Contents

1 Basic

1.1 .vimrc

linenumber, relative-linenumber, mouse, cindent, expandtab, shiftwidth, softtabstop, nowrap, ignorecase
nornu when enter insert mode

```
se nu rnu mouse=a cin et sw=2 sts=2 nowrap ic
au InsertLeave * se rnu
au InsertEnter * se nornu
```

1.2 default code

```
#include<bits/stdc++.h>
#define ll long long
#define ld long double
#define INF 0x3f3f3f3f
#define LLINF 0x3f3f3f3f3f3f3f3f
#define X first
#define Y second
#define PB emplace_back
using namespace std;
const int MXN = 4e5+5;

void sol(){}
int main(){
  int t=1;
  cin >> t;
  while(t--){
    sol(); } }
```

2 flow

2.1 MinCostFlow

2.2 Dinic

2.3 Hungarian

2.4 Kuhn Munkres 最大完美二分匹配

3 Math

3.1 Fast Pow & Inverse & Combination

$fpow(a, b, m) = a^b \pmod m$
$fa[i] = i! \pmod{MOD}$
$fi[i] = i!^{-1} \equiv 1 \pmod{MOD}$
$c(a, b) = \binom{a}{b} \pmod{MOD}$

```
ll fpow(ll a, ll b, ll m){
  ll ret = 1;
  a %= m;
  while(b){
    if(b&1) ret = ret * a % m;
    a = a * a % m;
    b >>= 1; }
  return ret; }

ll fa[MXN], fi[MXN];
void init(){
  fa[0] = 1;
  for(ll i = 1; i < MXN; ++i)
    fa[i] = fa[i - 1] * i % MOD;
  fi[MXN - 1] = fpow(fa[MXN - 1], MOD - 2, MOD);
  for(ll i = MXN - 1; i > 0; --i)
    fi[i - 1] = fi[i] * i % MOD; }

ll c(ll a, ll b){
  return fa[a] * fi[b] % MOD * fi[a - b] % MOD; }
```

4 Geometry

5 Tree

5.1 LCA

LCA(n) ⇒ n: 點數
lca.addEdge(u, v);
lca.build(root, root);
lca.anc[i][j] ⇒ i 的第 2^j 個祖先
lca.query(x, y) ⇒ x,y 的 LCA
lca.qdis(x, y) ⇒ x,y 的最短距離 (可以用倍增法帶權)
編譯指令: g++ -Wl,-stack,500000000 test.cpp a

```
const int MXN = 5e5+5;
struct LCA{
  int n, lgn, ti = 0;
  int anc[MXN][24], in[MXN], out[MXN];
  vector<int> g[MXN];
  LCA(int n) : n(n + 1), lgn(__lg(n) + 5){}
  void addEdge(int x, int y){ g[x].PB(y), g[y].PB(x); }
  void build(int x, int fx){
    in[x] = ti++;
    int cur = fx;
    for(int i = 0; i < lgn; ++i)
      anc[x][i] = cur, cur = anc[cur][i];
    for(auto i : g[x]) if(i != fx) build(i, x);
    out[x] = ti++; }
  bool isanc(int a, int x){
    return in[a] <= in[x] && out[a] >= out[x]; }
  int query(int x, int y){
    if(isanc(x, y)) return x;
    if(isanc(y, x)) return y;
    for(int i = lgn-1; i >= 0; --i)
      if(!isanc(anc[x][i], y)) x = anc[x][i];
    return anc[x][0]; }
  int qdis(int x, int y){
    int dis = !isanc(x, y) + !isanc(y, x);
    for(int i = lgn - 1; i >= 0; --i){
      if(!isanc(anc[x][i], y))
        x = anc[x][i], dis += 1<<i;
      if(!isanc(anc[y][i], x))
        y = anc[y][i], dis += 1<<i; }
    return dis; } };
```

6 Graph

7 String

8 Data Structure

8.1 Treap

Treap *th = nullptr
th = merge(th, new Treap(val)) ⇒ 新增元素到 th
th = merge(merge(tl, tm), tr) ⇒ 合併 tl,tm,tr 到 th
split(th, k, tl, tr) ⇒ 分割 th, tl 的元素 $\le$ k (失去 BST 性質後不能用)
kth(th, k, tl, tr) ⇒ 分割 th, gsz(tl) $\le$ k (< when gsz(th) < k)
gsz ⇒ get size | gsum ⇒ get sum | th->rev ^= 1 ⇒ 反轉 th
帶懶標版本, 並示範 sum/rev 如何 pull/push
注意 Treap 複雜度好但常數大, 動作能用其他方法就用, 並做 io 等優化

```
struct Treap{
  Treap *l, *r;
  int pri, sz, rev;
  ll val, sum;
  Treap(int _val): l(nullptr), r(nullptr),
    pri(rand()), sz(1), rev(0),
    val(_val), sum(_val){} };

ll gsz(Treap *x){ return x ? x->sz : 0; }
ll gsum(Treap *x){ return x ? x->sum : 0; }

Treap* pull(Treap *x){
  x->sz = gsz(x->l) + gsz(x->r) + 1;
  x->sum = x->val + gsum(x->l) + gsum(x->r);
  return x; }
void push(Treap *x){
  if(x->rev){
    swap(x->l, x->r);
    if(x->l) x->l->rev ^= 1;
    if(x->r) x->r->rev ^= 1;
    x->rev = 0; } }

Treap* merge(Treap* a, Treap* b){
  if(!a || !b) return a ? a : b;
  push(a), push(b);
  if(a->pri > b->pri){
    a->r = merge(a->r, b);
    return pull(a); }
  else{
    b->l = merge(a, b->l);
    return pull(b); } }

void split(Treap *x, int k, Treap *&a, Treap *&b){
  if(!x) a = b = nullptr;
  else{
    push(x);
```

```
    if(x->val <= k) a = x, split(x->r, k, a->r, b);
    else            b = x, split(x->l, k, a, b->l);
    pull(x); } }

void kth(Treap *x, int k, Treap *&a, Treap *&b){
  if(!x) a = b = nullptr;
  else{
    push(x);
    if(gsz(x->l) < k)
         a = x, kth(x->r, k - gsz(x->l) - 1, a->r, b);
    else b = x, kth(x->l, k, a, b->l);
    pull(x); } }
```

8.2 BIT

支援單點加值、前綴求和、第 k 小
not finished, not verified

```
struct BIT{
  int sz;
  vector<int> dat;
  void init(int _sz){
    sz = _sz;
    dat.assign(sz + 1, 0);
  }
  void upd(int id, int x){
    for(int i = id; i <= sz; i += i & -i)
      dat[i] += x;
  }
  int sum(int id){
    int res = 0;
    for(int i = id; i > 0; i -= i & -i)
      res += dat[i];
    return res;
  }
  int kth(int k){
    int res = 0;
    for(int i = 1 << __lg(sz); i > 0; i >>= 1)
      if(res + i <= sz && dat[res + i] < k)
        k -= dat[res += i];
    return res;
  }
};
```

9 Others