Contents

1 Data Structure

1.1 segment tree

```
class Seg{
  private:
    int arr[MAXN*4+5];
    int tag[MAXN*4+5];
    void pull(int now) {
      arr[now] = arr[now*2] + arr[now*2+1];
    }
    void push(int now, int len) {
      arr[now] += tag[now];
      if (len > 1) {
        tag[now*2  ] += tag[now];
        tag[now*2+1] += tag[now];
      }
      tag[now] = 0;
    }
  public:
    void build(int now=1, int l=1, int r=n) {
      if (l == r) {
        arr[now] = 0;
        return;
      }
      int mid = l + r >> 1;
      build(now*2  , l, mid);
      build(now*2+1,mid+1,r);
      pull(now);
      return;
    }
    void mdy(int ml, int mr, int k, int now=1, int l=1,
         int r=n) {
      push(now, r-l+1);
      if (ml <= l && r <= mr) {
        tag[now] += k;
        push(now, r-l+1);
        return;
      } else if (l > mr || r < ml) return;
      int mid = l + r >> 1;
      mdy(ml, mr, k, now*2  , l, mid);
      mdy(ml, mr, k, now*2+1,mid+1,r);
      pull(now);
      return;
    }
    int qry(int ql, int qr, int now=1, int l=1, int r=n
        ) {
      push(now, r-l+1);
      if (ql <= l && r <= qr) {
        return arr[now];
      } else if (l > mr || r < ml) return 0;
      int mid = l + r >> 1, sum = 0;
      sum += qry(ql, qr, now*2  , l, mid);
      sum += qry(ql, qr, now*2+1,mid+1,r);
      pull(now);
      return sum;
    }
} seg;
```

1.2 treap

```
size_t Rand = 7122;
inline size_t Random() {
  return Rand = Rand * 0xdefaced + 1;
}
class Treap{
  private:
    struct node{
      int l, r, pri, key, sze;
      node() {
        l = r = sze = 0;
      }
      node(int _k) {
        l = r = 0, pri = Random(), key = _k, sze = 1;
      }
    } arr[MAXN+1];
    void pull(int now) {
      if (!now) return;
```

```
      arr[now].sze = arr[arr[now].l].sze + arr[arr[now
          ].r].sze + 1;
    }
    int cnt;
  public:
    int Merge(int a, int b) {
      if (!a || !b) return a ? a : b;
      if (arr[a].pri > arr[b].pri) {
        arr[a].r = Merge(arr[a].r, b);
        pull(a);
        return a;
      } else {
        arr[b].l = Merge(a, arr[b].l);
        pull(b);
        return b;
      }
    }
    void Split_by_key(int o, int &a, int &b, int k) {
      if (!o) a = b = 0;
      else if (arr[o].key <= k) {
        a = o;
        Split_by_key(arr[o].r, arr[a].r, b, k);
      } else {
        b = o;
        Split_by_key(arr[o].l, a, arr[b].l, k);
      }
      pull(o);
    }
    void Split_by_sze(int o, int &a, int &b, int s) {
      if (!o) a = b = 0;
      else if (arr[arr[o].l].sze + 1 <= s) {
        a = o;
        Split_by_sze(arr[o].r, arr[a].r, b, s-(arr[arr[
            o].l].sze+1));
      } else {
        b = o;
        Split_by_sze(arr[o].l, a, arr[b].l, s);
      }
      pull(o);
    }
    bool Insert(int x, int &root) {
      int a = 0, b = 0, c = 0;
      Split_by_key(root, b, c, x), root = b;
      Split_by_key(root, a, b, x-1);
      if (arr[b].sze) {
        root = Merge(a, Merge(b, c));
        return 0;
      }
      arr[++cnt] = node(x);
      root = Merge(Merge(a, cnt), c);
      return 1;
    }
    bool Erase(int x, int &root) {
      int a = 0, b = 0, c = 0;
      Split_by_key(root, b, c, x), root = b;
      Split_by_key(root, a, b, x-1);
      root = Merge(a, c);
      if (!arr[b].sze) return 0;
      return 1;
    }
    int kth(int k, int &root) {
      if (k < 1 || k > arr[root].sze) return -1;
      int a = 0, b = 0, c = 0;
      Split_by_sze(root, a, b, arr[root].sze - k), root
          = b;
      Split_by_sze(root, b, c, arr[root].sze - k + 1);
      root = Merge(a, Merge(b, c));
      return arr[b].key;
    }
} treap;
```

