

Linear_DP.cpp

```

1  /**
2   *   Author: devesh95
3   *
4   *   Topic: Linear Dynamic Programming (1D DP)
5   *
6   *   Description:
7   *   This file contains a collection of example problems solved using
8   *   linear dynamic programming. The idea behind 1D DP is to build an
9   *   array (or vector) 'dp' where each entry dp[i] represents the solution
10  *   for a subproblem involving the first i elements or a state 'i'.
11  *
12  *   Use Cases Covered:
13  *       1. Fibonacci Sequence
14  *       2. Climbing Stairs Problem
15  *       3. Minimum Coin Change (Unbounded Knapsack)
16  *       4. Maximum Subarray Sum (Kadane's Algorithm as DP)
17  *       5. Longest Increasing Subsequence (LIS) - O(n^2) approach
18  *       6. Rod Cutting Problem
19  *
20  *   Each function is self-contained with input, processing, and output.
21  *   The comments within each function provide step-by-step explanations
22  *   of the DP approach used.
23  *
24  *   Note: All problems use a 1D DP formulation where each state depends
25  *         on one or more previous states.
26  *
27  *   Compile with:
28  *       g++ -std=c++17 -O2 -Wall Linear_Dynamic_Programming.cpp -o ldp
29  *
30  *   Run with:
31  *       ./ldp
32  */
33
34  #include <bits/stdc++.h>
35  using namespace std;
36
37  #define int long long
38  #define pb push_back
39  #define F first
40  #define S second
41
42  //////////////////////////////////////
43  // 1. Fibonacci Sequence using DP
44  //////////////////////////////////////
45  /*
46   Problem Statement:
47   Given n, compute the nth Fibonacci number.
48   Fibonacci numbers are defined as:
49       F(0) = 0, F(1) = 1
50       F(n) = F(n-1) + F(n-2) for n >= 2
51

```

```

52     DP Approach:
53     We use a dp array where dp[i] stores F(i).
54     Base cases: dp[0] = 0, dp[1] = 1.
55     For each i from 2 to n, we compute dp[i] = dp[i-1] + dp[i-2].
56 */
57 void solve_fibonacci() {
58     cout << "\n----- Fibonacci Sequence using DP ----- \n";
59     int n;
60     cout << "Enter n (0-indexed): ";
61     cin >> n;
62
63     if(n < 0) {
64         cout << "Invalid input. n must be non-negative.\n";
65         return;
66     }
67     // Create a DP array of size n+1
68     vector<int> dp(n+1, 0);
69     dp[0] = 0;
70     if(n > 0)
71         dp[1] = 1;
72
73     for (int i = 2; i <= n; i++) {
74         dp[i] = dp[i-1] + dp[i-2];
75     }
76
77     cout << "Fibonacci number F(" << n << ") = " << dp[n] << "\n";
78 }
79
80 ///////////////////////////////////////////////////////////////////
81 // 2. Climbing Stairs Problem using DP
82 ///////////////////////////////////////////////////////////////////
83 /*
84     Problem Statement:
85     You are climbing a staircase. It takes n steps to reach the top.
86     Each time you can climb either 1 or 2 steps.
87     In how many distinct ways can you climb to the top?
88
89     DP Approach:
90     Let dp[i] be the number of ways to reach step i.
91     Base cases: dp[0] = 1 (one way to stand at the base), dp[1] = 1.
92     For i >= 2: dp[i] = dp[i-1] + dp[i-2] because you can come
93     to i from i-1 (1 step) or i-2 (2 steps).
94 */
95 void solve_climbing_stairs() {
96     cout << "\n----- Climbing Stairs using DP ----- \n";
97     int n;
98     cout << "Enter the number of stairs: ";
99     cin >> n;
100
101     vector<int> dp(n+1, 0);
102     dp[0] = 1; // 1 way to be at the ground level
103     if(n >= 1)
104         dp[1] = 1; // Only one step possible
105

