LECTURE 10

DYNAMIC PROGRAMMING – PART I

STAIRCASE + COIN CHANGE PROBLEM

Big-O Coding

Website: www.bigocoding.com

Quy hoạch động là gì?

Dynamic Programming (Quy hoạch động): là một kĩ thuật thiết kế thuật toán theo kiểu chia bài toán lớn thành các bài toán con, sử dụng lời giải của các bài toán con để tìm lời giải cho bài toán ban đầu.

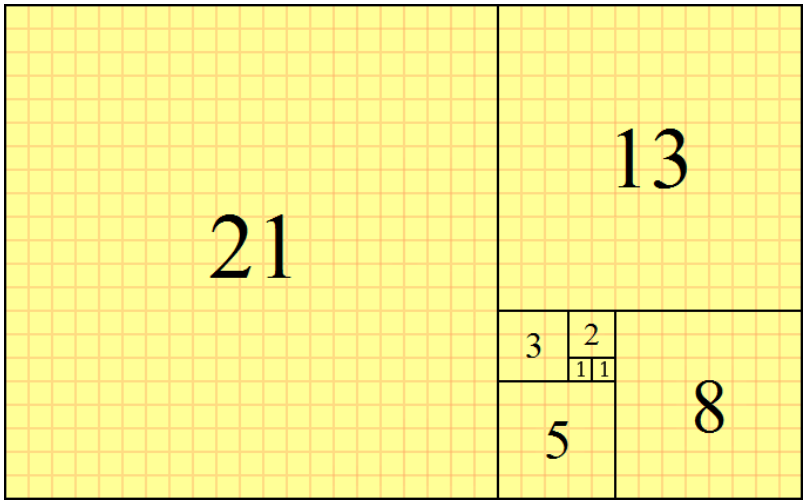
Các lời giải của quy hoạch động sẽ **lưu vào bộ nhớ** (thường là mảng 1 chiều hoặc 2 chiều), sau đó lấy lời giải của bài toán con ở trong mảng đã tính trước đó để giải bài toán lớn. Việc lưu lại lời giải vào bộ nhớ giúp cho ta không phải tính lại lời giải của các bài toán con mỗi khi cần, do đó tiết kiệm được thời gian tính toán.

Ví dụ minh họa

Viết hàm tính phần tử thứ n của dãy Fibonacci. Biết dãy Fibonacci có công thức truy hồi như sau:

$$F(n) := \begin{cases} 0, khi\ n = 0 \\ 1, khi\ n = 1 \\ F(n-1) + F(n-2), khi\ n > 1 \end{cases}$$

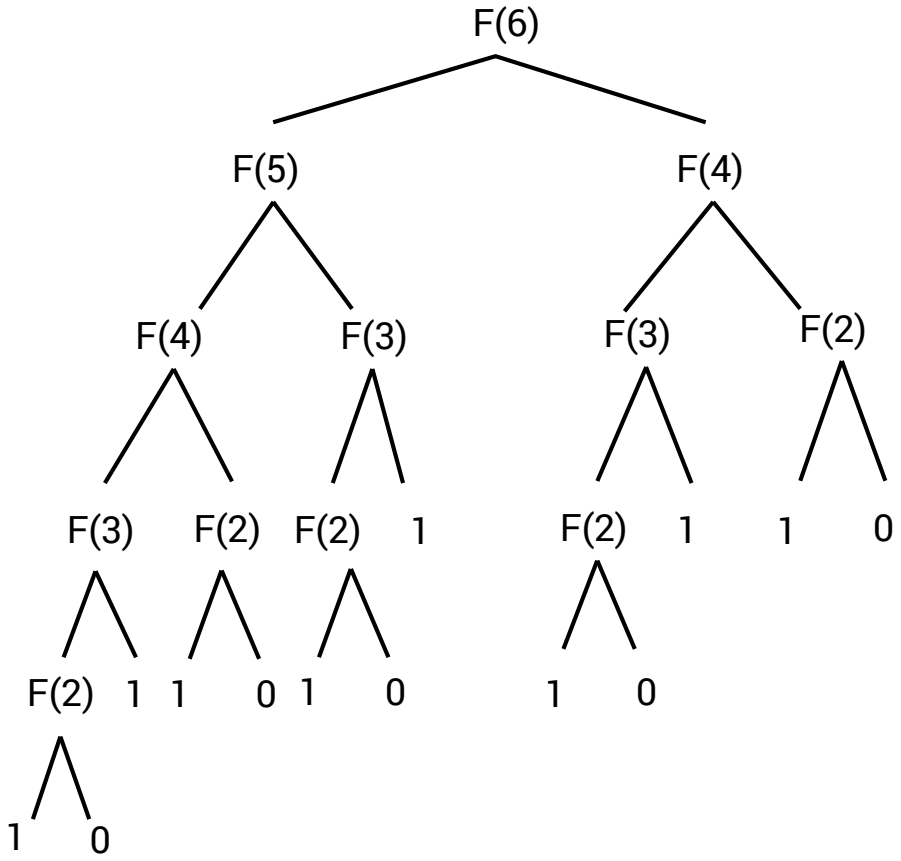
Ví dụ: Dãy Fibonacci có dạng 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...



Sự khác nhau Recursion và Dynamic programming

Recursion

Phân tích bài toán thành các bài toán con, tạo thành cây đệ quy để giải bài toán lớn.



Time Complexity: $O(2^n)$

Dynamic programming

Phân tích bài toán thành bài toán con, giải bài toán con này **lưu vào mảng**. Tổng hợp kết quả lại để giải bài toán lớn.

- Top-Down Approach.
- Bottom-Up Approach.

F(0)	F(1)	F(2)	F(3)	F(4)	F(5)	F(6)
0	1	1	2	3	5	8

Time Complexity: $O(n)$

Source Code Fibonacci recursion

```cpp
#include <iostream>
using namespace std;
int fibonacci(int n)
{
    if (n <= 1)
        return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
int main()
{
    int n = 6;
    cout << fibonacci(n);
    return 0;
}
```

Source Code Fibonacci recursion

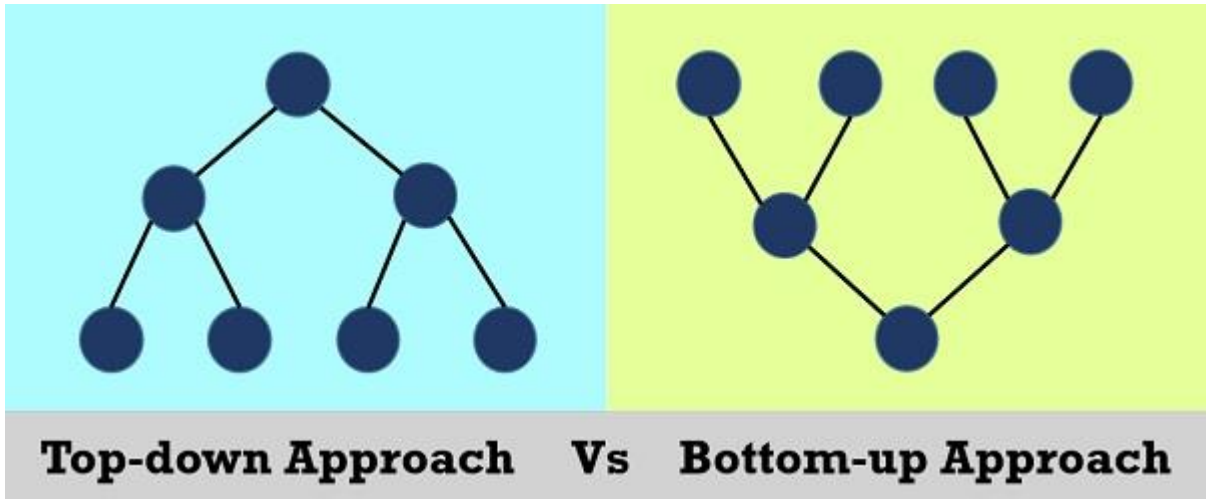
```
1. def fibonacci(n):
2.     if n <= 1:
3.         return n
4.     return fibonacci(n - 1) + fibonacci(n - 2)
5. if __name__ == '__main__':
6.     n = 6
7.     print(fibonacci(n))
```

Source Code Fibonacci recursion

```java
public class Main {
    private static int fibonacci(int n) {
        if (n <= 1) {
            return n;
        }
        return fibonacci(n - 1) + fibonacci(n - 2);
    }
    public static void main(String[] args) {
        int n = 6;
        System.out.println(fibonacci(n));
    }
}
```

So sánh Top-Down và Bottom-Up

- **Top-Down Approach (Memoization):** Sử dụng đệ quy để tìm kết quả các bài toán con, sau đó lưu vào mảng kết quả để tìm đáp án của bài toán lớn ban đầu. Khi gặp lại bài toán con sẽ trả về kết quả mà không cần phải tính toán lại lần nữa.
- **Bottom-Up Approach (Tabulation)**: phương pháp này sẽ không sử dụng đệ quy, dùng vòng lặp để giải lần lượt các bài toán từ nhỏ đến lớn.



Top-Down Approach

Khởi tạo mảng kết quả ban đầu **result:**

0	1	2	3	4	5	6
0	1	0	0	0	0	0

Xét các điều kiện sau để trả về kết quả:

- Nếu n ≤ 1 → return n.
- Nếu result[n] != 0 → return result[n].
- Ngược lại: **result[n] =** fibonacci(n - 1) + fibonacci(n - 2) → return result[n]

Bước 1: Giảm n đến 0 và 1, chạy n = 0 và n = 1

Giá trị n giảm từ 6 đến 0, với n = 0 và n = 1:

- Xét n = 0 (do n ≤ 1) → return 0.
- Xét n = 1 (do n ≤ 1) → return 1.

0	**1**	2	3	4	5	6
0	**1**	0	0	0	0	0

Bước 2: Chạy thuật toán n = 2

Với n = 2:

- Xét n = 1 (do n ≤ 1) → return 1.
- Xét n = 0 (do n ≤ 1) → return 0.

➔ **result[2] =** fibonacci(n - 1) + fibonacci(n - 2) = 1 + 0 = 1.

0	1	**2**	3	4	5	6
0	1	**1**	0	0	0	0

Bước 3: Chạy thuật toán n = 3

Với n = 3:

- Xét n = 2 (do result[2] != 0) → return 1.
- Xét n = 1 (do n ≤ 1) → return 1.

➔ **result[3] =** fibonacci(n - 1) + fibonacci(n - 2) = 1 + 1 = 2.

0	1	2	**3**	4	5	6
0	1	1	**2**	0	0	0

Bước 4: Chạy thuật toán n = 4

Với n = 4:

- Xét n = 3 (do result[3] != 0) → return 2.
- Xét n = 2 (do result[2] != 0) → return 1.

➔ **result[4] =** fibonacci(n - 1) + fibonacci(n - 2) = 2 + 1 = 3.

0	1	2	3	**4**	5	6
0	1	1	2	**3**	0	0

Bước 5: Chạy thuật toán n = 5

Với n = 5:

- Xét n = 4 (do result[4] != 0) → return 3.
- Xét n = 3 (do result[3] != 0) → return 2.

➔ **result[5] =** fibonacci(n - 1) + fibonacci(n - 2) = 3 + 2 = 5.

0	1	2	3	4	**5**	6
0	1	1	2	3	**5**	0

Bước 6: Chạy thuật toán n = 6

Với n = 6:

- Xét n = 5 (do result[5] != 0) → return 5.
- Xét n = 4 (do result[4] != 0) → return 3.

➔ **result[6] =** fibonacci(n - 1) + fibonacci(n - 2) = 5 + 3 = 8.

0	1	2	3	4	5	**6**
0	1	1	2	3	5	**8**

Kết quả result[6] = 8.

Source Code Top-Down

```cpp
#include <iostream>
#include <vector>
using namespace std;
vector<int> result;
int fibonacci(int n)
{
    if (n <= 1)
        return n;
    if (result[n] != 0)
        return result[n];
    else
    {
        result[n] = fibonacci(n - 1) + fibonacci(n - 2);
        return result[n];
    }
}
```

Source Code Top-Down

```
1.  int main()
2.  {
3.      int n = 6;
4.      result.resize(n + 1);
5.      result[0] = 0;
6.      result[1] = 1;
7.      cout << fibonacci(n) << endl;
8.      return 0;
9.  }
```

Source Code Top-Down

```
1.  def fibonacci(n):
2.      if n <= 1:
3.          return n
4.      if result[n] != 0:
5.          return result[n]
6.      else:
7.          result[n] = fibonacci(n - 1) + fibonacci(n - 2)
8.          return result[n]
9.
10. if __name__ == "__main__":
11.     n = 6
12.     result = [0] * (n + 1)
13.     result[0] = 0
14.     result[1] = 1
15.     print(fibonacci(n))
```

Source Code Top-Down

```java
public class Main {
    private static int[] result;
    private static int fibonacci(int n) {
        if (n <= 1)
            return n;
        if (result[n] != 0)
            return result[n];
        else {
            result[n] = fibonacci(n - 1) + fibonacci(n - 2);
            return result[n];
        }
    }
    public static void main(String[] args) {
        int n = 6;
        result = new int[n + 1];
        result[0] = 0;
        result[1] = 1;
        System.out.println(fibonacci(n));
    }
}
```

Bottom-Up Approach

Khởi tạo mảng kết quả ban đầu **result:**

- result[0] = 0.
- result[1] = 1.
- Các giá trị còn lại của mảng bằng 0.

0	1	2	3	4	5	6
0	1	0	0	0	0	0

Duyệt i = 2 đến i = n:

- **Kết quả quả bước hiện tại là kết quả của 2 bước trước đó.**

➔ result[i] = result[i – 1] + result[i – 2]

Bước 1: Chạy thuật toán với i = 2

Với i = 2:

- result[i] = result[i – 1] + result[i – 2] = 1 + 0 = 1.

➔ **result[2] = 1.**

0	1	2	3	4	5	6
0	1	1	0	0	0	0

Bước 2: Chạy thuật toán với i = 3

Với i = 3:

- result[i] = result[i – 1] + result[i – 2] = 1 + 1 = 2.

➔ **result[3] = 2.**

0	1	2	3	4	5	6
0	1	1	2	0	0	0

Bước 3: Chạy thuật toán với i = 4

Với i = 4:

- result[i] = result[i – 1] + result[i – 2] = 2 + 1 = 3.

➔ **result[4] = 3.**

0	1	2	3	4	5	6
0	1	1	2	3	0	0

Bước 4: Chạy thuật toán với i = 5

Với i = 5:

- result[i] = result[i – 1] + result[i – 2] = 3 + 2 = 5.

➔ result[5] = 5.

0	1	2	3	4	5	6
0	1	1	2	3	5	0

Bước 5: Chạy thuật toán i = 6

Với i = 6:

- result[i] = result[i – 1] + result[i – 2] = 5 + 3 = 8.

➔ **result[6] = 8**

0	1	2	3	4	5	6
0	1	1	2	3	5	8

Kết quả result[6] = 8.

