0; k < vertex; ++k)
    for(int i = 0; i < vertex; ++i)
        for(int j = 0; j < vertex; ++j)
            dist[i][j] =
                min(dist[i][j],dist[i][k]+dist[k][j]);
```

2 競賽數學

2.1 數論

2.1.1 取模

對於任兩非負整數 p, q：

$$(p+q)\%M = (p\%M + q\%M)\%M$$
$$(p*q)\%M = (p\%M * q\%M)\%M$$
$$(p-q)\%M = (p\%M(M - q\%M))\%M$$

2.1.2 質數埃氏篩法

```
const int N = 20000000;
bool notprime[N + 5];
void eratosthenes_sieve()
{
    notprime[0] = notprime[1] = true;
    for(int i = 4; i <= N; i += 2)
        notprime[i] = true;
    for(int i = 3; i * i <= N; i += 2)
        if(!notprime[i])
            for(int j=N/i, k=i*j; j >= i; --j, k -= i)
                if(!notprime[j])
                    notprime[k] = true;
}
```

2.1.3 質數線性篩法

```
const int N = 20000000;
bool notprime[N + 5];
vector<int> prime;

void linear_sieve(){
    for(int i = 2; i <= N; ++i){
        if(!notprime[i])
            prime.push_back(i);
        for(auto p1 : prime){
            if (p1 * i > N) break;
            notprime[p1 * i] = true;
            if (i % p1 == 0) break;
        }
    }
}
```

2.1.4 篩法 DP

在篩法過程中，把篩法表型別由 bool 改成 int sieve
存當前數字被哪個質數篩掉。
1. 求質因數個數 (重覆也算，$2^3 \times 3^5$ 算 8)

```
dp[1] = 0;
dp[i] = dp[i / sieve[i]] + 1;
```

2. 拿來求質因數個數 (重覆不算)

```
dp[1] = 0;
int t = i;
while (t % sieve[i] == 0)
    t /= sieve[i];
dp[i] = dp[t] + 1;
```

3. 拿來求各質因數次數 2. 拿來求質因數個數 (重覆不算)

```
memset(dp[1], 0, sizeof(dp[1]))
memcpy(dp[i], dp[i / sieve[i]], sizeof(dp[0]));
++dp[i][idx[sieve[i]]];
```

2.1.5 倍數

被 3 整除:
若是由 1 n 組成的數字 (例如:123456)，看 n 是否被 3 整除或餘 2，若是問 n，則看位數合是否爲三個倍數
7, 13: 從個位數開始，每三個位數分成一節，奇數節和減掉偶數節和爲該數倍數
11: 奇位數和減掉偶位數和爲該 11 的倍數

2.1.6 因數和

建立質數表後進行質因數分解，$int = a^x \times b^y \times c^z$ 之因數和爲 $\frac{a^{x+1}-1}{a-1} \times \frac{b^{y+1}-1}{b-1} \times \frac{c^{z+1}-1}{c-1}$

2.1.7 找到介於 1 - n 之間之平方數與立方數

```
res = (ll)sqrt(input) + (ll)cbrt(input) - (ll)(sqrt(
    cbrt(input)));
```

2.1.8 歐拉函數

小於 x 的正整數中與 x 互質的數的數目 1. 類埃氏篩法求法

```
void sieve(int N)
{
    for (int i = 1; i <= N; ++i)
        phi[i] = i; // 一開始讓 phi_i = i
    for (int i = 1; i <= N; ++i)
        for (int j = 2 * i; j <= N; j += i)
            phi[j] -= phi[i];
}
```

2. 單個歐拉函數求法

```
int Phi(int x)
{
    if (x < 2)
        return 0;
    int ret = x;
    int sq = sqrt(x);
    for (int p=2; p<=sq; p++)
    {
        if (x % p == 0)
        {
            while (x % p == 0) x /= p;
            ret -= ret / p;
        }
        if (x == 1)
            break;
    }
    if (x > 1)
        ret -= ret / x;
    return ret;
}
```

2.1.9 擴展歐幾里得算法

用於求出形式如 ax + by = c 的整數解

```
void extgcd(int a, int b, int& x, int& y)
{
    if(b)
    {
        extgcd(b, a % b, y, x); // 計算子問題的解
        y -= (a / b) * x;
    }
    else
        x = 1, y = 0; // 最小子問題解
}
```

2.1.10 模逆元

$a \times a^{-1} = 1(modp)$，此時的 a^{-1} 爲模逆元

