DNA Sequence Reconstruction

- DNA can only be sequenced in relatively small pieces, up to about 1,000 nucleotides.
- By chemistry a much longer DNA sequence can be broken up into overlapping sequences called clones. Clones are 10's of thousands of nucleotides long.

Contiguous Ordering - PQ Trees

Course Summary

Tagging the Clones

- By chemistry the clones can be tagged by identifying a region of the DNA uniquely.
- Each clone is then tagged correspondingly.

Problem to Solve

- Given a set of tagged clones, find a consistent ordering of the tags that determines a possible ordering of the DNA molecule.

Contiguous Ordering Solutions

Contiguous ordering problem

Solution

Alternate Solutions

Linear Time Algorithm

- Booth and Lueker, 1976, designed an algorithm that runs in time $O(n+m+s)$.
- It requires a novel data structure called the PQ tree that represents a set of orderings.
- PQ trees can also be used to test whether an undirected graph is planar.
PQ Trees

- PQ trees are built from three types of nodes

<table>
<thead>
<tr>
<th>P node</th>
<th>Q node</th>
<th>leaf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Children can be reordered.</td>
<td>Children can be reversed.</td>
<td>Each leaf has a unique label.</td>
</tr>
</tbody>
</table>

Example PQ-Tree

T

The frontier of T defines the ordering \( F(T) = FCABDE \), just read the leaves left to right.

T' is equivalent to T if T can be transformed into T by reordering the children of P nodes and reversing the children of Q nodes.

Equivalent PQ Trees

T

FCABDE

T'

FEBDAC

Orderings Defined by a PQ Tree

- Given a PQ tree T the orderings defined by T is

\[ \text{PQ}(T) = \{ F(T') : \text{T'} \text{ is equivalent to } T \} \]

There are \( 6 \times 2 \times 2 = 24 \) distinct orderings in \( \text{PQ}(T) \).

Generally, if a PQ tree T has \( q \) Q node and \( p \) P nodes with number of children \( c_1, c_2, \ldots, c_p \), then the number of orderings in \( \text{PQ}(T) \) is

\[ 2^q \prod_{i=1}^{p} c_i! \]

\( n! = 1 \times 2 \times \ldots \times n \)

Example PQ Tree Solution for the Contiguous Ordering Problem

- Input: A universe \( U \) and a set \( S = \{ S_1, S_2, \ldots, S_m \} \) of subsets of \( U \).
- Output: A PQ tree T with leaves U with the property that \( \text{PQ}(T) \) is the set of all orderings of U where each set in S is contiguous in the ordering.

Example Solution

\( U = \{A, B, C, D, E, F\} \)

\( S = \{\{A, C, D\}, \{A, C, F\}, \{B, D, E\}\} \)

There are 8 orderings that are possible in keeping each of these sets contiguous.
PQ Tree Restriction

- Let \( U = \{A_1, A_2, \ldots, A_n\} \), \( S = \{A_1, A_2, \ldots, A_k\} \), and \( T \) a PQ tree.
- We will define a function Restrict with the following properties:
  - \( \text{Restrict}(T, S) \) is a PQ tree.
  - \( \text{PQ}(\text{Restrict}(T, S)) = \text{PQ}(T) \cap \text{PQ}(T') \) where

High Level PQ tree Algorithm

- Input is \( U = \{A_1, A_2, \ldots, A_n\} \), and subsets \( S_1, S_2, \ldots, S_m \) of \( U \).
- Initialization:
  - \( T = \text{P node with children } A_1, A_2, \ldots, A_n \)
- Calculate \( m \) restrictions:
  - for \( j = 1 \) to \( m \)
    - \( T := \text{Restrict}(T, S_j) \)
- At the end of iteration \( k \):
  - \( \text{PQ}(T) = \text{the set of ordering of } U \) where each set \( S_1, S_2, \ldots, S_k \) are contiguous.

Marking Nodes

- Given a set \( S \) and PQ tree \( T \) we can mark nodes either full or partial.
  - A leaf is full if it is a member of \( S \).
  - A node is full if all its children are full.
  - A node is partial if either it has both full and non-full children or it has a partial child.
  - A node is doubly partial if it has two partial children.

Marks of Nodes

Mark the leaves in \( S \) full.
Bottom up mark the nodes full or partial.
The members of \( S \) will become contiguous.

Restrict\((T, S)\)

- Mark the full and partial nodes from the bottom up.
  - In the process the marked leaves become contiguous.
- Locate the key node.
  - Deepest node with the property that all its proper ancestors have exactly one partial child.
- Restrict the key node.
  - In the process of restricting the key node we will have to recursively direct partial nodes.
  - Directing a node returns a sequence of nodes.
Restricting a P Node with Partial Children

Restrict a P node

left partial

right partial

full

direct then attach

Directing a P Node

left direct a P node

Restricting a P node with no Partial Children

Restrict a P node

full

Restricting a Q node

left direct a Q node

Directing a Q Node

Example (1)

U = \{A,B,C,D,E,F,G,H,I,J\}
S_1 = \{A,C,E,G,I\}

mark

CSE 589 - Lecture 19 - Spring 1999
19

CSE 589 - Lecture 19 - Spring 1999
20

CSE 589 - Lecture 19 - Spring 1999
21

CSE 589 - Lecture 19 - Spring 1999
22

CSE 589 - Lecture 19 - Spring 1999
23

CSE 589 - Lecture 19 - Spring 1999
24
Example (2)

\[ U = \{A, B, C, D, E, F, G, H, I, J\} \]
\[ S_1 = \{A, C, E, G, I\} \]

special case because no partial child.

