SP-51 C-10: Code Library

目錄

1 基本配置

1.1 快速讀寫

基於緩衝區的快速讀寫。

```
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;
```

2 字符串

2.1 Z 算法

z_i 後綴與原串的 LCP。

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

2.2 Manacher

p_i: i 爲中心的最長迴文串長度。

```
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 mxr=0, pos=0;
  for (int i=1; i<n; ++i) {
    p[i] = i<mxr ? min(p[2*pos-i], mxr-i) : 1;
    while (s[i-p[i]]==s[i+p[i]]) ++p[i];
    if (chkmx(mxr, i+p[i])) pos=i;
  }
}
```

2.3 迴文自動機

diff slink 每個等差序列中需要單獨處理的後綴長度爲 len[slink]+diff。
trans 指向長度小於等於當前迴文串一半的後綴。

```
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 indx, last;
  char s[MAX_N];

  int new_node(int l) {
    ++indx; len[indx] = l; memset(ch[indx], 0, sizeof(ch[indx]));
    return indx;
  }
  void init() {
    indx = -1; new_node(0); new_node(-1);
    s[0] = '#'; fail[0] = 1; 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'];
      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'];
  }
}
```