
Lowest Common Ancestor (LCA) Queries

A technique with application to approximate matching

Chris Lewis

Approximate Matching

- Match pattern to text
 - Insertion/Deletion/Substitution
 - Applications
 - Bioinformatics, Text Retrieval, Signal Recognition
 - Limited number of mismatches
 - Index methods tend to be poor performers
-

Online algorithms

- **Dynamic Programming**
 - $O(kn)$ time & $O(m)$ space
 - **Automata**
 - $O(n)$ time & exponential space
 - **Bit parallelism**
 - Exploit parallel nature of bitwise operations
 - **Filtering algorithms**
 - Reject impossible regions
-

Dynamic Programming - alignment

- Row 1 - Col 1 with 0
- Fill using Relation:

$$M_{i,j} = \text{MAX} ($$

$$M_{i-1, j-1} + S_{i,j},$$

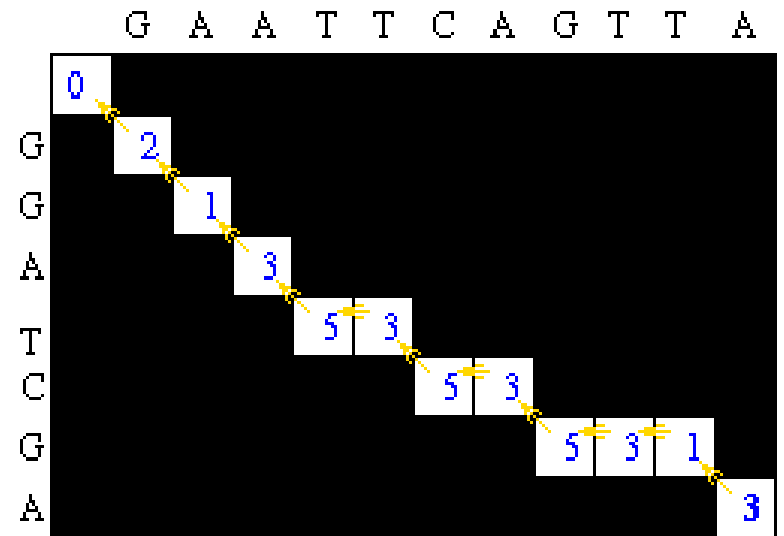
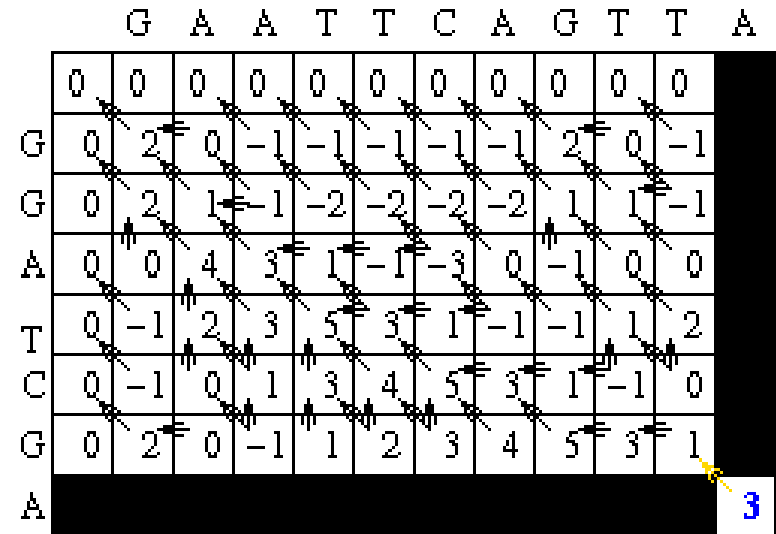
$$M_{i,j-1} + w,$$

