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

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

$C = 6 \times 6 = 36$

Implementation code in c++.

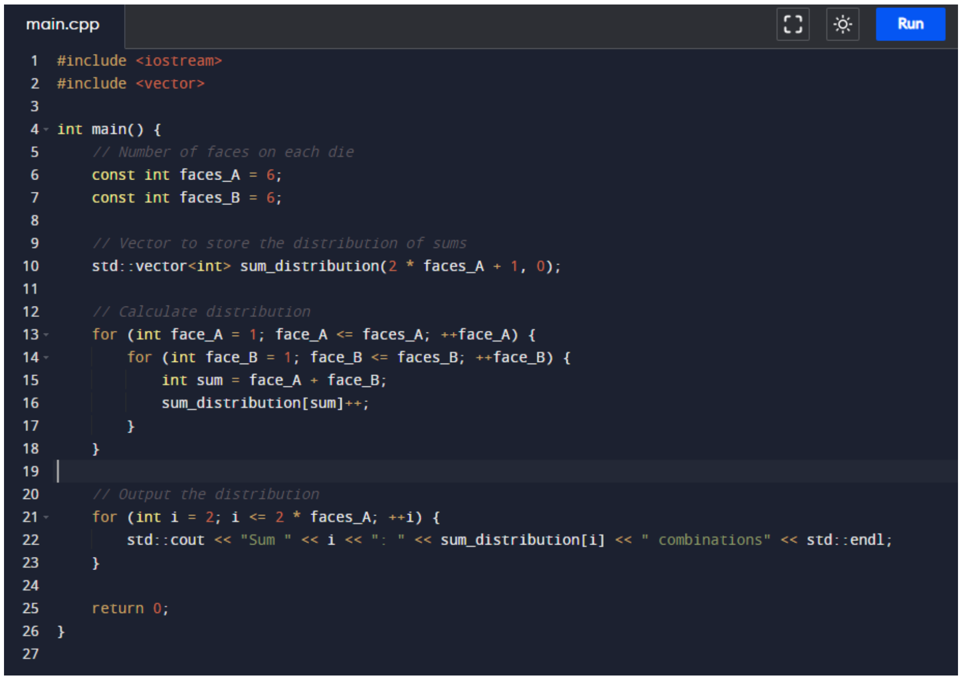
```
#include <iostream>

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**.

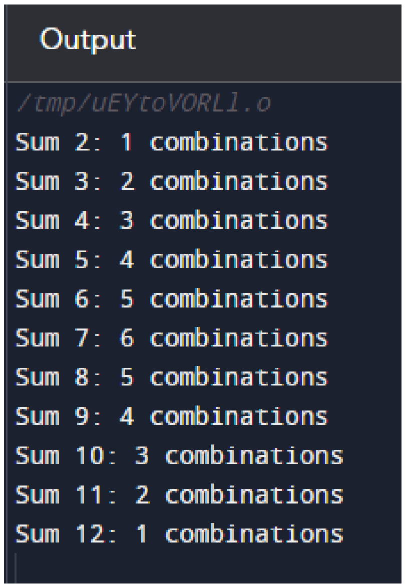
2.Calculate and display the distribution of all possible combinations that can be obtained when rolling both Die A and Die B together. Show the math along with the code!

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:


main.cpp

```cpp
#include <iostream>
#include <vector>

int main() {
    // Number of faces on each die
    const int faces_A = 6;
    const int faces_B = 6;

    // Vector to store the distribution of sums
    std::vector<int> sum_distribution(2 * faces_A + 1, 0);

    // Calculate distribution
    for (int face_A = 1; face_A <= faces_A; ++face_A) {
        for (int face_B = 1; face_B <= faces_B; ++face_B) {
            int sum = face_A + face_B;
            sum_distribution[sum]++;
        }
    }

    // Output the distribution
    for (int i = 2; i <= 2 * faces_A; ++i) {
        std::cout << "Sum " << i << ": " << sum_distribution[i] << " combinations" << std::endl;
    }

