Theory of Computation: Key Concepts

Explore fundamental concepts in computer science. From finite automata to undecidable problems, we'll cover essential topics in computational theory.

By Shravan Bobade

Rollno.: TBCO22142

Finite Automata for Spell Checking

1

Defination

Finite Automata (FA) is a model of computation that processes strings of symbols and recognizes patterns by moving between states according to rules defined by the automaton.

2

Application in Spell Checker

A spell checker can be implemented using FA by treating each word in a dictionary as a path through the automaton. The FA transitions through states corresponding to each letter of a word. If a word exists in the language (dictionary), the automaton reaches an accepting state. If not, the word is flagged as incorrect.

3

Example

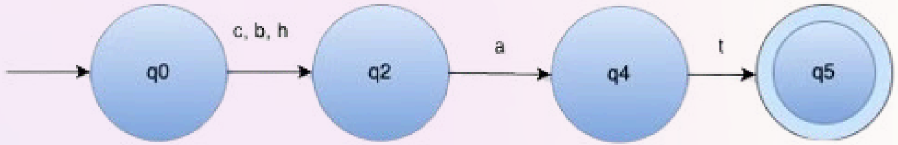
For the words "cat," "bat," or "hat," the FA would transition through states as follows:

- Start → c/b/h → a → t → Accepting state.

If a user types "cattt," "batt," or "hattt," the FA won't reach the accepting state, signaling a spelling error.

Σ = {c, a, t, b, h}

Current State	Input Symbol	Next State
Start	**c**	q1
Start	**b**	q2
Start	**h**	q3
q1	**a**	q4
q2	**a**	q4
q3	**a**	q4
q4	**t**	q5
Any State	Any other letter or extra input	Non-Accepting State



Example Transitions:

1. **Correct Input (cat):**
    - Input: **"cat"**
    - FA Transitions:
      Start → q1 → q2 → qAccept
2. **Correct Input (bat):**
    - Input: **"bat"**
    - FA Transitions:
      Start → q3 → q4 → qAccept
3. **Correct Input (hat):**
    - Input: **"hat"**
    - FA Transitions:
      Start → q5 → q6 → qAccept
4. **Incorrect Input ("rat"):**
    - Input: **"rat"**
    - FA Transitions:
      Start → (no transition for "r") → Non-Accepting State

Regular Expressions in Text Search & Replace

Definition

Regular Expressions (RE) are symbolic representations of string patterns used for matching specific sequences of characters within a text.

Application

REs are widely used in text processing tasks, such as search and replace functions in text editors. By using RE, users can find all instances of a word or pattern and replace them with another.

Example

To find all instances of "cat" and replace them with "dog" in a document, an RE could be written as **\bcat\b.** The **\b** ensures that only whole words are matched. This can then be replaced with "dog."

Problem Statement:

Find all strings where the word "cat" appears at least once, either at the beginning, middle, or end of the string, and may be mixed with other characters or patterns. Represent "cat" as 'a' and other characters as 'b'. Use the regular expression:

(a+b)*a+(a+b)*

Solution:

Given the regular expression:

(a+b)*a+(a+b)*

Interpretation:

- **(a+b)**: Represents any sequence of "cat" (a) and other characters (b).
- **a+**: Matches one or more occurrences of "cat" (represented by 'a').
- **(a+b)***: Ensures there can be any combination of "cat" or other characters before or after "cat."

Examples of matching strings:

1. **"bcatb"**:
   - Here, **b** represents other characters before and after "cat."
2. **"bbbcat"**:
   - "cat" appears at the end of the string.
3. **"bbcatbbcat"**:
   - "cat" appears in the middle and again at the end, satisfying the condition of appearing at least once.

Examples of non-matching strings:

- **"bbbb"**:
  - No occurrence of "cat," so it does not match.
- **"abc"**:
  - "cat" does not appear, so this also does not match.

Context-Free Grammar for Palindromes

1

Defination

Context-Free Grammar (CFG) is a type of grammar used to define the syntax of programming languages and natural languages. CFGs can describe recursive structures, such as palindromes.

