

Problem Register for “Bioinformatics Algorithms: An Active Learning Approach”

Parker Côté and Ryan Eveloff

June 2021



Preface

After finishing the “Bioinformatics Algorithms” course at the University of California, San Diego in early 2021, we came to the conclusion that a centralized, syntactically consistent register of all problems, including reproducible test cases, would have been an invaluable resource to us and our classmates. Namely, we often found ourselves stuck trying to find an edge case or confused about the parameters of a problem, but no resource existed to satisfy that need. With this problem register, we aim to fill that need. Our immense thanks goes out to Dr. Pevzner and Dr. Compeau for giving us the opportunity to pursue this register and for offering creative directions, as well as Andrey Bzikadze and Vikram Sirupurapu for assisting in the brainstorming and proofreading processes. We hope that you can utilize this register as a resource to enrich your own learning and that of those around you.

Best,
Parker & Ryan

Prologue

Things to know before solving Programming Challenges

Unless specified otherwise, the following are *soft* rules and will not count for grading, they simply lay out the format for *our* test case output.

- Space separation for list elements
- Inputs and outputs are case sensitive
- When outputting integers, do not include the floating point (E.g. 1 instead of 1.0)
- Lexicographical ordering

Unless specified otherwise, the following are *hard* rules and *will* count for grading.

- Zero-indexing

Good	Bad
$[0, 1, 2, 3][0] = 0$	$[0, 1, 2, 3][1] = 0$

- Space separation for list elements

Good	Bad
0 1 2 3	0, 1, 2, 3

- Newline separation for arguments

Good	Bad
1	1, 2
2	

Test Cases

While we provide a range of test cases for each problems, these are *not* exhaustive. We encourage you to investigate each problem to determine the potential edge cases and relevant parameters, then write your own tests as you see fit. Similarly, many of our test cases are based on the DNA alphabet (A, C, G, and T). When designing your algorithms, unless explicitly stated otherwise, you should be prepared to test on all characters, not just the characters in the genetic alphabet. Please note that the constraints provided for each question will not change and you will not be tested on datasets outside of those parameters. You should consider the effects of these constraints on runtime and memory when designing your algorithmic solutions.

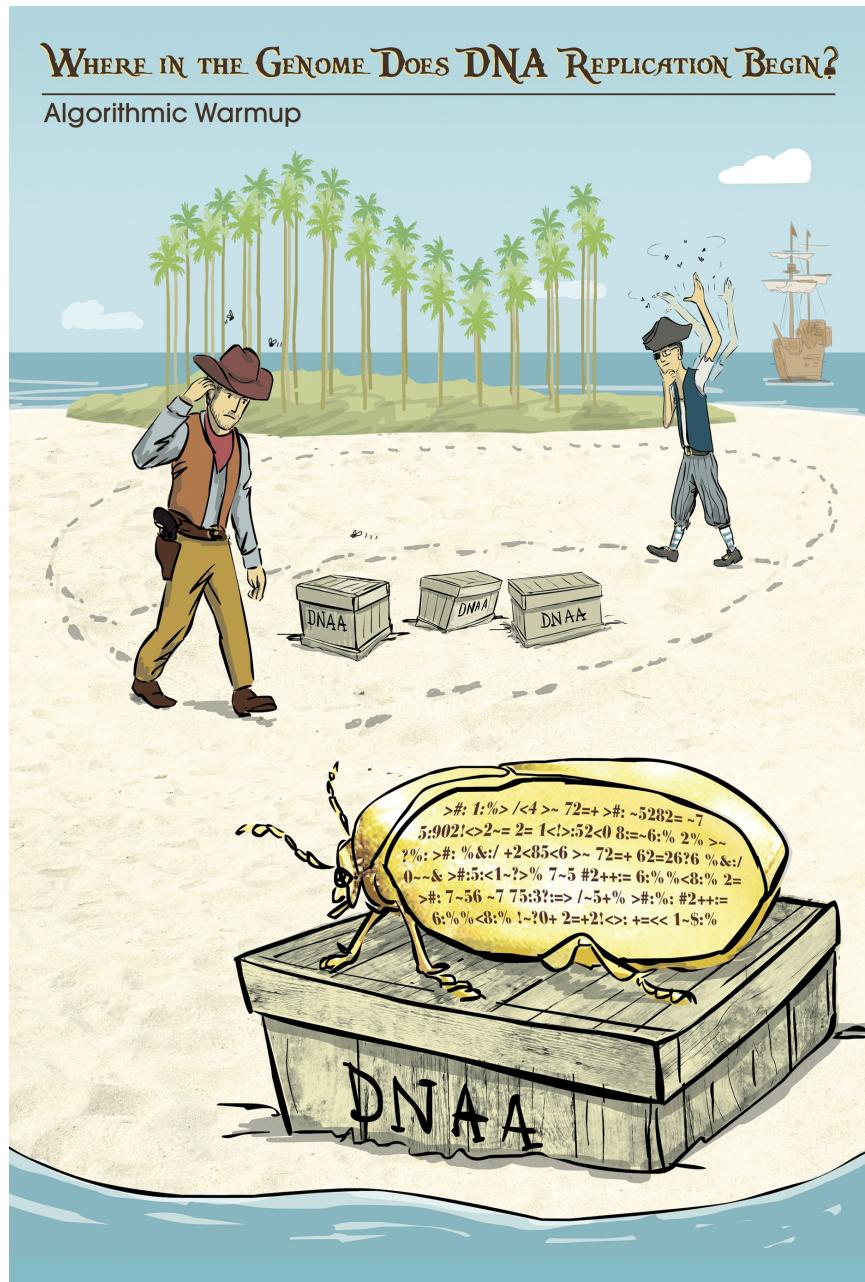
Contents

1 Where in the Genome Does DNA Replication Begin?	
<i>Algorithmic Warmup</i>	6
1A Compute the Number of Times a Pattern Appears in a Text	7
1B Find the Most Frequent Words in a String	10
1C Find the Reverse Complement of a String	13
1D Find All Occurrences of a Pattern in a String	16
1E Find Patterns Forming Clumps in a String	19
1F Find a Position in a Genome Minimizing the Skew	20
1G Compute the Hamming Distance Between Two Strings	21
1H Find All Approximate Occurrences of a Pattern in a String	24
1I Find the Most Frequent Words with Mismatches in a String	27
1J Find Frequent Words with Mismatches and Reverse Complements	30
1K Generate the Frequency Array of a String	33
1L Implement PatternToNumber	34
1M Implement NumberToPattern	37
1N Generate the d -Neighborhood of a String	40
2 Which DNA Patterns Play the Role of Molecular Clocks?	
<i>Randomized Algorithms</i>	43
2A Implement MotifEnumeration	44
2B Find a Median String	45
2C Find a Profile-most Probable k -mer in a String	46
2D Implement GreedyMotifSearch	47
2E Implement GreedyMotifSearch with Pseudocounts	48
2F Implement RandomizedMotifSearch	49
2G Implement GibbsSampler	50
2H Implement DistanceBetweenPatternAndStrings	51
9 How Do We Locate Disease-Causing Mutations?	
<i>Combinatorial Pattern Matching</i>	52
9A Construct a Trie from a Collection of Patterns	53
9B Implement TrieMatching	57
9C Construct the Suffix Tree of a String	60
9D Find the Longest Repeat in a String	64
9E Find the Longest Substring Shared by Two Strings	67
9F Find the Shortest Non-Shared Substring of Two Strings	70
9G Construct the Suffix Array of a String	73
9H Pattern Matching with the Suffix Array	76
9I Construct the Burrows-Wheeler Transform of a String	80
9J Reconstruct a String from its Burrows-Wheeler Transform	83
9K Generate the Last-to-First Mapping of a String	86
9L Implement BWMatching	89
9M Implement BetterBWMatching	90
9N Find All Occurrences of a Collection of Patterns in a String	91

9O	Find All Approximate Occurrences of a Collection of Patterns in a String	92
9P	Implement TreeColoring	95
9Q	Construct the Partial Suffix Array of a String	97
9R	Construct a Suffix Tree from a Suffix Array	101

1 Where in the Genome Does DNA Replication Begin?

Algorithmic Warmup



1A Compute the Number of Times a Pattern Appears in a Text

Pattern Count Problem

Implement PatternCount.

Input: Strings *Text* and *Pattern*.

Output: $\text{Count}(\text{Text}, \text{Pattern})$.

AGAGATCAGA
AGAGA AGA
1 2 3

Formatting

Input: Newline-separated strings *Text* and *Pattern*.

Output: An integer representing the number of times *Pattern* appears in *Text*.

Constraints

- The length of *Text* will be between 1 and 10^4 .
- The length of *Pattern* will be between 1 and 10^1 .
- *Text* and *Pattern* will be DNA strings.

Test Cases

Case 1

Description: A small and hand-solvable dataset taken from the example problem on Stepik.

Input:

GCGCG

GCG

Output:

2

Figure:

c1/figures/1A.png

TODO

Case 2

Description: There are no matches in *Text* to *Pattern*.

Input:

ATATATATAT
GT

Output:

0

Case 3

Description: *Text* contains overlapping occurrences of *Pattern*.

Input:

bananas
ana

Output:

2

Case 4

Description: Large regions of *Text* being a single character or short tandem repeat (STR).

Input:

AAACAA
AA

Output:

3

Case 5

Description: *Text* is palindromic or has substrings that are palindromic.

Input:

GAGCAT
GA

Output:

1

1B Find the Most Frequent Words in a String

Frequent Words Problem

Find the most frequent words k -mers in a string.