```
int exgcd(int a, int b, int &x, int &y)
{
    if(!b)
    {
        x = 1, y = 0;
        return a;
    }
    int ans = exgcd(b, a % b, x, y);
    int t = x, x = y, y = t - a / b * y;
    return ans;
}
int main()
{
    int x, y; // a is input, mod is const
    int g = exgcd(a, mod, x, y);
    if (g != 1)
        cout << "No solution!";
    else
    {
        x = (x % mod + mod) % m;
        cout << x << endl;
    }
}
```

2.1.11 模逆元應用

計算 $C_m^n = \frac{n!}{m! \times (n-m)!}$ $C_m^n = pre[n] \times pre_i[m] \times pre_i[n-m]$

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

const ll MAXN = 100005, ll mod = 998244353;
ll pre[MAXN];
ll pre_i[MAXN];
ll inv[MAXN];

ll f_pow(ll a, ll b)
{
    ll base = a, cou = 1;
    while(b)
    {
        if(b & 1)
            cou *= base;
        base *= base;
        cou %= mod;
        base %= mod;
        b >>= 1;
    }
    return cou;
}
ll f_inv(ll n)
{
    return f_pow(n, mod - 2);
}
void build(ll n)
{
    pre[0] = pre[1] = 1;
    pre_i[0] = pre_i[1] = 1;
    inv[0] = inv[1] = 1;
    for(int i = 2; i <= n; ++i)
    {
        pre[i] = pre[i-1] * i % mod;
        inv[i] = f_inv(i);
        pre_i[i] = pre_i[i - 1] * inv[i] % mod;
    }
}
ll C(ll n, ll m)
{
    return pre[n] * pre_i[m] % mod * pre_i[n - m] %
    mod;
}
int main()
{
    build(MAXN); // 模逆元表建立
    cout << inv[4] << endl; // 4's inverse
    cout << C(6,3) << endl;
    return 0;
}
```

2.1.12 miller-robin

已知數字範圍在 int 內，取 a = 2, 7, 61，如果三個都通過考驗，那麼它必定是質數；long long 內，取 2, 325, 9375, 28178, 450775, 9780504, 1795265022。

```
bool miller_rabin(int n, int a)
{
    if((n & 1) == 0) return n == 2;
    int u = n - 1, t = 0; // 分解次方
    while((u & 1) == 0)
        u >>= 1, ++t;
    int x = pow(a, u, n); // x = a ^ u % n;
    if(x == 1 || x == n - 1)
        return true;
    for(int i = 0; i < t - 1; ++i)
    { // 算冪次剩下的部分，最後一次不用算
        x = mul(x, x, n); // x = x * x % n;
        if(x == 1) return false;  // 方根測試失敗
        if(x == n - 1) return true; // 偷窺未來必成功
    }
    return false;
}
```

2.1.13 三分搜

```
int l = -10000, r = 10000, iterations = 100;
for(int i = 0; i < iterations; i++)
{
    double mr = (l + r) / 2.0;
    double ml = (l + mr) / 2.0;
    if (f(ml) < f(mr)) // f( ): 目標函數
        r = mr;
    else
        l = ml;
}
```

2.2 應用數學

2.2.1 矩陣

```
const int MAXROW = 3;
const int MAXCOL = 3;

struct row_vec {
    double col[MAXCOL];
};
row_vec A[MAXROW];
```

2.2.2 高斯消去法

基礎列運算

```
void FirstRowOpt(int i, int j){
    swap(A[i], A[j]);
}
void SecondRowOpt(int i, double k){
    for (int j = 0; j < MAXCOL; ++j)
        A[i].col[j] *= k;
}
void ThirdRowOpt(int i, int j, double k){
    for (int l = 0; l < MAXCOL; ++l)
        A[j].col[l] += k * A[i].col[l];
}
```

```
void GaussianElimination()
{
    int pivot = -1;
    for(int i = 0; i < MAXROW; ++i)
    { // pick a pivot row
        bool flag = 1;
        while(flag)
        {
            ++pivot;
            if(pivot == MAXCOL)
                break;
            if(A[i].col[pivot])
                flag = 0;
            int maxabsi = i;
            for(int j = i + 1; j < MAXROW; ++j)
            {
                if(fabs(A[j].col[pivot]) >
                    fabs(A[maxabsi].col[pivot]))
                {
                    maxabsi = j;
                    flag = 0;
                }
            }
            FirstRowOpt(i, maxabsi);
        }
        SecondRowOpt(i, 1.0 / A[i].col[pivot]);
        for(int j = i + 1; j < MAXROW; ++j)
            ThirdRowOpt(i, j, -A[j].col[pivot]);
    }
    // 以下code將矩陣運算爲簡化列梯形
    for(int i = MAXROW - 1; i >= 1; --i)
    {
        for(int j = 0; j < MAXCOL; ++j)
        { // find pivot
            if(A[i].col[j])
            {
```

```
                for(int k = i - 1; k >= 0; --k)
                    ThirdRowOpt(i, k, -A[k].col[j]);
                break;
            }
        }
    }
}
```

