순열

202007003 임미현

순열

서로 다른 n개 중 r개를 뽑아 정렬하는 경우의 수(가능한 모든 배열의 합)
같은 배열은 같은 것으로, 다른 배열은 다른 것으로 판단한다.

계산

n부터 시작해서 하나씩 줄여가며 곱한다. 곱하는 수의 개수가 총 r개가 될 때까지 곱하는 것.
n!으로 요약할 수 있다.
시간복잡도 : O(n!) -> nPr이기 때문.

조합 vs 순열?

뽑은 숫자의 순서가 중요하느냐의 차이!

구현

가장 간단한 방법 : for문을 r번 중첩
단, 중복 배제 코드가 반드시 들어가야 한다.

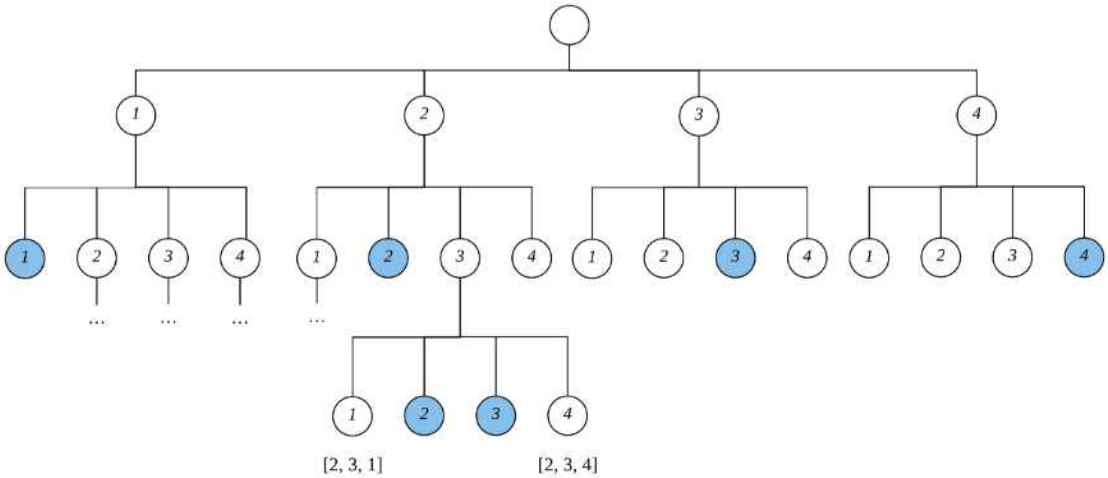
한줄 요약 : 0번부터 n번까지 순회하면서 사용하지 않은 숫자를 체크해 배열에 넣음.

중복 배제 방법 : boolean[] 사용 or 재귀로 구현

r이 커지면 일일이 적어야 하는 코드량이 많아지기 때문에 재귀로 구현하기도 한다

재귀함수 : 함수 정의 단계에서 자기 자신을 재참조하는 함수.

구현 예제를 트리로 표현



구현 예제

C++

예시) {'a', 'b', 'c', 'd'} 배열의 4P3 의 순열 출력하기

```cpp
#include <iostream>

using namespace std;

void swap(char & a, char & b)
{
    char temp = a;
    a = b;
    b = temp;
}

void permutation(char data [], int depth, int n, int r)
{
    if (depth == r)
    {
        for(int i = 0; i < r; i++)
            cout << data[i] << " ";
        cout << endl;

        return;
    }

    for(int i = depth; i < n; i++)
    {
        swap(data[depth], data[i]);   // 스왑
        permutation(data, depth + 1, n, r);  // ★재귀
        swap(data[depth], data[i]);  // 다시 원래 위치로 되돌리기
    }
}
```

```cpp
int main()
{
    char arr [] = {'a', 'b', 'c', 'd'};

    permutation(arr, 0, 4, 3); // 4P3

    return 0;
}
```

💎출력💎

```
a b c
a b d
a c b
a c d
a d c
a d b
```

C++(2)

-집합의 원소 번호를 order 배열에 순열의 형태로 저장하고 출력 시 arr 배열의 인덱싱 용도로 사용.

