Discussion Section 4

- HW2 comments/HW3 questions
- Edit graph optimization
- Useful data structures
HW2/3 Questions?
HW2/3 Questions?

• Comment on HW2:
  – Hard coding an initial negative weight for non-start nodes is problematic
  • Any suggestions for what else you could do?
HW2/3 Questions?

• Comment on HW2:
  – Hard coding an initial negative weight for non-start nodes is problematic
    • Any suggestions for what else you could do?
      – Iteratively remove nodes without parents except for the start node
HW2/3 Questions?

• Comment on HW2:
  – Hard coding an initial negative weight for non-start nodes is problematic
    • Any suggestions for what else you could do?
      – Iteratively remove nodes without parents except for the start node
      – Give each node a flag indicating if the path to it includes the start node
HW 4: Edit graph

- Create an edit graph for 3 sequences using the BLOSUM62 score matrix
- Output in the same format as HW2
- Run your highest-weight path program on the edit graph to find the highest scoring path (local alignment)
HW 4: Edit graph

Protein 1: M R Y I I V Y ...
Protein 2: M L V V L A N ...
Protein 3: M Y V I L V Y ...
HW 4: Edit graph

Protein 1:  M  R  Y  I  I  V  Y  ...
Protein 2:  M  L  V  V  L  A  N  ...
Protein 3:  M  Y  V  I  L  V  Y  ...
HW 4: Edit graph

Protein 1: M R Y I I V Y ...
Protein 2: M L V V L A N ...
Protein 3: M Y V I L V Y ...

Possible edges:
HW 4: Edit graph

Protein 1: M R Y I I V Y ...
Protein 2: M L V V L A N ...
Protein 3: M Y V I L V Y ...

Possible edges:

MMM  MM-  M-M  M--  -MM  -M-  --M
HW 4: Edit graph

Protein 1: M R Y I I V Y ...

Protein 2: M L V V L A N ...

Protein 3: M Y V I L V Y ...

Possible edges:

MMM MM- M-M M-- -MM -M- --M
HW 4: Edit graph

Protein 1: M R Y I I V Y ...

Protein 2: M L V V L A N ...

Protein 3: M Y V I L V Y ...

Possible edges:

MMM MM- M-M M-- -MM -M- --M

RVI RV- R-I R-- -VI -V- --I
The *Edit Graph* for a Pair of Sequences
The *Edit Graph* for a Pair of Sequences
Method of Four Russians
Method of Four Russians

- Suppose scores are either 1 for a match (diagonal) or 0 for a skip (horizontal or vertical)
Method of Four Russians

Set top corner to be 0

- Suppose scores are either 1 for a match (diagonal) or 0 for a skip (horizontal or vertical)
Method of Four Russians

Set top corner to be 0

Encode adjacent vertices as the relative difference

• Suppose scores are either 1 for a match (diagonal) or 0 for a skip (horizontal or vertical)
Method of Four Russians

Maximum difference between these two values?
Method of Four Russians

Maximum difference between these two values?

- By definition, $Y \geq X - \min(\text{horizontal transition score})$
Method of Four Russians

Maximum difference between these two values?

- By definition, $Y \geq X - \min(\text{horizontal transition score})$
- What about $Y > X + \max(\text{difference in transition scores})$?
Method of Four Russians

- By definition, $Y \geq X - \min(\text{horizontal transition score})$
- What about $Y > X + \max(\text{difference in transition scores})$?
- If the best path to $Y$ came from the vertical or diagonal edge, then that came from some vertex $Z$ in the same column as $X$
Method of Four Russians

The maximum difference between these two values?

- By definition, $Y \geq X - \min(\text{horizontal transition score})$
- What about $Y > X + \max(\text{difference in transition scores})$?
  - If the best path to $Y$ came from the vertical or diagonal edge, then that came from some vertex $Z$ in the same column as $X$
  - The vertical path from $Z$ to $X$ differs from the path from $Z$ to $Y$ by either
Method of Four Russians

- By definition, \( Y \geq X - \min(\text{horizontal transition score}) \)
- What about \( Y > X + \max(\text{difference in transition scores}) \)?
  - If the best path to \( Y \) came from the vertical or diagonal edge, then that came from some vertex \( Z \) in the same column as \( X \)
  - The vertical path from \( Z \) to \( X \) differs from the path from \( Z \) to \( Y \) by either
    - a single horizontal
Method of Four Russians

Maximum difference between these two values?

