**Lecture 20**

**Review**

- Algorithm design paradigm: Dynamic Programming, amortized computing
- Algorithms covered:
  - Divide and conquer: Binomial, Trinomial, Fibonacci, Fast Exponentiation
  - Greedy: Fractional Knapsack, Huffman Coding
  - Divide and conquer + Dynamic Programming
- NP-Completeness:
  - Reducibility, Isomorphism, Graph Coloring, Number Partitioning
- Approximation:
  - Approximation Algorithms: Greedy, Randomization
  - Approximation hardness of problems: On the hardness of approximating problems, hardness of approximation

**Algorithm and Paradigm**

**Dynamic Programming**

1. **KMP algorithm**
   
   - Given a pattern and a text string, find the first occurrence of the pattern.
   - The KMP algorithm works by pre-computing the partial match table.
   - The table helps in finding the next position in the pattern without re-computation.
   - Example:
     
     ```
     T: banana banana
     P: x
     i = 0, j = 0
     D[i][j] = 0
     i = 1, j = 0
     D[i][j] = 0
     i = 2, j = 1
     D[i][j] = 1
     i = 3, j = 1
     D[i][j] = 2
     i = 4, j = 2
     D[i][j] = 3
     i = 5, j = 3
     D[i][j] = 4
     i = 6, j = 4
     D[i][j] = 5
     i = 7, j = 5
     D[i][j] = 6
     i = 8, j = 6
     D[i][j] = 7
     i = 9, j = 7
     D[i][j] = 8
     i = 10, j = 8
     D[i][j] = 9
     ```
   - The first occurrence of the pattern is at position 3 in the text.

2. **Boyer-Moore algorithm**
   
   - The Boyer-Moore algorithm uses a different approach to find the pattern.
   - It skips positions in the text based on the pattern.
   - Example:
     
     ```
     Text: the quick brown fox
     Pattern: quick brown
     ```
     - First, skip the first occurrence of "quick" from position 1 to position 6.
     - Then, match the rest of the pattern.
     - The first occurrence is at position 2.

**Example questions:**

- Given a pattern and a text string, find the first occurrence of the pattern.
- Dynamic programming:
  - Procedure, and
  - Applications: string alignment, e.g., sw5-pb
- Polynomials and Matrices:
  - Understand Horner’s rule and Strassen’s algorithm
- NP-complete:
  - Concepts: P, NP, NP-Complete, NP-hard, polynomial reduction
  - Given a problem, prove it is NP complete
  - Given problems A and B, prove A is no harder than B.
- Parallel algorithms:
  - PRAM
  - NC class
  - Design parallel algorithms that solve a given problem using specified model (or variations).

---

**The KMP algorithm**

1. Skip outer iteration
   
   ```
   i = 3
   ```

2. Skip first inner iteration testing “n” vs “n” at outer iteration 1: 4

3. 9 comparisons are made for the first occurrence.

---

**The Boyer-Moore algorithm**

- If you wish to understand others you must

The numbers of positions we can “jump” forward when there is a mismatch depends on the test character being mad, say \( T_j \), more precisely, depends on \( T_j \)’s occurrence in pattern \( P \).

Only 18 comparisons are needed to find the first occurrence.

---

**Example questions:**

- Given a pattern and a text string, find the first occurrence of the pattern.
- Dynamic programming:
  - Procedure, and
  - Applications: string alignment, e.g., sw5-pb
- Polynomials and Matrices:
  - Understand Horner’s rule and Strassen’s algorithm
- NP-complete:
  - Concepts: P, NP, NP-Complete, NP-hard, polynomial reduction
  - Given a problem, prove it is NP complete
  - Given problems A and B, prove A is no harder than B.
- Parallel algorithms:
  - PRAM
  - NC class
  - Design parallel algorithms that solve a given problem using specified model (or variations).

---

**Example problems:**

- Given a pattern and a text string, find the first occurrence of the pattern.
- Dynamic programming:
  - Procedure, and
  - Applications: string alignment, e.g., sw5-pb
- Polynomials and Matrices:
  - Understand Horner’s rule and Strassen’s algorithm
- NP-complete:
  - Concepts: P, NP, NP-Complete, NP-hard, polynomial reduction
  - Given a problem, prove it is NP complete
  - Given problems A and B, prove A is no harder than B.
- Parallel algorithms:
  - PRAM
  - NC class
  - Design parallel algorithms that solve a given problem using specified model (or variations).

---

**Example problems:**

- Given a pattern and a text string, find the first occurrence of the pattern.
- Dynamic programming:
  - Procedure, and
  - Applications: string alignment, e.g., sw5-pb
- Polynomials and Matrices:
  - Understand Horner’s rule and Strassen’s algorithm
- NP-complete:
  - Concepts: P, NP, NP-Complete, NP-hard, polynomial reduction
  - Given a problem, prove it is NP complete
  - Given problems A and B, prove A is no harder than B.
- Parallel algorithms:
  - PRAM
  - NC class
  - Design parallel algorithms that solve a given problem using specified model (or variations).
Horners algorithm

To observe that \( p(x) \) can be rewritten as follows.

\[
p(x) = (\ldots (a_n x + a_{n-1}) x + a_{n-2}) x + \ldots + a_1) x + a_0.
\]

Horners poly(a, n, x)

1. \( p = a[n] \)
2. for i ← n-1 to 0
3. \( p = p \times x + a[i] \)
4. return p

Running time: \( n \) multiplications and \( n \) additions.

---

**Handling Write Conflicts**

- **CREW (Concurrent Read, Exclusive Write):** only one processor write in a particular cell at any one step; It is illegal to have more than one processor write to one cell at the same time.
- **CRCW (Concurrent Read, Concurrent Write):**
  - Common-Write model
  - Arbitrary-Write model
  - Priority-Write model

---

**Example:** A dominant set of a graph \( G \) is a subset of vertices \( W \) such that every vertex of \( G \) is either in \( W \) or is adjacent to a vertex in \( W \).

**Decision Problem**

- Input: A graph \( G \) and integer \( k \)
- Question: Does \( G \) have a dominant set of size at most \( k \)?

**Optimization Problem:**

- Input: A graph \( G \)
- Output: A smallest dominant set of \( G \)

**The Claim:** The optimization problem for dominant set is no harder than the decision problem.

**Proof:** Use Polynomial reduction:

1. For \( k \) from 1 to \( |G| \), call the algorithm for the decision problem as a subroutine on input \( (G, k) \).
2. Initialize all vertices in \( G \) as unmarked. Now choose an unmarked vertex \( v \) in \( G \) and add a new vertex \( X \) attached by an edge to \( v \). Ask the decision algorithm if the modified graph \( G' \) has a dominant set of size \( k \). If yes, mark \( v \) as a member and remove the vertex \( X \). Repeat until \( k \) vertices are marked as members of the dominant set.

---

**Performance:** \( O(g(n)) \) time, \( n \) processors.
Problem: Boolean AND on n Bits
Input: Bits $x_0, \ldots, x_{n-1}$, as 0’s and 1’s, in $M[0], \ldots, M[n-1]$
Output: $x_0 \land \ldots \land x_{n-1}$ in $M[0]$

Solution 2 (CRCW)
commonWriteOr($M$, $n$)
P: reads $x_i$ from $M[i]$;
    If $x_i$ is 0; then P writes 0 in $M[0]$.

Correctness: Since all the processors that write into $M[0]$ write the
same value, the program is a legal program under Common-Write
model.
Running time: It takes just two steps, regardless the input size.