Source Code Bottom-Up


```cpp
#include <iostream>
#include <vector>
using namespace std;
vector<int> result;
int fibonacci(int n)
{
    result[0] = 0;
    result[1] = 1;
    for (int i = 2; i <= n; i++)
        result[i] = result[i - 1] + result[i - 2];
    return result[n];
}
int main()
{
    int n = 6;
    result.resize(n + 1);
    cout << fibonacci(n) << endl;
    return 0;
}
```

Source Code Bottom-Up

```
1.  def fibonacci(n):
2.      result[0] = 0
3.      result[1] = 1
4.      for i in range(2, n + 1):
5.          result[i] = result[i - 1] + result[i - 2]
6.      return result[n]
7.
8.  if __name__ == "__main__":
9.      n = 6
10.     result = [0] * (n + 1)
11.     print(fibonacci(n))
```

Source Code Bottom-Up

```java
public class Main {
    private static int[] result;

    private static int fibonacci(int n) {
        result[0] = 0;
        result[1] = 1;
        for (int i = 2; i <= n; i++)
            result[i] = result[i - 1] + result[i - 2];
        return result[n];
    }

    public static void main(String[] args) {
        int n = 6;
        result = new int[n + 1];
        System.out.println(fibonacci(n));
    }
}
```

Bài toán minh họa 1

Staircase Problem (Bài toán leo cầu thang): Có một cầu thang có N bậc (N là số nguyên dương). Tại mỗi bước bạn có thể chọn bước lên 1 hoặc 2 bước. Hỏi có bao nhiêu cách để bạn có thể đi hết được cầu thang với vị trí xuất phát là ở mặt đất?

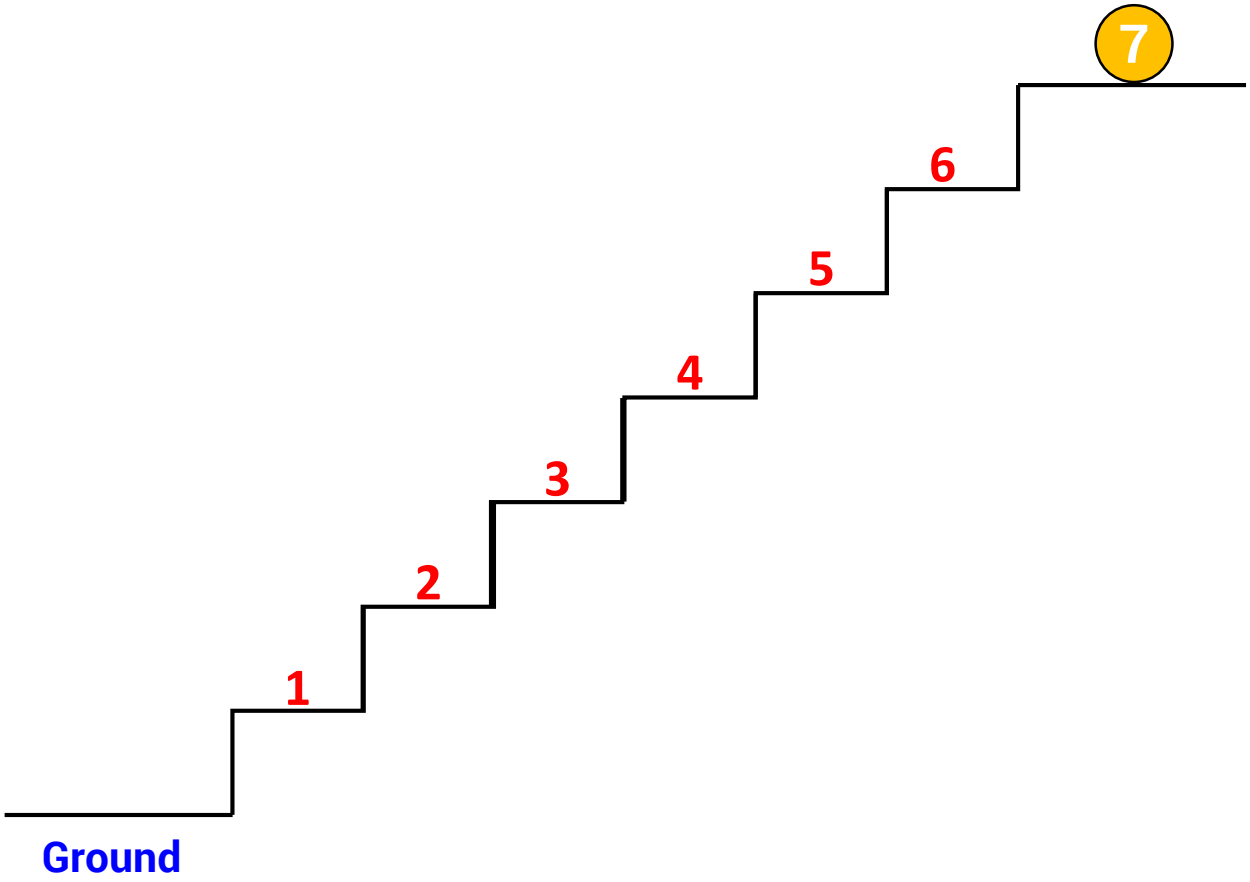
Ví dụ 1: Cho bạn số bậc thang N = 4, bạn có các cách đi như sau để đi từ mặt đất đến bậc cuối cầu thang:

- { 1, 1, 1, 1 }
- { 1, 1, 2 }
- { 1, 2, 1 }
- { 2, 1, 1 }
- { 2, 2 }

➔ Tổng cộng 5 cách đi.

Bài toán minh họa 1

Bây giờ sẽ chạy tay với dữ liệu lớn hơn để hiểu rõ giải bài toán trên, cho cầu thang có 7 bậc như hình vẽ:

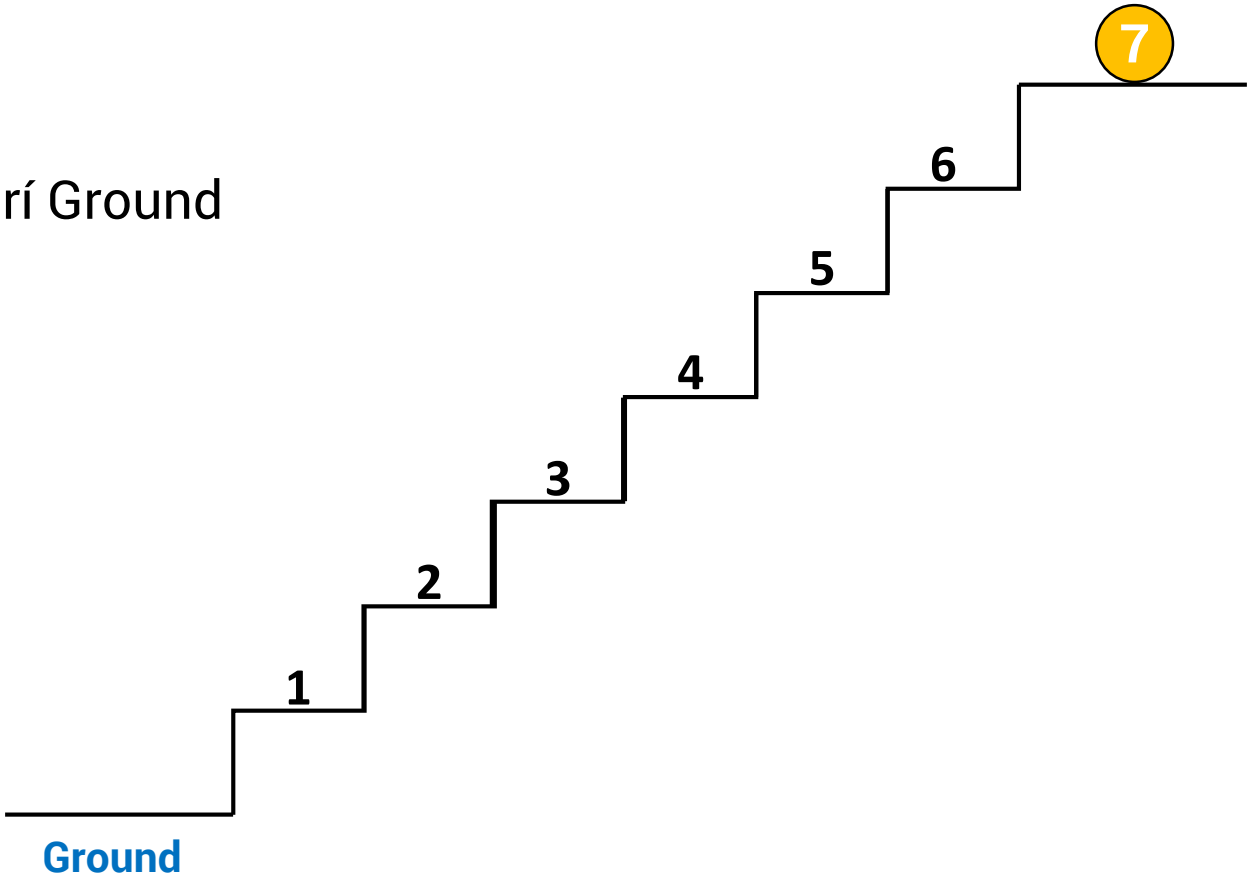


Bước 0: đang ở trước cầu thang

Với i = 0:

- { }

➔ Có 1 cách để đến vị trí Ground

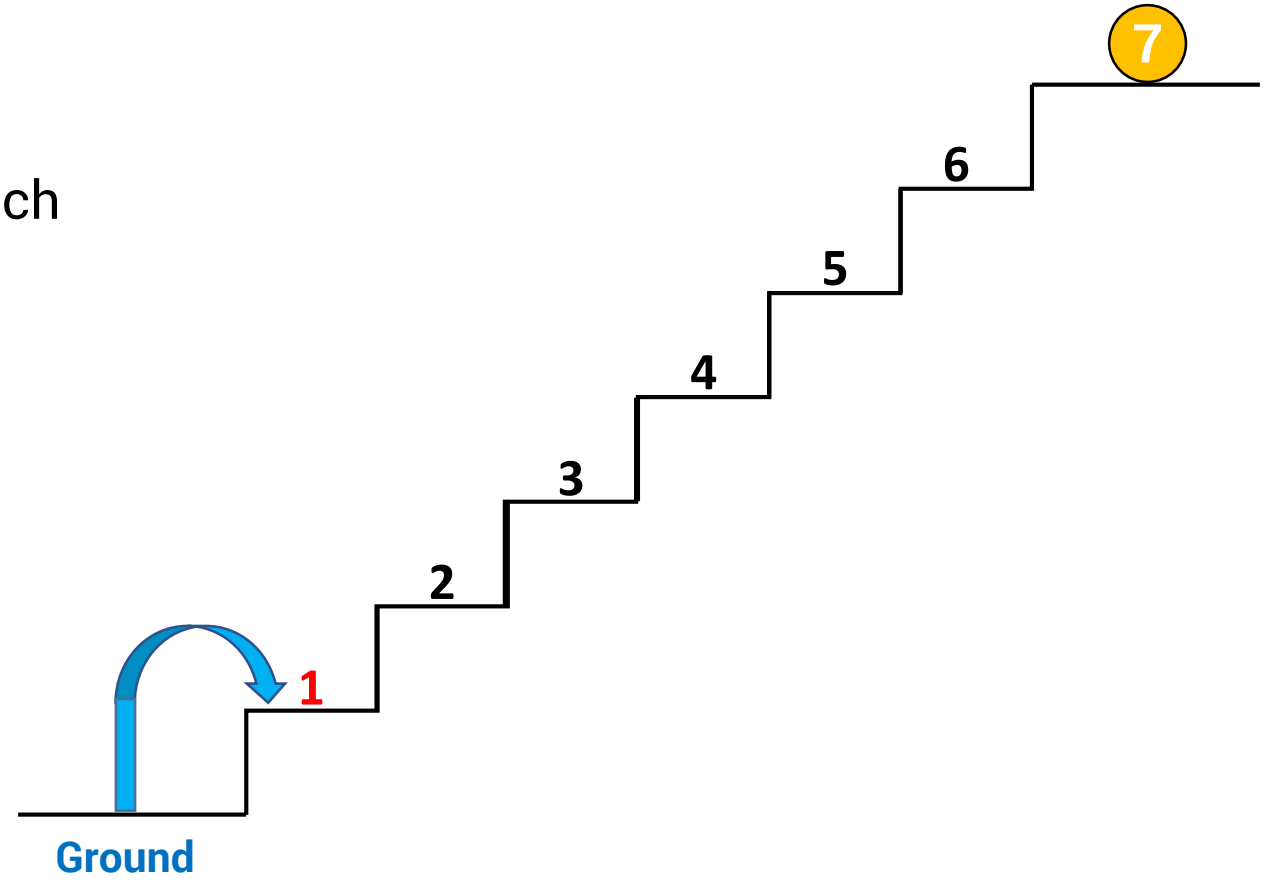


Bước 1: bậc thang chỉ có 1 bậc

Với i = 1:

- { 1 }

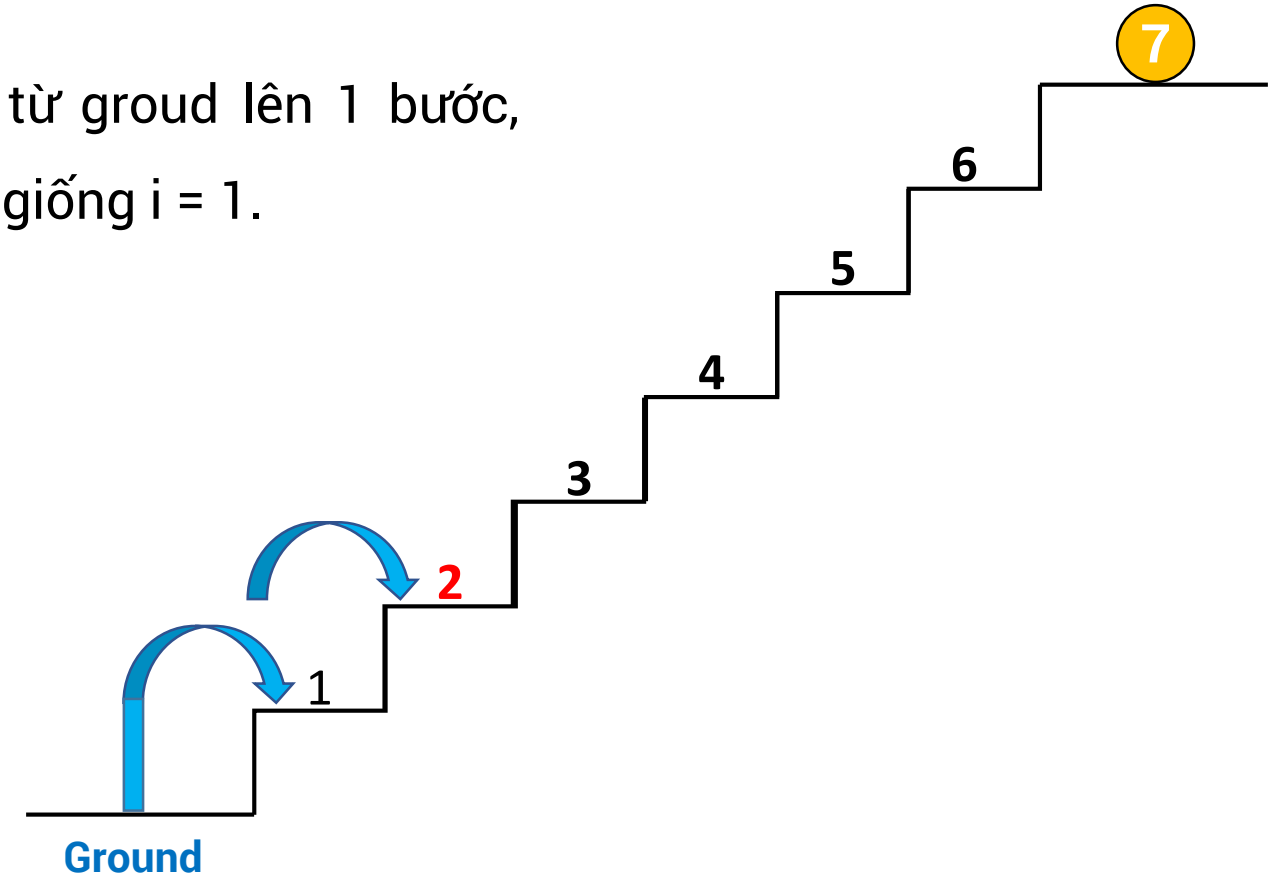
➔ Chỉ có duy nhất 1 cách



Bước 2: bậc thang có 2 bậc

Với i = 2:

