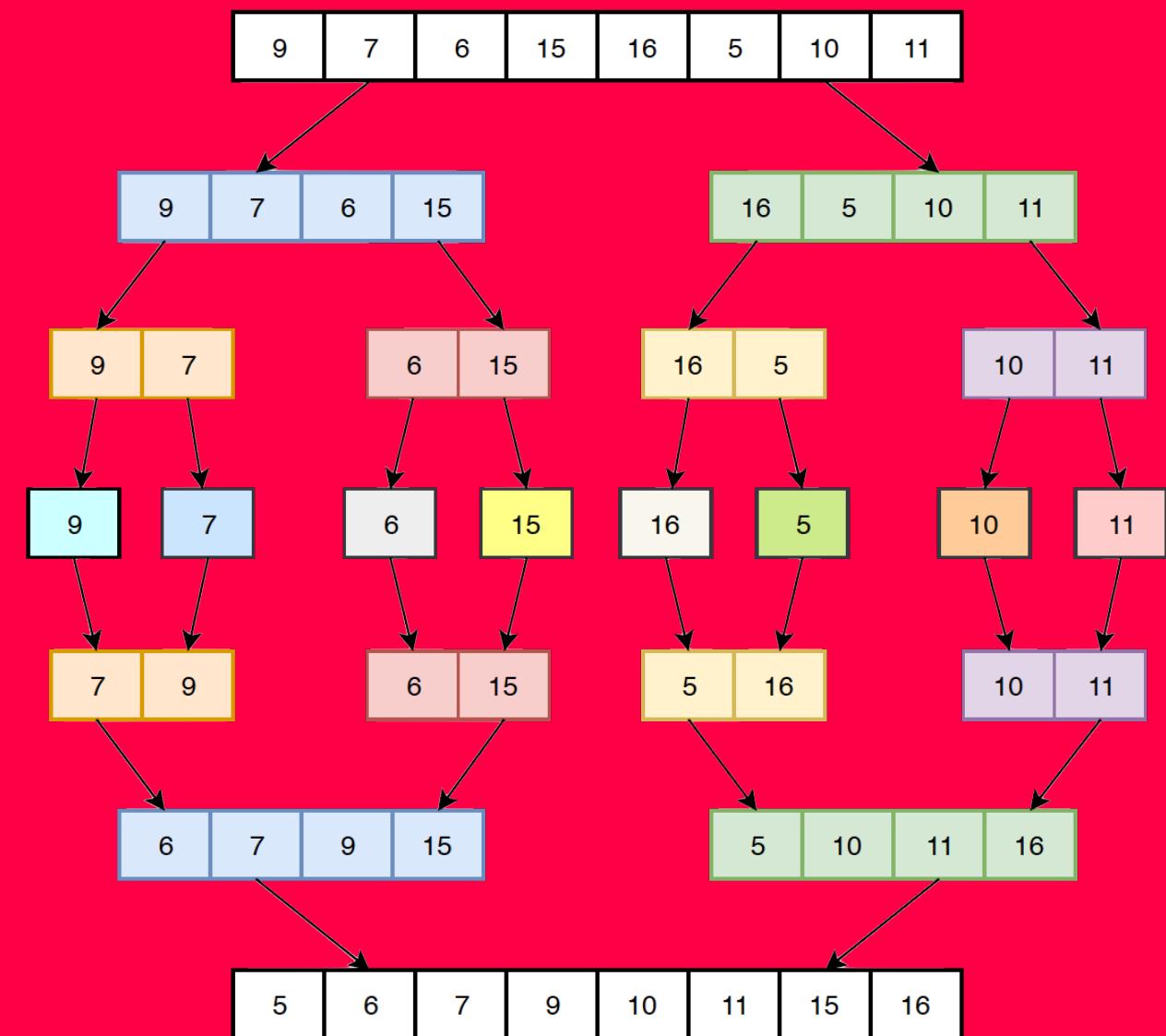


الگوريتم:

تقسيم و غلبه



(بررسی مرتب سازی ادغامی، برج هانوی و اعداد فیبوناچی)



یکی از روش‌های پرکاربرد و محبوب برای طراحی الگوریتم‌ها روش تقسیم و حل یا **Divide and Conquer** است.