2

Generate

CFG can be used to generate palindromes, strings that read the same forward and backward. For example, a CFG that generates palindromes could use the following rules:

3

CFG for palindromes:

S → aSa | bSb | ε

Where:

- S generates the palindrome.
- aSa or bSb adds a character at both ends of the palindrome.
- ε is the empty string.

4

Example

For the string "abba," the derivation using the CFG would be:

- S → aSa → abSba → abba

Pushdown Automata for Parsing

Defination

Pushdown Automata (PDA) is an extension of finite automata with a stack, allowing it to handle context-free languages. PDAs are essential for parsing tasks in compilers.

Top-Down Parsing

Top-down parsing builds the parse tree from the root (start symbol) down to the leaves, using a leftmost derivation. The parser tries to match the input string with a production rule by expanding non-terminals.

Bottom-Up Parsing

Bottom-up parsing, also known as shift-reduce parsing, constructs the parse tree from the leaves (input symbols) up to the root (start symbol) by reducing substrings of the input to non-terminals.

PDA Simulation

In top-down parsing, the PDA pushes symbols onto the stack based on expected inputs. In bottom-up parsing, the PDA shifts symbols onto the stack and reduces them based on grammar rules until it arrives at the start symbol.

Top-Down Parsing

Example: Parse the string "ab" using the following grammar:

Grammar:

1. S → A B
2. A → a
3. B → b

Steps:

- Start with the Start Symbol:
  Begin by placing the start symbol S on the stack.
  Stack: S
- Expand S → A B:
  Apply the production S → A B. Replace S with A B on the stack.
  Stack: A B
- Match A → a:
  The leftmost non-terminal on the stack is A. Since the next input symbol is "a", we apply A → a and match A with a.
  Stack: B
  Input: "b"
- Match B → b:
  The next non-terminal on the stack is B. Since the input symbol is "b", we apply B → b and match B with b.
  Stack: (empty)
  Input: (empty)

Result: The string "ab" is successfully parsed by expanding the non-terminals according to the grammar, following a top-down approach.

Bottom-Up Parsing

Example: Parse the string "ab" using the following grammar:

Grammar:

1. S → A B
2. A → a
3. B → b

Steps:

- Start with the input string:
  Input: "ab"
- Shift "a":
  The first symbol "a" is shifted (read from input and pushed onto the stack).
  Stack: a
- Reduce "a" → A:
  Apply rule A → a to reduce a to A.
  Stack: A
- Shift "b":
  Next, shift the symbol "b".
  Stack: A b
- Reduce "b" → B:
  Apply rule B → b to reduce b to B.
  Stack: A B
- Reduce "A B" → S:
  Finally, apply rule S → A B to reduce A B to S.
  Stack: S

Result: The string "ab" is successfully parsed by reducing the input from the bottom (leaves) up to the start symbol S.

PDA Simulation

Example: Parse the string "(()())"

- States:

  Q = {q0, q1, qAccept}

  (q0 is the start state, qAccept is the accepting state)
- Alphabet:

  Σ = { (,) }
- Stack Alphabet:

