



DIGITAL ASSIGNMENT 2 [BACKTRACKING AND DYNAMIC PROGRAMMING]

CSE2012–DESIGN AND ANALYSIS OF ALGORITHMS (L25-26)[MRS GAYATHRI P]



**FEBURARY 10, 2023
ANIRUDH VADERA
20BCE2940**

QUESTION 1:

Design an algorithm and implement matrix chain multiplication problem using dynamic programming approach.

PSEUDOCODE:

Algorithm:

The basic idea behind dynamic programming for matrix chain multiplication is to build a table or array that stores the optimal cost of multiplying each subchain of matrices. The table is filled in a bottom-up manner, starting with subchains of length 2 and gradually building up to the full chain of matrices.

At each step, we consider all possible ways to split the subchain into two smaller subchains, compute the cost of multiplying each smaller subchain, and then add the cost of multiplying the resulting two matrices. We then choose the split that yields the minimum cost and store that cost in the table.

By storing the optimal costs of each subchain, we can avoid computing the same subproblem multiple times and thus achieve a more efficient solution.

We can start filling up the array dp with base cases, where $dp[i][i]$ equals zero since there is no multiplication involved for a single matrix.

- Iterate from $l = 2$ to $N-1$ which denotes the length of the range:
 - Iterate from $i = 0$ to $N-1$:
 - Find the right end of the range (j) having l matrices.
 - Iterate from $k = i+1$ to j which denotes the point of partition.
 - Multiply the matrices in range (i, k) and (k, j) .
 - This will create two matrices with dimensions $arr[i-1]*arr[k]$ and $arr[k]*arr[j]$.
 - The number of multiplications to be performed to multiply these two matrices (say X) are $arr[i-1]*arr[k]*arr[j]$.
 - The total number of multiplications is $dp[i][k] + dp[k+1][j] + X$.
- The value stored at $dp[1][N-1]$ is the required answer.

Time Complexity: $O(N^3)$

Auxiliary Space: $O(N^2)$

ANIRUDH VADERA
DIGITAL ASSIGNMENT 2 [BACKTRACKING AND DYNAMIC PROGRAMMING]

SOURCE CODE:

```
#include <bits/stdc++.h>
#include <algorithm>
using namespace std;

// Driver Code
int main()
{
    cout << "Enter the Number of Elements of the dimension array : ";
    int n;
    cin >> n;
    int *arr = (int *)malloc(n * sizeof(int));
    cout << "Enter the elements of the dimension array : "
         << "\n";
    for (int i = 0; i < n; i++)
    {
        cout << "Element "
              << i << " : ";
        cin >> arr[i];
    }

    // Main Logic
    /* One extra row and one extra column are
    allocated in dp[][]. 0th row and 0th
    column of dp[][] are not used */
    // The DP matrix -> Used for memoization
    int dp[n][n];
    int i, j, k, L;
    /* dp[i, j] = Minimum number of scalar
    multiplications needed to compute the
    matrix A[i]A[i+1]...A[j] = A[i..j] where
    dimension of A[i] is arr[i-1] x arr[i] */
    // arr is the given dimension array

    // cost is zero when multiplying
    // one matrix.
    for (i = 1; i < n; i++)
    {
        dp[i][i] = 0;
    }

    // L is chain length.
    for (L = 2; L < n; L++)
    {
        for (i = 1; i < n - L + 1; i++)
```

ANIRUDH VADERA
DIGITAL ASSIGNMENT 2 [BACKTRACKING AND DYNAMIC PROGRAMMING]

```
{
    j = i + L - 1;
    // Initialize with a very high value so that we can find minimum
    dp[i][j] = INT_MAX;
    for (k = i; k <= j - 1; k++)
    {
        dp[i][j] = std::min(dp[i][j], dp[i][k] + dp[k + 1][j] + arr[i
- 1] * arr[k] * arr[j]);
    }
}
}
cout << "Minimum number of multiplications is : " << dp[1][n - 1];
getchar();
return 0;
}
```