Input: A DNA string $Text$ and an integer k .

Output: All most frequent k -mers in $Text$ (in any order).

AGAGACGTGAGAG
AGAGA AGA
GAG GAGAG

Formatting

Input: A DNA string $Text$ followed by an integer k .

Output: All most frequent k -mers in $Text$ (in any order).

Constraints

- The length of $Text$ will be between 1 and 10^4 .
- The integer k will be between 1 and 10^2 .
- $Text$ will be a DNA string.

Test Cases

Case 1

Description: A small and hand-solvable dataset taken from the example problem on Stepik.

Input:

ACGTTGCATGTCGCATGATGCATGAGAGCT

4

Output:

CATG GCAT

Figure:

ACGTTGCATGTCGCATGATGCATGAGAGCT
CATG CATG CATG
GCAT GCAT GCAT

CATG and GCAT are the most common 4-mers in *Text*, both of them appear 3 times.

Case 2

Description: There is only one most-frequent kmer.

Input:

ATATAT
2

Output:

AT

Case 3

Description: *Text* contains overlapping k-mers.

Input:

bananas
3

Output:

ana

Case 4

Description: Large regions of *Text* being a single character or short tandem repeat (STR).

Input:

AAACAA
2

Output:

AA

Case 5

Description: *Text* has a large number of most frequent patterns.

Input:

GAGCAT
2

Output:

GA AG GC CA AT

1C Find the Reverse Complement of a String

Reverse Complement Problem

Find the reverse Complement of a DNA string.

Input: A DNA string *Pattern*.

Output: $\overline{Pattern}$, the reverse complement of the string *Pattern*.

GGTCATCGACA
TGTGATGACC

Formatting

Input: A DNA string *Pattern*.

Output: A string representing $\overline{Pattern}$, the reverse complement of *Pattern*.

Constraints

- The length of *Pattern* will be between 1 and 10^4 .
- *Pattern* will be a DNA string.

Test Cases

Case 1

Description: A small and hand-solvable dataset taken from the example problem on Stepik.

Input:

AAAACCCGGT

Output:

ACCGGGTTTT

Figure:

AAAACCCGGT $\xrightarrow{\text{Reverse}}$ TGGCCCAAAA $\xrightarrow{\text{Complement}}$ ACCGGGTTTT

The reverse complement of AAAACCCGGT is ACCGGGTTTT.

Case 2

Description: *Pattern* is only one character long.

Input:

A

Output:

T

Case 3

Description: *Pattern* is equal to its reverse complement.

Input:

CATG

Output:

CATG

Case 4

Description: *Pattern* is palindromic.

Input:

AGGA

Output:

TCCT

Case 5

Description: *Pattern* contains repeats.

Input:

AGAGCATGAG

Output:

CTCATGCTCT

1D Find All Occurrences of a Pattern in a String

Pattern Matching Problem

Find all occurrences of a Pattern in a string.

Input: DNA strings *Pattern* and *Genome*.

Output: All starting positions in *Genome* where *Pattern* appears as a substring.

CGACTAGTTT
ACT
2

Formatting

Input: DNA strings *Pattern* and *Genome*.

Output: A space-separated list of integers representing each starting position in *Genome* where *Pattern* appears as a substring.

Constraints

- The length of *Pattern* will be between 1 and 10^1 .
- The length of *Genome* will be between 1 and 10^4 .
- *Pattern* and *Genome* will be DNA strings.

Test Cases

Case 1

Description: A small and hand-solvable dataset taken from the example problem on Stepik.

Input:

GATATATGCATATACTT
ATAT

Output:

1 3 9

Figure:

GATATATGCATATACTT
ATATAT ATAT
1 3 9

By sliding our *Pattern* ATAT along our *Genome* GATATATGCATATACTT, we find that it occurs three times at the indices 1, 3, and 9.

Case 2

Description: There are no matches in *Text* to any pattern in *Patterns*.

Input:

ATATATATAT

GT

Output:

Case 3

Description: *Text* contains overlapping occurrences of *Patterns*.

Input:

bananas

ana

Output:

1 3

Case 4

Description: Large regions of *Text* being a single character or short tandem repeat (STR).

Input:

AAACAA

AA

Output:

0 1 4

Case 5

Description: *Text* is palindromic or has substrings that are palindromic.

Input:

GAGCAT

GA

Output:

0

1E Find Patterns Forming Clumps in a String

Clump Finding Problem

Find patterns forming clumps in a string.

Input: A DNA string *Genome*, and integers k , L , and t .

Output: All distinct k -mers forming (L, t) -clumps in *Genome*.

 c1/logos/1E.png

Formatting

Input: A DNA string *Genome* followed by space-separated integers k , L , and t .

Output: A space-separated list of strings containing all distinct k -mers forming (L, t) -clumps in *Genome*.

Constraints

- The length of *Genome* will be between 1 and 10^4 .
- The integer k will be between 1 and 10^1 .
- The integer L will be between 1 and 10^3 .
- The integer t will be between 1 and 10^2 .
- *Genome* will be a DNA string.

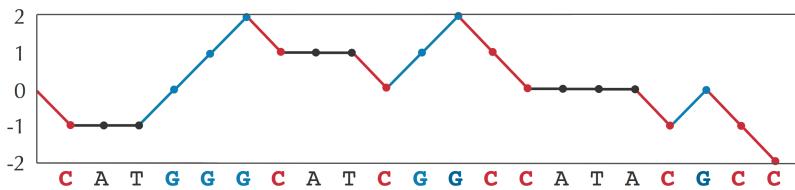
1F Find a Position in a Genome Minimizing the Skew

Minimum Skew Problem

Find a position in a genome minimizing the skew.

Input: A DNA string $Genome$.

Output: All integers i minimizing $Skew(Prefix_i(Genome))$ over all values of i (from 0 to $|Genome|$).



Formatting

Input: A DNA string $Genome$.

Output: A space-separated list of integers i minimizing $Skew(Prefix_i(Genome))$ over all values of i (from 0 to $|Genome|$).

Constraints

- The length of $Genome$ will be between 1 and 10^5 .
- $Genome$ will be a DNA string.

1G Compute the Hamming Distance Between Two Strings

Hamming Distance Problem

Compute the Hamming distance between two strings.

Input: Two strings of equal length.

Output: The Hamming distance between these strings.

T	C	T	G	A	C
T	C	C	G	A	C
1		2			

Formatting

Input: Two DNA strings $Text_1$ and $Text_2$.

Output: An integer representing the Hamming distance between $Text_1$ and $Text_2$.

Constraints

- The length of $Text_1$ and $Text_2$ will be between 1 and 10^4 .
- $Text_1$ and $Text_2$ will have equal lengths.
- $Text_1$ and $Text_2$ will be DNA strings.

Test Cases

Case 1

Description: A small and hand-solvable dataset taken from the example problem on Stepik.

Input:

panama
banana

Output:

2

Figure:

The diagram shows two strings, "panama" and "banana", aligned vertically. The first string has its second character, 'a', colored red. The second string has its second character, 'n', colored red. Below the strings, the number "1" is positioned under the first character of the first string, and the number "2" is positioned under the second character of the second string.

panama
banana
1 2

When lining up our two input strings panama and banana, there are only two places where these strings have different characters, so the hamming distance is 2.

Case 2

Description: $Text_1$ and $Text_2$ are the same strings.

Input:

ACGTACGT

ACGTACGT

Output:

0

Case 3

Description: $Text_1$ and $Text_2$ do not have any characters in common.

Input:

AAACCC

GTGTGT

Output:

6

1H Find All Approximate Occurrences of a Pattern in a String

Approximate Pattern Matching Problem

Find all approximate occurrences of a pattern in a string.

Input: DNA strings *Pattern* and *Text* along with an integer d .

Output: All starting positions where *Pattern* appears as a substring of *Text* with at most d mismatches.

CGACTAGTTT
CGACGA
0 3

Formatting

Input: DNA strings *Pattern* and *Text* along with an integer d .

Output: A space-separated list of integers containing all starting positions where *Pattern* appears as a substring of *Text* with at most d mismatches.

Constraints

- The length of *Pattern* will be between 1 and 10^2 .
- The length of *Text* will be between 1 and 10^5 .
- The integer d will be between 1 and 10^1 .
- *Pattern* and *Text* will be DNA strings.

Test Cases

Case 1

Description: A small and hand-solvable dataset taken from the example problem on Stepik.

Input:

ATTCTGGA
CGCCCGAATCCAGAACGCAATTCCATATTCGGGACCACTGGCCTCCACGGTACGGACGTCAATCAAAT
3

Output:

6 7 26 27

Figure:

c1/figures/1H.png

TODO

Case 2

Description: *Patterns* contains partial and complete matches.

Input:

ACGT
GG
1

Output:

1 2

Case 3

Description: *Patterns* contains no matches.

Input:

GGGGG
TT
1

Output:

Case 4

Description: *Patterns* contains no matches, but has exactly d mismatches.

Input:

GG
TT
2

Output:

0

1I Find the Most Frequent Words with Mismatches in a String

Frequent Words with Mismatches Problem

Find the most frequent k -mers with mismatches in a string.

Input: A DNA string $Text$ as well as integers k and d .

Output: All most frequent k -mers with up to d mismatches in $Text$.

CGACTAGTTT
ATT ATT
ATT

Formatting

Input: A DNA string $Text$ as well as integers k and d .

