CS311H

Prof: Peter Stone

Department of Computer Science
The University of Texas at Austin

Good Morning, Colleagues

Good Morning, Colleagues

Are there any questions?

Logistics

- No discussion tomorrow

Logistics

- No discussion tomorrow
- Tricky module due next Tuesday

Logistics

- No discussion tomorrow
- Tricky module due next Tuesday
- Official course surveys

Who Comes Out Ahead?

Who Comes Out Ahead?

- If a girl can never propose to a boy, she has no better strategy than the one she uses in TMA.

Who Comes Out Ahead?

- If a girl can never propose to a boy, she has no better strategy than the one she uses in TMA. Why?

Who Comes Out Ahead?

- If a girl can never propose to a boy, she has no better strategy than the one she uses in TMA. Why?
- So if the boys use TMA, the boys and girls will be running TMA.

Who Comes Out Ahead?

- If a girl can never propose to a boy, she has no better strategy than the one she uses in TMA. Why?
- So if the boys use TMA, the boys and girls will be running TMA.
- Is it in the boys' interest to use TMA?

Who Comes Out Ahead?

- If a girl can never propose to a boy, she has no better strategy than the one she uses in TMA. Why?
- So if the boys use TMA, the boys and girls will be running TMA.
- Is it in the boys' interest to use TMA?
  - What if there are multiple stable pairings?

Who Comes Out Ahead?

- If a girl can never propose to a boy, she has no better strategy than the one she uses in TMA. Why?
- So if the boys use TMA, the boys and girls will be running TMA.
- Is it in the boys' interest to use TMA?
  - What if there are multiple stable pairings?
  - How should we define a person's optimal mate? Pessimal mate?

Who Comes Out Ahead?

- If a girl can never propose to a boy, she has no better strategy than the one she uses in TMA. Why?
- So if the boys use TMA, the boys and girls will be running TMA.
- Is it in the boys' interest to use TMA?
  - What if there are multiple stable pairings?
  - How should we define a person's optimal mate? Pessimal mate?
  - Theorem: TMA is optimal for the males and pessimal for the females

Male Optimality (ack: Steven Rudich)

- Suppose not.

Male Optimality (ack: Steven Rudich)

- Suppose not.
- There must be a **first time** in TMA that some boy b gets rejected by his optimal girl g because she said "maybe" to some better $\hat{b}$.

Male Optimality (ack: Steven Rudich)

- Suppose not.
- There must be a **first time** in TMA that some boy b gets rejected by his optimal girl g because she said "maybe" to some better $\hat{b}$.
- Since its the first time a boy gets rejected by his optimal, $\hat{b}$ has not yet been rejected by his optimal.

Male Optimality (ack: Steven Rudich)

- Suppose not.
- There must be a **first time** in TMA that some boy b gets rejected by his optimal girl g because she said "maybe" to some better $\hat{b}$.
- Since its the first time a boy gets rejected by his optimal, $\hat{b}$ has not yet been rejected by his optimal.
- So $\hat{b}$ likes g at least as much as his optimal.

Male Optimality (ack: Steven Rudich)

- Suppose not.
- There must be a **first time** in TMA that some boy b gets rejected by his optimal girl g because she said "maybe" to some better $\hat{b}$.
- Since its the first time a boy gets rejected by his optimal, $\hat{b}$ has not yet been rejected by his optimal.
- So $\hat{b}$ likes g at least as much as his optimal.
- Let Δ be a stable pairing in which b and g are paired (why does it exist?)

Male Optimality (ack: Steven Rudich)

- Suppose not.
- There must be a **first time** in TMA that some boy b gets rejected by his optimal girl g because she said "maybe" to some better $\hat{b}$.
- Since its the first time a boy gets rejected by his optimal, $\hat{b}$ has not yet been rejected by his optimal.
- So $\hat{b}$ likes g at least as much as his optimal.
- Let Δ be a stable pairing in which b and g are paired (why does it exist?)
- Δ pairs $\hat{b}$ with some $\hat{g}$

Male Optimality (ack: Steven Rudich)

- Suppose not.
- There must be a **first time** in TMA that some boy b gets rejected by his optimal girl g because she said "maybe" to some better $\hat{b}$.
- Since its the first time a boy gets rejected by his optimal, $\hat{b}$ has not yet been rejected by his optimal.
- So $\hat{b}$ likes g at least as much as his optimal.
- Let Δ be a stable pairing in which b and g are paired (why does it exist?)
- Δ pairs $\hat{b}$ with some $\hat{g}$
- $\hat{b}$ and g form a rogue couple in Δ

Female Pessimality

- The pairing output by TMA, T, is male-optimal

Female Pessimality

- The pairing output by TMA, T, is male-optimal

- Assume there is a stable pairing Δ where g does worse in Δ than in T.

Female Pessimality

- The pairing output by TMA, T, is male-optimal

- Assume there is a stable pairing Δ where g does worse in Δ than in T.

