字符串

C/C++ 标准库中的字符串

C 标准库

- `sscanf(const char *__source, const char *__format, ...)`：从字符串 `__source` 里读取变量，比如 `sscanf(str,"%d",&a)` 。
- `sprintf(char *__stream, const char *__format, ...)`：将 `__format` 字符串里的内容输出到 `__stream` 中，比如 `sprintf(str,"%d",i)` 。
- `strlen(const char *str)`：返回从 `str[0]` 开始直到 `'\0'` 的字符数。注意，未开启 O2 优化时，该操作写在循环条件中复杂度是 的。
- `strcmp(const char *str1, const char *str2)`：按照字典序比较 `str1` `str2` 若 `str1` 字典序小返回负值，两者一样返回 `0` ， `str1` 字典序更大则返回正值。请注意，不要简单的认为返回值只有 `0` 、 `1` 、 `-1` 三种，在不同平台下的返回值都遵循正负，但并非都是 `0` 、 `1` 、 `-1` 。
- `strcpy(char *str, const char *src)`：把 `src` 中的字符复制到 `str` 中， `str` `src` 均为字符数组头指针，返回值为 `str` 包含空终止符号 `'\0'` 。
- `strncpy(char *str, const char *src, int cnt)`：复制至多 `cnt` 个字符到 `str` 中，若 `src` 终止而数量未达 `cnt` 则写入空字符到 `str` 直至写入总共 `cnt` 个字符。
- `strcat(char *str1, const char *str2)`：将 `str2` 接到 `str1` 的结尾，用 `*str2` 替换 `str1` 末尾的 `'\0'` 返回 `str1` 。

- strstr(char *str1, const char *str2)：若 str2 是 str1 的子串，则返回 str2 在 str1 的首次出现的地址；如果 str2 不是 str1 的子串，则返回 NULL 。
- strchr(const char *str, int c)：找到在字符串 str 中第一次出现字符 c 的位置，并返回这个位置的地址。如果未找到该字符则返回 NULL 。
- strrchr(const char *str, char c)：找到在字符串 str 中最后一次出现字符 c 的位置，并返回这个位置的地址。如果未找到该字符则返回 NULL 。

C++ 标准库

- 重载了赋值运算符 + ，当 + 两边是 string/char/char[]/const char* 类型时，可以将这两个变量连接，返回连接后的字符串（ string ）。
- 赋值运算符 = 右侧可以是 const string/string/const char*/char* 。
- 访问运算符 [cur] 返回 cur 位置的引用。
- 访问函数 data()/c_str() 返回一个 const char* 指针，内容与该 string 相同。
- 容量函数 size() 返回字符串字符个数。
- find(ch, start = 0) 查找并返回从 start 开始的字符 ch 的位置； rfind(ch) 从末尾开始，查找并返回第一个找到的字符 ch 的位置（皆从 0 开始）（如果查找不到，返回 -1 ）。
- substr(start, len) 可以从字符串的 start （从 0 开始）截取一个长度为 len 的字符串（缺省 len 时代码截取到字符串末尾）。
- append(s) 将 s 添加到字符串末尾。
- append(s, pos, n) 将字符串 s 中，从 pos 开始的 n 个字符连接到当前字符串结尾。
- replace(pos, n, s) 删除从 pos 开始的 n 个字符，然后在 pos 处插入串 s 。
- erase(pos, n) 删除从 pos 开始的 n 个字符。
- insert(pos, s) 在 pos 位置插入字符串 s 。
- std::string 重载了比较逻辑运算符，复杂度是 的。

KMP

```cpp
#include <cstdio>
#include <cstring>
char a[1000005], b[1000005];
int al, bl, nxt[1000005];
int main() {
	scanf("%s%s", a + 1, b + 1);
	al = strlen(a + 1), bl = strlen(b + 1);
	for (int i = 2, j = 0; i <= bl; i++) {
		while(j && b[j + 1] != b[i]) j = nxt[j];
		if (b[j + 1] == b[i]) j++;
		nxt[i] = j;
	}
	for (int i = 1, j = 0; i <= al; i++) {
		while(j && b[j + 1] != a[i]) j = nxt[j];
		if (b[j + 1] == a[i]) j++;
		if (j == bl) printf("%d\n", i - j + 1);
	}
	for (int i = 1; i <= bl; i++) printf("%d ", nxt[i]);
}
```

