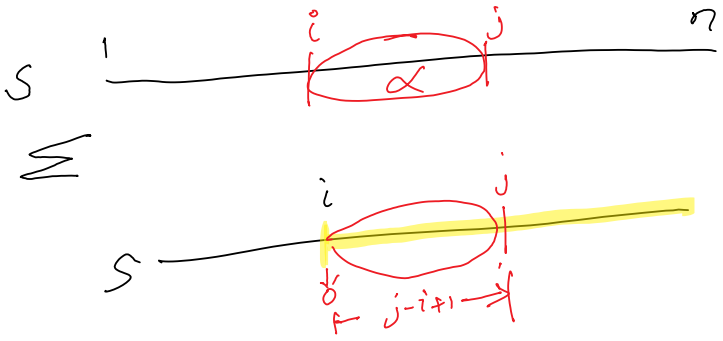
Suffix Tree Applications

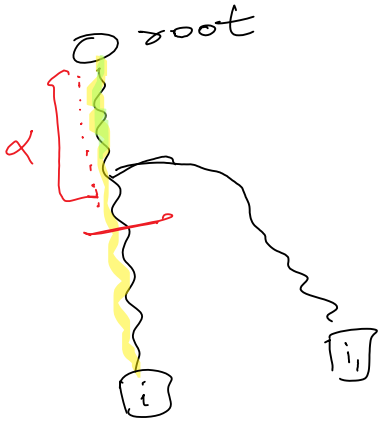
Monday, March 15, 2021 10:42 AM



Suffix Trees

1) Compacted trie of all suffixes in S

2) Storing index for all substrings in S.

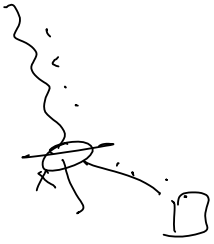


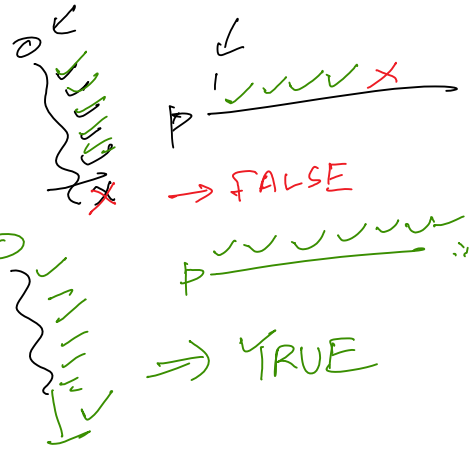
Every substring α of S, will have a unique path from the root $\Rightarrow$ "index".

FindPath

"write" mode → ST construction

"read mode" → read-only ST





Pattern Matching

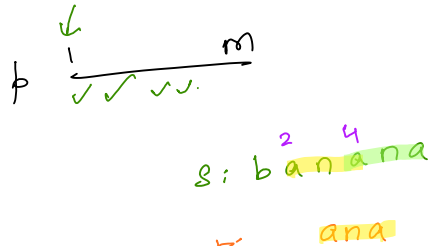
Monday, March 15, 2021 11:01 AM

(practical assumption: $n >> m$)

Problem: Given a string $S[1..n]$ and a query pattern $p[1..m]$, find if pattern p occurs as a substring in S.

Input: $S[1..n]$, $p[1..m]$

Output: i) Yes or No (decision problem) → yes
ii) starting indices (location) → {2, 4}



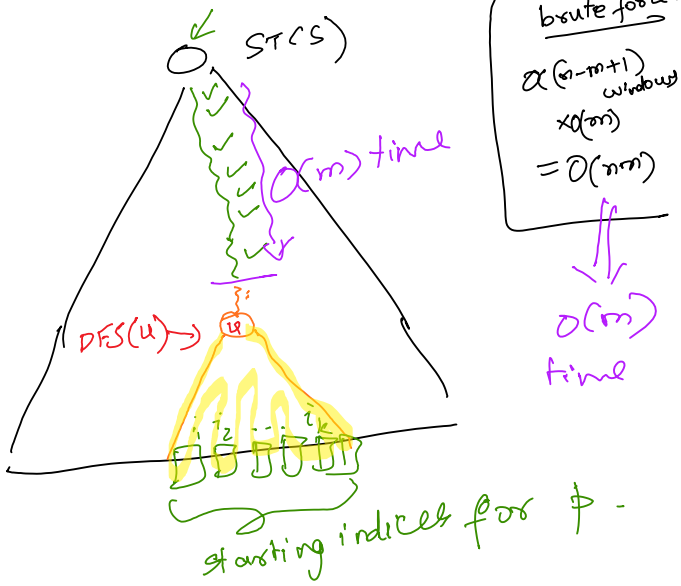
Algorithm:

