7.1-7.4, 7.6
First Applications of Suffix Trees

2015/6/2 (Tue)

ISA

Donghyuk Kim

Contents

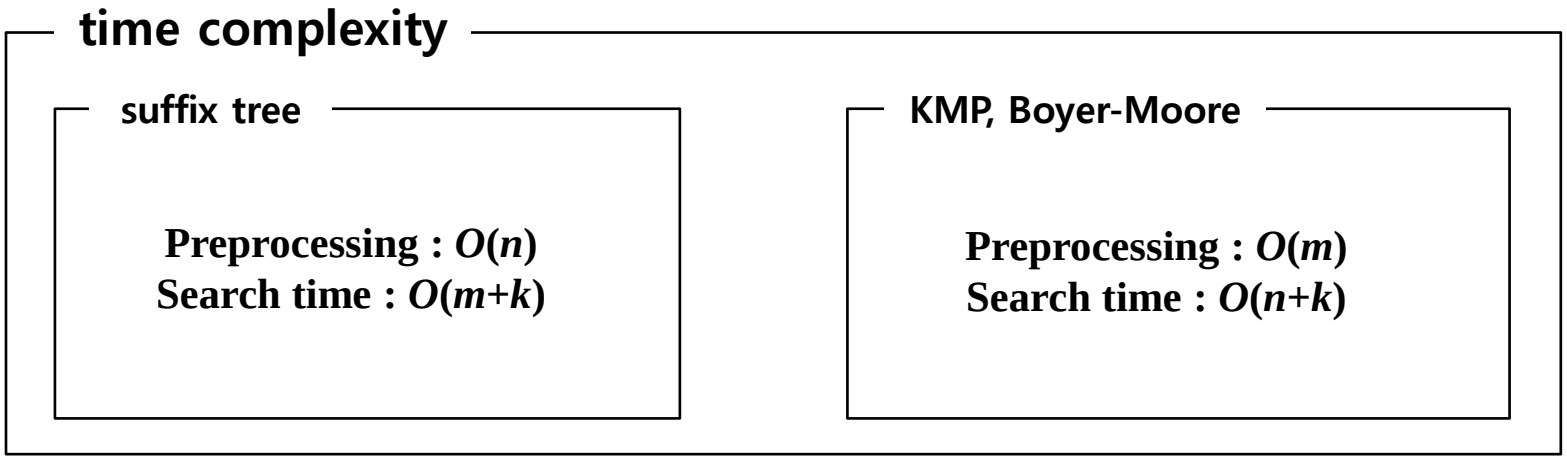
Contents

7.1 Exact string matching

- **Case 1) Pattern and text are both known.**
- **Case 2) Only text is known first.**
- **Case 3) Only pattern is known first.**

7.1 Exact string matching

- **Case 1) Considering the case that the pattern *P* and the text *T* are both known.**
  - The use of suffix tree achieves the same worst-case bound, $O(n+m+k)$, as the KMP or Boyer-Moore algorithms.
    - n : length of text T.
    - m : length of pattern P.



7.1 Exact string matching

- **Case 2) When the text *T* is known first and kept fixed.**
  - Using a suffix tree for *T*, all occurrences can be found in $O(m+k)$ time.
    - m : length of pattern P.
    - k : the number of occurrences of P in T.
  - Totally independent of the size of *T*

- **KMP and Boyer-Moore can't.**

7.1 Exact string matching

- **Case 3) When a pattern is first fixed and can be preprocessed before the text is known,**
  - is the classic situation handled by KMP or Boyer-Moore algorithm.
  - Those algorithms spend $O(m)$ preprocessing time,
    - so that the search can be done in $O(n)$ time whenever a text T is specified.
  - Suffix trees can be used in this scenario to achieve the same time bound with reverse suffix trees.

7.1 Exact string matching

- **Case 1) Pattern and text are both known.**
  - Suffix tree can run same performance as KMP and Boyer-Moore.
- **Case 2) Only text is known first.**
  - Suffix tree can run, but KMP and Boyer-Moore can't.
- **Case 3) Only pattern is known first.**
  - Reverse suffix tree can run same performance as KMP and Boyer-Moore.

7.1 Exact string matching

- **Case 1) Pattern and text are both known.**
  - Suffix tree can run same performance as KMP and Boyer-Moore.
- **Case 2) Only text is known first.**
  - Suffix tree can run, but KMP and Boyer-Moore can't.
- **Case 3) Only pattern is known first.**
  - Reverse suffix tree can run same performance as KMP and Boyer-Moore.

This situation will be discussed along with a more general problem in Section 7.8.

Contents

7.2 Suffix trees and the exact set matching problem

- **Exact set matching problem**
  - The problem of finding all occurrences from a set of strings *P* in a text *T*, where the set is input all at once.

7.2 Suffix trees and the exact set matching problem

- **Exact set matching problem**
  - The problem of finding all occurrences from a set of strings *P* in a text *T*, where the set is input all at once.