$$M_{i-1,j} + w$$

$$)$$

- Trace-back produces

G	A	A	T	T	C	A	G	T	T	A
G	G	A	T	_	C	_	G	_	_	A

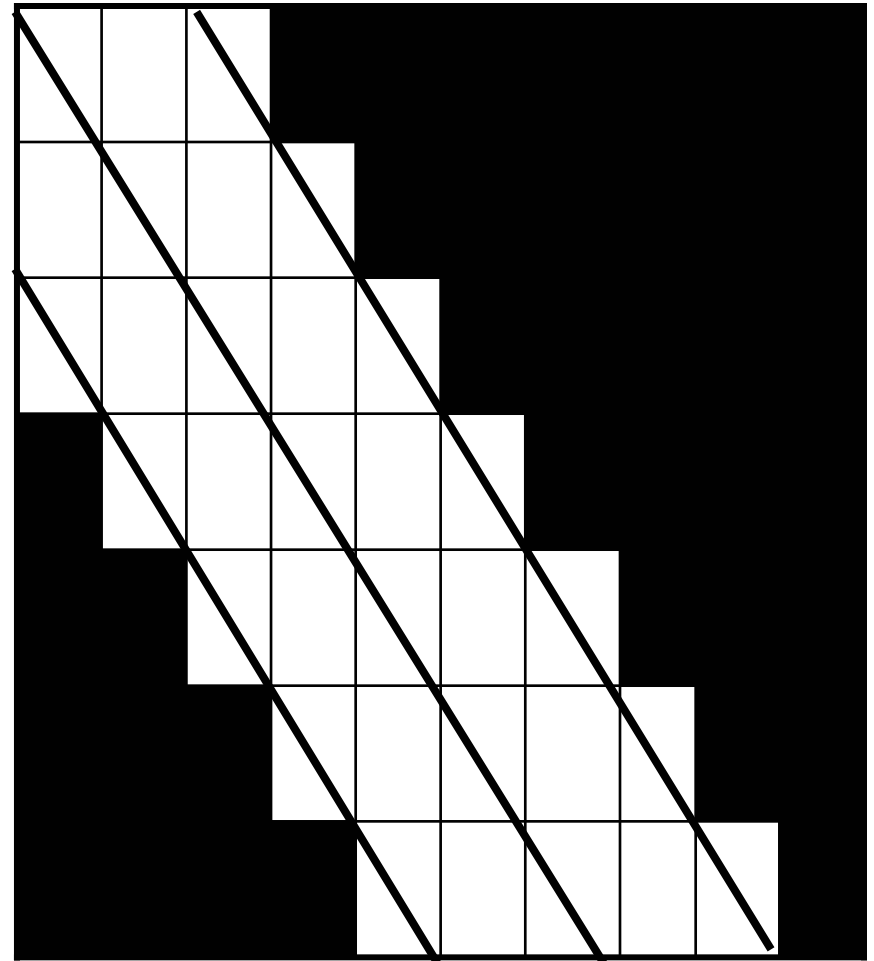


Dynamic Programming – k difference alignment

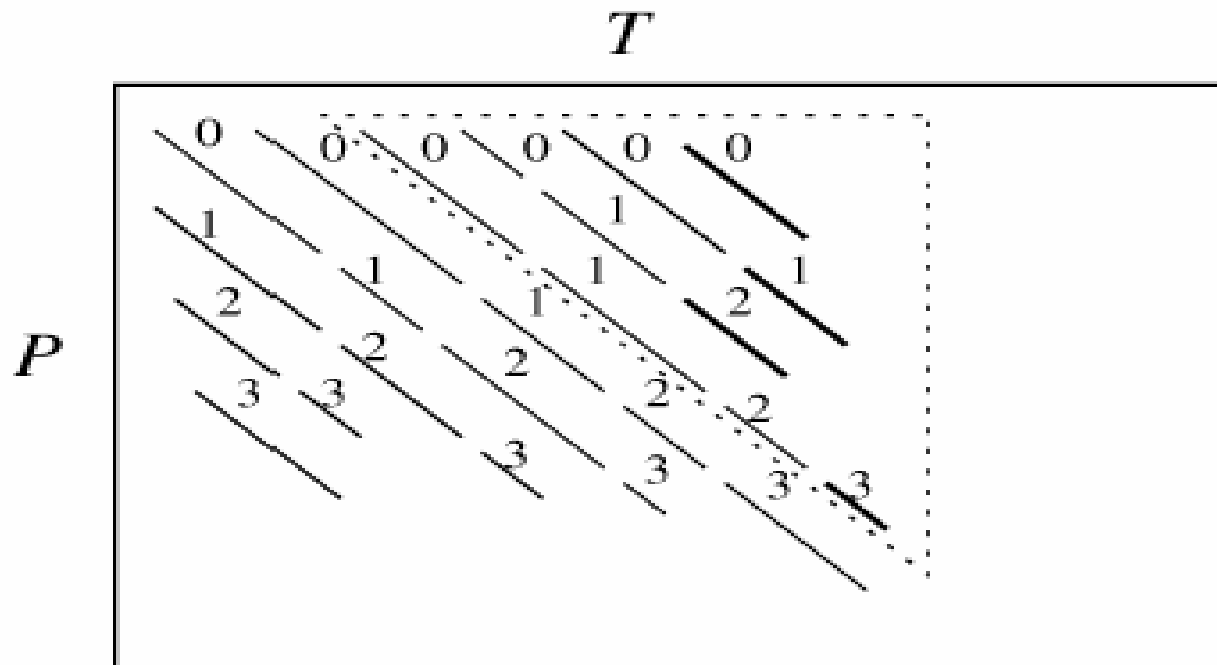
Key observation

- More than k spaces off the diagonal doesn't matter
- Horizontal -> space
- Vertical -> space
- Path will exist within $2k+1$ strip

Doesn't (quite) hold
when matching



Dynamic Programming – k difference inexact matching



