**Problem Statement: The Doomed Dice Challenge**

**Part-A**

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

**Solution:**

To calculate the total number of combinations when rolling two dice together, we can use the concept of combinatorics. The number of combinations is equal to the product of the number of faces on each die. In this case, both dice have 6 faces.

Mathematically, the total number of combinations (C) is given by:

C=Number of faces on Die A×Number of faces on Die B*C*=Number of faces on Die A×Number of faces on Die B

For a standard pair of six-sided dice, this is:

C = 6×6 = 36

Implementation of code of c++.

#include <iostream>

Using namespace std;

int main() {

// Number of faces on each die

const int faces\_A = 6;

const int faces\_B = 6;

// Calculate total combinations

int total\_combinations = faces\_A \* faces\_B;

// Output the result

cout << "Total combinations: " << total\_combinations << std::endl;

return 0;

}

Output of this program is: **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!**

To calculate the distribution of all possible combinations when rolling both Die A and Die B together, we need to consider every possible sum that can be obtained. Each sum corresponds to a unique combination of faces on Die A and Die B.

Here's the code to calculate and display the distribution:



A screenshot of a computer

Description automatically generated

Explanation:

1. We use a vector **sum\_distribution** to store the count of combinations for each possible sum. The vector is initialized to have elements for sums from 2 to 12 (the maximum sum for two six-sided dice).
2. Nested loops iterate over all possible combinations of faces on Die A and Die B. For each combination, the corresponding sum is calculated, and the count in **sum\_distribution** is incremented.
3. The final step is to output the distribution for each sum.

The output should display the distribution of all possible combinations for sums ranging from 2 to 12.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Col1 | col2 | col3 | col4 | col5 | col6 |
| row1 | 2 | 3 | 4 | 5 | 6 | 7 |
| row2 | 3 | 4 | 5 | 6 | 7 | 8 |
| row3 | 4 | 5 | 6 | 7 | 8 | 9 |
| row4 | 5 | 6 | 7 | 8 | 9 | 10 |
| row5 | 6 | 7 | 8 | 9 | 10 | 11 |
| row6 | 7 | 8 | 9 | 10 | 11 | 12 |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| sum | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| Probability | 1 | 2 | 3 | 4 | 5 | 6 | 5 | 4 | 3 | 2 | 1 |

**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**

Solution:

To calculate the probability of each possible sum occurring among the number of combinations, you need to divide the number of combinations for each sum by the total number of combinations. The probability (*P*) of a specific sum (*S*) is given by:

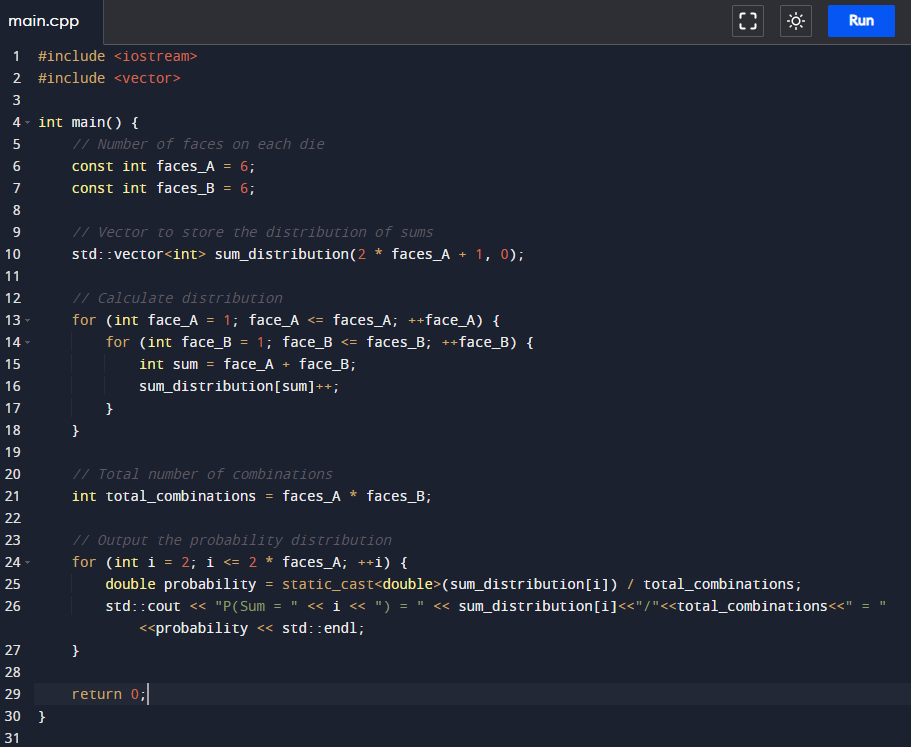
Below is the code to calculate and display the probability distribution:

Explanation:

1. We calculate the probability for each sum by dividing the number of combinations for that sum by the total number of combinations.
2. We use **static\_cast<double>** to ensure that the division results in a floating-point number (to obtain accurate probabilities).
3. The output displays the probability for each sum.

The output should show the probability of each possible sum occurring among the number of combinations.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| sum | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| Probability | 1 | 2 | 3 | 4 | 5 | 6 | 5 | 4 | 3 | 2 | 1 |



A screenshot of a computer

Description automatically generated

**Part-B**

Give condition New\_die\_a not have spots more than 4.but I we can take multiple spots in New\_die\_a and we also take greater than 6 spots in New\_die\_b.

Observation

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Col1 | col2 | col3 | col4 | col5 | col6 |
| row1 | 2 | 3 | 4 | 5 | 6 | 7 |
| row2 | 3 | 4 | 5 | 6 | 7 | 8 |
| row3 | 4 | 5 | 6 | 7 | 8 | 9 |
| row4 | 5 | 6 | 7 | 8 | 9 | 10 |
| row5 | 6 | 7 | 8 | 9 | 10 | 11 |
| row6 | 7 | 8 | 9 | 10 | 11 | 12 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Col1 | col2 | col3 | col4 | col5 | col6 |
| row1 | 2 | 3 | 4 | 5 |  |  |
| row2 | 3 | 4 | 5 | 6 |  |  |
| row3 | 4 | 5 | 6 | 7 |  |  |
| row4 | 5 | 6 | 7 | 8 |  |  |
| row5 | 6 | 7 | 8 | 9 |  |  |
| row6 | 7 | 8 | 9 | 10 |  |  |

The values 5 and 6 are excluded from our solution because they are greater than 4. Currently, our New\_Die\_A[x] is {1, 2, 3, 4}, and we need to add two more numbers. However, we cannot include 1 and 4. If we include multiple ones in New\_Die\_A, we cannot achieve the probability of obtaining (sum=2) = 1/36. Similarly, we cannot include 4 multiple times.

example:

suppose New\_Die\_A [] = {1,1,2,3,4,4}

New\_Die\_B [] = {1,2,3,4,5,6}

if we find the probability of getting sum=2.

possible outcome = (1,1), (1,1) = 2;

total outcome =36

P(sum=2) =2/36.

Our only viable option is to include (2, 3) in New\_Die\_A multiple time. Therefore, we have three possible cases:

Case 1 if New\_Die\_A = {1,2,2,2,3,4}.

Case 1 if New\_Die\_A = {1,2,2,3,3,4}.

Case 1 if New\_Die\_A = {1,2,3,3,3,4}.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| sum | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| Probability | 1 | 2 | 3 | 4 | 5 | 6 | 5 | 4 | 3 | 2 | 1 |

We use probability and New\_die\_A to find New\_die\_B.