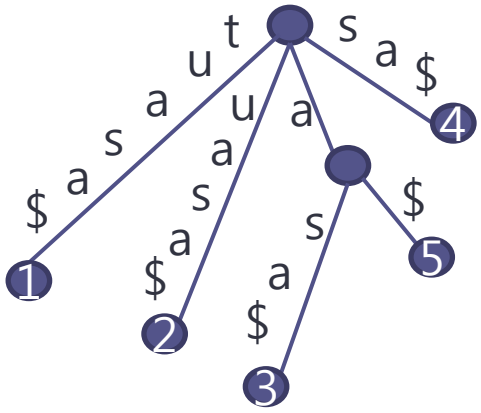
text : | t | u | a | s | a | **n=5**

P

Pattern 1 : | a | s | a | **m=3**

Pattern 2 : | t | u | b | **m=3**

Pattern 3 : | a | s | d | **m=3**



Exact set matching by suffix tree
- preprocessing : *O*(*n*)
- searching : *O*(*m*)

7.2 Suffix trees and the exact set matching problem

- **Exact set matching problem**
  - The problem of finding all occurrences from a set of strings *P* in a text *T*, where the set is input all at once.

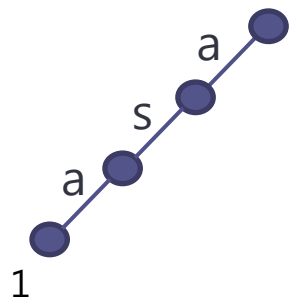
text : | t | u | a | s | a | n=5

P

Pattern 1 : a s a m=3

Pattern 2 : t u b m=3

Pattern 3 : a s d m=3



Exact set matching by keyword tree

7.2 Suffix trees and the exact set matching problem

- **Exact set matching problem**
  - The problem of finding all occurrences from a set of strings *P* in a text *T*, where the set is input all at once.

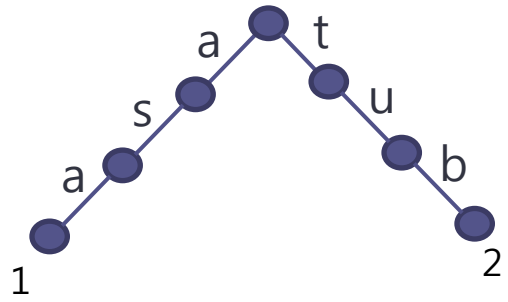
text : | t | u | a | s | a | n=5

P

Pattern 1 : a s a m=3

Pattern 2 : t u b m=3

Pattern 3 : a s d m=3



Exact set matching by keyword tree

7.2 Suffix trees and the exact set matching problem

- **Exact set matching problem**
  - The problem of finding all occurrences from a set of strings *P* in a text *T*, where the set is input all at once.

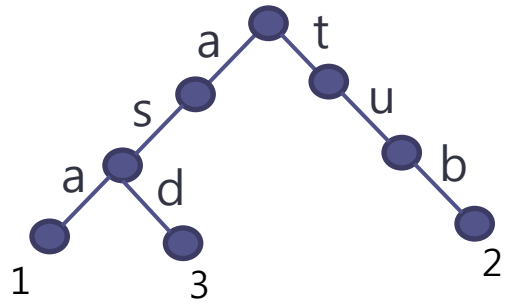
text : | t | u | a | s | a | **n=5**

P

Pattern 1 : a s a **m=3**

Pattern 2 : t u b **m=3**

Pattern 3 : a s d **m=3**



Exact set matching by keyword tree
- preprocessing : $O(m)$
- searching : $O(n)$

7.2.1. Comparing suffix trees and keyword trees for exact set matching

- **The relative advantages of keyword trees versus suffix trees for the exact set matching problem.**

n : length of text T
m : length of pattern P
k : the number of occurrences

7.2.1. Comparing suffix trees and keyword trees for exact set matching

- **The relative advantages of keyword trees versus suffix trees for the exact set matching problem.**
  - Aho-Corasick method (a keyword tree)
    - Build time : $O(m)$
    - Search time : $O(n)$
    - Find all occurrences in T of any pattern from P : $O(m+n+k)$

n : length of text T
m : length of pattern P
k : the number of occurrences

7.2.1. Comparing suffix trees and keyword trees for exact set matching

- **The relative advantages of keyword trees versus suffix trees for the exact set matching problem.**
  - Aho-Corasick method (a keyword tree)
    - Build time : ***O*(*m*)**
    - Search time : ***O*(*n*)**
    - Find all occurrences in T of any pattern from P : ***O*(*m*+*n*+*k*)**
  - Suffix tree $\mathcal{T}$ for T
    - Build time : ***O*(*n*)**
    - Search time : ***O*(*m*)**
    - Total time to search for all occurrences of each pattern in P : ***O*(*n*+*m*+*k*)**

n : length of text T
m : length of pattern P
k : the number of occurrences

7.2.1. Comparing suffix trees and keyword trees for exact set matching

- **Case 1) set of patterns is larger than the text ($n < m$)**
  - Space : suffix tree < Aho-Corasick method
  - Search time : suffix tree > Aho-Corasick method

- **Case 2) total size of the patterns is smaller than the text ($n > m$)**
  - Space : suffix tree > Aho-Corasick method
  - Search time : suffix tree < Aho-Corasick method

n : length of text T
m : length of pattern P
k : the number of occurrences

7.2.1. Comparing suffix trees and keyword trees for exact set matching

- **There is a time/space trade-off and neither method is uniformly superior to the other in time and space.**

Contents

7.3 The substring problem for a database of patterns

- **The substring problem, a set of strings, or a database**
  - Is first known and fixed.
  - Later, a sequence of strings will be presented and for each presented string *S*, the algorithm must find all the strings in the database containing *S* as a substring.