- Let b be her mate in T.

Female Pessimality

- The pairing output by TMA, T, is male-optimal

- Assume there is a stable pairing Δ where g does worse in Δ than in T.

- Let b be her mate in T.

- Let $\hat{b}$ be her mate in Δ.

Female Pessimality

- The pairing output by TMA, T, is male-optimal
- Assume there is a stable pairing Δ where g does worse in Δ than in T.
- Let b be her mate in T.
- Let $\hat{b}$ be her mate in Δ.
- g and b form a rogue couple in Δ.

Lessons

- Boys act in their own self-interest if they follow TMA

Lessons

- Boys act in their own self-interest if they follow TMA
- If girls don't propose to boys, they will follow TMA

Lessons

- Boys act in their own self-interest if they follow TMA
- If girls don't propose to boys, they will follow TMA
- Dating advice for girls. . .

Linear Majority

Recall from last week:

Linear Majority

Recall from last week:

- Suppose that the votes of n people for several (more than 2) candidates for a particular office are the elements of a sequence. To win, a candidate must receive a majority (more than half) of the votes. Devise a divide-and-conquer algorithm that determines whether a candidate received a majority and if so determine who this candidate is. (must use constant, i.e. O(1), memory) What is it's Big-O runtime?

Linear Majority

Recall from last week:

- Suppose that the votes of n people for several (more than 2) candidates for a particular office are the elements of a sequence. To win, a candidate must receive a majority (more than half) of the votes. Devise a divide-and-conquer algorithm that determines whether a candidate received a majority and if so determine who this candidate is. (must use constant, i.e. O(1), memory) What is it's Big-O runtime?
- It was O(n log n)

Linear Majority

Recall from last week:

- Suppose that the votes of n people for several (more than 2) candidates for a particular office are the elements of a sequence. To win, a candidate must receive a majority (more than half) of the votes. Devise a divide-and-conquer algorithm that determines whether a candidate received a majority and if so determine who this candidate is. (must use constant, i.e. O(1), memory) What is it's Big-O runtime?
- It was O(n log n)
- There is a simple algorithm that is linear: $O(n)$

Linear Majority

Recall from last week:

- Suppose that the votes of n people for several (more than 2) candidates for a particular office are the elements of a sequence. To win, a candidate must receive a majority (more than half) of the votes. Devise a divide-and-conquer algorithm that determines whether a candidate received a majority and if so determine who this candidate is. (must use constant, i.e. O(1), memory) What is it's Big-O runtime?
- It was O(n log n)
- There is a simple algorithm that is linear: $O(n)$
  - Correctness proof doesn't (technically) use induction

Linear Majority

Recall from last week:

- Suppose that the votes of n people for several (more than 2) candidates for a particular office are the elements of a sequence. To win, a candidate must receive a majority (more than half) of the votes. Devise a divide-and-conquer algorithm that determines whether a candidate received a majority and if so determine who this candidate is. (must use constant, i.e. O(1), memory) What is it's Big-O runtime?
- It was O(n log n)
- There is a simple algorithm that is linear: $O(n)$
  - Correctness proof doesn't (technically) use induction
  - First lets see the algorithm illustrated

Some notation

- concat(A, B): the concatenation of lists A and B

Some notation

- concat(A, B): the concatenation of lists A and B
- append(A, x): the list obtained by appending integer x to the list A

Some notation

- concat(A, B): the concatenation of lists A and B
- append(A, x): the list obtained by appending integer x to the list A
- bad(A): the predicate "List A is of even length and does not have a majority element"

Some notation

- concat(A, B): the concatenation of lists A and B
- append(A, x): the list obtained by appending integer x to the list A
- bad(A): the predicate "List A is of even length and does not have a majority element"
- count(A, x): the number of times integer x occurs in list A

Some notation

- concat(A, B): the concatenation of lists A and B
- append(A, x): the list obtained by appending integer x to the list A
- bad(A): the predicate "List A is of even length and does not have a majority element"
- count(A, x): the number of times integer x occurs in list A

Some simple facts:

Some notation

- concat(A, B): the concatenation of lists A and B
- append(A, x): the list obtained by appending integer x to the list A
- bad(A): the predicate "List A is of even length and does not have a majority element"
- count(A, x): the number of times integer x occurs in list A

Some simple facts:

1. If bad(A) and bad(B), then bad(concat(A, B)).

Some notation

- concat(A, B): the concatenation of lists A and B
- append(A, x): the list obtained by appending integer x to the list A
- bad(A): the predicate "List A is of even length and does not have a majority element"
- count(A, x): the number of times integer x occurs in list A

Some simple facts:

1. If bad(A) and bad(B), then bad(concat(A, B)).
2. If L has a majority element and L = concat(A, B) and bad(A), then B has a majority element and the majority element of B is equal to the majority element of L.

