Agenda

1. **Diversity, Equity, and Inclusion**: Today’s Guest
2. **Review**:
   a. Recursion and DFS
   b. Back-Tracking
   c. Dynamic Programming
3. **Discussion Problem**
Today’s Guest: Clarence “Skip” Ellis

- b. 1943: Chicago, IL
- Security guard at 15yo, taught himself to program using punch cards on a mainframe
- 1st African-American PhD in CS
- 1st African-American elected Fellow of ACM
- “Operational transformation” i.e. Google Docs!
Recursion and DFS: The Problem

**Question:** Is \((\text{fib}(n) > C)\)?
Recursion and DFS: Naive Recursive Method

**Question:** Is \( \text{fib}(n) > C \) ?

**Example:** Is \( \text{fib}(7) > C \) ?
**Recursion and DFS: Naive Recursive Method**

**Question:** Is \( \text{fib}(n) > C \)?

**Example:** Is \( \text{fib}(7) > C \)?
Recursion and DFS: Naive Recursive Method

**Question:** Is $(\text{fib}(n) > C)$?

**Example:** Is $(\text{fib}(7) > C)$?
Recursion and DFS: Naive Recursive Method

**Question:** Is \( \text{fib}(n) > C \)?

**Example:** Is \( \text{fib}(7) > C \)?
Recursion and DFS: Naive Recursive Method

**Question:** Is \( \text{fib}(n) > C \) ?

**Example:** Is \( \text{fib}(7) > C \) ?
Recursion and DFS: Naive Recursive Method

**Question:** Is $\text{fib}(n) > C$?

**Example:** Is $\text{fib}(7) > C$?
Recursion and DFS: Naive Recursive Method

**Question:** Is \( \text{fib}(n) > C \)?

**Example:** Is \( \text{fib}(7) > C \)?
Recursion and DFS: Naive Recursive Method

**Question:** Is \( \text{fib}(n) > C \)?

**Example:** Is \( \text{fib}(7) > C \)?
Recursion and DFS: Naive Recursive Method

**Question:** Is \((\text{fib}(n) > C)\)?

**Example:** Is \((\text{fib}(7) > C)\)?
Recursion and DFS: Naive Recursive Method

**Question:** Is \( (\text{fib}(n) > C) \)?

**Example:** Is \( (\text{fib}(7) > C) \)?
Recursion and DFS: Naive Recursive Method

**Question:** Is $\text{fib}(n) > C$?

**Example:** Is $\text{fib}(7) > C$?
Recursion and DFS: Naive Recursive Method

**Question:** Is \( \text{fib}(n) > C \)?

**Example:** Is \( \text{fib}(7) > C \)?
Recursion and DFS: Naive Recursive Method

**Question:** Is \( \text{fib}(n) > C \)?

**Example:** Is \( \text{fib}(7) > C \)?
Recursion and DFS: Naive Recursive Method

**Question:** Is \( \text{fib}(n) > C \)?

**Example:** Is \( \text{fib}(7) > C \)?
**Recursion and DFS: Naive Recursive Method**

**Question:** Is \( \text{fib}(n) > C \)?

**Example:** Is \( \text{fib}(7) > C \)?
Recursion and DFS: Naive Recursive Method

**Question:** Is \( \text{fib}(n) > C \)?

**Example:** Is \( \text{fib}(7) > C \)?
Recursion and DFS: Naive Recursive Method

**Question:** Is $(\text{fib}(n)) > C$?

**Example:** Is $(\text{fib}(7)) > C$?
Recursion and DFS: Relevance of DFS?

**Question:** Is $\text{fib}(n) > C$?

How did we solve this? What does this remind you of?
Recursion and DFS: Relevance of DFS?

**Question:** Is \( \text{fib}(n) > C \)?

How did we solve this? What does this remind you of?
Recursion and DFS: Relevance of DFS?

**Question:** Is $\text{fib}(n) > C$?

How did we solve this? What does this remind you of?
Recursion and DFS: Relevance of DFS?

**Question:** Is \( \text{fib}(n) > C \)?

How did we solve this? What does this remind you of?
Recursion and DFS: Relevance of DFS?

**Question**: Is \( (\text{fib}(n) > C) \)?

How did we solve this? What does this remind you of?
Recursion and DFS: Relevance of DFS?

**Question:** Is \( \text{fib}(n) > C \)?

How did we solve this? What does this remind you of?
Recursion and DFS: Relevance of DFS?

**Question:** Is \( \text{fib}(n) > C \)?

How did we solve this? What does this remind you of?
Recursion and DFS: Relevance of DFS?