1) Build ST(S) ⇒ O(n) time ← preprocessing

2) Find p in S: ⇒ O(m) time

FindPath(root, $p[1..m]$)
"read-only" mode

- if p is exhausted ⇒ return FOUND
- else // mismatch ⇒ return NOTFOUND



$O(n + m)$
preprocessing cost
querying cost

3) Go down to the next internal node (u) ⇒ ∝ size of the subtree under u. ⇒ $O(k)$ time
& enumerate all leaves (suffids) ~~for~~ under u.
output size

Q) Try to prove that this algorithm is correct: a) if p is in S, ⇒ algo will find it.
b) if algo outputs Yes/No ⇒ p is in S (or not)

Longest Repeat

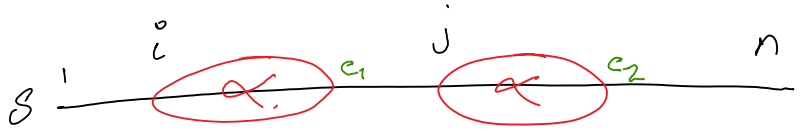
Monday, March 15, 2021 10:44 AM

Problem: Given a string S of length n (over alphabet Σ), find a longest repeat in S.

Definition: A "repeat" in a string S is a substring that occurs atleast twice in S.

Input: $S[1..n]$

Output: longest Repeat α.



$i \neq j$

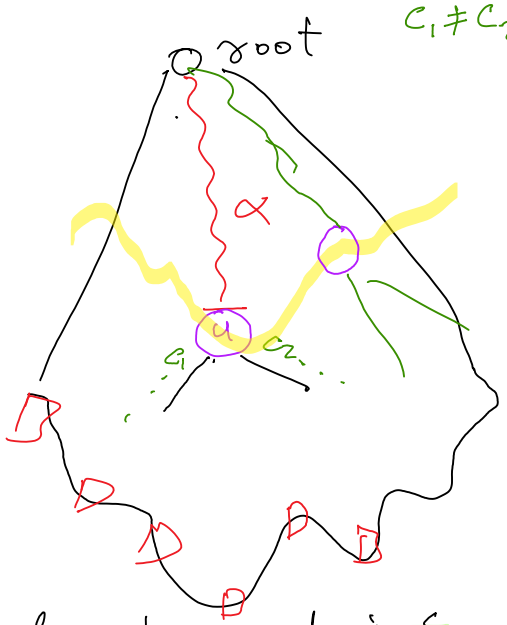
$c_1 \neq c_2$

Algorithm:

1) Build $ST(S)$

2) a) Locate the "deepest" internal node (u)
   ↗ by string-depth

   b) Output path-label(u)



□

Proof of correctness:

If algorithm outputs α
$\Rightarrow$ α is a pathlabel of u
$\Rightarrow$ α is a repeat $\longrightarrow$ (1)
Since u is the deepest
$\Rightarrow$ α is a longest repeat

□

If α is a longest repeat in S
$\Rightarrow$ it is present in at least 2 places i & j s.t. $c_1 \neq c_2$

$\Rightarrow$ There has to exist an internal node u at the end of the path labelled α
$\Rightarrow$ u is also the deepest (since α is the longest)
$\Rightarrow$ Algorithm will output α.

□

Suffix-Prefix Match

Monday, March 15, 2021 10:50 AM

A "superstring" is a string that contains other strings.

Problem: Shortest Superstring Problem ⇒ Suffix-Prefix Exact Matching

Algo: O(m+n) time

Input: $S_1[1..m]$, $S_2[1..n]$

Output: Find the longest suffix-prefix matching between S_1 & S_2.