7.3 The substring problem for a database of patterns

- **In the context of databases for genomic DNA data,**
  - The problem of finding substrings is cannot be solved by exact set matching.

- **The DNA database contains a collection of previously sequenced DNA strings.**

7.3 The substring problem for a database of patterns

- **When a new DNA string is sequenced,**
  - It could be contained in an already sequenced string.
  - An efficient method to check that is of value.

7.3 The substring problem for a database of patterns

- **Example)**

A set of strings, or a database

S_1 = a p p l e

S_2 = t o m a t o

S_3 = m e l o n

S_4 = b a n a n a

P_1: tom

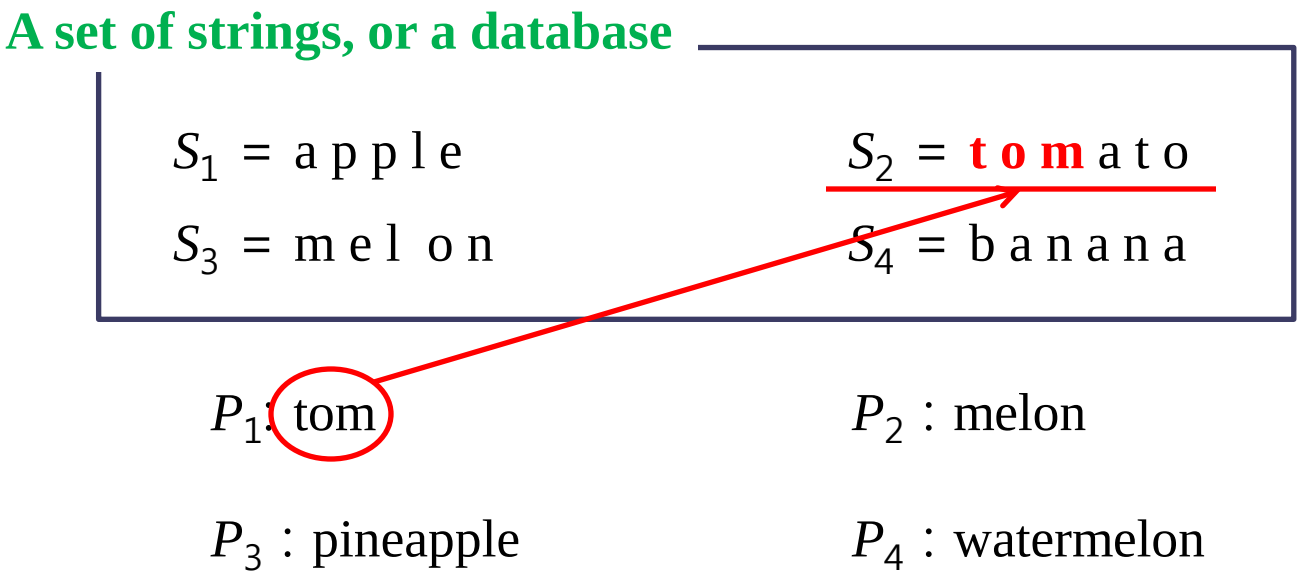
P_2 : melon

P_3 : pineapple

P_4 : watermelon

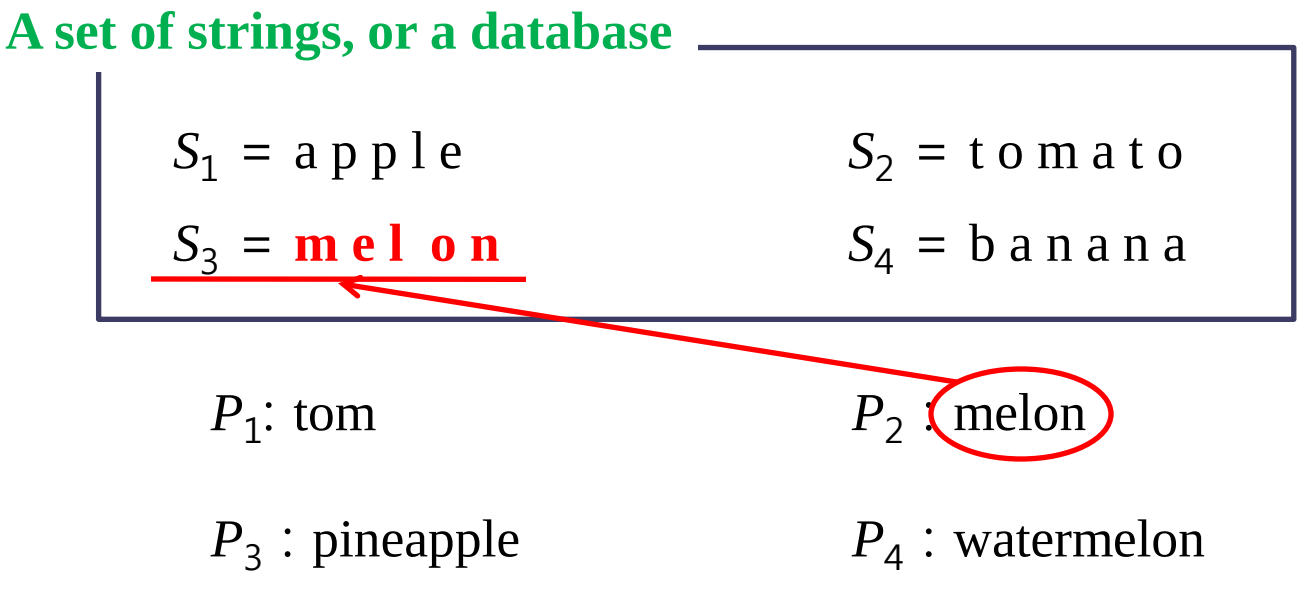
7.3 The substring problem for a database of patterns