**Question:** Is \( \text{fib}(n) > C \)?

How did we solve this? What does this remind you of?
Back-Tracking: The Problem

**Question:** Is \((\text{fib}(n) > C)\)?

Can we do better than naive recursion?
**Back-Tracking:** Pruning i.e. Assertions

**Question:** Is \( \text{fib}(n) > C \)?

Can we do better than naive recursion? **YES**
Back-Tracking: Pruning i.e. Assertions

**Question:** Is \( \text{fib}(n) > C \)?

Can we do better than naive recursion? **YES**
Back-Tracking: Pruning i.e. Assertions

Question: Is \( \text{fib}(n) > C \)?

Can we do better than naive recursion? **YES**
Dynamic Programming: The Problem

**Question:** Is $\text{fib}(n) > C$?

How many times do I need to answer the same question?
Dynamic Programming: Memoization

Question: Is \((\text{fib}(n) > C)\)?

How many times do I need to answer the same question? **ONCE**
Dynamic Programming: Memoization

**Question:** Is \( \text{fib}(n) > C \)?

How many times do I need to answer the same question? \textbf{ONCE}
Dynamic Programming: Memoization

**Question:** Is \( \text{fib}(n) > C \)?

How many times do I need to answer the same question? **ONCE**
Dynamic Programming: Memoization

Question: Is \( \text{fib}(n) > C \)?

How many times do I need to answer the same question? ONCE
Summary

Recursion vs. DFS vs. Back-Tracking vs. Dynamic Programming

1. **Recursion**: Algorithm where functions call themselves repeatedly
2. **DFS**: A search algorithm to find a solution to a problem that can be represented as a tree (i.e. a problem that is solved by using a recursion algorithm to build a tree -> then solve tree using DFS)
3. **Back-Tracking**: Recursion with DFS exploration + pruning for efficiency
4. **Dynamic Programming**: Recursion with DFS exploration + Re-use of solutions not necessarily with pruning
Discussion Problem
Discussion 7

Part 2
3-SAT

Input: a 3-CNF formula
Want: whether there exists a satisfying assignment

\((x \lor y \lor z) \land (x \lor \overline{y} \lor \overline{z})\)
3-SAT

Input: a 3-CNF formula
Want: whether there exists a satisfying assignment

Brute force: Go through every possibility until we find a satisfying assignment
3-SAT

Input: a 3-CNF formula
Want: whether there exists a satisfying assignment

Brute force: Go through every possibility until we find a satisfying assignment

n variables, m clauses: $O(m \cdot 2^n)$
3-SAT

Input: a 3-CNF formula
Want: whether there exists a satisfying assignment

Brute force: Go through every possibility until we find a satisfying assignment

\[ (x \lor y \lor z) \land (x \lor \overline{y} \lor \overline{z}) \]

n variables, m clauses: \( O(m \cdot 2^n) \)

Want to avoid going through all possible assignments
How will backtracking help?

\((x_1 \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2}) \land (\overline{x_1}) \land (x_1 \lor \overline{x_3})\)
How will backtracking help?

Try assigning a variable and look at the resulting formula:

\[ (x_1 \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2}) \land (\overline{x_1}) \land (x_1 \lor \overline{x_3}) \quad x_1 = 0 \]
How will backtracking help?

Try assigning a variable and look at the resulting formula:

- If a clause evaluates to 1, remove the entire clause

\[(x_1 \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2}) \land (\overline{x_1}) \land (x_1 \lor \overline{x_3})\]  

\[x_1 = 0\]
How will backtracking help?

Try assigning a variable and look at the resulting formula:

- If a clause evaluates to 1, remove the entire clause
- If a variable evaluates to 0, remove the variable

\[
(x_1 \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2}) \land (\overline{x_1}) \land (x_1 \lor \overline{x_3})
\]

\[
x_1 = 0
\]
How will backtracking help?

Try assigning a variable and look at the resulting formula:

- If a clause evaluates to 1, remove the entire clause
- If a variable evaluates to 0, remove the variable
- If we reach an empty clause (all variables in it are 0), then backtrack

\[(x_1 \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2}) \land (\overline{x_1}) \land (x_1 \lor \overline{x_3})\]

\[x_1 = 0\]
How will backtracking help?

Try assigning a variable and look at the resulting formula:
- If a clause evaluates to 1, remove the entire clause
- If a variable evaluates to 0, remove the variable
- If we reach an empty clause (all variables in it are 0), then backtrack

\[(x_1 \lor x_2 \lor x_3) \land (x_1 \lor \bar{x}_2) \land (\bar{x}_1) \land (x_1 \lor \bar{x}_3)\]

\[x_1 = 0\]
\((x_1 \vee x_2 \vee x_3) \land (x_1 \vee \overline{x_2}) \land (\overline{x_1}) \land (x_1 \vee \overline{x_3})\)
\[(x_1 \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2}) \land (\overline{x_1}) \land (x_1 \lor \overline{x_3})\]

\[x_1 = 0\]

\[(x_2 \lor x_3) \land (\overline{x_2}) \land (\overline{x_3})\]
\( (x_1 \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2}) \land (\overline{x_1}) \land (x_1 \lor \overline{x_3}) \)

