Securin

ROHETH S

roheth1908@gmail.com

Final Year at Sri Venkateswara College of Engineering (SVCE)

Problem Statement: The Doomed Dice Challenge

You are given two six-sided dice, Die A and Die B, each with faces numbered from 1 to 6.

You can only roll both the dice together & your turn is guided by the obtained sum.

Example: Die A = 6, Die B = 3. Sum = 6 + 3 = 9

You may represent Dice as an Array or Array-like structure.

Die A = [1, 2, 3, 4, 5, 6] where the indices represent the 6 faces of the die & the value on each face.

Part-A:

1. How many total combinations are possible? Show the math along with the code!

Code:

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


void taski_totalcombination(vector<int>Die1, vector<int>Die2)
{

    int totalcombination=Die1.size()*Die2.size(); //(direct approach)

    // int totalcombination=0;

    // for(int i=0; i<Die1.size(); i++)
    // {
    //     for(int j=0; j<Die2.size(); j++)
    //     {
    //         totalcombination++;
    //     }

    // }

    cout<<"Total Combination :"<<" Size of Die1 X Size of Die2: "<<totalcombination<<endl;
    cout<<endl;
}
```

Explanation:

The code calculates the total number of combinations when choosing one element from the vector Die1 and one element from the vector Die2. It does this by multiplying the sizes of the two vectors. The result is then printed as the total combination. The commented-out section provides an alternative implementation using nested loops to achieve the same result.

Output:

```
Total Combination : Size of Die1 X Size of Die2: 36
```

2. **Calculate and display the distribution of all possible combinations that can be obtained when rolling both Die A and Die B together. Show the math along with the code! Hint: A 6 x 6 Matrix.**

Code:

```
void taskii_combinationspossible(vector<int>Die1, vector<int>Die2)
{
    cout<<"All Possible Combinations:"<<endl;
    cout<<endl;

    for(int i=1; i<=Die1.size(); i++)
    {
        for(int j=1; j<=Die2.size(); j++)
        {
            cout<<'('<<i<<","<<j<<')'<<" ";
        }

        cout<<endl;
    }

    cout<<endl;
}
```

Explanation:

The code generates and prints all possible combinations of indices (i, j) where i ranges from 1 to the size of Die1 and j ranges from 1 to the size of Die2. It represents the Cartesian product of the index sets [1, size of Die1] and [1, size of Die2]. Each combination is displayed in the format (i, j).

Output:

```
All Possible Combinations:

(1,1) (1,2) (1,3) (1,4) (1,5) (1,6)
(2,1) (2,2) (2,3) (2,4) (2,5) (2,6)
(3,1) (3,2) (3,3) (3,4) (3,5) (3,6)
(4,1) (4,2) (4,3) (4,4) (4,5) (4,6)
(5,1) (5,2) (5,3) (5,4) (5,5) (5,6)
(6,1) (6,2) (6,3) (6,4) (6,5) (6,6)
```

3. Calculate the Probability of all Possible Sums occurring among the number of combinations from (2).

Example: P(Sum = 2) = 1/X as there is only one combination possible to obtain Sum = 2. Die A = Die B = 1.

Code:

```
void taskiii_probability(vector<int>Die1, vector<int>Die2)
{

    cout<<"All Possible Sums:"<<endl;
    cout<<endl;

    map<int, float>combination;
    set<int>unique;
    map<int, vector<vector<int>>>values;

    for(int i=0; i<Die1.size(); i++)
    {
        for(int j=0; j<Die2.size(); j++)
        {
            vector<int>val;
            val.push_back(i+1);
            val.push_back(j+1);

            int sum=0;
            sum=Die1[i]+Die2[j];

            values[sum].push_back(val);

            if(!unique.count(sum))
            {
                cout<<sum<<" ";
            }
            unique.insert(sum);
            combination[sum]++;
        }
    }
    cout<<endl;
    cout<<endl;
```

