**QUESTION:**

Write menu-driven program to implement 0/1 and fractional knapsack problem using greedy approach.

**PSEUDOCODE:**

function main():

    numberOfItems, knapsackSize = input()

    choice = input()

    if choice = 'A':

        print(zeroByOneKnapsack(numberOfItems, knapsackSize))

    else:

        print(fractionalKnapSack(numberOfItems, knapsackSize))

function zeroByOneKnapsack():

    items[numberOfItems][3] = input(itemValue, itemWeight, itemNumber)

    float: ratio[numberOfItems]

    for element in items:

        ratio[element] = element[0] / element[1]

    items.sort(descendingly, ratio)

    value = 0

    for element in sortedItems:

        if element[1] <= knapsackSize:

            value += element[0]

            knapsackSize -= element[1]

    return value

function zeroByOneKnapsack():

    items[numberOfItems][3] = input(itemValue, itemWeight, itemNumber)

    float: ratio[numberOfItems]

    for element in items:

        ratio[element] = element[0] / element[1]

    items.sort(descendingly, ratio)

    value = 0

    for element in sortedItems:

        if element[1] <= knapsackSize:

            value += element[0]

            knapsackSize -= element[1]

        else:

            if ratio[element] <= knapsackSize:

                value += ratio[element] \* knapsackSize

    return value

**CODE:**

#include <stdio.h>

void zeroByOneKnapsack(int numberOfItems, int knapsackSize)

{

    int itemsToInclude[numberOfItems], items[numberOfItems][3], temporaryStorage[1][3], value = 0, index, secondaryIndex, counter = 0;

    float temporaryRatio, ratio[numberOfItems];

    printf("\n\n");

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

    {

        printf("Please enter the value and weight of item %d: ", index + 1);

        scanf("%d %d", &items[index][0], &items[index][1]);

        ratio[index] = (float)items[index][0] / (float)items[index][1];

        items[index][2] = index + 1;

    }

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

    {

        for (secondaryIndex = index + 1; secondaryIndex < numberOfItems; ++secondaryIndex)

        {

            if (items[index][2] < items[secondaryIndex][2])

            {

                temporaryStorage[0][0] = items[index][0];

                temporaryStorage[0][1] = items[index][1];

                temporaryStorage[0][2] = items[index][2];

                temporaryRatio = ratio[index];

                items[index][0] = items[secondaryIndex][0];

                items[index][1] = items[secondaryIndex][1];

                items[index][2] = items[secondaryIndex][2];

                ratio[index] = ratio[secondaryIndex];

                items[secondaryIndex][0] = temporaryStorage[0][0];

                items[secondaryIndex][1] = temporaryStorage[0][1];

                items[secondaryIndex][2] = temporaryStorage[0][2];

                ratio[secondaryIndex] = temporaryRatio;

            }

        }

    }

    if (items[0][1] > knapsackSize)

        {

            printf("\n\nNone of the items can be in the knapsack of given size.");

            return;

        }

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

    {

        if (items[index][1] <= knapsackSize)

        {

            itemsToInclude[counter] = items[index][2];

            value += items[index][0];

            knapsackSize -= items[index][1];

            counter++;

        }

        else break;

    }

    printf("\n\nThe following are the items that can be put in the knapsack:");

    for (index = 0; index < counter; index++) printf(" %d", itemsToInclude[index]);

    printf("\nThe total profit was calculated to be: %d", value);

    return;

}

void fractionalKnapsack(int numberOfItems, int knapsackSize)

{

    int itemsToInclude[numberOfItems], items[numberOfItems][3], temporaryStorage[1][3], index, secondaryIndex, counter = 0, additionalItem = 0;

    float temporaryRatio, ratio[numberOfItems], value = 0;

    printf("\n\n");

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

    {

        printf("Please enter the value and weight of item %d: ", index + 1);

        scanf("%d %d", &items[index][0], &items[index][1]);

        ratio[index] = (float)items[index][0] / (float)items[index][1];

        items[index][2] = index + 1;