- Find $O(kn)$ strokes
- $O(1)$ time per stroke

Finding the length of the stroke

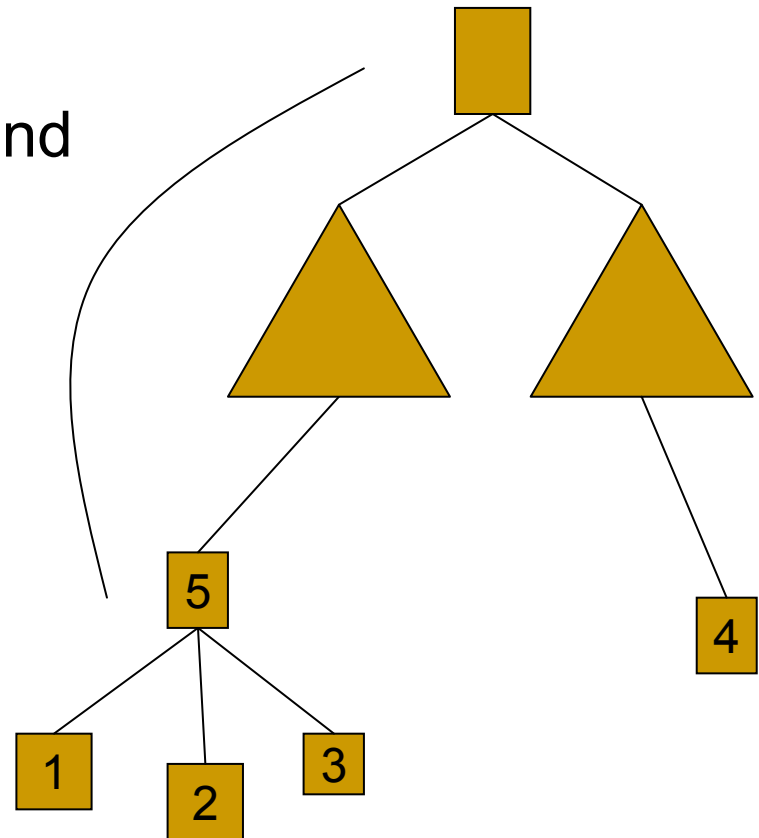
- How far can we extend from $P[i]$ and $T[j]$ until we encounter a mismatch?
 - Longest Common Extension Query
 - Create a suffix tree of P and T
 - Prefix from $P[j..n]$ common to prefix in $T[j..m]$
 - Relies on the Lowest Common Ancestor Method
-

Lowest Common Ancestor \approx Longest Common Extension

Lowest Common Ancestor of 1 and 2 is 5.