  Γ = { (, Z }

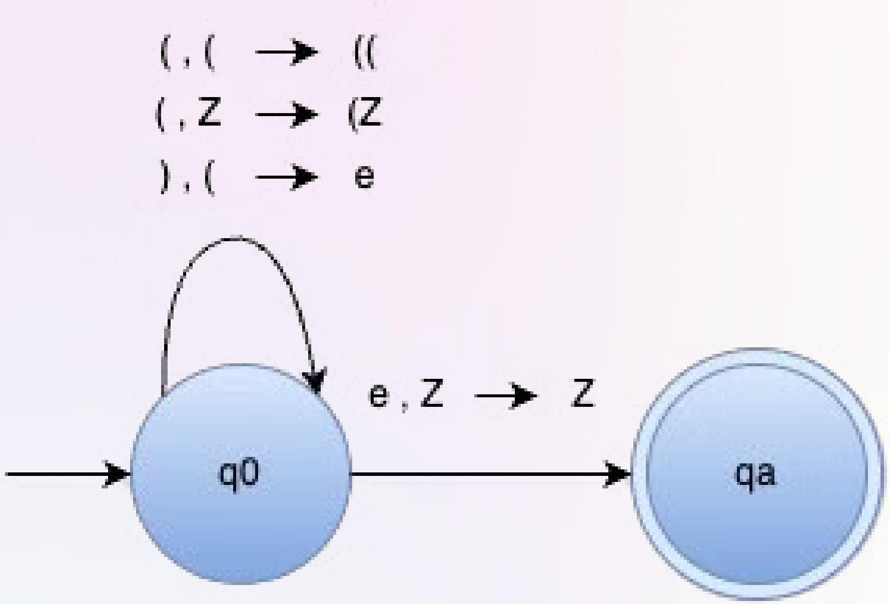
  (Z is the initial stack symbol)
- Transitions:

  δ(q0, "(", Z) → (q0, (Z)

  δ(q0, "(", "(") → (q0, (( )

  δ(q0, ")", "(") → (q0, ε)

  δ(q0, ε, Z) → (qAccept, Z)



Turing Machines and Algorithms

1

Definition

A Turing Machine is a theoretical computational model that can simulate any algorithm. It consists of a tape (infinite memory) and a head that reads and writes symbols on the tape.

2

Application

Turing Machines are used to solve various computational problems by defining step-by-step rules for modifying the tape based on the current state.

3

Example

A simple Turing Machine for the successor function scans a unary number, moves to the rightmost 1, and appends an additional 1. The machine then halts, producing the unary number incremented by 1.

Example:

- Construct a TM for a successor function for a given unary number i.e. f(1) = n+1

The input tape consists of 4 the successor function will give output as 5.

1111# move to right by keeping all I's as it is

↑

1111# Move right

↑

1111# Move right

↑

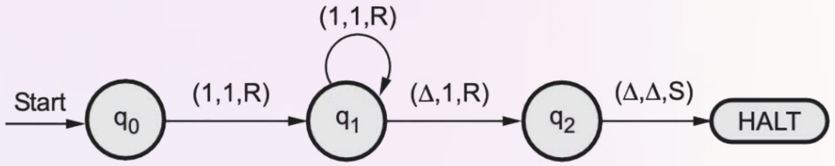
1111# Move right

↑

1111# Convert # to 1

↑

111114 HALT



Traveling Salesman Problem

1

Definition:

The Traveling Salesman Problem is an NP-hard optimization problem. It seeks to find the shortest possible route that visits each city exactly once and returns to the origin city.

2

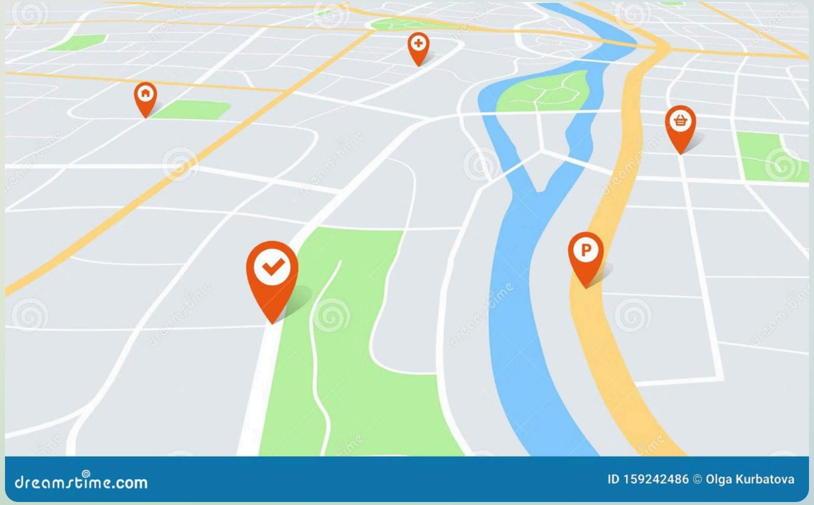
Application in TOC:

Though TSP is computationally hard, approximation algorithms and heuristic methods are used to find near-optimal solutions. TSP is an example of problems whose complexity is studied in the context of computational limits (P vs NP).

