조합

조합(Combination)이란?

: n개의 값 중에서 r 개의 숫자를 순서를 고려하지 않고 나열한 경우의 수

- 순서를 따지지 않음 → (1,2) = (2,1)
- 중복을 허용하지 않음

ex) 1,2,3 중 2개 뽑기

순열	조합
(1,2) (1,3) (2,1) (2,3) (3,1) (3,2)	(1,2) = (2,1) (3,2) = (2,3) (3,1) = (1,3)
→ 6개	→ 3개

조합 수식

${}_nC_r$	${}_nC_r = {}_{n-1}C_{r-1} + {}_{n-1}C_r$

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

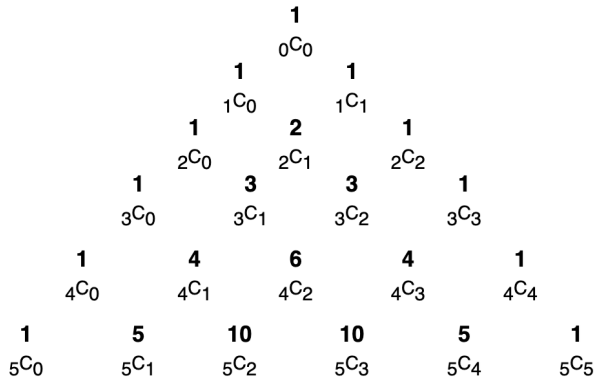
ex) 1,2,3 중 2개 뽑기 → ${}_3C_2$

순열	조합
(1,2) (1,3) (2,1) (2,3) (3,1) (3,2)	(1,2) = (2,1) (3,2) = (2,3) (3,1) = (1,3)

파스칼의 삼각형 공식

점화식 : 어떤 함수를 자신과 똑같은 함수를 이용해서 나타내는 것

→ 즉, 재귀함수를 갖는 알고리즘 만 점화식으로 표현 할 수 있음



재귀함수 : 함수내에서 자기 자신을 호출하여 반복되는 함수

for문으로 구현

nCr → 원소 n개 중에서 r개를 뽑는 모든 경우(순서 X)

원소 n개가 들어 있는 배열에 대해 r개의 for문을 통해 원소를 선택하는 과정을 구현

ex) $_{12}C_3$

```
int list[12] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
for(int i=0; i<=11; i++) {
   for(int j=i+1; j<=11; j++) {                    // 중복된 경우는 처리할 필요가 없으니, j는 i+1부터
      for(int k=j+1; k<=11; k++) {                 // 중복된 경우는 처리할 필요가 없으니, k는 j+1부터
                  // {i,j,k} 를 선택
      }
   }
}
```

재귀함수로 구현

: 함수내에서 자기 자신을 호출하여 반복되는 함수

nCr = n-1Cr-1 + n-1Cr → 뽑은 경우와 뽑지 않은 경우로 나뉨

```
int combination(int n, int r)

{
        nCn             nC0
    if(n == r || r == 0)

        return 1;

    else

        return combination(n - 1, r - 1) + combination(n - 1, r);

  }
```

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

using namespace std;

void Combination(vector<char> arr, vector<char> comb, int r, int index, int depth)
{
    if (r == 0)                                             // 다 뽑았을 때
    {
        for(int i = 0; i < comb.size(); i++)
            cout << comb[i] << endl;
    }
    else if (depth == arr.size())               // r이 0 이 되지 않은 경우 → 조합의 경우들 중 하나가 될 수 없는 케이스
    {
        return;
    }
    else
    {
        // arr[depth] 를 뽑은 경우
        comb[index] = arr[depth];
        Combination(arr, comb, r - 1, index + 1, depth + 1);

        // arr[depth] 를 뽑지 않은 경우
        Combination(arr, comb, r, index, depth + 1);
    }
}
```

```
int main()
{
    vector<char> vec = {'a', 'b', 'c', 'd', 'e'};  // n = 5

    int r = 3;
    vector<char> comb(r);