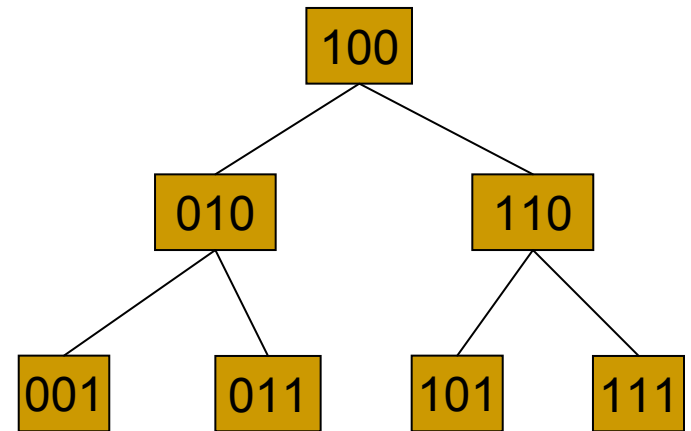
Label from root to 5 is
common to nodes 1,2 (and 3).

Length common label =
Longest Common Extension =
String depth LCA



Lowest Common Ancestor Algorithm

- Consider a complete binary tree with inorder numbering
 - Path number
- These numbers trace path to node
 - 1 right, 0 left
- XOR of two numbers
 - Left most 1 indicates first difference in path

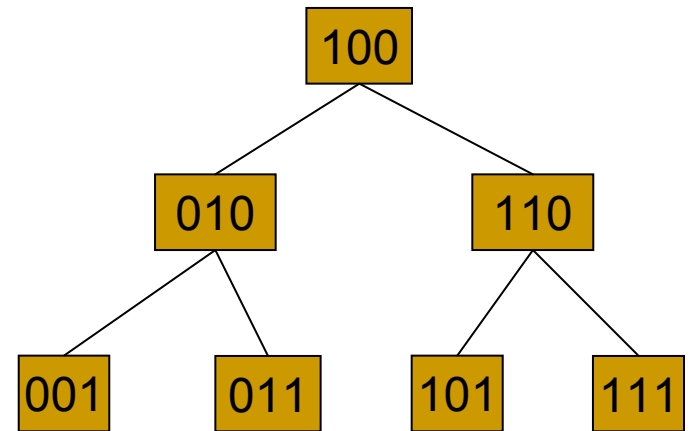


$$101 \wedge 111 = 010$$

2nd branch different

LCA Algorithm Continued

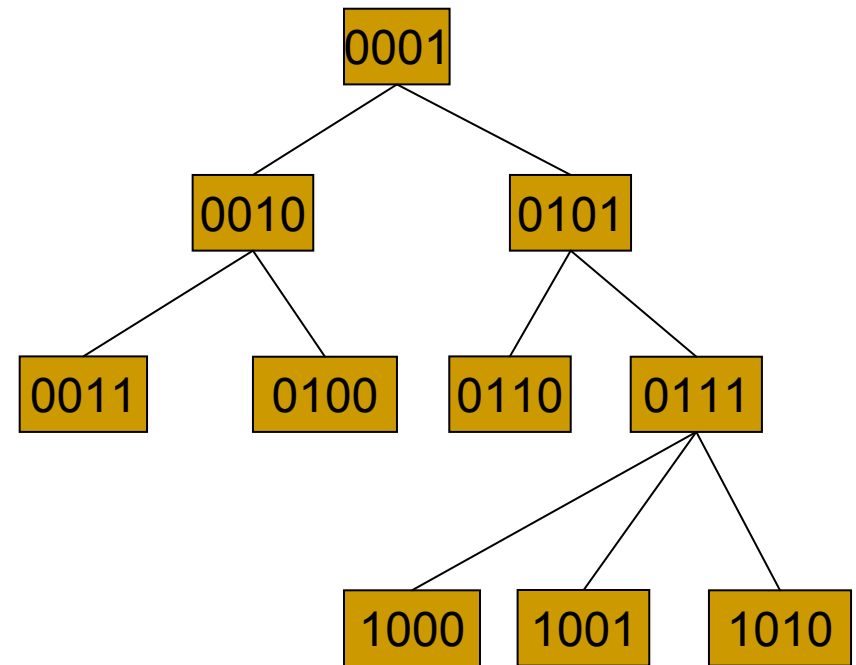
- First difference bit $k=2$
- $k-1 = 1$ bit(s) the same
- 1st bit in both 101 and 111 is 1
- How do we turn this into a node number?
- Add a 1 and sufficient 0's to give us a $\log_2 p$ number
- Node: 110 = lca



Example $001 \wedge 111$ on board.