AC 自动机

```
#include <cstdio>
#include <queue>
#include <cstring>
using namespace std;
int n;
char s[200][100], t[1000005];
struct AC {
    int ch[11000][26], tot, fail[11000], val[11000], idx[11000], cnt[11000];
    bool end[11000];
    AC() {
        memset(ch, 0, sizeof(ch));
        memset(fail, 0, sizeof(fail));
        memset(val, 0, sizeof(val));
        memset(idx, 0, sizeof(idx));
        memset(cnt, 0, sizeof(cnt));
        tot = 0;
    }
    void insert(char s[], int id) {
        int u = 0;
        for (int i = 0; s[i]; i++) {
            if (!ch[u][s[i] - 'a']) ch[u][s[i] - 'a'] = ++tot;
            u = ch[u][s[i] - 'a'];
        }
        idx[u] = id;
    }
    void build() {
        queue<int> q;
        for (int i = 0; i < 26; i++) if(ch[0][i]) q.push(ch[0][i]);
        while(!q.empty()) {
            int u = q.front();
            q.pop();
            for (int i = 0; i < 26; i++)
                if (ch[u][i]) fail[ch[u][i]] = ch[fail[u]][i], q.push(ch[u][i]);
                else ch[u][i] = ch[fail[u]][i];
        }
    }
    int query(char s[]) {
        int u = 0, v, ans = 0;
        for (int i = 0; s[i]; i++) {
            v = u = ch[u][s[i] - 'a'];
            while (v) val[v]++, v = fail[v];
        }
        for (int i = 1; i <= tot; i++) if (idx[i]) ans = max(ans, val[i]), cnt[idx[i]] =
        return ans;
    }
};
int main() {
```

```
    while(scanf("%d", &n), n) {
        AC ac;
        for (int i = 1; i <= n; i++) scanf("%s", s[i]), ac.insert(s[i], i);
        ac.build();
        scanf("%s", t);
        int ans = ac.query(t);
        printf("%d\n", ans);
        for (int i = 1; i <= n; i++)
            if (ac.cnt[i] == ans)
                printf("%s\n", s[i]);
    }
}
```

后缀数组

```cpp
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <iostream>

using namespace std;

const int N = 1000010;

char s[N];
int n, sa[N], rk[N], oldrk[N << 1], id[N], key1[N], cnt[N];
// key1[i] = rk[id[i]]（作为基数排序的第一关键字数组）
int n, sa[N], rk[N], oldrk[N << 1], id[N], px[N], cnt[N];

bool cmp(int x, int y, int w) {
  return oldrk[x] == oldrk[y] && oldrk[x + w] == oldrk[y + w];
}

int main() {
  int i, m = 127, p, w;

  scanf("%s", s + 1);
  n = strlen(s + 1);
  for (i = 1; i <= n; ++i) ++cnt[rk[i] = s[i]];
  for (i = 1; i <= m; ++i) cnt[i] += cnt[i - 1];
  for (i = n; i >= 1; --i) sa[cnt[rk[i]]--] = i;

  for (w = 1;; w <<= 1, m = p) {  // m=p 就是优化计数排序值域
    for (p = 0, i = n; i > n - w; --i) id[++p] = i;
    for (i = 1; i <= n; ++i)
      if (sa[i] > w) id[++p] = sa[i] - w;

    memset(cnt, 0, sizeof(cnt));
    for (i = 1; i <= n; ++i) ++cnt[key1[i] = rk[id[i]]];
    // 注意这里px[i] != i，因为rk没有更新，是上一轮的排名数组

    for (i = 1; i <= m; ++i) cnt[i] += cnt[i - 1];
    for (i = n; i >= 1; --i) sa[cnt[key1[i]]--] = id[i];
    memcpy(oldrk + 1, rk + 1, n * sizeof(int));
    for (p = 0, i = 1; i <= n; ++i)
      rk[sa[i]] = cmp(sa[i], sa[i - 1], w) ? p : ++p;
    if (p == n) {
      for (int i = 1; i <= n; ++i) sa[rk[i]] = i;
      break;
    }
  }
```

```
  for (i = 1; i <= n; ++i) printf("%d ", sa[i]);

  return 0;
}
```

后缀自动机

```
struct state {
  int len, link;
  std::map<char, int> next;
};
const int MAXLEN = 100000;
state st[MAXLEN * 2];
int sz, last;
void sam_init() {
  st[0].len = 0;
  st[0].link = -1;
  sz++;
  last = 0;
}
void sam_extend(char c) {
  int cur = sz++;
  st[cur].len = st[last].len + 1;
  int p = last;
  while (p != -1 && !st[p].next.count(c)) {
    st[p].next[c] = cur;
    p = st[p].link;
  }
  if (p == -1) {
    st[cur].link = 0;
  } else {
    int q = st[p].next[c];
    if (st[p].len + 1 == st[q].len) {
      st[cur].link = q;
    } else {
      int clone = sz++;
      st[clone].len = st[p].len + 1;
      st[clone].next = st[q].next;
      st[clone].link = st[q].link;
      while (p != -1 && st[p].next[c] == q) {
        st[p].next[c] = clone;
        p = st[p].link;
      }
      st[q].link = st[cur].link = clone;
    }
  }
  last = cur;
}
```

