**QUESTION:**

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

**PSEUDOCODE:**

function findOptimalSolution(sTable, index, secondaryIndex):

    if index = secondaryIndex print("A" + index)

    else:

        print("(")

        findOptimalSolution(sTable, index, sTable[index][secondaryIndex])

        findOptimalSolution(sTable, sTable[index][secondaryIndex] + 1, secondaryIndex)

        print(")")

function matrixChainMultiplication(dimension, numberOfMatrices, mTable, sTable):

    for index from 1 to numberOfMatrices mTable[index][index] = 0

    for element from 2 to numberOfMatrices:

        for index from 1 to numberOfMatrices - element + 1:

            secondaryIndex = index + element - 1

            mTable[index][secondaryIndex] = maxValueOfIntDataType

            for tertiaryIndex from index to secondaryIndex - 1:

                temporaryValue = mTable[index][tertiaryIndex] + mTable[tertiaryIndex + 1][secondaryIndex] + dimension[index - 1] \* dimension[tertiaryIndex] \* dimension[secondaryIndex]

                if temporaryValue < mTable[index][secondaryIndex]:

                    mTable[index][secondaryIndex], sTable[index][secondaryIndex] = temporaryValue, tertiaryIndex

function main():

    numberOfMatrices = input()

    dimension[numberOfMatrices] = input()

    initialize sTable[100][100], mTable[100][100]

    matrixChainMultiplication(dimension, numberOfMatrices, mTable, sTable)

    print("Minimum number of scalar multiplications: mTable[1][numberOfMatrices]")

    print("Optimal solution: findOptimalSolution(sTable, 1, numberOfMatrices)")

    print("M table: ")

    for index from 1 to numberOfMatrices:

        for secondaryIndex from 1 to numberOfMatrices:

            print(mTable[index][secondaryIndex])

        newLine;

    print("S table: ")

    for index rom 1 to numberOfMatrices:

     for secondaryIndex from 1 to numberOfMatrices:

        print(sTable[index][secondaryIndex])

    newLine;

**CODE:**

#include <stdio.h>

#include <limits.h>

void findOptimalSolution(int sTable[][100], int index, int secondaryIndex) {

    if (index == secondaryIndex) printf("A%d", index);

    else

    {

        printf("(");

        findOptimalSolution(sTable, index, sTable[index][secondaryIndex]);

        findOptimalSolution(sTable, sTable[index][secondaryIndex] + 1, secondaryIndex);

        printf(")");

    }

}

void matrixChainMultiplication(int dimension[], int numberOfMatrices, int mTable[][100], int sTable[][100])

{

    int index, secondaryIndex, tertiaryIndex, element, temporaryValue;

    for (index = 1; index <= numberOfMatrices; index++) mTable[index][index] = 0;

    for (element = 2; element <= numberOfMatrices; element++)

    {

        for (index = 1; index <= numberOfMatrices - element + 1; index++)

        {

            secondaryIndex = index + element - 1;

            mTable[index][secondaryIndex] = INT\_MAX;

            for (tertiaryIndex = index; tertiaryIndex <= secondaryIndex - 1; tertiaryIndex++)

            {

                temporaryValue = mTable[index][tertiaryIndex] + mTable[tertiaryIndex + 1][secondaryIndex] + dimension[index - 1] \* dimension[tertiaryIndex] \* dimension[secondaryIndex];

                if (temporaryValue < mTable[index][secondaryIndex])

                {

                    mTable[index][secondaryIndex] = temporaryValue;

                    sTable[index][secondaryIndex] = tertiaryIndex;

                }

            }

        }

    }

}

int main()

{

    printf("Name: Afraaz Hussain\nAdmission number: 20BDS0374\n\n\n");

    int numberOfMatrices, index, secondaryIndex;

    printf("Enter the number of matrices: ");

    scanf("%d", &numberOfMatrices);

    int dimension[numberOfMatrices + 1], mTable[100][100], sTable[100][100];

    printf("Enter dimensions of matrices: ");

    for (index = 0; index <= numberOfMatrices; index++) {

        scanf("%d", &dimension[index]);

    }

    matrixChainMultiplication(dimension, numberOfMatrices, mTable, sTable);

    printf("\n\nMinimum number of scalar multiplications: %d\n", mTable[1][numberOfMatrices]);

    printf("Optimal parenthesization: ");

    findOptimalSolution(sTable, 1, numberOfMatrices);

    printf("\n\n\n");

    printf("The M-table is given as follows:\n\n");

    for (index = 1; index <= numberOfMatrices; index++) {

        for (secondaryIndex = 1; secondaryIndex <= numberOfMatrices; secondaryIndex++) {

            if (index > secondaryIndex) printf("       .");

            else printf("%8d", mTable[index][secondaryIndex]);

        }

        printf("\n\n");