```
    cout << "Combinations for Each Sum:" << endl;
    cout<<endl;

    for (auto x : values)
    {
        cout << "Sum " << x.first << ": ";

        for (auto v : x.second)
        {
            cout << "(" << v[0] << ", " << v[1] << ") ";
        }

        cout << endl;
    }
    cout<<endl;

    int totalcombination=Die1.size()*Die2.size();

    cout<<"Probability of All Possible Sums:"<<endl;
    cout<<endl;

    for(auto x: combination)
    {
        cout<<"probability(sum="<<x.first<<")="<<" ";
        cout<<x.second<<"(occurence)/"<<totalcombination<<"(totalcombination)="<<" ";
        float probability=x.second/totalcombination;
        cout<<probability;
        cout<<endl;
    }

}
```

Explanation:

The code calculates and displays the probabilities of all possible sums when rolling two dice (represented by vectors Die1 and Die2). It computes the occurrence of each sum, stores the combinations leading to those sums, and then calculates the probability for each sum by dividing its occurrence by the total number of possible combinations. The final output includes unique sums, the combinations contributing to each sum, and their respective probabilities.

Output:

```
All Possible Sums:

2 3 4 5 6 7 8 9 10 11 12

Combinations for Each Sum:

Sum 2: (1, 1)
Sum 3: (1, 2) (2, 1)
Sum 4: (1, 3) (2, 2) (3, 1)
Sum 5: (1, 4) (2, 3) (3, 2) (4, 1)
Sum 6: (1, 5) (2, 4) (3, 3) (4, 2) (5, 1)
Sum 7: (1, 6) (2, 5) (3, 4) (4, 3) (5, 2) (6, 1)
Sum 8: (2, 6) (3, 5) (4, 4) (5, 3) (6, 2)
Sum 9: (3, 6) (4, 5) (5, 4) (6, 3)
Sum 10: (4, 6) (5, 5) (6, 4)
Sum 11: (5, 6) (6, 5)
Sum 12: (6, 6)
```

```
Probability of All Possible Sums:

probability(sum=2)= 1(occurence)/36(totalcombination)= 0.0277778
probability(sum=3)= 2(occurence)/36(totalcombination)= 0.0555556
probability(sum=4)= 3(occurence)/36(totalcombination)= 0.0833333
probability(sum=5)= 4(occurence)/36(totalcombination)= 0.111111
probability(sum=6)= 5(occurence)/36(totalcombination)= 0.138889
probability(sum=7)= 6(occurence)/36(totalcombination)= 0.166667
probability(sum=8)= 5(occurence)/36(totalcombination)= 0.138889
probability(sum=9)= 4(occurence)/36(totalcombination)= 0.111111
probability(sum=10)= 3(occurence)/36(totalcombination)= 0.0833333
probability(sum=11)= 2(occurence)/36(totalcombination)= 0.0555556
probability(sum=12)= 1(occurence)/36(totalcombination)= 0.0277778
```

Main function:

```
int main()
{
    vector<int>Die1={1,2,3,4,5,6};
    vector<int>Die2={1,2,3,4,5,6};

    taski_totalcombination(Die1, Die2);
    taskii_combinationspossible(Die1, Die2);
    taskiii_probability(Die1, Die2);

