



University of Washington

# Evirir's templates

Jun Xing Go

2024-02-23

- 1 Contest
- 2 Data structures
- 3 Number theory
- 4 Graph
- 5 Strings
- 6 Various

# Contest (1)

template.cpp41 lines

```
#include <bits/stdc++.h>
// #include <ext/pb_ds/assoc_container.hpp>
// #include <ext/pb_ds/tree_policy.hpp>
using namespace std;
// using namespace __gnu_pbds;

#pragma GCC optimize("O3,unroll-loops")
#pragma GCC target("avx2")
#define watch(x) cout<<(#x)<<"="<<(x)<<'\\n'
#define mset(d,val) memset(d,val,sizeof(d))
#define cbug if(DEBUG) cout
#define setp(x) cout<<fixed<<setprecision(x)
#define sz(x) (int)(x).size()
#define all(x) begin(x), end(x)
#define forn(i,a,b) for(int i=(a);i<(b);i++)
#define fore(i,a,b) for(int i=(a);i<=(b);i++)
#define pb push_back
#define F first
#define S second
#define fbo find_by_order
#define ook order_of_key
typedef long long ll;
typedef long double ld;
typedef pair<ll,ll> ii;
typedef vector<ll> vi;
typedef vector<ii> vii;
// template<typename T>
// using pbds = tree<T, null_type, less<T>, rb_tree_tag,
//     tree_order_statistics_node_update>;
void SD(int t=0){ cout<<"PASSED "<<t<<endl; }
ostream& operator<<(ostream &out, ii x){ out<<("(<<x.F<<","<<x.
    S<<")"); return out; }
const ll INF = 1e18;
const int MOD = 998244353;

const bool DEBUG = 0;
const int MAXN = 100005;
const int LG = 21;
```

```
int main() {
    cin.tie(0)->sync_with_stdio(0);
    cin.exceptions(cin.failbit);
}
```

.bashrc3 lines

```
alias c='g++ -Wall -Wconversion -Wfatal-errors -g -std=c++14 \
    -fsanitize=undefined,address'
xmodmap -e 'clear lock' -e 'keycode 66=less greater' #caps =>
```

.vimrc6 lines

```
set cin aw ai is ts=4 sw=4 tm=50 nu noeb bg=dark ru cul
sy on | im jk <esc> | im kj <esc> | no ; :
" Select region and then type :Hash to hash your selection.
" Useful for verifying that there aren't mistypes.
ca Hash w !cpp -dD -P -fpreprocessed \\| tr -d '[:space:]' \
    \\ md5sum \\| cut -c6
```

hash.sh3 lines

```
# Hashes a file, ignoring all whitespace and comments. Use for
# verifying that code was correctly typed.
cpp -dD -P -fpreprocessed | tr -d '[:space:]'| md5sum |cut -c-6
```

.vscode-tasks.json53 lines

```
{
  "version": "2.0.0",
  "tasks": [
    {
      "type": "cppbuild",
      "label": "C/C++: MSYS g++.exe build active file",
      "command": "C:/msys64/ucrt64/bin/g++.exe",
      "args": [
        "-std=c++20",
        "-Wall",
        "-Wextra",
        "-WL,--stack,268435456",
        "-IC:/ac-library",
        "-fdiagnostics-color=always",
        "-g",
        "${file}",
        "-o",
        "${fileDirname}\\${fileBasenameNoExtension}.exe"
      ],
      "options": {
        "cwd": "C:/msys64/ucrt64/bin"
      },
      "problemMatcher": [
        "$gcc"
      ],
      "group": {
        "kind": "build",
        "isDefault": true
      },
      "detail": "compiler: C:/msys64/ucrt64/bin/g++.exe"
    },
    {
      "type": "shell",
      "label": "C/C++: cl.exe build active file",
      "command": "cl.exe",
      "args": [
        "/Zi",
        "/EHsc",
        "/Fe:",
        "${fileDirname}\\${fileBasenameNoExtension}.exe",
        "${file}"
      ],
      "options": {
        "cwd": "${fileDirname}"
      },
      "problemMatcher": [
        "$msCompile"
      ],
      "group": "build",
      "detail": "compiler: cl.exe"
    }
  ]
}
```

