**Task 6**

1. **Detailed assumptions:**

* The 12 coins are divided into 3 groups each group has 4 coins
* We can enter the weight of the coins or use the default weights
* The coins[] array by default has a fake coin is the last one. to use this assumption we put the first weight equals -1 and the last coin will be set by default as a fake coin and will be printed
* The coins weight are stored in coins[] array
* The states of groups that we divided our coins into are stored in groupsStates[] array and it stores whether the group has a fake coin or not
* The state of every coin in the group that we know it contain a fake coin is stored in coinsInGroupState[] array
* The group[] array contains the sum of weights of each group

1. **Problem Description:**

* There are 12 coins identical in appearance
* either all are genuine or exactly one of them is fake. It is unknown whether
* the fake coin is lighter or heavier than the genuine one.
* You have a two-pan balance scale without weights.
* The problem is to find whether all the coins are genuine and, if not, to find the fake coin and establish whether it is lighter or heavier than the genuine ones.

1. **Explaining solution:**

**Dynamic programming:**

Using memorizing to reduce complexity and dividing the problem into sub problems using the following steps:

1. Initialize the problem:
2. Declare an array 'coins' that holds the weights of the 12 coins.
3. Declare three elements array 'groupsState' that represents the state of each group (unknown, fake or good).
4. Declare four elements array 'coinsInGroupState' that represents the state of each coin in a group (unknown, fake or good).
5. Declare an integer 'fakeIndex' that represents the index of the fake coin (if any).
6. Define helper functions:
7. Define a function 'sum(w1, w2, w3, w4)' that returns the sum of four weights.
8. Define a function 'sum(w1, w2)' that returns the sum of two weights.
9. Define a function 'balance(w1, w2)' that returns true if two weights are equal, and false otherwise.
10. Calculate the fake group (if any):
11. Calculate the sum of the first four coins, the second four coins and the third four coins and store them in the 'group' array.
12. Check if group 0 and 1 are balanced, if so, set the state of groups 0 and 1 to 'good'. If group 0 and 2 are balanced, set the state of groups 0 and 2 to 'good'. If group 1 and 2 are balanced, set the state of groups 1 and 2 to 'good'.
13. If one group is fake, determine which group is fake and set its state to 'fake' in the 'groupsState' array.
14. Calculate the fake coin (if any):
15. For each coin in the fake group, determine its state by checking its weight against the weights of the other coins in the group. Set the state of each coin in the 'coinsInGroupState' array accordingly.
16. If one coin in the fake group is fake, determine which coin is fake and return its index as 'fakeIndex'.
17. Print the results:
18. If 'fakeIndex' is -1, print "no fake coin". Otherwise, print the index of the fake coin, its group number, and its position in the group.
19. End the program.

**Explanation:**

Sure, I can explain the dynamic programming solution to the problem of finding the minimum number of jumps needed to reach the end of the array.

The key idea behind the dynamic programming solution is to break down the problem into smaller subproblems and store the solutions to these subproblems in an array. We can then use these solutions to solve the larger problem.

1. **Pseudo code:**

function findFakeCoin(coins)

group1 = sum(coins[0], coins[1], coins[2], coins[3])

group2 = sum(coins[4], coins[5], coins[6], coins[7])

group3 = sum(coins[8], coins[9], coins[10], coins[11])

group\_sums = [group1, group2, group3]

# Initialize the memoization table with -1 for unknown state

memo = [[[-1 for \_ in range(13)] for \_ in range(13)] for \_ in range(13)]

# Helper function for checking if two groups are balanced

def is\_balanced(g1, g2):

return g1 == g2

# Helper function for checking if a group has a fake coin

def has\_fake(group):

return sum(group) - min(group) < min(group)

# Helper function to get the fake coin index from a group

def get\_fake\_index(group):

for i in range(len(group)):

if group[i] == min(group):

return i

return -1

def dp\_helper(n\_g1, n\_g2, n\_g3):

if memo[n\_g1][n\_g2][n\_g3] != -1:

return memo[n\_g1][n\_g2][n\_g3]

if n\_g1 == 1 and n\_g2 == 1 and n\_g3 == 0:

memo[n\_g1][n\_g2][n\_g3] = get\_fake\_index(group1)

return memo[n\_g1][n\_g2][n\_g3]

elif n\_g1 == 1 and n\_g2 == 0 and n\_g3 == 1:

