• HW4 questions?

• BLAST algorithm

• If time: the Stack vs. the Heap
  • Using new and delete in C++
HW4 Questions?
Basic Local Alignment Search Tool (BLAST)

<table>
<thead>
<tr>
<th>BLAST Assembled Genomes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find Genomic BLAST pages:</td>
</tr>
<tr>
<td>Enter organism name or id–completions will be suggested</td>
</tr>
<tr>
<td>GO</td>
</tr>
<tr>
<td>Human</td>
</tr>
<tr>
<td>Mouse</td>
</tr>
<tr>
<td>Rat</td>
</tr>
<tr>
<td>Cow</td>
</tr>
<tr>
<td>Pig</td>
</tr>
<tr>
<td>Dog</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

**Basic BLAST**

Choose a BLAST program to run.

<table>
<thead>
<tr>
<th>Program</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nucleotide blast</td>
<td>Search a nucleotide database using a nucleotide query</td>
</tr>
<tr>
<td>protein blast</td>
<td>Search protein database using a protein query</td>
</tr>
<tr>
<td>blastx</td>
<td>Search protein database using a translated nucleotide query</td>
</tr>
<tr>
<td>tblastn</td>
<td>Search translated nucleotide database using a protein query</td>
</tr>
<tr>
<td>tblastx</td>
<td>Search translated nucleotide database using a translated nucleotide query</td>
</tr>
</tbody>
</table>
Basic Local Alignment Search Tool (BLAST)

Original paper has been cited 63,813 times
The general problem to solve
The general problem to solve

• Given a reference string of length $n$ and a query string of length $p$
The general problem to solve

• Given a reference string of length $n$ and a query string of length $p$

Reference: ABBBAABABBBABAABABABABAABBBAAABBBBBAAAAABBBABBA
The general problem to solve

- Given a reference string of length $n$ and a query string of length $p$

Reference: ABBBAABABBABABAABABABAABBBAAAAABBABBA

Query: ABBA
The general problem to solve

• Given a reference string of length $n$ and a query string of length $p$

• Find matches to the query string in the reference string with up to $e$ differences

Reference: ABBBAABABBBABABAABABABABAABBABABABAABBBAAAAAABBBABBA

Query: ABBA
The general problem to solve

• Given a reference string of length $n$ and a query string of length $p$

• Find matches to the query string in the reference string with up to $e$ differences

• Differences are the number of insertions, deletions, and substitutions

Reference: ABBBAABABBBBABAABABABAAABBBAAAAABBBABBA

Query: ABBA
The general problem to solve

- Given a reference string of length $n$ and a query string of length $p$

- Find matches to the query string in the reference string with up to $e$ differences

- Differences are the number of insertions, deletions, and substitutions

Reference: ABBBAABABBABABAABABABAABBBAAAAABBABBA

Query: ABBA

e = 1
The general problem to solve

• Given a reference string of length $n$ and a query string of length $p$

• Find matches to the query string in the reference string with up to $e$ differences

• Differences are the number of insertions, deletions, and substitutions

Reference: ABBBABAABBAABABAABABABABABAABBBAAAAABBBABBBABBA

Query: ABBA

e = 1
Categories of approximate match algorithms
Categories of approximate match algorithms

• Deterministic
  – Find the exact set of locations in the reference where the query matches given some threshold
Categories of approximate match algorithms

- **Deterministic**
  - Find the exact set of locations in the reference where the query matches given some threshold

- **Filter**
  - Returns false positives but no false negatives
Categories of approximate match algorithms

• Deterministic
  – Find the exact set of locations in the reference where the query matches given some threshold

• Filter
  – Returns false positives but no false negatives

• Heuristic
  – Some false negatives (misses some matches)
  – Some false positives (some incorrect matches)
Neighborhoods are sets of approximate matches

Reference: ABBBAABABABAAABABABABAABBBBBAAAAABBBABBA

Query: ABBA

e = 1
Neighborhoods are sets of approximate matches

Reference: ABBBAABABABBAABABABBAABBBBAAAAABBBABBA

Query: ABBA

$e = 1$

1 difference neighborhood of ABBA:

$N(ABBA) = \{ABBA, BBBA, AABA, ABAA, ABBB, AABBA, BABBA, ABBBA, ABABA, ABBA, ABBAA, ABBAB, BBA, ABA, ABB\}$
Neighborhoods are sets of approximate matches

Reference: ABBBAABABBBABABAABABABABAAABBBAAAAABBBABBA

Query: ABBA

e = 1

Hey look, a match!

