Chulalongkorn University

’’ ’’’,\0\n

Teetat T., Thawin T., Thitrin S.

2025-07-17

Data structures (1)

HashMap.h

Description: Hash map with mostly the same API as unordered_map, but ~3x faster. Uses 1.5x memory. Initial capacity must be a power of 2 (if provided).

d77092, 7 lines

```
#include <bits/extc++.h>
// To use most bits rather than just the lowest ones:
struct chash { // large odd number for C
  const uint64_t C = ll(4e18 * acos(0)) | 71;
  ll operator()(ll x) const { return __builtin_bswap64(x*C); }
};
__gnu_pbds::gp_hash_table<ll,int,chash> h({},{},{},{},{1<<16});
```

LineContainer.hpp

Description: Line Container (Minimize).
Time: $\mathcal{O}(\log N)$

"src/contest/template.hpp" 24a5c4, 37 lines

```
struct Line{
  static bool querymode;
    ll m,c;
    mutable ll p;
    Line(ll m,ll c):m(m),c(c),p(0){}
    Line(ll p):m(0),c(0),p(p){}
    bool operator<(const Line &o)const{
        return querymode?p<o.p:m>o.m;
    }
};

bool Line::querymode=false;

struct LineContainer:multiset<Line>{
    ll div(ll a,ll b){
        return a/b-((a^b)<0&&a%b);
    }
    bool isect(iterator x,iterator y){
        if(y==end())return x->p=LINF,false;
        if(x->m==y->m)x->p=x->c<=y->c?LINF:-LINF;
        else x->p=div(x->c-y->c,y->m-x->m);
        return x->p>=y->p;
    }
    void add(ll m,ll c){
        auto x=insert(Line(m,c)),y=next(x);
        while(isect(x,y))y=erase(y);
        if((y=x)!=begin()&&isect(--x,y))isect(x,erase(y));
        while((y=x)!=begin()&&(--x)->p>=y->p)isect(x,erase(y));
    }
    ll get(ll x){
        if(empty())return LINF;
        Line::querymode=true;
        auto l=lower_bound(Line(x));
        Line::querymode=false;
        return l->m*x+l->c;
    }
};
```

Treap.h

Description: A short self-balancing tree. It acts as a sequential container with log-time splits/joins, and is easy to augment with additional data.
Time: $\mathcal{O}(\log N)$

1754b4, 53 lines

```
struct Node {
  Node *l = 0, *r = 0;
  int val, y, c = 1;
  Node(int val) : val(val), y(rand()) {}
  void recalc();
};

int cnt(Node* n) { return n ? n->c : 0; }
void Node::recalc() { c = cnt(l) + cnt(r) + 1; }

template<class F> void each(Node* n, F f) {
  if (n) { each(n->l, f); f(n->val); each(n->r, f); }
}

pair<Node*, Node*> split(Node* n, int k) {
  if (!n) return {};
  if (cnt(n->l) >= k) { // "n->val >= k" for lower_bound(k)
    auto [L,R] = split(n->l, k);
    n->l = R;
    n->recalc();
    return {L, n};
  } else {
    auto [L,R] = split(n->r,k - cnt(n->l) - 1); // and just "k"
    n->r = L;
    n->recalc();
    return {n, R};
  }
}

Node* merge(Node* l, Node* r) {
  if (!l) return r;
  if (!r) return l;
  if (l->y > r->y) {
    l->r = merge(l->r, r);
    return l->recalc(), l;
  } else {
    r->l = merge(l, r->l);
    return r->recalc(), r;
  }
}

Node* ins(Node* t, Node* n, int pos) {
  auto [l,r] = split(t, pos);
  return merge(merge(l, n), r);
}

// Example application: move the range [l, r) to index k
void move(Node*& t, int l, int r, int k) {
  Node *a, *b, *c;
  tie(a,b) = split(t, l); tie(b,c) = split(b, r - l);
  if (k <= l) t = merge(ins(a, b, k), c);
  else t = merge(a, ins(c, b, k - r));
}
```

LinkCutTree.hpp

Description: Link Cut Tree (1-indexed)

"src/contest/template.hpp" 38324f, 78 lines

```
template<int N,class T>
struct LinkCutTree{
    int ch[N][2],par[N],lz[N],rev[N];
    T val[N],sum[N],rsum[N];
    void toggle(int v){
        if(!v)return;
        swap(ch[v][0],ch[v][1]);
        swap(sum[v],rsum[v]);
        rev[v]^=1;
    }
    void push(int v){
        if(!v||!rev[v])return;
        toggle(ch[v][0]);
        toggle(ch[v][1]);
        rev[v]=0;
    }
    void pull(int v){
        if(!v)return;
        sum[v]=sum[ch[v][0]]+val[v]+sum[ch[v][1]];
        rsum[v]=rsum[ch[v][0]]+val[v]+rsum[ch[v][1]];
    }
    bool is_root(int v){
        return ch[par[v]][0]!=v&&ch[par[v]][1]!=v;
    }
    bool pos(int v){
        return ch[par[v]][1]==v;
    }
    void rotate(int v){
        int u=par[v],g=par[u];
        bool x=pos(v);
        if(!is_root(u))ch[g][pos(u)]=v;
        ch[u][x]=ch[v][!x];
        if(ch[u][x])par[ch[u][x]]=u;
        ch[v][!x]=u,par[u]=v,par[v]=g;
        pull(u),pull(v);
    }
    void splay(int v){
        if(!v)return;
        for(push(v);!is_root(v);rotate(v)){
            int u=par[v];
            if(is_root(u))push(u),push(v);
            else push(par[u]),push(u),push(v),rotate(pos(u)==
                 pos(v)?u:v);
        }
    }
    void access(int v){
        for(int u=v,c=0;u;u=par[u]){
            splay(u);
            ch[u][1]=c;
            pull(c=u);
        }
        splay(v);
    }
    void evert(int v){
        access(v),toggle(v);
    }
    void link(int u,int v){
        evert(u);
        access(v);
        par[u]=v;
    }
    void cut(int u,int v){
        evert(u);
        access(v);
        assert(par[u]==v);
        ch[v][0]=par[u]=0;
        pull(v);
    }
    T aggregate(int u,int v){
        evert(u);
        access(v);
        return sum[v];
    }
    void set(int u,T v){
        evert(u);
        val[u]=v;
        pull(u);
    }
};
```