- Bước 1 bước → đi từ groud lên 1 bước, sau đó đi một bước giống i = 1.
  - {1} + 1 → {1, 1}

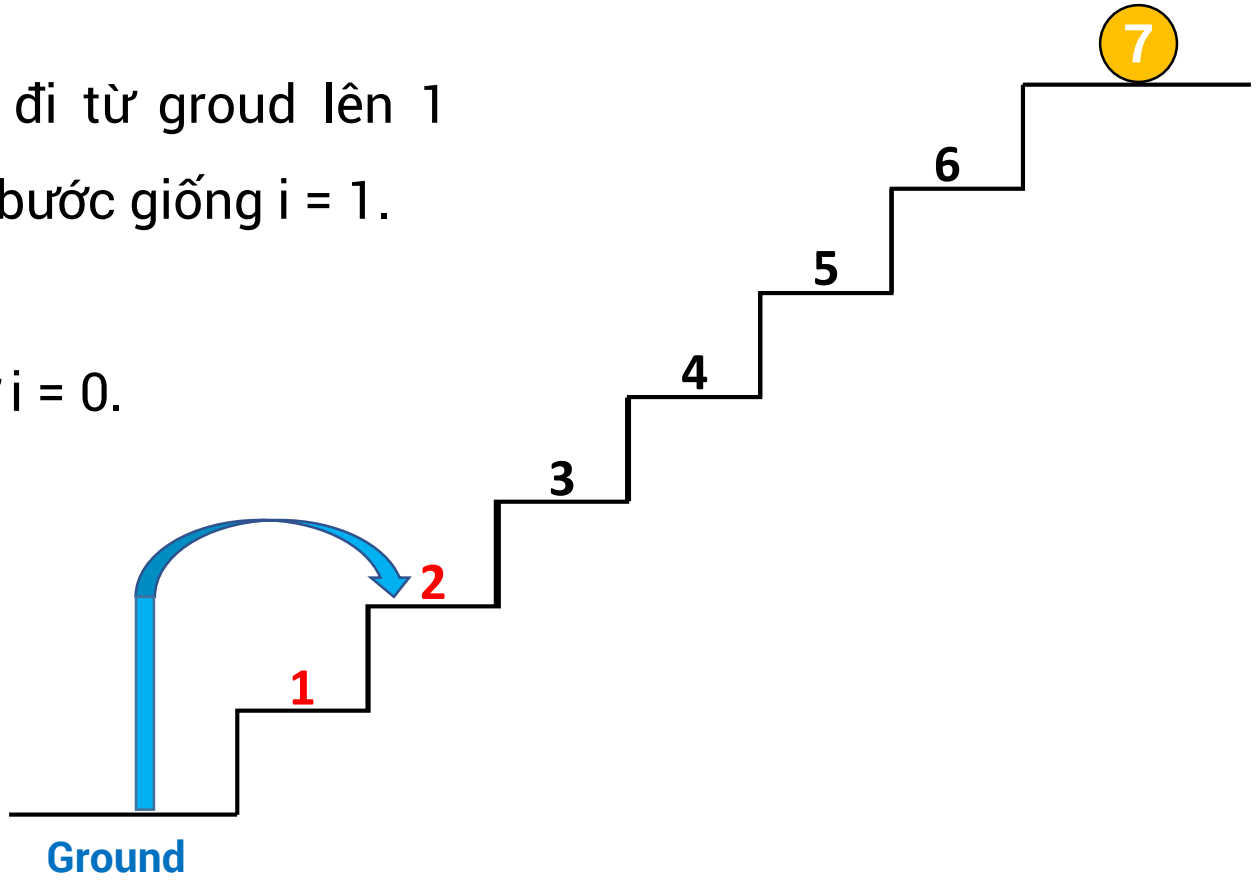


Bước 2: bậc thang có 2 bậc

Với i = 2:

- Bước 1 bước → → đi từ groud lên 1 bước, sau đó đi một bước giống i = 1.
  - {1} + 1 → {1, 1}
- Bước 2 bước → đi từ i = 0.
  - { } + 2 → {2}

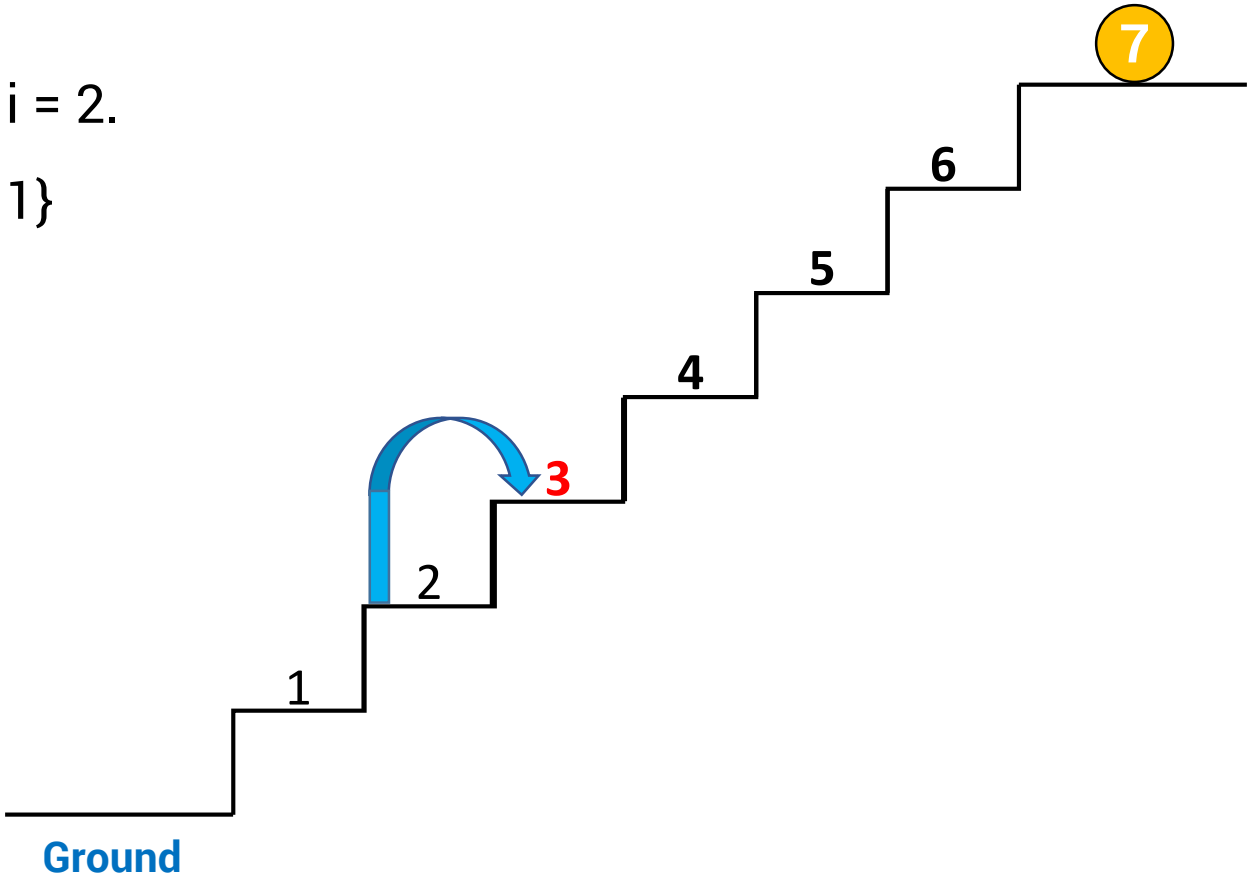
➔ **Tổng cộng 2 cách.**



Bước 3: bậc thang có 3 bậc

Với i = 3:

- Bước 1 bước → đi từ i = 2.
  - {1, 1} + 1 → {1, 1, 1}
  - {2} + 1 → {2, 1}

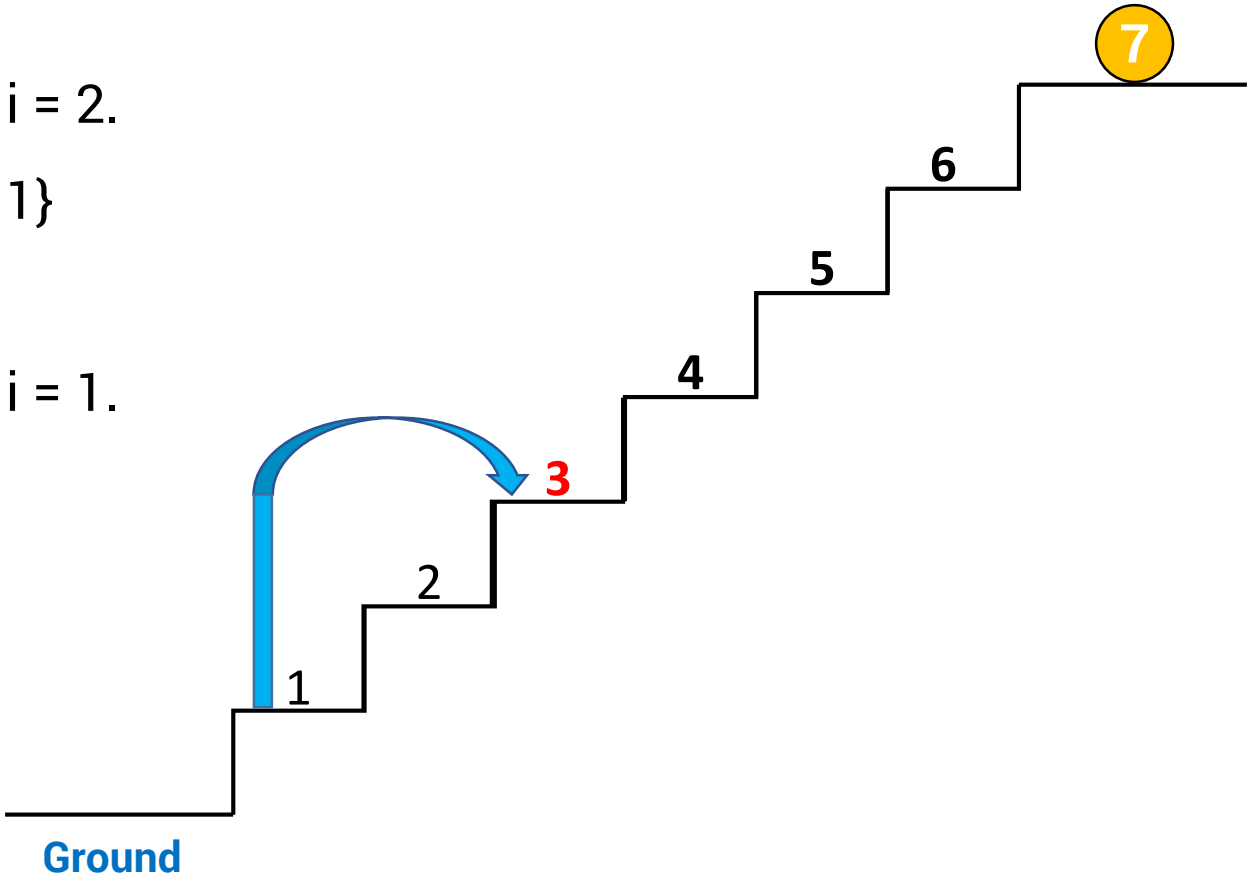


Bước 3: bậc thang có 3 bậc

Với i = 3:

- Bước 1 bước → đi từ i = 2.
  - {1, 1} + 1 → {1, 1, 1}
  - {2} + 1 → {2, 1}
- Bước 2 bước → đi từ i = 1.
  - {1} + 2 → {1, 2}

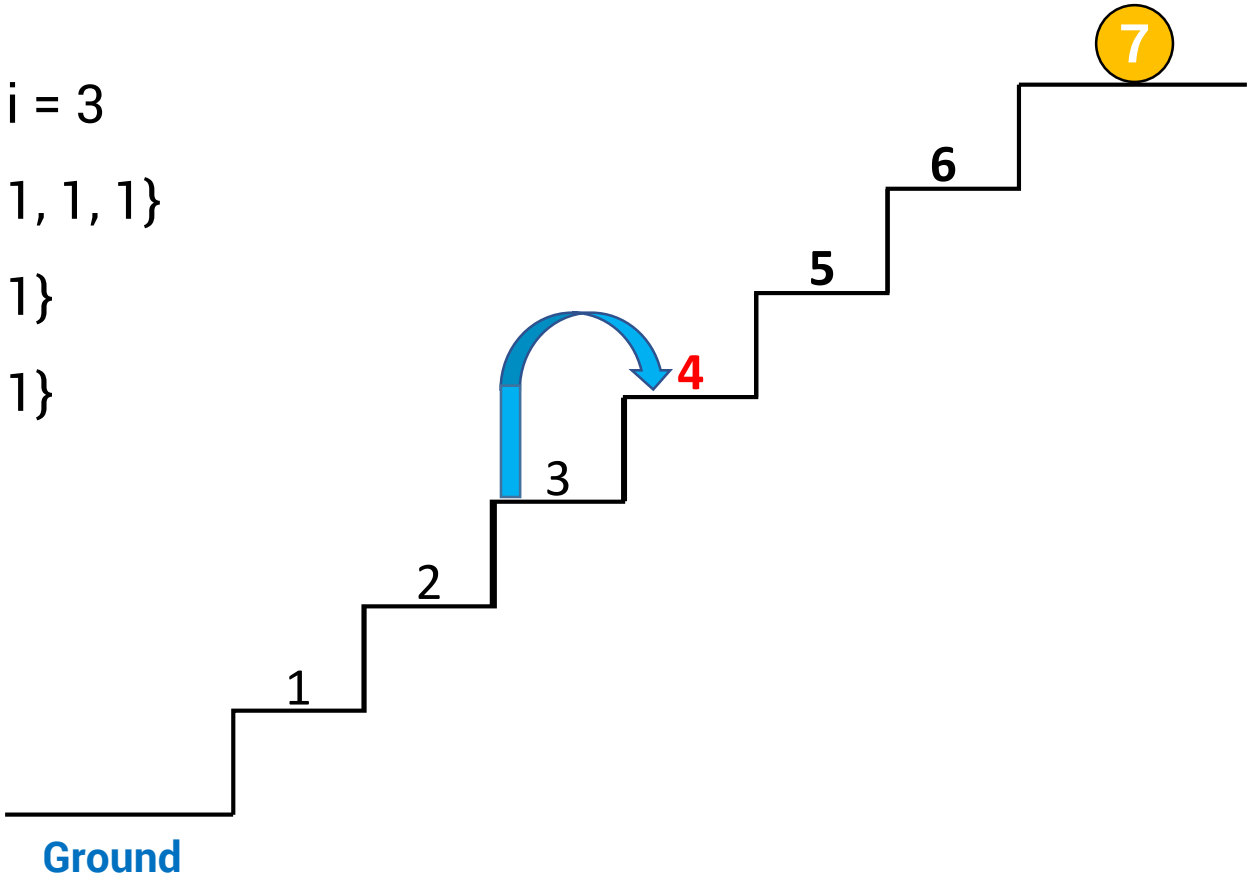
➔ **Tổng cộng 3 cách.**



Bước 4: bậc thang có 4 bậc

Với i = 4:

