堯哥 cheat

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

using ll = long long;
using vi = vector<int>;
using pii = pair<int, int>;
using pll = pair<ll, ll>;
template<typename T> using vec = vector<T>;
template<typename T> using Prior = priority_queue<T>;
template<typename T> using prior = priority_queue<T, vector<T>,
greater<T>>;

#define yccc ios_base::sync_with_stdio(false), cin.tie(0)
#define al(a) a.begin(),a.end()
#define F first
#define S second
#define REP(i, n) for(int i = 0; i < n; i++)
#define REP1(i, n) for(int i = 1; i <= n; i++)
#define eb emplace_back
#define pb push_back
#define mp(a, b) make_pair(a, b)
#define debug(x) cout << " > " << #x << ": " << x << endl;
#define devec(v) cout << " > " << #v << ": "; for (auto i : v) cout << i
<< ' '; cout << endl;
#define devec2(v) cout << " > " << #v << ":\n"; for (auto i : v) { for
(auto k : i) cout << ' ' << k; cout << endl; }

const int INF = 1e9;
const int nINF = -1e9;
const ll llINF = 4*1e18;
const int MOD = 1e9+7;

ll& pmod(ll& a, ll b) { a = (a+b) % MOD; return a; }
ll& pmod(ll& a, ll b, ll c) { a = (a+b) % c; return a; }
ll& mmod(ll& a, ll b) { a = (a-b+MOD) % MOD; return a; }
ll& mmod(ll& a, ll b, ll c) { a = (a-b+c) % c; return a; }
ll& tmod(ll& a, ll b) { a = (a*b) % MOD; return a; }
ll mul(ll a, ll b) { return (a*b) % MOD; }
ll mul(ll a, ll b, ll c) { return (a*b) % c; }
```

```
vim setting :
syntax enable
syntax off
set number
set nonumber
set tabstop=4
colo default/darkblue/koehler/desert/ron/torte
```

---

data structure

Segment Tree

```cpp
#define IL(X) ((X * 2) + 1)
#define IR(X) ((X * 2) + 2)
#define MAXN 500005
struct Node{
    long long sum;
    long long lazy_tag;
    int size;
};
short dataseq[MAXN];
Node seq[MAXN * 4 + 5];
void build(int l, int r, int L, int R, int index);
void  modify(int l, int r, int L, int R, int index, long long Add);
long long Query(int l, int r, int L, int R, int index);
void pull(int index);
void push(int index);

void pull(int index){
    seq[index].sum = seq[IL(index)].sum + seq[IR(index)].sum;
    seq[index].size = seq[IL(index)].size + seq[IR(index)].size;
    seq[index].lazy_tag = 0;
}
void push(int index){
    seq[IL(index)].lazy_tag += seq[index].lazy_tag;
    seq[IL(index)].sum += seq[index].lazy_tag * seq[IL(index)].size;
    seq[IR(index)].lazy_tag += seq[index].lazy_tag;
    seq[IR(index)].sum += seq[index].lazy_tag * seq[IR(index)].size;
    seq[index].lazy_tag = 0;
}

void build(int l, int r, int L, int R, int index){
    if(l == r){
        seq[index].sum = dataseq[l];
        seq[index].size = 1;
        seq[index].lazy_tag = 0;
        return;
    }
    int M = (L + R) / 2;
    build(l, M, L, M, IL(index));
        build(M + 1, r, M + 1, R, IR(index));
    pull(index);
}

void modify(int l, int r, int L, int R, int index, long long Add){
    if(l == L && r == R){
        seq[index].lazy_tag += Add;
        seq[index].sum += Add * seq[index].size;
        return;
    }
    push(index);
    int M = (L + R) / 2;

    if(r <= M){
        modify(l, r, L, M, IL(index), Add);
    }else if(l > M){
        modify(l, r, M + 1, R, IR(index), Add);
    }else{
        modify(l, M, L, M, IL(index), Add);
        modify(M + 1, r, M + 1, R, IR(index), Add);
    }
    pull(index);
}

long long Query(int l, int r, int L, int R, int index){
    if(l == L && r == R){
        return seq[index].sum;
    }
    int M = (L + R) / 2;
    push(index);

    pull(index);
    if(r <= M){
        return Query(l, r, L, M, IL(index));
    }else if(l > M){
        return Query(l, r, M + 1, R, IR(index));
    }else{
        return Query(l, M, L, M, IL(index)) +
        Query(M + 1, r, M + 1, R, IR(index));
    }
}
```

樹狀樹組：