1 difference neighborhood of ABBA:

N(ABBA) = {ABBA, BBBA, AABA, ABAA, ABBB, AABBA, BABBA, ABBBA, ABABA, ABBAA, ABBAB, BBA, ABA, ABA, ABB}
Neighborhoods are sets of approximate matches

Reference: ABBBAABABBBABABABABABABABABABABAAAABBA

Query: ABBA

\[ e = 1 \]

1 difference neighborhood of ABBA:
\[ N(ABBA) = \{ABBA, BBBA, AABA, ABAA, ABBB, AABBA, BABBA, ABBBA, ABABA, ABAAA, ABBAB, BBA, ABA, ABB\} \]
Neighborhoods are sets of approximate matches

Reference: ABBBAABABBABABAABABABABAAABBBAAAAABBBABBA

Query: ABBA

e = 1

1 difference neighborhood of ABBA:
N(ABBA) = {ABBA, BBBA, AABA, ABAA, ABBB, AABBA, BABBA, ABBBA, ABABA, ABBAA, ABBAB, BBA, ABA, ABB}

1 difference condensed neighborhood of ABBA:
NC(ABBA) = {BBBA, AABA, AABBA, BABBA, BBA, ABA, ABB}
Neighborhoods are sets of approximate matches

Reference: ABBBAABABBABABAABABABABAABBA

Query: ABBA

e = 1

1 difference neighborhood of ABBA:

\[ N(ABBA) = \{ABBA, BBBA, AABA, ABAA, ABBB, AABBA, BABBA, ABBBA, ABABA, ABBAA, ABBAB, BBA, ABA, ABB\} \]

1 difference condensed neighborhood of ABBA:

\[ NC(ABBA) = \{BBBA, AABA, AABBA, BABBA, BBA, ABA, ABB\} \]

Only members without prefixes in the neighborhood (why?)
Basic idea for BLAST
Basic idea for BLAST

1. Partition query sequence in $p/k$ $k$-mers
Basic idea for BLAST

1. Partition query sequence in $p/k$ $k$-mers

2. Generate the e-neighborhood of the query $k$-mers and find exact matches in the reference
Basic idea for BLAST

1. Partition query sequence in \( p/k \) \( k \)-mers

2. Generate the e-neighborhood of the query \( k \)-mers and find exact matches in the reference

3. Check each candidate match by extending from the ends of the \( k \)-mer
Basic idea for BLAST

1. Partition query sequence in \( plk k \)-mers

2. Generate the \( e \)-neighborhood of the query \( k \)-mers and find exact matches in the reference

3. Check each candidate match by extending from the ends of the \( k \)-mer

What \( k \)-mer size?
Basic idea for BLAST

1. Partition query sequence in \( plk \) k-mers

2. Generate the e-neighborhood of the query k-mers and find exact matches in the reference

3. Check each candidate match by extending from the ends of the k-mer

What k-mer size? Something around \( \log n \)
BLAST in detail: Step 1

• Defining $k$-mers
BLAST in detail: Step 1

• Defining $k$-mers
  – Should we use adjacent $k$-mers?
BLAST in detail: Step 1

- Defining $k$-mers
  - Should we use adjacent $k$-mers?

Query: PPRHKKMFYAVG

$k = 6$
BLAST in detail: Step 1

• Defining $k$-mers
  – Should we use adjacent $k$-mers?