- **Example)**



7.3 The substring problem for a database of patterns

- **Example)**



7.3 The substring problem for a database of patterns

- **Example)**
  - The opposite case is also possible
    - The new string contains one of the database strings.
      - But that is the case of exact set matching.

A set of strings, or a database

S_1 = a p p l e

S_2 = t o m a t o

S_3 = m e l o n

S_4 = b a n a n a

P_1: tom

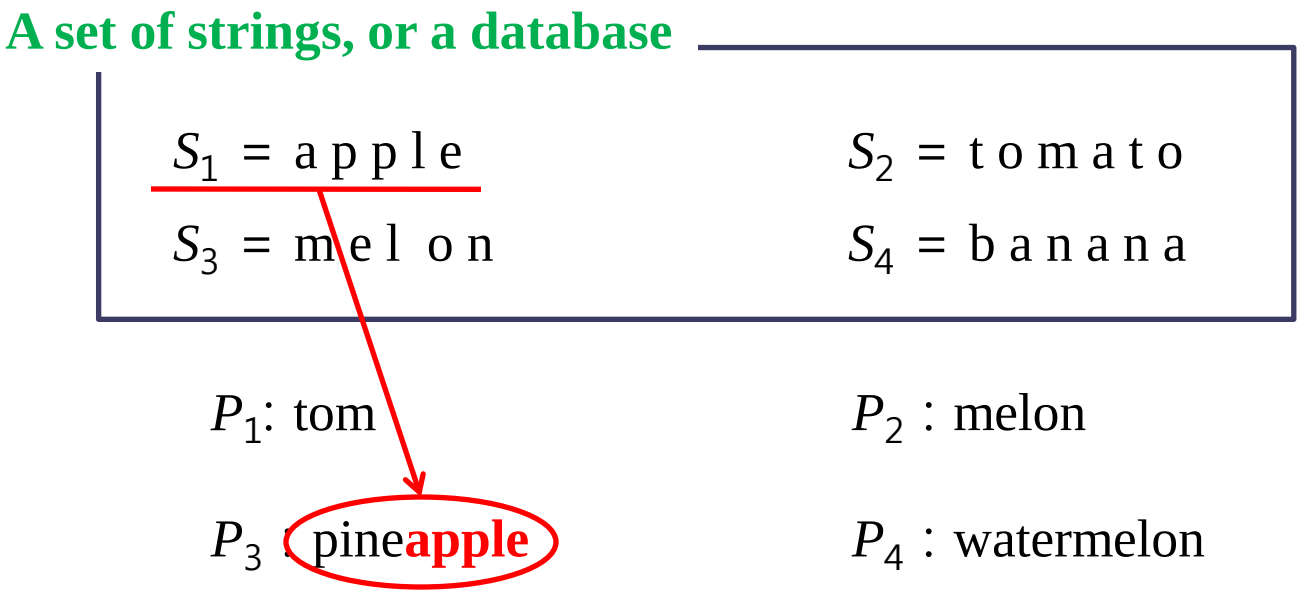
P_2 : melon

P_3 : pineapple

P_4 : watermelon

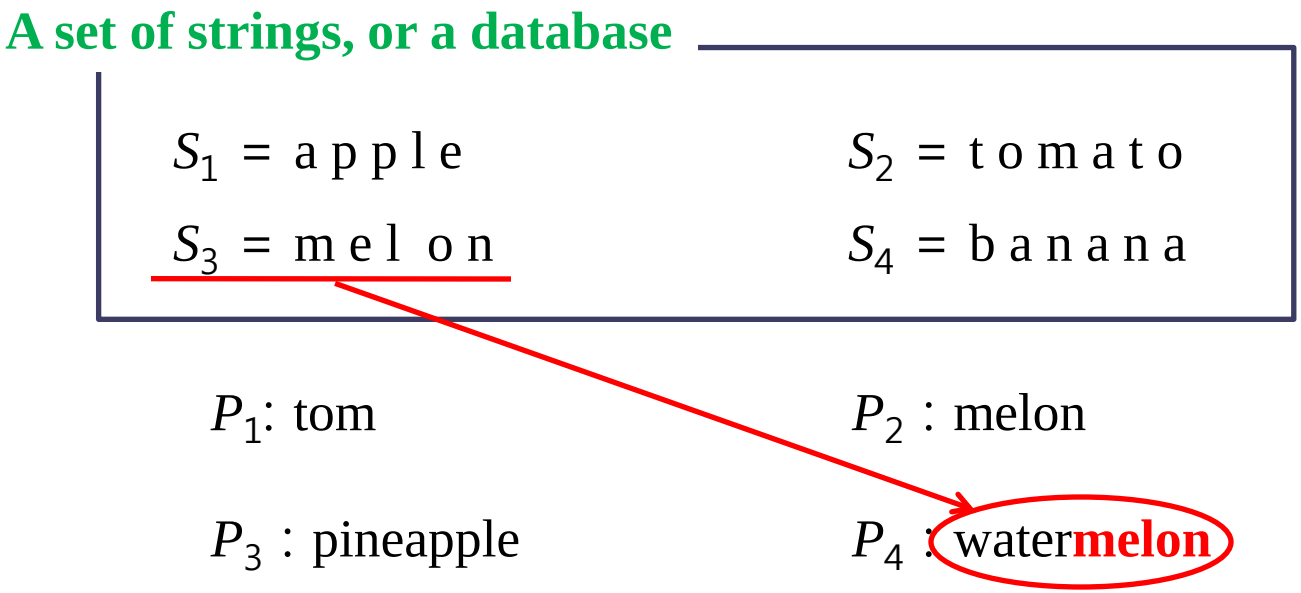
7.3 The substring problem for a database of patterns