- Bước 1 bước → đi từ i = 3
  - {1, 1, 1} + 1 → {1, 1, 1, 1}
  - {2, 1} + 1 → {2, 1, 1}
  - {1, 2} + 1 → {1, 2, 1}

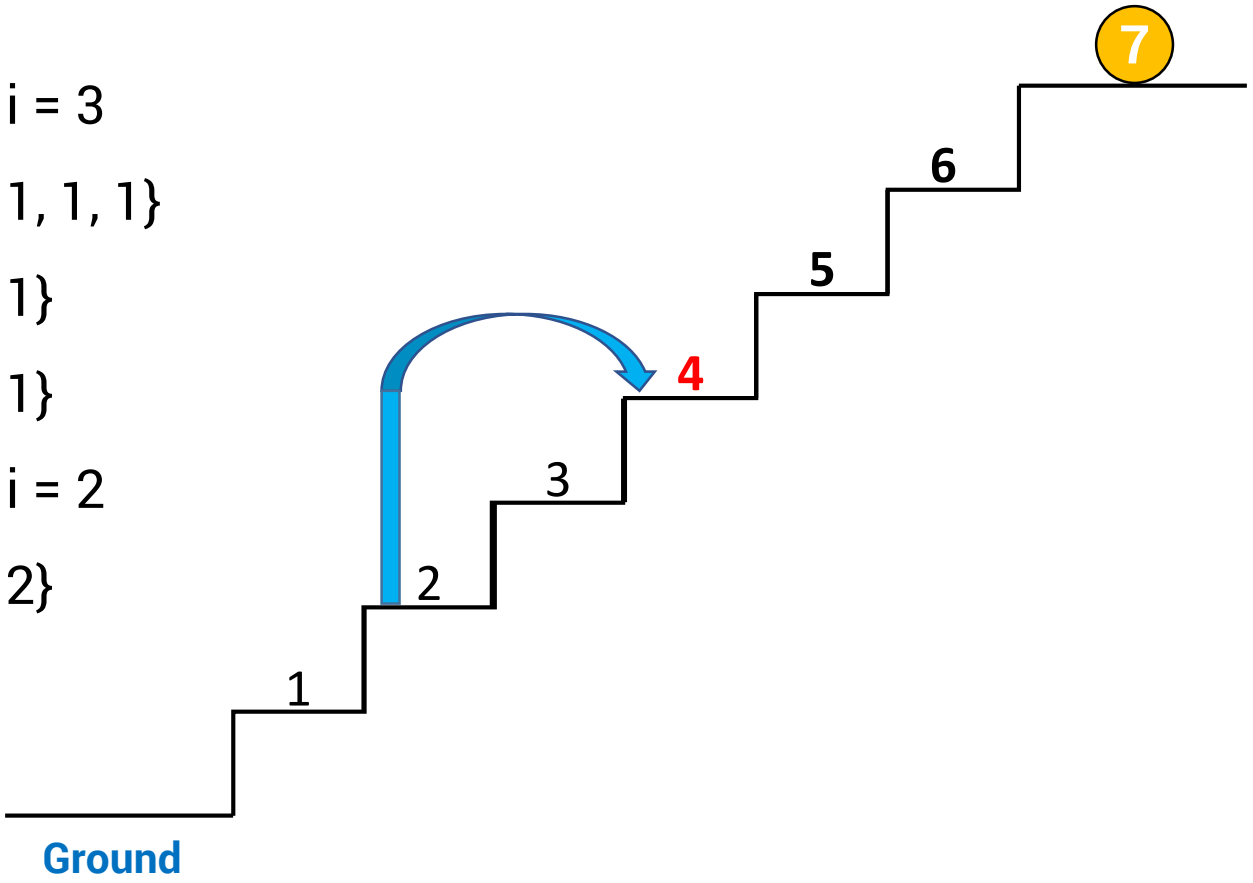


Bước 4: bậc thang có 4 bậc

Với i = 4:

- Bước 1 bước → đi từ i = 3
  - {1, 1, 1} + 1 → {1, 1, 1, 1}
  - {2, 1} + 1 → {2, 1, 1}
  - {1, 2} + 1 → {1, 2, 1}
- Bước 2 bước → đi từ i = 2
  - {1, 1} + 2 → {1, 1, 2}
  - {2} + 2 → {2, 2}

➔ **Tổng cộng 5 cách.**

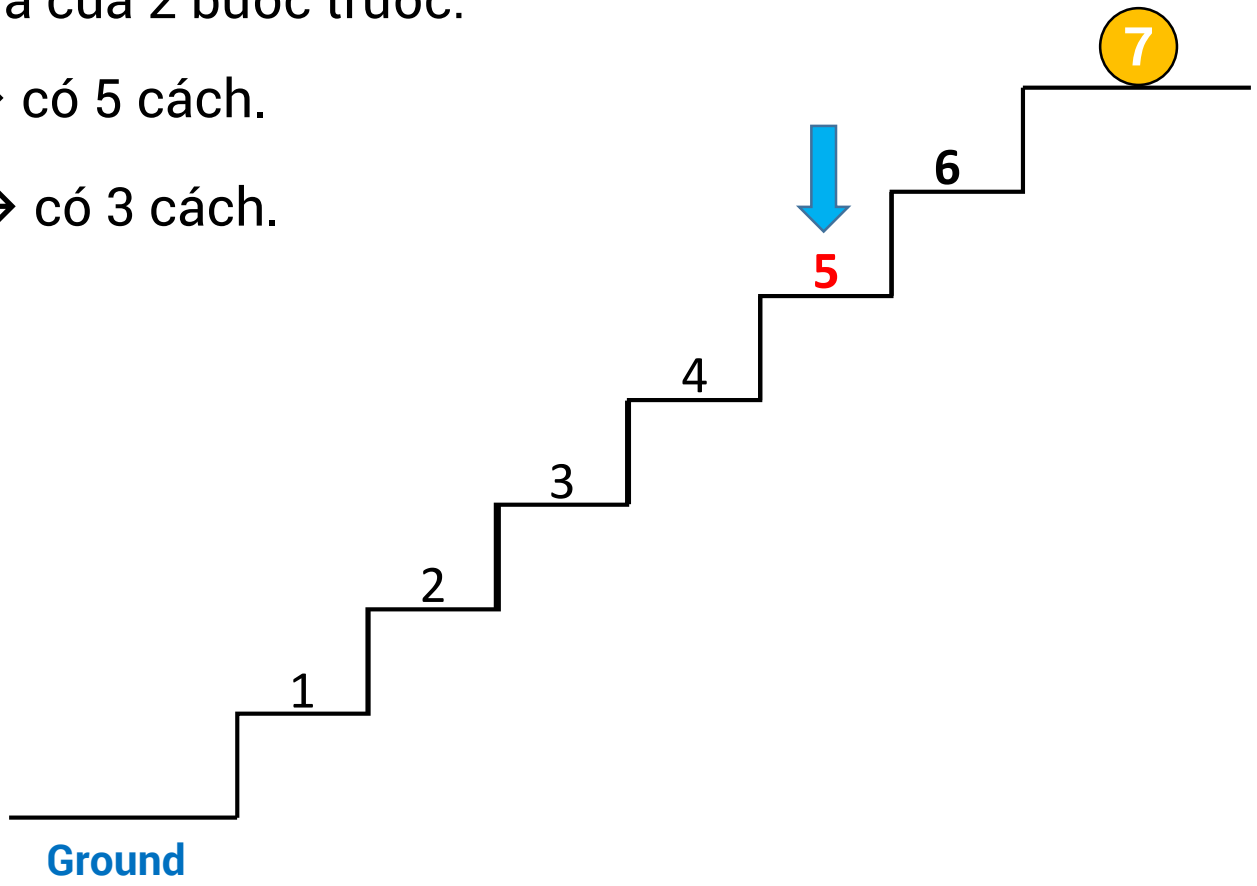


Bước 5: bậc thang có 5 bậc

Với i = 5, dựa vào kết quả của 2 bước trước:

- i = 4 (bước 1 bước)→ có 5 cách.
- i = 3 (bước 2 bước) → có 3 cách.

➔ **Tổng cộng 8 cách.**

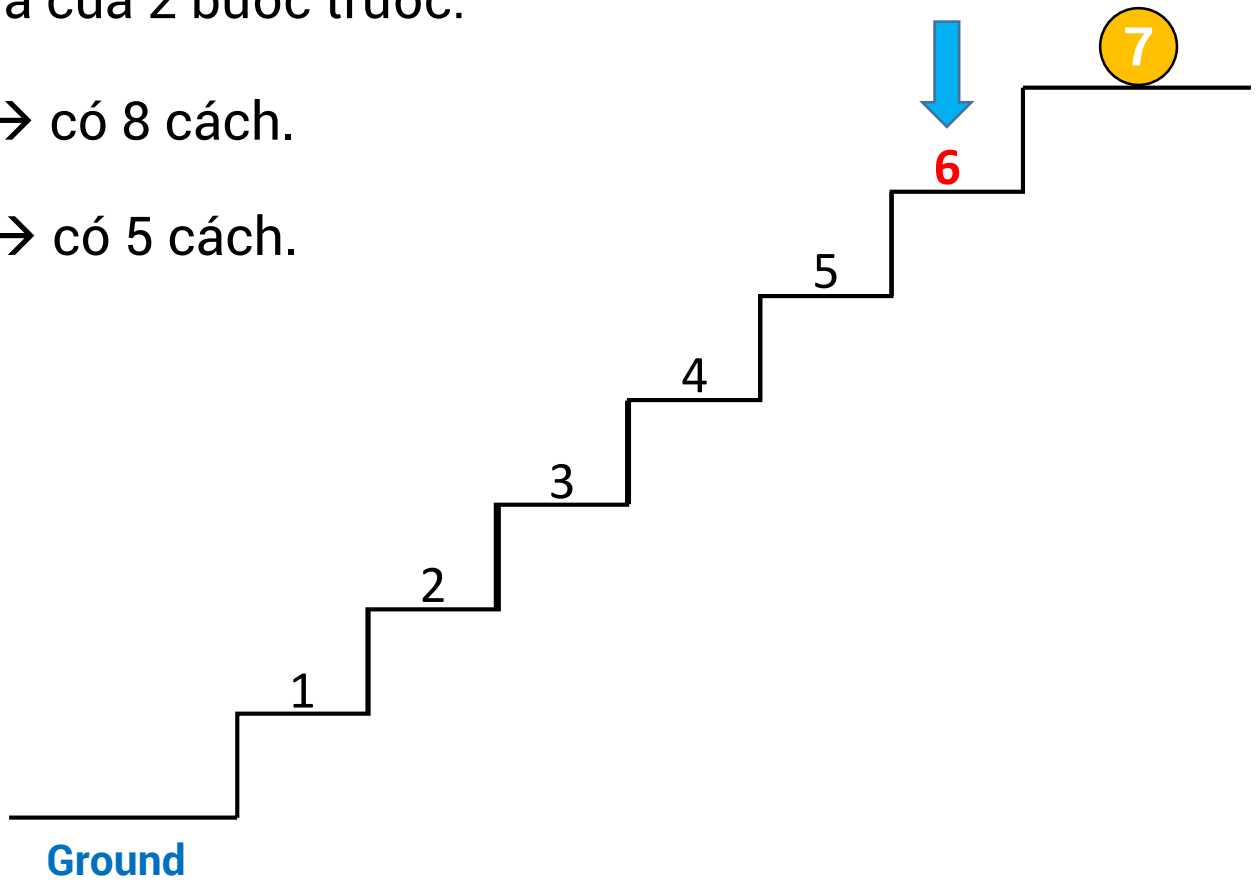


Bước 6: bậc thang có 6 bậc

Với i = 6, dựa vào kết quả của 2 bước trước:

- i = 5 (bước 1 bước) → có 8 cách.
- i = 4 (bước 2 bước) → có 5 cách.

➔ **Tổng cộng 13 cách.**

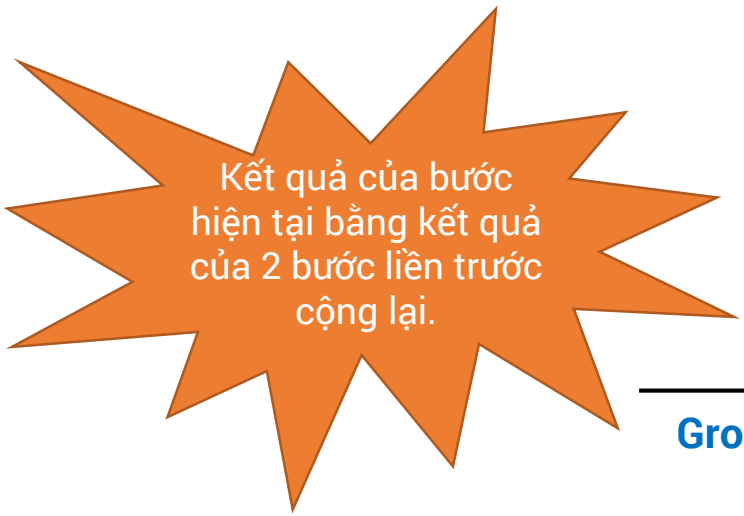


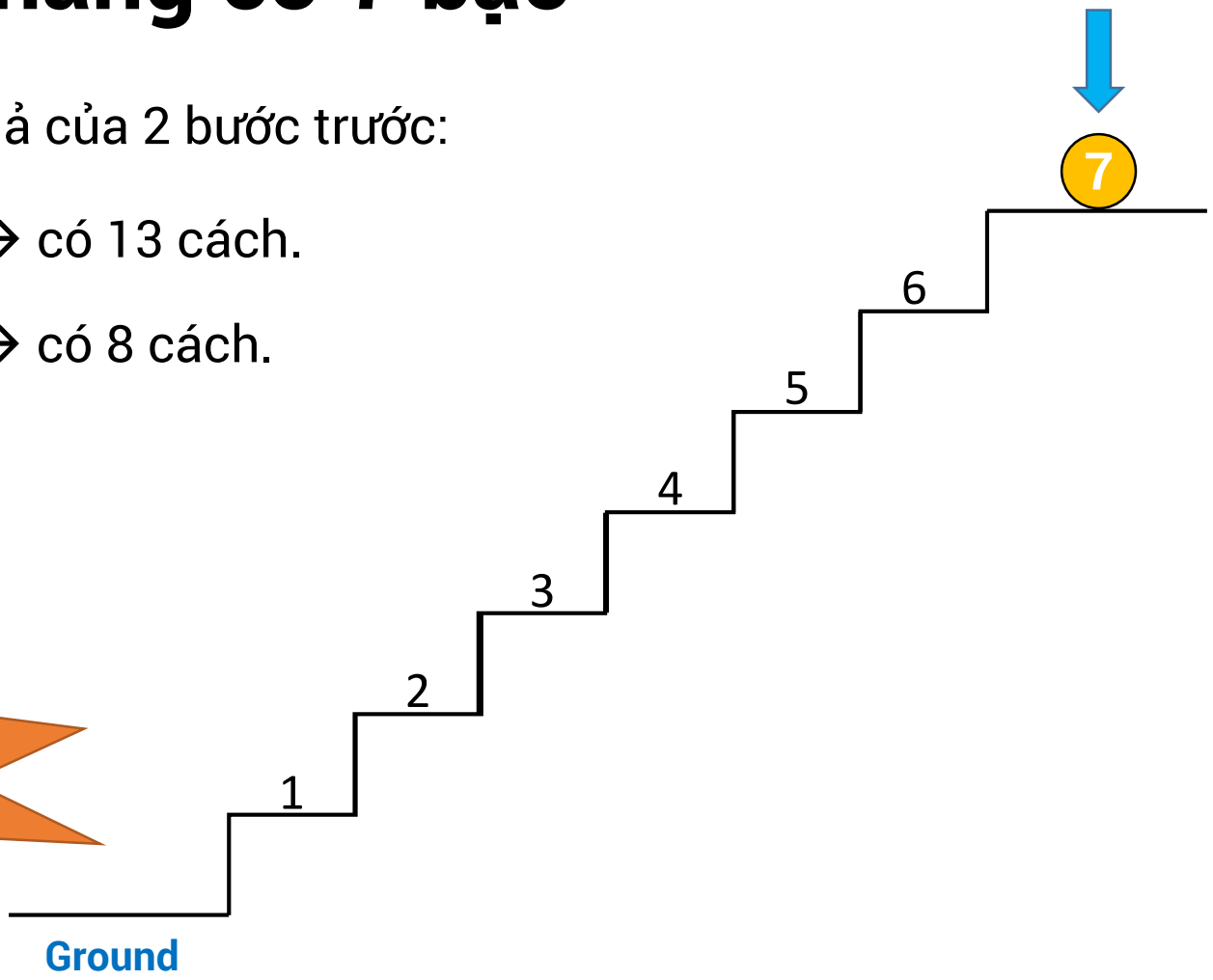
Bước 7: bậc thang có 7 bậc

Với i = 7, dựa vào kết quả của 2 bước trước:

- i = 6 (bước 1 bước) → có 13 cách.
- i = 5 (bước 2 bước) → có 8 cách.