Query: PPRHKKMFYAVG

$k = 6$

Adjacent $k$-mers: {PPRHKK, MFYAVG}
BLAST in detail: Step 1

- Defining $k$-mers
  - Should we use adjacent $k$-mers?
    - There could be some matches to regions spanning $k$-mer boundaries

Query: PPRHKKKMFYAVG

$k = 6$

Adjacent $k$-mers: {PPRHKK, MFYAVG}
BLAST in detail: Step 1

• Defining $k$-mers
  – Should we use adjacent $k$-mers?
    • There could be some matches to regions spanning $k$-mer boundaries
  – Try $p-k+1$ overlapping $k$-mers instead

Query: PPRHKKMFYAVG

$k = 6$

Adjacent $k$-mers: \{PPRHKK, MFYAVG\}
BLAST in detail: Step 1

- Defining $k$-mers
  - Should we use adjacent $k$-mers?
    - There could be some matches to regions spanning $k$-mer boundaries
  - Try $p-k+1$ overlapping $k$-mers instead

Query: PPRHKKMFYAVG

$k = 6$

Adjacent $k$-mers: {PPRHKK, MFYAVG}

Overlapping $k$-mers: {PPRHKK, PRHKKM, RHKKMF, HKKMFY, KKMFYA, KMFYAV, MFYAVG}
BLAST in detail: Step 2

• Generating the e-neighborhood of each $k$-mer
BLAST in detail: Step 2

• Generating the e-neighborhood of each $k$-mer
  – How should we define differences?
BLAST in detail: Step 2

• Generating the e-neighborhood of each $k$-mer
  – How should we define differences?
    • BLOSUM62 matrix for protein similarity
BLAST in detail: Step 2

- Generating the e-neighborhood of each $k$-mer
  - How should we define differences?
    - BLOSUM62 matrix for protein similarity
  - Given $k$-mer “PQG”, some possible neighbors
BLAST in detail: Step 2

• Generating the e-neighborhood of each $k$-mer
  – How should we define differences?
    • BLOSUM62 matrix for protein similarity
  – Given $k$-mer “PQG”, some possible neighbors
    • PEG: score is 15
    • PQA: score is 12
BLAST in detail: Step 2

• Generating the e-neighborhood of each $k$-mer
  – How should we define differences?
    • BLOSUM62 matrix for protein similarity
  – Given $k$-mer “PQG”, some possible neighbors
    • PEG: score is 15
    • PQA: score is 12
  – Use a cutoff score to define neighborhood
BLAST in detail: Step 2

• Generating the e-neighborhood of each $k$-mer
  – How should we define differences?
    • BLOSUM62 matrix for protein similarity
  – Given $k$-mer “PQG”, some possible neighbors
    • PEG: score is 15
    • PQA: score is 12
  – Use a cutoff score to define neighborhood

• Use an efficient method for identifying exact matches
BLAST in detail: Step 2

• Generating the e-neighborhood of each $k$-mer
  – How should we define differences?
    • BLOSUM62 matrix for protein similarity
  – Given $k$-mer “PQG”, some possible neighbors
    • PEG: score is 15
    • PQA: score is 12
  – Use a cutoff score to define neighborhood

• Use an efficient method for identifying exact matches
  – Finite automaton, like a Mealy machine
Brief Mealy machine example
Brief Mealy machine example

Input Sequence:
0010110
Brief Mealy machine example

Input Sequence:
0010110
Input Sequence: 0010110

Brief Mealy machine example

Start

S0

S1
Brief Mealy machine example

Input Sequence: 0010110
Brief Mealy machine example

Input Sequence: 0010110
Brief Mealy machine example

Input Sequence: 0010110
Brief Mealy machine example

Input Sequence: 0010110
Brief Mealy machine example

Input Sequence: 0010110

When does it output a 1?
Input Sequence: 0010110

When does it output a 1?
When the previous value is different
BLAST in detail: Step 3

- Extend each 'seed' match into a local alignment
BLAST in detail: Step 3

• Extend each 'seed' match into a local alignment
  – How do we extend?
“Double and Check” extension
“Double and Check” extension

Query: PPRHKKMFYAVG

\(k = 3, \text{ } k\text{-mer 'HKK'}\)
“Double and Check” extension

Query: PPRHKKMFYAVG

$k = 3$, $k$-mer 'HKK'

Candidate match: GAMPRHKKQFFM
“Double and Check” extension

Query: PPRHKKMFYAVG

$k = 3$, $k$-mer 'HKK'