Algo: // first compute superstring for case (S_1 on left)

1) Build ST(S_1)
2) Do FindPath(root, $S_2[1..n]$) going as deep as possible until first mismatch
3) If matching stopped at a $-leaf
   - let i ← label of the $-leaf
   - superstring$_{1,2}$ = $S_1[1..m]$ & $S_2[m-i+2..n]$

// Next compute for case S_2 on left

4) Build ST(S_2)
5) Repeat steps 2 & 3 for (S_1 on right)
6) |superstring$_{1,2}$| vs. |superstring$_{2,1}$|

Shorter of the two ⇒ superstring

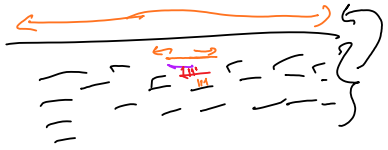
if no superstring way found ⇒ Not found

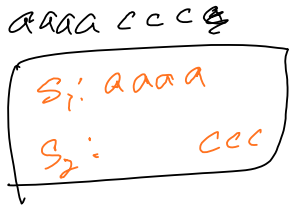
e.g.

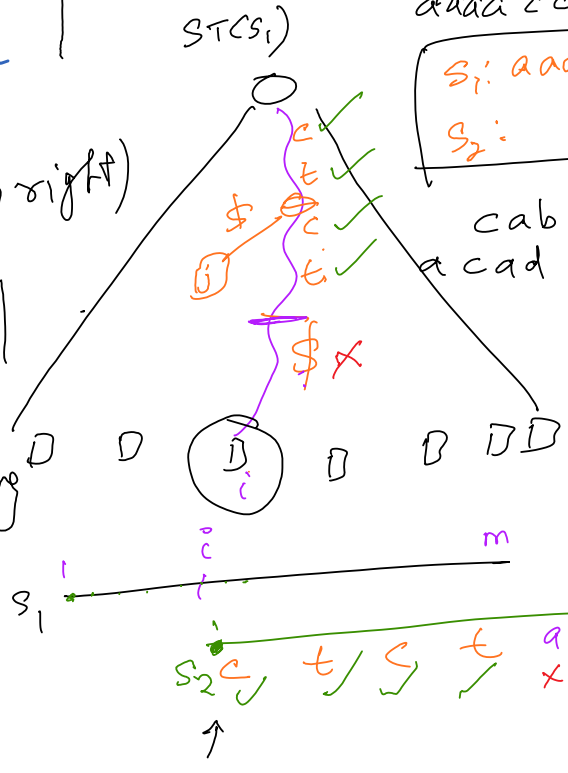
S_1: a c g c t c t c t $ - - -
S_2: - - - - - c t c t a g c