در این روش، داده‌ها به دو یا چند دسته تقسیم شده و حل می‌شوند. سپس با ترکیب مناسب نتایج به دست آمده از این زیرمسئله‌ها، مسئله‌ی اصلی حل می‌شود. در صورتی که زیرمسئله خود به اندازه‌ی کافی بزرگ باشد، می‌توان از همین روش برای حل آن استفاده کرد.

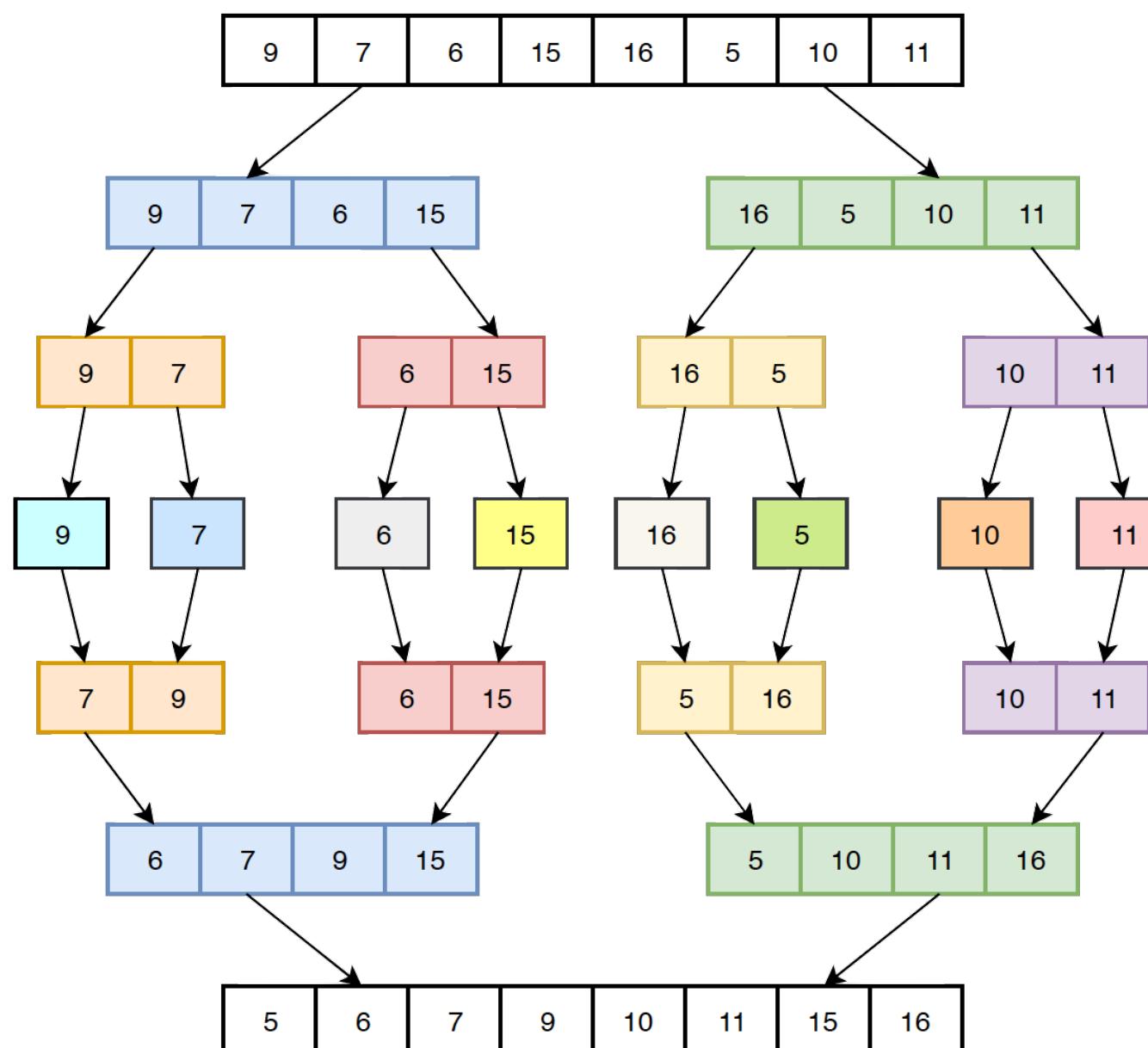
در ادامه پست مرتب سازی ادغامی، برج هانوی و اعداد فیبوناچی که با روش **تقسیم و غلبه** قابل حل هستند را بررسی می‌کنیم.