# Schwartz-Zippel lemma

Let  $P \in F[x_1, x_2, \dots, x_n]$  be a non-zero polynomial of degree  $d \geq 0$  over a field  $F$ . Let  $S$  be a finite subset of  $F$  and let  $r_1, r_2, \dots, r_n$  be uniformly independently randomly selected from  $S$ . Then  $Pr[P(r_1, r_2, \dots, r_n) = 0] \leq \frac{d}{|S|}$ .

# Data structures (2)

LazyRecursiveSegmentTree.h

Description: Segment tree with lazy propagation.

Memory:  $\mathcal{O}(\log N)$  per update/query.

53b213, 75 lines

```
class LazySegmentTree {
private:
    int size_;
    vector<ll> v, lazy;

    void update(int s, int e, ll val, int k, int l, int r)
    {
        push(k, l, r);
        if (r < s || e < l)
            return;
        if (s <= l && r <= e)
        {
            lazy[k] = val;
            push(k, l, r);
        }
        else
        {
            update(s, e, val, k * 2, l, (l + r) >> 1);
            update(s, e, val, k * 2 + 1, ((l + r) >> 1) + 1, r);
            ;
            v[k] = merge(v[k * 2], v[k * 2 + 1]);
        }
    }

    ll query(int s, int e, int k, int l, int r)
    {
        push(k, l, r);
        if (r < s || e < l)
            return 0; // dummy value
        if (s <= l && r <= e)
            return v[k];
        ll lc = query(s, e, k * 2, l, (l + r) >> 1);
        ll rc = query(s, e, k * 2 + 1, ((l + r) >> 1) + 1, r);
        return merge(lc, rc);
    }

public:
    LazySegmentTree() : v(vector<ll>()), lazy(vector<ll>()) {}
    LazySegmentTree(int n)
    {
        for (size_ = 1; size_ < n;)
            size_ <= 1;
        v.resize(size_ * 4);
        lazy.resize(size_ * 4);
    }
    void reset()
    {
        v.assign(size_ * 4, 0);
        lazy.assign(size_ * 4, 0);
    }
    inline void push(int k, int l, int r)
    {
        if (lazy[k] != 0)
        {
```

```
v[k] += (r - l + 1) * lazy[k]; // remember to
    consider the range!
if (l != r)
{
    lazy[k * 2] += lazy[k];
    lazy[k * 2 + 1] += lazy[k];
}
lazy[k] = 0;
}
}
inline ll merge(ll x, ll y)
{
    return x + y;
}
inline void update(int l, int r, ll val)
{
    update(l, r, val, 1, 0, size_ - 1);
}
inline ll query(int l, int r)
{
    return query(l, r, 1, 0, size_ - 1);
}
};
```