```cpp
#define lowbit(x) (x˙&(-x))
int bit [MAX_N+1]={0};
int i=1
while(i<=n){
            //陣列a存放原始資料
            int ans=0;
            for(int ii=i ; ii>=(i-lowbit(i)+1); ){
                        if ( bit[ii] != 0){
                                    ans+=bit[ii];
                                    ii-=lowbit(ii);
                        }
                        else{
                                    ans+=a[ii];
                                    ii--
                        }
            }
            bit[i]=ans;
            i++;
}
long long query_sum(int x){
            long long ans=0;
            for(;x;x-=lowbit(x)) ans+=bit[x];
            return ans;
}
void add(int x,int v){
            for(;x<=N;x+=lowbit(x)) bit[x]+=v; //N應該是元素個數
}
```

Sparse Table

```cpp
int n;
int v[1000009];
int sparse[22][1000009];
// O(nlogn) preprocess O(1)Query
// sp[x][y] is the answer from (v[x], v[x+2^y-1])
inline void init()
{
    for (int i = 0; i < n; ++i)
        sparse[0][i] = v[i];
    for (int j = 1; (1 << j) <= n; ++j)
        for (int i = 0; i + (1 << j) <= n; ++i)
        sparse[j][i] = min(
        sparse[j - 1][i],
        sparse[j - 1][i + (1 << (j - 1))]
        );
}

inline int query(int l, int r)
{
    int k = __lg(r - l + 1);
    return min(sparse[k][l], sparse[k][r - (1 << k) + 1]);
}
```

Graph

Minimum Spanning Tree

Kruskal

```cpp
struct DSU // disjoint set no rank-comp-merge
{
    vector<int> fa;
    DSU(int n) : fa(n) { iota(fa.begin(), fa.end(), 0); } // auto fill fa
from 0 to n-1
    int find(int x) { return fa[x] == x ? x : fa[x] = find(fa[x]); }
    void merge(int x, int y) { fa[find(x)] = find(y); }
};
int kruskal(int V, vector<tuple<int, int, int>> E) // save all edges into
E, instead of saving graph via adjacency list
{
    sort(E.begin(), E.end());
    DSU dsu(V);
    int mcnt = 0;
    int ans = 0;
    for (auto e : E)
    {
        int w, u, v; // w for start, u for des, v for val
        tie(w, u, v) = E;
        if (dsu.find(u) == dsu.find(v))
            continue;
        dsu.merge(u, v);
        ans += w;
        if (++mcnt == V - 1)
            break;
    }
    return ans;
}
```

Prim

```cpp
#define enp pair<int, int> // pair<edge_val, node>
int prim_pq(vector<vector<enp>> E){
    vector<bool> vis;
    vis.resize(E.size(), false);
    vis[0] = true;
    priority_queue<enp> pq;
    for(auto e: E[0]){
        pq.emplace(-e.first, e.second);
```

```
    }
    int ans = 0; // min value for MST
    while(pq.size()){
        int w, v; // edge-weight, vertex index
        tie(w, v) = pq.top();
        pq.pop();
        if(vis[v])
            continue;
        w = -w;
        vis[v] = true;
        ans += w;
        for(auto e: E[v]){
            pq.emplace(-e.first, e.second);
        }
    }
    return ans;
}
```

Shortest Path Bellman-Ford

```
#include <iostream>
#include <vector>
#include <list>
#include <utility>          // for std::pair<>
#include <iomanip>          // for std::setw()

const int Max_Distance = 100;
class Graph_SP{             // SP serves as Shortest Path
private:
    int num_vertex;
    std::vector<std::list<std::pair<int,int>>> AdjList;
    std::vector<int> predecessor, distance;
public:
    Graph_SP():num_vertex(0){};
    Graph_SP(int n):num_vertex(n){
        AdjList.resize(num_vertex);
    }
    void AddEdge(int from, int to, int weight);
    void PrintDataArray(std::vector<int> array);
    void InitializeSingleSource(int Start);     // 以Start作為起點
    void Relax(int X, int Y, int weight);       // 對edge(X,Y)進行Relax
    bool BellmanFord(int Start = 0);            // 以Start作為起點
                                                // if there is negative
cycle, return false
};

bool Graph_SP::BellmanFord(int Start){

    InitializeSingleSource(Start);

    for (int i = 0; i < num_vertex-1; i++) {                // |V-1|次的
iteration
        // for each edge belonging to E(G)
        for (int j = 0 ; j < num_vertex; j++) {             // 把AdjList最
外層的vector走一遍
            for (std::list<std::pair<int,int> >::iterator itr =
AdjList[j].begin();
                 itr != AdjList[j].end(); itr++) {          // 各個vector
中，所有edge走一遍
                Relax(j, (*itr).first, (*itr).second);
            }
        }
    }