    }

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

    {

        for (secondaryIndex = index + 1; secondaryIndex < numberOfItems; ++secondaryIndex)

        {

            if (items[index][2] < items[secondaryIndex][2])

            {

                temporaryStorage[0][0] = items[index][0];

                temporaryStorage[0][1] = items[index][1];

                temporaryStorage[0][2] = items[index][2];

                temporaryRatio = ratio[index];

                items[index][0] = items[secondaryIndex][0];

                items[index][1] = items[secondaryIndex][1];

                items[index][2] = items[secondaryIndex][2];

                ratio[index] = ratio[secondaryIndex];

                items[secondaryIndex][0] = temporaryStorage[0][0];

                items[secondaryIndex][1] = temporaryStorage[0][1];

                items[secondaryIndex][2] = temporaryStorage[0][2];

                ratio[secondaryIndex] = temporaryRatio;

            }

        }

    }

    if (items[0][1] > knapsackSize)

        {

            printf("\n\nNone of the items can be in the knapsack of given size.");

            return;

        }

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

    {

        if (items[index][1] <= knapsackSize)

        {

            itemsToInclude[counter] = items[index][2];

            value += items[index][0];

            knapsackSize -= items[index][1];

            counter++;

        }

        else

        {

            if (ratio[index] < knapsackSize)

            {

                value += ratio[index] \* (float)knapsackSize;

                additionalItem = items[index][2];

                knapsackSize = 0;

                break;

            }

        }

    }

    printf("\n\nThe following are the items that can be put in the knapsack:");

    for (index = 0; index < counter; index++) printf(" %d", itemsToInclude[index]);

    if (additionalItem != 0) printf(" %d", additionalItem);

    printf("\nThe total profit was calculated to be: %f", value);

    return;

}

void main()

{

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

    int knapsackSize, numberOfItems;

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

    scanf("%d", &numberOfItems);

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

    scanf("%d", &knapsackSize);

    while (1)

    {

        char choice;

        printf("\nPlease select an option...\n");

        printf("(A) Perform 0|1 knapsack\n");

        printf("(B) Perform fractional knapsack\n");

        printf("\nYour choice: ");

        scanf(" %c", &choice);

        if (choice == 'a' || choice == 'A')

        {

            zeroByOneKnapsack(numberOfItems, knapsackSize);

            break;

        }

        else if (choice == 'b' || choice == 'B')

        {

            fractionalKnapsack(numberOfItems, knapsackSize);

            break;

        }

        else

        {

            printf("\nPlease try again by choosing a valid option...\n\n");

            continue;

        }

    }

}

**OUTPUT:**

* Menu:

A screenshot of a computer

Description automatically generated with medium confidence

* 0/1 knapsack:

A screenshot of a computer

Description automatically generated with medium confidence

* Fractional knapsack:

Text

Description automatically generated

**QUESTION:**

Design and implement Huffman encoding algorithm using greedy approach.

**PSEUDOCODE:**

function main():

    huffmanEncoding()

function huffmanEncoding():

    int: numberOfCharacters, index, secondaryIndex

    numberOfCharacters = input()

    char: characters[numberOfCharacters]

    int: frequency[numberOfCharacters]

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

    {

        printf("Enter character %d and its frequency: ", index + 1)

        scanf(" %c %d", &characters[index], &frequency[index])

    }

    struct MinHeapNode: *\*left, \**right, *\*top*

    priority\_queue<MinHeapNode\*, vector<MinHeapNode\*>, compare> minHeap;