➔ **Tổng cộng 21 cách.**





Fibonacci series

Source Code Staircase

Big-O Orange

```cpp
#include <iostream>
#include <vector>
using namespace std;
int staircase(int n)
{
    vector<int> s(n + 1);
    s[0] = 1;
    s[1] = 1;
    for (int i = 2; i <= n; i++)
        s[i] = s[i - 1] + s[i - 2];
    return s[n];
}
int main()
{
    int n = 7;
    cout << staircase(n);
    return 0;
}
```

Source Code Staircase

```
1.  def staircase(n):
2.      s = [0] * (n + 1)
3.      s[0] = 1
4.      s[1] = 1
5.      for i in range(2, n + 1):
6.          s[i] = s[i - 1] + s[i - 2]
7.      return s[n]
8.
9.  if __name__ == "__main__":
10.     n = 7
11.     print(staircase(n))
```

Source Code Staircase


```java
public class Main {
    private static int staircase(int n) {
        int[] s = new int[n + 1];
        s[0] = 1;
        s[1] = 1;
        for (int i = 2; i <= n; i++)
            s[i] = s[i - 1] + s[i - 2];
        return s[n];
    }

    public static void main(String[] args) {
        int n = 7;
        System.out.println(staircase(n));
    }
}
```

Bài toán minh họa 2

Coin Change Problem (Bài toán đổi tiền): Cho bạn một khoản tiền và N loại tiền lẻ (với số lượng vô hạn). Hãy tính xem bạn có bao nhiêu cách đổi tiền.

Ví dụ 1: Bạn có số tiền: 5$, các đồng tiền lẻ có mệnh giá {1, 2, 3}.

Các cách đổi tiền như sau:

- { 1, 1, 1, 1, 1 }
- { 1, 1, 1, 2 }
- { 1, 1, 3 }
- { 1, 2, 2 }
- { 2, 3 }

➔ Tổng cộng 5 cách đổi tiền.

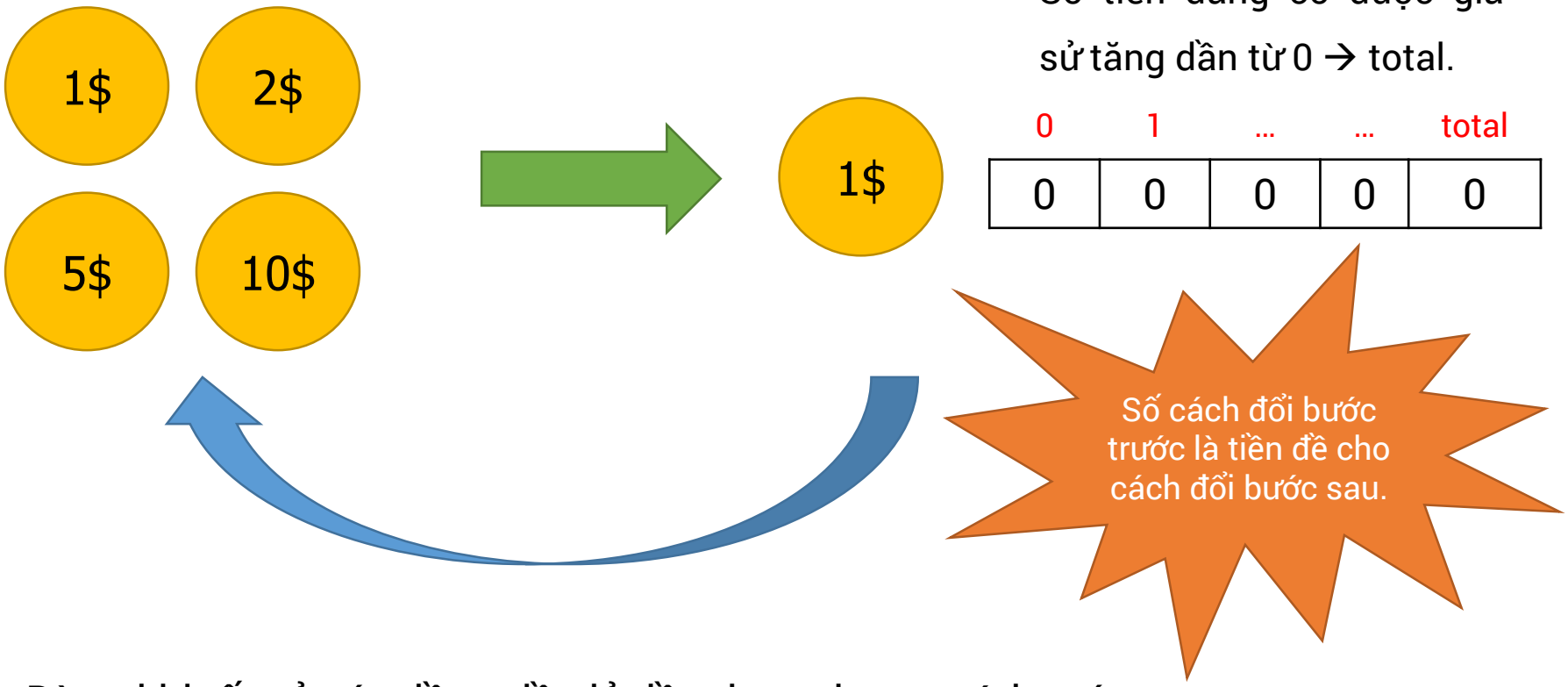
Coin Change Problem

Bây giờ sẽ chạy tay với dữ liệu lớn hơn để hiểu rõ giải bài toán trên. Cho bạn thông tin như sau:

- **Số tiền: 10$**
- **Danh sách các đồng xu: 1$, 2$, 5$, 10$**

Ý tưởng giải quyết bài toán

Lần lượt đem từng loại đồng ra để đổi các giá trị của tổng tiền đang có.



➔ Dừng khi tất cả các đồng tiền lẻ đều được đem ra tính toán xong.

Time Complexity: **O(total*N)**

Bước 0: Chuẩn bị dữ liệu

Mảng danh sách các đồng tiền.

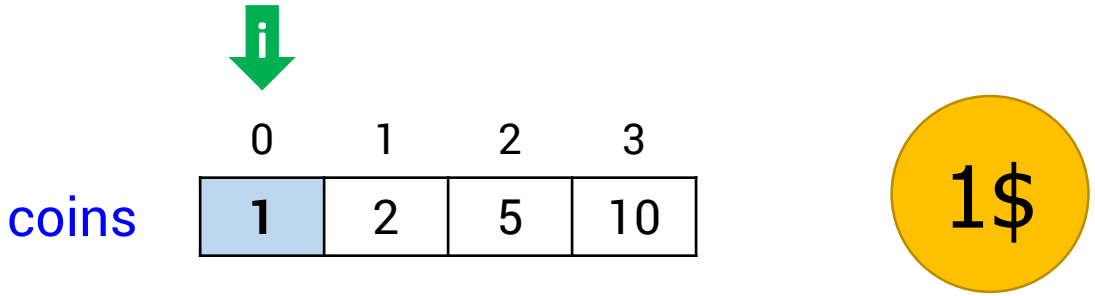
	0	1	2	3
coins	1	2	5	10

Số tiền 10$ tạo thành mảng có 11 phần tử. Phần tử đầu tiên có giá trị là 1. Các phần tử còn lại giá trị là 0.

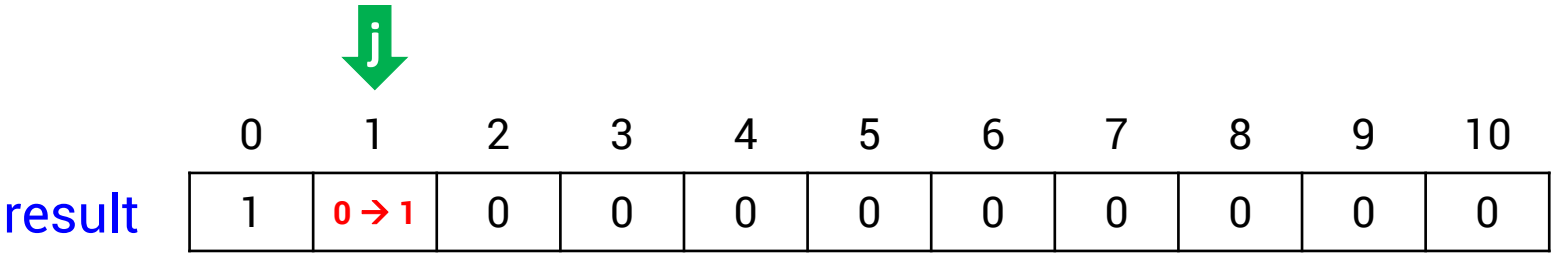
	0	1	2	3	4	5	6	7	8	9	10
result	1	0	0	0	0	0	0	0	0	0	0

**** Lưu ý: khi bạn có 0$ thì số lượng đồng tiền của mỗi mệnh giá sử dụng là 0 lần. Khi đó bạn có 1 cách đổi.*

Bước 1: Với đồng 1$ chạy thuật toán j = 1



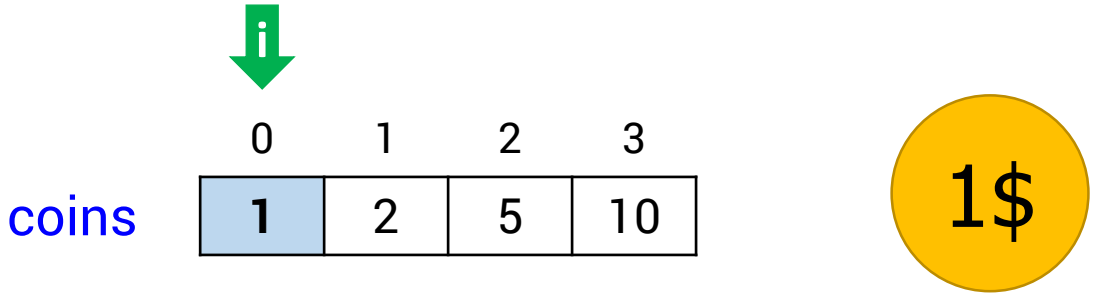
Giả sử chúng ta sử dụng loại đồng 1$ để đổi các số tiền mà chúng ta đang có.



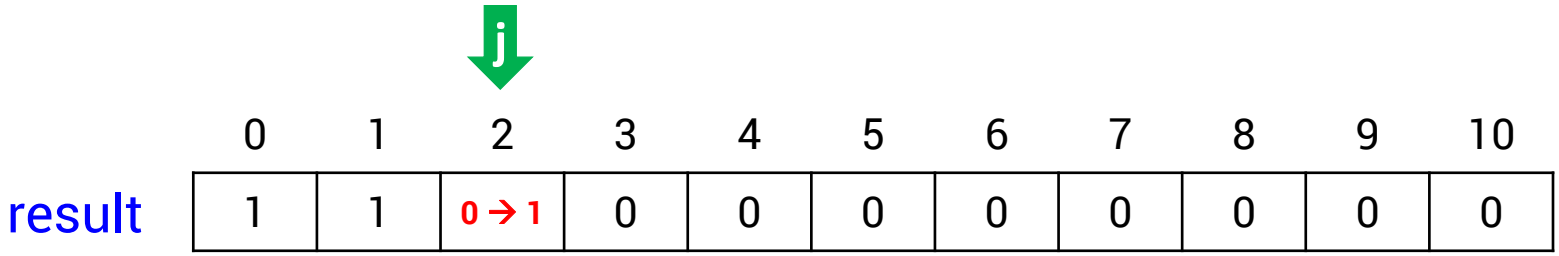
➔ Có 1 cách đổi:

- {1}

Bước 1: Với đồng 1$ chạy thuật toán j = 2



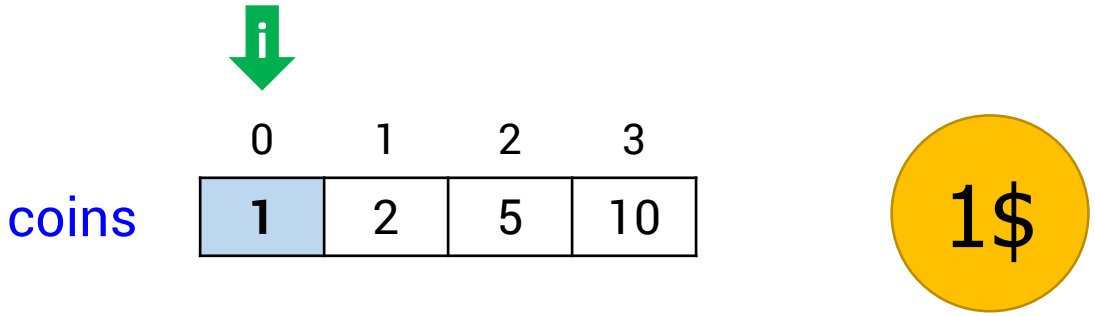
Giả sử chúng ta sử dụng loại đồng 1$ để đổi các số tiền mà chúng ta đang có.