⇒ shortest superstring = a c g c t c t c t a g c

⇒ output <S_1, i>





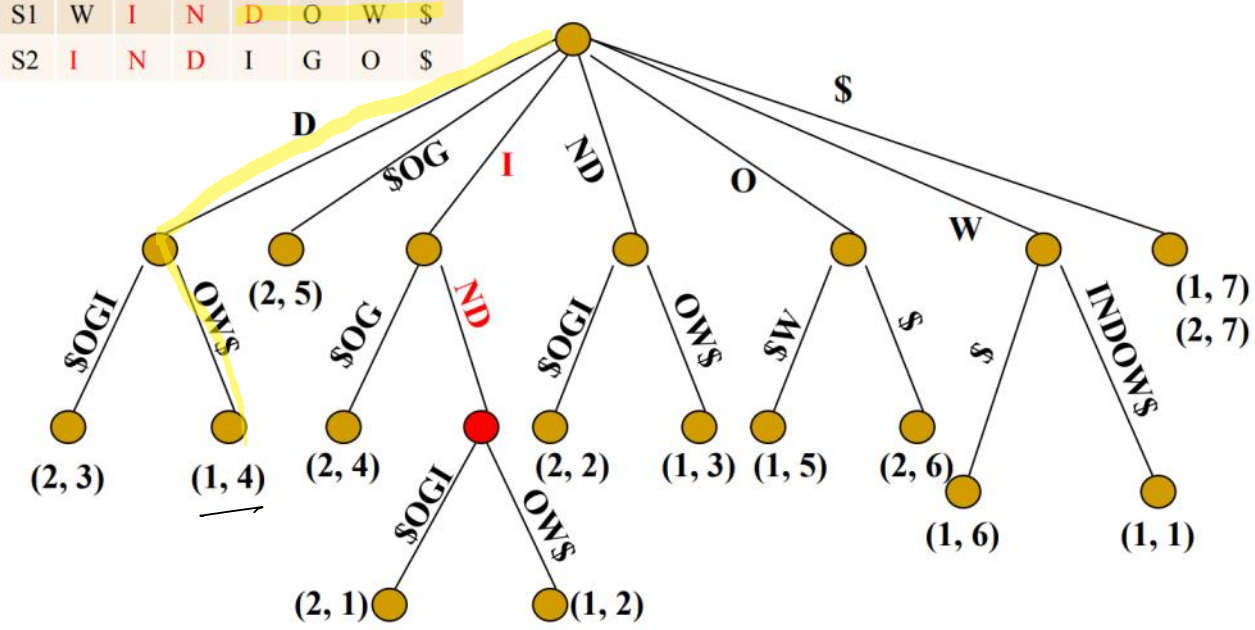


Generalized Suffix Trees (GST)

Monday, March 15, 2021 10:44 AM

Definition: Given a set $S = \{S_1, S_2, \ldots S_K\}$ of K strings, the Generalized Suffix Tree (GST) of S is the compacted trie of all suffixes of all the strings in S.

	1	2	3	4	5	6	7
S1	W	I	N	D	O	W	$
S2	I	N	D	I	G	O	$



Leaf label format:
<string id, suffix id>

Longest Common Substring (LCS)

Monday, March 15, 2021 10:45 AM

Problem: Given K strings, find an lcs between every pair of strings.

Input: $\mathcal{S} = \{S_1, S_2, \ldots S_K\}$ of lengths $\{n_1, n_2, \ldots n_K\}$ resp.

Output: For each pair $\langle S_i, S_j \rangle$ find lcs.

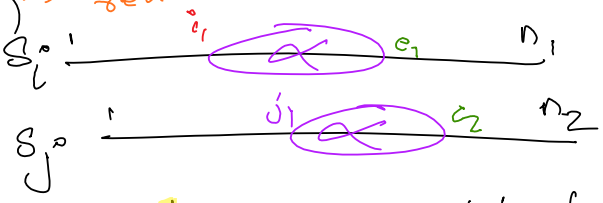
$$N = \sum_{i=1}^{K} |S_i|$$

Algo:

1) Build $GST(\{S_1, S_2 \ldots S_K\})$ $\Rightarrow O(N)$ time

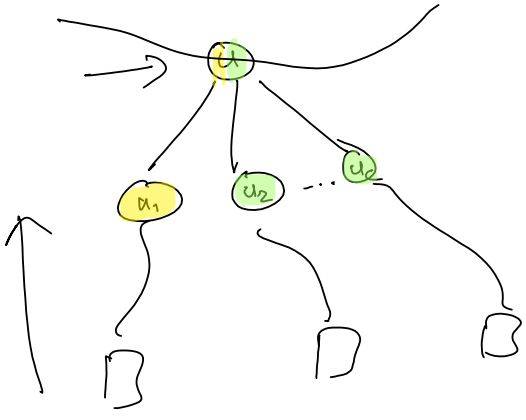
$\Rightarrow O(N)$ → can it be reduced?

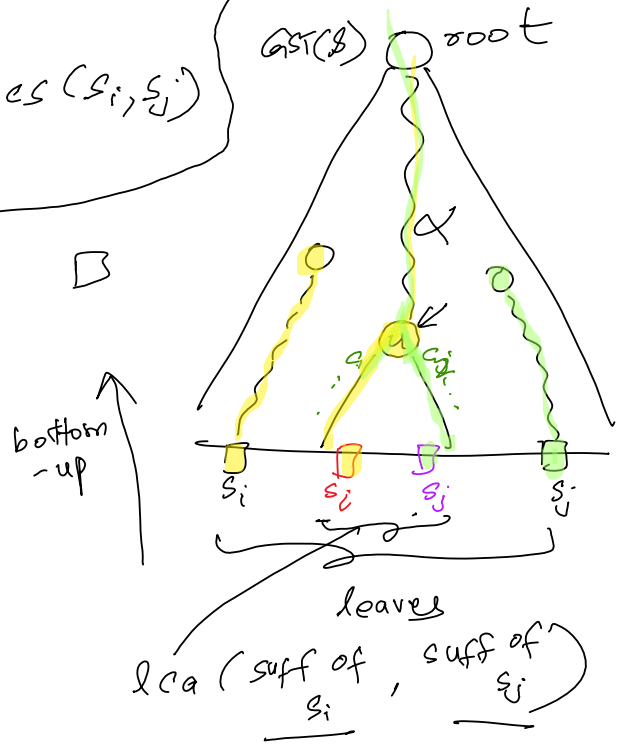
2) For each $\langle S_i, S_j \rangle \in \mathcal{S}$



a) find the deepest internal node u that has at least one leaf from S_i & one leaf from S_j under its subtree.