\( x_1 = 0 \)

\( (x_2 \lor x_3) \land (\overline{x_2}) \land (\overline{x_3}) \)

\( x_2 = 0 \)

\( (x_3) \land (\overline{x_3}) \)
\[(x_1 \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2}) \land (\overline{x_1}) \land (x_1 \lor \overline{x_3})\]

- \(x_1 = 0\)
  
  \[(x_2 \lor x_3) \land (\overline{x_2}) \land (\overline{x_3})\]

- \(x_2 = 0\)

  \[(x_3) \land (\overline{x_3})\]

- \(x_3 = 0\)

  ()
\[(x_1 \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2}) \land (\overline{x_1}) \land (x_1 \lor \overline{x_3})\]

- \(x_1 = 0\)
  \[(x_2 \lor x_3) \land (\overline{x_2}) \land (\overline{x_3})\]

- \(x_2 = 0\)
  \[(x_3) \land (\overline{x_3})\]
- \(x_3 = 0\)
  ()
- \(x_3 = 1\)
  ()
\[ (x_1 \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2}) \land (\overline{x_1}) \land (x_1 \lor \overline{x_3}) \]

\[
x_1 = 0
\]

\[ (x_2 \lor x_3) \land (\overline{x_2}) \land (\overline{x_3}) \]

\[
x_2 = 0
\]

\[ x_3 = 0 \]

\[ () \land (\overline{x_3}) \]

\[ x_2 = 1 \]

\[ (x_3) \land (\overline{x_3}) \]

\[ x_3 = 1 \]

\[
\]
\[(x_1 \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2}) \land \overline{x_1} \land (x_1 \lor \overline{x_3})\]

- \(x_1 = 0\), then:
  - \((x_2 \lor x_3) \land (\overline{x_2}) \land (\overline{x_3})\)
    - \(x_2 = 0\), then:
      - \((x_3) \land (\overline{x_3})\)
        - \(x_3 = 0\), then: (),
        - \(x_3 = 1\), then: ()
    - \(x_2 = 1\), then: () \land (\overline{x_3})
      - ()
  - \(x_2 = 1\), then: ()

- \(x_1 = 1\), then: ()
The given expression is:

\[(x_1 \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2}) \land \overline{x_1} \land (x_1 \lor \overline{x_3})\]

### Decision Tree:

1. **Root Node:**
   - If \( x_1 = 0 \):
     - **Branch 1:**
       - \( x_2 = 0 \):
         - **Leaf:** \( (x_3) \land (\overline{x_3}) \)
         - Valid
         - \( x_3 = 0 \):
           - **Leaf:** \( () \land (\overline{x_3}) \)
           - Valid
         - \( x_3 = 1 \):
           - **Leaf:** \( () \land (\overline{x_3}) \)
           - Valid
     - **Branch 2:**
       - **Leaf:** \( () \land (\overline{x_3}) \)
       - Valid
   - **Branch 2:**
     - **Leaf:** \( () \land (\overline{x_3}) \)
     - Valid

2. **Root Node:**
   - If \( x_1 = 1 \):
     - **Branch 1:**
       - **Leaf:** \( () \land (\overline{x_3}) \)
       - Valid

### Conclusion:

The given expression is unsatisfiable.
3SAT(C):
    if C has no clauses: return true
    if C contains an empty clause: return false
    Choose an unassigned variable x
    if 3SAT(C(x = 0)) = true: return true
    if 3SAT(C(x = 1)) = true: return true
3SAT(C):
if C has no clauses: return true
if C contains an empty clause: return false
Choose an unassigned variable x
if 3SAT(C(x = 0)) = true: return true
if 3SAT(C(x = 1)) = true: return true

Correctness: proof by induction

Base case: C is empty
3SAT(C):
    if C has no clauses: return true
    if C contains an empty clause: return false
    Choose an unassigned variable x
    if 3SAT(C(x = 0)) = true: return true
    if 3SAT(C(x = 1)) = true: return true

Correctness: proof by induction

Base case: C is empty

Induction step: assume algorithm correctly determines whether or not there is an assignment for all formulas smaller than C
3SAT(C):
    if C has no clauses: return true
    if C contains an empty clause: return false
    Choose an unassigned variable x
    if 3SAT(C(x = 0)) = true: return true
    if 3SAT(C(x = 1)) = true: return true

Correctness: proof by induction

Base case: C is empty

Induction step: assume algorithm correctly determines whether or not there is an assignment for all formulas smaller than C
  - C(x=0) and C(x=1) are both smaller than C
  - By induction hypothesis, 3SAT(C(x=0)) and 3SAT(C(x=1)) are correct
  - The assignment for C would be the same, plus x = 0 (if 3SAT(C(x=0)) is true) or x = 1 (if 3SAT(C(x=1)) is true)