1.3 persistent segment tree

```
class Per_seg{
  private:
    struct node{
      int l, r, c;
    } arr[MAXN*C];
    int cnt;
    int new_mem() {
      return ++cnt;
    }
  public:
    void build(int now=1, int l=1, int r=len) {
      if (l == r) return;
      int mid = l + r >> 1;
      arr[now].l = new_mem();
      arr[now].r = new_mem();
      build(arr[now].l, l, mid);
      build(arr[now].r,mid+1,r);
    }
    void add(int id, int k) {
      int o = root[id-1];
      root[id] = r = new_mem();
      arr[r] = arr[o];
      int L = 1, R = len, mid;
      while (L < R) {
        arr[r].c++;
        mid = L + R >> 1;
        if (k <= mid) {
          arr[r].l = new_mem();
          r = arr[r].l;
          arr[r] = arr[o = arr[o].l];
          R = mid;
        } else {
          arr[r].r = new_mem();
          r = arr[r].r;
          arr[r] = arr[o = arr[o].r];
          L = mid+1;
        }
      }
      arr[r].c++;
    }
    int kth(int l, int r, int k) {
      r = root[r], l = root[l-1];
      int L = 1, R = len, mid;
      while (L < R) {
        int t = arr[arr[r].l].c - arr[arr[l].l].c;
        mid = L + R >> 1;
        if (k <= t) {
          r = arr[r].l, l = arr[l].l;
          R = mid;
        } else {
          k -= t;
          r = arr[r].r, l = arr[l].r;
          L = mid+1;
        }
      }
      return L;
    }
} seg;
```

1.4 lichao segment tree

```
struct line{
  double a, b;
  int l, r;
};
class LiChao_Seg{
  private:
    int arr[MAXN*4+5];
    double calc(int id, int x) {
      return p[id].a * x + p[id].b;
    }
  public:
    void mdy(int ml, int mr, int v, int now=1, int l=1,
         int r=MAXN) {
      int mid = l + r >> 1;
      if (ml <= l && r <= mr) {
        int o = arr[now];
        double reso = calc(o, mid), resv = calc(v, mid)
            ;
        if (resv > reso) arr[now] = v;
        if (l == r) return;
        if (p[v].a < p[o].a) {
          if (reso >= resv)
            mdy(ml    , mr    , v, now*2  , l, mid);
          else
            mdy(p[o].l, p[o].r, o, now*2+1,mid+1,r);
        } else if (p[v].a > p[o].a) {
```

```
        if (resv >= reso)
          mdy(p[o].l, p[o].r, o, now*2  , l, mid);
        else
          mdy(ml    , mr    , v, now*2+1,mid+1,r);
      }
      return;
    } else if (l > mr || r < ml) return;
    mdy(ml, mr, v, now*2  , l, mid);
    mdy(ml, mr, v, now*2+1,mid+1,r);
  }
  pdi qry(int d, int now=1, int l=1, int r=MAXN) {
    pdi res = pdi(calc(arr[now], d), arr[now]);
    if (l == d && r == d) {
      return res;
    } else if (l > d || r < d) return pdi(-INF, 0);
    int mid = l + r >> 1;
    res = max(res, qry(d, now*2  , l, mid));
    res = max(res, qry(d, now*2+1,mid+1,r));
    return res;
  }
} seg;
```

```
    queue <int> que;
    for (int i = 1; i <= cnt; i++)
      if (!arr[i].chd) que.push(i);
    while (que.size()) {
      int now = que.front();
      que.pop();
      if (arr[now].is_pre) arr[now].t++;
      arr[arr[now].pa].t += arr[now].t;
      arr[arr[now].pa].chd--;
      if (arr[now].pa && !arr[arr[now].pa].chd)
        que.push(arr[now].pa);
    }
  }
  int solve(string &p) {
    int now = 0;
    for (int i = 0; i < p.size(); i++) {
      if (arr[now].ch[p[i]-'a'])
        now = arr[now].ch[p[i]-'a'];
      else return 0;
    }
    return arr[now].t;
  }
};
```

2 String

2.1 SAM

```
class SAM{
  private:
    struct node{
      int ch[26];
      int len, pa, t, chd;
      bool is_pre;
      node() {
        memset(ch, 0, sizeof(ch));
        len = pa = t = chd = 0;
        is_pre = 0;
      }
    } arr[MAXN<<1];
    vector <int> reBFS[MAXN];
    int cnt, las;
    void add(int c) {
      int p = las;
      int cur = las = ++cnt;
      arr[cur].len = arr[p].len + 1;
      arr[cur].is_pre = 1;
      while (p && !arr[p].ch[c]) {
        arr[p].ch[c] = cur;
        p = arr[p].pa;
      }
      if (!arr[p].ch[c]) {
        arr[cur].pa = 0;
        arr[0].chd++;
        arr[p].ch[c] = cur;
      } else {
        int q = arr[p].ch[c];
        if (arr[q].len == arr[p].len + 1) {
          arr[cur].pa = q;
          arr[q].chd++;
        } else {
          int nq = ++cnt;
          arr[nq] = arr[q];
          arr[nq].is_pre = 0;
          arr[nq].len = arr[p].len + 1;
          arr[q].pa = arr[cur].pa = nq;
          arr[nq].chd = 2;
          for (; arr[p].ch[c] == q; p = arr[p].pa)
            arr[p].ch[c] = nq;
        }
      }
    }
  public:
    void init(string s) {
      for (int i = 0; i <= cnt; i++)
        arr[i] = node();
      cnt = las = 0;
      arr[0].t = 1;
      for (int i = 0; i < s.size(); i++)
        add(s[i] - 'a');
```