Candidate match: GAMPRHKKQFFM

• Look at $2k$-mers that span 'HKK' in the query and see if they have an e-match spanning the candidate match
“Double and Check” extension

Query: PPRHKKMFYAVG

$k = 3$, $k$-mer 'HKK'

Candidate match: GAMPRIHKKQFFM

- Look at $2k$-mers that span 'HKK' in the query and see if they have an $e$-match spanning the candidate match
  - If yes, continue doubling
  - If no, then stop (provably correct)
“Double and Check” extension

Query: PPRHKKMFYAVG

\( k = 3, \) \( k \)-mer 'HKK'

Candidate match: GAMPRHKKQFFM

- Look at 2\( k \)-mers that span 'HKK' in the query and see if they have an e-match spanning the candidate match
  - If yes, continue doubling
  - If no, then stop (provably correct)

Example:

2\( k \)-mer: 'PRHKKM'

New candidate match: GAMPRHKKQFFM
BLAST in detail: Step 3

• Extend each 'seed' match into a local alignment
  – How do we extend? “double and check”
BLAST in detail: Step 3

• Extend each 'seed' match into a local alignment
  – How do we extend? “double and check”
  – Stop when score drops too far below the best score seen
BLAST in detail: Step 3

• Extend each 'seed' match into a local alignment
  – How do we extend? “double and check”
  – Stop when score drops too far below the best score seen
    • Hmm, this sounds similar to something we've done
BLAST in detail: Step 3

- Extend each 'seed' match into a local alignment
  - How do we extend? “double and check”
  - Stop when score drops too far below the best score seen
    - Hmm, this sounds similar to something we've done
- If the score is high enough, report the full match
BLAST in detail: Step 3

- Extend each 'seed' match into a local alignment
  - How do we extend? “double and check”
  - Stop when score drops too far below the best score seen
    - Hmm, this sounds similar to something we've done
- If the score is high enough, report the full match
  - What is high enough?
BLAST in detail: Step 3

- Extend each 'seed' match into a local alignment
  - How do we extend? “double and check”
  - Stop when score drops too far below the best score seen
    - Hmm, this sounds similar to something we've done
- If the score is high enough, report the full match
  - What is high enough?
  - Need some measure of significance
BLAST in detail: Step 3

- Extend each 'seed' match into a local alignment
  - How do we extend? “double and check”
  - Stop when score drops too far below the best score seen
    - Hmm, this sounds similar to something we've done

- If the score is high enough, report the full match
  - What is high enough?
  - Need some measure of significance
    - E-values based on Gumbel extreme value distribution
BLAST time complexity?
BLAST time complexity?

- Costly operations
BLAST time complexity?

- Costly operations
  - Generating the condensed \( k \)-mer neighborhoods
BLAST time complexity?

• Costly operations
  – Generating the condensed $k$-mer neighborhoods
  – Identifying candidate matches
BLAST time complexity?

- Costly operations
  - Generating the condensed $k$-mer neighborhoods
  - Identifying candidate matches
  - Extending candidate matches
BLAST time complexity?

- Costly operations
  - Generating the condensed $k$-mer neighborhoods
  - Identifying candidate matches
  - Extending candidate matches

- Generating neighborhoods and extending matches can be done in sublinear time (in terms of the size of the reference)
BLAST time complexity?

• Costly operations
  – Generating the condensed $k$-mer neighborhoods
  – Identifying candidate matches
  – Extending candidate matches

• Generating neighborhoods and extending matches can be done in sublinear time (in terms of the size of the reference)
  – Use a precomputed hash index to look up $k$-mer matches
What about gapped alignment?

• Similar for steps 1 and 2 (list \(k\)-mers neighborhoods, find candidate matches
What about gapped alignment?

- Similar for steps 1 and 2 (list $k$-mers neighborhoods, find candidate matches

- Instead of extending
  - Look for candidate match pairs closer together than some distance threshold
What about gapped alignment?

- Similar for steps 1 and 2 (list $k$-mers neighborhoods, find candidate matches.

- Instead of extending:
  - Look for candidate match pairs closer together than some distance threshold
  - Then extend from ends of paired matches
What about gapped alignment?

