Applications

Tai, Wei Hsuan

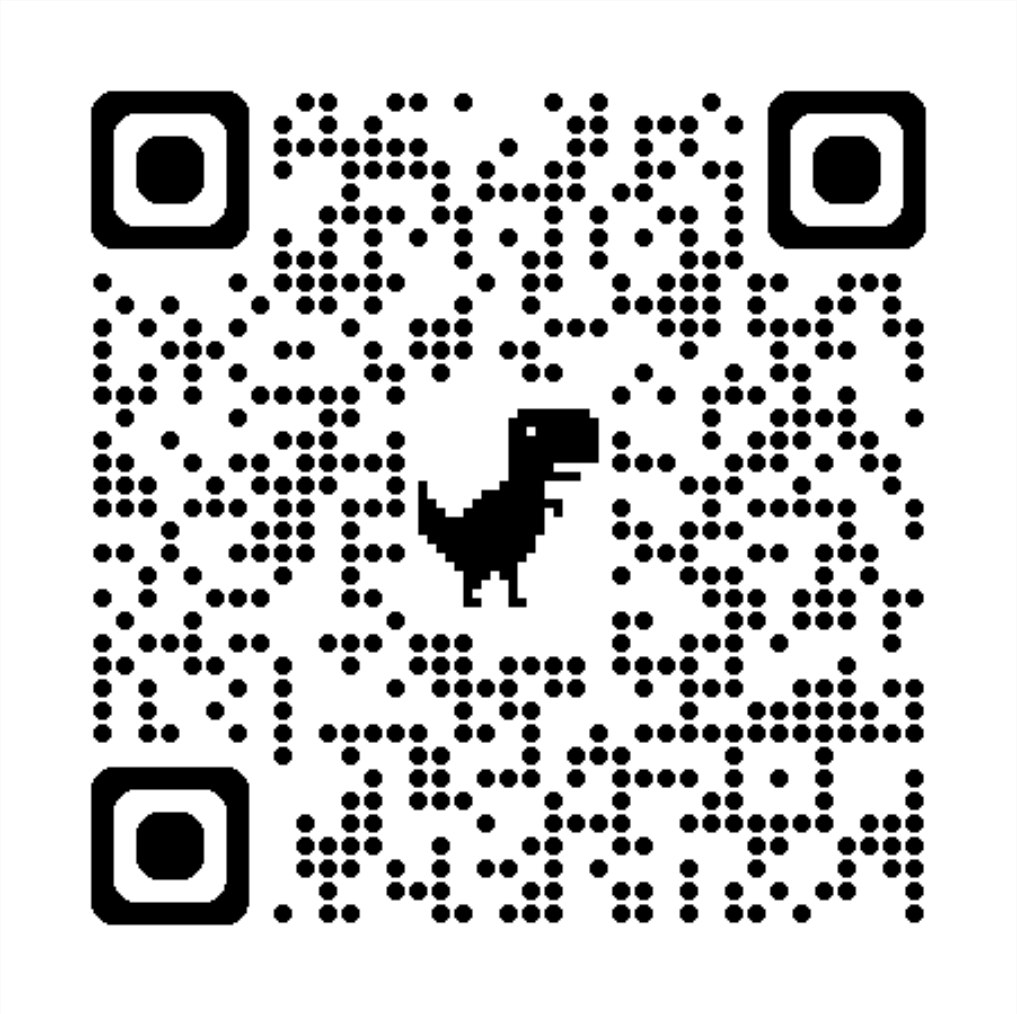
week 8

Announcement

Recording or photographing the slides or any content displayed on the screen is permitted for personal use only.

Redistribution, modification, or any commercial use of the materials is strictly prohibited.

Course Materials



Outline

Warn Up: Calculate Pi with Monte Carlo method

Outline

What is Pi?

Pi (π) is a mathematical constant that represents the ratio of a circle's circumference to its diameter. It is an irrational number, meaning it cannot be expressed as a simple fraction, and its decimal representation goes on infinitely without repeating. The value of π is approximately 3.14159.

If a circle has a radius of r, then its area would be

$$A = \pi r^2.$$

Monte Carlo method

Intuitively, if we drop random points into a square that encloses a quarter circle, the ratio of points that fall inside the quarter circle to the total number of points should approximate the ratio of the areas of the quarter circle to the square.

Outline

How to generate random numbers?

In C++, we can leverage the `<random>` library to generate random numbers. With `rand()` function, we can generate a random integer between 0 and `RAND_MAX`. We can use remainder operation to limit the range. However, there may be some issue with this function.

