严锦 1700011049

4.1

```
int findAB(string A, string B) {
    int b = 0, a = -1;
    int aLen = A.length();
    int bLen = B.length();
    int *next = new int[bLen + 1];
    memset(next, 0, bLen * sizeof(int));
    next[0] = -1;
    while (b < bLen) {
        if (a == -1 || A[a] == B[b]) {
            next[++b] = ++a;
        } else
            a = next[a];
    }
    return next[bLen];
}
```

在该算法中，while循环里的b变量最多只会增加bLen次，而a变量增加的次数不会超过b变量增加的次数，因此a、b变量总增加次数不会超过2bLen。

设B串长度为m，则算法复杂度为$\mathcal{O}(m)$.

4.2

$next$数组的定义如下：

$$next[j] = \begin{cases} -1, & j = 0 \\ \max\{k : 0 < k < j \wedge P[0...k-1] = P[j-k...j-1]\}, & \text{首尾配串最长为k} \\ 0, & \text{其他情况} \end{cases}$$

设模式串为$p_0 \ldots p_j$，假设已经计算出了$next[0 \ldots j]$，现在需要计算$next[j+1]$.

设$next[j] = k$，则此时有$p_0 \ldots p_{k-1} == p_{j-k} \ldots p_{j-1}$

非优化版算法

1. 若$k = -1$，则说明目前$j = 0$，属于“其他情况”，因此$next[j+1] = 0$.
2. 若$p_k = p_j$，则显然有$p_0 \ldots p_{k-1}p_k = p_{j-k} \ldots p_{j-1}p_j$，由定义得，$next[j+1] = k+1$.

3. 若$p_k \neq p_j$，则$\exists k' < k$, $p_0p_1p_2\dots p_{k'-1} = p_{j-k'}p_{j-k'+1}\dots p_{j-1}$，又由于$p_{j-k'}p_{j-k'+1}\dots p_{j-1}$是$p_0\dots p_{j-1}$长度为$k'-1$的尾串，$p_{k-k'}p_{k-k'+1}\dots p_{k-1}$是$p_0\dots p_{k-1}$长度为$k'-1$的尾串，因此有$p_{j-k'}p_{j-k'+1}\dots p_{j-1} = p_{k-k'}p_{k-k'+1}\dots p_{k-1}$.因此有$p_0p_1p_2\dots p_{k'-1} = p_{k-k'}p_{k-k'+1}\dots p_{k-1}$，由定义知，$k' = next[k]$.

   若$p_{k'} \neq p_j$，同理，此时$\exists k'' < k'$, $p_0p_1p_2\dots p_{k''-1} = p_{j-k''}p_{j-k''+1}\dots p_{j-1} \wedge k'' = next[k']$.

   重复此操作，若出现$p_{k^{(n)}} = p_j$，则$next[j+1] = k^{(n)} + 1$，否则一定有$k^{(n)} = -1$，则此时有$next[j+1] = 0$.

因此算法正确.

优化版算法

若$P[j]$ 与$T[i]$失配，同时$P[j] = P[N^{(n)}[j]]$ ，则$P[N^{(n)}[j]]$一定与$T[i]$ 失配，因此若$P[j] = P[N^{(n)}[j]]$,则该比较一定失配，无需考虑。

因此算法正确