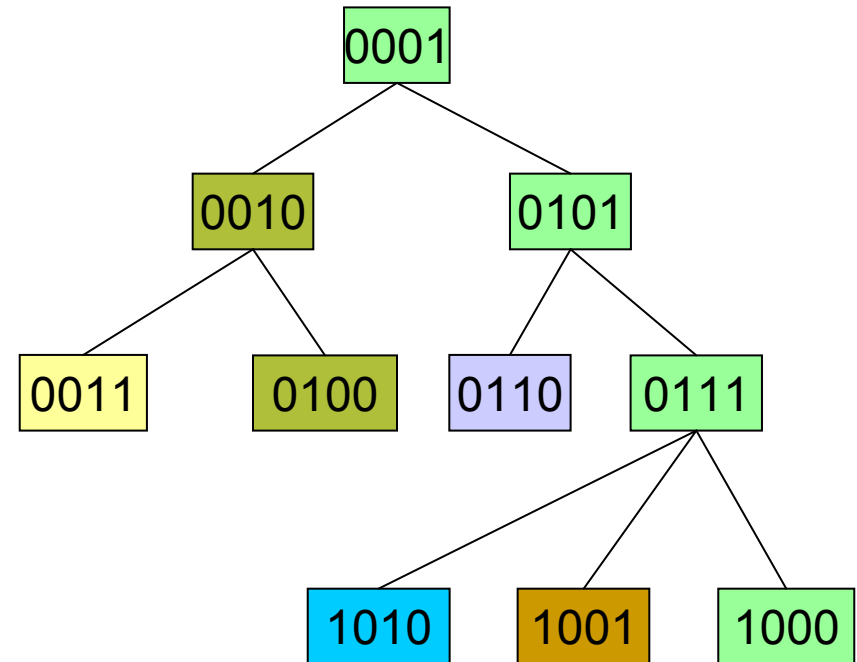
How about general trees

- Map the general tree to a binary tree $O(n)$
- Pre-order node numbering
- $h(j)$
 - position from right of least significant 1-bit
- $l(v)$
 - node w for which $h(w)$ is max in subtree of v



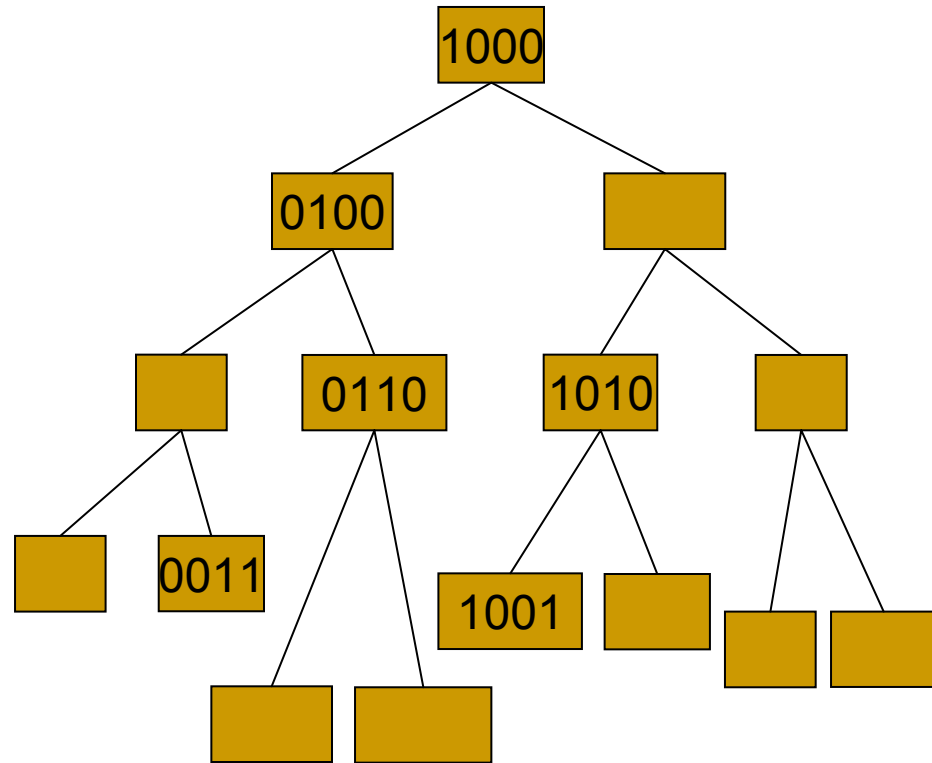
Partition general tree into runs

- This allows nodes to be partitioned wrt $l(v)$.
- This creates runs in the tree
 - Maximal set of nodes which all have same l value.
- Maintain (just) enough ancestry information.