- **Example)**
  - The opposite case is also possible
    - The new string contains one of the database strings.
      - But that is the case of exact set matching.



7.3 The substring problem for a database of patterns

- **Example)**
  - The opposite case is also possible
    - The new string contains one of the database strings.
      - But that is the case of exact set matching.



7.3 The substring problem for a database of patterns

- **Identifying the remains of U.S. military personnel.**

7.3 The substring problem for a database of patterns

- **Identifying the remains of U.S. military personnel.**
  - Mitochondrial DNA from live military personnel is collected and a small **interval** of each person's DNA is sequenced.
  - The interval is used as a "**nearly unique**" identifier.

Mitochondrial DNA

DNA interval - { 1324521 }

DNA interval - { 3224321 }

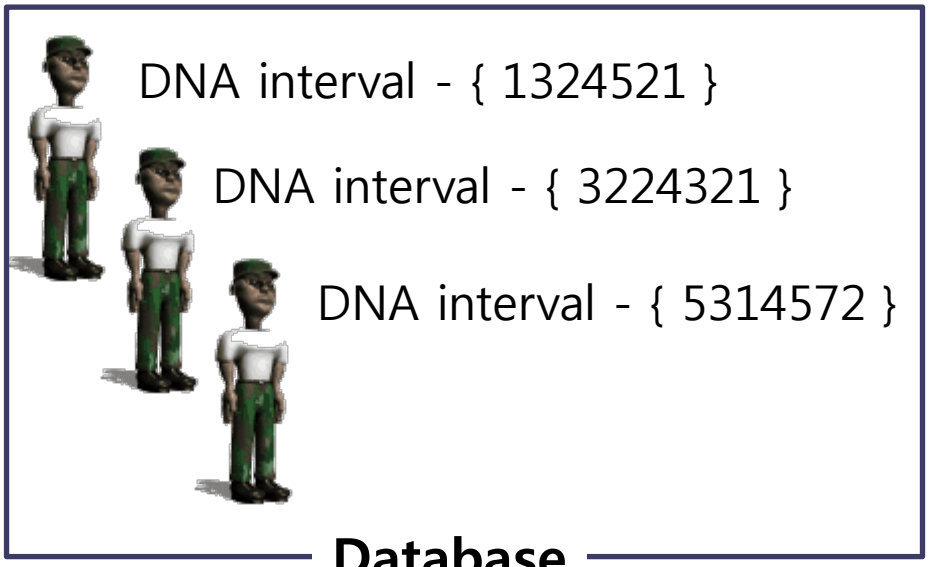
DNA interval - { 5314572 }

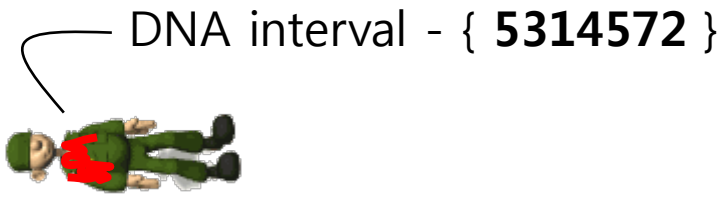
each intervals are difference each other