    }

    printf("\nThe S-table is give as follows:\n\n");

    for (index = 1; index <= numberOfMatrices; index++) {

        for (secondaryIndex = 1; secondaryIndex <= numberOfMatrices; secondaryIndex++) {

            if (index >= secondaryIndex) printf("    .");

            else printf("%5d", sTable[index][secondaryIndex]);

        }

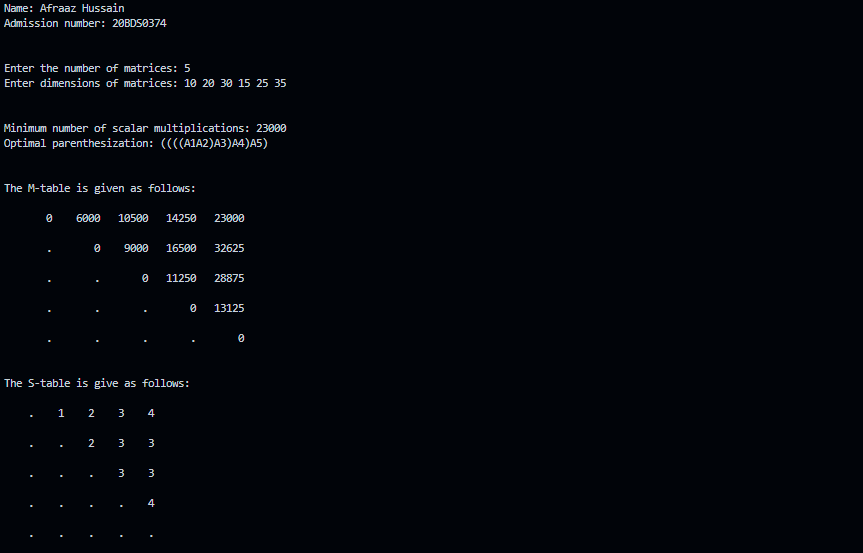
        printf("\n\n");

    }

    return 0;

}

**OUTPUT:**



**QUESTION:**

Implement N-Queens problem using backtracking technique.

**PSEUDOCODE:**

function printSolution(board, numberOfQueens):

    for row from 0 to numberOfQueens:

        for column from 0 to numberOfQueens:

            print(board[row][column])

        print(newLine)

function underAttack(board, row, column):

    for index from 0 to row:

        if any of:

            board[index] == column or

            board[index] - index == column - row or

            board[index] + index == column + row

            then: return true

        return false

function nQueens(board, row, numberOfQueens):

    if row is numberOfQueens: return true

    for column from 0 to numberOfQueens:

        if not underAttack(board, row, column):

            board[row] = column

            if nQueens(board, row + 1, numberOfQueens): return true

            board[row] = -1

    return false

function main():

    numberOfQueens = input()

    board[numberOfQueens][numberOfQueens] = -1

    if nQueens(board, 0, numberOfQueens) printSolution(board, numberOfQueens)

    else print("No solution")

    return

**CODE:**

#include <stdio.h>

#include <stdbool.h>

void printSolution(int board[], int numberOfQueens)

{

    for (int index = 0; index < numberOfQueens; index++)

    {

        for (int secondaryIndex = 0; secondaryIndex < numberOfQueens; secondaryIndex++)

        {

            if (board[index] == secondaryIndex) printf("Q\t");

            else printf(".\t");

        }

        printf("\n\n\n");

    }

    printf("\n");

}

bool underAttack(int board[], int row, int column)

{

    for (int index = 0; index < row; index++) if (board[index] == column || board[index] - index == column - row || board[index] + index == column + row) return true;

    return false;

}

bool nQueens(int board[], int row, int numberOfQueens)

{

    if (row == numberOfQueens) return true;

    for (int column = 0; column < numberOfQueens; column++)

    {

        if (!underAttack(board, row, column))

        {

            board[row] = column;

            if (nQueens(board, row + 1, numberOfQueens)) return true;

            board[row] = -1;

        }

    }

    return false;

}

int main()

{

    printf("Name: Afraaz Hussain\nAdmission number: 20BDS0374\n\n\n");

    int numberOfQueens;

    printf("Enter the number of queens: ");

    scanf("%d", &numberOfQueens);

    int board[numberOfQueens];

    for (int index = 0; index < numberOfQueens; index++) board[index] = -1;

    if (nQueens(board, 0, numberOfQueens))

    {

        printf("\n\nHere is one of many solutions for the given number of queens:\n\n");

        printSolution(board, numberOfQueens);

    }

    else printf("\nAn appropriate solution for the given number of queens was not found.\n");

    return 0;

}

**OUTPUT:**



**QUESTION:**

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

**PSEUDOCODE:**

function naiveMethod():

    textSize, text, patternSize, pattern = input()

    patternCount = 0

    for index from 0 to textSize:

        counter = 0

        for secondaryIndex from 0 to patternSize:

            if pattern[secondaryIndex] = text[index + secondaryIndex] counter++

            else break

        if counter = patternSize:

            print index of pattern

            patternCount++

    if patternCount = 0 print("No pattern was found!")

