**NAME : Arshdeep Palial**

**ROLLNO: 102216099**

**CS14**

1. **0/1 knapsac**

#include <bits/stdc++.h>

using namespace std;

int knapsack(vector<int>& weight, vector<int>& value, int index, int capacity, vector<vector<int>>& dp) {

if (index < 0 || capacity == 0) {

return 0;

}

if (dp[index][capacity] != -1) {

return dp[index][capacity]; }

int include = 0;

if (weight[index] <= capacity) {

include = value[index] + knapsack(weight, value, index - 1, capacity - weight[index], dp);

}

int exclude = knapsack(weight, value, index - 1, capacity, dp);

dp[index][capacity] = max(include, exclude);

return dp[index][capacity];

}

int main() {

vector<int> weight = {/\* your weights here \*/};

vector<int> value = { /\* your values here \*/ };

int capacity = /\* your capacity here \*/;

int n = weight.size();

vector<vector<int>> dp(n, vector<int>(capacity + 1, -1));

int result = knapsack(weight, value, n - 1, capacity, dp);

cout << "Maximum value: " << result << endl;

return 0;

}

**2.OPTIMAL BINARY SEARCH TREE**

#include <bits/stdc++.h>

using namespace std;

#define MAX 1000

int cost[MAX][MAX];

int Sum(int freq[], int i, int j) {

int s = 0;

for (int k = i; k <= j; k++)

s += freq[k];

return s;

}

int optCost\_memoized(int freq[], int i, int j) {

//base case

if (cost[i][j])

return cost[i][j];

int fsum = Sum(freq, i, j);

int Min = INT\_MAX;

for (int r = i; r <= j; r++) {

int c = optCost\_memoized(freq, i, r - 1) + optCost\_memoized(freq, r + 1, j) + fsum;

if (c < Min) {

Min = c;

cost[i][j] = c;

}

}

return cost[i][j];

}

int optimalSearchTree(int keys[], int freq[], int n) {

return optCost\_memoized(freq, 0, n - 1);

}

int main() {

// catch (keys should be sorted)

int keys[] = {10, 12, 20};

int freq[] = {34, 8, 50};

int n = sizeof(keys) / sizeof(keys[0]);

memset(cost, 0, sizeof(cost));

for (int i = 0; i < n; i++)

cost[i][i] = freq[i];

cout << "Cost of Optimal BST is " << optimalSearchTree(keys, freq, n) << endl;

return 0;

}

**3.LONGEST COMMON SUBSEQUENCE**

**#include<iostream>**

**using namespace std;**

**int LCS(int x, int y, string s1, string s2) {**

**// Base case**

**if ( x == 0 || y == 0 )**

**return 0;**

**if (s1[x-1] == s2[y-1])**

**return 1 + LCS(x - 1, y - 1, s1, s2);**

**else**

**return max(LCS(x - 1, y, s1, s2), LCS(x, y - 1, s1, s2));**

**}**

**int main() {**

**string s1 = {'s','t','o','n','e'};**

**string s2 = {'l','o','n','g','e','s','t'};**

**int m = s1.length();**

**int n = s2.length();**

**cout << LCS(m, n, s1, s2);**

**return 0;**

**}**

**4.MATRIX CHAIN MULTIPLICATION**

**#include <bits/stdc++.h>**

**using namespace std;**

**int dp[100][100];**

**int matrixChainMemoised(int\* p, int i, int j)**

**{**

**if (i == j)**

**{**

**return 0;**

**}**

**if (dp[i][j] != -1)**

**{**

**return dp[i][j];**

**}**

**dp[i][j] = INT\_MAX;**

**for (int k = i; k < j; k++)**

**{**

**dp[i][j] = min(**

**dp[i][j], matrixChainMemoised(p, i, k)**

**+ matrixChainMemoised(p, k + 1, j)**

**+ p[i - 1] \* p[k] \* p[j]);**

**}**

**return dp[i][j];**

**}**

**// Driver Code**

**int main()**

**{**

**int arr[] = { 1, 2, 3, 4 };**

**int n = sizeof(arr) / sizeof(arr[0]);**

**memset(dp, -1, sizeof dp);**

**cout << "Minimum number of multiplications is "**

**<< matrixChainMemoised(arr, 1, n-1);**

**}**