2.2.3 Determinant

將方陣消去至上三角矩陣並計算其行列式值
記錄過程中列運算對行列式值的影響
即可得到原矩陣的行列式值
第一型列運算：$det(R_{ij}A) = -det(A)$
第二型列運算：$det(R_i^{(k)}A) = k \cdot det(A)$
第三型列運算：$det(R_{ij}^{(k)}A) = det(A)$

2.2.4 盧卡斯定理

C_m^n mod p，將 n 跟 m 轉成 p 進位則上式結果可以拆成 p 進位後各自計算後再相乘。
舉例 $C_6^{13} \equiv C_0^1 \cdot C_6^6 \equiv 1 \pmod 7$

10 進位	7 進位
13	16
6	06

2.3 位元運算性質

2.3.1 計算 1 - n 的 XOR

```
int computeXOR(int n)
{
    if(n % 4 == 0)
        return n;
    if(n % 4 == 1)
        return 1;
    if(n % 4 == 2)
        return n + 1;
    return 0;
}
```

2.4 計算幾何

2.4.1 向量模板

```
struct Point
{
    double x, y;
    bool operator < (const Point &b)const
    {   // 比較
        return tie(x, y) < tie(b.x, b.y);
    }
    Point operator + (const Point &b)const
    {   // 向量加法
        return {x + b.x, y + b.y};
    }
    Point operator - (const Point &b)const
    {   // 向量減法
        return {x - b.x, y - b.y};
    }
    Point operator * (const double &d)const
    {  // 向量伸縮
        return {d * x, d * y};
    }
    double operator * (const Point &b)const
    {  // 向量內積
        return x * b.x + y * b.y;
    }
    double operator % (const Point &b)const
    {  // 向量外積
        return x * b.y - y * b.x;
    }
    double operator == (const Point &b)const
    {
        if(abs(x - b.x) <= eps && abs(y - b.y) <= eps)
            return true;
        return false;
    }
};
```

2.4.2 外積

二維之外積 $(x_1, y_1) \times (x_2, y_2) = x_1y_2 + x_2y_1$
$\vec{AB} \times \vec{AP} > 0$，則點 P 在 $\vec{AB}$ 的逆時針方向 $\vec{AB} \times \vec{AP} < 0$，則點 P 在 $\vec{AB}$ 的順時針方向
$\vec{AB} \times \vec{AP} = 0$，則討論以下情況:
1. $\vec{AB} \cdot \vec{AP} < 0$，則點 P 在 $\vec{AB}$ 的反向延長線上
2. $\vec{AB} \cdot \vec{AP} \geq 0$ and $\| \vec{AB} \| < \| \vec{AP} \|$，點 P 在 $\vec{AB}$ 的延長線上
3. $\vec{AB} \cdot \vec{AP} \geq 0$ and $\| \vec{AB} \| \geq \| \vec{AP} \|$，點 P 在 $\vec{AB}$ 上

3 Python

3.1 Input and Output Buffer

```
from sys import stdin, stdout
x, y = map(int, stdin.readline().strip().split())
# 讀取兩個整數
inputlist = [int(x) for x in stdin.readline().strip().
    split()]
# 讀取一整行整數
stdout.write(ans + '\n')
# ans 必須爲一個字串、行尾不會自動換行
# eof
for i in sys.stdin:
    n = int(i) // n 行
    ans = ""
    for i in range(n):
        s = sys.stdin.readline()
        data = s.split()
        for j in data:
            ans += int(j)
    stdout.write(ans + '\n')
```

3.2 十進位浮點數 & 分數

```
from decimal import *
from sys import stdout
from fractions import *
getcontext().rounding = getattr(decimal,'ROUND_CEILING
    ')
# 無條件進位
getcontext().rounding = getattr(decimal,'ROUND_FLOOR')
# 無條件捨去
getcontext().prec = 10
# 顯示到小數點後第9位，第10位四捨五入
x = Decimal('0.1') # 建議使用字串，才不會有誤差
y = Decimal('0.1')
print(x + y) # output = 0.2
print(float(x + y) == 0.2) # output == true
mol = Decimal(-7) % Decimal(4)
# mol = -3, 乃C++之取餘數規則
ans = str(Fraction('1.2')) # 建議搭配decimal使用
stdout.write(ans)
# output = 6/5 最簡分數 .numerator 分子
# .denominator 分母
```

4 心法

4.1 如果一直錯誤時要注意甚麼

1. 注意變數名稱，進行各種運算或判斷時有沒有判斷錯變數
2. 注意開頭跟結尾依照題目要求在不在判斷範圍內

4.2 秘訣

1. 遇到排序問題可以考慮把值 index 交換找規律
2. 數學題找規律