مرتب‌سازی ادغامی

مرتب‌سازی ادغامی نوعی از الگوریتم‌های مرتب‌سازی (sort) است.
طرز کار این الگوریتم تقریباً به صورت زیر است:
توالی n عدد را به دو نیمه تقسیم می‌کنیم.
به طور بازگشتی دو نیمه را مرتب می‌کنیم.
دو نیمه مرتب شده را در یک توالی مرتب منفرد ادغام می‌کنیم.



هر سطح به زمانی برابر با $O(n)$ نیاز دارد، زیرا تعداد کل اعداد صحیحی در هر سطح برابر با n است.

در کل $O(n \log n)$ سطح وجود دارد و از این رو زمان کلی برای اجرای این الگوریتم برابر با $O(n \log n)$ است.

برای دانلود فایل کامل کدها به لینک

گیت هاب

۵۶ در رهایلات درج شده مراجعه کنید.

C++



```
// Merge_Sort
//
// Created by Shayan Aryania
//
#include <iostream>
using namespace std;

// Merges two subarrays of array[].
// First subarray is arr[begin..mid]
// Second subarray is arr[mid+1..end]
void merge(int array[], int const left, int const mid, int const right)
{
    auto const subArrayOne = mid - left + 1;
    auto const subArrayTwo = right - mid;

    // Create temp arrays
    auto *leftArray = new int[subArrayOne],
        *rightArray = new int[subArrayTwo];

    // Copy data to temp arrays leftArray[] and rightArray[]
    for (auto i = 0; i < subArrayOne; i++)
        leftArray[i] = array[left + i];
    for (auto j = 0; j < subArrayTwo; j++)
        rightArray[j] = array[mid + 1 + j];

    auto indexOfSubArrayOne = 0, // Initial index of first sub-array
        indexOfSubArrayTwo = 0; // Initial index of second sub-array
    int indexOfMergedArray = left; // Initial index of merged array

    // Merge the temp arrays back into array[left..right]
    while (indexOfSubArrayOne < subArrayOne && indexOfSubArrayTwo < subArrayTwo) {
        if (leftArray[indexOfSubArrayOne] <= rightArray[indexOfSubArrayTwo]) {
            array[indexOfMergedArray] = leftArray[indexOfSubArrayOne];
            indexOfSubArrayOne++;
        }
        else {
            array[indexOfMergedArray] = rightArray[indexOfSubArrayTwo];
            indexOfSubArrayTwo++;
        }
        indexOfMergedArray++;
    }

    // Copy remaining elements of leftArray, if any
    while (indexOfSubArrayOne < subArrayOne) {
        array[indexOfMergedArray] = leftArray[indexOfSubArrayOne];
        indexOfSubArrayOne++;
        indexOfMergedArray++;
    }

    // Copy remaining elements of rightArray, if any
    while (indexOfSubArrayTwo < subArrayTwo) {
        array[indexOfMergedArray] = rightArray[indexOfSubArrayTwo];
        indexOfSubArrayTwo++;
        indexOfMergedArray++;
    }

    delete[] leftArray;
    delete[] rightArray;
}

Given array is
12 11 13 5 6 7
Sorted array is
5 6 7 11 12 13 Program ended with exit code: 0
```