- By definition, $Y \geq X - \min(\text{horizontal transition score})$
- What about $Y > X + \max(\text{difference in transition scores})$?
  - If the best path to $Y$ came from the vertical or diagonal edge, then that came from some vertex $Z$ in the same column as $X$
  - The vertical path from $Z$ to $X$ differs from the path from $Z$ to $Y$ by either
    - a single horizontal
    - a diagonal edge that replaced a vertical edge
Method of Four Russians

Set top corner to be 0

- Suppose scores are either 1 for a match (diagonal) or 0 for a skip (horizontal or vertical)

Encode adjacent vertices as the relative difference
Method of Four Russians

Set top corner to be 0

0 or +1

- Suppose scores are either 1 for a match (diagonal) or 0 for a skip (horizontal or vertical)

Encode adjacent vertices as the relative difference
Method of Four Russians

Set top corner to be 0

0 or +1

0, +1, or +2

• Suppose scores are either 1 for a match (diagonal) or 0 for a skip (horizontal or vertical)

Encode adjacent vertices as the relative difference
Method of Four Russians

Set top corner to be 0

0 or +1

0, +1, or +2

0 or +1

• Suppose scores are either 1 for a match (diagonal) or 0 for a skip (horizontal or vertical)

Encode adjacent vertices as the relative difference
Method of Four Russians

Set top corner to be 0

0 or +1

0, +1, or +2

Suppose scores are either 1 for a match (diagonal) or 0 for a skip (horizontal or vertical)

Encode adjacent vertices as the relative difference

0 or +1

0, +1, or +2
Method of Four Russians

Set top corner to be 0

- Suppose scores are either 1 for a match (diagonal) or 0 for a skip (horizontal or vertical)
- This can be relaxed as long as all transition scores are integers and bounded

Encode adjacent vertices as the relative difference

0 or +1
0, +1, or +2
0, +1, or +2

0 or +1
The *Edit Graph* for a Pair of Sequences
The *Edit Graph* for a Pair of Sequences
The *Edit Graph* for a Pair of Sequences
The *Edit Graph* for a Pair of Sequences
The *Edit Graph* for a Pair of Sequences
The *Edit Graph* for a Pair of Sequences

G
C
A
G
A
G
A
C
G
A
T
G
A
T
G
A
C
C
C
A
The *Edit Graph* for a Pair of Sequences

```
 A   C   G   T   T   G   A   A   T   G   A   C   C   C   A
G   C   A   T   G   A   C   G   A
C   A   T   G   A   C   G   A
T   G   A   C   G   A
G   A   C   G   A
A
```
The Edit Graph for a Pair of Sequences
How fast is this?
How fast is this?

- Given two sequences of length $N$ and $M$
How fast is this?

- Given two sequences of length $N$ and $M$
- Use $t$-by-$t$ blocks
How fast is this?

- Given two sequences of length $N$ and $M$
- Use $t$-by-$t$ blocks
  - Reduces the dimensions to $O(N/t * M/t)$
How fast is this?

- Given two sequences of length N and M
- Use t-by-t blocks
  - Reduces the dimensions to $O(\frac{N}{t} \times \frac{M}{t})$
  - For every block you compute $O(t)$ new scores
How fast is this?

• Given two sequences of length N and M
• Use t-by-t blocks
  – Reduces the dimensions to $O(N/t \times M/t)$
  – For every block you compute $O(t)$ new scores
  – So new time is $O(NM/t)$ + time to precompute blocks
How fast is this?

• Given two sequences of length N and M
• Use t-by-t blocks
  – Reduces the dimensions to $O(N/t \times M/t)$
  – For every block you compute $O(t)$ new scores
  – So new time is $O(NM/t) + \text{time to precompute blocks}$
    • Usually you choose $t = O(\log N)$ or $O(\log M)$, which you can show will give a total time of $O(NM/\log N)$ or $O(NM/\log M)$
How fast is this?