    return 0;
}
```

Explanation:

The main function initializes two vectors, Die1 and Die2, representing the faces of two six-sided dice. It then calls three functions (taski_totalcombination, taskii_combinationspossible, and taskiii_probability) passing these vectors as arguments. Each function performs a different task related to the simulation of rolling two dice, such as calculating total combinations, listing possible combinations, and computing probability distributions.

Code link: https://onlinegdb.com/hftuwGwsC

Part-B:

Now comes the real challenge. You were happily spending a lazy afternoon playing your board game with your dice when suddenly the mischievous Norse God Loki (You love Thor too much & Loki didn't like that much) appeared. Loki dooms your dice for his fun removing all the "Spots" off the dice.

No problem! You have the tools to re-attach the "Spots" back on the Dice. However, Loki has doomed your dice with the following conditions:

● Die A cannot have more than 4 Spots on a face.

● Die A may have multiple faces with the same number of spots.

● Die B can have as many spots on a face as necessary i.e. even more than 6.

But in order to play your game, the probability of obtaining the Sums must remain the same!

So if you could only roll P(Sum = 2) = 1/X, the new dice must have the spots reattached such that those probabilities are not changed.

Input:

● Die_A = [1, 2, 3, 4, 5, 6] & Die B = Die_A = [1, 2, 3, 4, 5, 6]

Output:

● A Transform Function undoom_dice that takes (Die_A, Die_B) as input &

outputs New_Die_A = [?, ?, ?, ?, ?, ?],New_Die_B = [?, ?,

?, ?, ?, ?] where,

● No New_Die A[x] > 4

Code:

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

unordered_map<int, int> frequency;
vector<int>DieA;
vector<int>DieB;
```

Explanation:

This code includes necessary headers and declares an unordered map frequency to store integer frequencies. It also declares two vectors DieA and DieB to represent the faces of two dice.

Undoom_Dice Function:

```
void undoom_dice(vector<int> &Die3, vector<int> &Die4)
{
    unordered_map<int, int> frequency2;
    bool flag = true;

    for (int i = 0; i < Die3.size(); i++)
    {
        for (int j = 0; j < Die4.size(); j++)
        {
            frequency2[Die3[i] + Die4[j]]++;

            if (frequency2[Die3[i] + Die4[j]] > frequency[(Die3[i] + Die4[j])])
            {
                flag = false;
                break;
            }
        }

        if (!flag)
        {
            break;
        }
    }

    if (flag)
    {
        DieA=Die3;
        DieB=Die4;

    }
}
```

Explanation:

The code checks if the sum of the corresponding faces of two dice (represented by vectors Die3 and Die4) satisfies a predefined frequency condition. It iterates through each combination of faces, updating a frequency counter for the sum of faces. If at any point the frequency exceeds a specified limit (determined by the global frequency map), the process is terminated. If all sums meet the frequency condition, the faces of the dice are stored in global vectors DieA and DieB. The function essentially identifies valid combinations of dice faces that satisfy a given frequency constraint.

Possibledie4 and Possible3 Functions:

```
void possibledie4(vector<int> &Die3, vector<int> &Die4, int n)
{
    if (n == 5)
    {
        undoom_dice(Die3, Die4);
        return;
    }

    for (int i = 0; i < 8; i++)
    {
        Die4[n] = i + 1;
        possibledie4(Die3, Die4, n + 1);
    }
}

void possibledie3(vector<int> &Die3, int n)
{
    if (n == 5)
    {
        vector<int> Die4 = {1, 0, 0, 0, 0, 8};
        possibledie4(Die3, Die4, 1);
        return;
    }

    for (int i = 0; i < 4; i++)
    {
        Die3[n] = i + 1;
        possibledie3(Die3, n + 1);
    }
}
```

Explanation:

The code recursively generates all possible combinations of faces for two six-sided dice, represented by vectors Die3 and Die4. The function possibledie3 iterates through possible faces for the first die, and for each combination, it calls possibledie4 to explore all faces for the second die. The process continues until all possible combinations are explored, and a function undoom_dice is called when the combination for the second die is completed. The ultimate goal is to explore and process all possible configurations of faces for the two dice.