PersistentSegmentTree.h

Description: rawr o.=.o cd45ec, 58 lines

```
inline ll merge(ll x, ll y){
    return x+y;
}
struct Node {
    Node *l, *r;
    ll sum=0;
    Node(ll val): l(nullptr), r(nullptr), sum(val) {}
    Node(Node *l, Node *r): l(l), r(r), sum(0) {
        if(l) sum=merge(sum, l->sum);
        if(r) sum=merge(sum, r->sum);
    }
};
class PersistSegmentTree {
private:
    int size_;
    Node* build(int l, int r)
    {
        if(l==r) return new Node(0);
        int mid=(l+r)>>1;
        return new Node(build(l, mid), build(mid+1, r));
    }
    Node* build(ll a[], int l, int r)
    {
        if(l==r) return new Node(a[l]);
        int mid=(l+r)>>1;
        return new Node(build(a, l, mid), build(a, mid+1, r));
    }
    Node* update(Node* k, int p, ll val, int l, int r)
    {
        if(l==r) return new Node(k->sum + val); //modification
        int mid=(l+r)>>1;
        if(p<=mid) return new Node(update(k->l, p, val, l, mid), k
            ->r);
        return new Node(k->l, update(k->r, p, val, mid+1, r));
    }
    ll query(Node* k, int s, int e, int l, int r)
    {
        if(r<s || e<l) return 0; //dummy value
        if(s<=l && r<=e) return k->sum;
        int mid=(l+r)>>1;
        return merge(query(k->l, s, e, l, mid), query(k->r, s, e,
            mid+1, r));
    }
};
```

```
}
public:
    PersistSegmentTree(): size_(0) {}
    PersistSegmentTree(int n): size_(n) {}
    inline Node* build(){
        return build(0, size_-1);
    }
    inline Node* build(ll a[]){
        return build(a, 0, size_-1);
    }
    inline Node* update(Node* k, int p, ll val){
        return update(k, p, val, 0, size_-1);
    }
    inline ll query(Node* k, int l, int r){
        return query(k, l, r, 0, size_-1);
    }
};
```

SegmentTree2D.h

Description: rawr o.=.o 740da4, 93 lines

```
class SegmentTree2D {
private:
    int size_n, size_m;
    vector<vector<ll>> v;
    void build(const vector<vector<ll>> &a, int k, int l, int r)
    {
        if(r >= size_n) return;
        if(l != r){
            int mid = (l+r)>>1;
            build(a, k*2, l, mid);
            build(a, k*2+1, mid+1, r);
        }
        build2(a, k, l, r, 1, 0, size_m-1);
    }
    void build2(const vector<vector<ll>> &a, int k, int l, int r,
        int k2, int l2, int r2)
    {
        if(l2 == r2){
            if(l >= a.size() || l2 >= a[0].size()) return;
            if(l == r)
                v[k][k2] = a[l][l2];
            else
                v[k][k2] = merge(v[k*2][k2], v[k*2+1][k2]);
            return;
        }
        int mid2 = (l2+r2)>>1;
        build2(a, k, l, r, k*2, l2, mid2);
        build2(a, k, l, r, k*2+1, mid2+1, r2);
        v[k][k2] = merge(v[k][k*2], v[k][k*2+1]);
    }
    void update(int p1, int p2, ll val, int k, int l, int r)
    {
        if(p1 < l || r < p1) return;
        if(l != r){
            int mid = (l+r)>>1;
            update(p1, p2, val, k*2, l, mid);
            update(p1, p2, val, k*2+1, mid+1, r);
        }
        update2(p1, p2, val, k, l, r, 1, 0, size_m-1);
    }
    void update2(int p1, int p2, ll val, int k, int l, int r, int
        k2, int l2, int r2)
    {
        if(p2 < l2 || r2 < p2) return;
        if(l2 == r2){
            if(l == r)
                v[k][k2] ^= val; //modification
        }
    }
};
```

```
else
    v[k][k2] = merge(v[k*2][k2], v[k*2+1][k2]);
return;
}
int mid2 = (l2+r2)>>1;
update2(p1, p2, val, k, l, r, k*2, l2, mid2);
update2(p1, p2, val, k, l, r, k*2+1, mid2+1, r2);
v[k][k2] = merge(v[k][k*2], v[k][k*2+1]);
}
ll query(int s, int e, int s2, int e2, int k, int l, int r)
{
    if(e < l || r < s) return 0; //dummy value
    if(s <= l && r <= e) return query2(s2, e2, k, l, 0, size_m
        -1);
    int mid = (l+r)>>1;
    ll lc = query(s, e, s2, e2, k*2, l, mid);
    ll rc = query(s, e, s2, e2, k*2+1, mid+1, r);
    return merge(lc, rc);
}
ll query2(int s2, int e2, int k, int k2, int l2, int r2)
{
    if(e2 < l2 || r2 < s2) return 0; //dummy value
    if(s2 <= l2 && r2 <= e2) return v[k][k2];
    int mid2 = (l2+r2)>>1;
    ll lc = query2(s2, e2, k, k*2, l2, mid2);
    ll rc = query2(s2, e2, k, k*2+1, mid2+1, r2);
    return merge(lc, rc);
}
};
```

```
public:
    SegmentTree2D(): v(vector<vector<ll>>()) {}
    SegmentTree2D(int n, int m){
        for(size_n=1;size_n<n;) size_n<=1;
        for(size_m=1;size_m<m;) size_m<=1;
        v.resize(4*size_n, vector<ll>(4*size_m));
    }
    inline ll merge(ll x, ll y){
        return x+y;
    }
    inline void build(const vector<vector<ll>> &a){
        build(a, 1, 0, size_n-1);
    }
    inline void update(int p1, int p2, ll val){
        update(p1, p2, val, 1, 0, size_n-1);
    }
    inline ll query(int l, int r, int l2, int r2){
        return query(l, r, l2, r2, 1, 0, size_n-1);
    }
};
```