    Combination(vec, comb, r, 0, 0);  // {'a', 'b', 'c', 'd', 'e'}의 '5C3' 구하기

    return 0;
   }
```

추가

STL을 이용한 조합

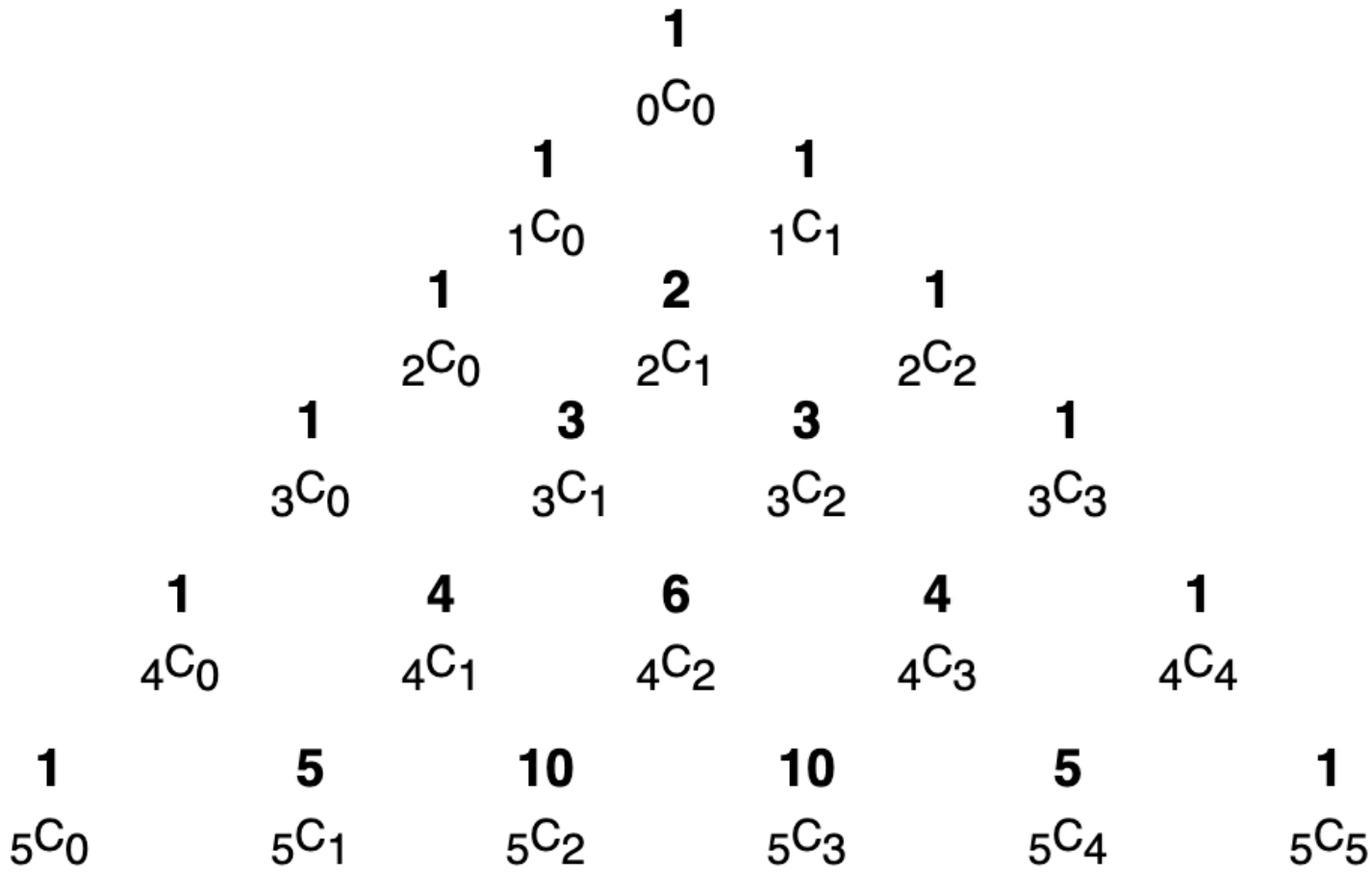
https://ansohxxn.github.io/algorithm/combination/

백트래킹을 이용한 조합

https://velog.io/@soyeon207/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98-%EC%A1%B0%ED%95%A9-Combination

DFS를 활용한 재귀문

https://velog.io/@sjoonb/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98-%EC%A1%B0%ED%95%A9-%EC%A4%91%EC%B2%A9-%EB%B0%98%EB%B3%B5%EB%AC%B8-%EB%8C%80%EC%B2%B4%ED%95%98%EA%B8%B0


1
$_0C_0$
1 1
$_1C_0$ $_1C_1$
1 2 1
$_2C_0$ $_2C_1$ $_2C_2$
1 3 3 1
$_3C_0$ $_3C_1$ $_3C_2$ $_3C_3$
1 4 6 4 1
$_4C_0$ $_4C_1$ $_4C_2$ $_4C_3$ $_4C_4$
1 5 10 10 5 1
$_5C_0$ $_5C_1$ $_5C_2$ $_5C_3$ $_5C_4$ $_5C_5$

중복 조합

중복 조합(combination with repetition)이란?

: 서로 다른 n개에서 중복을 허락하여 r개를 택하는 조합

조합(nCr)	중복 조합(nHr)
• 순서를 따지지 않음 → (1,2) = (2,1) • 중복을 허용하지 않음 ex) 1,2,3 중 2개 뽑기 (1,2) (1,3) (2,3) → 3개	• 순서를 따지지 않음 → (1,2) = (2,1) • 중복을 허용함 ex) 1,2,3 중 2개 뽑기 (1,1) (1,2) (1,3) (2,2) (2,3) → 5개 (3,3)

중복 조합 수식(nHr = n+r-1Cr)

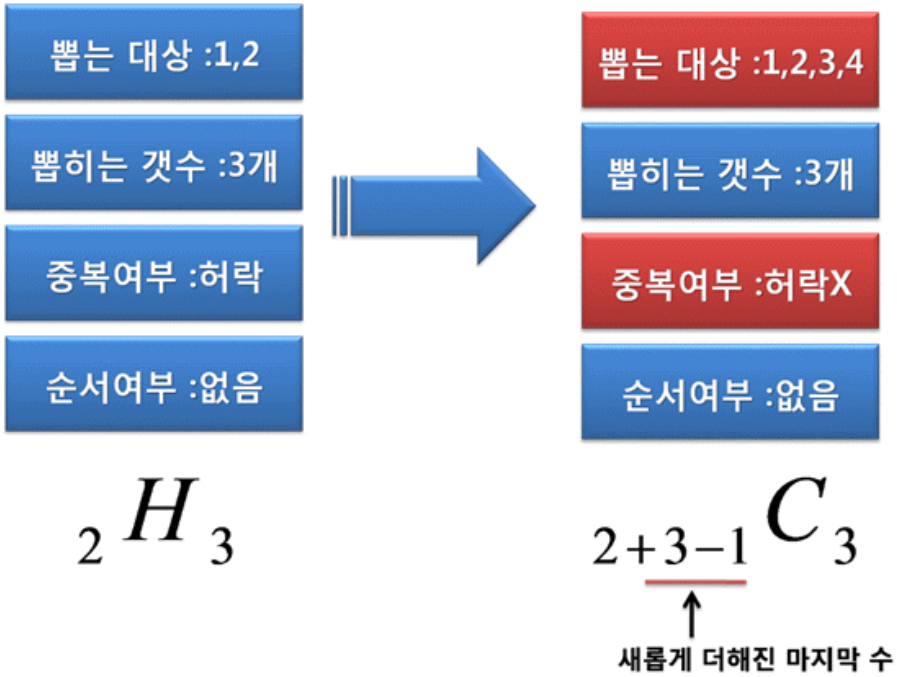
1,2중 중복을 허용하여 3개 뽑기 (2H3)

(111) (112) (122) (222)

+0 +1 +2

(123) (124) (134) (234)

nHr = n+r-1Cr



$${}_2H_3$$

$${}_{2+3-1}C_3$$