Probability Check Function:

```
void probabilitycheck(vector<int>Die1, vector<int>Die2)
{
    map<int, float>combination;

    for(int i=0; i<Die1.size(); i++)
    {
        for(int j=0; j<Die2.size(); j++)
        {
            int sum=0;
            sum=Die1[i]+Die2[j];
            combination[sum]++;
        }
    }

    int totalcombination=Die1.size()*Die2.size();

    cout<<"Probability Check of New Dice:"<<endl;
    cout<<endl;

    for(auto x: combination)
    {
        cout<<"probability(sum="<<x.first<<")="<<" ";
        cout<<x.second<<"(occurence)/"<<totalcombination<<"(totalcombination)="<<" ";
        float probability=x.second/totalcombination;
        cout<<probability;
        cout<<endl;
    }

}
```

Explanation:

The code calculates and prints the probability distribution of all possible sums when combining the faces of two dice represented by vectors Die1 and Die2. It counts the occurrences of each sum, computes the probability for each sum by dividing its occurrence by the total number of possible combinations, and then displays the result. The function provides insights into the likelihood of obtaining different sums when rolling the two dice.

Print Function:

```
void print(vector<int>DieA, vector<int>DieB)
{
    cout<<"Loki's Doomed Dice with No Change in Probability";
    cout<<endl<<endl;

    sort(DieA.begin(), DieA.end());
    sort(DieB.begin(), DieB.end());

    cout << "New Die1: ";

        for (int i = 0; i < DieA.size(); i++)
        {
            cout << DieA[i]<<" ";

        }
        cout << endl;

        cout << "New Die2: ";

        for (int i = 0; i < DieB.size(); i++)
        {
            cout << DieB[i]<<" ";
        }

        cout << endl<< endl;

}
```

Explanation:

The code prints the faces of two dice, represented by vectors DieA and DieB, after sorting them in ascending order. It is part of a larger context where these dice are manipulated, and the function provides a visual representation of the ordered faces for analysis or display purposes. The sorting ensures a clear presentation of the dice faces, and the output reflects the current configuration of the dice.

Main Function:

```
int main()
{
    vector<int> Die1 = {1, 2, 3, 4, 5, 6};
    vector<int> Die2 = {1, 2, 3, 4, 5, 6};

    for (int i = 0; i < Die1.size(); i++)
    {
        for (int j = 0; j < Die2.size(); j++)
        {
            int sum = Die1[i] + Die2[j];
            frequency[sum]++;
        }
    }

    vector<int> Die3 = {1, 0, 0, 0, 0, 4};

    possibledie3(Die3, 1);

    print(DieA, DieB);

    probabilitycheck(DieA, DieB);


    return 0;
}
```

Explanation:

The main function simulates rolling two six-sided dice (Die1 and Die2) and calculates the frequency distribution of their sums using a global frequency map. It then sets up a predefined configuration for a third die (Die3) and generates all possible configurations for a fourth die (Die4) using recursive functions. Afterward, it prints the ordered faces of the third and fourth dice using the print function and checks the probability distribution of the sums of the third and fourth dice using the probabilitycheck function. The program aims to analyze and manipulate dice configurations and their probabilities in a specific context.

Output:

```
Loki's Doomed Dice with No Change in Probability

New Die1: 1 2 2 3 3 4
New Die2: 1 3 4 5 6 8

Probability Check of New Dice:

probability(sum=2)= 1(occurence)/36(totalcombination)= 0.0277778
probability(sum=3)= 2(occurence)/36(totalcombination)= 0.0555556
probability(sum=4)= 3(occurence)/36(totalcombination)= 0.0833333
probability(sum=5)= 4(occurence)/36(totalcombination)= 0.111111
probability(sum=6)= 5(occurence)/36(totalcombination)= 0.138889
probability(sum=7)= 6(occurence)/36(totalcombination)= 0.166667
probability(sum=8)= 5(occurence)/36(totalcombination)= 0.138889
probability(sum=9)= 4(occurence)/36(totalcombination)= 0.111111
probability(sum=10)= 3(occurence)/36(totalcombination)= 0.0833333
probability(sum=11)= 2(occurence)/36(totalcombination)= 0.0555556
probability(sum=12)= 1(occurence)/36(totalcombination)= 0.0277778
```

Code Link: https://onlinegdb.com/R5KFFRllw