LiChaoTree.h

Description: rawr o.=.o 2c2be7, 42 lines

```
struct Line {
    ll m,c;
    Line(): m(0), c(INF) {}
    Line(ll m, ll c): m(m), c(c) {}
    ll eval(ll x){ return m*x+c; }
};

struct LiChaoTree {
    int sz;
    bool isMax; // whether this maintains max
    vector<Line> v;
    LiChaoTree(): sz(0), isMax(false), v(vector<Line>()) {}
    LiChaoTree(int sz, bool isMax): sz(sz), isMax(isMax) {
        v.resize(sz*4, {0,INF});
    }
    void addline(Line& val) {
```

```
    if(isMax) {
        val.m = -val.m;
        val.c = -val.c;
    }
    addline(val, 1, 0, sz-1);
}
ll query(int x) {
    return (isMax ? -1 : 1) * query(x, 1, 0, sz-1);
}
void addline(Line& val, int k, int l, int r) {
    int mid = (l+r)>>1;
    bool lc = val.eval(l) <= v[k].eval(l);
    bool mc = val.eval(mid) <= v[k].eval(mid);
    if(mc) swap(val, v[k]);
    if(l==r) return;
    if(lc==mc) addline(val, k*2, l, mid);
    else addline(val, k*2+1, mid+1, r);
}
ll query(int x, int k, int l, int r) {
    ll cur = v[k].eval(x);
    if(l==r) return cur;
    int mid=(l+r)>>1;
    if(x<=mid) return min(cur, query(x, k*2, l, mid));
    return min(cur, query(x, k*2+1, mid+1, r));
}
};
```