b) Report pathlabel(u) → lcs(S_i, S_j)





LCP problem revisited, using LCA

Friday, March 19, 2021 10:54 AM

Lowest Common Ancestor

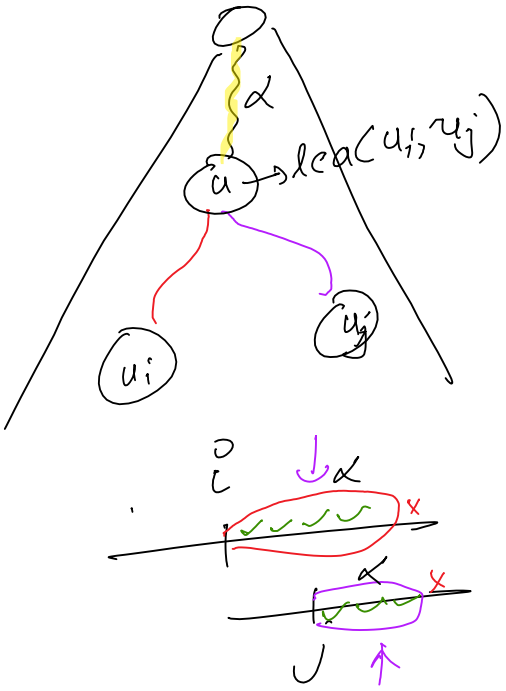
LCA (u_i, u_j) ⇒ lca Function

lcp ⟺ lca

↓ ↓

longest common prefix (suff$_i$, suff$_j$)

lowest common ancestor (u_i, u_j) in the tree



String ⟹ reduces to tree

lca = ? O(1)

Suffix-Prefix Match (revisited) → but using GSTs

Monday, March 15, 2021 10:50 AM

Problem: Shortest Superstring Problem ⇒ Suffix-Prefix Exact Matching

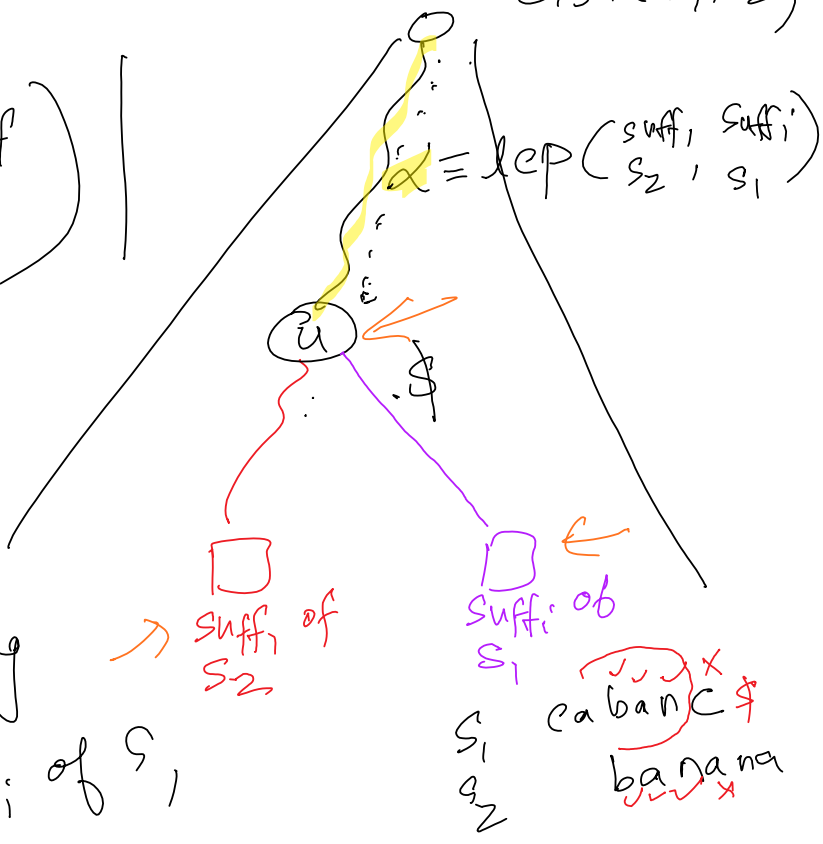
Input: $S_1[1..m]$, $S_2[1..n]$

Output: Find the longest suffix-prefix matching between S_1 & S_2.