Output: A space-separated list of strings representing all most frequent k -mers with up to d mismatches in $Text$.

Constraints

- The length of $Text$ will be between 1 and 10^3 .
- The integer k will be between 1 and 10^1 .
- The integer d will be between 1 and 10^1 .
- $Text$ will be a DNA string.

Test Cases

Case 1

Description: A small and hand-solvable dataset taken from the example problem on Stepik.

Input:

TODO

Output:

TODO

Figure:

c1/figures/1I.png

TODO

Case 2

Description: *Text* contains partial and complete matches for the most frequent word.

Input:

AGGT
2
1

Output:

GG

Case 3

Description: $d = 0$.

Input:

AGGT
2
0

Output:

GG

Case 4

Description: *Text* has multiple most frequent words.

Input:

AGGC GG
3
0

Output:

AGG GGC GCG CGG

1J Find Frequent Words with Mismatches and Reverse Complements

Frequent Words with Mismatches and Reverse Complements Problem

Find the most frequent k -mers (with mismatches and reverse complements) in a DNA string.

Input: A DNA string $Text$ as well as integers k and d .

Output: All k -mers $Pattern$ maximizing the sum $Count_d(Text, Pattern) + Count_d(Text, \overline{Pattern})$ over all possible k -mers.

CGACTAGTTT
 AAT
 AAT AAT
 AAT
 AAT AAT
AAACTAGTCG

Formatting

Input: A DNA string $Text$ as well as integers k and d .

Output: A space-separated list of strings representing all k -mers $Pattern$ maximizing the sum $Count_d(Text, Pattern) + Count_d(Text, \overline{Pattern})$ over all possible k -mers.

Constraints

- The length of $Text$ will be between 1 and 10^3 .
- The integer k will be between 1 and 10^1 .
- The integer d will be between 1 and 10^1 .
- $Text$ will be a DNA string.

Test Cases

Case 1

Description: A small and hand-solvable dataset taken from the example problem on Stepik.

Input:

TODO

Output:

TODO

Figure:

c1/figures/1J.png

TODO

Case 2

Description: *Text* contains partial and complete matches for the most frequent word.

Input:

AGGT
2
1

Output:

GG CC

Case 3

Description: $d = 0$.

Input:

AGGT
2
0

Output:

GG CC

Case 4

Description: *Text* has multiple most frequent words.

Input:

AGGC GG
3
0

Output:

AGG GGC GCG CGG CCG CGC GCC CCT

1K Generate the Frequency Array of a String

Frequency Array Problem

Generate the frequency array of a DNA string.

Input: A DNA string *Text* and an integer *k*.

Output: The frequency array of *k*-mers in *Text*.

 c1/logos/1K.png

Formatting

Input: A DNA string *Text* and an integer *k*.

Output: A space-separated list of integers representing the frequency array of *k*-mers in *Text*).

Constraints

- The length of *Text* will be between 1 and 10^3 .
- The integer *k* will be between 1 and 10^1 .
- *Text* will be a DNA string.

1L Implement PatternToNumber

PatternToNumber Problem

Convert a DNA string to a number.

Input: A DNA string *Pattern*.

Output: *PatternToNumber(Pattern)*.

$$\text{GAC} \longrightarrow (2^*4^2) + (0^*4^1) + (1^*4^0) \longrightarrow 33$$

Formatting

Input: A DNA string *Pattern*.

Output: An integer representing the output of *PatternToNumber(Pattern)*.

Constraints

- The length of *Pattern* will be between 1 and 10^2 .
- *Pattern* will be a DNA string.

Test Cases

Case 1

Description: A small and hand-solvable dataset taken from the example problem on Stepik.

Input:

TODO

Output:

TODO

Figure:

c1/figures/1L.png

Case 2

Description: *Pattern* is made up of only one character.

Input:

CCC

Output:

21

Case 3

Description: *Pattern* is long, but is 'A'-dense.

Input:

AAAAAAAAAAAG

Output:

2

Case 4

Description: *Pattern* has a length of 1.

Input:

T

Output:

3

1M Implement NumberToPattern

NumberToPattern Problem

Convert a number to its corresponding DNA string.

Input: Integers $index$ and k .

Output: $NumberToPattern(index, k)$.

$$33 \longrightarrow (2*4^2)+(0*4^1)+(1*4^0) \longrightarrow \text{GAC}$$

Formatting

Input: Integers $index$ and k .

Output: A string representing the output of $NumberToPattern(index, k)$.

Constraints

- The integer $index$ will be between 1 and 10^4 .
- The integer k will be between 1 and 10^1 .

Test Cases

Case 1

Description: A small and hand-solvable dataset taken from the example problem on Stepik.

Input:

TODO

Output:

TODO

Figure:

c1/figures/1M.png

Case 2

Description: k is small.

Input:

1
8

Output:

G

Case 3

Description: k codes for an empty string

Input:

0
0

Output:

Case 4

Description: k is large(r)

Input:

4
60

Output:

ATTA

1N Generate the d -Neighborhood of a String

d -Neighborhood Problem

Find all the neighbors of a pattern.

Input: A DNA string $Pattern$ and an integer d .

Output: The collection of strings $Neighbors(Pattern, d)$.

CGA AAA AGC

AGA GGA ACA AGG

TGA ATA AGT

Formatting

Input: A DNA string $Pattern$ and an integer d .

Output: A space-separated list of strings containing all $Neighbors(Pattern, d)$.

Constraints

- The length of $Pattern$ will be between 1 and 10^1 .
- The integer d will be between 1 and 10^1 .
- $Pattern$ will be a DNA string.

Test Cases

Case 1

Description: A small and hand-solvable dataset taken from the example problem on Stepik.

Input:

TODO

Output:

TODO

Figure:

c1/figures/1N.png

Case 2

Description: $d = 0$.

Input:

AGA
0

Output:

Case 3

Description: *Pattern* is made up of one character.

Input:

AAA
1

Output:

AAC AAG AAT ACA AGA ATA CAA GAA TAA

Case 4

Description: *Pattern* has a length of 1.

Input:

A
1

Output:

C G T

2 Which DNA Patterns Play the Role of Molecular Clocks?

Randomized Algorithms



2A Implement MotifEnumeration

Implanted Motif Problem

Find all (k, d) -motifs in a collection of strings.

Input: A collection of strings Dna , and integers k and d .

Output: All (k, d) -motifs in Dna .

 c2/logos/2A.png

Formatting

Input: Space-separated integers k and d , followed by a newline-separated collection of strings Dna .

Output: A space-separated list of integers containing all starting positions where $Pattern$ appears as a substring of $Text$ with at most d mismatches.

Constraints

- The integer k will be between 1 and 10^1 .
- The integer d will be between 1 and 10^1 .
- The number of strings in Dna will be between 1 and 10^2 .
- Each string in Dna will have a length between 1 and 10^2 .
- All strings in Dna will be DNA strings.

2B Find a Median String

Median String Problem

Find a median string.

Input: A collection of strings Dna and an integer k .

Output: A k -mer $Pattern$ that minimizes $d(Pattern, Dna)$ among all possible choices of k -mers.

 c2/logos/2B.png

Formatting

Input: An integer k , followed by a newline-separated collection of strings Dna .

Output: A string representing a k -mer $Pattern$ that minimizes $d(Pattern, Dna)$ over all k -mers $Pattern$ (If multiple answers exist, you may return any one).

Constraints

- The integer k will be between 1 and 10^1 .
- The number of strings in Dna will be between 1 and 10^2 .
- The length of each string in Dna will be between 1 and 10^2 .
- Each string in Dna will be a DNA string.

2C Find a Profile-most Probable k -mer in a String

Profile-most Probable k -mer Problem

Find a profile-most probable k -mer in a string.

Input: A string $Text$, an integer k , and a $4 \times k$ matrix $Profile$.

Output: A $Profile$ -most probable k -mer in $Text$.

 c2/logos/2C.png

Formatting

Input: A string $Text$, an integer k , and a $4 \times k$ matrix $Profile$ of floats.

Output: A string representing a $Profile$ -most probable k -mer in $Text$ (If multiple answers exist, you may return any one).

Constraints

- The length of $Text$ will be between 1 and 10^3 .
- The integer k will be between 1 and 10^1 .
- $Text$ will be a DNA string.

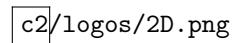
2D Implement GreedyMotifSearch

Greedy Motif Search Problem

Implement GreedyMotifSearch.

Input: A collection of strings Dna , and integers k and t .

Output: A collection of strings resulting from running $GreedyMotifSearch(Dna, k, t)$.

 c2/logo/2D.png

Formatting

Input: Space-separated integers k and t , followed by a newline-separated collection of strings Dna .

Output: A space-separated list of strings containing a collection of strings resulting from running $GreedyMotifSearch(Dna, k, t)$ (If at any step you find more than one *Profile*-most probable k -mer in a given string, use the one occurring first).

Constraints

- The integer k will be between 1 and 10^2 .
- The integer t will be between 1 and 10^2 .
- The number of strings in Dna will be between 1 and 10^2 .
- The length of each string in Dna will be between 1 and 10^2 .
- Each string in Dna will be a DNA string.

2E Implement GreedyMotifSearch with Pseudocounts

Greedy Motif Search with Pseudocounts Problem

Implement *GreedyMotifSearch* with pseudocounts.

Input: A collection of strings Dna , and integers k and t .

Output: A collection of strings resulting from running $\text{GreedyMotifSearch}(Dna, k, t)$ with pseudocounts.