    else print the number of patterns found

function main():

    naiveMethod()

**CODE:**

#include <stdio.h>

void naiveMethod()

{

    int textSize, patternSize, index, secondaryIndex, counter, patternCount = 0;

    printf("Enter the size of the text: ");

    scanf("%d", &textSize);

    char text[textSize + 1];

    printf("Enter the text: ");

    scanf("%s", &text);

    printf("\nEnter the size of the pattern: ");

    scanf("%d", &patternSize);

    char pattern[patternSize];

    printf("Enter the pattern: ");

    scanf("%s", &pattern);

    for (index = 0; index < textSize; index++)

    {

        counter = 0;

        for (secondaryIndex = 0; secondaryIndex < patternSize; secondaryIndex++)

        {

            if (pattern[secondaryIndex] == text[index + secondaryIndex]) counter++;

            else break;

        }

        if (counter == patternSize)

        {

            printf("\nAn instance of the pattern was found at index %d", index);

            patternCount++;

        }

    }

    if (patternCount == 0) printf("\nThe given pattern was not found in the string provided.");

    else printf("\n\nA total of %d pattern(s) were found!", patternCount);

}

int main()

{

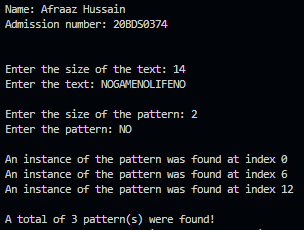
    printf("Name: Afraaz Hussain\nAdmission number: 20BDS0374\n\n\n");

    naiveMethod();

    return 0;

}

**OUTPUT:**



**QUESTION:**

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

**PSEUDOCODE:**

function rabinKarp(textSize, text, patternSize, pattern, hashValue):

    base = input()

    int: spuriousHits = 0

    for index from 0 to patternSize:

        hash = (hash \* base) % hashValue

    for index from 0 to patternSize:

        patternHash = (base \* patternHash + pattern[index]) % hashValue

        textHash = (base \* textHash + text[index]) % hashValue

    for index from 0 to textSize + 1:

        if patternHash = textHash:

            for secondaryIndex from 0 to patternSize:

                if text[index + secondaryIndex] != pattern[secondaryIndex]:

                break

            if secondaryIndex = patternSize print("Match found at {index}")

            else add 1 to spuriousHits

        if index < textSize - patternSize:

            textHash = (base \* (textHash - text[index] \* hash) + text[index + patternSize]) % hashValue

            if textHash < 0:

                textHash = textHash + hashValue

    print(spuriousHits)

function main():

    textSize, text, patternSize, pattern = input()

    hashValue = input()

    rabinKarp(textSize, text, patternSize, pattern, hashValue)

**CODE:**

#include <stdio.h>

#include <string.h>

void rabinKarp(int textSize, char text[], int patternSize, char pattern[], int hashValue)

{

    int index, secondaryIndex, textHash = 0, patternHash = 0, hash = 1, spuriousHits = 0, base = 256;

    printf("Choose a base for the hash function: ");

    scanf("%d", &base);

    printf("\n\n");

    for (index = 0; index < patternSize - 1; index++) hash = (hash \* base) % hashValue;

    for (index = 0; index < patternSize; index++)

    {

        patternHash = (base \* patternHash + pattern[index]) % hashValue;

        textHash = (base \* textHash + text[index]) % hashValue;

    }

    for (index = 0; index <= textSize - patternSize; index++)

    {

        if (patternHash == textHash)

        {

            for (secondaryIndex = 0; secondaryIndex < patternSize; secondaryIndex++) if (text[index + secondaryIndex] != pattern[secondaryIndex]) break;

            if (secondaryIndex == patternSize) printf("Pattern found at index %d \n", index);

            else spuriousHits++;

        }

        if (index < textSize - patternSize)

        {

            textHash = (base \* (textHash - text[index] \* hash) + text[index + patternSize]) % hashValue;

            if (textHash < 0) textHash = textHash + hashValue;

        }

    }

    printf("\nA total of %d spurious hit(s) were encountered!", spuriousHits);

}

int main()

{

    printf("Name: Afraaz Hussain\nAdmission number: 20BDS0374\n\n\n");

    int textSize, patternSize, hashValue = 121;

    printf("Enter the size of the text: ");

    scanf("%d", &textSize);

    char text[textSize + 1];

    printf("Enter the text: ");

    scanf("%s", &text);

    printf("\nEnter the size of the pattern: ");

    scanf("%d", &patternSize);

    char pattern[patternSize];

    printf("Enter the pattern: ");

    scanf("%s", &pattern);

    printf("\nEnter a hash value (preferablly a prime number): ");

    scanf("%d", &hashValue);

    rabinKarp(textSize, text, patternSize, pattern, hashValue);

    return 0;

}

**OUTPUT:**

A screenshot of a computer

Description automatically generated with medium confidence