- Given two sequences of length N and M
- Use t-by-t blocks
  - Reduces the dimensions to $O(N/t \times M/t)$
  - For every block you compute $O(t)$ new scores
  - So new time is $O(NM/t) + \text{time to precompute blocks}$
    - Usually you choose $t = O(\log N) \text{ or } O(\log M)$, which you can show will give a total time of $O(NM/\log N) \text{ or } O(NM/\log M)$
      - The exact value of $t$ depends on the total number of possible transition scores
Some useful data structure features
Some useful data structure features

- Arrays
  - Fast, pointer math is easy
Some useful data structure features

- **Arrays**
  - Fast, pointer math is easy

- **Linked lists**
  - Inserting/deleting/reordering is easy
Some useful data structure features

• Arrays
  – Fast, pointer math is easy

• Linked lists
  – Inserting/deleting/reordering is easy

• Hash tables/maps
  – Good for looking up things
Some useful data structure features

- **Arrays**
  - Fast, pointer math is easy
- **Linked lists**
  - Inserting/deleting/reordering is easy
- **Hash tables/maps**
  - Good for looking up things
- **Trees**
  - Useful for sorting/searching
Some useful data structure features

- **Arrays**
  - Fast, pointer math is easy
- **Linked lists**
  - Inserting/deleting/reordering is easy
- **Hash tables/maps**
  - Good for looking up things
- **Trees**
  - Useful for sorting/searching
- **Heaps**
  - Keeping track of extreme values
Some useful data structure applications
Some useful data structure applications

- Arrays
  - Storing a sequence, accessing the N-th position
  - Many other data structures use an underlying array
Some useful data structure applications

- **Arrays**
  - Storing a sequence, accessing the N-th position
  - Many other data structures use an underlying array

- **Linked lists**
  - Graph searching (breadth-first)
Some useful data structure applications

- **Arrays**
  - Storing a sequence, accessing the N-th position
  - Many other data structures use an underlying array

- **Linked lists**
  - Graph searching (breadth-first)

- **Hash tables/maps**
  - Storing sample features
  - Counting k-mer frequency
Some useful data structure applications

- **Arrays**
  - Storing a sequence, accessing the N-th position
  - Many other data structures use an underlying array

- **Linked lists**
  - Graph searching (breadth-first)

- **Hash tables/maps**
  - Storing sample features
  - Counting k-mer frequency

- **Trees**
  - $k$-dimensional trees for nearest neighbor searching (PyCogent)
Some useful data structure applications

- **Arrays**
  - Storing a sequence, accessing the N-th position
  - Many other data structures use an underlying array

- **Linked lists**
  - Graph searching (breadth-first)

- **Hash tables/maps**
  - Storing sample features
  - Counting k-mer frequency

- **Trees**
  - k-dimensional trees for nearest neighbor searching
    (PyCogent)

- **Heaps**
  - Constructing a minimum spanning tree (Monocle does this,
    not sure if it uses a heap though)
Arrays

- Getting the element at a particular index is fast
Arrays

- Getting the element at a particular index is fast
Arrays

- Getting the element at a particular index is fast
Arrays

- Getting the element at a particular index is fast
Arrays

- Getting the element at a particular index is fast
Arrays

- Getting the element at a particular index is fast
Arrays

- Getting the element at a particular index is fast
Arrays

- Getting the element at a particular index is fast
Arrays

- Getting the element at a particular index is fast
Linked Lists

- Easier to modify than an array
Linked Lists

- Easier to modify than an array
Linked Lists

- Easier to modify than an array
Linked Lists

- Easier to modify than an array
Linked Lists

- Easier to modify than an array
Fast for looking up values
Hash Tables and Hash Maps

• Fast for looking up values
Hash Tables and Hash Maps

- Fast for looking up values

Contiguous Memory

Cow
Hash Tables and Hash Maps

- Fast for looking up values

Contiguous Memory

Cow ➔ Super Secret Hash Function
Hash Tables and Hash Maps

- Fast for looking up values

Contiguous Memory

Cow

Super Secret Hash Function

4
Hash Tables and Hash Maps

- Fast for looking up values

Contiguous Memory

Cow

Super Secret Hash Function

Cow
Hash Tables and Hash Maps

- Fast for looking up values

Contiguous Memory

Cow
Chicken

Super Secret Hash Function

Chicken ➔ 7
Hash Tables and Hash Maps

• Fast for looking up values

Contiguous Memory

Cow

Cow?