    // check if there is negative cycle
    for (int i = 0; i < num_vertex; i++) {
        for (std::list<std::pair<int,int> >::iterator itr =
AdjList[i].begin();
             itr != AdjList[i].end(); itr++) {
            if (distance[(*itr).first] > distance[i]+(*itr).second) {   //
i是from, *itr是to
                return false;
            }
        }
    }

    return true;
}
void Graph_SP::InitializeSingleSource(int Start){

    distance.resize(num_vertex);
    predecessor.resize(num_vertex);

    for (int i = 0; i < num_vertex; i++) {
        distance[i] = Max_Distance;
        predecessor[i] = -1;
    }
    distance[Start] = 0;
}
void Graph_SP::Relax(int from, int to, int weight){
    if (distance[to] > distance[from] + weight) {
        distance[to] = distance[from] + weight;
        predecessor[to] = from;
    }
}
void Graph_SP::AddEdge(int from, int to, int weight){
    AdjList[from].push_back(std::make_pair(to,weight));
}
```

Dijkstra

```
vector<vector<con>> Graph; //
vector<int> dis; // distance;
priority_queue<con> pq;
    pq.emplace(con(dis[0] = 0, 0));

    while(pq.size()){
        con cur = pq.top();
        pq.pop();
        for(auto it: Graph[cur.second]){

            if(dis[it.second] != it.first) continue;
            dis[it.second] = min(dis[it.second], cur.first + it.first);
            pq.push(con(dis[it.second], it.second));
        }
    }
```

Floyd-Warshall

```
// all pairs shortest path
#include <iostream>
#include <vector>
#include <iomanip>      // for setw()

const int MaxDistance = 1000;
class Graph_SP_AllPairs{
private:
    int num_vertex;
    std::vector< std::vector<int> > AdjMatrix, Distance, Predecessor;
public:
    Graph_SP_AllPairs():num_vertex(0){};
    Graph_SP_AllPairs(int n);
    void AddEdge(int from, int to, int weight);
    void PrintData(std::vector< std::vector<int> > array);
    void InitializeData();
    void FloydWarshall();
};

Graph_SP_AllPairs::Graph_SP_AllPairs(int n):num_vertex(n){
    // Constructor, initialize AdjMatrix with 0 or MaxDistance
    AdjMatrix.resize(num_vertex);
    for (int i = 0; i < num_vertex; i++) {
        AdjMatrix[i].resize(num_vertex, MaxDistance);
        for (int j = 0; j < num_vertex; j++) {
            if (i == j){
                AdjMatrix[i][j] = 0;
            }
        }
    }
}
void Graph_SP_AllPairs::InitializeData(){
    Distance.resize(num_vertex);
    Predecessor.resize(num_vertex);
    for (int i = 0; i < num_vertex; i++) {
        Distance[i].resize(num_vertex);
        Predecessor[i].resize(num_vertex, -1);
        for (int j = 0; j < num_vertex; j++) {
            Distance[i][j] = AdjMatrix[i][j];
              if (Distance[i][j] != 0 && Distance[i][j] != MaxDistance) {
                Predecessor[i][j] = i;
            }
        }
    }
}
void Graph_SP_AllPairs::FloydWarshall(){

    InitializeData();

    std::cout << "initial Distance[]:\n";
    PrintData(Distance);
    std::cout << "\ninitial Predecessor[]:\n";
    PrintData(Predecessor);

    for (int k = 0; k < num_vertex; k++) {
        std::cout << "\nincluding vertex(" << k << "):\n";
        for (int i = 0; i < num_vertex; i++) {
            for (int j = 0; j < num_vertex; j++) {
                if ((Distance[i][j] > Distance[i][k]+Distance[k][j])
                     && (Distance[i][k] != MaxDistance)) {
                    Distance[i][j] = Distance[i][k]+Distance[k][j];
                    Predecessor[i][j] = Predecessor[k][j];
                }
            }
        }
        // print data after including new vertex and updating the shortest
paths
        std::cout << "Distance[]:\n";
        PrintData(Distance);
        std::cout << "\nPredecessor[]:\n";
        PrintData(Predecessor);
    }
}
void Graph_SP_AllPairs::PrintData(std::vector< std::vector<int> > array){

    for (int i = 0; i < num_vertex; i++){
        for (int j = 0; j < num_vertex; j++) {
            std::cout << std::setw(5) << array[i][j];
        }
        std::cout << std::endl;
    }
}
void Graph_SP_AllPairs::AddEdge(int from, int to, int weight){
    AdjMatrix[from][to] = weight;
}
```

Lowest_Common_Ancestor