3

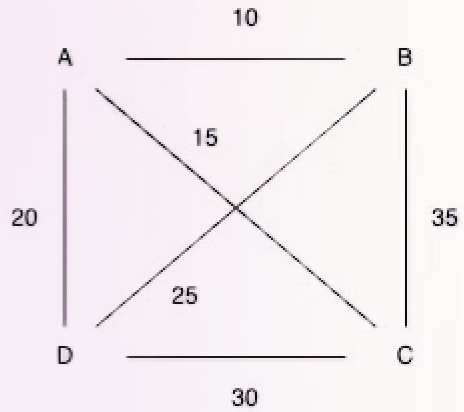
Example:

Given a list of cities, the TSP algorithm tries to find the shortest path that covers all cities without revisiting any.



Steps:

1. Calculate the total distance for each route:
   - Route 1 (A → B → C → D → A):
     Distance = A → B (10) + B → C (35) + C → D (30) + D → A (20) = 95
   - Route 2 (A → B → D → C → A):
     Distance = A → B (10) + B → D (25) + D → C (30) + C → A (15) = 80
   - Route 3 (A → C → B → D → A):
     Distance = A → C (15) + C → B (35) + B → D (25) + D → A (20) = 95
   - Route 4 (A → C → D → B → A):
     Distance = A → C (15) + C → D (30) + D → B (25) + B → A (10) = 80
   - Route 5 (A → D → B → C → A):
     Distance = A → D (20) + D → B (25) + B → C (35) + C → A (15) = 95
   - Route 6 (A → D → C → B → A):
     Distance = A → D (20) + D → C (30) + C → B (35) + B → A (10) = 95
2. Find the shortest route: The shortest routes are:
   - A → B → D → C → A with a total distance of 80
   - A → C → D → B → A with a total distance of 80



Result:

- The optimal route for the Traveling Salesman Problem in this example is either A → B → D → C → A or A → C → D → B → A, with a minimum total distance of 80.

Post Correspondence Problem

1

Definition:

The Post Correspondence Problem (PCP) is an undecidable problem that asks whether a set of string pairs can be arranged so that the concatenation of the first elements of the pairs equals the concatenation of the second elements.

2

Application in TOC:

PCP demonstrates that some problems cannot be solved by any algorithm, which is a key concept in understanding the limits of computation.

3

Example:

Given two sequences of strings, A = {a, ab}, B = {aa, b}, the question is whether a sequence can be found such that A and B concatenate to the same string. In this case, no such sequence exists, demonstrating undecidability.

Example:

Given two lists of strings, A and B, find a sequence of indices such that the concatenation of the strings from A and B results in identical strings.

- List A:
  A1 = "a", A2 = "ab", A3 = "b"
- List B:
  B1 = "ab", B2 = "a", B3 = "b"

Steps:

1. Start with an empty sequence of indices.
2. Try the sequence of indices:
   i1 = 1, i2 = 2, i3 = 3.
   - Concatenation of A:
     A1 + A2 + A3 = "a" + "ab" + "b" = "aabb"
   - Concatenation of B:
     B1 + B2 + B3 = "ab" + "a" + "b" = "aabb"
3. The concatenation of strings from List A and List B using the sequence 1, 2, 3 gives the same result: "aabb".

Result:

The sequence i1 = 1, i2 = 2, i3 = 3 satisfies the condition, as the concatenation of strings from both lists results in the same string. Therefore, (1, 2, 3) is a solution to the Post Correspondence Problem in this case.

Thank you!

Made with Gamma