## Number theory (3)

Mod.h	3e8014, 72 lines
-------	------------------

```
vector<ll> fact,ifact,inv,pow2;
ll add(ll a, ll b, ll m = MOD)
{
    a+=b;
    if(abs(a)>=m) a%=m;
    if(a<0) a+=m;
    return a;
}
ll mult(ll a, ll b, ll m = MOD)
{
    if(abs(a)>=m) a%=m;
    if(abs(b)>=m) b%=m;
    a*=b;
    if(abs(a)>=m) a%=m;
    if(a<0) a+=m;
    return a;
}
void radd(ll &a, ll b, ll m = MOD){ a=add(a,b,m); }
void rmult(ll &a, ll b, ll m = MOD){ a=mult(a,b,m); }
ll pw(ll a, ll b, ll m = MOD)
{
    assert(b >= 0); // can return 0 if desired
    if(abs(a)>=m) a%=m;
    if(a==0 && b==0) return 0; // value of 0^0
    ll r=1;
    while(b){
        if(b&1) r=mult(r,a,m);
        a=mult(a,a,m);
        b>>=1;
    }
    return r;
}
ll inverse(ll a, ll m = MOD)
{
    return pw(a,m-2);
}
```

```
ll choose(ll a, ll b)
{
    if(a<b) return 0;
    if(b==0) return 1;
    if(a==b) return 1;
    return mult(fact[a],mult(ifact[b],ifact[a-b]));
}
void init(ll _n)
{
    fact.clear(); ifact.clear(); inv.clear(); pow2.clear();
    fact.resize(_n+1); ifact.resize(_n+1); inv.resize(_n+1); pow2
        .resize(_n+1);
    pow2[0]=1; ifact[0]=1; fact[0]=1;
    for(int i=1;i<=_n;i++){
        pow2[i]=add(pow2[i-1],pow2[i-1]);
        fact[i]=mult(fact[i-1],i);
    }
    ifact[_n] = inverse(fact[_n]);
    for(int i=_n-1;i>=1;i--){
        ifact[i] = mult(ifact[i+1], i+1);
    }
    for(int i=1;i<=_n;i++){
        inv[i] = mult(fact[i-1], ifact[i]);
    }
}
// partition n into k blocks of size >= 0
ll nonneg_partition(ll n, ll k)
{
    assert(k >= 1); // can return 0 if desired
    return choose(n + k - 1, k - 1);
}
// partition n into k blocks of size >= minVal
ll partition(ll n, ll k, ll minVal = 1)
{
    assert(k >= 1); // can return 0 if desired
    return nonneg_partition(n - k * minVal, k);
}

NumberTheory.h
Description: rawr o.=o
6dea17, 159 lines
vector<ll> primes, totient, sumdiv, bigdiv, lowprime, mobius;
vector<bool> isprime;
void Sieve(ll n) // linear Sieve
{
    isprime.assign(n+1, 1);
    lowprime.assign(n+1, 0);
    isprime[1] = false;
    for(ll i = 2; i <= n; i++)
    {
        if(lowprime[i] == 0)
        {
            primes.pb(i);
            lowprime[i] = i;
        }
        for(int j=0; j<sz(primes) && primes[j]<=lowprime[i] && i*
            primes[j]<=n; j++)
        {
            isprime[i*primes[j]] = false;
            lowprime[i*primes[j]] = primes[j];
        }
    }
}
void SieveMobius(ll n)
{
    mobius.resize(n + 1);
    mobius[1] = 1;
    for (ll i = 2; i <= n; i++)
    {
```

```
        if (lowprime[i] == i) mobius[i] = -1;
        for (int j = 0; j < sz(primes) && primes[j] <= lowprime[i]
            && i * primes[j] <= n; j++)
        {
            ll cur = i * primes[j];
            if (primes[j] == lowprime[i]) mobius[cur] = 0;
            else mobius[cur] = -mobius[i];
        }
    }
}
ll phi(ll x)
{
    map<ll,ll> pf;
    ll num = 1; ll num2 = x;
    for(ll i = 0; primes[i]*primes[i] <= x; i++)
    {
        if(x%primes[i]==0)
        {
            num2/=primes[i];
            num*=(primes[i]-1);
        }
        while(x%primes[i]==0)
        {
            x/=primes[i];
            pf[primes[i]]++;
        }
    }
    if(x>1)
    {
        pf[x]++; num2/=x; num*=(x-1);
    }
    x = 1;
    num*=num2;
    return num;
}
bool isprime(ll x)
{
    if(x==1) return false;
    for(ll i = 0; primes[i]*primes[i] <= x; i++)
    {
        if(x%primes[i]==0) return false;
    }
    return true;
}
void SievePhi(ll n)
{
    totient.resize(n+1);
    for (int i = 1; i <= n; ++i) totient[i] = i;
    for (int i = 2; i <= n; ++i)
    {
        if (totient[i] == i)
        {
            for (int j = i; j <= n; j += i)
            {
                totient[j] -= totient[j] / i;
            }
        }
    }
}
void SieveSumDiv(ll n)
{
    sumdiv.resize(n+1);
    for(int i = 1; i <= n; ++i)
    {
        for(int j = i; j <= n; j += i)
        {
            sumdiv[j] += i;
        }
    }
}
```

```

}
ll getPhi(ll n)
{
    return totient[n];
}
ll getSumDiv(ll n)
{
    return sumdiv[n];
}
ll pw(ll a, ll b, ll mod)
{
    ll r = 1;
    if(b < 0) b += mod*100000LL;
    while(b)
    {
        if(b&1) r = (r*a)%mod;
        a = (a*a)%mod;
        b>>=1;
    }
    return r;
}
ll inv(ll a, ll mod)
{
    return pw(a, mod - 2, mod);
}
ll invgeneral(ll a, ll mod)
{
    ll ph = phi(mod);
    ph--;
    return pw(a, ph, mod);
}
void getpf(vector<ii>& pf, ll n)
{
    for(ll i = 0; primes[i]*primes[i] <= n; i++)
    {
        int cnt = 0;
        while(n%primes[i]==0)
        {
            n/=primes[i]; cnt++;
        }
        if(cnt>0) pf.pb(ii(primes[i], cnt));
    }
    if(n>1)
    {
        pf.pb(ii(n, 1));
    }
}
void getdiv(vector<ll>& div, vector<ii>& pf, ll n = 1, int i = 0)
{
    if (pf.empty()) // divisors of 1
    {
        div = {1};
        return;
    }
    ll x, k;
    if(i >= sz(pf)) return;
    x = n;
    for(k = 0; k <= pf[i].S; k++)
    {
        if(i == sz(pf) - 1) div.pb(x);
        getdiv(div, pf, x, i + 1);
        x *= pf[i].F;
    }
}
}
```