Algorithm for generating New\_die\_B:

1. **Initialization:**
   * Set **s** to 1.
   * Set **i** to 1.
2. **Main Loop:**
   * While **i** is less than or equal to 8, repeat the following steps:
     1. Calculate the **next number** as the sum of **i** and **s**.
     2. Check If **sum[next]** is greater than 0:
        + For each face in **die**:
          1. Calculate **temp** as the sum of **i** and the current face.
          2. Decrement **sum[temp]** by 1.
          3. Add the current face to the end of **die\_b**.
     3. Increment **i** by 1.
3. **Check Remaining Sums:**
   * For each index **i** from 2 to the size of **sum**:
     1. If **sum[i]** is not equal to 0, return 0.
4. **Successful Execution:**
   * If the loop completes without returning, return 1.

Case 1 if New\_Die\_A = {1,2,2,2,3,4}.

The find function will Return 0 because of this combination will not generate same probability of all sums.

Case 2 New\_Die\_A = {1,2,2,3,3,4}.

The find function will Return 1 because of this combination will generate same probability of all sums and give the New\_die\_B= {1,3,4,5,6,8}.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 2 | 3 | 3 | 4 |
| 1 | 2 | 3 | 3 | 4 | 4 | 5 |
| 3 | 4 | 5 | 5 | 6 | 6 | 7 |
| 4 | 5 | 6 | 6 | 7 | 7 | 8 |
| 5 | 6 | 7 | 7 | 8 | 8 | 9 |
| 6 | 7 | 8 | 8 | 9 | 9 | 10 |
| 8 | 9 | 10 | 10 | 11 | 11 | 12 |

Case 3 if New\_Die\_A = {1,2,2,2,3,4}.

The find function will Return 0 because of this combination will not generate same probability of all sums.

Below is the implementation code:

#include<bits/stdc++.h>

using namespace std;

void print(vector<int> die\_a,vector<int> die\_b)

{

int n=die\_a.size();

for(int i=0;i<n;i++)

{

cout<<die\_a[i]<<" ";

}

cout<<endl;

for(int i=0;i<n;i++)

{

cout<<die\_b[i]<<" ";

}

cout<<endl;

}

int find(vector<int> die,vector<int> sum,vector<int> &die\_b)

{

int s=1;

int i=1;

while(i<=8)

{

int next=i+s;

if(sum[next]>0)

{

for(int k=0;k<die.size();k++)

{

int temp=i+die[k];

sum[temp]=sum[temp]-1;

}

die\_b.push\_back(i);

}

i++;

}

for(int i=2;i<sum.size();i++)

{

if(sum[i]!=0)

return 0;

}

return 1;

}

void undoom\_dice(vector<int> &die\_a,vector<int> &die\_b)

{

int n=die\_a.size();

vector<int> new\_die\_a={1,2,3,4};

vector<int> new\_die\_b;

vector<int> sum={0,0,1,2,3,4,5,6,5,4, 3, 2, 1};

// 2 3 4 5 6 7 8 9 10 11 12

// case 1 =(2 2)

new\_die\_a.push\_back(2);

new\_die\_a.push\_back(2);

int f1=find(new\_die\_a,sum,new\_die\_b);

if(f1==1)

{

die\_a=new\_die\_a;

die\_b=new\_die\_b;

return;

}

new\_die\_b.clear();

// cse 2 = (2,3)

new\_die\_a[5]=3;

int f2=find(new\_die\_a,sum,new\_die\_b);

if(f2==1)

{

die\_a=new\_die\_a;

die\_b=new\_die\_b;

return;

}

new\_die\_b.clear();

// case 3 = (3,3)

new\_die\_a[4]=4;

int f3=find(new\_die\_a,sum,new\_die\_b);

if(f3==1)

{

die\_a=new\_die\_a;

die\_b=new\_die\_b;

return;

}

}

int main()

{

vector<int> die\_a={1,2,3,4,5,6};

vector<int> die\_b={1,2,3,4,5,6};

int n=die\_a.size();

undoom\_dice(die\_a,die\_b);

print(die\_a,die\_b);

return 0;

}