OUTPUT SCREENSHOT:

```
PS C:\Users\Anirudh\OneDrive\Desktop\c codes\DAA\AS03\output> cd 'c:\Users\Anirudh\OneDrive\Desktop\c codes\DA
A\AS03\output'
● PS C:\Users\Anirudh\OneDrive\Desktop\c codes\DAA\AS03\output> & .\'matrixChainMultiplication.exe'
Enter the Number of Elements of the dimension array : 6
Enter the elements of the dimension array :
Element 0 : 4
Element 1 : 10
Element 2 : 3
Element 3 : 12
Element 4 : 20
Element 5 : 7
○ Minimum number of multiplications is : 1344
PS C:\Users\Anirudh\OneDrive\Desktop\c codes\DAA\AS03\output> █
```

QUESTION 2:

Implement N-Queens problem using backtracking technique.

PSEUDOCODE:

The idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes with already placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes, then we backtrack and return false.

- Place the queens column wise, start from the left most column
- If all queens are placed:
 return true and print the solution matrix.
 Else
 Try all the rows in the current column.
- Check if queen can be placed here safely if yes mark the current cell in solution matrix as 1 and try to solve the rest of the problem recursively.
- If placing the queen in above step leads to the solution return true.
- If placing the queen in above step does not lead to the solution , BACKTRACK, mark the current cell in solution matrix as 0 and return false.
- If all the rows are tried and nothing worked, return false and print NO SOLUTION.

Time Complexity: $O(N!)$

Auxiliary Space: $O(N^2)$

SOURCE CODE:

```
#include <stdio.h>
#include <math.h>

// Stores the number of Solutions
int count = 0;

// Printing the solution
void print(int n, int **board)
{
    printf("\n\nSolution %d:\n\n", ++count);
```

ANIRUDH VADERA
DIGITAL ASSIGNMENT 2 [BACKTRACKING AND DYNAMIC PROGRAMMING]

```
for (int i = 1; i <= n; i++)
{
    printf("\t%d", i);
}

for (int i = 1; i <= n; i++)
{
    printf("\n\n%d", i);
    for (int j = 1; j <= n; ++j)
    {
        if (board[i] == j)
        {
            printf("\tQ");
        }
        else
        {
            printf("\t-");
        }
    }
}

// function to check for proper positioning of queen
void queen(int *board, int row, int n)
{
    int column;
    for (column = 1; column <= n; ++column)
    {
        if (place(board, row, column))
        {
            board[row] = column; // no conflicts so place queen
            if (row == n)
            {
                print(n, board); // printing the board configuration
            }
            else
            {
                queen(board, row + 1, n); // try queen with next position
            }
        }
    }
}

int main()
{
    int n;
    printf("\n\nEnter number of Queens:");
    scanf("%d", &n);
```

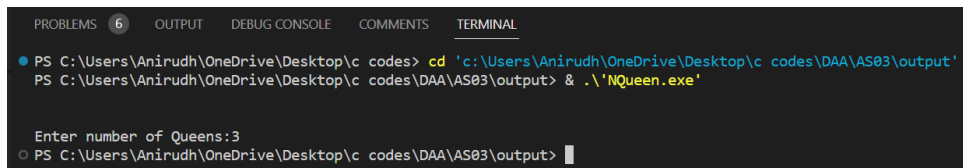
ANIRUDH VADERA
DIGITAL ASSIGNMENT 2 [BACKTRACKING AND DYNAMIC PROGRAMMING]

```
// Creating the board
int *board = (int *)malloc(n * sizeof(int));
queen(board, 1, n);
return 0;
}

/*function to check conflicts
If no conflict for desired position returns 1 otherwise returns 0*/
int place(int *board, int row, int column)
{
    for (int i = 1; i <= row - 1; ++i)
    {
        // checking column and diagonal conflicts
        if (board[i] == column)
        {
            return 0;
        }
        else if (abs(board[i] - column) == abs(i - row))
        {
            return 0;
        }
    }
    return 1; // no conflicts
}
```

OUTPUT SCREENSHOT:

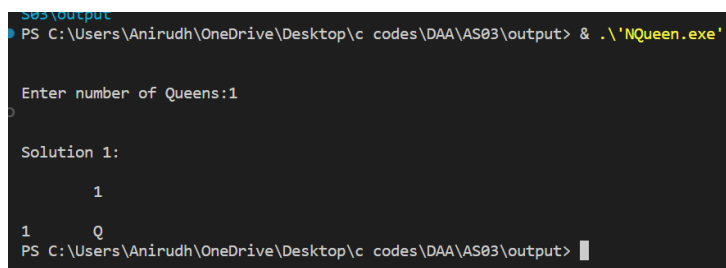
For 3 queens No Solutions Possible:



```
PROBLEMS 6 OUTPUT DEBUG CONSOLE COMMENTS TERMINAL
PS C:\Users\Anirudh\OneDrive\Desktop\c codes> cd 'c:\Users\Anirudh\OneDrive\Desktop\c codes\DAA\AS03\output'
PS C:\Users\Anirudh\OneDrive\Desktop\c codes\DAA\AS03\output> & .\NQueen.exe

Enter number of Queens:3
PS C:\Users\Anirudh\OneDrive\Desktop\c codes\DAA\AS03\output> 
```

For 1 queens (1 Solution Possible):



```
PS C:\Users\Anirudh\OneDrive\Desktop\c codes\DAA\AS03\output> & .\NQueen.exe

Enter number of Queens:1

Solution 1:

1

1 Q
PS C:\Users\Anirudh\OneDrive\Desktop\c codes\DAA\AS03\output> 
```

ANIRUDH VADERA
DIGITAL ASSIGNMENT 2 [BACKTRACKING AND DYNAMIC PROGRAMMING]

For 6 queens (4 Solution Possible):

```
PS C:\Users\Anirudh\OneDrive\Desktop\c codes\DAA\AS03\output> & .\'NQueen.exe'  
  
Enter number of Queens:6  
  
Solution 1:  


|   | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | - | Q | - | - | - | - |
| 2 | - | - | - | Q | - | - |
| 3 | - | - | - | - | - | Q |
| 4 | Q | - | - | - | - | - |
| 5 | - | - | Q | - | - | - |
| 6 | - | - | - | - | Q | - |


```

```
Solution 2:  


|   | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | - | - | Q | - | - | - |
| 2 | - | - | - | - | - | Q |
| 3 | - | Q | - | - | - | - |
| 4 | - | - | - | - | Q | - |
| 5 | Q | - | - | - | - | - |
| 6 | - | - | - | Q | - | - |

  
Solution 3:  


|   | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | - | - | - | Q | - | - |
| 2 | Q | - | - | - | - | - |
| 3 | - | - | - | - | Q | - |
| 4 | - | Q | - | - | - | - |
| 5 | - | - | - | - | - | Q |
| 6 | - | - | Q | - | - | - |

  
Solution 4:  


|   | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | - | - | - | - | Q | - |
| 2 | - | - | Q | - | - | - |
| 3 | Q | - | - | - | - | - |
| 4 | - | - | - | - | - | Q |
| 5 | - | - | - | Q | - | - |
| 6 | - | Q | - | - | - | - |

  
PS C:\Users\Anirudh\OneDrive\Desktop\c codes\DAA\AS03\output>
```


QUESTION 3:

Design an algorithm using Naïve approach to check whether given pattern P is plagiarized in given Text T.

PSEUDOCODE:

Inputs:

Text (string)

Pattern (string)

Output:

List of starting positions of the pattern in the text

Initialize an empty list to store the starting positions of the pattern in the text.

Calculate the length of the text and the pattern.

Loop through the text from 0 to $n - m$, where n is the length of the text and m is the length of the pattern.

For each index i in the loop, compare the pattern with the substring of the text starting at index i and having length m .

If the substring and the pattern are identical, add the index i to the list of starting positions.

Continue the loop until all possible substrings of the text have been compared with the pattern.

Return the list of starting positions.