/logos/2E.png

Formatting

Input: Space-separated integers k and t , followed by a newline-separated collection of strings Dna .

Output: A space-separated list of strings containing a collection of strings resulting from running $\text{GreedyMotifSearch}(Dna, k, t)$ with pseudocounts (If at any step you find more than one *Profile*-most probable k -mer in a given string, use the one occurring first).

Constraints

- The integer k will be between 1 and 10^2 .
- The integer t will be between 1 and 10^2 .
- The number of strings in Dna will be between 1 and 10^2 .
- The length of each string in Dna will be between 1 and 10^2 .
- Each string in Dna will be a DNA string.

2F Implement RandomizedMotifSearch

Randomized Motif Search Problem

Implement RandomizedMotifSearch.

Input: A collection of strings Dna , and integers k and t .

Output: A collection of strings resulting from running $RandomizedMotifSearch(Dna, k, t)$ 1000 times. Remember to use pseudocounts!

 c2/logos/2F.png

Formatting

Input: Space-separated integers k and t , followed by a newline-separated collection of strings Dna .

Output: A space-separated list of strings containing a collection of strings resulting from running $RandomizedMotifSearch(Dna, k, t)$ 1000 times. Remember to use pseudocounts!

Constraints

- The integer k will be between 1 and 10^2 .
- The integer t will be between 1 and 10^2 .
- The number of strings in Dna will be between 1 and 10^2 .
- The length of each string in Dna will be between 1 and 10^2 .
- Each string in Dna will be a DNA string.

2G Implement GibbsSampler

Gibbs Sampler Problem

Implement GibbsSampler.

Input: A collection of DNA strings Dna , and integers k , t , and N .

Output: The strings resulting from running $GibbsSampler(Dna, k, t, N)$ with 20 random starts. Remember to use pseudocounts!

 c2/logos/2G.png

Formatting

Input: Space-separated integers k , t , and N , followed by a newline-separated collection of DNA strings Dna .

Output: A space-separated list of strings containing the strings resulting from running $GibbsSampler(Dna, k, t, N)$ with 20 random starts. Remember to use pseudocounts!

Constraints

- The integer k will be between 1 and 10^2 .
- The integer t will be between 1 and 10^2 .
- The integer N will be between 1 and 10^4 .
- The number of strings in Dna will be between 1 and 10^2 .
- The length of each string in Dna will be between 1 and 10^3 .
- Each string in Dna will be a DNA string.

2H Implement DistanceBetweenPatternAndStrings

Distance Between Pattern and Strings Problem
Compute DistanceBetweenPatternAndStrings.

Input: A DNA string *Pattern* and a collection of DNA strings *Dna*.
Output: Distance $d(\text{Pattern}, \text{Dna})$ between *Pattern* and *Dna*.

 c2/logos/2H.png

Formatting

Input: A DNA string *Pattern*, followed by a newline-separated collection of DNA strings *Dna*.

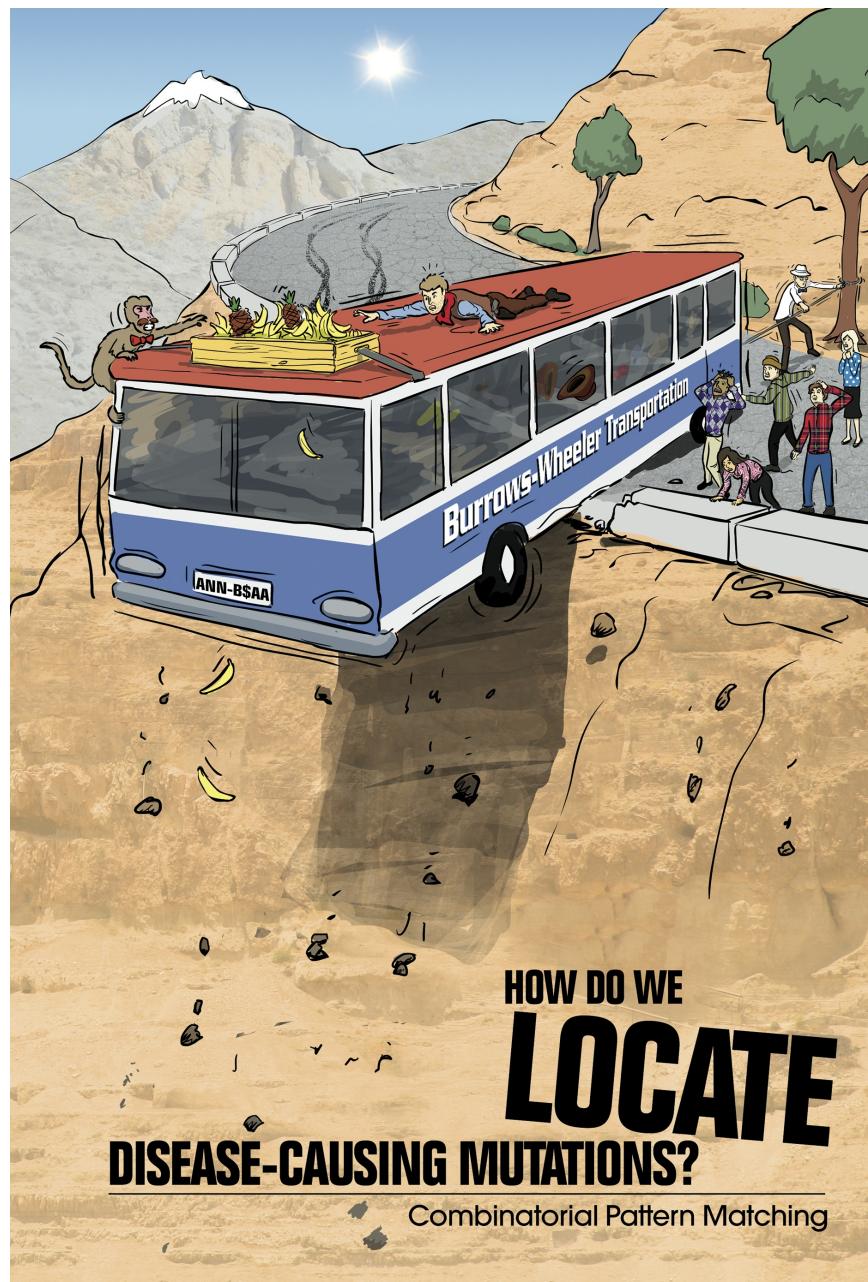
Output: An integer representing the output of *DistanceBetweenPatternAndStrings*(*Pattern*, *Dna*).

Constraints

- The length of *Pattern* will be between 1 and 10^1 .
- The number of strings in *Dna* will be between 1 and 10^2 .
- The length of each string in *Dna* will be between 1 and 10^2 .
- *Pattern* and each string in *Dna* will be DNA strings.

9 How Do We Locate Disease-Causing Mutations?

Combinatorial Pattern Matching



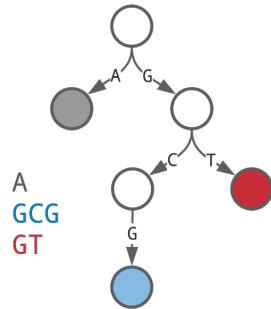
9A Construct a Trie from a Collection of Patterns

Trie Construction Problem

Construct a trie from a set of patterns.

Input: A collection of strings *Patterns*.

Output: $\text{Trie}(\text{Patterns})$.



Formatting

Input: A space-separated list of strings *Patterns*.

Output: Each edge of the adjacency list of $\text{Trie}(\text{Patterns})$ will be newline-separated and encoded by a triple: the first two members of the triple must be the integers labeling the initial and terminal nodes of the edge, respectively; the third member of the triple must be the symbol labeling the edge.

Constraints

- The number of patterns in the string-set *Patterns* will be between 1 and 10^3 .
- The length of any one pattern in *Patterns* will be between 1 and 10^3 .
- No pattern is a prefix of another pattern.

Test Cases

Case 1

Description: A small and hand-solvable dataset taken from the example problem on Stepik.

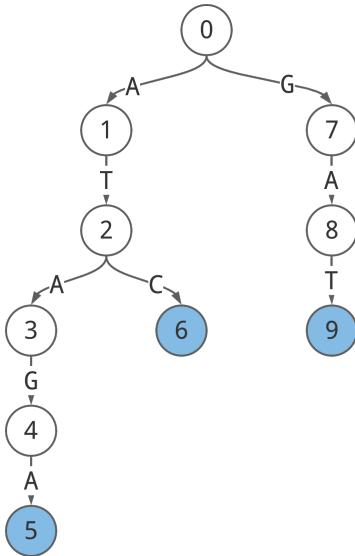
Input:

ATAGA ATC GAT

Output:

```
0 1 A
0 7 G
1 2 T
2 3 A
2 6 C
3 4 G
4 5 A
7 8 A
8 9 T
```

Figure:



Shown above is the trie containing the words ATAGC, ATC, and GAT. These words are outlined by the paths from the root node (labeled 0) to the leaf nodes (labeled 5, 6, and 9, colored blue).

Case 2

Description: No two patterns share the same prefix.

Input:

ATCG TCGA CGAT

Output:

0 1 A
0 5 T
0 9 C
1 2 T
2 3 G
3 4 C
5 6 C
6 7 G
7 8 A
9 10 G
10 11 A
11 12 T

Case 3

Description: All patterns share a prefix, but have distinct suffixes.

Input:

GAGC GAGA GAGT

Output:

0 1 G
1 2 A
2 3 G
3 4 C
3 5 A
3 6 T

Case 4

Description: Patterns have common prefixes and suffixes.