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

        minHeap.push(new MinHeapNode(characters[index], frequency[index]));

    while minHeap.size() is not 1:

        left = minHeap.top()

        minHeap.pop()

        right = minHeap.top()

        minHeap.pop()

        top = new MinHeapNode('$', left->frequency + right->frequency)

        top->left = left

        top->right = right

        minHeap.push(top)

    printCodes(minHeap.top(), "")

function printCodes(struct MinHeapNode: root, str):

    if not root return

    if root -> data not '$' print(root -> data + ": " + str)

    printCodes(root -> left, str = '0')

    printCodes(root -> right, str + '1')

structure compare:

    bool operator()(MinHeapNode\* l, MinHeapNode\* r) return 1 -> frequency > r -> frequency

structure MinHeapNode:

    char: dataunsigned: frequency

    MinHeapNode: *\*left, \**right

    function MinHeapNode(data, frequency):

        left = right = NULL

        self.data = data

        self.frequency = frequency

**CODE:**

#include <bits/stdc++.h>

using namespace std;

struct MinHeapNode

{

    char data;

    unsigned frequency;

    MinHeapNode \*left, \*right;

    MinHeapNode(char data, unsigned frequency)

    {

        left = right = NULL;

        this->data = data;

        this->frequency = frequency;

    }

};

struct compare

{

    bool operator()(MinHeapNode\* l, MinHeapNode\* r)

    {

        return (l->frequency > r->frequency);

    }

};

void printCodes(struct MinHeapNode\* root, string str)

{

    if (!root)

        return;

    if (root->data != '$')

        cout << root->data << ": " << str << "\n";

    printCodes(root->left, str + "0");

    printCodes(root->right, str + "1");

}

void huffmanEncoding()

{

    int numberOfCharacters, index, secondaryIndex;

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

    scanf("%d", &numberOfCharacters);

    char characters[numberOfCharacters];

    int frequency[numberOfCharacters];

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

    {

        printf("Enter character %d and its frequency: ", index + 1);

        scanf(" %c %d", &characters[index], &frequency[index]);

    }

    struct MinHeapNode \*left, \*right, \*top;

    // Create a min heap & inserts all characters of data[]

    priority\_queue<MinHeapNode\*, vector<MinHeapNode\*>, compare> minHeap;

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

        minHeap.push(new MinHeapNode(characters[index], frequency[index]));

    // Iterate while size of heap doesn't become 1

    while (minHeap.size() != 1) {

        left = minHeap.top();

        minHeap.pop();

        right = minHeap.top();

        minHeap.pop();

        top = new MinHeapNode('$', left->frequency + right->frequency);

        top->left = left;

        top->right = right;

        minHeap.push(top);

    }

    printCodes(minHeap.top(), "");

}

int main()

{

    printf("Name: Afraaz Hussain\nAdmission number: 20BDs0374\n\n\n");

    huffmanEncoding();

    return 0;

}

**OUTPUT:**

Text

Description automatically generated

**QUESTION:**

Implement 0/1 knapsack problem using dynamic programming technique.

**PSEUDOCODE:**

function zeroByOneKnapsack(numberOfItems, knapsackSize):

    items[numberOfItems][2]

    ratios[numberOfItems]

    for index from 0 to numberOfItems:

        input(values)

        ratios[index] = items[index][0] / items[index][1]

    for index from 0 to knapsackSize:

        knapsack[0, index] = 0

    for index from 1 to numberOfItems:

        knapsack[index, 0] = 0

        for secondaryIndex from 1 to knapsackSize:

            if items[index][1] <= secondaryIndex:

                if items[index][0] + knapsack[index - 1, secondaryIndex - items[index][1]]:

                    knapsack[index, secondaryIndex] = knapsack[index - 1, secondaryIndex]

                else:

                    knapsack[index, secondaryIndex] = knapsack[index - 1, secondaryIndex]

            else:

                knapsack[index, secondaryIndex] = knapsack[index - 1, secondaryIndex]

    Total profit is knapsack[numberOfItems][knapsackSize]

**CODE:**

#include <stdio.h>

int findMaximum(int numberOne, int numberTwo) { return (numberOne > numberTwo) ? numberOne : numberTwo; }

void zeroByOneKnapsack(int numberOfItems, int knapsackSize)

