3주차 - 2

The Coding Test Academic
C o n f e r e n c e

코딩테스트 학술회

INDEX

2. 순열

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

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

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

using namespace std;


void printP(vector<int> &v){
    for(int i=0; i<v.size(); i++){
        cout << v[i] << " ";
    }
    cout <<"\n";
}

int main(){
    int a[3] = {1,2,3};
    vector<int> v;
    //오름차순
    for(int i =0; i< 3; i++)v.push_back(a[i]);
    do{
        printP(v);
    }while(next_permutation(v.begin(), v.end()));

    v.clear();

    //내림차순

    for(int i = 2; i >= 0; i--)v.push_back(a[i]);
    do{
        printP(v);
    }while(prev_permutation(v.begin(), v.end()));
    return 0;
}
```

A 순열

nPr : 서로 다른 n개, 중복을 허락하지 않고 r개를 일렬로 나열하는 수(뽑는 수)

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

using namespace std;

int a[3] = {1, 2, 3};
vector<int> v;

void printP(vector<int> &v){
    for(int i=0; i<v.size(); i++){
        cout << v[i] << " ";
    }
    cout <<"\n";
}

void makePermutation(int n, int r, int depth){
    if(r == depth){
        printP(v);
        return;
    }
    for(int i = depth; i < n; i++){
        swap(v[i], v[depth]);
        makePermutation(n, r, depth + 1);
        swap(v[i], v[depth]);
    }
    return;
}

int main(){
    for(int i =0; i< 3; i++)v.push_back(a[i]);
    makePermutation(3, 3, 0);
    return 0;
}
```

2. 조합

$${}_nC_r = \text{서로 다른 n 개 중에서 겹치지 않도록 r 개를 선택하는 방법의 수}$$

$${}_nC_r = \frac{\text{뽑아서 줄세우기}}{\text{줄세우기}}$$

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

n! : 전체 / r! : 뽑힌것 / (n-r)! : 안뽑힌것

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

int n = 4;
int r = 2;

int a[4] = {1, 2, 3, 4};
vector<int> b;

void print(vector<int> b){
for(int i = 0; i < b.size(); i++){
    cout << b[i] << " ";
    }
    cout << endl;
}

void combi(int start, vector<int> b){
    if(b.size() == r){
        print(b);
        return;
    }

    for(int i = start + 1; i < n; i++){
        b.push_back(i);
        combi(i, b);
        b.pop_back();
    }
    return;
}

int main() {
    combi(-1, b);
    return 0;
}
```

A 조합

n개 중 순서 없이 r개 뽑는 수

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

int n = 4;
int r = 3;
int a[4] = {1, 2, 3, 4};

int main() {
    for(int i = 0; i < n; i++){
        for(int j = 0; j < i; j++){
            for(int k = 0; k < j; k++){
                cout << i << " " << j << " " << k << '\n';
            }
        }
    }

    return 0;
}
```

3. 자료구조 - stack

A stack

LIFO(Last In First Out)

삽입, 삭제: O(1), 탐색: O(n)

<stack>

-.push() : 괄호 값을 스택에 넣어줌
-.pop() : 스택 가장 위의 값을 삭제
-.top() : 스택 가장 위의 값을 리턴
-.empty(): 비었을 때 true 리턴
-.size(): 스택의 길이 리턴

https://www.acmicpc.net/problem/10828

3. 자료구조 - 큐

A queue

FIFO(First In First Out)

삽입, 삭제: O(1), 탐색: O(n)

<queue>

-.push() : 괄호 값을 큐에 넣어줌
-.pop() : 큐의 front 데이터 값을 삭제
-.front() : 큐의 front 데이터 반환
-.back(): 큐의 back 데이터 반환
-.size(): 큐의 길이 리턴
-.empty(): 비었으면 true 리턴

https://www.acmicpc.net/problem/18258

3. 자료구조 - 덱

A deque

앞뒤로 참조 가능한 자료 구조

삽입, 삭제: O(1), 탐색: O(1)

<deque>

-.push_front() : 괄호 값을 front에 삽입
-.push_back(): 괄호 값을 뒤에 삽입
-.pop_front() : front 값 삭제
-.pop_back() : rear 값 삭제
-.front, .back(): front, back 값 리턴-
.pop_front() : front 값 삭제

-.empty(): 비었을 때 true 리턴
-.size(): 스택의 길이 리턴

https://www.acmicpc.net/problem/10866

3. 자료구조 – 시간 복잡도 정리

Common Data Structure Operations

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

3. 자료구조 – 우선순위 큐

A Priority queue

내부 구조 heap으로 구현
삽입 시, 우선순위에 맞춰 정렬 되고,
삭제 시, 정렬된 큐의 앞에서 이뤄짐

정렬: O(log n)

<priority queue>

-선언: priority_queue<자료형, container, 비교 함수> 이름;
-push, top() …

<Heap>

- 최대값 또는 최소값을 빠르게 구하기 위한 자료 구조이자, 완전 이진 트리를 기본으로 한 자료구조

https://www.acmicpc.net/problem/9934