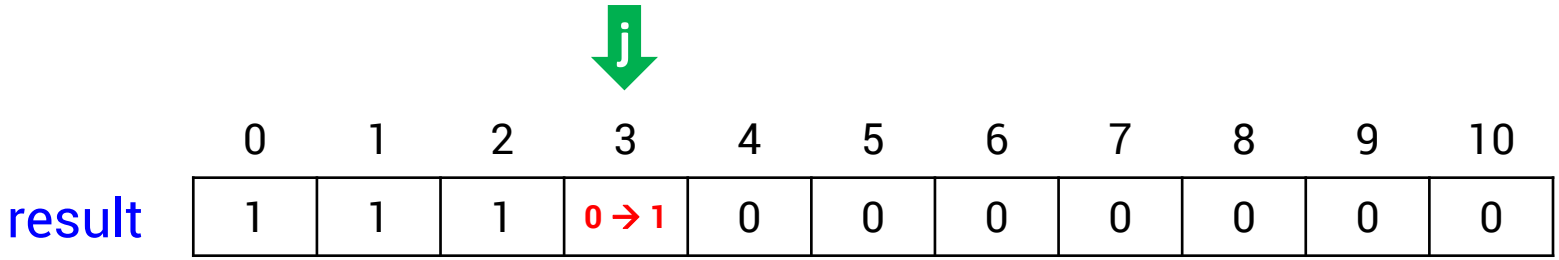
➔ Có 1 cách đổi:

- {1, 1}

Bước 1: Với đồng 1$ chạy thuật toán j = 3



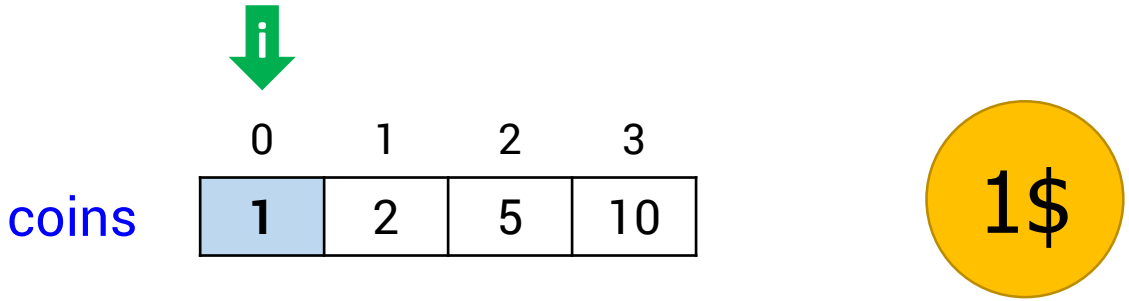
Giả sử chúng ta sử dụng loại đồng 1$ để đổi các số tiền mà chúng ta đang có.



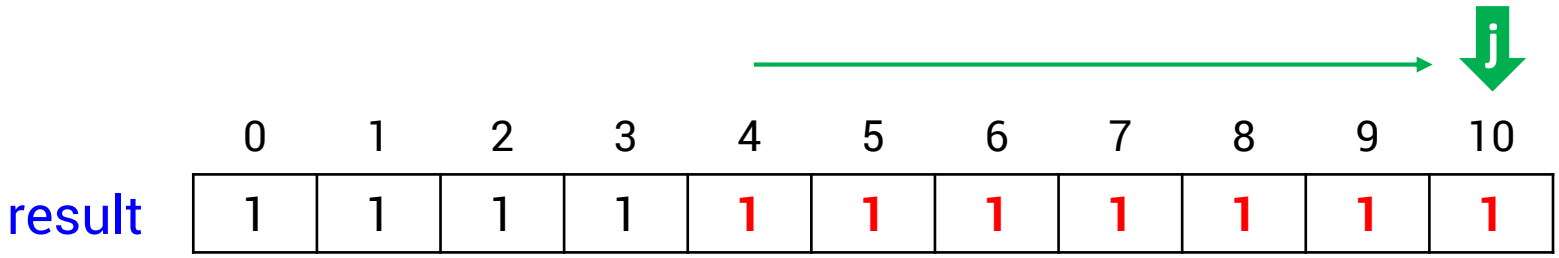
➔ Có 1 cách đổi:

- {1, 1, 1}

Bước 1: Với đồng 1$ chạy thuật toán j = 4 → 10

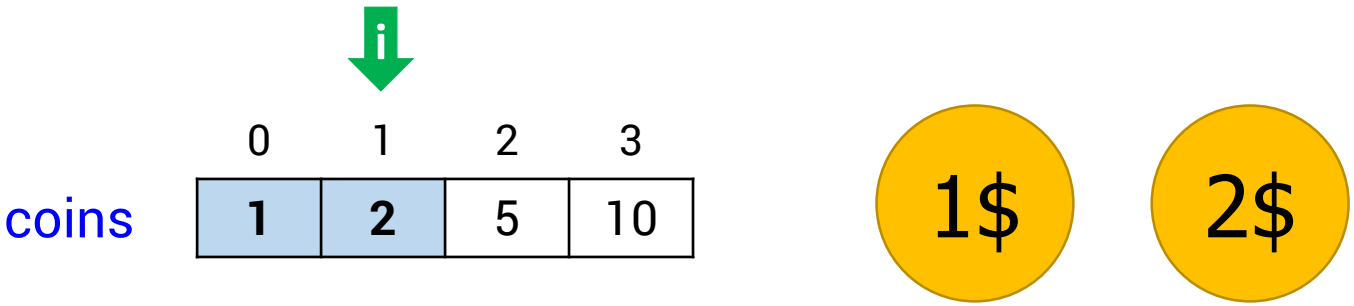


Giả sử chúng ta sử dụng loại đồng 1$ để đổi các số tiền mà chúng ta đang có.

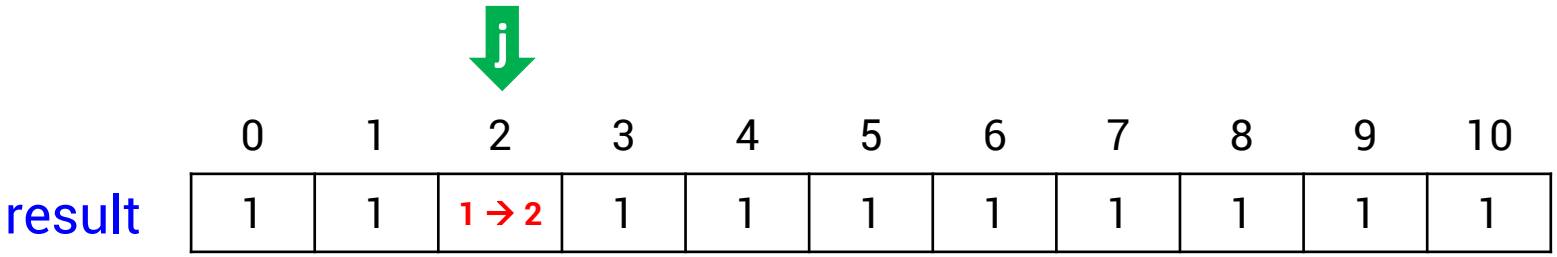


➔ Tương tự chỉ có một cách đổi đối với các số tiền còn lại.

Bước 2: Với đồng 1$ 2$ chạy thuật toán j = 2



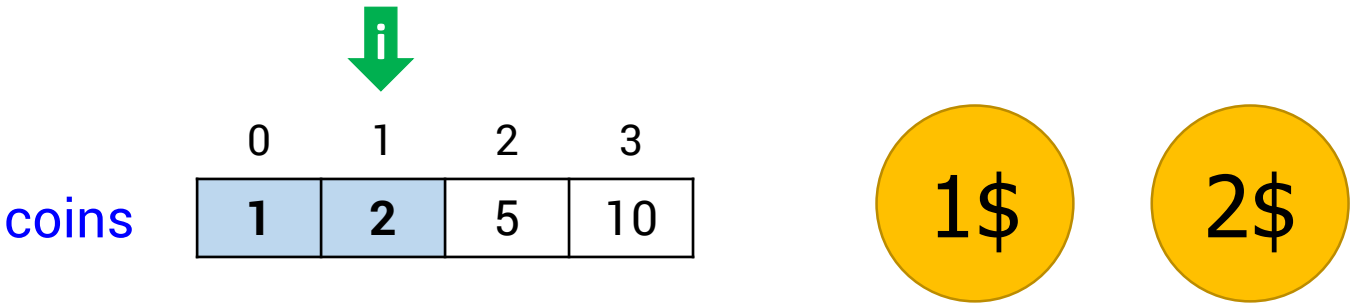
Giả sử chúng ta sử dụng loại đồng 1$, 2$ để đổi các số tiền mà chúng ta đang có.



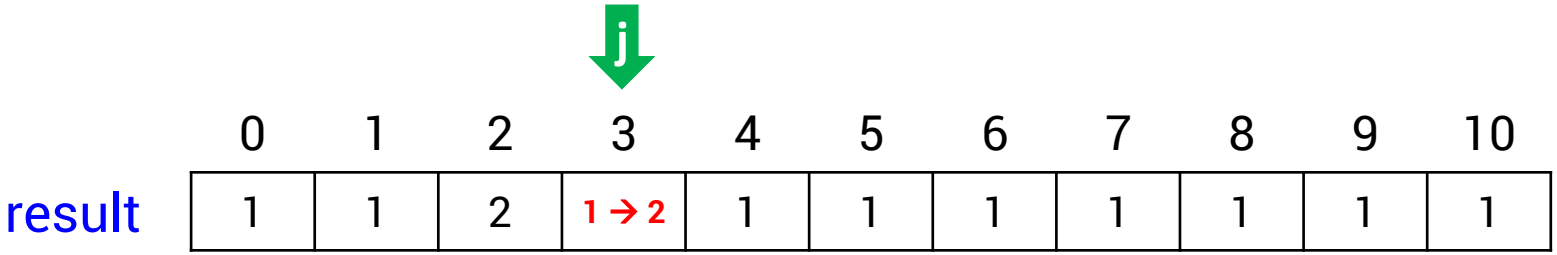
➔ Có 2 cách đổi:

- {1, 1} (cách đổi có được từ bước 1)
- **{2}**

Bước 2: Với đồng 1$ 2$ chạy thuật toán j = 3



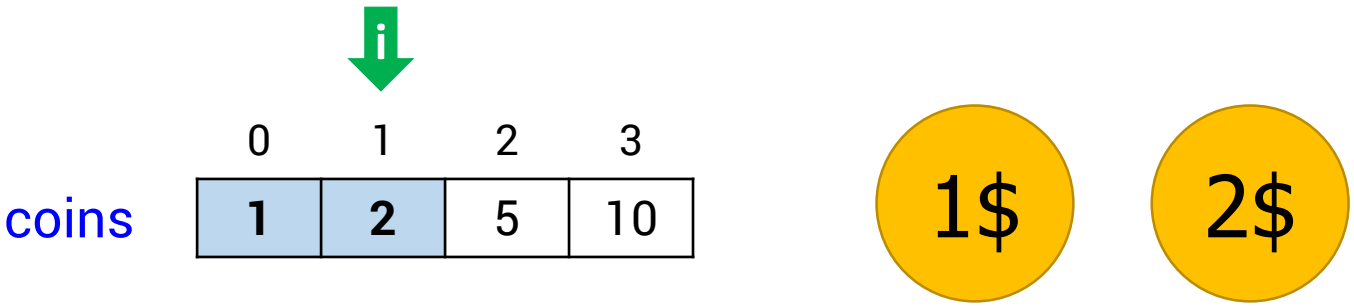
Giả sử chúng ta sử dụng loại đồng 1$, 2$ để đổi các số tiền mà chúng ta đang có.



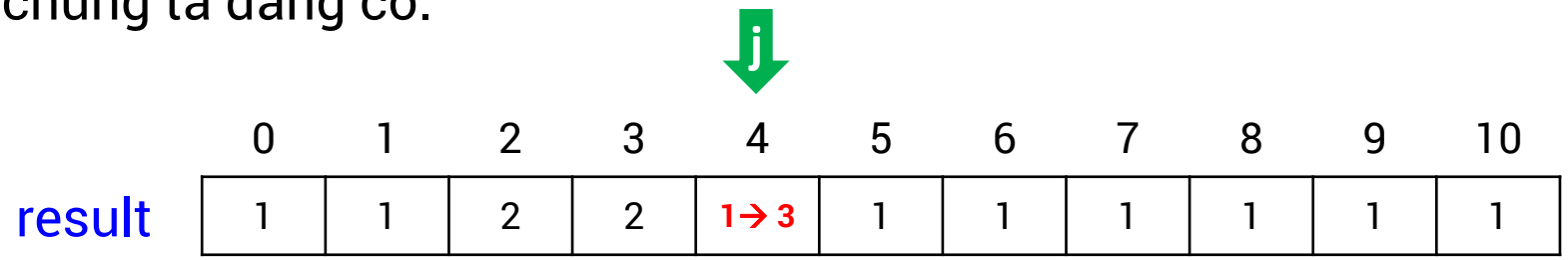
➔ Có 2 cách đổi:

- {1, 1, 1} (cách đổi có được từ bước 1)
- **{1, 2}**

Bước 2: Với đồng 1$ 2$ chạy thuật toán j = 4



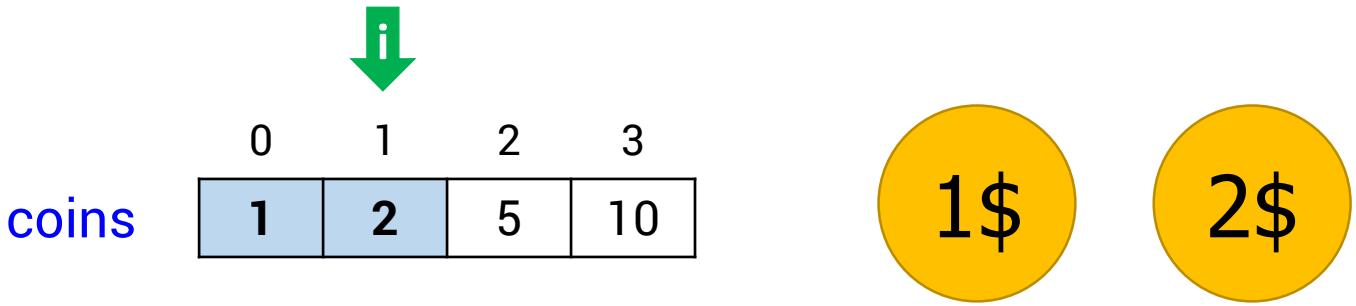
Giả sử chúng ta sử dụng loại đồng 1$, 2$ để đổi các số tiền mà chúng ta đang có.



