

SH

hi can you teach me how to solve coin base change problem in c++



Sure, I can help you with that! The coin change problem is a classic dynamic programming problem in which you need to find the minimum number of coins required to make a given amount of money. Here is how you can solve the coin change problem in C++:

1. Define an array to store the minimum number of coins required for each value from 0 to the target amount. Let's call this array dp.
2. Initialize dp[0] to 0 and all other elements to a very large number.
3. Loop through all the coins and for each coin, update the dp array as follows:

```
for(int j=coinValue;j<=targetAmount;j++){  
    dp[j]=min(dp[j],dp[j-coinValue]+1);  
}
```

Here, coinValue is the value of the current coin and j is the index of the dp array.
4. Once the loop is finished, the minimum number of coins required to make the target amount will be stored in dp[targetAmount].
5. If dp[targetAmount] is still a very large number, it means it is not possible to make the target amount with the given coins.

Here is the complete C++ code for solving the coin change problem:

css

Copy code

```
#include<bits/stdc++.h>  
using namespace std;  
  
int main() {  
    int n,targetAmount;  
    cin>>n>>targetAmount;  
    int coins[n];  
    for(int i=0;i<n;i++) cin>>coins[i];
```

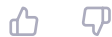
Regenerate
responseGenerate
PNGDownload
PDFShare
Link

```
for(int i=0;i<n;i++){
    for(int j=coins[i];j<=targetAmount;j++){
        dp[j]=min(dp[j],dp[j-coins[i]]+1);
    }
}

if(dp[targetAmount]==INT_MAX){
    cout<<"Not possible to make the target amount with given coins";
}
else{
    cout<<dp[targetAmount]<<" coins required to make the target amount";
}

return 0;
}
```

Hope this helps! Let me know if you have any questions.



SH

can you explain me line line with live input in details



Sure! Here's a detailed explanation of the code with an example input:

c

Copy code

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

int main() {
    int n,targetAmount;
    cin>>n>>targetAmount;
    int coins[n];
    for(int i=0;i<n;i++) cin>>coins[i];
```

This section of the code is just taking the input from the user. The first line is defining two integer variables n and targetAmount. The second line is taking the input for n and targetAmount.

Let's say the input is: $n=3$, $\text{targetAmount}=7$.

The next two lines define an integer array called `coins`, with n elements. The for loop then takes input for each element of the array `coins`.

Let's say the input for the array is: `coins={1, 3, 5}`.

css

 Copy code

```
int dp[targetAmount+1];
dp[0]=0;
for(int i=1;i<=targetAmount;i++) dp[i]=INT_MAX;
```

In this section, we are initializing the `dp` array. The `dp` array will store the minimum number of coins required for each value from 0 to the target amount.

The first line defines an integer array called `dp`, with $\text{targetAmount}+1$ elements. This is because we need to store the minimum number of coins required for all values from 0 to targetAmount .

The second line initializes `dp[0]` to 0, because the minimum number of coins required to make 0 is always 0.

The for loop then initializes all other elements of the `dp` array to a very large number, which is represented by `INT_MAX`. This is because we will be taking the minimum of `dp[j]` and `dp[j-coinValue]+1`, and we need to make sure that `dp[j]` is initially a very large number to ensure that the minimum value is updated correctly.

Let's say, after this section of the code, `dp={0, INT_MAX, INT_MAX, INT_MAX, INT_MAX, INT_MAX, INT_MAX, INT_MAX}`.

css

 Copy code

```
for(int i=0;i<n;i++){
    for(int j=coins[i];j<=targetAmount;j++){
        dp[j]=min(dp[j],dp[j-coins[i]]+1);
    }
}
```

This is the main loop of the code, where we update the dp array.

The outer loop runs through all the coins, and the inner loop runs through all values from the current coin value to the target amount.

For each value j , we update $dp[j]$ to be the minimum of its current value and $dp[j - \text{coinValue}] + 1$. This means that we are checking whether it is better to use the current coin to make the value j or to use coins we have already considered.

Let's go through an example:

For the first coin (value=1), the loop starts at $j=1$ and goes up to $j=7$ (target amount).

- For $j=1$, we have $dp[1] = \min(\text{INT_MAX}, dp[1-1]+1) = \min(\text{INT_MAX}, dp[0]+1) = \min(\text{INT_MAX}, 0+1) = 1$. So we update $dp[1]$ to 1.
- For $j=2$, we have $dp[2] = \min(\text{INT_MAX}, dp[2-1]+1) = \min(\text{INT_MAX}, dp[1]+1) = \min(\text{INT_MAX}, 1+1) = 2$. So we update $dp[2]$ to 2.
- For $j=3$, we have $dp[3] = \min(\text{INT_MAX}, dp[3-1]+1) = \min(\text{INT_MAX}, dp[2]+1) = \min(\text{INT_MAX},$



SH

can represent the example little more neatly



Sure! Here's a neater example:

Suppose we have the following input:

```
makefile
```

Copy code

```
n = 3
```

```
targetAmount = 7
```

```
coins = {1, 3, 5}
```

We start by initializing the dp array:

```
makefile
```

Copy code

```
dp = {0, INF, INF, INF, INF, INF, INF, INF}
```

Here, INF is a large number (we use `INT_MAX`) that represents the value of the minimum number of coins when no solution is possible.

Next, we loop through each coin and update the dp array accordingly. For the first coin with a value of 1, the loop looks like this:

scss

 Copy code

```
for(int j = 1; j <= targetAmount; j++){  
    dp[j] = min(dp[j], dp[j - coinValue] + 1);  
}
```

- For `j=1`, we update `dp[1]` to be the minimum of `dp[1]` and `dp[1-1]+1`. Since `dp[0] = 0`, this evaluates to `min(INF, 0+1)`, which is 1.
- For `j=2`, we update `dp[2]` to be the minimum of `dp[2]` and `dp[2-1]+1`. Since we just updated `dp[1]` to be 1, `dp[2-1]` is also 1. Therefore, we have `min(INF, 1+1)`, which is 2.
- For `j=3`, we update `dp[3]` to be the minimum of `dp[3]` and `dp[3-1]+1`. Again, since we just updated `dp[2]` to be 2, `dp[3-1]` is also 2. Therefore, we have `min(INF, 2+1)`, which is 3.

We continue this process for each coin, and after the loop is complete, the `dp` array will look like this:

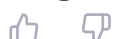
makefile

 Copy code

```
dp = {0, 1, 2, 1, 2, 1, 2, 3}
```

This means that we can make the target amount of 7 using 3 coins: one coin with a value of 5 and two coins with a value of 1.

If `dp[targetAmount]` is still INF, then it is not possible to make the target amount with the given coins.



SH

how did you understand the final dp array?

how did you conclude as below:-

one coin with a value of 5 and two coins with a value of 1.



We can understand the final `dp` array by interpreting each value as the minimum number of coins needed to make the corresponding amount. For example, `dp[1] = 1` means that it takes 1 coin to make an amount of 1, which is the coin with the smallest value (1).

Similarly, `dp[2] = 2` means that it takes 2 coins to make an amount of 2. We know from the available coins that we can make 2 using a coin with value 1, so we need two of those coins.

In general, if `dp[j] = k`, it means that it takes `k` coins to make an amount of `j`. To determine which coins are needed, we can work backwards through the `dp` array and check which coins were used to update each value.

In our example, we have `dp[7] = 3`, which means that it takes 3 coins to make an amount of 7. To determine which coins are needed, we can start at `dp[7]` and work backwards as follows:

- We know that we used a coin with value 5 to update `dp[7]` (i.e., `dp[7-5] + 1 = 2 + 1 = 3`).
- We now need to make an amount of 2. We know from the `dp` array that we can make 2 using two coins with a value of 1, so we use those.
- Finally, we need to make an amount of 1. We can use another coin with a value of 1 to do that.

Therefore, to make an amount of 7 using coins with values {1, 3, 5}, we need one coin with a value of 5 and two coins with a value of 1.



SH

ok does our code have back track logic ?

include inline comments to show algo+debug info



The coin change problem solved using dynamic programming does not require backtracking. It is a bottom-up approach that builds the `dp` array iteratively, and the final solution is simply the value of `dp[targetAmount]`.

Here's the code with inline comments that explain the algorithm and provide debug information:

C++

 Copy code

```
#include <iostream>
#include <vector>
#include <limits> // For INT_MAX

using namespace std;

int coinChange(vector<int>& coins, int targetAmount) {
    int n = coins.size();

    // Initialize the dp array with INF (i.e., no solution)
    vector<int> dp(targetAmount + 1, INT_MAX);
    dp[0] = 0;

    // Build the dp array iteratively
    for (int i = 0; i < n; i++) {
        int coinValue = coins[i];
        for (int j = coinValue; j <= targetAmount; j++) {
            dp[j] = min(dp[j], dp[j - coinValue] + 1);
        }
    }

    // Debug information
    for (int i = 0; i <= targetAmount; i++) {
        cout << "dp[" << i << "] = " << dp[i] << endl;
    }

    // Return the solution (i.e., the minimum number of coins needed to make targetAmount)
    return (dp[targetAmount] == INT_MAX) ? -1 : dp[targetAmount];
}

int main() {
    vector<int> coins = {1, 3, 5};
```

```
int targetAmount = 7;  
int ans = coinChange(coins, targetAmount);  
cout << "Minimum number of coins needed: " << ans << endl;  
return 0;  
}
```

The debug information section simply prints out the values of the `dp` array to check if the algorithm is working correctly. It can be removed for the final version of the code.