Algorithm-NAVE_STRING_MATCHING (T, P)

```
for  $i \leftarrow 0$  to  $n - m$  do
    if  $P[1 \dots m] == T[i+1 \dots i+m]$  then
        print "Match Found"
    end if
end
```

Time Complexity = $O(m * (n - m))$

SOURCE CODE:

```
#include <bits/stdc++.h>
using namespace std;

// Driver's Code
int main()
{
    int l1, l2;
    cout << "Enter the length of the Text : ";
    cin >> l1;
    char *text = (char *)malloc(l1 * sizeof(char));
    cout << "Enter the Text : ";
    cin >> text;
    cout << "Enter the length of the Pattern : ";
    cin >> l2;
    char *pattern = (char *)malloc(l2 * sizeof(char));
    cout << "Enter the Pattern : ";
    cin >> pattern;
    /* A loop to slide pat[] one by one */
    int f = 0;
    for (int i = 0; i <= l1 - l2; i++)
    {
        int j;
        /* Check for pattern starting from index i */
        for (j = 0; j < l2; j++)
        {
            if (text[i + j] != pattern[j])
            {
                break;
            }
        }

        if (j == l2) // if we found a match
        {
            f = 1;
            cout << "Pattern found at index : " << i << endl;
        }
    }
    if (f == 0)
    {
        cout << "The pattern is not in given Text";
    }
    return 0;
}
```

ANIRUDH VADERA
DIGITAL ASSIGNMENT 2 [BACKTRACKING AND DYNAMIC PROGRAMMING]

OUTPUT SCREENSHOT:

```
AS03\output
● PS C:\Users\Anirudh\OneDrive\Desktop\c codes\DAA\AS03\output> & .\'patternMatching.exe'
Enter the length of the Text : 18
Enter the Text : AABAACAADAABAAABAA
Enter the length of the Pattern : 4
Enter the Pattern : AABA
○ Pattern found at index : 0
Pattern found at index : 9
Pattern found at index : 13
PS C:\Users\Anirudh\OneDrive\Desktop\c codes\DAA\AS03\output> |
```

```
AS03\output
● PS C:\Users\Anirudh\OneDrive\Desktop\c codes\DAA\AS03\output> & .\'patternMatching.exe'
Enter the length of the Text : 21
Enter the Text : ABAAABCDDBBABCDDDEBCABC
Enter the length of the Pattern : 3
Enter the Pattern : ABC
○ Pattern found at index : 4
Pattern found at index : 10
Pattern found at index : 18
PS C:\Users\Anirudh\OneDrive\Desktop\c codes\DAA\AS03\output> |
```

Pattern Not in Text:

```
AS03\output
● PS C:\Users\Anirudh\OneDrive\Desktop\c codes\DAA\AS03\output> & .\'patternMatching.exe'
Enter the length of the Text : 5
Enter the Text : ABCDE
Enter the length of the Pattern : 3
Enter the Pattern : FGH
○ The pattern is not in given Text
PS C:\Users\Anirudh\OneDrive\Desktop\c codes\DAA\AS03\output> |
```

Compiled successfully

QUESTION 4:

Implement Rabin Karp algorithm to check whether given pattern P is plagiarized in given Text T.

PSEUDOCODE:

The Rabin-Karp algorithm uses the hash values of the pattern and the text to quickly compare substrings. If the hash values match, the algorithm compares the actual substrings to ensure a match. If the hash values do not match, the algorithm uses a rolling hash function to compute the hash value for the next substring. This approach can be more efficient than traditional pattern matching algorithms for large texts and patterns, where computing the hash values is faster than comparing all substrings.

Input: A text string T and a pattern string P.

1. Compute the hash value $H(P)$ for the pattern string P.
2. Compute the hash value $H(T[0..m-1])$ for the first m characters of the text string T, where m is the length of the pattern string P.
3. For $i = 0$ to $n-m$ do the following:
 - a. If $H(P) = H(T[i..i+m-1])$, then compare each character of P with the corresponding character of $T[i..i+m-1]$. If all characters match, then the pattern P is found at position i in the text string T.
 - b. If $H(P) \neq H(T[i..i+m-1])$, then compute the hash value $H(T[i+1..i+m])$ for the next m characters of the text string T using the rolling hash function.
 - c. Repeat step 3a and 3b until a match is found or all possible positions in the text string T have been checked.
4. If the pattern P is not found in the text string T, return "Pattern not found". Otherwise, return the position(s) in T where the pattern occurs.

