Let S and T be two strings of lengths n and m, respectively. The symbols in the strings are σ lower-case alphabetic letters $a, b, c, \ldots$. For example, if $\sigma = 5$, then the symbols are a, b, c, d, e.

For $k \geqslant 0$, the string T_k is generated by repeating each symbol of T exactly k times. For example, if $T = bccab$, then $T_3 = bbbccccccaaabbb$. We take T_0 to be the empty string (which is a subsequence of any string). Your task is to find the largest k such that T_k is a subsequence (not necessarily a substring) of S. Assume that $n \geqslant m$. Clearly, we have $0 \leqslant k \leqslant \lfloor n/m \rfloor$.

Part 1: Helping Functions

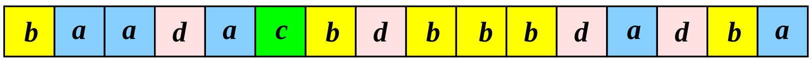
Write a function $issubseq(A, B)$ that returns the decision whether B is a subsequence of A. Write another function $repsymbols(T, k)$ that returns the k-fold repetition T_k of k. Finally, write a function $prnsubseq(A, B)$ that, given a subsequence B of A, prints the occurrence of B in A in the format specified in the sample output.

Part 2: Exhaustive Search

Since $0 \leqslant k \leqslant \lfloor n/m \rfloor$ and T_0 is always a subsequence of S, keep on generating $T_1, T_2, T_3, \ldots$ as long as these strings are subsequences of S. Return the largest k. Implement this strategy in a function $exhs(S, T)$. The running time of this function is evidently $O(n^2/m)$.

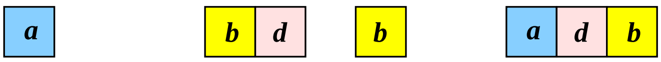
Part 3: Divide-and-Conquer Strategy 1

Write a function $dnc1(S, T, \sigma)$ which works as follows. If $n < 2m$, the function checks whether T is a subsequence of S, and returns 1 or 0 accordingly. So suppose that $n \geqslant 2m$. The function first breaks S into two subsequences S_1 and S_2 by taking alternate occurrences of *each* symbol from S. This procedure is demonstrated in the following figure. For $S = baadacbdbbbdadba$, the two subsequences are $S_1 = badacbbda$ and $S_2 = abdbadb$. These are roughly of size $n/2$ each.



The original array

The first subsequence



The second subsequence

Two recursive calls are made on S_1 and S_2. Let the return values be k_1 and k_2. The maximum k for S is related to the two return values in the following manner.

> **Case 1:** $k = 2l$ **is even**
>
> In this case, we must have $k_1 = k_2 = l$. To see why, first note that each of S_1 and S_2 contains T_l as a subsequence. If one of them, say S_1, contains T_{l+1} as a subsequence, then for each symbol α in T, the $l+1$ occurrences of α from T_{l+1} have, in S, l intermediate occurrences of α that were sent to S_2. Therefore S contains T_{2l+1} as a subsequence, a contradiction.
>
> **Case 2:** $k = 2l + 1$ **is odd**
>
> In this case, $k_1, k_2 \in \{l, l+1\}$. For the proof, note that each of S_1 and S_2 must contain T_l and may contain T_{l+1}, but neither contains T_{l+2}. This can be proved as in Case 1. All the four cases are possible. This is demonstrated now (for $k = 1$ and $l = 0$). We take $T = abcd$ in all the cases.

$S = abdcbacd$, so $S_1 = abdc$ and $S_2 = bacd$, which implies that $k_1 = k_2 = 0$.
$S = dcbaabcd$, so $S_1 = dcba$ and $S_2 = abcd$, which implies that $k_1 = 0$, $k_2 = 1$.
$S = abcddcba$, so $S_1 = abcd$ and $S_2 = dcba$, which implies that $k_1 = 1$, $k_2 = 0$.
$S = abcdabcd$, so $S_1 = abcd$ and $S_2 = abcd$, which implies that $k_1 = k_2 = 1$.

From these observations, it follows that $k \in \{k_1 + k_2 - 1, k_1 + k_2, k_1 + k_2 + 1\}$. Which one is the correct value of k can be verified by exhaustive search (over three possibilities only). The running time of this divide-and-conquer algorithm is $O(n \log n)$ (actually, $O(n \log(n/m))$).

Part 4: Divide-and-Conquer Strategy 2

The divide-and-conquer algorithm of Part 3 can be readily modified to an $O(n)$-time algorithm. Implement this linear-time algorithm in a function $dnc2(S, T, \sigma)$.

The *main*() function

- Read the number σ of symbols (this is s in the sample output), the sizes n and m, and the strings S and T from the user. Instead of reading n and m, you can compute these lengths by calling *strlen* on S and T. Note that σ is needed in Parts 3 and 4.
- Call $exhs(S, T)$, and print the value returned.
- Call $dnc1(S, T, \sigma)$, and print the value returned.
- Call $dnc2(S, T, \sigma)$, and print the value returned.
- Print the match of T_k (where k is any of the three return values) in S by calling *repsymbols* and *prnsubseq*.

Sample output

```
s = 3
n = 80
m = 4

S = bbcbbbcacbacbacbacabccaaabcccccbabacbcaabbacbabcccbcccbcbbcbabacabbcbcccaaccbabb
T = cacb

+++ Exhaustive search
    k = 8

+++ Divide and Conquer Strategy 1
    k = 8

+++ Divide and Conquer Strategy 2
    k = 8

+++ The subsequence is:
    bbcbbbcacbacbacbacabccaaabcccccbabacbcaabbacbabcccbcccbcbbcbabacabbcbcccaaccbabb
      c   c c  c  c  c  ccaaa       a a   aa  ac   ccc ccc cbb b b   bb b       b
```

Submit a single C/C++ source file. Do not use global/static variables.