Algo:

1. Build GST(S_1, S_2)

2. $l = |lcp(\text{suff}_1 \text{ of } S_2, \text{suff}_i \text{ of } S_1)|$

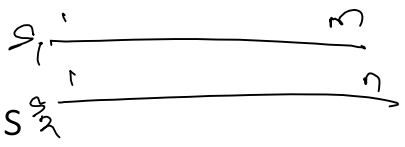


if ($l == m-i+1$)

Yes → superstring exists $S_1 \rightarrow S_2$ ⇓ involves suff$_i$

NO → no superstring exist from suff$_i$ of S_1

Suffix-Prefix Match - revisited using LCA queries



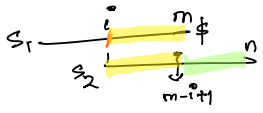
Monday, March 22, 2021 10:52 AM

$lcp(suff_i, suff_j) \equiv lca(leaf_i, leaf_j)$

ALGORITHM:

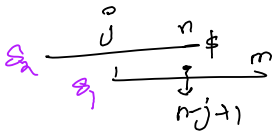
Build GST(S_1, S_2)

// check if there exists a superstring with (S_1 on left)




```
for (i = 1 to m)
    l ← lcp(suff_i of S1, suff_1 of S2)
    check if (l == m-i+1)                ← S1[1..m] . S2[m-i+2..n]
        YES : super_1,2 ← S1[1..m] . S2[m-i+2..n]
              break;
        NO  : continue;
    If super_1,2 not found inside the loop,
    then super_1,2 ← S1[1..m] . S2[1..n]   // set trivial
```

// check if there exists a superstring with (S_2 on the left)




```
for (j = 1 to n)
    l ← lcp(suff_j of S2, suff_1 of S1)
    check if (l == n-j+1)                ← S2[1..n] . S1[n-j+2..m]
        YES : super_2,1 ← S2[1..n] . S1[n-j+2..m]
              break;
        NO  : continue;
    If super_2,1 not found inside the loop,
    then super_2,1 ← S2[1..n] . S1[1..m]   // set trivial
```

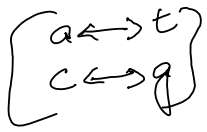
// Report the shortest

superstring ← shortest { $super_{1,2}$, $super_{2,1}$ }

B.

Restriction Enzymes

Monday, March 15, 2021 10:50 AM



cut site

Restriction enzymes cut at sites that have DNA palindromes.

DNA:

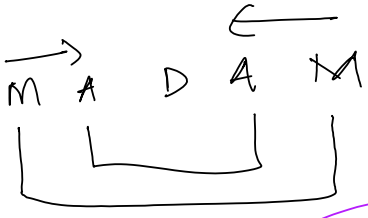
palindrome

madam

racecar

malayalam

ABBA



Definition: A "DNA palindrome" is a sequence S of some length l, such that:

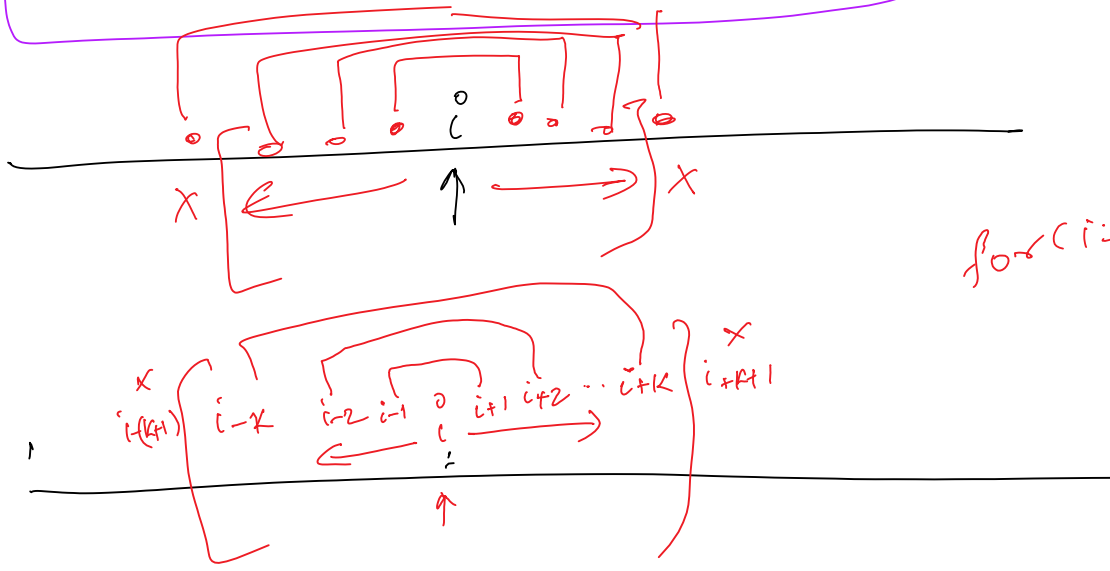
$S[i+1] = \text{rev.comp}(S[l-i])$, $\forall 0 \le i \le \lfloor \frac{l}{2} \rfloor$