```
#define MAXN 200005
#define MAXLOG 200
int D[MAXN];
int P[MAXLOG][MAXLOG];
#include <cmath>
#include <algorithm>
using namespace std;
#define MAXN 200005
#define MAXLOG 200
int N = MAXN;
int lgN = log(N) / log(2);
int D[MAXN];
int P[MAXLOG][MAXLOG];
int LCA(int u, int v)
{
    if (D[u] > D[v])
        swap(u, v);
    int s = D[v] - D[u]; // adjust D until D[v] = D[u]
    for (int i = 0; i <= lgN; ++i)
        if (s & (1 << i))
            v = P[v][i];
    if (u == v)
        return v;
```

```
    // because they are at same depth
    // jump up if they are different
    // think about that if P[u][i] == P[v][i]
    // then that point must be the ancestor of LCA or LCA itself
    // by this, we will stop at LCA's child
    for (int i = lgN; i >= 0; --i)
        //
        if (P[u][i] != P[v][i])
        {
            u = P[u][i];
            v = P[v][i];
        }
    return P[u][0];
}

void ComputeP()
{
    int n = N;
    for (int i = 0; i < lgN; ++i) // to lgN enough
    {
        for (int x = 0; x < n; ++x)
        {
            if (P[x][i] == -1)
                P[x][i + 1] = -1;
            else
                P[x][i + 1] = P[P[x][i]][i]; // equal to move on the
parent direction
                // And P[x][i] move 2 ^ n steps to a parent we call it y
                // P[y][i] means continue move 2 ^ n step from y to a
parent we call z
                // so the total equal to move 2 ^ n * 2 ^ n steps from x
to z
                // which is move 2 ^ (n + 1) steps to z
        }
    }
}
```

TREE-Centroid

```
  int subTsize[200005];
vector<int> adj[200005];
int n; // n for node num ??
pair<int, int> Tree_Centroid(int v, int pa)
{
    // return (最 大 子 樹 節 點 數 , 節 點ID)
    subTsize[v] = 1;
    pair<int, int> res(INT_MAX, -1); // ans: tree cnetroid
    int max_subT = 0; // 最大子樹節點數
    for (size_t i = 0; i < adj[v].size(); ++i)
    {
        int x = adj[v][i];
        if (x == pa)
            continue;
        res = min(res, Tree_Centroid(x, v));
        subTsize[v] += subTsize[x];
        max_subT = max(max_subT, subTsize[x]);
    }
    res = min(res, make_pair(max(max_subT, n - subTsize[v]), v)); // (n -
subTsize[v]) for maybe parent tree is the biggest
    // min because all res will be greater than n/2;
    // the min one is the tree centroid
    return res;
}
```

Kosaraju_for_SCC

```
class Kosaraju_for_SCC{
    int NodeNum;
    vector<vector<int>> G;
    vector<vector<int>> GT;
    stack<int> st;
    vector<bool> visited;
    vector<int> scc;
    int sccID;

public:
    void init(int N){
        NodeNum = N;
        G.clear();
        G.resize(N + 5);
        GT.clear();
        GT.resize(N + 5);
        while(!st.empty())
            st.pop();
        visited.clear();
        visited.resize(N + 5, false);
        scc.clear();
        scc.resize(N + 5);
        sccID = 1;
    }
    void addEdge(int w, int v){
        G[w].emplace_back(v);
        GT[v].emplace_back(w);
    }
    void DFS(bool isG, int v, int k = -1){
        visited[v] = true;
        scc[v] = k;
        vector<vector<int>> &dG = (isG ? G : GT);
        for(int w: dG[v])
        {
            if(!visited[w]){
                DFS(isG, w, k);
            }
        }
        if(isG){
            st.push(v);
        }
    }
    void Kosaraju(int N){
        visited.clear();
        visited.resize(N + 5, false);
        for (int i = 1; i <= N; i++){
            if(!visited[i])
                DFS(true, i);
        }
        visited.clear();
        visited.resize(N + 5, false);
        while(!st.empty()){
            if(!visited[st.top()])
                DFS(false, st.top(), sccID++);
            st.pop();
        }
    }
    vector<vector<int>> generateReG(){
        vector<vector<int>> reG;
        reG.resize(sccID);
        for (int i = 1; i <= NodeNum; i++){
            for(int w: G[i]){
               if(scc[i] == scc[w])
                   continue;
               reG[scc[i]].emplace_back(scc[w]);
            }
        }
        return reG;
    }
};
```

Tarjan_for_SCC