广义后缀自动机

给定　个由小写字母组成的字符串　　，求本质不同的子串个数。（不包含空串）

```cpp
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 2000000;  // 双倍字符串长度
const int CHAR_NUM = 30;   // 字符集个数，注意修改下方的 (-'a')

struct exSAM {
  int len[MAXN];              // 节点长度
  int link[MAXN];             // 后缀链接，link
  int next[MAXN][CHAR_NUM];  // 转移
  int tot;                    // 节点总数：[0, tot)

  void init() {  // 初始化函数
    tot = 1;
    link[0] = -1;
  }

  int insertSAM(int last, int c) {  // last 为父 c 为子
    int cur = next[last][c];
    if (len[cur]) return cur;
    len[cur] = len[last] + 1;
    int p = link[last];
    while (p != -1) {
      if (!next[p][c])
        next[p][c] = cur;
      else
        break;
      p = link[p];
    }
    if (p == -1) {
      link[cur] = 0;
      return cur;
    }
    int q = next[p][c];
    if (len[p] + 1 == len[q]) {
      link[cur] = q;
      return cur;
    }
    int clone = tot++;
    for (int i = 0; i < CHAR_NUM; ++i)
      next[clone][i] = len[next[q][i]] != 0 ? next[q][i] : 0;
    len[clone] = len[p] + 1;
    while (p != -1 && next[p][c] == q) {
      next[p][c] = clone;
      p = link[p];
    }
    link[clone] = link[q];
    link[cur] = clone;
    link[q] = clone;
    return cur;
  }
```

```
  int insertTrie(int cur, int c) {
    if (next[cur][c]) return next[cur][c];  // 已有该节点 直接返回
    return next[cur][c] = tot++;            // 无该节点 建立节点
  }

  void insert(const string &s) {
    int root = 0;
    for (auto ch : s) root = insertTrie(root, ch - 'a');
  }

  void insert(const char *s, int n) {
    int root = 0;
    for (int i = 0; i < n; ++i)
      root =
          insertTrie(root, s[i] - 'a');  // 一边插入一边更改所插入新节点的父节点
  }

  void build() {
    queue<pair<int, int>> q;
    for (int i = 0; i < 26; ++i)
      if (next[0][i]) q.push({i, 0});
    while (!q.empty()) {  // 广搜遍历
      auto item = q.front();
      q.pop();
      auto last = insertSAM(item.second, item.first);
      for (int i = 0; i < 26; ++i)
        if (next[last][i]) q.push({i, last});
    }
  }
} exSam;

char s[1000100];

int main() {
  int n;
  cin >> n;
  exSam.init();
  for (int i = 0; i < n; ++i) {
    cin >> s;
    int len = strlen(s);
    exSam.insert(s, len);
  }
  exSam.build();
  long long ans = 0;
  for (int i = 1; i < exSam.tot; ++i) {
    ans += exSam.len[i] - exSam.len[exSam.link[i]];
  }
  cout << ans << endl;
}
```

Manacher

```
a[0] = '~', a[++cnt] = '|';
while((c = getchar()) >= 'a' && c <= 'z') a[++cnt] = c, a[++cnt] = '|';
for (int i = 0, c = 0, r = 0; i <= cnt; i++) {
  if (i <= r) f[i] = min(r - i + 1, f[c * 2 - i]);
  while (a[i - f[i]] == a[i + f[i]]) f[i]++;
  if (i + f[i] > r) r = i + f[i] - 1, c = i;
  if (f[i] > ans) ans = f[i];
}
printf("%d\n", ans - 1);
```

回文树

定义 s 的一个子串的存在值为这个子串在 s 中出现的次数乘以这个子串的长度。对于给定的字符串 s，求所有回文子串中的最大存在值。

```cpp
#include <bits/stdc++.h>
using namespace std;
const int maxn = 300000 + 5;

namespace pam {
int sz, tot, last;
int cnt[maxn], ch[maxn][26], len[maxn], fail[maxn];
char s[maxn];

int node(int l) {  // 建立一个新节点，长度为 l
  sz++;
  memset(ch[sz], 0, sizeof(ch[sz]));
  len[sz] = l;
  fail[sz] = cnt[sz] = 0;
  return sz;
}

void clear() {  // 初始化
  sz = -1;
  last = 0;
  s[tot = 0] = '$';
  node(0);
  node(-1);
  fail[0] = 1;
}

int getfail(int x) {  // 找后缀回文
  while (s[tot - len[x] - 1] != s[tot]) x = fail[x];
  return x;
}

void insert(char c) {  // 建树
  s[++tot] = c;
  int now = getfail(last);
  if (!ch[now][c - 'a']) {
    int x = node(len[now] + 2);
    fail[x] = ch[getfail(fail[now])][c - 'a'];
    ch[now][c - 'a'] = x;
  }
  last = ch[now][c - 'a'];
  cnt[last]++;
}

long long solve() {
  long long ans = 0;
  for (int i = sz; i >= 0; i--) {
    cnt[fail[i]] += cnt[i];
  }
  for (int i = 1; i <= sz; i++) {  // 更新答案
    ans = max(ans, 1ll * len[i] * cnt[i]);
```

```
  }
  return ans;
}
}  // namespace pam

char s[maxn];

int main() {
  pam::clear();
  scanf("%s", s + 1);
  for (int i = 1; s[i]; i++) {
    pam::insert(s[i]);
  }
  printf("%lld\n", pam::solve());
  return 0;
}
```