- Range is too small (i.e., `RAND_MAX` is at least 32767)
- Not thread-safe, people can estimate the next number.
- Not uniformly distributed.
- Remainder operation may introduce bias.

Mt19937

Mt19937 is a widely used pseudorandom number generator (PRNG) known for its high quality and long period. It is based on the Mersenne Twister algorithm, which was developed by Makoto Matsumoto and Takuji Nishimura in 1997. The ”19937” in its name refers to the fact that it has a period of $2^{19937} - 1$, meaning it can generate a vast sequence of random numbers before repeating.

Usage

```
random_device rd;  // Obtain a random number from hardware
mt19937 eng(rd()); // Seed the generator
uniform_int_distribution<int>(a,b) rand_int(a, b); // Define the range
uniform_real_distribution<double> rand_real(0.0, 1.0); // Define the range
double random_value = rand_real(eng); // Generate random number
```

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

random_device rd;
mt19937 g(rd());
uniform_int_distribution<int> dist(0, 1000000);

int main(){
    cout<<dist(g)<<"\n";
}
```

Outline

Estimation of Pi

Here, the task is:

- Generate N random points within a square of side length 1.
- Count how many points fall within the quarter circle of radius 1.
- Estimate the value of π using the ratio of points inside the quarter circle to the total number of points.

Generate N points within the Square

```
random_device rd;
mt19937 g(rd());
uniform_real_distribution<double> rand_real(0.0, 1.0);
int N=100000;
for(int i = 0; i < N; i++){
    double x = rand_real(g);
    double y = rand_real(g);
}
```

Count points within the Quarter Circle

```
int inside_circle=0;
for(int i=0;i<N;i++){
    double x=rand_real(g);
    double y=rand_real(g);
    if(x*x+y*y<=1.0) inside_circle++;
}
```

Estimate the value of Pi

The area of the quarter circle is $\pi(\frac{r}{2})^2 = \frac{\pi}{4}$ (since $r = 1$), and the area of the square is $1 \times 1 = 1$. Therefore, the ratio of the area of the quarter circle to the area of the square is $\frac{\pi}{4}$. Thus, we can estimate π as follows:

```
1 double pi_estimate=4.0*inside_circle/N;
2 cout<<"Estimated value of Pi: "<<pi_estimate<<"\n";
```

Finally, combining all parts together, we get the complete code for estimating π using the Monte Carlo method.

More

- Try different values of N (e.g., 1000, 10000, 100000, 1000000) and observe how the estimate of π improves with larger sample sizes.
- Can you think of any improvements or optimizations to the code?

Application: 1A2B

Outline

What is 1A2B?

1A2B is a number guessing game. The classic version is: the computer randomly generates a 4-digit number with no repeated digits, and the player tries to guess it. After each guess, the computer provides feedback in the form of ”A” and ”B”. A indicates the number of digits that are correct and in the correct position, while B indicates the number of correct digits but in the wrong position. The game continues until the player guesses the correct number (4A0B).

Outline

Game Process

1. Computer generates answer.
2. Player makes a guess.
3. Computer provides feedback in the form of ”A” and ”B”.
4. Repeat steps 2 and 3 until the player guesses the correct number.

Function Analysis

To implement the game, we need to define several functions:

- Generate Answer
- Check Input Validity
- Calculate A and B

Outline

Random Function

The most important part of the game is to generate a random n-digit number with no repeated digits. (n is between 2 and 9)

To generate n unique digits, generating them one by one is obvously not a good idea. Instead, we can use an array to store all digits from 0 to 9, shuffle them randomly, and then take the first n digits as the answer.

Shuffle Function

```
1  string num="123456789";
2  shuffle(num.begin(), num.end(), g);
3
4  string res="";
5  for(int i=0; i<n; i++) res+=num[i];
```

With the above code, we can shuffle the whole string of the array `num`. After shuffling, we can take the first n elements as our answer. Remember to check if n is between 2 and 9.

Check Input Validity

To ensure the player's guess is valid, we need to check three conditions:

- The input length must be equal to n.
- The input must only contain digits (1-9).
- All digits in the input must be unique.

Calculate A and B

To calculate the number of A's and B's:

- For A's: Compare each digit of the player's guess with the corresponding digit of the answer. If they match, increment the A count.
- For B's: For each digit in the player's guess, check if it exists in the answer but in a different position. If it does, increment the B count.