```
class tarjan_for_SCC{
private:
    vector<vector<int>> G; // adjacency list
    vector<int> D;
    vector<int> L;
    vector<int> sccID;
    stack<int> st; // for SccID
    vector<bool> inSt;
    vector<vector<int>> reG;
    int timeStamp, sccIDstamp;
public:
    void init(int size = 1){
        G.clear();
        G.resize(size + 3);
        D.clear();
        D.resize(size + 3, 0);
        L.clear();
        L.resize(size + 3, 0);
        sccID.clear();
        sccID.resize(size + 3, 0);
        while(!st.empty())
            st.pop();
        inSt.clear();
        inSt.resize(size + 3, false);
        reG.clear();
        sccIDstamp = timeStamp = 1;
    }
    void addEdge(int from, int to){
        G[from].emplace_back(to);
    }
    void DFS(int v, int pa){ //call DFS(v,v) at first
        D[v] = L[v] = timeStamp++; //timestamp > 0
        st.push(v);
        inSt[v] = true;

        for(int w: G[v]){ // directed graph don't need w == pa
            if(!D[w]){ // D[w] = 0 if not visited
                DFS(w, v);
                L[v] = min(L[v], L[w]);
            }else if(inSt[w])
            { /* w has been visited.
                if we don't add this, the L[v] will think that v can back
to node whose index less to v.
                !inSt[w] is true that v -> w is a forward edge
                opposite it's a cross edge
            */
                L[v] = min(L[v], D[w]); // why D[w] instead of L[w]??
            }
        }
        if(D[v] == L[v]){
            int w;
            do{
                w = st.top();
                st.pop();
                sccID[w] = sccIDstamp; // scc ID for this pooint at which
SCC
                inSt[w] = false;
            } while (w != v);
            sccIDstamp++;
        }
    }
    void generateReG(int N = 1){
        reG.clear();
        reG.resize(sccIDstamp);
        for (int i = 1; i <= N; i++){
            for(int w: G[i]){
                if(sccID[i] == sccID[w])
                    continue;
                reG[sccID[i]].emplace_back(sccID[w]);
            }
        }
    }
    bool visited(int v){
        return D[v];
    }
};
```

Tarjan for ArticulationPointBridge

```
class tarjan{
    vector<vector<int>> G; // adjacency List
    vector<int> D;  // visit or visited and D-value
    vector<int> L;  // for L-value
    vector<con> edgeBridge;
    vector<int> APnode;
    int timestamp;
    tarjan(int size = 1){
        timestamp = 0;
        G.resize(size);
        D.resize(size, 0);
        L.resize(size, 0);
        edgeBridge.clear();
```

```
        APnode.clear();
    }
    void init(int size = 1){
        tarjan(size);
    }
    void addedge(int u, int v)
    {   // undirected graph
        G[u].push_back(v);
        G[v].push_back(u);
    }
    void DFS(int v, int pa)
    { // 使 用 DFS(v,v) 來 呼 叫 函 數
        D[v] = L[v] = timestamp++;
        int Childcount = 0;
        bool isAP = false;
        for(int w: G[v]){
            if(w == pa)
                continue;
            if(!D[w])
            { // 用 D[w] = 0 當 作 沒 走 過
                DFS(w, v);
                Childcount++;
                if(D[v] <= L[w])
                    isAP = true; // 結 論 2 對於除了 r 點以外的所有點 v，v 點
在 G 上為 AP 的充要條件為其在 T 中至少有一個子節點 w 滿足 D(v) ≤ L(w)
                if(D[v] < L[w])
                    edgeBridge.emplace_back(v,w);// 結 論 3 對於包含 r 在內
的所有點 v 和 v 在 T 中的子節點 w，邊 e(v, w) 在圖 G 中為bridge 的充要條件為
D(v) < L(w)。
                L[v] = min(L[v], L[w]);
            }
            L[v] = min(L[v], D[w]);
        }
        if(v == pa && Childcount < 2)
            isAP = false;
        if(isAP)
            APnode.emplace_back(v);
    }
};
```

Tarjan_for_BridgeCC