Database

7.3 The substring problem for a database of patterns

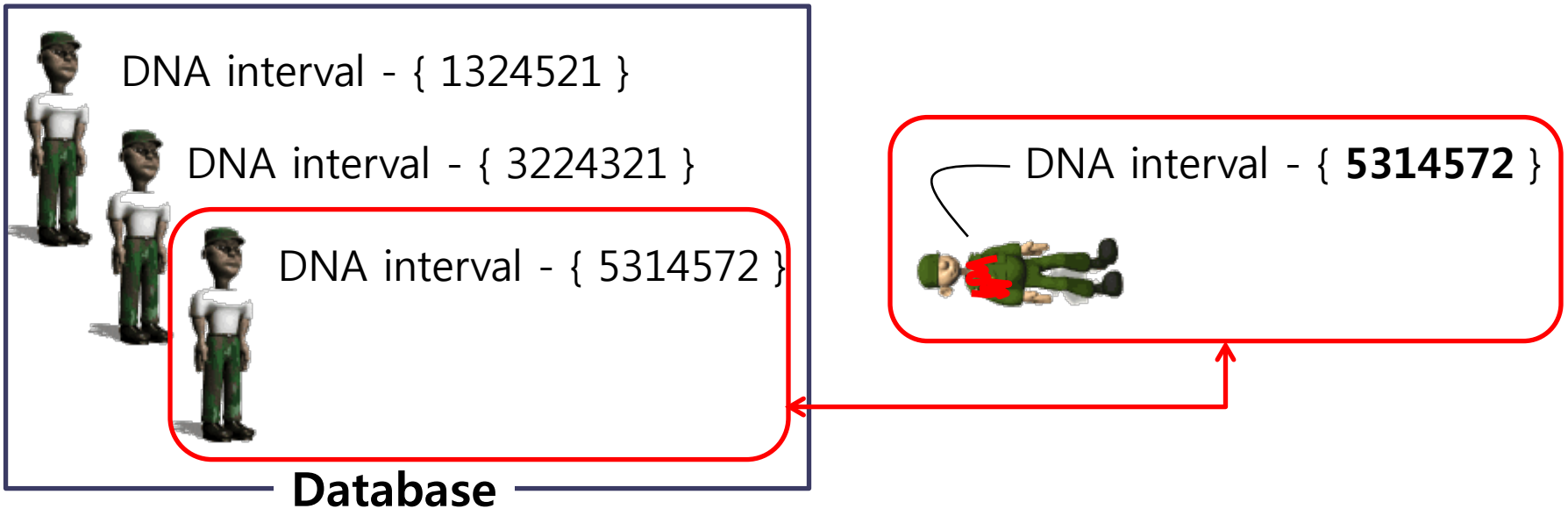
- **Later, a person who have been killed can be identified**
  - That matching the database of the strings with the person's DNA interval.





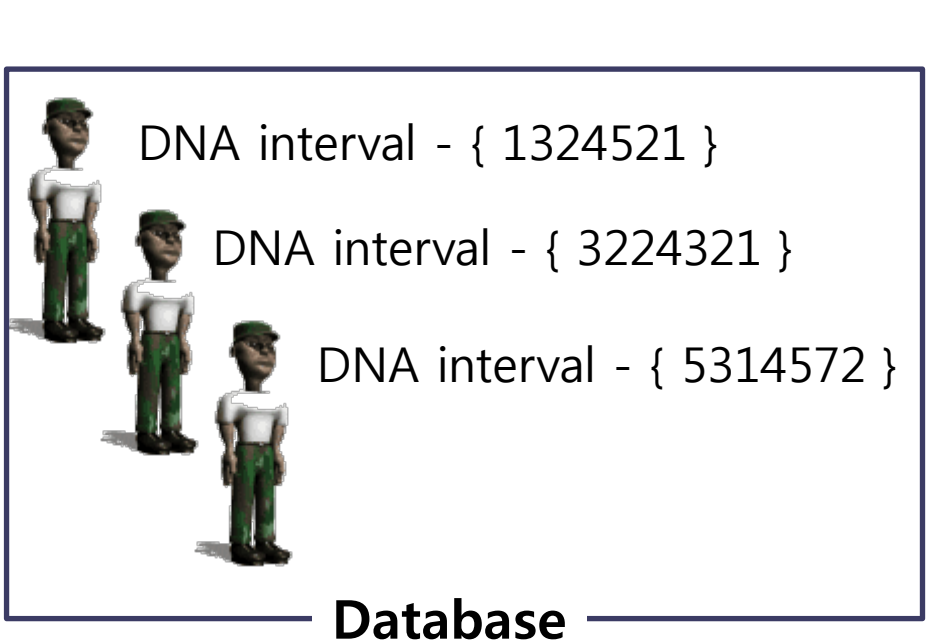
7.3 The substring problem for a database of patterns

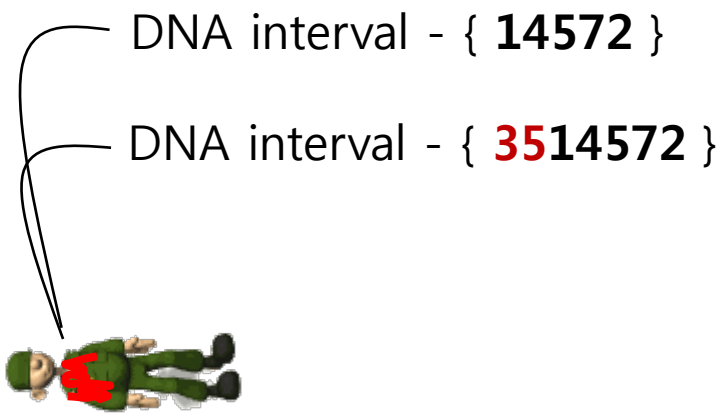
- **Later, a person who have been killed can be identified,**
  - That matching the database of the strings with the person's DNA interval.