Python



```
#Shayan Aryania
# Python program for implementation of MergeSort
def mergeSort(arr):
    if len(arr) > 1:
        # Finding the mid of the array
        mid = len(arr)//2

        # Dividing the array elements
        L = arr[:mid]

        # into 2 halves
        R = arr[mid:]

        # Sorting the first half
        mergeSort(L)

        # Sorting the second half
        mergeSort(R)

        i = j = k = 0

        # Copy data to temp arrays L[] and R[]
        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                arr[k] = L[i]
                i += 1
            else:
                arr[k] = R[j]
                j += 1
            k += 1

        # Checking if any element was left
        while i < len(L):
            arr[k] = L[i]
            i += 1
            k += 1

        while j < len(R):
            arr[k] = R[j]
            j += 1
            k += 1

        # Code to print the list
    def printList(arr):
        for i in range(len(arr)):
            print(arr[i], end=" ")
        print()

printList(arr)
```

000+000>>



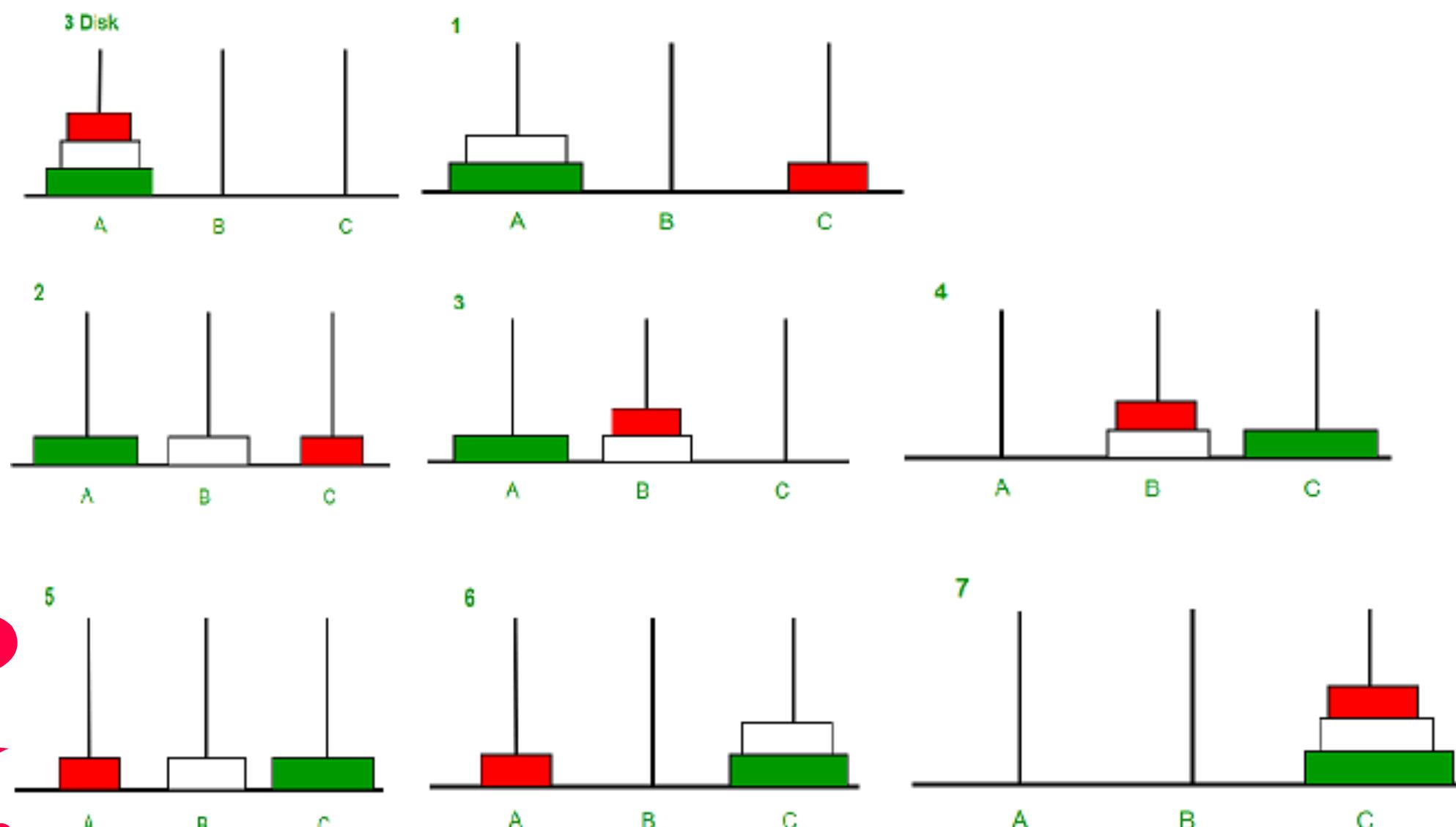
برج‌های هانوی

برج‌های هانوی یک مسئله ریاضیاتی است که شامل ۳ میله و چند دیسک است.
(ما در مثال خود از ۳ دیسک استفاده می‌کنیم)