```
int timestamp = 1;
int bccid = 1;
int D[MAX_N];
int L[MAX_N];
int bcc[MAX_N];
stack<int> st;
vector<int> adj[MAX_N];
bool inSt[MAX_N];
void DFS(int v, int fa) { //call DFS(v,v) at first
    D[v] = L[v] = timestamp++; //timestamp > 0
    st.emplace(v);

    for (int w:adj[v]) {
        if( w==fa ) continue;
        if ( !D[w] ) { // D[w] = 0 if not visited
            DFS(w,v);
            L[v] = min(L[v], L[w]);
        }
        L[v] = min(L[v], D[w]);
    }
    if (L[v]==D[v]) {
        bccid++;
        int x;
        do {
            x = st.top(); st.pop();
            bcc[x] = bccid;
        } while (x!=v);
    }
    return ;
}
```

FLOW

Ford_Fulkerson

```
// O((V+E)F)
#define maxn 101
// remember to change used into the maxNode size -- kattis elementary math
bool used[maxn];
int End;
vector<int> V[maxn];
vector<tuple<int, int>> E;

// x=>y 可以流 C
// if undirected or 2-direc edge, bakcward Capacity become C;
// Graph build by edge array
// 反向邊的編號只要把自己的編號 xor 1 就能取得
void add_edge(int x, int y,int c)
{
    V[x].emplace_back( E.size() );
    E.emplace_back(y,c);
    V[y].emplace_back( E.size() );
    E.emplace_back(x,0);
}
int dfs(int v, int f)
{
    if( v==End ) return f;
    used[v] = true;
    int e,w;
    for( int eid : V[v] )
    {
        tie(e,w) = E[eid];
        if( used[e] || w==0 ) continue;

        w = dfs(e, min(w,f));
        if( w>0 )
        {
            // 更新流量
            get<1>(E[eid  ]) -= w;
            get<1>(E[eid^1]) += w;
            return w;
        }
    }
    return 0;// Fail!
}
int ffa(int s,int e)
{
    int ans = 0, f;
    End = e;
    while(true)
    {
        memset(used, 0, sizeof(used));
        f = dfs(s, INT_MAX);
        if( f<=0 ) break;
        ans += f;
    }
    return ans;
}
```

Dinic’s Algorithm

```
// O(V^2E) O(VE) finding argument path
// if unit capacity networe then O(min(V^3/2, E^1/2) E)
// solving bipartite matching O(E V^1/2)
#define maxn 101
#define INT_MAX 10000000
int End, dist[maxn];
vector<tuple<int, int, int>> V[maxn];
// 1st for node-index, 2nd for cap, 3nd for the index of the edge and the
vector[u];
void addEdge(int u, int v, int c){
    V[u].emplace_back(v, c, V[v].size());
    V[v].emplace_back(u, 0, V[u].size() - 1);
}
bool bfs(int s) {
    memset(dist, -1, sizeof(dist));
    queue<int> qu;
    qu.emplace(s);
    dist[s]=0;

    while( !qu.empty() ) {
        int S = qu.front(); qu.pop();
        for(auto &p : V[S]) {
            int E, C;
            tie(E, C, ignore) = p;
            if( dist[E]==-1 && C!=0 ) {
                dist[E]=dist[S]+1;
                qu.emplace(E);
            }
        }
    }
    return dist[End] != -1;
}
int dfs(int v, int f) {
    int e,w,rev;
    if( v==End || f==0 ) return f;
    for( auto &t : V[v] )
    {
        tie(e,w,rev) = t;
        if( dist[e]!=dist[v]+1 || w==0 )
            continue;

        w = dfs(e, min(w,f));
        if( w>0 )
        {
            get<1>(t) -= w;
            get<1>(V[e][rev]) += w;
            return w;
        }
    }
    dist[v] = -1; //優化，這個點沒用了
    return 0;// Fail!
}
int dinic(int s,int e)
{
    int ans = 0, f;
    End = e;
    while(bfs(s))
    {
        while( f = dfs(s, INT_MAX) )
            ans += f;
    }
    return ans;
}
```

MinCost MaxFlow // by jinkela

Bipartite Matching
konig’ algorithm