• Similar for steps 1 and 2 (list $k$-mers neighborhoods, find candidate matches

• Instead of extending
  – Look for candidate match pairs closer together than some distance threshold
  – Then extend from ends of paired matches

• Can also consider continuing to combine paired matches
Alternatively, you could have Gene Myers explain BLAST

- https://www.youtube.com/watch?v=pVFX3V0Q2Rg
Stack vs. Heap

Using *new* and *delete* in C++
Stack vs. Heap
Using \textit{new} and \textit{delete} in C++

- The stack is a preallocated piece of memory given to your program when it is run
Stack vs. Heap
Using *new* and *delete* in C++

- The stack is a preallocated piece of memory given to your program when it is run.
Stack vs. Heap
Using *new* and *delete* in C++

- The stack is a preallocated piece of memory given to your program when it is run.

- Variables get put on the stack as your program runs.
Stack vs. Heap
Using *new* and *delete* in C++

- The stack is a preallocated piece of memory given to your program when it is run.

- Variables get put on the stack as your program runs.

```cpp
my_string
```
Stack vs. Heap
Using *new* and *delete* in C++

- The stack is a preallocated piece of memory given to your program when it is run.

- Variables get put on the stack as your program runs.

```plaintext
The Stack

my_int
my_string
```
Stack vs. Heap
Using *new* and *delete* in C++

- The stack is a preallocated piece of memory given to your program when it is run.

- Variables get put on the stack as your program runs.
Stack vs. Heap
Using *new* and *delete* in C++

- The stack is a preallocated piece of memory given to your program when it is run.

- Variables get put on the stack as your program runs.

- Different functions may define part of the stack for local variables.

```plaintext
The Stack
- my_array
- my_int
- my_string
```
Stack vs. Heap

Using *new* and *delete* in C++

- The stack is a preallocated piece of memory given to your program when it is run.

- Variables get put on the stack as your program runs.

- Different functions may define part of the stack for local variables.
Stack vs. Heap

Using *new* and *delete* in C++

- The stack is a preallocated piece of memory given to your program when it is run.
- Variables get put on the stack as your program runs.
- Different functions may define part of the stack for local variables.
  - This is how scoping gets determined.
Stack vs. Heap
Using *new* and *delete* in C++
Stack vs. Heap
Using *new* and *delete* in C++

- The heap is all non-allocated space in memory
Stack vs. Heap
Using *new* and *delete* in C++

- The heap is all non-allocated space in memory
Stack vs. Heap

Using *new* and *delete* in C++

- The heap is all non-allocated space in memory
- Using *new* allocates space on the heap for a variable
Stack vs. Heap

Using *new* and *delete* in C++

- The heap is all non-allocated space in memory
- Using *new* allocates space on the heap for a variable

---

**The Stack**
- `new int`
- `function_1`
- `my_array`
- `my_int`
- `my_string`

**The Heap**
Stack vs. Heap

Using *new* and *delete* in C++

- The heap is all non-allocated space in memory
- Using *new* allocates space on the heap for a variable

The Stack

- *new int*
- function_1 variables
- my_array
- my_int
- my_string

The Heap

- int
Stack vs. Heap

Using *new* and *delete* in C++

- The heap is all non-allocated space in memory
- Using *new* allocates space on the heap for a variable
- *delete* frees the space allocated by *new*
Stack vs. Heap
Using *new* and *delete* in C++

- Technically, you don't have to use *delete*
Stack vs. Heap
Using *new* and *delete* in C++

- Technically, you don't have to use *delete*
  - But what problems could this cause?
Stack vs. Heap
Using *new* and *delete* in C++

- Technically, you don't have to use *delete*
  - But what problems could this cause?
    - Memory can't be used again while your program is running
Stack vs. Heap
Using *new* and *delete* in C++

- Technically, you don't have to use *delete*
  - But what problems could this cause?
    - Memory can't be used again while your program is running
      - It can't even be used by other programs
Stack vs. Heap
Using *new* and *delete* in C++

- Technically, you don't have to use *delete*
  - But what problems could this cause?
    - Memory can't be used again while your program is running
      - It can't even be used by other programs
  - It's a good idea to *delete* things when you're done using them so you're not wasting memory