SP-51 C-10: Code Library

目錄

1 基本配置

1.1 快速讀寫

```cpp
struct FastIS {
  static const int LEN = 1<<21|1;
  char buf[LEN], *l, *r, c, *t; bool neof, neg;
  FastIS() : l(buf), r(buf), neof(1) {}
  inline operator bool() { return neof; }
  inline char gc() {
    return
      !neof || (l==r && (r = (l=buf) + fread(buf, 1, LEN, stdin), l==r)) ?
      neof=0, -1 : *l++;
  }
  FastIS& operator>>(char &c) {
    for (c=gc(); isspace(c); c=gc()) {} return *this;
  }
  FastIS& operator>>(char *s) {
    for (c=gc(); isspace(c); c=gc()) {}
    for (t=s; ~c and !isspace(c); c=gc()) *t++=c;
    *t='\0'; return *this;
  }
  template<typename T> FastIS& operator>>(T &x) {
    for (c=gc(); ~c and !isdigit(c); c=gc()) neg ^= c == '-';
    for (x=0; isdigit(c); c=gc()) x = 10*x+c-'0';
    if (neg) { x=-x, neg=0; } return *this;
  }
} fast_in;

struct FastOS {
  static const int LEN=1<<21|1, W=21;
  char buf[LEN], *l, *r, *t, stk[W]; int top;
  FastOS() : l(buf), r(buf+LEN) {}
  inline void flush() { fwrite(buf, 1, l-buf, stdout); l=buf; }
  FastOS& operator<<(char c) { *l++=c; if (l==r) flush(); return *this; }
  FastOS& operator<<(char *s) {
    for (t=s; *t!='\0'; ++t) { *this<<*t; } return *this;
  }
  FastOS& operator<<(const char *s) { return *this<<(char*)s; }
  template<typename T> FastOS& operator<<(T x) {
    if (!x) *this<<'0'; else if (x<0) *this<<'-', x=-x;
    for (top=0; x; x/=10) stk[++top] = x%10+'0';
    while (top) { *this<<stk[top--]; } return *this;
  }
  ~FastOS() { flush(); }
} fast_out;
```

1.2 Montogomery 乘法

```cpp
using u64 = unsigned long long;
using u32 = unsigned int;
using i64 = long long;

struct u128 {
  u64 h, l;
  static u128 mul(const u64 &x, const u64 &y) {
    u32 xh = x>>32, yh = y>>32, xl = x, yl = y;
    u64 hh = static_cast<u64>(xh) * yh, ll = static_cast<u64>(xl) * yl;
    u64 hl = static_cast<u64>(xh) * yl, lh = static_cast<u64>(xl) * yh;
    u64 c = (hl<<32>>32) + (lh<<32>>32) + (ll>>32);
    u64 h = hh + (hl>>32) + (lh>>32) + (c>>32);
    u64 l = (hl<<32) + (lh<<32) + ll;
    return {h, l};
  }
};

struct Montgomery {
  u64 mod, inv, r2;
  u64 reduce(u128 x) {
    u64 q = x.l * inv;                  // q ≡ x / n  (mod r)
    i64 a = x.h - u128::mul(q, mod).h; // x - qn ≡ 0 (mod r)
    return a<0 ? a+=mod : a;
  }
  u64 mul(u64 x, u64 y) { return reduce(u128::mul(x, y)); }
  Montgomery(u64 mod) : mod(mod), inv(1), r2(-mod % mod) {
    // 2 ^ (2 ^ 6)     = 2 ^ 64
    for (int i=0; i<6; ++i) inv *= 2 - mod * inv;
    // 2 ^ (4 * 2 ^ 4) = 2 ^ 64
    for (int i=0; i<4; ++i) if ((r2 <<= 1) >= mod) r2 -= mod;
    for (int i=0; i<4; ++i) r2 = mul(r2, r2);
  }
  inline u64 init(u64 x) { return mul(x, r2); }
  inline u64 reduce(u64 x) { return reduce({0, x}); }
};
```

1.3 暫用 Treap 隨機數生成器

```cpp
struct rot_xor {
  int seed = 0x95abcfad;
  inline unsigned int operator()() {
    return seed = (seed << 1) ^ ((seed >> 31) & 0xa53a9be9);
```

```
  }
};
```