Time Complexity:

Best/Average Case: $O(n+m)$

Worst Case: $O(nm)$

Auxiliary Space: $O(1)$

SOURCE CODE:

```
#include <bits/stdc++.h>
using namespace std;

#define d 10 // any arbitrary value

void rabinKarp(char *pattern, char *text, int q, int l1, int l2)
{
    int p = 0;
    int t = 0;
    int h = 1; // hash value
    int j;     // Looping variable

    int f = 0; // Flag variable

    for (int i = 0; i < l2 - 1; i++)
    {
        h = (h * d) % q;
    }

    // Calculate hash value for pattern and text
    for (int i = 0; i < l2; i++)
    {
        p = (d * p + pattern[i]) % q;
        t = (d * t + text[i]) % q;
    }

    for (int i = 0; i <= l1 - l2; i++)
    {
        if (p == t)
        {
            for (j = 0; j < l2; j++)
            {
                if (text[i + j] != pattern[j])
                    break;
            }

            if (j == l2) // The match is found
            {
                f = 1;
                printf("Pattern is found at index: %d \n", i);
            }
        }

        if (i < l1 - l2)
        {

```

ANIRUDH VADERA
DIGITAL ASSIGNMENT 2 [BACKTRACKING AND DYNAMIC PROGRAMMING]

```
        t = (d * (t - text[i] * h) + text[i + l2]) % q;

        if (t < 0)
        {
            t = (t + q);
        }
    }
}
if (f == 0)
{
    printf("The pattern is not in given Text");
}
}

int main()
{
    int l1, l2;
    cout << "Enter the length of the Text : ";
    cin >> l1;
    char *text = (char *)malloc(l1 * sizeof(char));
    cout << "Enter the Text : ";
    cin >> text;
    cout << "Enter the length of the Pattern : ";
    cin >> l2;
    char *pattern = (char *)malloc(l2 * sizeof(char));
    cout << "Enter the Pattern : ";
    cin >> pattern;
    int q = 13;
    rabinKarp(pattern, text, q, l1, l2);
}
```

OUTPUT SCREENSHOT:

```
S03\output
PS C:\Users\Anirudh\OneDrive\Desktop\c codes\DAA\AS03\output> & .\'rabinKarp.exe'
Enter the length of the Text : 18
Enter the Text : AABAACAADAABAAABAA
Enter the length of the Pattern : 4
Enter the Pattern : AABA
Pattern is found at index: 0
Pattern is found at index: 9
Pattern is found at index: 13
PS C:\Users\Anirudh\OneDrive\Desktop\c codes\DAA\AS03\output> █
```

ANIRUDH VADERA
DIGITAL ASSIGNMENT 2 [BACKTRACKING AND DYNAMIC PROGRAMMING]

```
PS C:\Users\Anirudh\OneDrive\Desktop\c codes\DAA\AS03\output> & .\'rabinKarp.exe'  
Enter the length of the Text : 21  
Enter the Text : ABAAABCDBBABCDDDEBCABC  
Enter the length of the Pattern : 3  
Enter the Pattern : ABC  
○ Pattern is found at index: 4  
Pattern is found at index: 10  
Pattern is found at index: 18  
PS C:\Users\Anirudh\OneDrive\Desktop\c codes\DAA\AS03\output> █
```

Pattern Not in Text:

```
S03\output'  
PS C:\Users\Anirudh\OneDrive\Desktop\c codes\DAA\AS03\output> & .\'rabinKarp.exe'  
● Enter the length of the Text : 5  
Enter the Text : ABCDE  
Enter the length of the Pattern : 3  
Enter the Pattern : FGH  
○ The pattern is not in given Text  
PS C:\Users\Anirudh\OneDrive\Desktop\c codes\DAA\AS03\output> █
```

i Compiled successfully!