Contents

1 Basic

1.1 .vimrc

1.2 Increase Stack Size

```
//stack resize
asm( "mov %0,%%esp\n" ::"g"(mem+10000000) );
//change esp to rsp if 64-bit system

//stack resize (linux)
#include <sys/resource.h>
void increase_stack_size() {
  const rlim_t ks = 64*1024*1024;
  struct rlimit rl;
  int res=getrlimit(RLIMIT_STACK, &rl);
  if(res==0){
    if(rl.rlim_cur<ks){
      rl.rlim_cur=ks;
      res=setrlimit(RLIMIT_STACK, &rl);
    }
  }
}
```

2 Graph

2.1 HLD

```
struct segment_tree{
    #define MAXN 100100
    #define right(x) x << 1 | 1
    #define left(x) x << 1
    int* arr;
    LL sum[4*MAXN];
    const int inf = 1e9;

    void pull(int ind) {
        sum[ind] = sum[right(ind)]+sum[left(ind
            )];
    };
    /// root => 1
    void build(int ind, int l, int r) {

        if( r - l == 1) {
            sum[ind] = 0;
            return;
        }
        int mid = (l+r)>>1;
        build( left(ind), l, mid );
        build( right(ind), mid, r );
        pull(ind);
    }
    LL query_sum(int ind, int L, int R, int ql,
         int qr) {
        if( L >= qr || R <= ql ) return 0;
```

```
        if( R <= qr && L >= ql ) {
            return sum[ind];
        }
        int mid = (L+R)>>1;
        return query_sum(left(ind), L, mid, ql,
             qr) + query_sum(right(ind), mid, R
            , ql, qr);
    }
    void modify(int ind, int L, int R, int ql,
        int qr, int x) {
        if( L >= qr || R <= ql ) return;
        if( R <= qr && L >= ql ) {
            sum[ind] = x;
            return;
        }
        int mid = (L+R)>>1;
        modify(left(ind), L, mid, ql, qr, x);
        modify(right(ind), mid, R, ql, qr, x);
        pull(ind);
    }
};

struct Tree{
    segment_tree seg;
    #define MAXN 100010
    #define maxm (maxn<<1)
    int n;
    struct edge { int u, v; };
    vector<edge> e;
    void addedge(int x, int y) {
        G[x].pb( SZ(e) );
        G[y].pb( SZ(e) );
        e.pb( edge{x, y}  );
    }
    int siz[MAXN],max_son[MAXN],pa[MAXN],dep[
        MAXN];
    /*size of subtreeindex of max_son, parent
         indexdepth*/
    int link_top[MAXN],link[MAXN],Time;
    /*chain topindex in segtreetime stamp*/
    std::vector<int >G[MAXN];

    void init(int N) {
        n = N;
        e.clear();
        for(int i = 1; i <= n; i++) G[i].clear
            ();
    }
    void find_max_son(int x){
        siz[x]=1;
        max_son[x]=-1;
        for(int e_ind : G[x]) {
            int v = e[e_ind].u == x ? e[e_ind].
                v : e[e_ind].u ;
            if( v == pa[x] )continue;
            pa[v] = x; dep[v] = dep[x] + 1;
            find_max_son(v);
            if(max_son[x] == -1 || siz[v] > siz
                [max_son[x]])
                max_son[x] = v;
            siz[x] += siz[v];
        }
    }
    void build_link(int x,int top){
        link[x] = ++Time;/*   x*/
        link_top[x] = top;
        if(max_son[x] == -1)return;
        build_link( max_son[x], top);/*    */
        for(int e_ind : G[x]) {
            int v = e[e_ind].u == x ? e[e_ind].
                v : e[e_ind].u ;
            if( v == max_son[x] || v == pa[x] )
                continue;
            build_link(v, v);
        }
    }
    inline int lca(int a,int b){
        /* ,      LCA*/
        int ta=link_top[a],tb=link_top[b];
        while(ta != tb){
            if(dep[ta]<dep[tb]){
                std::swap(ta,tb);
                std::swap(a,b);
            }
            //interval [ link[ta], link[a] ]
            a = pa[ta];
            ta = link_top[a];
        }
        return dep[a] < dep[b] ? a:b;
    }
    int query(int a,int b){
        int ret = 0;
        int ta=link_top[a],tb=link_top[b];
        while(ta != tb){
            if(dep[ta]<dep[tb]){
                std::swap(ta,tb);
                std::swap(a,b);
            }
            //interval [ link[ta],link[a] ]
            a = pa[ta];
            ta = link_top[a];
        }
        if( a == b ) return ret;
        else {
            if(dep[a]>dep[b])
                swap(a,b);
            //interval [ link[a],link[b] ]
            // if operate on edges ==> [ link[
                max_son[ta] ], link[b] ]
        }
    }
    /// Heavy Light Decomposition
    void HLD() {
        // root is indexed 1 here !
```

```
        find_max_son(1);
        build_link(1, 1);
    }
     void modify(int a, int b, int x) {
        // modify the path from a -> b to x
        //( which is [ link[a] .. link[b] ] on
            the segment tree)
        seg.modify(1, 1, n+1, link[a], link[b
            ]+1, x);
        // this segment tree uses [ 1 ..n+1 )
    }
}tree;
```

3 Data Structure

3.1 Disjoint Set

```
struct Disjoint_set {
    #define MAX_N 500005
    // define MAX_N
    int pa[MAX_N], Rank[MAX_N];
    int sz[MAX_N];
    void init_union_find(int V) {
        for(int i=0; i<V; i++) {
            pa[i] = i;
            Rank[i] = 0;
            sz[i] = 1;
        }
    }
    int find(int x) {
        return x == pa[x] ? x : pa[x] = find(pa
            [x]);
    }

    int unite(int x, int y) {
        x = find(x), y = find(y);
        int S = sz[x]+sz[y];
        if(x != y){
            if(Rank[x] < Rank[y]) {
                pa[x] = y;
                sz[y]=S;
                return y;
            }
            else{
                pa[y] = x;
                sz[x] = S;
                if(Rank[x] == Rank[y]) Rank[x]
                    ++;
                return x;
            }
        }
    }
    bool same(int x, int y) {
        return find(x) == find(y);
    }
}dsj;
```