Graph (4)

DSU.h

Description: rawr o.=.o639c76, 18 lines

```

struct DSU {
    struct Node{ int p, sz; };
    vector<Node> dsu; int cc;
    Node& operator[] (int id){ return dsu[rt(id)]; }
    DSU(int n){ dsu.resize(n);
        forn(i,0,n){ cc=n; dsu[i]={i,1}; }
    }
    inline int rt(int u){ return (dsu[u].p==u) ? u : dsu[u].p=rt(
        dsu[u].p); }
    inline bool sameSet(int u, int v){ return rt(u)==rt(v); }
    void merge(int u, int v){
        u = rt(u); v = rt(v);
        if(u == v) return;
        if(dsu[u].sz < dsu[v].sz) swap(u,v);
        dsu[v].p = u;
        dsu[u].sz += dsu[v].sz;
        cc--;
    }
};
```

Dijkstra.h

Description: rawr o.=.o90108e, 25 lines

```

vector<ii> adj[MAXN]; // (node, distance)
ll dist[MAXN];
// int parents[MAXN];

void dijkstra(int src)
{
    priority_queue<ii, vector<ii>, greater<ii>> q; // (distance,
        node)
    fill(dist, dist + n, INF);
    // fill(parents, parents + n, -1);
    dist[src] = 0;
    q.push({dist[src], src});
    while (!q.empty())
    {
        auto [cur_dist, u] = q.top();
        q.pop();
        if (cur_dist > dist[u]) continue;
        for (auto [v, w] : adj[u])
        {
            if (dist[v] <= cur_dist + w) continue;
            dist[v] = cur_dist + w;
            // parents[v] = u;
            q.push({dist[v], v});
        }
    }
}
```