{

    int itemsToInclude[numberOfItems], items[numberOfItems][3], temporaryStorage[1][3], value = 0, index, secondaryIndex, counter = 0;

    float temporaryRatio, ratio[numberOfItems];

    printf("\n\n");

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

    {

        printf("Please enter the value and weight of item %d: ", index + 1);

        scanf("%d %d", &items[index][0], &items[index][1]);

        ratio[index] = (float)items[index][0] / (float)items[index][1];

        items[index][2] = index + 1;

    }

    int knapsack[numberOfItems + 1][knapsackSize + 1];

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

    {

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

        {

            if (index == 0 || secondaryIndex == 0) knapsack[index][secondaryIndex] = 0;

            else if (items[index - 1][1] <= secondaryIndex) knapsack[index][secondaryIndex] = findMaximum(items[index - 1][0] + knapsack[index - 1][secondaryIndex - items[index - 1][1]],  knapsack[index - 1][secondaryIndex]);

            else knapsack[index][secondaryIndex] = knapsack[index - 1][secondaryIndex];

        }

    }

    printf("\nThe total profit was calculated to be: %d", knapsack[numberOfItems][knapsackSize]);

    return;

}

void main()

{

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

    int knapsackSize, numberOfItems;

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

    scanf("%d", &numberOfItems);

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

    scanf("%d", &knapsackSize);

    zeroByOneKnapsack(numberOfItems, knapsackSize);

}

**OUTPUT:**

A screenshot of a computer

Description automatically generated with medium confidence

**QUESTION:**

Implement LCS problem using dynamic programming technique.

**PSEUDOCODE:**

sequenceOne = input()

sequenceTwo = input()

lCSTable[length(sequenceOne)][length(sequenceTwo)]

sequenceOne.label = sequenceOne

sequenceTwo.label = sequenceTwo

lCSTable[0][] = 0

lCSTable[][0] = 0

Start from lCSTable[1][1]

Compare sequenceOne[row] and sequenceTwo[column]

    if sequenceOne[row] = sequenceTwo[column]

        lCSTable[row][column] = 1 + lCSTable[row - 1, column - 1]

        Point an arrow to lCSTable[row][column]

    else

        lCSTable[row][column] = max(lCSTable[row - 1][column], lCSTable[row][column - 1])

        Point an arrow to max(lCSTable[row - 1][column], lCSTable[row][column - 1])

**CODE:**

#include <stdio.h>

#include <string.h>

void longestCommonSubsequence()

{

    int lCSTable[20][20], index, secondaryIndex;

    char stringOne[20], stringTwo[20], crossMatrix[20][20];

    printf("Enter the first string: ");

    scanf("%s", stringOne);

    printf("Enter the second string: ");

    scanf("%s", stringTwo);

    int lengthOne = strlen(stringOne), lengthTwo = strlen(stringTwo);

    for (index = 0; index <= lengthOne; index++) lCSTable[index][0] = 0;

    for (index = 0; index <= lengthTwo; index++) lCSTable[0][index] = 0;

    for (index = 1; index <= lengthOne; index++)

    {

        for (secondaryIndex = 1; secondaryIndex <= lengthTwo; secondaryIndex++)

        {

            if (stringOne[index - 1] == stringTwo[secondaryIndex - 1]) lCSTable[index][secondaryIndex] = lCSTable[index - 1][secondaryIndex - 1] + 1;

            else if (lCSTable[index - 1][secondaryIndex] >= lCSTable[index][secondaryIndex - 1]) lCSTable[index][secondaryIndex] = lCSTable[index - 1][secondaryIndex];

            else lCSTable[index][secondaryIndex] = lCSTable[index][secondaryIndex - 1];

        }

    }

    int element = lCSTable[lengthOne][lengthTwo], counter = 0;

    char lCSAlgorithm[element + 1];

    lCSAlgorithm[index] = '\0';

    index = lengthOne;

    secondaryIndex = lengthTwo;

    while (index > 0 && secondaryIndex > 0)