هر دیسک اندازه متفاوتی دارد. ما می‌خواهیم همه دیسک‌ها را به میله C انتقال دهیم.
(به طوری که بزرگ‌ترین دیسک در پایین، دیسک متوسط در میانه و دیسک کوچک‌تر در سطح بالایی قرار بگیرد)

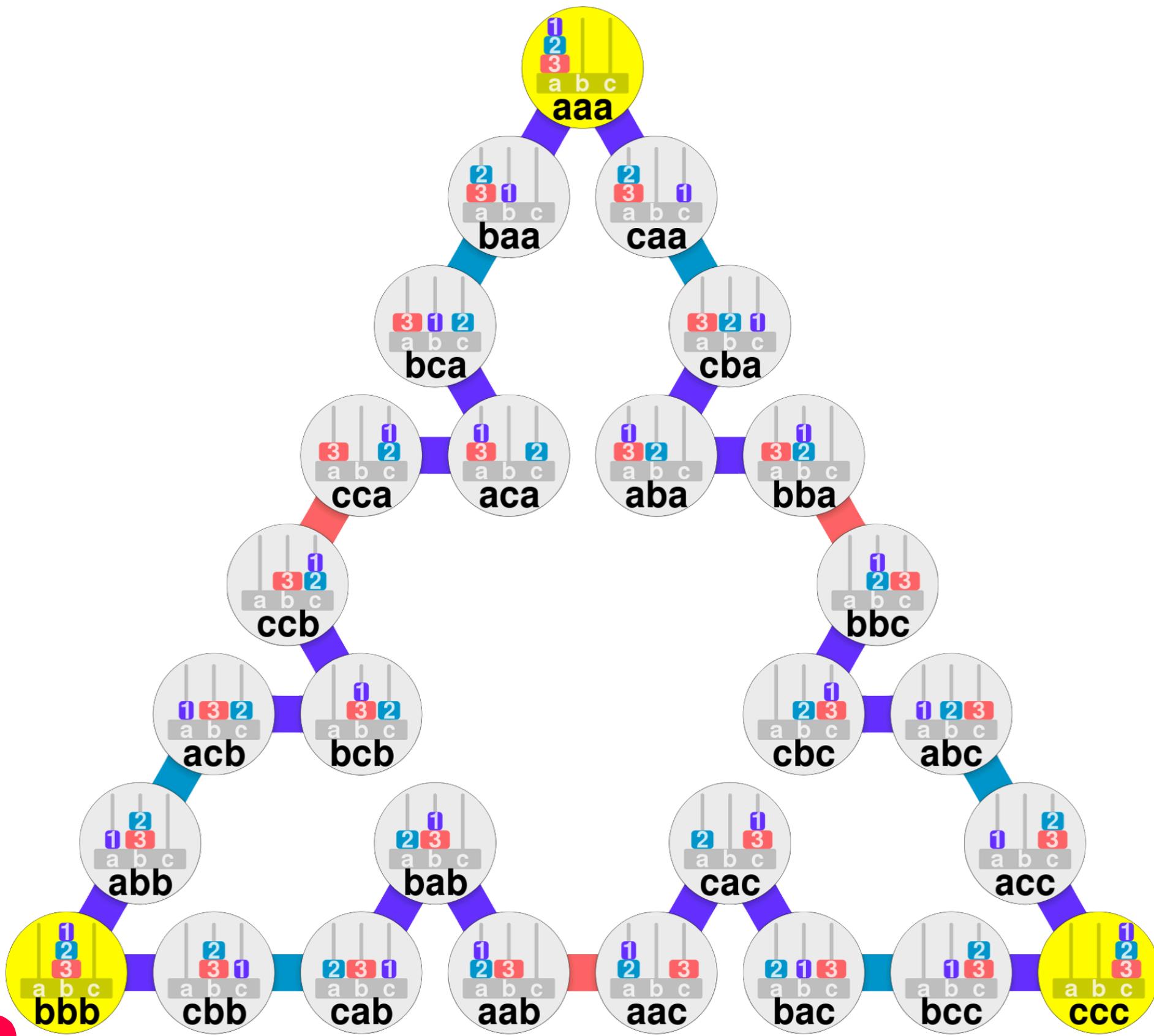
ما هر بار تنها 1 دیسک را می‌توانیم جا به جا کنیم.

