定义

栈和队列是操作受限的线性表，而串是内容受限的线性表
串是由零个或多个字符组成的有限序列
$a_i(1 \leq 1 \leq n)$：字符（字母，数字或其他字符）

- 空串：用$\emptyset$表示
- 子串：一个串中任意个连续字符组成的子序列（含空串）
  - 真子串：不包含自身的所有子串
  - 空串是任意串的子串，任意串s都是其自身的子串
- 主串：包含子串的串
- 字符位置：一个字符在序列中的序号
- 子串位置：子串的第一个字符在主串的位置
- 空格串：由一个或多个空格组成的串，与空串不同
- 串相等：当且仅当两个串的长度相等，并且各对应位置的字符都相等

ADT String {
数据对象：$D = \{a_i | a_i \in CharacterSet, i = 1, 2, \cdots, n\}$
数据关系：$R1 = \{< a_{i-1}, a_i > | a_{i-1}, a_i \in D\}$
基本操作：
StrAssign(&T, chars); StrCopy(&T, S)
StrEmpty(S); StrCompare(S, T);
StrLength(S); ClearString(&S);
Concat(&T, S1, S2); SubString(&Sub, S, pos, len);
Index(S, T, pos); Replace(&S, T, V);
StrInsert(&S, pos, T); StrDelete(&S, pos, len);
DestoyString(&S)
} ADT String

寻找子串索引

FxMarkBrown's Blog

```C++
int Index(String S, String T, int pos) {
    if(pos <= 0) return 0;

    n = StrLength(S); m = StrLength(T);
    i = pos;
    while(i <= n-m+1) { // n-m+1时此时T的结尾与S的结尾重合，i指向T的第一个值
        SubString(sub, s, i, m);
        if(StrCompare(sub, T) != 0) ++i;
        else return i;
    }

    return 0;
}
```

顺序定长表示

定义

```C++
#define MAXSTRLEN 255
typedef unsigned char SString[MAXSTRLEN+1]
```

基本操作

连接

```C++
bool Concat(SString &T, SString S1, SString S2) {
    if(S1[0] + S2[0] <= MAXSTRLEN) { // 假设第一位存长度
        T[1..S[0]] = S1[1..S1[0]];
        T[S1[0] + 1..S1[0] + S2[0]] = S2[1..S2[0]];
        T[0] = S1[0] + S2[0];
        uncut = true;
    } else if(S1[0] < MAXSTRLEN) {
        T[1..S1[0]] = S1[1..S1[0]];
        T[S1[0]+1..MAXSTRLEN] = S2[1..MAXSTRLEN-S1[0]];
        T[0] = MAXSTRLEN;
        uncut = false;
    } else {
        T[1..MAXSTRLEN] = S1[1..MAXSTRLEN];
        T[0] = MAXSTRLEN;
        uncut = false;
    }
    return uncut;
```

FxMarkBrown's Blog

求子串

```C++
Status SubString(SString &Sub, SString S, int pos, int len) {
    if(pos < 1 || pos > S[0] || len < 0 || pos + len > S[0] - 1) return ERROR;

    Sub[1..len] = S[pos..pos+len-1];
    Sub[0] = len;
    return OK;
}
```

堆分配表示

定义

```C++
typedef struct {
    char* ch;
    int length;
} HString;
```

基本操作

插入

```C++
Status StrInsert(HString &S, int pos, HString T) {
    if(pos < 1 || pos > S.length + 1) return ERROR;
    if(T.length) { // T内有字符
        S.ch = (char*)realloc(
            S.ch,
            (S.length + T.length) * sizeof(char)
        )
        if(!S.ch) exit(OVERFLOW);

        for(i = S.lenth - 1;i >= pos-1;--i) { // 将原来的串后面的部分右移，此时i是索引而不是位置
            S.ch[i+T.length] = S.ch[i]

        S.ch[pos-1..(pos-1)+T.length-1] = T[0..T.length-1] // []内是索引而不是位置
        S.length += T.length;
    }
    return OK;
}
```

FxMarkBrown's Blog

串的模式匹配算法

子串的位置

使用顺序定长表示
用两个位置索引i,j遍历S和T，i开始为pos

- 如果S[i] == T[j]：++i;++j
  - 如果匹配成功，此时j > T[0]（走完了），此时i指向匹配完毕后的下一个位置，返回i - T[0]即为pos
  - 否则返回0
- 如果S[i] != T[j]：j = 1（归位），i = i - j + 1 + 1 （i的位置变到pos的下一个位置，+1返回pos，再+1到pos的下一个位置）

```C++
int Index(SString S, SString T, int pos) {
  i = pos; j = 1;

  while(i <= S[0] && j <= T[0]) {
    if(S[i] == T[j]) {
      ++i;++j;
    }
    else {
        i = i - j +2; j = 1;
    }
  }

  if(j > T[0]) return i - T[0];
  else return 0;
}
```

分析其复杂度，最好情况下(1)，最坏情况下前$n-m$位都匹配m次，最后m位匹配m次，假设$(n-m)m = nm - m^2$，若m远小于n，则为$O(n\ m)$

KMP

考虑当主串S和模式串P，当$S_i\ \ P_j$时有$p_1 \cdots p_{j-1} = s_{i-j+1} \cdots s_{i-1}$，即前面都是匹配的
如果又有条件$p_1, \cdots p_{k-1} = s_{i-k+1} \cdots s_{i-1}$，即模式字符串中前某一部分和主串中的不匹配位置之前的某一部分相等
则会得到$p_1 \cdots p_{k-1} = p_{j-1-k} \cdots p_{j-1}$，即模式串中有重复的部分

若模式串中存在满足上式的两个子串，则当匹配过程中S[i] != P[j]时，仅需将模式串向右滑

FxMarkBrown's Blog

Next数组的计算方法

这里的**next**数组记录的是回朔的位置，因为这里的字符串字符部分索引从**1**开始，所以没有公共前后缀时不存**0**而存**0+1=1**

1. 朴素计算方法
    1. 第一个字符对应$next0 = 0$
    2. 从第二个字符开始，计算前方子串的最长公共前后缀长度 **+1**
2. 已知前方子串计算后方字符的next值
    1. 通过前一个字符C1的next值找到左侧子串的最后一个字符C2位置
    2. 将C1与C2比较
        1. 相等则该字符的next值为C1的next值+1
        2. 不相等，则再回朔C1，直到到达第一个字符，若相等则为$j+1$，若还不相等，则填 **0+1=1**

FxMarkBrown's Blog