memo[n\_g1][n\_g2][n\_g3] = get\_fake\_index(group2)

return memo[n\_g1][n\_g2][n\_g3]

elif n\_g1 == 0 and n\_g2 == 1 and n\_g3 == 1:

memo[n\_g1][n\_g2][n\_g3] = get\_fake\_index(group3)

return memo[n\_g1][n\_g2][n\_g3]

g1\_fake = has\_fake(group\_sums[:2])

g2\_fake = has\_fake(group\_sums[1:])

g3\_fake = has\_fake([group\_sums[2], group\_sums[0]])

if g1\_fake and dp\_helper(n\_g1 - 1, n\_g2, n\_g3) != -1:

memo[n\_g1][n\_g2][n\_g3] = dp\_helper(n\_g1 - 1, n\_g2, n\_g3)

return memo[n\_g1][n\_g2][n\_g3]

if g2\_fake and dp\_helper(n\_g1, n\_g2 - 1, n\_g3) != -1:

memo[n\_g1][n\_g2][n\_g3] = dp\_helper(n\_g1, n\_g2 - 1, n\_g3)

return memo[n\_g1][n\_g2][n\_g3]

if g3\_fake and dp\_helper(n\_g1, n\_g2, n\_g3 - 1) != -1:

memo[n\_g1][n\_g2][n\_g3] = dp\_helper(n\_g1, n\_g2, n\_g3 - 1)

return memo[n\_g1][n\_g2][n\_g3]

memo[n\_g1][n\_g2][n\_g3]

1. **Implementation:**

#include <iostream>

using namespace std;

int sum(int w1, int w2, int w3, int w4);

int sum(int w1, int w2);

/\*macros for states\*/

#define unknown -1

#define fake 0

#define good 1

/\*any weights we can change it as we want\*/

#define normalWeight 10

#define heavyWeight 11

#define loghtWeight 9

/\*the weights of coins we can change it as we want\*/

int x;

int coins[12] = { normalWeight,normalWeight,normalWeight,normalWeight,normalWeight,normalWeight,normalWeight,normalWeight,normalWeight,normalWeight,heavyWeight,normalWeight};

int groupsState[3] = { unknown,unknown,unknown };

int coinsInGroupState[4] = { unknown,unknown,unknown,unknown };

int group[3] = { sum(coins[0], coins[1], coins[2], coins[3]) ,sum(coins[4], coins[5], coins[6], coins[7]),sum(coins[8], coins[9], coins[10], coins[11]) };

int fakeIndex = -1;

int sum(int w1, int w2, int w3, int w4) {

return w1 + w2 + w3 + w4;

}

int sum(int w1, int w2) {

return w1 + w2;

}

bool balance(int w1, int w2) {

if (w1 == w2) return true;

else return false;

}

int getFakeIndex() {

/\*calculating fake group\*/

if (balance(group[0], group[1])) {//no fake in group 0 or 1

groupsState[0] = good;

groupsState[1] = good;

if (group[0] != group[2]) {

groupsState[2] = fake;

}

}

if (balance(group[1], group[2])) {//no fake in group 1 or 2

groupsState[1] = good;

groupsState[2] = good;

if (group[0] != group[2]) {

groupsState[0] = fake;

}

}

if (balance(group[2], group[0])) {//no fake in group 2 or 0

groupsState[2] = good;

groupsState[0] = good;

if (!balance(group[0], group[1])) {

groupsState[1] = fake;

}

}

/\*fake group calculated if exists\*/

/\*finding fake coin\*/

for (int i = 0; i < 4; i++) {

if (groupsState[i] == fake) {

if (coins[i \* 4 + 0] == coins[i \* 4 + 1]) {

coinsInGroupState[0] = good;

coinsInGroupState[1] = good;

}

if (balance(coins[i \* 4 + 1], coins[i \* 4 + 2])) {

coinsInGroupState[1] = good;

coinsInGroupState[2] = good;

}

if (balance(coins[i \* 4 + 2], coins[i \* 4 + 3])) {

coinsInGroupState[2] = good;

coinsInGroupState[3] = good;

}

if (balance(coins[i \* 4 + 3], coins[i \* 4 + 0])) {

coinsInGroupState[3] = good;

coinsInGroupState[0] = good;

}

for (int j = 0; j < 4; j++) {

if (coinsInGroupState[j] == unknown) {

coinsInGroupState[j] = fake;

return i \* 4 + j;