    return 0;
}
```


Output

```
/tmp/uEYtoVORLl.o
Sum 2: 1 combinations
Sum 3: 2 combinations
Sum 4: 3 combinations
Sum 5: 4 combinations
Sum 6: 5 combinations
Sum 7: 6 combinations
Sum 8: 5 combinations
Sum 9: 4 combinations
Sum 10: 3 combinations
Sum 11: 2 combinations
Sum 12: 1 combinations
```

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:

$$P(S) = \frac{\text{Number of combinations for sum S}}{\text{Tatal number of combinations}}$$

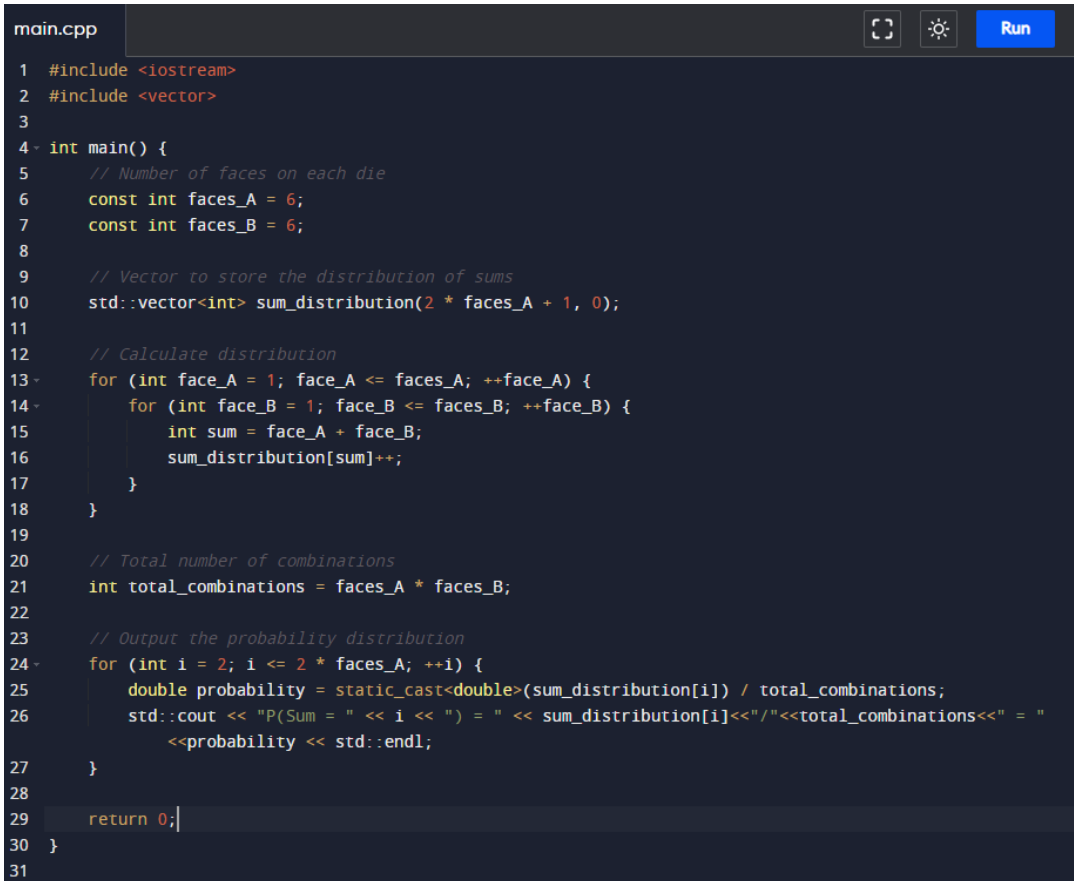
Below is the code to calculate and display the probability distribution:

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.

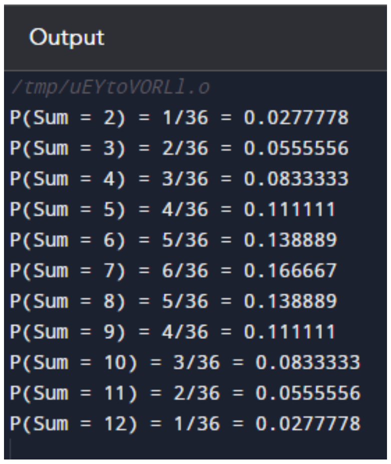
The output should show the probability of each possible sum occurring among the number of combinations.

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


main.cpp

Run

```cpp
#include <iostream>
#include <vector>

int main() {
    // Number of faces on each die
    const int faces_A = 6;
    const int faces_B = 6;

    // Vector to store the distribution of sums
    std::vector<int> sum_distribution(2 * faces_A + 1, 0);

    // Calculate distribution
    for (int face_A = 1; face_A <= faces_A; ++face_A) {
        for (int face_B = 1; face_B <= faces_B; ++face_B) {
            int sum = face_A + face_B;
            sum_distribution[sum]++;
        }
    }

    // Total number of combinations
    int total_combinations = faces_A * faces_B;

    // Output the probability distribution
    for (int i = 2; i <= 2 * faces_A; ++i) {
        double probability = static_cast<double>(sum_distribution[i]) / total_combinations;
        std::cout << "P(Sum = " << i << ") = " << sum_distribution[i]<<"/"<<total_combinations<<" = "
            <<probability << std::endl;
    }

    return 0;
}
```


Output

```
/tmp/uEYtoVORLl.o
P(Sum = 2) = 1/36 = 0.0277778
P(Sum = 3) = 2/36 = 0.0555556
P(Sum = 4) = 3/36 = 0.0833333
P(Sum = 5) = 4/36 = 0.111111
P(Sum = 6) = 5/36 = 0.138889
P(Sum = 7) = 6/36 = 0.166667
P(Sum = 8) = 5/36 = 0.138889
P(Sum = 9) = 4/36 = 0.111111
P(Sum = 10) = 3/36 = 0.0833333
P(Sum = 11) = 2/36 = 0.0555556
P(Sum = 12) = 1/36 = 0.0277778
```

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**:
        - 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}.

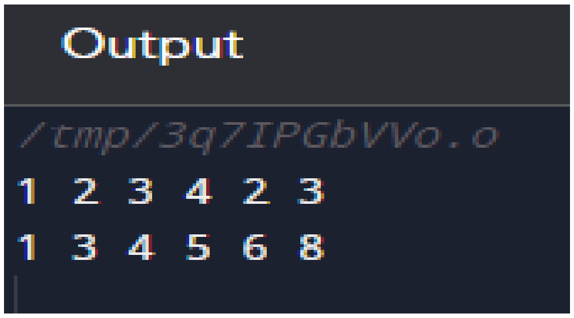
	1	2	2	3	3	4
1	2	3	3	4	4	5
3	4	5	5	6	6	7
4	5	6	6	7	7	8
5	6	7	7	8	8	9
6	7	8	8	9	9	10
8	9	10	10	11	11	12

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;
}
```



Output

```
/tmp/3q7IPGbVVo.o
1 2 3 4 2 3
1 3 4 5 6 8
```