Input:

ATAGC ATGGC

Output:

0 1 A
1 2 T
2 3 A

3 4 G
4 5 C
2 6 G
6 7 G
7 8 C

Case 5

Description: Patterns comprised of repeats or palindromes.

Input:

ATA AGGA

Output:

0 1 A
1 2 T
2 3 A
1 4 G
4 5 G
5 6 A

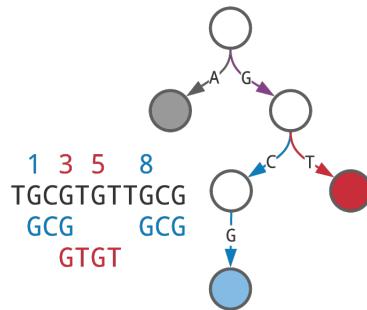
9B Implement TrieMatching

Trie Matching Problem

Find all occurrence of a collection of patterns in a text.

Input: A string *Text* and a collection of (shorter) strings *Patterns*.

Output: Each string *Pattern* in *Patterns* followed by the starting positions in *Text* where *Pattern* appears as a substring.



Formatting

Input: A string *Text* and a space-separated list of strings *Patterns*.

Output: Each string *Pattern* in *Patterns* followed by the starting positions in *Text* where *Pattern* appears as a substring.

Constraints

- The length of *Text* will be between 1 and 10^4 .
- The number of patterns in the string-set *Patterns* will be between 1 and 10^1 .
- The length of any one pattern in *Patterns* will be between 1 and 10^4 .

Test Cases

Case 1

Description: A small and hand-solvable dataset taken from the example problem on Stepik.

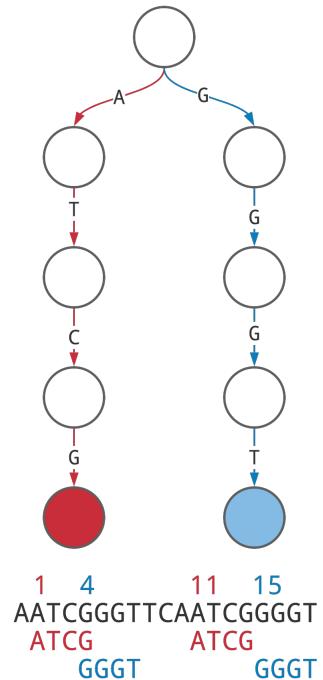
Input:

AATCGGGTTCAATCGGGGT
ATCG GGGT

Output:

ATCG: 1 11
GGGT: 4 14

Figure:



Above is the trie for ATCG and GGGT (shown in red and blue respectively). Shown below the trie is our input string *Text* as well as a representation of where our *Patterns* appear in *Text*.

Case 2

Description: There is no match for any *Pattern* in *Text*.

Input:

AATCGGGTTCAATCGGGGT
GGGA

Output:

GGGA:

Case 3

Description: A *Pattern* in *Patterns* is identical to *Text*.

Input:

AATC
AATC

Output:

AATC: 0

Case 4

Description: Patterns with repeats or palindromic sequences, overlapping occurrences of a pattern, and/or incomplete matches at *any* point in *Text*.

Input:

ATATATA
ATA TAT

Output:

ATA: 0 2 4
TAT: 1 3

Case 5

Description: Matches that only occur once (beginning and end as well).

Input:

GAGCAT
GAG

Output:

GAG: 0

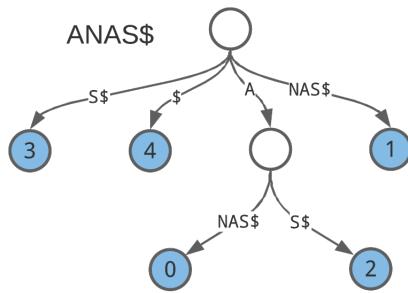
9C Construct the Suffix Tree of a String

Suffix Tree Construction Problem

Construct the suffix tree of a string.

Input: A string *Text*.

Output: *SuffixTree*(*Text*).



Formatting

Input: A string *Text* with an appended dollar-sign ("\$").

Output: A newline-separated list of edge labels from the constructed suffix tree (in any order).

Constraints

- The length of *Text* will be between 1 and 10^3 .

Test Cases

Case 1

Description: A small and hand-solvable dataset taken from the example problem on Stepik.

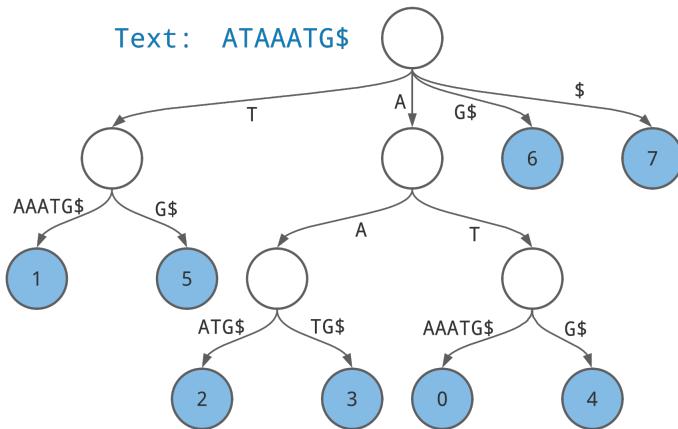
Input:

ATAAAATG\$

Output:

\$
\$
A
A
AAATG\$
AAATG\$
ATG
G\$
G\$
G\$
T
T
TG\$

Figure:



Above is the suffix tree for the string ATAAATG\$ (notice the \$ appended to the end of our input string ATAAATG). Each path from the root to each of the leaves (shown in blue) represents the suffix of ATAAATG\$ corresponding to the index in the leaf.

Case 2

Description: There are repeats in *Text*.

Input:

AATCAATC\$

Output:

\$
\$
\$
\$
A
AATC\$
AATC\$
AATC\$
ATC
C
TC
TC

Case 3

Description: There are no repeats in *Text*.

Input:

ATCG\$

Output:

\$
ATCG\$
CG\$
G\$
TCG\$

Case 4

Description: Large regions of *Text* being a single character or short tandem repeat (STR).

Input:

AAACAA\$

Output:

\$
\$
A

AACA\$

ACA\$

C\$

CA\$

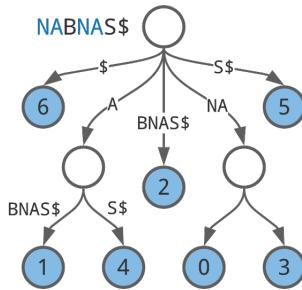
9D Find the Longest Repeat in a String

Longest Repeat Problem

Find the longest repeat in a string.

Input: A string *Text*.

Output: A longest substring of *Text* that appears in *Text* more than once.



Formatting

Input: A string *Text*.

Output: The longest substring that appears more than once in *Text*.

Constraints

- The length of *Text* will be between 1 and 10^3 .

Test Cases

Case 1

Description: A small and hand-solvable dataset from the textbook.

Input:

panamabananas

Output:

ana

Case 2

Description: The longest repeating sequence in *Text* overlaps with another instance of itself.

Input:

GAGAGAG

Output:

GAGAG

Case 3

Description: Multiple repeats occur with the same frequency in *Text* and are the same length.

Input:

AGAGCTCT

Output:

AG or CT (*you will not be penalized for having one over the other, but make sure you only output one*).

Case 4

Description: Repeats that occur at the beginning and end of *Text*.

Input:

AAGCTGAA

Output:

AA

Case 5

Description: There are no repeats in *Text*.

Input:

ABCDEFG

Output:

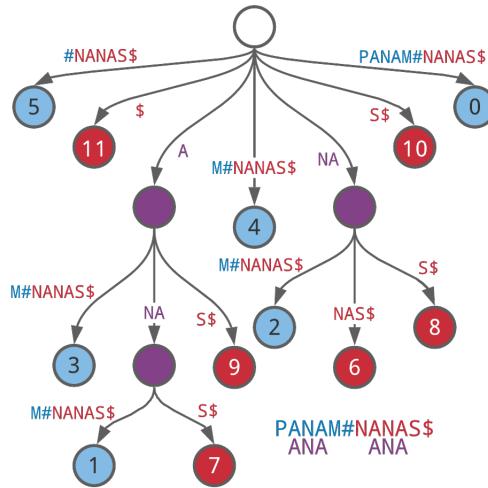
9E Find the Longest Substring Shared by Two Strings

Longest Shared Substring Problem

Find the longest substring shared by two strings.

Input: Strings $Text_1$ and $Text_2$.

Output: The longest substring that occurs in both $Text_1$ and $Text_2$.



Formatting

Input: A pair of strings $Text_1$ and $Text_2$

Output: The longest substring that occurs in both $Text_1$ and $Text_2$

Constraints

- The lengths of $Text_1$ and $Text_2$ will be between 1 and 10^3 .

Test Cases

Case 1

Description: A small and hand-solvable dataset from the textbook.