}

}

return -1;

}

}

//no coin is fake

return -1;

}

int main() {

int inputWeight;

for (int i = 0; i < 12; i++) {

cout << "enter the weight of coin " << i + 1 << ":" << endl;

cin >> inputWeight;

if (inputWeight == -1) break;

coins[i] = inputWeight;

}

int fakeIndex = getFakeIndex();

int groupNumber;

int coinNumber;

if (fakeIndex == -1) {

cout << "no fake coin";

}

else {

groupNumber = (fakeIndex / 4) + 1;

coinNumber = (fakeIndex % 4) + 1;

cout << "the fake coin is the number " << fakeIndex + 1 << " it is in group " << groupNumber << " and it is the number " << coinNumber << " in the group. (the numbers start from 1 not 0 in printing)" << endl;

}

cin >> x;

return 0;

}

1. **Time and Space Complexity:**

Time complexity:

The program has to iterate over all 12 coins to get their weight, which takes O(n) time, where n is the number of coins.

After that, the program calls the getFakeIndex function, which has to perform several operations:

Check if the first two groups are balanced, which takes O(1) time.

Check if the second and third groups are balanced, which takes O(1) time.

Check if the third and first groups are balanced, which takes O(1) time.

Iterate over the coins in the fake group to find the fake coin, which takes O(4) = O(1) time.

Overall, the getFakeIndex function takes O(1) time.

The program then prints the output, which takes O(1) time.

Therefore, the overall time complexity of the program is O(n).

Space complexity:

The program uses a constant amount of space for the macros and weight variables, which is O(1).

The program also uses the following arrays to store information:

coins array, which stores the weights of the coins. This array has a size of 12, so it takes O(12) = O(n) space.

groupsState array, which stores the state of each group (fake, good, or unknown). This array has a size of 3, so it takes O(3) = O(1) space.

coinsInGroupState array, which stores the state of each coin in the fake group (fake, good, or unknown). This array has a size of 4, so it takes O(4) = O(1) space.

group array, which stores the sum of weights for each group. This array has a size of 3, so it takes O(3) = O(1) space.

Therefore, the overall space complexity of the program is O(n).

In summary, the time complexity of the algorithm is linear in the input size.

1. **Another solution:**

**Brute Force Algorism**

#include <iostream>

using namespace std;

int balance(int w1, int w2);

int coins[12] = {10,10,10,10,10,10,10,10,10,10,10,10};//values are weights of coins

int fake\_coin\_index = -1;

int main()

{

if (balance(coins[0], coins[1]) == 0) {//the coins are equal no one is fake

for (int i = 2; i < 12; i++) {

if (balance(coins[0], coins[i]) != 0) {//if not equal

fake\_coin\_index = i;

}

}

}

else if (balance(coins[0], coins[1]) != 0) {//the coins are not equals

if (coins[0] != coins[2]) {//coins[0] is fake

fake\_coin\_index = 0;

}

else if(balance(coins[0],coins[2]) == 0) {//coins[1] is fake because 0,2 are equal

fake\_coin\_index = 1;

}

}

//print the coin

if (fake\_coin\_index == -1) {//default value

cout << "there is no fake coin";

}else if(fake\_coin\_index == 0) {//the first with index 0 us fake

cout << "the first coin is fake";

}else if (fake\_coin\_index == 1) {//the secound with index 1 is fake

cout << "the secound coin is fake";

}else if (fake\_coin\_index == 2) {//the secound with index 2 is fake

cout << "the third coin is fake";

}

else {//the i coin is fake

cout << "the "<<(fake\_coin\_index+1)<<"th is fake";

}

}

int balance(int w1,int w2) {

if (w1 == w2) return 0;

if (w1 > w2) return 1;

if (w1 < w2) return - 1;

}

1. **Input and outputs:**

Input: the coins weightings.

Output: the fake coin if exist.

1. **Conclusions:**

In conclusion, the time complexity of this program is O(1) and the space complexity is O(1) as well. This is because the program takes a fixed input size of 12 and does not require any additional memory allocation beyond the fixed size of the arrays used in the program. The program uses a combination of simple arithmetic operations and conditional statements to solve the problem and does not require any advanced algorithmic techniques.

1. **References:**

*https://centerofmathematics.blogspot.com/2017/09/think-thursday-9-21-17-twelve-coins.html*