```

```

106     for (int i = 2; i <= n; i++) {
107         dp[i] = dp[i-1] + dp[i-2];
108     }
109
110     cout << "Total distinct ways to climb " << n << " stairs: " << dp[n] << "\n";
111 }
112
113 ///////////////////////////////////////////////////
114 // 3. Minimum Coin Change (Unbounded Knapsack) using DP
115 ///////////////////////////////////////////////////
116 /*
117     Problem Statement:
118     Given coins of different denominations and a total amount,
119     find the minimum number of coins that you need to make up that amount.
120     If that amount cannot be made up by any combination of the coins, return -1.
121
122     DP Approach:
123     Let dp[i] be the minimum number of coins needed for amount i.
124     Base case: dp[0] = 0.
125     For each amount i from 1 to total amount:
126         For each coin value c:
127             if i-c >= 0, dp[i] = min(dp[i], dp[i-c] + 1)
128 */
129 void solve_coin_change() {
130     cout << "\n----- Minimum Coin Change using DP ----- \n";
131     int n;
132     cout << "Enter the number of coin denominations: ";
133     cin >> n;
134     vector<int> coins(n);
135     cout << "Enter the coin denominations: ";
136     for (int i = 0; i < n; i++) {
137         cin >> coins[i];
138     }
139     int amount;
140     cout << "Enter the total amount: ";
141     cin >> amount;
142
143     const int INF = 1e9;
144     vector<int> dp(amount + 1, INF);
145     dp[0] = 0;
146
147     for (int i = 1; i <= amount; i++) {
148         for (int coin : coins) {
149             if (i - coin >= 0) {
150                 dp[i] = min(dp[i], dp[i-coin] + 1);
151             }
152         }
153     }
154     if(dp[amount] == INF)
155         cout << "It is not possible to form the amount with given coins.\n";
156     else
157         cout << "Minimum coins required: " << dp[amount] << "\n";
158 }
159

```

```

160 //////////////////////////////////////////////////
161 // 4. Maximum Subarray Sum (Kadane's Algorithm as DP)
162 //////////////////////////////////////////////////
163 /*
164     Problem Statement:
165     Given an array of integers, find the contiguous subarray (containing at least one
166     number)
167     which has the largest sum.
168
169     DP Approach:
170     Let dp[i] be the maximum subarray sum ending at index i.
171     Then, dp[i] = max(a[i], dp[i-1] + a[i]).
172     The answer is the maximum value in dp[].
173 */
174 void solve_maximum_subarray() {
175     cout << "\n----- Maximum Subarray Sum using DP (Kadane's Algorithm) ----- \n";
176     int n;
177     cout << "Enter the number of elements in the array: ";
178     cin >> n;
179     vector<int> arr(n);
180     cout << "Enter the elements of the array:\n";
181     for (int i = 0; i < n; i++) {
182         cin >> arr[i];
183     }
184
185     vector<int> dp(n);
186     dp[0] = arr[0];
187     int maxSum = dp[0];
188     for (int i = 1; i < n; i++) {
189         dp[i] = max(arr[i], dp[i-1] + arr[i]);
190         maxSum = max(maxSum, dp[i]);
191     }
192     cout << "Maximum subarray sum is: " << maxSum << "\n";
193 }
194 //////////////////////////////////////////////////
195 // 5. Longest Increasing Subsequence (LIS) using DP (O(n^2))
196 //////////////////////////////////////////////////
197 /*
198     Problem Statement:
199     Given an array of integers, find the length of the longest strictly increasing
200     subsequence.
201
202     DP Approach:
203     Let dp[i] be the length of the longest increasing subsequence ending at i.
204     For each i, iterate j from 0 to i-1. If arr[j] < arr[i], then update dp[i] = max(dp[i],
205     dp[j] + 1).
206     The answer is the maximum value in dp[].
207 */
208 void solve_LIS() {
209     cout << "\n----- Longest Increasing Subsequence (LIS) using DP ----- \n";
210     int n;
211     cout << "Enter the number of elements in the array: ";
212     cin >> n;