Example (3)

\[ U = \{A, B, C, D, E, F, G, H, I, J\} \]
\[ S_2 = \{C, D, F, G, I, J\} \]

Example (4)

\[ U = \{A, B, C, D, E, F, G, H, I, J\} \]
\[ S_3 = \{C, D, F, G, I, J\} \]

Example (5)

\[ U = \{A, B, C, D, E, F, G, H, I, J\} \]
\[ S_4 = \{C, D, F, G, I, J\} \]

Example (6)

\[ U = \{A, B, C, D, E, F, G, H, I, J\} \]
\[ S_5 = \{C, D, F, G, I, J\} \]

Example (7)

\[ U = \{A, B, C, D, E, F, G, H, I, J\} \]
\[ S_6 = \{A, B, E, G\} \]
Example (8)

U = \{A,B,C,D,E,F,G,H,I,J\}
S_3 = \{A,B,E,G\}

No P node needed here

Example (9)

U = \{A,B,C,D,E,F,G,H,I,J\}
S_2 = \{A,B,E,G\}

direct Q node

Example (10)

U = \{A,B,C,D,E,F,G,H,I,J\}
S_3 = \{A,B,E,G\}

direct P node

Example (11)

U = \{A,B,C,D,E,F,G,H,I,J\}
S_3 = \{A,B,E,G\}

attach

Example (12)

U = \{A,B,C,D,E,F,G,H,I,J\}
S_1 = \{A,C,E,G,I\}
S_2 = \{C,D,F,G,I,J\}
S_3 = \{A,B,E,G\}

Linear Number of Nodes Processed

- Let n be the size of the universe, m the number of sets, and s the sum of the sizes of the sets.
  - Number of full nodes processed \( \leq 2s \).
  - Number of key nodes processed = m.
  - Number of partial nodes with partial children processed below the key node \( \leq m + n \).
  - Number of partial nodes with no partial children \( \leq 2m \).
  - Number of partial nodes processed above the key node \( \leq m + n \).
Number of Processed Nodes Amortized

- Partially with partial children \( \leq m+n \)
- Full \( \leq 2s \)
- Root partially with no partial children \( \leq 2m \)

Partials with Partial Children Below the Key Node
- Amortized complexity argument.
- Consider the quantities:
  - \( q \) = number of Q nodes,
  - \( cp \) = number of children of P nodes.
- We examine the quantity \( x = q + cp \).
- \( x \) is initially \( n \) and never negative.
- Each restrict of a key node increases \( x \) by at most 1.
- Each direct of a partial node with a partial child decreases \( x \) by at least 1.
- Since there are \( m \) restricts of a key node then there are most \( n + m \) directs of partials with partial children.

Restricting a P Node with Partial Children

- Restrict a P node
- Change in \( q + cp \) is at most +1.

Restricting a P node with no Partial Children

- Restrict a P node
- Change in \( q + cp \) is exactly +1.

Restricting a Q node

- Restrict a Q node
- No change in \( q, cp \)

Directing a P Node

- Assume partial child
- Left direct a P node
- Change in \( q + cp \) is -1
Directing a Q Node

change in q + cp is -1

PQ Tree Notes

- In algorithmic design only a linear number of nodes are ever processed.
- Designing the data structures to make the linear time processing a reality is very tricky.
- PQ trees solve the idealized DNA ordering problem.
- In reality, because of errors, the DNA ordering problem is NP-hard and other techniques are use.

Example of Data Structure Trick

- Linking the children of a Q node

Applied Algorithms in a Nutshell (1)

- There are genuinely hard problems that require approximate solutions.
  - NP-completeness
  - Branch and Bound for small input size
  - Local search techniques
  - Specialized techniques like GLA.
- Some apparently hard problems are not really so.
  - minimum spanning tree
  - contiguous ordering

Applied Algorithms in a Nutshell (2)

- Cache performance matters.
  - Understanding and controlling cache performance is possible.
- Data compression involves interesting algorithms and theory.
  - Entropy
  - Huffman and arithmetic coding
  - Dictionary coding
  - Sequitur
  - VQ - nearest neighbor search
  - Wavelet compression and SPIHT

Applied Algorithms in a Nutshell (3)

- Computational biology also has interesting algorithms.
  - Approximate matching using dynamic programming.
  - Contiguous ordering using PQ trees.
- Fundamental algorithms should always be available.
  - depth-first search
  - breadth-first search
  - disjoint union/find
  - priority queues (d-heaps)
  - sorting
Applied Algorithms in a Nutshell (4)

- Algorithm evaluation and analysis are critical for understanding correctness and performance.
  - Correctness
    - high level thinking about design.
    - development of good invariants.
  - Analysis of algorithms
    - time and storage analysis.
    - amortized analysis.
    - cache performance analysis.