    {

        if (stringOne[index - 1] == stringTwo[secondaryIndex - 1])

        {

            lCSAlgorithm[element - 1] = stringOne[index - 1];

            index--;

            secondaryIndex--;

            element--;

            counter++;

        }

        else if (lCSTable[index - 1][secondaryIndex] > lCSTable[index][secondaryIndex - 1]) index--;

        else secondaryIndex--;

    }

    lCSAlgorithm[counter] = '\0';

    printf("\n\nString one: %s \nString two: %s\n\n", stringOne, stringTwo);

    printf("Longest Common Subsequence: %s", lCSAlgorithm);

}

int main()

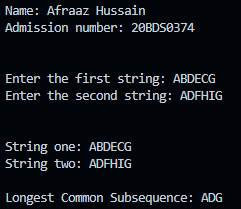
{

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

    longestCommonSubsequence();

}

**OUTPUT:**



**QUESTION:**

Design an algorithm and implement travelling salesman problem using dynamic programming approach.

**PSEUDOCODE:**

globalVariables:

    int: costMatrix[10][10], visitedNode[10], numberOfCities, cost = 0

function main():

    travellingSalemanProblem()

    print("The path is: ", minimumPath(startingCity))

    print("The least cost was calculated to be: ", cost)

function minimumCost(city):

    int: index, cityID

    visitedNode[city] = 1

    print(city + 1 + " -> ")

    cityID = least(city)

    if cityID = 999:

        cityID, cost = 0, cost + costMatrix[city][cityID]

        return

    minimumCost(cityID)

function least(number):

    int: index, newCity = 999, minimumValue = 999, kMinimumValue

    for index in range(numberOfCities):

        if((costMatrix[number][index] != 0) && (visitedNode[index] == 0)):

            if(costMatrix[number][index] + costMatrix[index][number] < minimumValue):

                minimumValue = costMatrix[index][0] + costMatrix[number][index]

                kMinimumValue = costMatrix[number][index]

                newCity = index

        if minimumValue is not equal to 999:

            cost += kMinimumValue

        retun newCity

function travellingSalemanProblem():

    int: index, secondaryIndex

    numberOfCities = input()

    costMatrix = input()

    visitedNode[index] = 0

**CODE:**

#include<stdio.h>

int costMatrix[10][10], visitedNode[10], numberOfCities, cost = 0;

void travellingSalesmanProblem()

{

    int index, secondryIndex;

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

    scanf("%d", &numberOfCities);

    printf("\nEnter the cost matrix...\n");

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

    {

        printf("Enter the cost from city %d to other cities: ", index + 1);

        for( secondryIndex = 0; secondryIndex < numberOfCities; secondryIndex++) scanf(" %d", &costMatrix[index][secondryIndex]);

        visitedNode[index] = 0;

    }

}

void minimumCost(int city)

{

    int index, cityID;

    visitedNode[city] = 1;

    printf("%d -> ", city + 1);

    cityID = least(city);

    if(cityID == 999)

    {

        cityID = 0;

        cost += costMatrix[city][cityID];

        return;

    }

    minimumCost(cityID);

}

int least(int number)

{

    int index, newCity = 999, minimumValue = 999, kMinimumValue;

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

    {

        if((costMatrix[number][index] != 0) && (visitedNode[index] == 0))

        if(costMatrix[number][index] + costMatrix[index][number] < minimumValue)

        {

            minimumValue = costMatrix[index][0] + costMatrix[number][index];

            kMinimumValue = costMatrix[number][index];

            newCity = index;

        }

    }

    if(minimumValue != 999) cost += kMinimumValue;

    return newCity;

}

int main()

{

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

    travellingSalesmanProblem();

    printf("\n\nThe Path is:\n");

    minimumCost(0);

    printf("\nThe minimum cost to travel all the cities was calculated to be: %d", cost);

    return 0;

}

**OUTPUT:**