Tree Map

- Having created runs in the general tree, we can map every node v to node $l(v)$ in the binary tree.



Preprocessing algorithm

- Depth first traversal of tree
 - assign pre-order number to the nodes
- compute $I(v)$ for each v , set $L(k)$ to point to the head of the run
- v is head of its run if the I value of v 's parent is not $I(v)$.
 - head of a run containing v can be located $O(1)$
 - Lookup $I(v)$ then lookup $L(I(v))$
- map each v in the general tree to $I(v)$ in the binary tree
- For each v create an $O(\log n)$ bit number A_v .
 - Bit $A_v(i) = 1$ iff v has ancestor in that maps to height i in the binary tree. i.e. iff v has an ancestor u such that $h(I(u)) = i$

Constant time lca algorithm

- Find the lowest common ancestor b in the binary tree of nodes $l(x)$ and $l(y)$.
- Find the least significant position $j \geq h(b)$ such that both numbers A_x and A_y have 1-bits in position j . Thus $j = h(l(z))$
- Find Node x' closest to x on run with z
 - Details on following page
- Find Node y' closest to y on run with z
- If $x' < y'$ $z = x'$, else $z = y'$

Node x' closest to x

- Find the position l of the right-most 1 bit in A_x
- If $l = j$, then set $x' = x$ and go to step 4.
- Find the position k of the left-most 1-bit in A_x that is to the right of position j . Form the number consisting of the bits of $l(x)$ to the left of the position k , followed by a 1-bit in position k , followed by all zeros. {That number will be $l(w)$ }
- Look up node $L(l(w)) = w$. Set node x' to be the parent of node w in the general tree.

Example

Questions

Online Algorithms

- **Dynamic Programming**
 - $O(kn)$ time & $O(m)$ space
 - **Automata**
 - $O(n)$ time & exponential space
 - **Bit parallelism**
 - Exploit parallel nature of bitwise operations
 - **Filtering algorithms**
 - Reject impossible regions
-

Automata

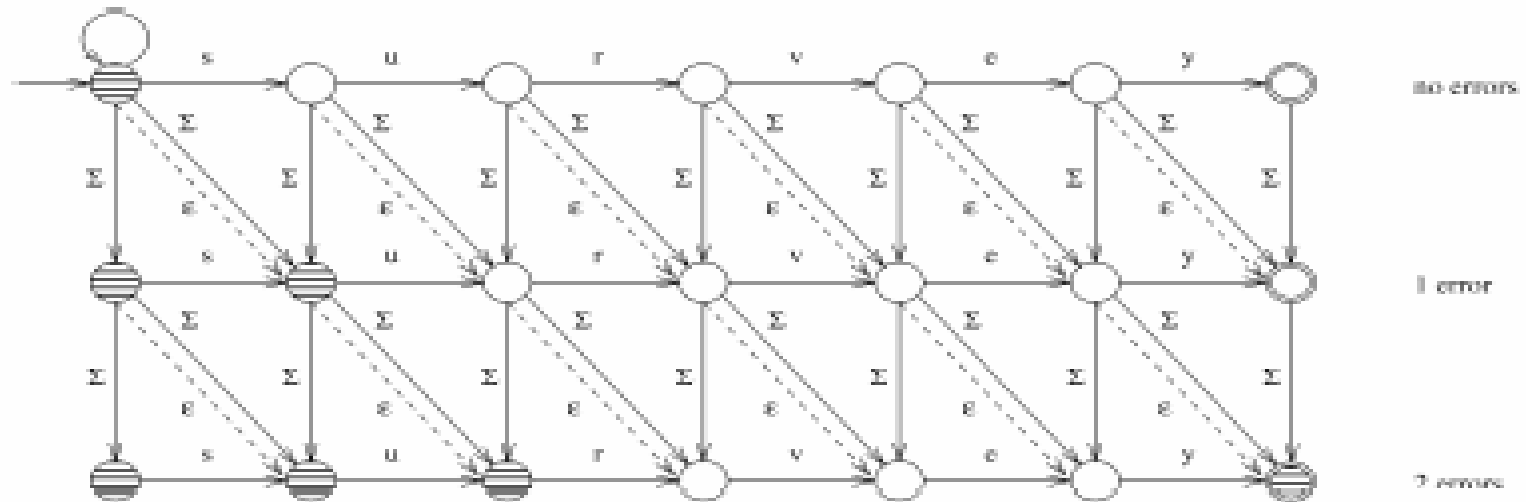


Fig. 15. An NFA for approximate string matching of the pattern "survey" with two errors. The shaded states are those active after reading the text "surgery".