```

```

211     vector<int> arr(n);
212     cout << "Enter the elements of the array:\n";
213     for (int i = 0; i < n; i++) {
214         cin >> arr[i];
215     }
216
217     vector<int> dp(n, 1);
218     int ans = 1;
219     for (int i = 0; i < n; i++) {
220         for (int j = 0; j < i; j++) {
221             if(arr[j] < arr[i]) {
222                 dp[i] = max(dp[i], dp[j] + 1);
223             }
224         }
225         ans = max(ans, dp[i]);
226     }
227     cout << "Length of Longest Increasing Subsequence is: " << ans << "\n";
228 }
229
230 ///////////////////////////////////////////////////
231 // 6. Rod Cutting Problem using DP
232 ///////////////////////////////////////////////////
233 /*
234     Problem Statement:
235     Given a rod of length n inches and an array of prices that contains prices of all
pieces of size 1 to n,
236     determine the maximum revenue obtainable by cutting up the rod and selling the pieces.
237
238     DP Approach:
239     Let dp[i] be the maximum revenue obtainable for a rod of length i.
240     For each length i from 1 to n:
241         For each possible cut j from 1 to i:
242             dp[i] = max(dp[i], price[j-1] + dp[i - j])
243 */
244 void solve_rod_cutting() {
245     cout << "\n----- Rod Cutting Problem using DP ----- \n";
246     int n;
247     cout << "Enter the rod length: ";
248     cin >> n;
249     vector<int> price(n);
250     cout << "Enter the prices for each rod length from 1 to " << n << ":\n";
251     for (int i = 0; i < n; i++) {
252         cin >> price[i];
253     }
254
255     vector<int> dp(n+1, 0);
256     // dp[0] = 0 is already set by default.
257     for (int i = 1; i <= n; i++) {
258         for (int j = 1; j <= i; j++) {
259             dp[i] = max(dp[i], price[j-1] + dp[i-j]);
260         }
261     }
262     cout << "Maximum revenue obtainable: " << dp[n] << "\n";
263 }

```

```
264
265 ///////////////////////////////////////////////////////////////////
266 // Main function with a menu to choose DP problems
267 ///////////////////////////////////////////////////////////////////
268 int32_t main() {
269     ios_base::sync_with_stdio(0);
270     cin.tie(0);
271     cout.tie(0);
272
273     cout << "=====\n";
274     cout << "  Linear Dynamic Programming (1D DP) Notes\n";
275     cout << "=====\n";
276     cout << "Select a problem to solve:\n";
277     cout << "1. Fibonacci Sequence\n";
278     cout << "2. Climbing Stairs\n";
279     cout << "3. Minimum Coin Change\n";
280     cout << "4. Maximum Subarray Sum\n";
281     cout << "5. Longest Increasing Subsequence (LIS)\n";
282     cout << "6. Rod Cutting Problem\n";
283     cout << "7. Run All Examples\n";
284     cout << "Enter your choice: ";
285
286     int choice;
287     cin >> choice;
288     cout << "\n";
289
290     switch(choice) {
291         case 1:
292             solve_fibonacci();
293             break;
294         case 2:
295             solve_climbing_stairs();
296             break;
297         case 3:
298             solve_coin_change();
299             break;
300         case 4:
301             solve_maximum_subarray();
302             break;
303         case 5:
304             solve_LIS();
305             break;
306         case 6:
307             solve_rod_cutting();
308             break;
309         case 7:
310             solve_fibonacci();
311             solve_climbing_stairs();
312             solve_coin_change();
313             solve_maximum_subarray();
314             solve_LIS();
315             solve_rod_cutting();
316             break;
317         default:
```

```
318         cout << "Invalid choice. Exiting...\n";
319     }
320
321     return 0;
322 }
323
```