Super Secret Hash Function

4

Chicken
Hash Tables and Hash Maps

- Fast for looking up values

Contiguous Memory

Goat? → Super Secret Hash Function → 2

Cow, Chicken
Brief aside: hash functions
Brief aside: hash functions

• Should generate uniformly distributed hash values
Brief aside: hash functions

• Should generate uniformly distributed hash values
  – Why?
Brief aside: hash functions

• Should generate uniformly distributed hash values
  – Why?

• Often used in cryptography to verify data
Brief aside: hash functions

• Should generate uniformly distributed hash values
  – Why?

• Often used in cryptography to verify data
  – Difficult to reverse engineer a hash value to a matching input
Brief aside: hash functions

- Should generate uniformly distributed hash values
  - Why?
- Often used in cryptography to verify data
  - Difficult to reverse engineer a hash value to a matching input

```c
size_t myhash(const string& s) {
    // Inspired by xkcd comic 153 and the Cha Cha slide
    size_t hash = 0;
    // This time we're gonna get funky
    // (Convert the string to a number and do some strange things to it)
    // (Inspired by Jenkins one-at-a-time hash function (found on Wikipedia))
    for (size_t i = 0; i < s.size(); ++i) {
        hash += s[i];
        hash += (hash << 8); // Two measure of 4 is 8 beats
        hash ^= (hash >> 5); // Funky has 5 letters
    }
    // Everybody clap your hands
    // Clap clap clap clap your hands
    // Clap clap clap clap your hands
    // (Flip the bits)
    hash = ~hash;
    // Alright we gonna do the basic steps
    // Slide to the left
    // (The Cha Cha slide is in 4)
    hash = hash << (hash % 4);
    // Slide to the right
    // (But it really should be in 3)
    hash = hash >> (hash % 3);
    // Take it back now y'all
    // (Subtract - No one wants to walk back that far, so 1)
    hash -= (hash >> 1);
    // One hop this time
    // (Bunnies hop and bunnies eat carrots. Some bunnies can hop 5 feet)
    hash ^= (hash % 5);
    // Right foot lets stomp
    // (Add, make it dependent on the hash function for more randomness)
}
```
Trees

- Good for searching ranges
Trees

- Good for searching ranges

Any values between 5 and 8 exclusive?

A binary search tree of size 9 and depth 3, with root 8 and leaves 1, 4, 7 and 13
Trees

- Good for searching ranges

A binary search tree of size 9 and depth 3, with root 8 and leaves 1, 4, 7 and 13.

Any values between 5 and 8 exclusive?
Trees

- Good for searching ranges

A binary search tree of size 9 and depth 3, with root 8 and leaves 1, 4, 7 and 13.

Any values between 5 and 8 exclusive?
Trees

- Good for searching ranges

Any values between 5 and 8 exclusive?

A binary search tree of size 9 and depth 3, with root 8 and leaves 1, 4, 7 and 13.
Heaps

- Good for keeping track of extreme values
Heaps

- Good for keeping track of extreme values

What's the largest value?
Trees and Heaps

• Similar in structure, but different rules
### Operation

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Access</th>
<th>Search</th>
<th>Insert</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array</td>
<td>O(1)</td>
<td>O(N)</td>
<td>O(1) or O(N)</td>
<td>O(1) or O(N)</td>
</tr>
<tr>
<td>Linked List</td>
<td>O(N)</td>
<td>O(N)</td>
<td>O(1) or O(N)</td>
<td>O(1) or O(N)</td>
</tr>
<tr>
<td>Hash Map</td>
<td>N/A</td>
<td>O(1)</td>
<td>O(1)*</td>
<td>O(1)*</td>
</tr>
<tr>
<td>Tree</td>
<td>O(log N)</td>
<td>O(log N)</td>
<td>O(log N)</td>
<td>O(log N)</td>
</tr>
<tr>
<td>Data Structure</td>
<td>Access</td>
<td>Search</td>
<td>Insert</td>
<td>Delete</td>
</tr>
<tr>
<td>----------------</td>
<td>---------</td>
<td>---------</td>
<td>-----------------</td>
<td>------------------</td>
</tr>
<tr>
<td>Array</td>
<td>O(1)</td>
<td>O(N)</td>
<td>O(1) or O(N)</td>
<td>O(1) or O(N)</td>
</tr>
<tr>
<td>Linked List</td>
<td>O(N)</td>
<td>O(N)</td>
<td>O(1) or O(N)</td>
<td>O(1) or O(N)</td>
</tr>
<tr>
<td>Hash Map</td>
<td>N/A</td>
<td>O(1)</td>
<td>O(1)*</td>
<td>O(1)*</td>
</tr>
<tr>
<td>Tree</td>
<td>O(log N)</td>
<td>O(log N)</td>
<td>O(log N)</td>
<td>O(log N)</td>
</tr>
</tbody>
</table>