7.3 The substring problem for a database of patterns

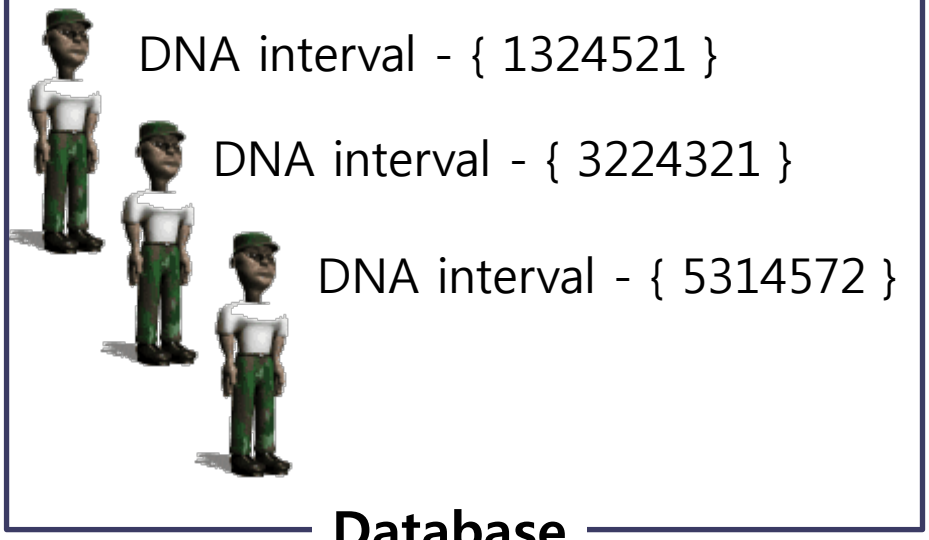
- **The substring variant of this problem arises.**
  - Because the condition of the remains may not allow complete extraction or sequencing of the desired DNA interval.

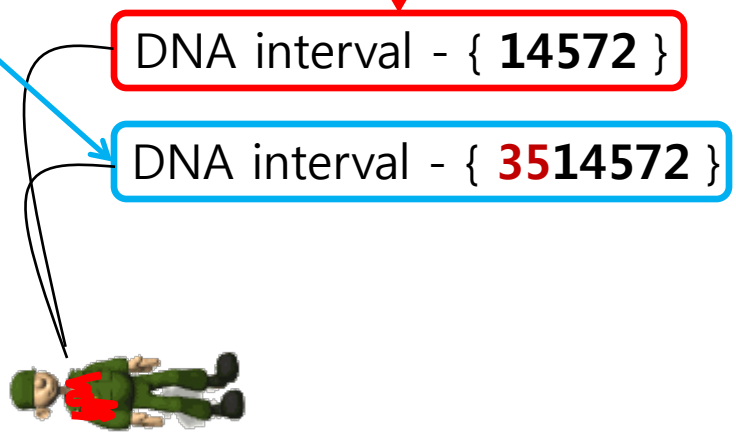




7.3 The substring problem for a database of patterns

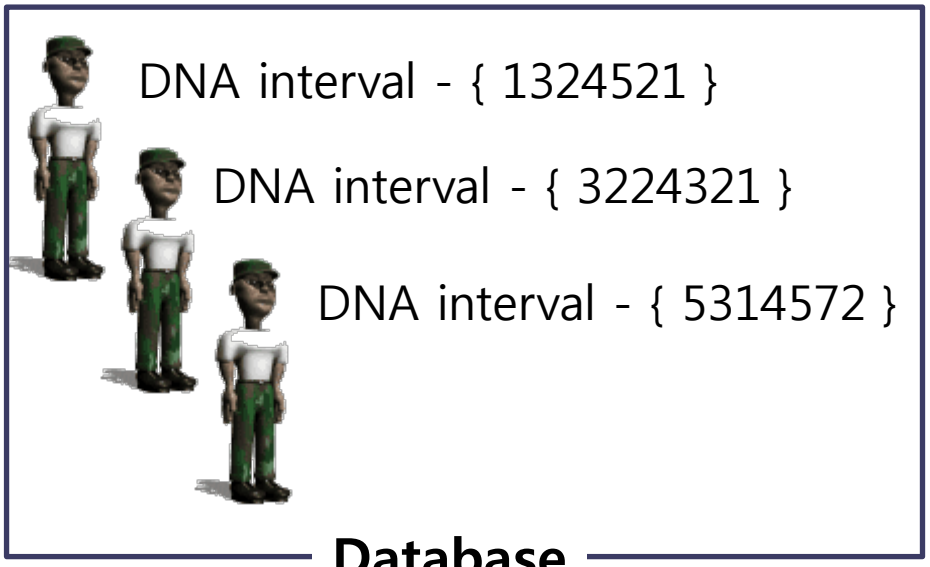
- **The substring variant of this problem arises.**
  - Because the condition of the remains may not allow complete extraction or sequencing of the desired DNA interval.

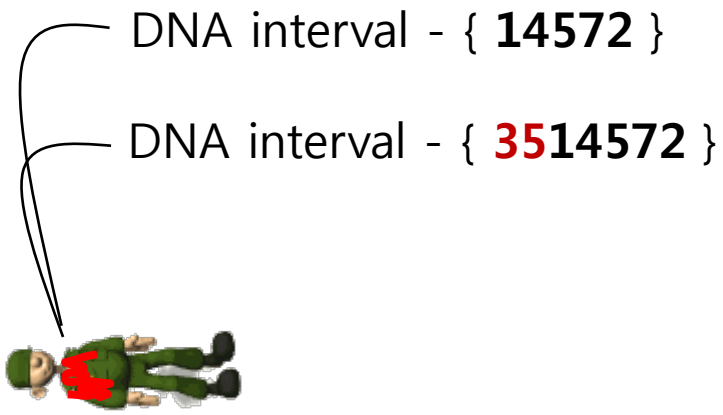




7.3 The substring problem for a database of patterns

- **In this case,**
  - One looks to see if the extracted and sequenced string is a substring of one of the strings in the database.