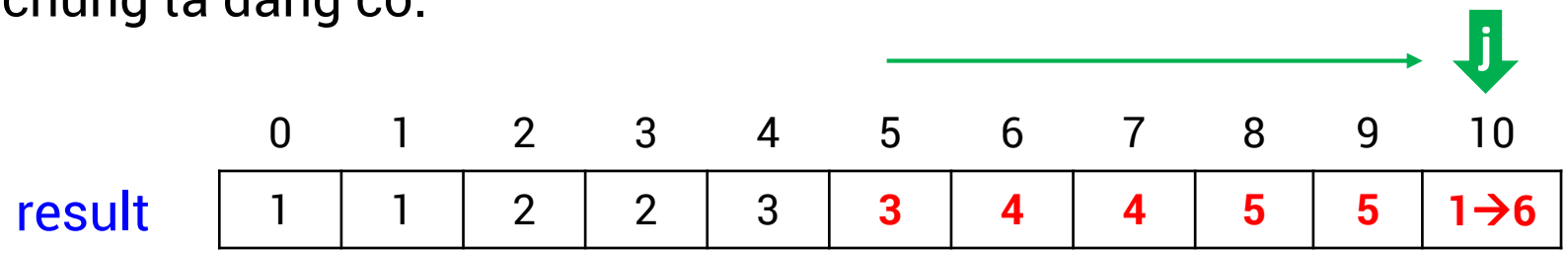
➔ Có 3 cách đổi:

- {1, 1, 1, 1} (cách đổi có được từ bước 1)
- **{1, 1, 2}**
- **{2, 2}**

Bước 2: Với đồng 1$ 2$ chạy thuật toán j = 5 → 10



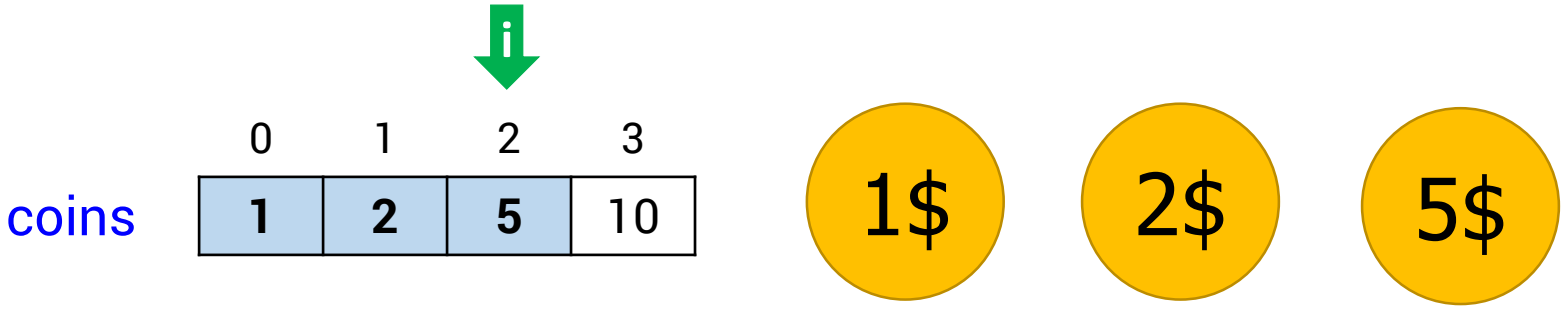
Giả sử chúng ta sử dụng loại đồng 1$, 2$ để đổi các số tiền mà chúng ta đang có.



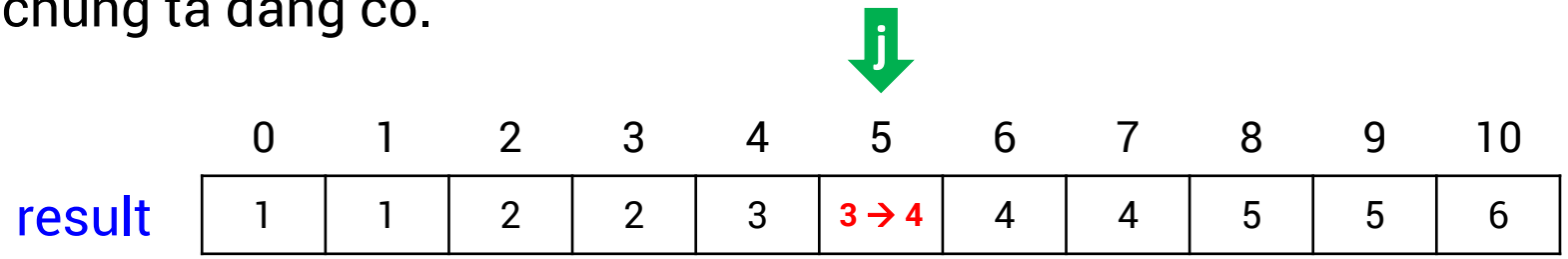
Kết luận: khi bạn có đồng 1$ và 2$ để đổi 10$ bạn có 6 cách:

- {1, 1, 1, 1, 1, 1, 1, 1, 1, 1} (cách đổi có được từ bước 1)
- **{1, 1, 1, 1, 1, 1, 1, 1, 2}, {1, 1, 1, 1, 1, 1, 2, 2}, {1, 1, 1, 1, 2, 2, 2}, {1, 1, 2, 2, 2, 2}, {2, 2, 2, 2, 2}**

Bước 3: Với đồng 1$ 2$ 5$, chạy thuật toán j = 5



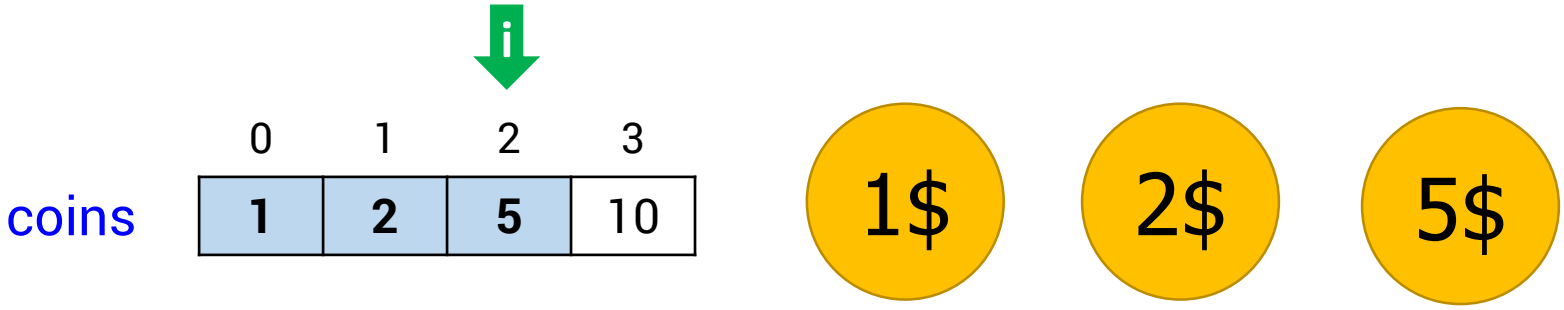
Giả sử chúng ta sử dụng loại đồng 1$, 2$, 5$ để đổi các số tiền mà chúng ta đang có.



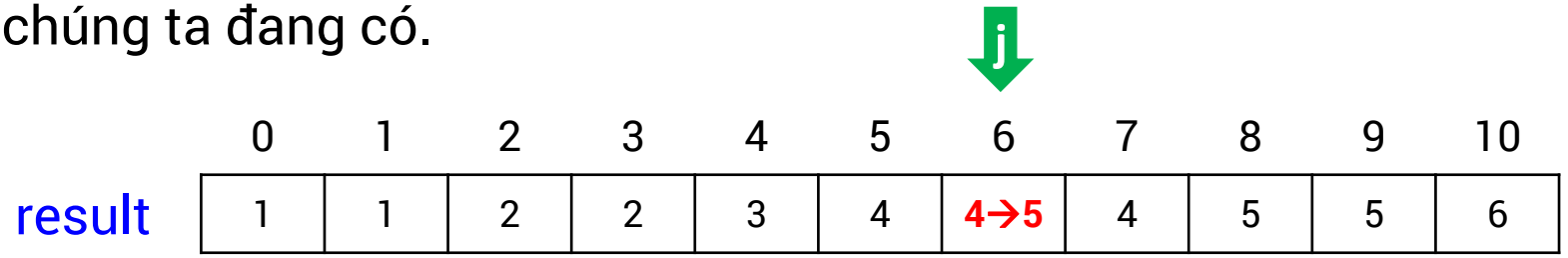
➔ Có 4 cách đổi:

- {1, 1, 1, 1, 1} (cách đổi có được từ bước 1)
- {1, 1, 1, 2}, {1, 2, 2} (cách đổi có được từ bước 2)
- **{5}**

Bước 3: Với đồng 1$ 2$ 5$ chạy thuật toán j = 6



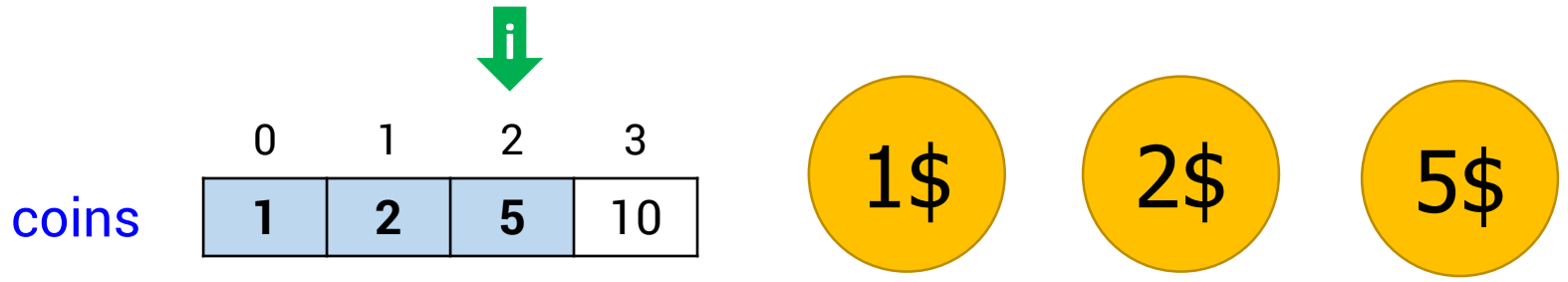
Giả sử chúng ta sử dụng loại đồng 1$, 2$, 5$ để đổi các số tiền mà chúng ta đang có.



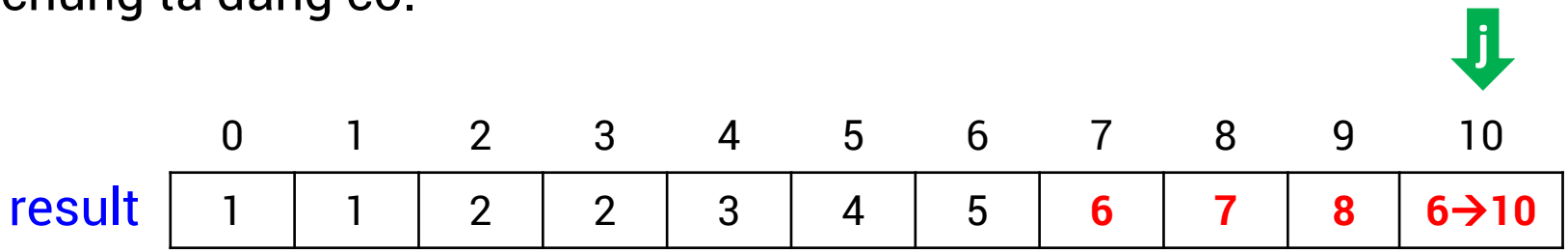
➔ Có 5 cách đổi:

- {1, 1, 1, 1, 1, 1} (cách đổi có được từ bước 1)
- {1, 1, 1, 1, 2}, {1, 1, 2, 2}, {2, 2, 2} (cách đổi có được từ bước 2)
- **{1, 5}**

Bước 3: Với đồng 1$ 2$ 5$ chạy thuật toán j = 6→10



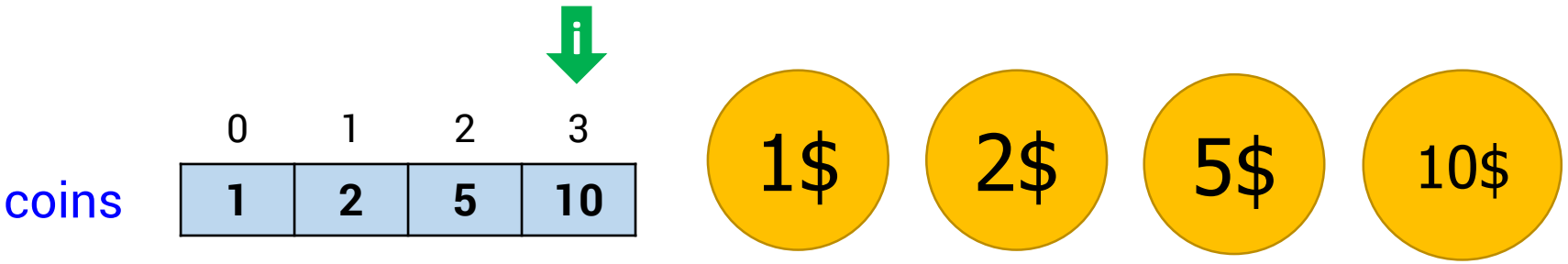
Giả sử chúng ta sử dụng loại đồng 1$, 2$, 5$ để đổi các số tiền mà chúng ta đang có.



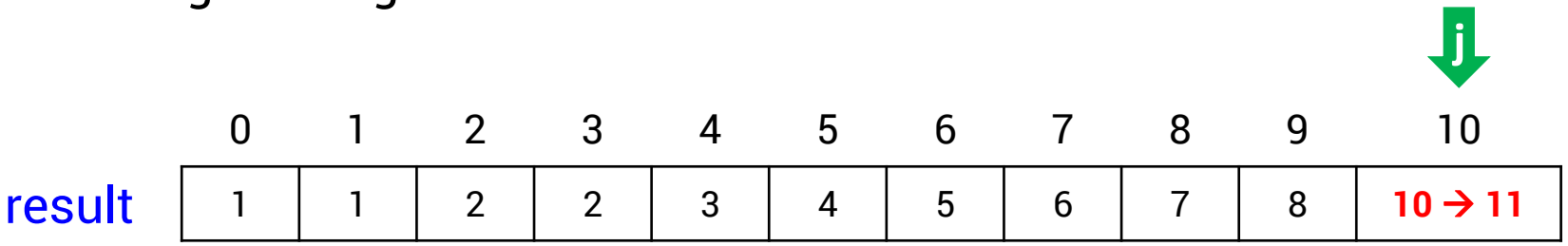
Kết luận: khi bạn có đồng 1$, 2$ và 5$ để đổi 10$ bạn có 10 cách:

- 6 cách bước trước.
- 4 cách mới: **{1, 1, 1, 1, 1, 5}, {1, 1, 1, 2, 5}, {1, 2, 2, 5}, {5, 5}**

Bước 4: Với đồng 1$ 2$ 5$ 10$ chạy thuật toán j = 10



Giả sử chúng ta sử dụng loại đồng 1$, 2$, 5$, 10$ để đổi các số tiền mà chúng ta đang có.



Kết luận: khi bạn có đồng 1$, 2$, 5$, 10$ để đổi 10$ bạn có 11 cách.

- 10 cách ở bước trước.
- 1 cách mới: **{10}**

Kết quả thuật toán

Kết quả của mỗi bước là tổng kết quả của bước trước và bước hiện tại.