재귀 함수로 구현

: 조합에서 쓰인 nCr = n-1Cr-1 + n-1Cr 이용

comb[index] = arr[depth];

Combination(arr, comb, r - 1, index + 1, depth + 1); // depth + 1 → 조합

Combination(arr, comb, r, index, depth + 1);

: 중복조합에서 쓰인 nHr = nHr-1 + n-1Hr 이용

comb[index] = arr[depth];

Combination(arr, comb, r - 1, index + 1, depth); // depth → 중복조합

Combination(arr, comb, r, index, depth + 1);

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

void Combination(vector<char> arr, vector<char> comb, int r, int index, int depth)
{
    if (r == 0)
    {
        for(int i = 0; i < comb.size(); i++)
            cout << comb[i] << " ";
        cout << endl;
    }
    else if (depth == arr.size())                    // depth == n
    {
        return;
    }
    else
    {
        comb[index] = arr[depth];
        Combination(arr, comb, r - 1, index + 1, depth);
          // arr[depth]를 뽑기로 결정. (중복 조합이므로 다음에 또 arr[depth ] 뽑을 수 있음)
        Combination(arr, comb, r, index, depth + 1); // arr[depth]를 뽑지 않기로 결정. 이제 arr[depth + 1]
    }
   }
```

```
int main()
{
    vector<char> vec = {'a', 'b', 'c'}; // n = 3

    int r = 2;
    vector<char> comb(r);

    Combination(vec, comb, r, 0, 0);  // {'a', 'b', 'c'}의 중복조합 '3H2' 구하기

    return 0;
   }
```