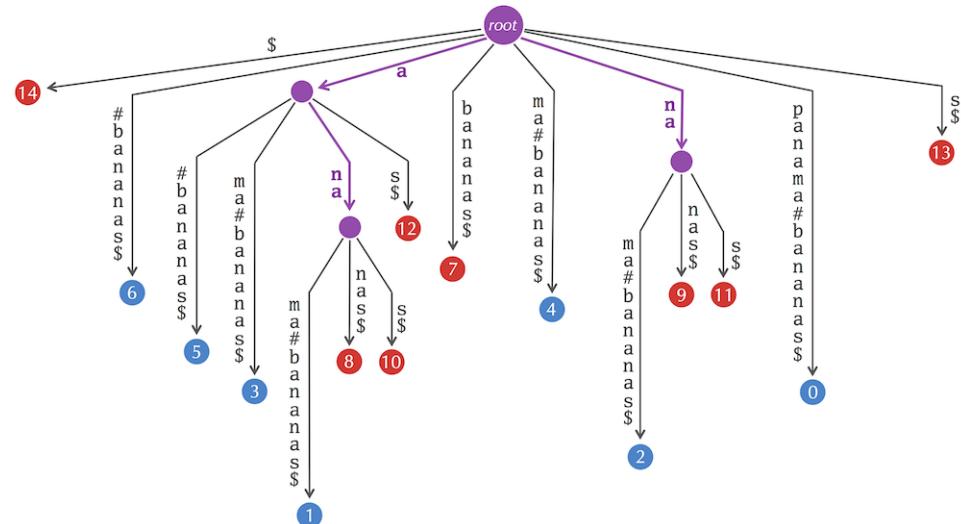
Input:

panama
bananas

Output:

ana

Figure:



Shown above is the suffix tree of the string `panama#bananas$`. Blue and red leaves represent suffixes that start in `panama` and `bananas`, respectively. An internal node is colored purple if it has both blue and red descendants. Each purple node is a shared substring of `panama` and `bananas`. The longest shared substring (purple node) is `ana`.

Case 2

Description: $Text_1$ and $Text_2$ have no common substring.

Input:

GAGA

CTCT

Output:

Case 3

Description: $Text_1$ and $Text_2$ only share 1-mers.

Input:

GAGT

GGCT

Output:

C or G or T (you will not be penalized for having one over the other, but make sure you only output one).

Case 4

Description: $Text_1 = Text_2$.

Input:

GAGCAT

GAGCAT

Output:

GAGCAT

Case 5

Description: The suffix of $Text_1$ and the prefix of $Text_2$ are the same.

Input:

GAGCAT

CATAGA

Output:

CAT

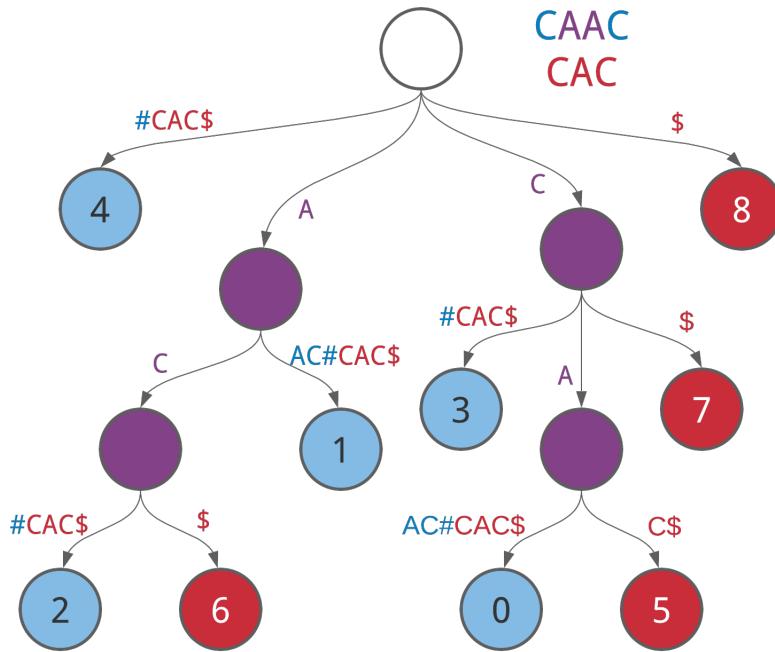
9F Find the Shortest Non-Shared Substring of Two Strings

Shortest Non-Shared Substring Problem

Find the shortest substring of one string that does not appear in another string.

Input: Strings $Text_1$ and $Text_2$.

Output: The shortest substring of $Text_1$ that does not appear in $Text_2$.



Formatting

Input: A pair of strings $Text_1$ and $Text_2$.

Output: The shortest substring of $Text_1$ that does not appear in $Text_2$.

Constraints

- The lengths of $Text_1$ and $Text_2$ will be between 1 and 10^3 .

Test Cases

Case 1

Description: A small and hand-solvable dataset.

Input:

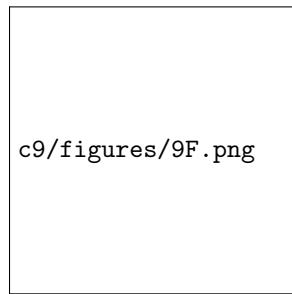
CAAC

CAC

Output:

AA

Figure:



TODO

Case 2

Description: $Text_1$ and $Text_2$ are identical.

Input:

GAGCAT
GAGCAT

Output:

Case 3

Description: $Text_1$ and $Text_2$ only differ by one character.

Input:

GAGT
GAGC

Output:

T

Case 4

Description: $Text_1$ and $Text_2$ are completely different (no shared characters).

Input:

GG
CT

Output:

G OR C OR T (*you will not be penalized for having one over the other, but make sure you only output one*).

Case 5

Description: $Text_1$'s prefix is the same as $Text_2$'s suffix, or vice versa.

Input:

CGAGCATA
ATACGAGC

Output:

CA OR AC (*you will not be penalized for having one over the other, but make sure you only output one*).

9G Construct the Suffix Array of a String

Suffix Array Construction Problem

Construct the suffix array of a string.

Input: A string *Text*.

Output: *SuffixArray*(*Text*).

7	\$
1	ANANAS\$
3	ANAS\$
5	AS\$
0	BANANAS\$
2	NANAS\$
4	NAS\$
6	S\$

Formatting

Input: A string *Text*.

Output: A space-separated list of integers corresponding to *SuffixArray*(*Text*).

Constraints

- The length of *Text* will be between 1 and 10^3 .

Test Cases

Case 1

Description: A small and hand-solvable dataset from the textbook.

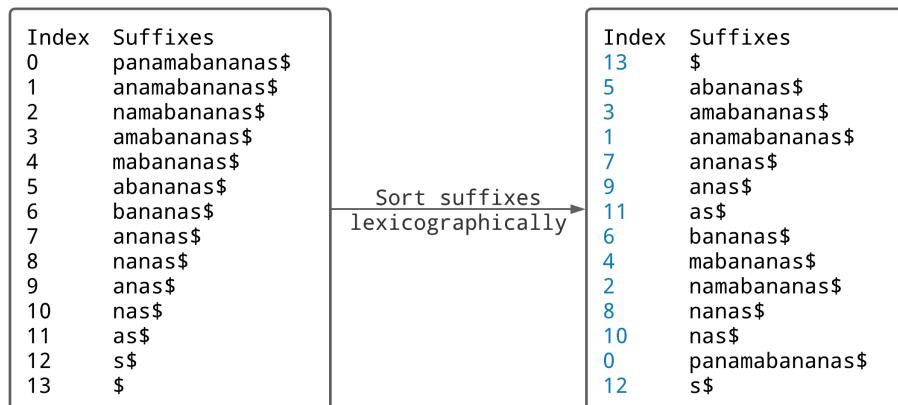
Input:

panamabananas

Output:

13 5 3 1 7 9 11 6 4 2 8 10 0 12

Figure:



Shown above is a general (and inefficient) construction of the suffix array of the input string panamabananas. We first generate all suffixes of *Text* before sorting the suffixes lexicographically and outputting the indices representing the sorted suffixes as the complete suffix array of *Text*.

Case 2

Description: There are repeats in *Text*.

Input:

AATCAATC

Output:

8 4 0 5 1 6 2 7 3

Case 3

Description: There are no repeats in *Text*.

Input:

ATCG

Output:

4 0 2 3 1

Case 4

Description: Large regions of *Text* being a single character or short tandem repeat (STR).

Input:

AAACA

Output:

5 4 0 1 2 3

Case 5

Description: Many different characters in one pattern.

Input:

ABCDEF

Output:

6 0 1 2 5 4 3

9H Pattern Matching with the Suffix Array

Multiple Pattern Matching with the Suffix Array

Use the suffix array of a string to find all occurrences of a collection of Patterns.

Input: A string *Text* and a collection *Patterns* containing (shorter) strings.

Output: All starting positions in *Text* where a string from *Patterns* appears as a substring.

```
7  $
1 ANANAS$
3 ANAS$
5 AS$      ANA: 1 3
0 BANANAS$  BAN: 0
2 NANAS$
4 NAS$
6 S$
```

Formatting

Input: A string *Text* and a space-separated list of strings *Patterns*

Output: A newline-separated list of strings from *Patterns*. Each *Pattern* in *Patterns* is followed by a colon (":") and a space-separated list of starting indices in *Text* where *Pattern* appears as a substring.

Constraints

- The length of *Text* will be between 1 and 10^4 .
- The number of patterns in the string-set *Patterns* will be between 1 and 10^1 .
- The length of any one pattern in *Patterns* will be between 1 and 10^4 .

Note

Please refer to this problem's test cases for formatting and correctness for problems 9L, 9M, and 9N.

Test Cases

Case 1

Description: A small and hand-solvable dataset taken from the example problem on Stepik.