e.g. S = caagtacttg

e.g. S = caagccttg

I have a racecar that is fast.

Build GST(S, S^r)



for (i: 1 to n)

RE Site Detection: Algorithm

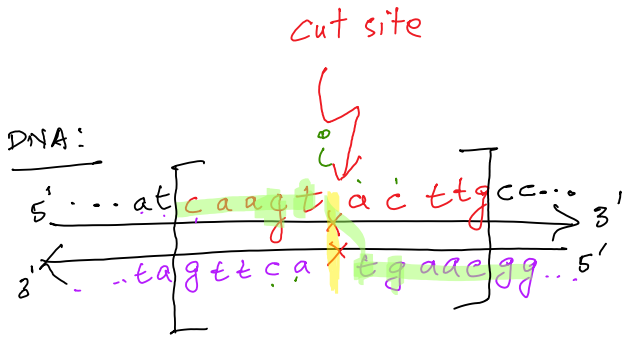
Wednesday, March 24, 2021 10:51 AM

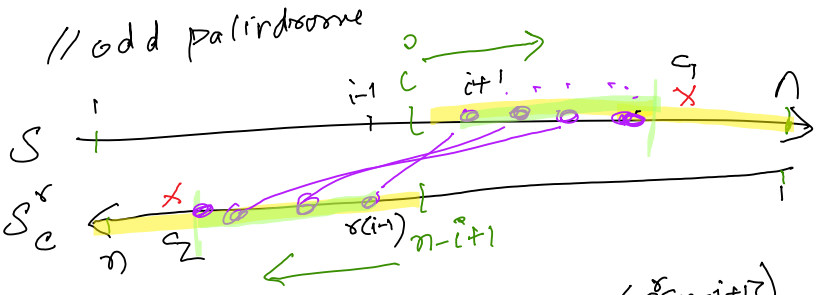
Input: A genomic sequence $S[1..n]$.

Output: All palindromic sites of length $\geq \bar{c}_{min}$

Algorithm:

1) Build GST $(S, S^{\bar{r}}_c)$ → $O(n)$ time





// odd palindrome case.

2) for (i = 1 to n) { → $O(n)$ time

a. $u \leftarrow lca(\langle S, i+1\rangle, \langle S^{\bar{r}}_c, r(i-1)\rangle)$ → $O(1)$ time

b. lcp ← pathlabel(u)

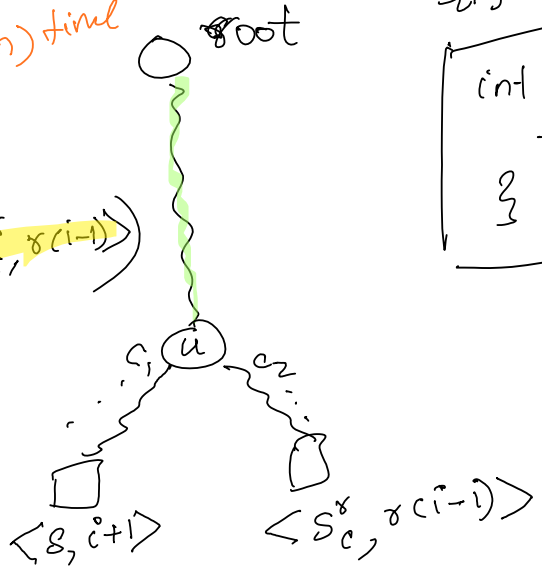
c. if ($|lcp| \geq \bar{c}_{min}$) {

print "odd palindrome centered at i exists"