	0	1	2	3	4	5	6	7	8	9	10
result	1	1	2	2	3	4	5	6	7	8	**11**

- **1 cách ở bước 1:**

  {1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
- **5 cách ở bước 2:**

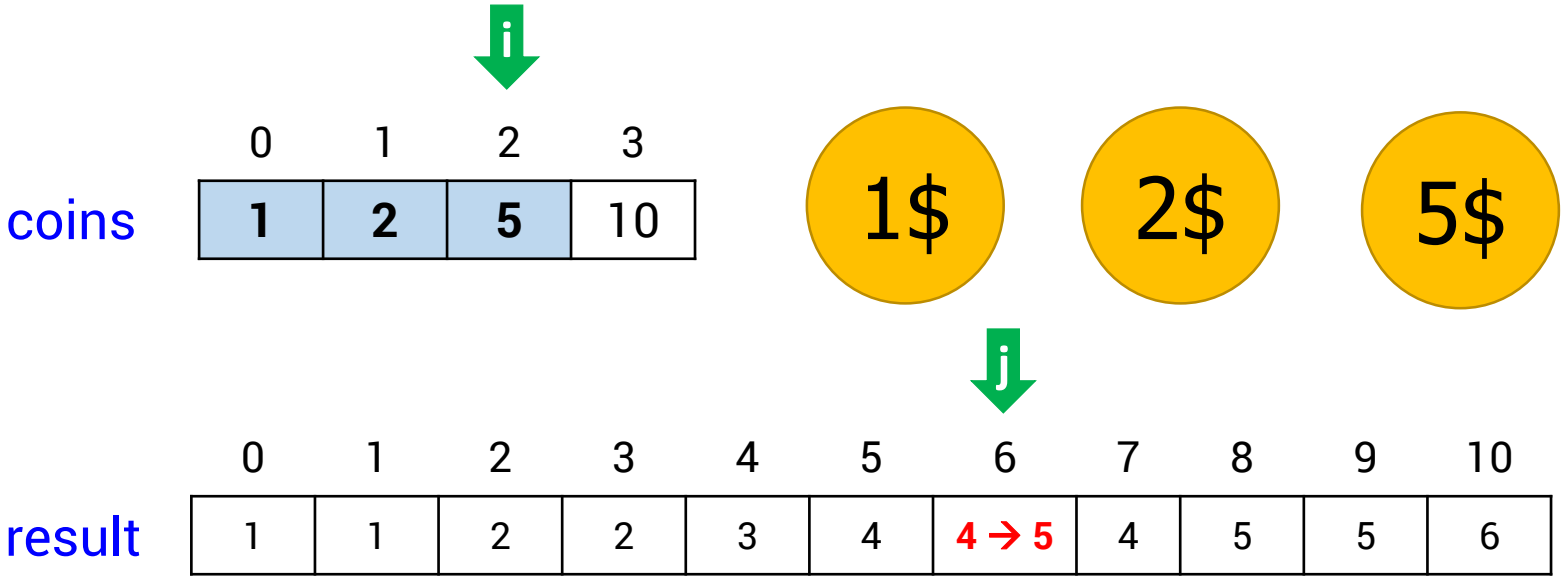
  {1, 1, 1, 1, 1, 1, 1, 1, 2}, {1, 1, 1, 1, 1, 1, 2, 2}, {1, 1, 1, 1, 2, 2, 2},

  {1, 1, 2, 2, 2, 2}, {2, 2, 2, 2, 2}
- **4 cách ở bước 3:**

  {1, 1, 1, 1, 1, 5}, {1, 1, 1, 2, 5}, {1, 2, 2, 5}, {5, 5}
- **1 cách ở bước 4:**

  {10}

Nhận xét 1: Tìm công thức QHĐ

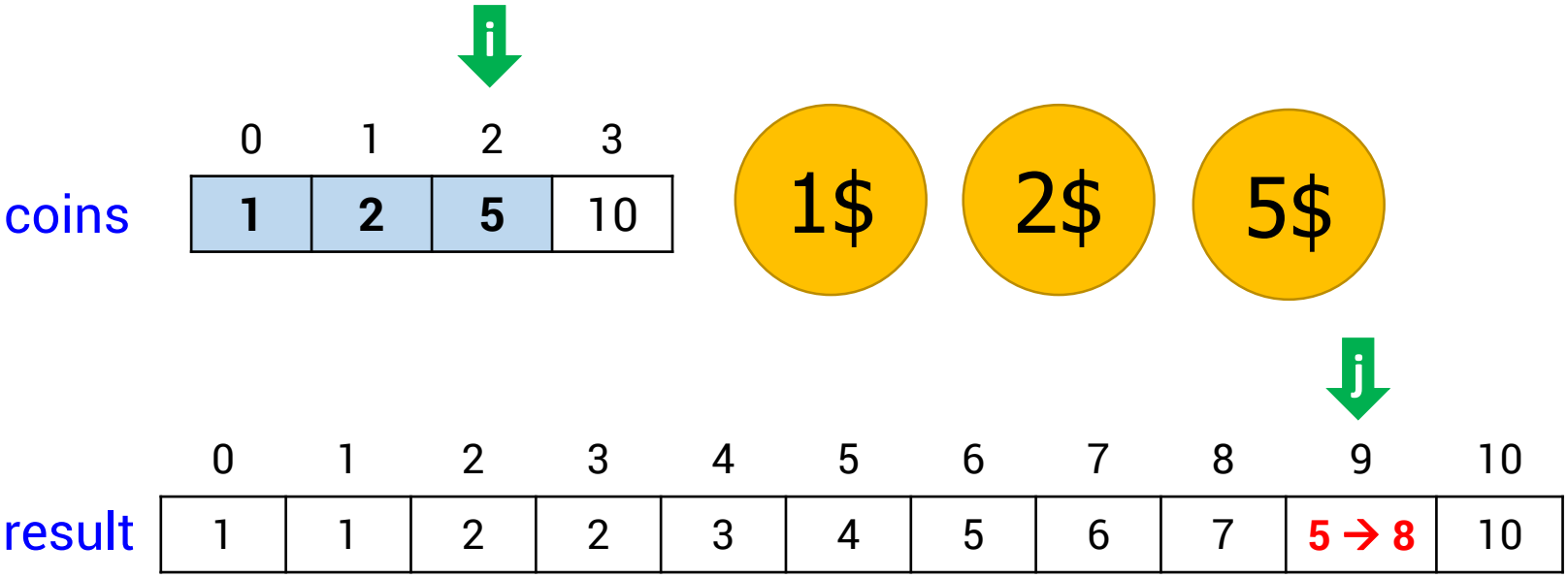


5 cách: 4 cách (trước đó) + 1 (cách mới {1, 5})

- Số cách đổi tiền trước đó result[j].
- Cách mới phải là cách mà có đồng 5$ trong đó. Tìm đến vị trí mà có thể chèn thêm đồng 5$ vào để ra được cách mới: result[j – coins[i]]

  ➔ result [6 – 5] = result[1] = 1.
  - {1} + **{5}**

Nhận xét 2: Tìm công thức QHĐ



8 cách: 5 cách (trước đó) + 3 (cách mới)

- Số cách đổi tiền trước đó result[j] = 5.
- Số cách mới: result[j – coins[i]] = result[9 – 5] = result[4] = 3.
  - {1, 1, 1, 1} + **{5}**
  - {1, 1, 2} + **{5}**
  - {2, 2} + **{5}**

result[j] = result[j] + resutl[j – coins[i]]

IN KẾT QUẢ CHI TIẾT BÀI TOÁN

Bước 1: In kết quả lần 1

Dùng phương pháp Backtracking để in kết quả bài toán:

- total = 10.
- Thêm đồng đầu tiên vào mảng cho đến khi nào không thêm được nữa.

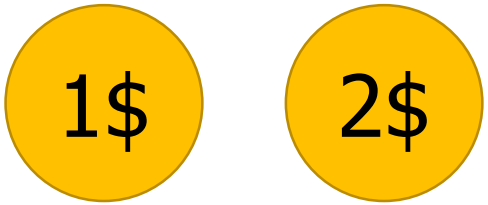
	0	1	2	3	4	5	6	7	8	9
result	1	1	1	1	1	1	1	1	1	1

1, 1, 1, 1, 1, 1, 1, 1, 1, 1

Bước 2: In kết quả lần 2

Dùng phương pháp Backtracking để in kết quả bài toán:

- total = 10.
- Bỏ lần lượt từng đồng tiền 1$ ra cho đến khi nào có thể thêm được đồng tiền khác vào thì đó là kết quả tiếp theo.

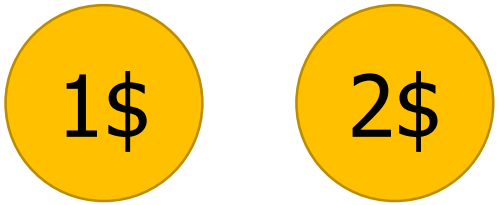


	0	1	2	3	4	5	6	7	8	9
result	1	1	1	1	1	1	1	1	2	

Bước 3: In kết quả lần 3

Dùng phương pháp Backtracking để in kết quả bài toán:

- total = 10.
- Bỏ lần lượt từng đồng tiền 1$ ra cho đến khi nào có thể thêm được đồng tiền khác vào thì đó là kết quả tiếp theo.



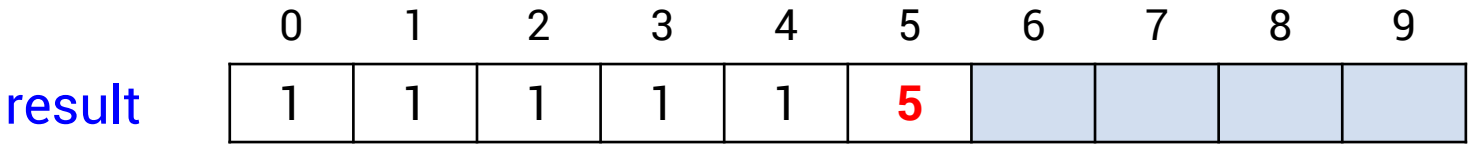
	0	1	2	3	4	5	6	7	8	9
result	1	1	1	1	1	1	2	2		

1, 1, 1, 1, 1, 1, 2, 2

Bước 4: In kết quả lần 4

Dùng phương pháp Backtracking để in kết quả bài toán:

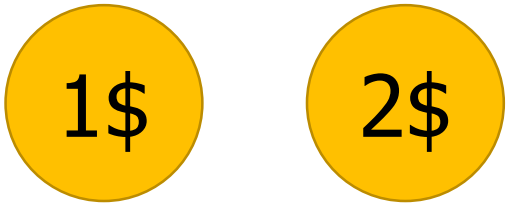
- total = 10
- Bỏ lần lượt từng đồng tiền 1$ ra cho đến khi nào có thể thêm được đồng tiền khác vào thì đó là kết quả tiếp theo.

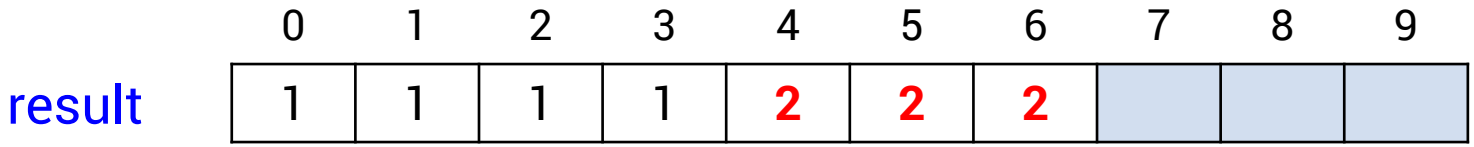


Bước 5: In kết quả lần 5

Dùng phương pháp Backtracking để in kết quả bài toán:

- total = 10
- Bỏ lần lượt từng đồng tiền 1$ ra cho đến khi nào có thể thêm được đồng tiền khác vào thì đó là kết quả tiếp theo.





Bước 6: In kết quả lần 6

Dùng phương pháp Backtracking để in kết quả bài toán:

- total = 10
- Bỏ lần lượt từng đồng tiền 1$ ra cho đến khi nào có thể thêm được đồng tiền khác vào thì đó là kết quả tiếp theo.

	0	1	2	3	4	5	6	7	8	9
result	1	1	1	2	5					

In kết quả bài toán

Tương tự như vậy bạn chạy cho đến khi nào xét hết các trường hợp thì dừng:

- Kết quả 1: {1, 1, 1, 1, 1, 1, 1, 1, 1, 1}
- Kết quả 2: {1, 1, 1, 1, 1, 1, 1, 1, 2}
- Kết quả 3: {1, 1, 1, 1, 1, 1, 2, 2}
- Kết quả 4: {1, 1, 1, 1, 1, 5}
- Kết quả 5: {1, 1, 1, 1, 2, 2, 2}
- Kết quả 6: {1, 1, 1, 2, 5}
- Kết quả 7: {1, 1, 2, 2, 2, 2}
- Kết quả 8: {1, 2, 2, 5}
- Kết quả 9: {2, 2, 2, 2, 2}
- Kết quả 10: {5, 5}
- Kết quả 11: {10}

Source Code Coin Change Problem


```cpp
#include <stdio.h>
#include <iostream>
#include <vector>
using namespace std;
int coinChangeProblem(int total, int coins[], int n)
{
    vector<int> result(total + 1);
    result[0] = 1;
    for (int i = 0; i < n; i++)
    {
        for (int j = coins[i]; j <= total; j++)
        {
            result[j] +=  result[j - coins[i]];
        }
    }
    return result[total];
}
```

Source Code Coin Change Problem

```
16. void printSolution(vector<int> &result, int total,
                                          int coins[], int n, int pos)
{
17.     if (total == 0)
18.     {
19.         for (int r : result)
20.             cout << r << " ";
21.         cout << endl;
22.     }
23.     for (int i = pos; i < n; i++)
24.     {
25.         if (total >= coins[i])
26.         {
27.             result.push_back(coins[i]);
28.             printSolution(result, total - coins[i], coins, n, i);
29.             result.pop_back();
30.         }
31.     }
32. }
```

Source Code Coin Change Problem

```
31. int main()
32. {
33.     int total = 10;
34.     int coins[] = { 1, 2, 5, 10 };
35.     int n = 4;
36.     cout << "Number Of Solutions: ";
37.     cout << coinChangeProblem(total, coins, n) << endl;
38.     vector<int> result;
39.     printSolution(result, total, coins, n, 0);
40.     return 0;
41. }
```

Source Code Coin Change Problem

```
1.  def coinChangeProblem(total, coins, n):
2.      result = [0] * (total + 1)
3.      result[0] = 1
4.      for i in range(n):
5.          for j in range(coins[i], total + 1):
6.              result[j] += result[j - coins[i]]
7.      return result[total]
8.
9.  def printSolution(result, total, coins, n, pos):
10.     if total == 0:
11.         for r in result:
12.             print(r, end = ' ')
13.         print()
14.     for i in range(pos, n):
15.         if total >= coins[i]:
16.             result.append(coins[i])
17.             printSolution(result, total - coins[i], coins, n, i)
18.             result.pop()
```

Source Code Coin Change Problem

```
19. if __name__ == "__main__":
20.     total = 10
21.     coins = [1, 2, 5, 10]
22.     n = 4
23.     print("Number Of Solutions:", coinChangeProblem(total, coins, n))
24.     result = []
25.     printSolution(result, total, coins, n, 0)
```

Source Code Coin Change Problem

```java
import java.util.ArrayList;

public class Main {
    private static int coinChangeProblem(int total, int[] coins, int n) {
        int[] result = new int[total + 1];
        result[0] = 1;
        for (int i = 0; i < n; i++) {
            for (int j = coins[i]; j <= total; j++) {
                result[j] += result[j - coins[i]];
            }
        }
        return result[total];
    }
```

Source Code Coin Change Problem

```
14.     private static void printSolution(ArrayList<Integer> result,
                                          int total, int[] coins, int n, int pos) {
15.         if (total == 0) {
16.             for (int r : result) {
17.                 System.out.printf("%d ", r);
18.             }
19.             System.out.println();
20.         }
21.         for (int i = pos; i < n; i++) {
22.             if (total >= coins[i]) {
23.                 result.add(coins[i]);
24.                 printSolution(result, total - coins[i], coins, n, i);
25.                 result.remove(result.size() - 1);
26.             }
27.         }
28.     }
```

Source Code Coin Change Problem

```
29.     public static void main(String[] args) {
30.         int total = 10;
31.         int[] coins = new int[]{ 1, 2, 5, 10 };
32.         int n = 4;
33.         System.out.print("Number Of Solutions: ");
34.         System.out.printf("%d\n", coinChangeProblem(total, coins, n));
35.         ArrayList<Integer> result = new ArrayList<>();
36.         printSolution(result, total, coins, n, 0);
37.     }
38. }
```

Hỏi đáp