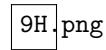
Input:

AATCGGGTTCAATCGGGGT
ATCG GGGT

Output:

ATCG: 1 11
GGGT: 4 15

Figure:



The complete Burrows-Wheeler matrix shown above can be inferred using only the Burrows-Wheeler transform of this string and the Last-To-First property of the Burrows-Wheeler transform. We then search for our query strings, ATCG and GGGT, as prefixes in the rows of our matrix. Finally, we can use the suffix array of the database string to locate the positions of query matches.

Case 2

Description: There are no matches in *Text* to any pattern in *Patterns*.

Input:

ATATATATAT
GT AGCT TAA AAT AATAT

Output:

GT:
AGCT:
TAA:
AAT:
AATAT:

Case 3

Description: *Text* contains overlapping occurrences of *Patterns*.

Input:

bananas
ana as

Output:

ana: 1 3
as: 5

Case 4

Description: Large regions of *Text* being a single character or short tandem repeat (STR).

Input:

AAACAA
AA

Output:

AA: 0 1 4

Case 5

Description: *Text* is palindromic or has substrings that are palindromic.

Input:

GAGCAT
GA AG

Output:

GA: 0

AG: 1

9I Construct the Burrows-Wheeler Transform of a String

Burrows-Wheeler Transform Construction Problem

Construct the Burrows-Wheeler transform of a string.

Input: A string *Text*.

Output: $BWT(Text)$.

```
$BANANA$  
ANANAS$B  
ANAS$BA  
AS$BANA  
BANANAS$  
NANAS$BA  
NAS$BANA  
S$BANANA
```

Formatting

Input: A string *Text*.

Output: A string *Transform* representing $BWT(Text)$.

Constraints

- The length of *Text* will be between 1 and 10^3 .

Test Cases

Case 1

Description: A small and hand-solvable dataset taken from the example problem on Stepik.

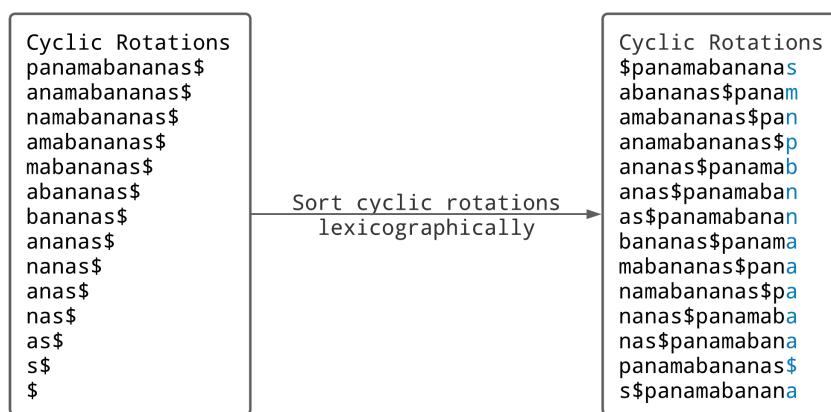
Input:

panamabananas

Output:

smnpbnnaaaaa\$a

Figure:



Shown above is a general (and inefficient) construction process for the Burrows-Wheeler Transform for the string panamabananas. We first generate all cyclic rotations of *Text* before sorting them lexicographically to build a Burrows-Wheeler matrix. Lastly, we output the last column of the Burrows-Wheeler matrix as the Burrows-Wheeler Transform of *Text*. This process is very similar to the process of generating a suffix array.

Case 2

Description: There are repeats in *Text*.

Input:

AATCAATC

Output:

CC\$AATTAA

Case 3

Description: *Text* is made up of only one character.

Input:

AAAAAAAAAA\$

Output:

AAAAAAAAAA\$

Case 4

Description: *Text* is palindromic or has substrings that are palindromic.

Input:

GAGCAT\$

Output:

TGCG\$AA

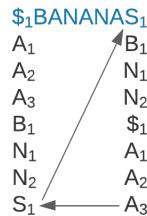
9J Reconstruct a String from its Burrows-Wheeler Transform

Inverse Burrows-Wheeler Transform Problem

Reconstruct a string from its Burrows-Wheeler transform.

Input: A string $Transform$ (with a single “\$” symbol).

Output: The string $Text$ such that $BWT(Text) = Transform$



Formatting

Input: A string $Transform$

Output: A string $Text$ such that $BWT(Text) = Transform$.

Constraints

- The length of $Transform$ will be between 1 and 10^3 .

Test Cases

Case 1

Description: A small and hand-solvable dataset taken from the example problem on Stepik.

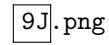
Input:

TTCCTAACG\$A

Output:

TACATCACGT\$

Figure:

 9J.png

Above is a general overview of the BWT inversion process. $\text{TTCCTAACG\$A}$ is $\text{BWT}(\text{Text})$, and we repeat the first-last traversal process until we have "filled" the top row of the BWT matrix. Lastly, we rotate the top row until the $\$$ is at the end of the string to obtain $\text{TACATCACGT\$}$.

Case 2

Description: There are no repeat characters in *Text*.

Input:

T\$ACG

Output:

ACGT\$

Case 3

Description: *Text* is made up of only one character.

Input:

AAAAAAA\$

Output:

AAAAAAA\$

Case 4

Description: *Text* is palindromic or has substrings that are palindromic.

Input:

TGCG\$AA

Output:

GAGCAT\$

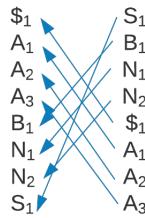
9K Generate the Last-to-First Mapping of a String

Last-to-First Mapping Problem

Construct a Last-to-First Array of a Burrows-Wheeler Transform.

Input: A string *Transform* representing the Burrows-Wheeler transform of an unknown string *Text*.

Output: An array *LastToFirst* that provides the following information: given a symbol at position *i* in *LastColumn* of the Burrows-Wheeler matrix of *Text*, *LastToFirst*(*i*) reveals its position in *FirstColumn*.



Formatting

Input: A string *Transform* representing the Burrows-Wheeler transform of an unknown string *Text*.

Output: A space-separated list of integers representing the array *LastToFirst*.

Constraints

- The length of *Transform* will be between 1 and 10^3 .
- The value of *i* will be between 1 and 10^3 .

Test Cases

Case 1

Description: A small and hand-solvable dataset from the textbook.

Input:

smnpbnaaaaa\$a

Output:

13 8 9 12 7 10 11 1 2 3 4 5 0 6

Figure:

[c9/figures/9K.png]

TODO

Case 2

Description: There are no repeat characters in *Text*.

Input:

T\$ACG

Output:

4 0 1 2 3

Case 3

Description: *Text* is made up of only one character.

Input:

AAAAAAAAAA\$

Output:

1 2 3 4 5 6 7 8 9 10 0

Case 4

Description: *Text* is palindromic or has substrings that are palindromic.

Input:

TGCG\$AA

Output:

5 0 3 2 6 1 4

9L Implement BWMatching

Implement BWMatching

Find all occurrences of a collection of patterns in a text.

Input: A string *Text* and a collection *Patterns* containing (shorter) strings.

Output: All starting positions in *Text* where a string from *Patterns* appears as a substring.

	\$BANANAS
	ANANAS\$B
	ANAS\$BAN
1 3 5	AS\$BANAN
BANANAS\$	BANANAS\$
ANA AS	NANAS\$BA
ANA	NAS\$BANA
	S\$BANANA

Formatting

Input: A string *Text* and a space-separated list of strings *Patterns*

Output: A newline-separated list of strings from *Patterns*. Each *Pattern* in *Patterns* is followed by a colon (":") and a space-separated list of starting indices in *Text* where *Pattern* appears as a substring.

Constraints

- The length of *Text* will be between 1 and 10^4 .
- The number of patterns in the string-set *Patterns* will be between 1 and 10^1 .
- The length of any one pattern in *Patterns* will be between 1 and 10^4 .

Note

Please refer back to problem 9H (Pattern Matching with Suffix Array) for relevant test cases.

9M Implement BetterBWMatching

Implement BetterBWMatching

Find all occurrences of a collection of patterns in a text.

Input: A string *Text* and a collection *Patterns* containing (shorter) strings.

Output: All starting positions in *Text* where a string from *Patterns* appears as a substring.

	\$BANANAS
	ANANAS\$B
	ANAS\$BAN
1 3 5	AS\$BANAN
BANANAS\$	BANANAS\$
ANA AS	NANAS\$BA
ANA	NAS\$BANA
	S\$BANANA

Formatting

Input: A string *Text* and a space-separated list of strings *Patterns*

Output: A newline-separated list of strings from *Patterns*. Each *Pattern* in *Patterns* is followed by a colon (":") and a space-separated list of starting indices in *Text* where *Pattern* appears as a substring.

Constraints

- The length of *Text* will be between 1 and 10^4 .
- The number of patterns in the string-set *Patterns* will be between 1 and 10^1 .
- The length of any one pattern in *Patterns* will be between 1 and 10^4 .

Note

Please refer back to problem 9H (Pattern Matching with Suffix Array) for relevant test cases.

9N Find All Occurrences of a Collection of Patterns in a String

Multiple Pattern Matching Problem

Find all occurrences of a collection of patterns in a text.

Input: A string *Text* and a collection *Patterns* containing (shorter) strings.

Output: All starting positions in *Text* where a string from *Patterns* appears as a substring.