یک دیسک نمی‌تواند روی دیسک‌هایی که کوچک‌تر از خود است قرار بگیرد.





اگر ۱ دیسک داشته باشیم، کافی است آن را به میله C انتقال دهیم.
اگر ۲ دیسک داشته باشیم، این مسئله با 3^2 حرکت حل می شود.
اگر ۳ دیسک داشته باشیم، این مسئله با 8^2 حرکت حل می شود.
اگر ۴ دیسک داشته باشیم، این مسئله با 15^2 حرکت حل می شود.
در مورد ۵ دیسک این عدد به 31^2 می رسد.



برای حل هر مسئله برج های هانوی با تعداد دیسک n ، به $(n-1)^2$ حرکت نیاز داریم.

C++



```
32 void move(vector<int>& from_peg, vector<int>& to_peg)
33 {
34     to_peg.push_back(from_peg.back());
35     from_peg.pop_back();
36     print_pegs();
37 }
38
39 void hanoi(vector<int>& from, vector<int>& to,
40             vector<int>& _using, int num_of_discs)
41 {
42     if (num_of_discs == 1)
43         move(from, to);
44     else {
45         hanoi(from, _using, to, num_of_discs - 1);
46         move(from, to);
47         hanoi(_using, to, from, num_of_discs - 1);
48     }
49 }
50
51 int main()
52 {
53     int num_of_discs;
54     cout << "How many discs? ";
55     cin >> num_of_discs;
56
57     for (int i = num_of_discs; i >= 1; i--)
58         a.push_back(i);
59
60     print_pegs();
61     hanoi(a, b, c, num_of_discs);
62 }
```

Python



```
Tower _Of _Hanoi.py
```

```
1 # Recursive Python function to solve the tower of hanoi
2
3 def TowerOfHanoi(n , source, destination, auxiliary):
4     if n==1:
5         print ("Move disk 1 from source",source,"to destination",destination)
6         return
7     TowerOfHanoi(n-1, source, auxiliary, destination)
8     print ("Move disk",n,"from source",source,"to destination",destination)
9     TowerOfHanoi(n-1, auxiliary, destination, source)
10
11
12 n = 5
13 TowerOfHanoi(n,'A','B','C')
```

Ln 1, Col 1 Tab Size: 4 UTF-8 LF Python 3.10.1 64-bit ⚡ 🔔

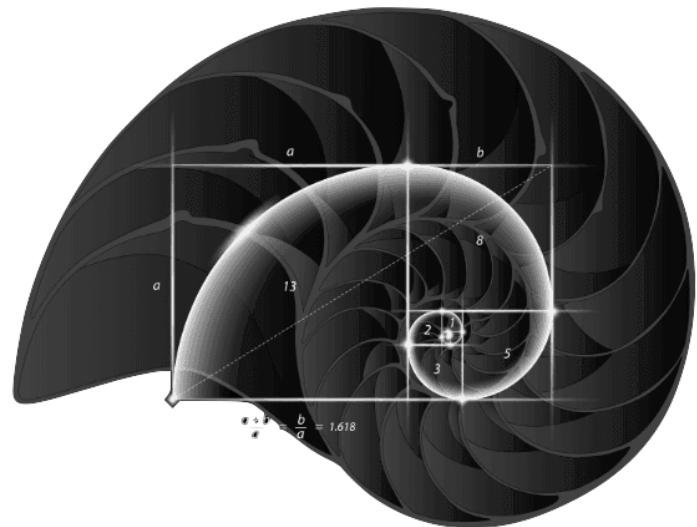
برای دانلود فایل کامل کدها به لینک
گیت هاب
که در هایلایت درج شده مراجعه کنید.