palindrome = $S[i+1..\ i+|lcp|-1]$

}

}



$S[i] = comp(S^{\bar{r}}_c[n-i+1])$

```
int r(i) {
    return (n-i+1)
}
```

⇒ $O(n)$ time total

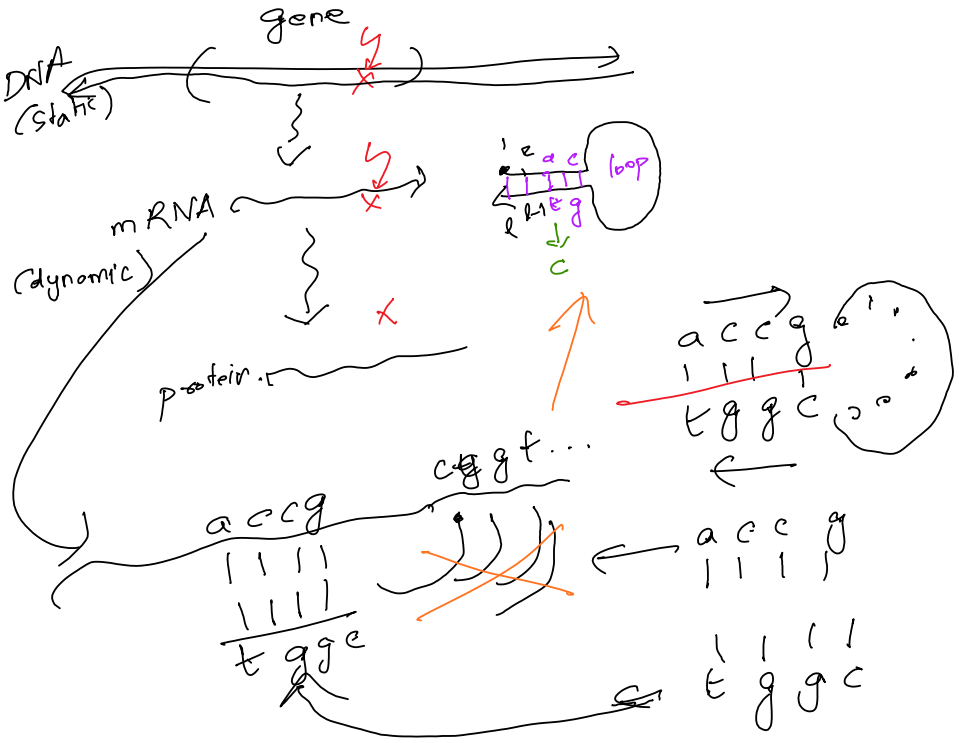
// do similar check for even length palindromic case.

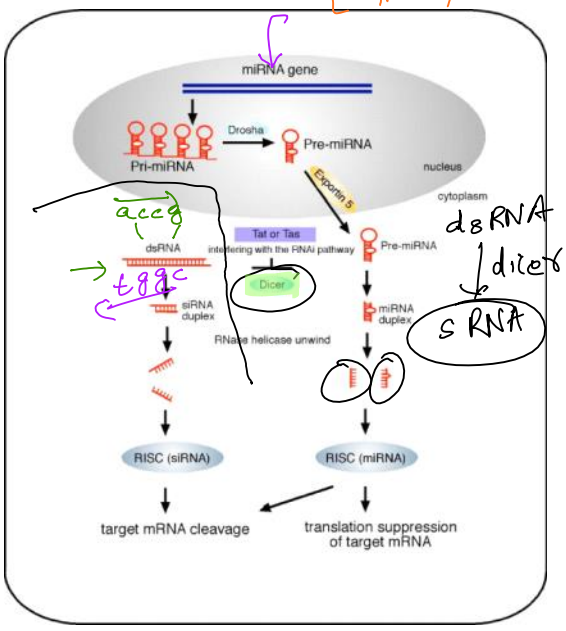
Gene Silencing

Wednesday, March 17, 2021 10:53 AM

RNA interference: Nobel prize, 2006 (Fire & Mello)
(discovery)

Q) How to "silence" a gene?





- DOI: 10.1186/1742-4690-2-35