- Space efficient Automata as NFA
- “Lazy” construction allows $O(mn)$ space

Bit Parallelism

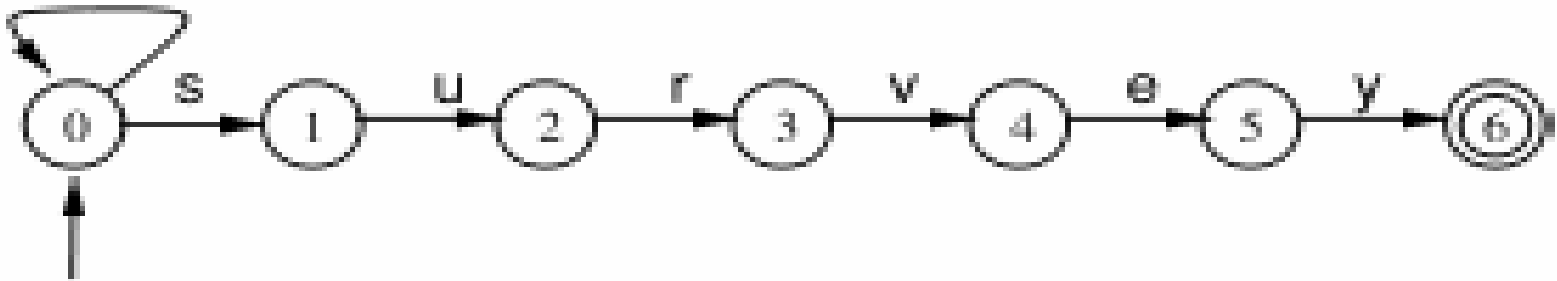


Fig. 18. Nondeterministic automaton that searches "survey" exactly.

- Non-parallel - $O(mn)$ & $O(kmn)$ time
- Bit-parallel - $O(mn/w)$ & $O(kmn/w)$ time
- Table of bitmasks for text
- Machine word track state of machine

Filtering Algorithms

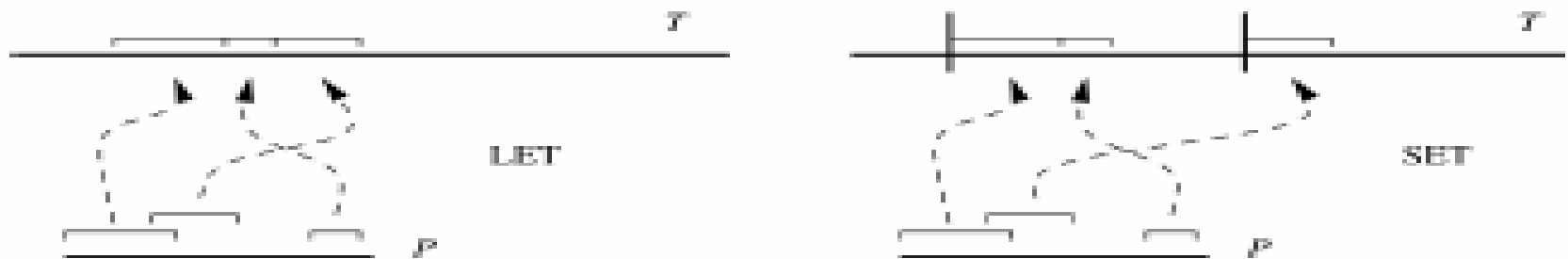


Fig. 22. Algorithms LET and SET. LET covers all the text with pattern substrings, while SET works only at block beginnings and stops when it finds a difference.

- Scan text looking for longest matches
- less than $m-k$ coverage \Rightarrow no match
- Verify with non-filtering algorithm
- $O(n)$ filter, $O(kn)$ verify