0000+00>>



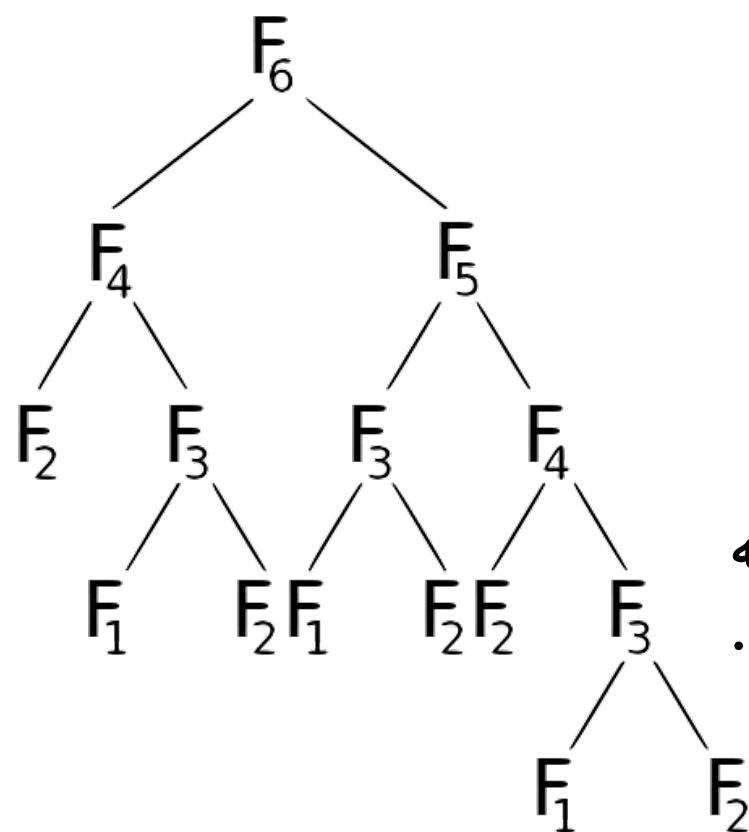
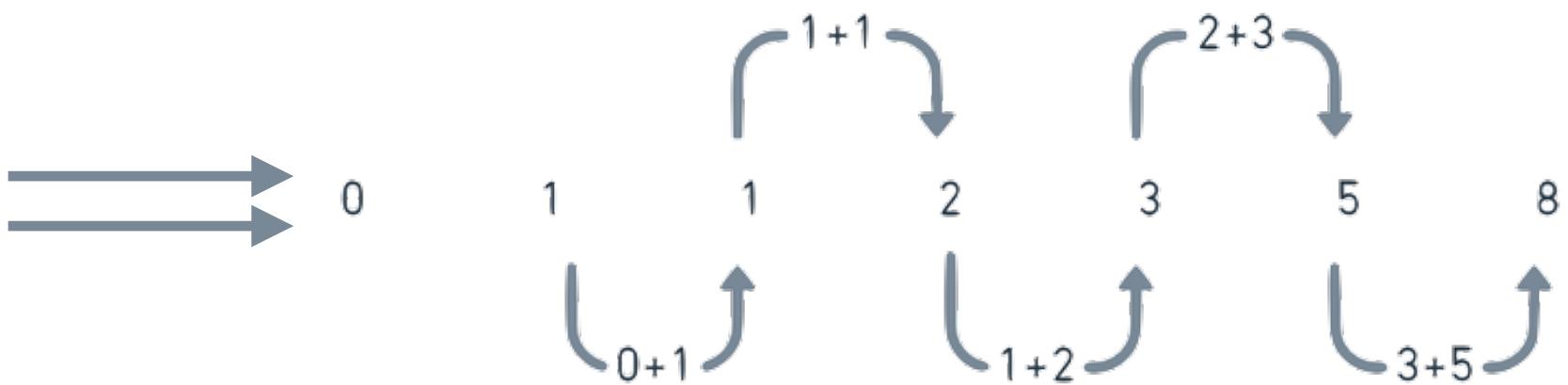
اعداد فیبوناچی

اعداد فیبوناچی را می‌توان در طبیعت مشاهده کرد.