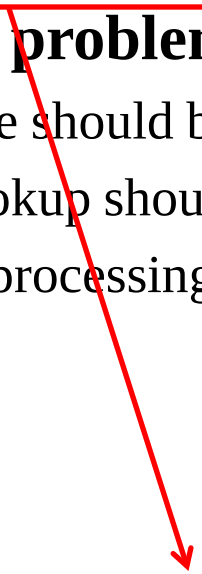


7.3 The substring problem for a database of patterns

- **Good data structure and lookup algorithm for the substring problem.**
  - Database should be stored in a small amount of space.
  - Each lookup should be fast.
  - The preprocessing of the database should be relatively fast.

7.3 The substring problem for a database of patterns

- **Good data structure and lookup algorithm for the substring problem.**
  - Database should be stored in a small amount of space.
  - Each lookup should be fast.
  - The preprocessing of the database should be relatively fast.



Suffix Tree

7.3 The substring problem for a database of patterns

- **Suffix trees yield a very attractive solution to this database problem.**
  - A generalized suffix tree *T* for the strings in the database is **built in**,
    - $O(m)$ time.
    - $O(m)$ space.
  - Any single string *S* of length n is **found** in the database, or declared not to be there,
    - $O(n)$ time.
  - The algorithm can **find all strings** in the database containing *S* as a substring.
    - $O(n+k)$ time.
      - *k* is the number of occurrences of the substring.

Contents

7.4 Longest common substring of two strings

- **The longest common substring problem.**
  - A classic problem in string analysis is to find the longest substring common to two given strings S_1 and S_2.

7.4 Longest common substring of two strings

- **The longest common substring problem.**
  - A classic problem in string analysis is to find the longest substring common to two given strings S_1 and S_2.

- **For example, if**
  - S_1 = s u p e r i o r c a l i f o r n i a l i v e s
  - S_2 = s e a l i v e r

7.4 Longest common substring of two strings

- **The longest common substring problem.**
  - A classic problem in string analysis is to find the longest substring common to two given strings S_1 and S_2.

- **For example, if**
  - S_1 = s u p e r i o r c a l i f o r n i **a l i v e** s
  - S_2 = s e **a l i v e** r
    - The longest common substring of S_1 and S_2 is **alive**.

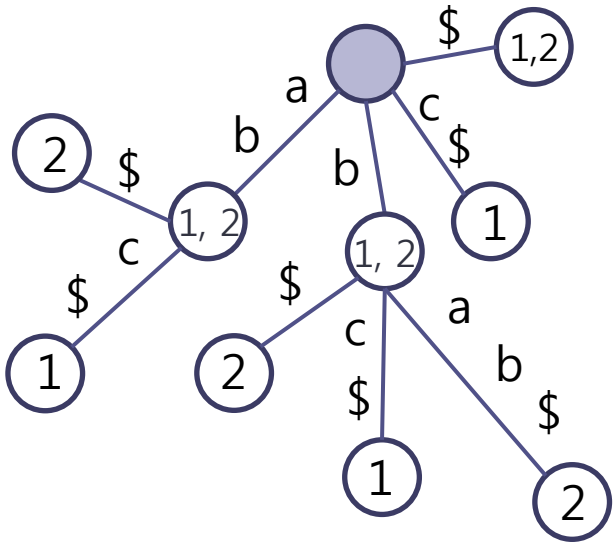
7.4 Longest common substring of two strings

- **An efficient and conceptually simple way to find a longest common substring is**
  - To build a ‘generalized suffix tree’.
    - Each leaf of the tree represents either
      - A suffix from one of the two strings or a suffix that occurs in both strings.

7.4 Longest common substring of two strings

- **For example)**

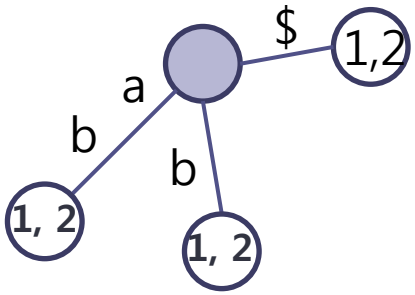
S_1 = a b c \$ S_2 = b a b \$



7.4 Longest common substring of two strings

- **For example)**

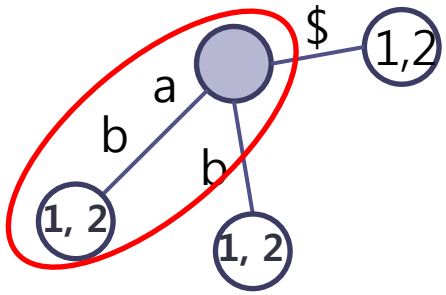
S_1 = a b c \$ S_2 = b a b \$



7.4 Longest common substring of two strings

- **For example)**

S_1 = a b c \$ S_2 = b a b \$



The longest common substring : ***ab***

7.4 Longest common substring of two strings

- **Construction of the suffix tree can be done in linear time.**
  - Proportional to the total length of S_1 and S_2.

- **The node markings and calculations of string-depth can be done by standard linear-time tree traversal methods.**