An Update Procedure

- update(x) will process one list element at a time

An Update Procedure

- update(x) will process one list element at a time
- L will be initially empty, and end up as the whole list

An Update Procedure

- update(x) will process one list element at a time
- L will be initially empty, and end up as the whole list
- z will be the majority element, if it exists (otherwise, it can be anything)

An Update Procedure

- update(x) will process one list element at a time
- L will be initially empty, and end up as the whole list
- z will be the majority element, if it exists (otherwise, it can be anything)
- k will be the algorithm's counter

An Update Procedure

- update(x) will process one list element at a time
- L will be initially empty, and end up as the whole list
- z will be the majority element, if it exists (otherwise, it can be anything)
- k will be the algorithm's counter
- A will be the front part of the list with no majority

An Update Procedure

- update(x) will process one list element at a time
- L will be initially empty, and end up as the whole list
- z will be the majority element, if it exists (otherwise, it can be anything)
- k will be the algorithm's counter
- A will be the front part of the list with no majority
- B will be the back part of the list with z as the majority

An Update Procedure

- update(x) will process one list element at a time
- L will be initially empty, and end up as the whole list
- z will be the majority element, if it exists (otherwise, it can be anything)
- k will be the algorithm's counter
- A will be the front part of the list with no majority
- B will be the back part of the list with z as the majority
- Invariant I: "L=concat(A,B) and bad(A) and

An Update Procedure

- update(x) will process one list element at a time
- L will be initially empty, and end up as the whole list
- z will be the majority element, if it exists (otherwise, it can be anything)
- k will be the algorithm's counter
- A will be the front part of the list with no majority
- B will be the back part of the list with z as the majority
- Invariant I: "L=concat(A,B) and bad(A) and $k = 2 \times \text{count}(B,z) - |B|$ and $k \geq 0$"

Initial Update Procedure

```
Initialize L=A=B={}, k=0, z=anything  // I
update(x)
  if (k = 0)
    A := concat(A, B)
    B := empty list
    z := x
// I and (k = 0 => z = x)
  L := append(L, x)
  B := append(B, x)
  if (z = x)
    k := k + 1
  else
    k := k - 1
  return z   // I
```

Lemmas

- Lemma 1: After initialization, I holds.

Lemmas

- Lemma 1: After initialization, I holds.

- Lemma 2: If I holds before first block, then "I and ($k = 0 \Rightarrow z = x$)" holds after.

Lemmas

- Lemma 1: After initialization, I holds.
- Lemma 2: If I holds before first block, then "I and ($k = 0 \Rightarrow z = x$)" holds after.
- Lemma 3: If "I and ($k = 0 \Rightarrow z = x$)" holds before 2nd block, then I holds after.

Lemmas

- Lemma 1: After initialization, I holds.
- Lemma 2: If I holds before first block, then "I and ($k = 0 \Rightarrow z = x$)" holds after.
- Lemma 3: If "I and ($k = 0 \Rightarrow z = x$)" holds before 2nd block, then I holds after.
- Lemma 4: If I holds and L has a majority element, then z is equal to the majority element of L.

Lemmas

- Lemma 1: After initialization, I holds.
- Lemma 2: If I holds before first block, then "I and ($k = 0 \Rightarrow z = x$)" holds after.
- Lemma 3: If "I and ($k = 0 \Rightarrow z = x$)" holds before 2nd block, then I holds after.
- Lemma 4: If I holds and L has a majority element, then z is equal to the majority element of L.
- These lemmas can be used to easily prove that the algorithm works correctly!

Lemmas

- Lemma 1: After initialization, I holds.

- Lemma 2: If I holds before first block, then "I and $(k = 0 \Rightarrow z = x)$" holds after.

- Lemma 3: If "I and $(k = 0 \Rightarrow z = x)$" holds before 2nd block, then I holds after.

- Lemma 4: If I holds and L has a majority element, then z is equal to the majority element of L.

- These lemmas can be used to easily prove that the algorithm works correctly! Why?

Lemmas

- Lemma 1: After initialization, I holds.
- Lemma 2: If I holds before first block, then "I and ($k = 0 \Rightarrow z = x$)" holds after.
- Lemma 3: If "I and ($k = 0 \Rightarrow z = x$)" holds before 2nd block, then I holds after.
- Lemma 4: If I holds and L has a majority element, then z is equal to the majority element of L.
- These lemmas can be used to easily prove that the algorithm works correctly! Why? Was this the same algorithm?

Final update procedure

- k and z do not depend on L, A, and B.

Final update procedure

- k and z do not depend on L, A, and B. Neither does the return value.

Final update procedure

- k and z do not depend on L, A, and B. Neither does the return value. So:

Final update procedure

- k and z do not depend on L, A, and B. Neither does the return value. So:

```
update(x)
  if (k = 0)
    z := x
  if (z = x)
    k := k + 1
  else
    k := k - 1
  return z
}
```

Challenge Problem

- Use divide and conquer to find the closest pair of points in a (planar) set in time $O(n \log n)$