```
#include <vector>
#include <cstring>
using namespace std;

vector<int> V[205];
// V[i]記錄了左半邊可以配到右邊的那些點
int match[205]; // A<=B
// match[i] 記錄了右半邊配對到左半邊的哪個點
bool used[205];
int n;
bool dfs(int v)
{
    for(int e:V[v])
    {
        if( used[e] ) continue;
        used[e] = true;
        if( match[e] == -1 || dfs( match[e] ) )
        {
            match[e] = v;
            return true;
        }
    }
    return false;
}
int konig()
{
    memset(match,-1,sizeof(match));
```

```cpp
    int ans=0;

    for(int i=1;i<=n;++i)
    {
        memset(used, 0, sizeof(used));
        if( dfs(i) )
            ans++;
    }

    return ans;
}
```

KM algorithm

```
// Max weight perfect bipartite matching // O(V^3) // by jinkela
```

Graph Matching(untest!!) blossom algorithm

```
// by jinkela // 最大圖匹配//  O(V²(V+E))
```

最近點對

```cpp
template <typename T>
T ClosestPairSquareDistance(typename vector<Point<T>>::iterator l,
                            typename vector<Point<T>>::iterator r)
{
    auto delta = numeric_limits<T>::max();
    if (r - l > 1)
    {
        auto m = l + (r - l >> 1);
        nth_element(l, m, r); // Lexicographical order in default
        auto x = m->x;
        delta = min(ClosestPairSquareDistance<T>(l, m),
                    ClosestPairSquareDistance<T>(m, r));
        auto square = [&](T y) { return y * y; };
        auto sgn = [=](T a, T b) {
            return square(a - b) <= delta ? 0 : a < b ? -1 : 1;
        };
        vector<Point<T>> x_near[2];
        copy_if(l, m, back_inserter(x_near[0]), [=](Point<T> a) {
            return sgn(a.x, x) == 0;
        });
        copy_if(m, r, back_inserter(x_near[1]), [=](Point<T> a) {
            return sgn(a.x, x) == 0;
        });
        for (int i = 0, j = 0; i < x_near[0].size(); ++i)
        {
            while (j < x_near[1].size() and
                   sgn(x_near[1][j].y, x_near[0][i].y) == -1)
                ++j;
            for (int k = j; k < x_near[1].size() and
                            sgn(x_near[1][k].y, x_near[0][i].y) == 0;
                 ++k)
            {
                delta = min(delta, (x_near[0][i] - x_near[1][k]).norm());
            }
        }
        inplace_merge(l, m, r, [](Point<T> a, Point<T> b) {
            return a.y < b.y;
        });
    }
    return delta;
}
```

k-d-tree

```cpp
template <typename T>
class KDTree
{
    struct KDNode
    {
        Point<T> v;
        KDNode *ch[2];
        KDNode(const Point<T> &v, KDNode *l, KDNode *r) : v{v}, ch{l, r}
{}
        ~KDNode()
        {
            for (size_t i : {0, 1})
                if (ch[i])
                    ch[i]->~KDNode();
        }
        T dfs(const Point<T> q,

              T dis = numeric_limits<T>::max(),
              bool parity = 0)
        {
            dis = min(dis, (v - q).norm());
            bool isRight = parity ? v.x < q.x : v.y < q.y;
            if (ch[isRight])
                dis = min(dis, ch[isRight]->dfs(q, dis, parity ^ 1));
            if (ch[isRight ^ 1] and [](T x) {
                    return x * x;
                }(parity ? v.x - q.x : v.y - q.y) < dis)
                dis = min(dis, ch[isRight ^ 1]->dfs(q, dis, parity ^ 1));
            return dis;
        }
    } * root;
    KDNode *buildKDTree(typename vector<Point<T>>::iterator l,
                        typename vector<Point<T>>::iterator r,
                        bool parity = 0)
    {
        if (r == l)
            return nullptr;
        auto m = l + (r - l >> 1);
        nth_element(l, m, r, [=](Point<T> a, Point<T> b) {
            return parity ? a.x < b.x : a.y < b.y;
        });
        return new KDNode(*m,
                          buildKDTree(l, m, parity ^ 1),
                          buildKDTree(m + 1, r, parity ^ 1));
    }

public:
    KDTree(vector<Point<T>> A) : root{buildKDTree(A.begin(), A.end())} {}
    T nearestNeighborSquareDistance(Point<T> q)
    {
        return root->dfs(q);
    }
};
```

nCr