재귀함수를 사용하는 코드

```cpp
#include <iostream>
using namespace std;

int arr[10] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
int cnt, n, r;
int visited[4];
int order[4];

void permutation(int now)
{
	if (now == r)
	{
		for (int i = 0; i < r; i++)
		{
			cout << arr[order[i]] << " ";
		}
		cout << endl;
		cnt++;

		return;
	}

	for (int i = 0; i < n; i++)
	{
		if (visited[i])
			continue;

		visited[i] = true;
		order[now] = i;
		permutation(now + 1);
		visited[i] = false;
	}
}
int main()
{
	cout << "n : ";
	cin >> n;
	cout << "r : ";
	cin >> r;
	cout << n << 'P' << r << endl;

	permutation(0);
	cout << "순열의 개수 : " << cnt;
}
```

JAVA

```java
import java.util.*;

public class Main {
	static int[] arr;
	static boolean[] check;
	static Stack<Integer> st;
	public static void main(String[] args) {
		System.out.println("========순열========");
		st = new Stack<>();
		arr = new int[] {1,2,3};
		check = new boolean[3];
		permutation(3, 2);
	}

	//순열
	static void permutation(int n, int r) {
		if(st.size() == r){
			for(int i : st){
				System.out.print(i+" ");
			}
			System.out.println();
			return;
		}

		for(int i=0; i<n; i++){
			if(!check[i]){
				check[i] = true;
				st.push(arr[i]);
				permutation(n, r);
				st.pop();
				check[i] = false;
			}
		}
	}
}
```

```
========순열========
1 2
1 3
2 1
2 3
3 1
3 2
```

중복 순열

서로 다른 n개에서 중복이 가능하게 r개를 뽑아서 정렬하는 경우의 수
순열과 달리 중복이 가능하다는 차이점만 존재!

JAVA 구현 예제

```java
public class AlgorithmStudy {
    public static void permutation(int[] arr, int[] out, int depth, int r){
        if(depth == r){
            for(int num: out) System.out.print(num);
            System.out.println();
            return;
        }
        for(int i=0; i<arr.length; i++){
            out[depth] = arr[i];
            permutation(arr, out, depth+1, r);
        }
    }

    public static void main(String[] args){
        int[] arr = {1, 2, 3};
        int r = 2;
        permutation(arr, new int[r], 0, r);
    }
}
```

순열 코드에서 중복 방지 부분만 제거해주면 됨.

수식 표현

-선택지를 동시에 여러 개 골라야 하므로 곱의 법칙이 사용된다.

순열

$$ {}_nP_r = n(n-1)(n-2)\cdots(n-r+1) $$

$$ = \frac{n!}{(n-r)!} $$

중복 순열

$$ {}_n\Pi_r = n^r $$

next_permutation

-n개 원소의 순열을 구할 수 있는 함수

-C++의 algorithm 헤더에 있음

형식

bool next_permutation(배열의 시작, 배열의 끝)

주의점

-오름차순으로 정렬된 값을 가진 배열로만 사용 가능

-오름차순으로 순열 생성

next_permutation 사용예시 : {1, 2, 3}의 모든 순열 출력하기

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main() {
    vector<int> v{ 1, 2, 3};

    sort(v.begin(), v.end());

    do {
        for (auto it = v.begin(); it != v.end(); ++it)
            cout << *it << ' ';
        cout << endl;
    } while (next_permutation(v.begin(), v.end()));
}
```



```
1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1
```

```cpp
#include <algorithm>
#include <string>
#include <iostream>

int main()
{
    std::string s = "aba";
    std::sort(s.begin(), s.end());
    do {
        std::cout << s << '\n';
    } while(std::next_permutation(s.begin(), s.end()));
}

/* OUTPUT */
/* aab    */
/* aba    */
/* baa    */
```

함수 구조

```cpp
template<class BidirIt>
bool next_permutation(BidirIt first, BidirIt last)
{
    if (first == last) return false;
    BidirIt i = last;
    if (first == --i) return false;

    while (true) {
        BidirIt i1, i2;

        i1 = i;
        if (*--i < *i1) {
            i2 = last;
            while (!(*i < *--i2))
                ;
            std::iter_swap(i, i2);
            std::reverse(i1, last);
            return true;
        }
        if (i == first) {
            std::reverse(first, last);
            return false;
        }
    }
}
```