	\$BANANAS
	ANANAS\$B
	ANAS\$BAN
1 3 5	AS\$BANAN
BANANAS\$	BANANAS\$
ANA AS	NANAS\$BA
ANA	NAS\$BANA
	S\$BANANA

Formatting

Input: A string *Text* and a space-separated list of strings *Patterns*

Output: A newline-separated list of strings from *Patterns*. Each *Pattern* in *Patterns* is followed by a colon (":") and a space-separated list of starting indices in *Text* where *Pattern* appears as a substring.

Constraints

- The length of *Text* will be between 1 and 10^4 .
- The number of patterns in the string-set *Patterns* will be between 1 and 10^1 .
- The length of any one pattern in *Patterns* will be between 1 and 10^4 .

Note

Please refer back to problem 9H (Pattern Matching with Suffix Array) for relevant test cases.

9O Find All Approximate Occurrences of a Collection of Patterns in a String

Multiple Approximate Pattern Matching Problem

Find all approximate occurrences of a collection of patterns in a text.

Input: A string *Text*, and a collection of strings *Patterns*, and an integer *d*.

Output: All positions in *Text* where a string from *Patterns* appears as a substring with at most *d* mismatches.

```
$BANANAS
ANANAS$B
ANAS$BAN
AS$BANAN  0 2 4
BANANAS$  BANANAS$  1
NANAS$BA  NAN  NAN
NAS$BANA   NAN
S$BANANA
```

Formatting

Input: A string *Text*, a space-separated list of strings *Patterns*, and an integer *d*

Output: A newline-separated list of strings from *Patterns*. Each *Pattern* in *Patterns* is followed by a colon (":") and a space-separated list of starting indices in *Text* where *Pattern* appears as a substring with at most *d* mismatches.

Constraints

- The length of *Text* will be between 1 and 10^4 .
- The number of patterns in the string-set *Patterns* will be between 1 and 10^3 .
- The length of any one pattern in *Patterns* will be between 1 and 10^2 .

Test Cases

Case 1

Description: A small and hand-solvable dataset taken from the example problem on Stepik.

Input:

```
ACATGCTACTTT  
ATT GCC GCTA TATT  
1
```

Output:

```
ATT: 2 7 8 9  
GCC: 4  
GCTA: 4  
TATT: 6
```

Figure:

 c9/figures/90.png

TODO

Case 2

Description: *Patterns* contains partial and complete matches.

Input:

ACGT
GG AC
1

Output:

GG: 1 2
AC: 0

Case 3

Description: *Patterns* contains no matches.

Input:

GGGG
TT AA
1

Output:

AA:
TT:

Case 4

Description: *Patterns* contains no matches, but has exactly d mismatches.

Input:

GG
TT AA
2

Output:

AA: 0
TT: 0

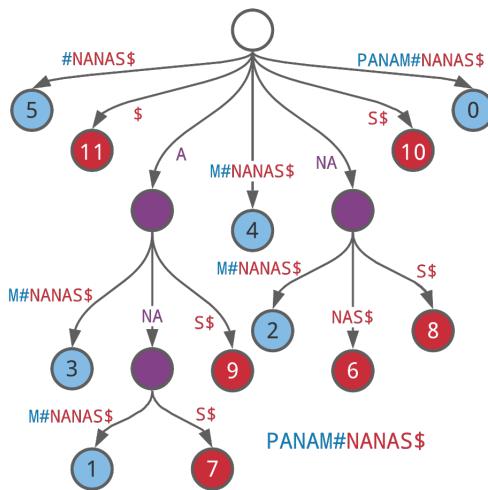
9P Implement TreeColoring

Tree Coloring Problem

Color the internal nodes of a suffix tree given colors of the leaves.

Input: A tree with leaves colored red or blue.

Output: Coloring of all internal nodes of the tree such that a node is colored red if all of its descendants are red, blue if all of its descendants are blue, and purple if it has both red and blue descendants.



Formatting

Input: An adjacency list, followed by color labels for leaf nodes.

Output: A newline-separated list of nodes and their color labels in the following form: *node label: color label*.

Constraints

- The number of nodes in the adjacency list will be between 1 and 10^2 .
- The number of edges in the adjacency list will be between 1 and 10^2 .

Test Cases

Case 1

Description: TODO

Input:

TODO

Output:

TODO

Figure:

 c9/figures/9P.png

TODO

9Q Construct the Partial Suffix Array of a String

Partial Suffix Array Construction Problem

Construct the partial suffix array of a string.

Input: A string $Text$ and a positive integer k .

Output: $SuffixArray_k(Text)$, in the form of a list of ordered pairs $(i, SuffixArray(i))$ for all nonempty entries in the partial suffix array.

```
7 $  
1 ANANAS$  
3 ANAS$  
5 AS$  
0 BANANAS$  
2 NANAS$  
4 NAS$  
6 S$
```

Formatting

Input: A string $Text$ and a positive integer k .

Output: A newline-separated list of space-separated ordered pairs $(i, SuffixArray(i))$ for all nonempty entries in $SuffixArray_k(Text)$.

Constraints

- The length of $Text$ will be between 1 and 10^5 .
- The integer k will be between 1 and 10^1 .

Test Cases

Case 1

Description: A small and hand-solvable dataset from the textbook.

Input:

panamabananas\$
5

Output:

1 5
11 10
12 0

Figure:



Shown above is a general (and inefficient) construction of the partial suffix array of the input string *panamabananas* with $k = 5$. We first generate all suffixes of *Text* before sorting the suffixes lexicographically and outputting the indices representing the sorted suffixes as the complete suffix array of *Text*. Finally, we output only the indices divisible by $k = 5$.

Case 2

Description: There are repeats in *Text*.

Input:

AATCAATC\$
4

Output:

0 8
1 4
2 0

Case 3

Description: There are no repeats in *Text*.

Input:

ATCG\$
3

Output:

1 0
3 3

Case 4

Description: Large regions of *Text* being a single character or short tandem repeat (STR).

Input:

AAACAA\$
5

Output:

0 5
2 0

Case 5

Description: Many different characters in one pattern.

Input:

ABCDEFED\$
3

Output:

0 6
1 0
6 3

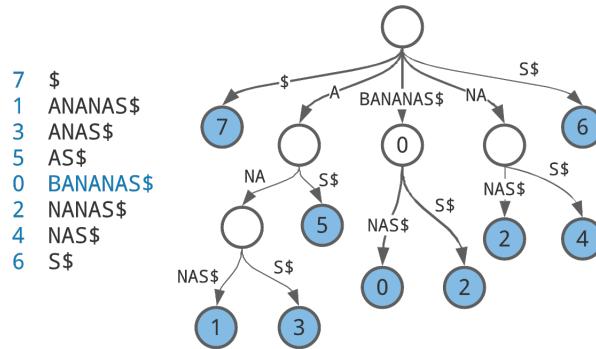
9R Construct a Suffix Tree from a Suffix Array

Suffix Tree Construction from Suffix Array Problem

Construct a suffix tree from the suffix array and LCP array of a string.

Input: A string $Text$, $\text{SuffixArray}(Text)$, $\text{LCP}(Text)$.

Output: The strings labeling the edges of $\text{SuffixTree}(Text)$, in any order.



Formatting

Input: A string $Text$, $\text{SuffixArray}(Text)$, and $\text{LCP}(Text)$.

Output: $\text{SuffixArray}_K(Text)$, in the form of a list of ordered pairs $(i, \text{SuffixArray}(i))$ for all nonempty entries in the partial suffix array.

Constraints

- The length of $Text$ will be between 1 and 10^3 .
- The length of $\text{SuffixArray}(Text)$ will be between 1 and 10^3 .
- The length of $\text{LCP}(Text)$ will be between 1 and 10^3 .

Test Cases

Case 1

Description: A small and hand-solvable dataset taken from the example problem on Stepik.

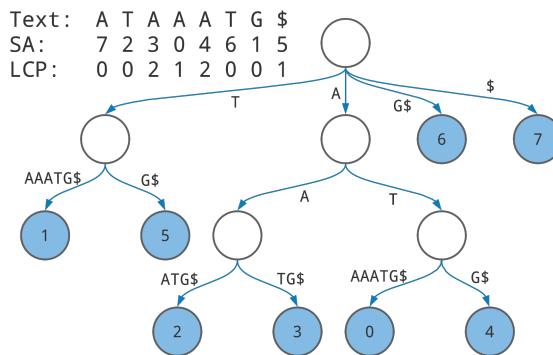
Input:

ATAAATG\$
7 2 3 0 4 6 1 5
0 0 2 1 2 0 0 1

Output:

\$
\$
A
A
AAATG\$
AAATG\$
ATG
G\$
G\$
G\$
T
T
TG\$

Figure:



Above is the suffix tree for the string ATAAATG\$ (notice the \$ appended to the end of our input string ATAAATG). Each path from the root to each of the leaves (shown in blue) represents the suffix of ATAAATG\$ corresponding to the index in the leaf.

Case 2

Description: There are repeats in *Text*.

Input:

AATCAATC\$
8 4 0 5 1 7 3 6 2
0 0 4 1 3 0 1 0 2

Output:

\$
\$
\$
\$
\$
A
AATC\$
AATC\$
AATC\$
ATC
C
TC
TC

Case 3

Description: There are no repeats in *Text*.

Input:

ATCG\$
4 0 2 3 1
0 0 0 0 0

Output:

\$
ATCG\$
CG\$
G\$
TCG\$

Case 4

Description: Large regions of *Text* being a single character or short tandem repeat (STR).

Input:

AAACAA\$
5 4 0 1 2 3

0 0 1 2 1 0

Output:

\$
\$
A
A
ACA\$
CA\$
CA\$
CA\$