CentroidDecomp.h

Description: rawr o.=.ob10504, 67 lines

```

int sz[MAXN];
bool vst[MAXN];
int cpnt[MAXN]; // centroid tree parent
vector<int> child[MAXN]; // subtree of centroid tree
mset(cpnt,-1);

void dfs_sz(int u, int p)
{
    sz[u]=1;
    for(int v: adj[u])
    {
        if(v==p || vst[v]) continue;
        dfs_sz(v,u);
        sz[u]+=sz[v];
    }
}
```

```

}
}
int centroid(int u, int p, int r)
{
    for(int v: adj[u])
    {
        if(v==p || vst[v]) continue;
        if(sz[v]*2>sz[r]) return centroid(v,u,r);
    }
    return u;
}
int build_tree(int u)
{
    dfs_sz(u,-1);
    u=centroid(u,-1,u);
    vst[u]=1;
    for(int v: adj[u])
    {
        if(vst[v]) continue;
        cpnt[build_tree(v)]=u;
    }
    return u;
}
void prep(int u, int p)
{
    for(int v: adj[u])
    {
        if(v==p || vst[v]) continue;

        prep(v, u);
    }
}
void solve(int u)
{
    dfs_sz(u,-1);
    u=centroid(u,-1,u);

    prep(u,-1);
    for(int v: adj[u])
    {
        if(vst[v]) continue;

    }

    // do stuffs

    vst[u]=1;
    for(int v: adj[u])
    {
        if(vst[v]) continue;
        solve(v);
    }
}
}
```

VirtualTree.h

Description: rawr o.=.o3a6025, 31 lines

```

int buildVirtualTree(vector<int> nodes, vi vadj[])
{
    // Change these as needed
    auto reset = [&](int u) {
        vadj[u].clear();
    };
    auto connect = [&](int u, int v) { // u is parent of v
        vadj[u].push_back(v);
    };

    auto cmpDfs = [&](int u, int v) {
        return in[u] < in[v];
    };
}
```

```
};
sort(nodes.begin(), nodes.end(), cmpDfs);
unordered_set<int> uniqueNodes(nodes.begin(), nodes.end());
for (int i{1}; i < sz(nodes); i++)
    uniqueNodes.insert(getLca(nodes[i - 1], nodes[i]));
nodes = vector<int>(uniqueNodes.begin(), uniqueNodes.end());
sort(nodes.begin(), nodes.end(), cmpDfs);
for_each(nodes.begin(), nodes.end(), reset);

stack<int> stk;
for (int u : nodes)
{
    if (stk.empty()) { stk.push(u); continue; }
    while (!isChild(stk.top(), u)) stk.pop();
    connect(stk.top(), u);
    stk.push(u);
}
return nodes[0];
}
```

Strings (5)

```
Trie.h
Description: rawr o.=.o
b33c93, 42 lines

struct TrieNode
{
    int next[26];
    bool leaf = false;
    TrieNode() { fill(begin(next), end(next), -1); }
};

struct Trie
{
    int siz;
    vector<TrieNode> tr;
    Trie() : siz(0), tr(vector<TrieNode>(1)) {}
    TrieNode &operator[](int u) { return tr[u]; }
    int size() { return siz; }
    void addstring(const string &s)
    {
        int v = 0;
        for (char ch : s)
        {
            int c = ch - 'a';
            if (tr[v].next[c] == -1)
            {
                tr[v].next[c] = tr.size();
                tr.emplace_back();
            }
            v = tr[v].next[c];
        }
        if (!tr[v].leaf)
            siz++;
        tr[v].leaf = true;
    }
};

template <class F>
void dfs(int u, F f)
{
    forn(i, 0, 26)
    {
        if (tr[u].next[i] != -1)
        {
            dfs(tr[u].next[i]);
        }
    }
}

};
```

Various (6)

```
Random.h
Description: rawr o.=.o
Usage:      rng() to generate random number, dis(rng) to use
distribution.
f0335a, 20 lines

mt19937 rng(chrono::steady_clock::now().time_since_epoch().
    count());
uniform_int_distribution<int>(1,6) (rng);
uniform_int_distribution<> dis(1,6);
mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().
    count());

// Custom hash for unordered_map: unordered_map<T,T,custom_hash
> mp;
struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        // http://xorshift.di.unimi.it/splitmix64.c
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM = chrono::
            steady_clock::now().time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }
};
```