این اعداد از ۱ آغاز می‌شوند و عدد بعدی برابر با عدد کنونی + عدد قبلی است.

$$\begin{cases} F(0) = 0 \\ F(1) = 1 \\ F(N) = F(N-1) + F(N-2) \end{cases}$$



در مثال رو به رو N برابر با 6 است.
از آنها که N بزرگ تر از 0 یا 1 است، آن بخش را نادیده می‌گیریم.
سپس $F(5) + F(4)$ را محاسبه می‌کنیم.
 $F(5)$ به صورت جمع $F(4) + F(2)$ محاسبه می‌شود.

برابر با 0 یا 1 باشد، اعداد 1 را F در نهایت وقتی به حالت‌های پایه بررسیم که خواهیم داشت. در هر مرحله عدد قبلی را به مرحله بالاتر بازگشت می‌دهیم.

0000+00>>

C++



```
1 // Fibonacci Series
2 //
3 // Created by Shayan Aryania
4
5
6 #include <iostream>
7 using namespace std;
8
9 int fib(int n)
10 {
11     if (n <= 1)
12         return n;
13     return fib(n-1) + fib(n-2);
14 }
15
16 int main ()
17 {
18     int n = 9;
19     cout << fib(n);
20     getchar();
21     return 0;
22 }
```

Line: 22 Col: 1

Python



```
1 def fib(n):
2     if n == 1:
3         return 1
4     elif n == 0:
5         return 0
6     else:
7         return fib(n-1) + fib(n-2)
8
9 r = int(input("Enter number: "))
10 print(fib(r))
11
12
```

Ln 1, Col 1 Spaces: 3 UTF-8 LF Python 3.10.1 64-bit

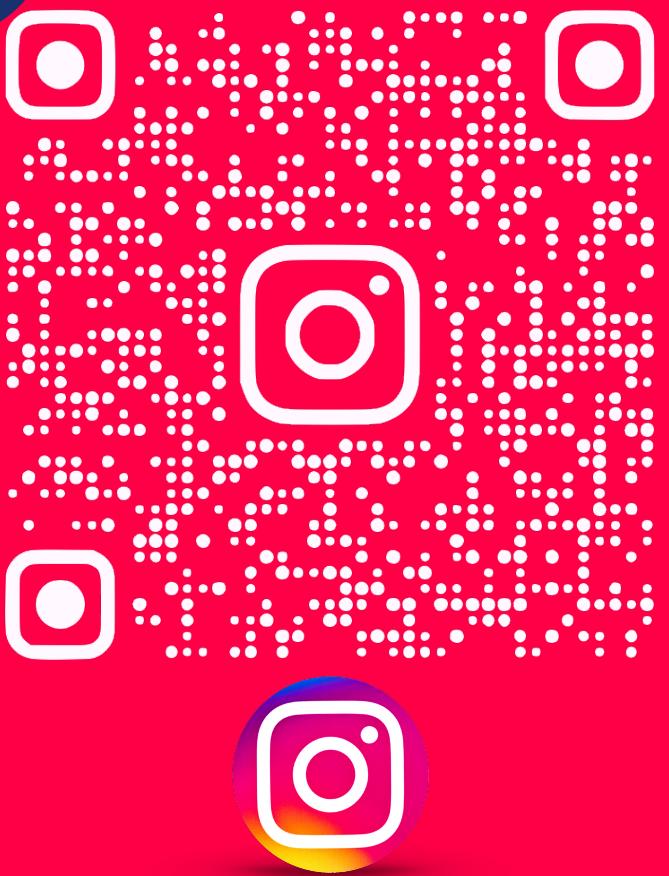
@ShayanAryania

برای دانلود فایل کامل کدها به لینک
کیت هاب ۵۶ در راهیلاست درج شده مراجعه کنید.

00000to>>



Follow



www.instagram.com/shayanaryania



www.linkedin.com/in/shayanaryania



Like & Share

Save