2 字符串

2.1 Z 算法

```
void z_func(char s[], int n, int z[]) {
  int l = 0, r = 0; z[0] = 0;
  for (int i=1; i<n; ++i) {
    z[i] = i < r ? min(z[i-l], r-i) : 0;
    while (i+z[i] < n && s[i+z[i]] == s[z[i]]) ++z[i];
    if (chkmx(r, i+z[i])) l = i;
  }
}
```

2.2 Manacher

```
int manacher_init(char in[], int n, char out[]) {
  int m = (n+1)<<1;
  out[0] = out[1] = '#'; out[m+1] = '@';
  for (int i=1; i<=n; ++i) out[i<<1] = in[i-1], out[i<<1|1] = '#';
  return m;
}
void manacher_work(char s[], int n, int p[]) {
  int mid = 0, r = 0; p[0] = 1;
  for (int i=1; i<n; ++i) {
    p[i] = i < r ? min(p[2*pos-i], r-i) : 1;
    while (s[i-p[i]] == s[i+p[i]]) ++p[i];
    if (chkmx(r, i+p[i])) mid = i;
  }
}
```

2.3 迴文自動機

slink 指向沿着 fail 邊長度不等差的第一個後綴。
trans 指向長度小於等於當前迴文串一半的最長後綴。

```
namespace PAM {
  int len[MAX_N], ch[MAX_N][26], fail[MAX_N];
  int diff[MAX_N], slink[MAX_N];
  int trans[MAX_N];
  int dep[MAX_N];
  int indx, last;
  char s[MAX_N];

  int new_node(int l) {
    len[indx] = l;
    memset(ch[indx], 0, sizeof(ch[indx])); return indx++;
  }
  void init() {
    indx = 0; new_node(0); new_node(-1);
    s[0] = '#'; fail[0] = 1; dep[0] = dep[1] = 0; diff[0] = 0; last = 0;
  }
  int getfail(int p, int i) {
    while (s[i-len[p]-1] != s[i]) p = fail[p];
    return p;
  }
  void insert(int i) {
    int p = getfail(last, i);
    if (!ch[p][s[i]-'a']) {
      int q = new_node(len[p]+2);
      fail[q] = ch[getfail(fail[p], i)][s[i]-'a'];
      dep[q] = dep[fail[q]] + 1;
      diff[q] = len[q]-len[fail[q]];
      slink[q] = diff[q]==diff[fail[q]] ? slink[fail[q]] : fail[q];
      if (len[q] <= 2) {
        trans[q] = fail[q];
      } else {
        int r = trans[p];
        while (s[i-len[r]-1] != s[i] || (len[r]+2)*2 > len[q]) r = fail[r];
        trans[q] = ch[r][s[i]-'a'];
      }
      ch[p][s[i]-'a'] = q;
    }
    last = ch[p][s[i]-'a'];
  }
} // namespace PAM
```

2.4 後綴自動機

```
namespace SAM {
int len[MAX_2N], ch[MAX_2N][26], fail[MAX_2N];
int cnt[MAX_2N];
int indx, last;

int new_node(int l) {
  len[indx] = l; cnt[indx] = 1;
  memset(ch[indx], -1, sizeof(ch[indx])); return indx++;
}
int copy_node(int q, int l) {
```

```cpp
  len[indx] = l; cnt[indx] = 0; fail[indx] = fail[q];
  memcpy(ch[indx], ch[q], sizeof(ch[q])); return indx++;
}
void init() {
  indx = 0; new_node(0);
  fail[0] = -1; last = 0;
}
void insert(int w) {
  int cur = new_node(len[last] + 1);
  int p = last;
  for (; ~p && ch[p][w] == -1; p = fail[p]) ch[p][w] = cur;
  if (p == -1) {
    fail[cur] = 0;
  } else if (int q = ch[p][w]; len[q] == len[p] + 1) {
    fail[cur] = q;
  } else {
    int r = copy_node(q, len[p] + 1);
    for (; ~p && ch[p][w] == q; p = fail[p]) ch[p][w] = r;
    fail[cur] = fail[q] = r;
  }
  last = cur;
}
}  // namespace SAM
```

2.5 Lyndon 分解

pms 前綴的最小後綴。

```cpp
void duval(const char s[], int n, int pms[]) {
  int i = 0;
  while (i < n) {
    pms[i] = i;
    int j = i, k = i+1;
    while (k < n && s[j] <= s[k]) {
      if (s[j] < s[k]) {
        pms[k] = i;
        j = i, ++k;
      } else {
        pms[k] = pms[j] + (k - j);
        ++j, ++k;
      }
    }
    while (i <= j) {
      // handle [i, i + (k - j))
      i += k - j;
    }
```

```cpp
  }
}
```