序列自动机

给你两个由小写英文字母组成的串　和　，求：

1. 　的一个最短的子串，它不是　 的子串；
2. 　的一个最短的子串，它不是　 的子序列；
3. 　的一个最短的子序列，它不是　 的子串；
4. 　的一个最短的子序列，它不是　 的子序列。

。

```cpp
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <iostream>

using namespace std;

const int N = 2005;

char s[N], t[N];
int na[N][26], nb[N][26], nxt[26];
int n, m, a[N], b[N], tot = 1, p = 1, f[N][N << 1];

struct SAM {
  int par, ch[26], len;
} sam[N << 1];

void insert(int x) {
  int np = ++tot;  // 新节点
  sam[np].len = sam[p].len + 1;
  while (p && !sam[p].ch[x]) {
    sam[p].ch[x] = np;
    p = sam[p].par;
  }
  if (p == 0)
    sam[np].par = 1;
  else {
    int q = sam[p].ch[x];
    if (sam[q].len == sam[p].len + 1)
      sam[np].par = q;
    else {
      int nq = ++tot;
      sam[nq].len = sam[p].len + 1;
      memcpy(sam[nq].ch, sam[q].ch, sizeof(sam[q].ch));
      sam[nq].par = sam[q].par;
      sam[q].par = sam[np].par = nq;
      while (p && sam[p].ch[x] == q) {
        sam[p].ch[x] = nq;
        p = sam[p].par;
      }
    }
  }
  p = np;
}

int main() {
  scanf("%s%s", s + 1, t + 1);

  n = strlen(s + 1);
  m = strlen(t + 1);
```

```
for (int i = 1; i <= n; ++i) a[i] = s[i] - 'a';
for (int i = 1; i <= m; ++i) b[i] = t[i] - 'a';

for (int i = 1; i <= m; ++i) insert(b[i]);

// nxt[S[i]]<-i
for (int i = 0; i < 26; ++i) nxt[i] = n + 1;
for (int i = n; i >= 0; --i) {
  memcpy(na[i], nxt, sizeof(nxt));
  nxt[a[i]] = i;
}

for (int i = 0; i < 26; ++i) nxt[i] = m + 1;
for (int i = m; i >= 0; --i) {
  memcpy(nb[i], nxt, sizeof(nxt));
  nxt[b[i]] = i;
}

// 四种情况计算答案
//  1
int ans = N;
for (int l = 1; l <= n; ++l) {
  for (int r = l, u = 1; r <= n; ++r) {
    u = sam[u].ch[a[r]];
    if (!u) {
      ans = min(ans, r - l + 1);
      break;
    }
  }
}

printf("%d\n", ans == N ? -1 : ans);

// 2
ans = N;

for (int l = 1; l <= n; ++l) {
  for (int r = l, u = 0; r <= n; ++r) {
    u = nb[u][a[r]];
    if (u == m + 1) {
      ans = min(ans, r - l + 1);
      break;
    }
  }
}

printf("%d\n", ans == N ? -1 : ans);

// 3
for (int i = n; i >= 0; --i) {
```

```
    for (int j = 1; j <= tot; ++j) {
      f[i][j] = N;
      for (int c = 0; c < 26; ++c) {
        int u = na[i][c];
        int v = sam[j].ch[c];
        if (u <= n) f[i][j] = min(f[i][j], f[u][v] + 1);
      }
    }
  }

  printf("%d\n", f[0][1] == N ? -1 : f[0][1]);

  // 4
  memset(f, 0, sizeof(f));

  for (int i = n; i >= 0; --i) {
    for (int j = 0; j <= m; ++j) {
      f[i][j] = N;
      for (int c = 0; c < 26; ++c) {
        int u = na[i][c];
        int v = nb[j][c];
        if (u <= n) f[i][j] = min(f[i][j], f[u][v] + 1);
      }
    }
  }

  printf("%d\n", f[0][0] == N ? -1 : f[0][0]);

  return 0;
}
```