* Actual worst case is O(N), but 'amortized' it's O(1)
<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Access</th>
<th>Search</th>
<th>Insert</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array</td>
<td>O(1)</td>
<td>O(N)</td>
<td>O(1) or O(N)</td>
<td>O(1) or O(N)</td>
</tr>
<tr>
<td>Linked List</td>
<td>O(N)</td>
<td>O(N)</td>
<td>O(1) or O(N)</td>
<td>O(1) or O(N)</td>
</tr>
<tr>
<td>Hash Map</td>
<td>N/A</td>
<td>O(1)</td>
<td>O(1)*</td>
<td>O(1)*</td>
</tr>
<tr>
<td>Tree</td>
<td>O(log N)</td>
<td>O(log N)</td>
<td>O(log N)</td>
<td>O(log N)</td>
</tr>
</tbody>
</table>

* Actual worst case is O(N), but 'amortized' it's O(1)

Heaps are specialized for find-min/max = O(1), delete-min/max = O(log N), and insert = O(log N)
Optimizing for space efficiency

Sequence 1

Sequence 2
Optimizing for space efficiency

Sequence 1

Sequence 2
Optimizing for space efficiency

Sequence 1

Midpoint in sequence 1
Optimizing for space efficiency

Sequence 1

Midpoint in sequence 1
Optimizing for space efficiency
Optimizing for space efficiency

Sequence 1

Sequence 2
Optimizing for space efficiency

Sequence 1

Sequence 2
How much time does this take?
How much time does this take?

- If sequence 1 has length N and sequence 2 has length M
  - First pass: NM time (update N nodes M times)
How much time does this take?

- If sequence 1 has length N and sequence 2 has length M
  - First pass: NM time (update N nodes M times)
  - Second pass: (NM)/2 time (half of the area)
How much time does this take?

- If sequence 1 has length N and sequence 2 has length M
  - First pass: NM time (update N nodes M times)
  - Second pass: (NM)/2 time (half of the area)
  - Third pass: (NM)/4 (quarter of the area)
How much time does this take?

- If sequence 1 has length $N$ and sequence 2 has length $M$
  - First pass: $NM$ time (update $N$ nodes $M$ times)
  - Second pass: $(NM)/2$ time (half of the area)
  - Third pass: $(NM)/4$ (quarter of the area)
  - And so on
How much time does this take?

- If sequence 1 has length N and sequence 2 has length M
  - First pass: $NM$ time (update N nodes M times)
  - Second pass: $(NM)/2$ time (half of the area)
  - Third pass: $(NM)/4$ (quarter of the area)
  - And so on
    - $1 + 1/2 + 1/4 + \ldots = 2$, so $2NM$ or $O(NM)$
How much time does this take?

- If sequence 1 has length $N$ and sequence 2 has length $M$
  - First pass: $NM$ time (update $N$ nodes $M$ times)
  - Second pass: $(NM)/2$ time (half of the area)
  - Third pass: $(NM)/4$ (quarter of the area)
  - And so on
    - $1 + 1/2 + 1/4 + \ldots = 2$, so $2NM$ or $O(NM)$
    - Awesome! That's the same asymptotic time as before!
How much time does this take?

- If sequence 1 has length $N$ and sequence 2 has length $M$
  - First pass: $NM$ time (update $N$ nodes $M$ times)
  - Second pass: $(NM)/2$ time (half of the area)
  - Third pass: $(NM)/4$ (quarter of the area)
  - And so on
    - $1 + 1/2 + 1/4 + \ldots = 2$, so $2NM$ or $O(NM)$
    - Awesome! That's the same asymptotic time as before!
    - But can we do better?