3 圖論

3.1 樹上啓發式合併

```cpp
void dsu(int u) {
  for (auto &v: e[u]) if (v != fa[u] && v != hson[u]) {
    dsu(v);
    for (int i=dfn[v]; i<=rdfn[v]; ++i) del(dfn[i]);
  }
  if (hson[u]) dfs(hson[u]);
  for (auto &v: e[u]) if (v != fa[u] && v != hson[u]) {
    for (int i=dfn[v]; i<=rdfn[v]; ++i) add(dfn[i]);
  }
}
```

4 數學

4.1 線性基

qry_kth 包括零，下標從零開始的第 k 大。只關注 k 的後 dim 位。
qry_rnk 返回小於 x 的向量個數（包括零），*eq* 表示能否表示出 x。

涉及刪除元素時可以考慮兩種方案：
- 記錄每個基的過期時間 *exp*。
- 記錄每個向量用基的表示方法。

```cpp
template<int w> struct LinearBasis {
  bitset<w> base[w]; int dim;
  LinearBasis() : dim(0) { for (int i=0; i<w; ++i) base[i].reset(); }
  int insert(bitset<w> x) {
    for (int i=w-1; i>=0; --i) if (x[i]) {
      if (base[i][i]) {
        x ^= base[i];
      } else {
        // qry_kth requires this:
```

```
        for (int j=i-1; j>=0; --j) if (x[j] && base[j][j]) x ^= base[j];
        for (int j=i+1; j<w; ++j) if (base[j][i]) base[j] ^= x;
        base[i] = x;
        ++dim; return i;
      }
    }
    return -1;
  }
  bitset<w> qry_kth(bitset<w> k) {
    bitset<w> res;
    for (int i=0; i<w; ++i) if (base[i][i]) {
      if (k[0]) res ^= base[i];
      k >>= 1;
    } return res;
  }
  bitset<w> qry_rnk(bitset<w> x, bool &eq) {
    int df = dim;
    bitset<w> res, cur;
    for (int i=w-1; i>=0; --i) {
      if (base[i][i]) --df;
      if (x[i] && (!cur[i] || base[i][i])) res.set(df);
      if (x[i] != cur[i]) {
        if (base[i][i]) cur ^= base[i];
        else break;
      }
    }
    eq = cur == x;
    return res;
  }
};
```

5 數據結構

5.1 左偏堆

二路分治進行合併操作可實現線性建堆。

```
namespace LeftistHeap {
int val[MAX_N], dist[MAX_N];
int lc[MAX_N], rc[MAX_N];

void push_up(int x) {
  if (dist[lc[x]] < dist[rc[x]]) swap(lc[x], rc[x]);
  dist[x] = dist[rc[x]] + 1;
}
int merge(int x, int y) {
  if (!x || !y) return x | y;
  if (val[x] > val[y]) swap(x, y);
  rc[x] = merge(rc[x], y);
  push_up(x); return x;
}
}  // namespace LeftistHeap
```

5.2 笛卡爾樹

```
static int stk[MAX_N];
int top = 0;
for (int i=1; i<=n; ++i) {
  int x = i, y = 0;
  while (top && h[i] < h[stk[top]]) y = stk[top--];
  lc[x] = y;
  rc[stk[top]] = x;
  stk[++top] = x;
}
return rc[0];
```

5.3 FHQ Treap

可持續化時只需要在 split 時複製結點。
⚠ 未測試 ⚠

```
namespace FHQTreap {
int rnd[MAX_N], val[MAX_N];
int siz[MAX_N], fa[MAX_N];
int lc[MAX_N], rc[MAX_N];
int indx, root;

int new_node(int v);
void push_up(int p);
void split(int p, int k, int &x, int &y) {
  if (!p) { x = 0; y = 0; }
  if (int t = siz[p] - siz[rc[p]]; t <= k) {
    x = p; split(rc[x], k-t, rc[x], y);
    fa[rc[x]] = x;
  } else {
    y = p; split(lc[y], k, x, lc[y]);
    fa[lc[y]] = y;
  } push_up(p);
}
```

```
int merge(int x, int y) {
  if (!x || !y) return x | y;
  if (rnd[x] < rnd[y]) {
    rc[x] = merge(rc[x], y);
    push_up(x); return fa[rc[x]] = x;
  } else {
    lc[y] = merge(x, lc[y]);
    push_up(y); return fa[lc[y]] = y;
  }
}
}  // namespace FHQTreap
```

