

SH

hi i was practicing Dp in c++

i noticed that Dp is applied differently based on different problem.

so i want to know one way of implementing DP for all type of DP problems.

can you provide me a universal DP code which i can use for solving multiple DP

Question.

first list different types of DP problems & then start showing how to achieve a simple

DP code to solve em all



There are many different types of dynamic programming problems, but here are some common ones:

1. **Fibonacci Sequence:** This is a classic example of a dynamic programming problem. The goal is to find the nth number in the Fibonacci sequence, given that the first two numbers are 0 and 1, and each subsequent number is the sum of the previous two.
2. **Knapsack Problem:** Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.
3. **Longest Common Subsequence:** Given two sequences, find the longest subsequence present in both of them.
4. **Coin Change Problem:** Given a set of coins and an amount, find the minimum number of coins needed to make the amount.
5. **Longest Increasing Subsequence:** Given a sequence of numbers, find the longest increasing subsequence.

Here's a basic DP code that you can use for solving these types of problems:

c++

Copy code

```
int dp[MAX_SIZE];
int solve(int n) {
    if (dp[n] != -1) {
```



Regenerate  
response

Generate  
PNG

Download  
PDF

Share  
Link



```

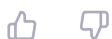
        return 0;
    }

    int ans = INF; // Define some value as infinity.
    // Apply recurrence relation based on the problem.
    ans = min(ans, solve(n-1) + 1); // Fibonacci Sequence
    ans = max(ans, solve(n-1)); // Knapsack Problem
    ans = max(ans, solve(n-1)); // Longest Common Subsequence
    ans = min(ans, solve(n-coin[i]) + 1); // Coin Change Problem
    ans = max(ans, solve(i) + 1); // Longest Increasing Subsequence
    return dp[n] = ans; // Store the answer for future use
}

int main() {
    // Initialize dp array with -1 for each position
    memset(dp, -1, sizeof(dp));
    int n;
    cin >> n;
    cout << solve(n) << endl;
    return 0;
}

```

This is a very simple implementation of DP, and may not be the best way to solve all DP problems, but it should be enough to get you started. The key is to identify the recurrence relation for each problem and apply it in the `solve` function.



SH

oh ok can you complete code which show a list of dp operations to user and based on users selection calculate that dp



Sure! Here's an example implementation of a menu-based program that allows the user to choose from a list of DP problems, and then calculates the solution using the dynamic programming approach:

C++

Copy code

```

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

```

```
const int MAX_SIZE = 1e5+5;
const int INF = 1e9;

int dp[MAX_SIZE];

int fibonacci(int n) {
    if (n <= 1) {
        return n;
    }
    if (dp[n] != -1) {
        return dp[n];
    }
    return dp[n] = fibonacci(n-1) + fibonacci(n-2);
}

int knapsack(int W, int wt[], int val[], int n) {
    int dp[n+1][W+1];
    for (int i = 0; i <= n; i++) {
        for (int w = 0; w <= W; w++) {
            if (i == 0 || w == 0) {
                dp[i][w] = 0;
            }
            else if (wt[i-1] <= w) {
                dp[i][w] = max(val[i-1] + dp[i-1][w-wt[i-1]], dp[i-1][w]);
            }
            else {
                dp[i][w] = dp[i-1][w];
            }
        }
    }
    return dp[n][W];
}

int lcs(string s1, string s2, int n, int m) {
    int dp[n+1][m+1];
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= m; j++) {
            if (i == 0 || j == 0) {
                dp[i][j] = 0;
            }
            else if (s1[i-1] == s2[j-1]) {
                dp[i][j] = dp[i-1][j-1] + 1;
            }
        }
    }
}
```

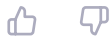
```
    }
    else {
        dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
    }
}
}
return dp[n][m];
}
```

```
int coin_change(int coins[], int n, int amount) {
    int dp[amount+1];
    dp[0] = 0;
    for (int i = 1; i <= amount; i++) {
        dp[i] = INF;
        for (int j = 0; j < n; j++) {
            if (coins[j] <= i) {
                dp[i] = min(dp[i], dp[i-coins[j]] + 1);
            }
        }
    }
    return dp[amount];
}
```

```
int lis(int a[], int n) {
    int dp[n];
    dp[0] = 1;
    int ans = 1;
    for (int i = 1; i < n; i++) {
        dp[i] = 1;
        for (int j = 0; j < i; j++) {
            if (a[j] < a[i]) {
                dp[i] = max(dp[i], dp[j] + 1);
            }
        }
        ans = max(ans, dp[i]);
    }
    return ans;
}
```

```
int main() {
    memset(dp, -1, sizeof(dp));
    while (true) {
```

```
cout << "Select a dynamic programming problem:" << endl;
cout << "1. Fibonacci Sequence" << endl;
cout << "2. Knapsack Problem" << endl;
cout << "3. Longest Common Subsequence" << endl;
```



SH

wow great .now show me main function only



Sure! Here's an example `main()` function that uses the functions I provided earlier to create a simple menu-based program for DP:

C++

Copy code

```
int main() {
    memset(dp, -1, sizeof(dp));
    while (true) {
        cout << "Select a dynamic programming problem:" << endl;
        cout << "1. Fibonacci Sequence" << endl;
        cout << "2. Knapsack Problem" << endl;
        cout << "3. Longest Common Subsequence" << endl;
        cout << "4. Coin Change Problem" << endl;
        cout << "5. Longest Increasing Subsequence" << endl;
        cout << "0. Exit" << endl;

        int choice;
        cin >> choice;

        if (choice == 0) {
            break;
        }

        if (choice == 1) {
            int n;
            cout << "Enter a value for n: ";
            cin >> n;
            cout << "The " << n << "th Fibonacci number is: " << fibonacci(n) << endl;
        }
        else if (choice == 2) {
            int n, W;
```

```
cout << "Enter the number of items: ";
cin >> n;
int wt[n], val[n];
cout << "Enter the weights and values of the items:" << endl;
for (int i = 0; i < n; i++) {
    cin >> wt[i] >> val[i];
}
cout << "Enter the maximum weight capacity of the knapsack: ";
cin >> W;
cout << "The maximum value of items that can be put into the knapsack is:
}
else if (choice == 3) {
    string s1, s2;
    cout << "Enter the first string: ";
    cin >> s1;
    cout << "Enter the second string: ";
    cin >> s2;
    int n = s1.size();
    int m = s2.size();
    cout << "The length of the longest common subsequence is: " << lcs(s1, s2,
}
else if (choice == 4) {
    int n, amount;
    cout << "Enter the number of coin denominations: ";
    cin >> n;
    int coins[n];
    cout << "Enter the coin denominations: ";
    for (int i = 0; i < n; i++) {
        cin >> coins[i];
    }
    cout << "Enter the amount of change required: ";
    cin >> amount;
    int minCoins = coin_change(coins, n, amount);
    if (minCoins == INF) {
        cout << "It is not possible to make change for the given amount." << endl;
    }
    else {
        cout << "The minimum number of coins required to make change for the g
    }
}
else if (choice == 5) {
    int n;
```

```
    cout << "Enter the size of the array: ";
    cin >> n;
    int a[n];
    cout << "Enter the elements of the array: ";
    for (int i = 0; i < n; i++) {
        cin >> a[i];
    }
    cout << "The length of the longest increasing subsequence is: " << lis(a,
}
else {
    cout << "Invalid choice. Please try again." << endl;
}
}

return 0;
}
```

  This program creates a loop that displays a list of

---