```cpp
using i64 = unsigned long long;
#define maxn 300005
i64 fact[maxn], tcaf[maxn];

#define P 998244353
#define REP1(i, n) for (int i = 1; i <= (int)(n); ++i)
#define REP(i, n) for (int i = (int)(n) - 1; i >= 0; --i)
void init(int n){
    fact[0] = 1;
    for (int i = 1; i <= n; i++)
        fact[i] = i * fact[i - 1] % P;
    for (int i = n; i >= 0; --i)
        tcaf[i] = deg(fact[i], -1);

}

i64 deg(i64 x, i64 d) {
    if (d < 0) d += P - 1;
    i64 y = 1;
    while (d) {
        if (d & 1) (y *= x) %= P;
        d /= 2;
        (x *= x) %= P;
    }
    return y;
}

i64 cnk(int n, int k) {
    if (k < 0 || k > n) return 0;
    return fact[n] * tcaf[k] % P * tcaf[n - k] % P;
}
```

some theorem

Maximum Independent Set
General: [NPC] maximum clique of complement of G
Tree: [P] Greedy
Bipartite Graph: [P] Maximum Cardinality Bipartite Matching

Minimum Dominating Set
General: [NPC]
Tree: [P] DP
Bipartite Graph: [NPC]

Minimum Vertex Cover
General: [NPC] V - MIS
Tree: [P] Greedy, from leaf to root
Bipartite Graph: [P] Maximum Cardinality Bipartite Matching

Minimum Edge Cover
General: [P] V - Maximum Matching
Bipartite Graph: [P] Greedy, strategy: cover small degree node first.
(Min/Max)Weighted: [P]: Minimum/Minimum Weight Matching

Pick's Theorem A = i + b/2 - 1

我们称Y中所有边的两个端点为被Y所匹配。我们遍历所有R中没有被Y匹配的顶点，寻找所有长度为偶数的路径，路径中匹配边和未匹配边交替出现。事实上不存在长度为奇数的路径，不然我们就找到了一个增广路径。我们将L中被标记过的顶点和R中未被标记的顶点合成一个新的点集合X，我们接下来证明X是最小顶点覆盖。

9.8 Formulas or Theorems

We have encountered some rarely used formulas or theorems in some programming contest problems before. Knowing them will give you an *unfair advantage* over other contestants if one of these rare formulas or theorems is used in the programming contest that you join.

1. Cayley's Formula: There are n^{n-2} spanning trees of a complete graph with n labeled vertices. Example: UVa 10843 - Anne's game.
2. Derangement: A permutation of the elements of a set such that none of the elements appear in their original position. The number of derangements $der(n)$ can be computed as follow: $der(n) = (n-1) \times (der(n-1) + der(n-2))$ where $der(0) = 1$ and $der(1) = 0$. A basic problem involving derangement is UVa 12024 - Hats (see Section 5.6).
3. Erdős Gallai's Theorem gives a necessary and sufficient condition for a finite sequence of natural numbers to be the *degree sequence* of a simple graph. A sequence of non-negative integers $d_1 \geq d_2 \geq \ldots \geq d_n$ can be the degree sequence of a simple graph on n vertices iff $\sum_{i=1}^{n} d_i$ is even and $\sum_{i=1}^{k} d_i \leq k \times (k-1) + \sum_{i=k+1}^{n} min(d_i, k)$ holds for $1 \leq k \leq n$. Example: UVa 10720 - Graph Construction.
4. Euler's Formula for Planar Graph[6]: $V - E + F = 2$, where F is the number of faces[7] of the Planar Graph. Example: UVa 10178 - Count the Faces.
5. Moser's Circle: Determine the number of pieces into which a circle is divided if n points on its circumference are joined by chords with no three internally concurrent. Solution: $g(n) = ^nC_4 + ^nC_2 + 1$. Example: UVa 10213 - How Many Pieces of Land?
6. Pick's Theorem[8]: Let I be the number of integer points in the polygon, A be the area of the polygon, and b be the number of integer points on the boundary, then $A = i + \frac{b}{2} - 1$. Example: UVa 10088 - Trees on My Island.
7. The number of spanning tree of a complete bipartite graph $K_{n,m}$ is $m^{n-1} \times n^{m-1}$. Example: UVa 11719 - Gridlands Airport.