5.4 Splay

⚠ 未測試 ⚠

```
namespace Splay {
#define lc(x) (ch[(x)][0])
#define rc(x) (ch[(x)][1])
#define isrc(x) (rc(fa[(x)]) == (x))
int val[MAX_N], cnt[MAX_N], siz[MAX_N];
int fa[MAX_N], ch[MAX_N][2];
int indx, root;

int new_node(int v);
void push_up(int x);
void rotate(int x) {
  int y = fa[x], z = fa[y], k = isrc(x), w = ch[x][!k];
  if (y) { ch[z][isrc(y)] = x; } ch[x][!k] = y; ch[y][k] = w;
  if (w) { fa[w] = y; } fa[y] = x; fa[x] = z;
  push_up(y);
}
void splay(int x) {
  for (int y; y = fa[x], x; rotate(x))
    if (y) rotate(isrc(x)^isrc(y) ? x : y);
  push_up(x);
  root = x;
}

void upd_ins(int v) {
  if (!root) return root = new_node(v);
  int p = root;
  while (1) {
    if (val[p] == v) {
      ++cnt[p]; return splay(p);
    } else {
      int k = v > val[p];
      if (!ch[p][k]) {
        int q = new_node(v);
        ch[p][k] = q; fa[q] = p;
        return splay(q);
      } else p = ch[p][k];
    }
  }
}
int qry_find(int v);
int qry_pre(int x);
void upd_del(int v) {
  int p = qry_find(v);
  if (!p) return;
  if (cnt[p] > 1) --cnt[p], --siz[p];
  else if (!lc(p) && !rc(p)) root = 0;
  else if (!lc(p) || !rc(p)) fa[root = lc(p) | rc(p)] = 0;
  else {
    fa[lc(p)] = 0;
    qry_pre(p);
    fa[rc(root) = rc(p)] = root;
    push_up(root);
  }
}
#undef lc
#undef rc
#undef isrc
}  // namespace Splay
```

5.5 Link Cut Tree

⚠ 未測試 ⚠

```
namespace LCT {
#define lc(x) (ch[(x)][0])
#define rc(x) (ch[(x)][1])
#define isrc(x) (rc(fa[(x)]) == (x))
#define nroot(x) (lc(fa[(x)]) == (x) || rc(fa[(x)]) == (x))
int fa[MAX_N], ch[MAX_N][2];
bool rev_tag[MAX_N];

void push_rev(int x);
void push_dn(int x);
void push_up(int x);
void rotate(int x) {
  int y = fa[x], z = fa[y], k = isrc(x), w = ch[x][!k];
  if (nroot(y)) { ch[z][isrc(y)] = x; } ch[x][!k] = y; ch[y][k] = w;
  if (w) { fa[w] = y; } fa[y] = x; fa[x] = z;
```

```cpp
  push_up(y);
}
void splay(int x) {
  static int stk[MAX_N], top;
  for (int y = stk[top = 1] = x; nroot(y); stk[++top] = (y = fa[y]));
  while (top) push_dn(stk[top--]);
  for (int y; y = fa[x], nroot(x); rotate(x))
    if (nroot(y)) rotate(isrc(x)^isrc(y) ? x : y);
  push_up(x);
}

int access(int x) {
  int y = 0;
  for(; x; x = fa[y=x]) splay(x), rc(x) = y, push_up(x);
  return y;
}
void evert(int x) { push_rev(access(x)); }
int split(int x,int y) { evert(x); return access(y); }
int find(int x) {
  for (x = access(x); lc(x); x = lc(x)) push_dn(x);
  return splay(x), x;
}
void link(int x,int y) { evert(x), splay(x); fa[x] = y; }
void cut(int x,int y) { split(x,y), splay(x); rc(x) = fa[y] = 0; push_up(x); }
bool chk_link(int x,int y) { evert(x); return find(y) != x ? fa[x] = y : 0; }
bool chk_cut(int x,int y){
  evert(x);
  if (find(y)!=x || fa[y]!=x || ch[y][0]) return 0;
  rc(x) = fa[y]=0; push_up(x); return 1;
}
#undef lc
#undef rc
#undef isrc
#undef nroot
}  // namespace LCT
```