순열/조합

함수를 이용 순열

STL에 algorithm 헤더파일을 추가하면 함수를 통해서 순열을 구할수 있음

#include <algorithm>

함수에 벡터의 iterator 혹은 배열의 주소를 넣으면

다음 순열(1-2-3-4의 다음 순열은 1-2-4-3) 혹은 이전 순열(1-2-4-3의 이전 순열은 1-2-3-4)의 결과가 벡터나 배열에 적용 됨

- next_permutation

- prev_permutation

** 주의사항 **

원소들은 오름차순/내림차순으로 정렬되어있어야 함

보통 sort() 이용해 정렬 후 사용 함

vector<int> v{1,2,3,4}; -> OK

vector<int> v{1,3,10,4}; -> NO

next_permutation 함수

: 현재 나와 있는 수열에서 인자로 넘어간 범위에 해당하는 다음 순열을 구하고 true를 반환 함

다음 순열이 없다면(다음에 나온 순열이 순서상 이전 순열보다 작다면) false를 반환 함

// 첫번째 인자가 구하고자 하는 순열의 시작, 두번째 인자가 순열의 끝

bool next_permutation (BidirectionalIterator first, BidirectionalIterator last);

// 아래처럼 직접 비교함수를 넣어줘도 됨

bool next_permutation (BidirectionalIterator first, BidirectionalIterator last, Compare comp);

** 원소들은 오름차순으로 정렬 되어 있어야 함

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

using namespace std;

int main() {

		// 1부터 4까지 저장할 벡터 선언
		vector<int> v(4);

		// 1부터 4까지 벡터에 저장(오름차순)
		for (int i = 0; i < 4; i++) {
			v[i] = i + 1;
		}

		// next_permutation을 통해서 다음 순열 구하기
		do {
			for (int i = 0; i < 4; i++) {
				cout << v[i] << " ";
			}
			cout << '\n';
	} while (next_permutation(v.begin(), v.end()));

		return 0;
}
```

[구현결과]

```
1 2 3 4
1 2 4 3
1 3 2 4

...

4 2 3 1
4 3 1 2

4 3 2 1
```

prev_permutation 함수

: 현재 나와 있는 수열에서 인자로 넘어간 범위에 해당하는 이전 순열을 구하고 true를 반환 함

이전 순열이 없다면(다음에 나온 순열이 순서상 이전 순열보다 크다면) false를 반환 함

// 첫번째 인자가 구하고자 하는 순열의 시작, 두번째 인자가 순열의 끝

```
bool prev_permutation (BidirectionalIterator first, BidirectionalIterator last);
```

// 아래처럼 직접 비교함수를 넣어줘도 됨

```
bool prev_permutation (BidirectionalIterator first, BidirectionalIterator last, Compare comp);
```

** 원소들은 내림차순으로 정렬 되어 있어야 함

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

using namespace std;

int main() {

		// 1부터 4까지 저장할 벡터 선언
		vector<int> v(4);

		// 4부터 1까지 벡터에 저장(내림차순)
		for(int i=0; i<4; i++){
			v[i] = 4-i;
		}

	// prev_permutation을 통해서 이전 순열 구하기
		do{
			for(int i=0; i<4; i++){
				cout << v[i] << " ";
			}
			cout << '\n';
	}while(prev_permutation(v.begin(),v.end()));

		return 0;

}
```

[구현결과]

```
4 3 2 1
4 3 1 2
4 2 3 1

…

1 3 2 4
1 2 4 3
1 2 3 4
```

함수를 이용한 조합

- next_permutation

- prev_permutation

중복이 있는 원소들은 중복을 제외하고 순열을 만들어 줌

{0, 0, 1}과 같은 배열의 순열을 구한다면 중복을 제외한 {0, 0, 1}, {0, 1, 0}, {1, 0, 0}이 됨

→ 이를 이용해 조합(Combination)을 구할 수 있음

[방법]

전체 n개의 원소들 중에서 k개를 뽑는 조합(=nCk)을 구한다면 n개의 벡터 원소에 1을 k개 0을 나머지인 n-k개 집어넣어서 순열을 돌리고 1에 해당하는 인덱스만 가져오면 됨

next_permutation 함수

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

int main() {
		vector<int> s{ 1,2,3,4 };
		vector<int>temp{ 0,0,1,1};  //오름차순
		do {
			for (int i = 0; i < s.size(); i++) {
				if (temp[i] == 1)
					cout << s[i] << ' ';
			}
			cout << endl;
		} while (next_permutation(temp.begin(), temp.end()));
}
```

[구현 결과]

3 4	0011
2 4	0101
2 3	0110
1 4	1001
1 3	1010
1 2	1100

4C2

prev_permutation 함수

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

int main() {
		vector<int> s{ 1,2,3,4 };
		vector<int>temp{ 1,1,0,0};  //내림차순
		do {
			for (int i = 0; i < s.size(); i++) {
				if (temp[i] == 1)
					cout << s[i] << ' ';
			}
			cout << endl;
		} while (prev_permutation(temp.begin(), temp.end()));
}
```

[구현 결과]

1 2	1100
1 3	1010
1 4	1001
2 3	0110
2 4	0101
3 4	0011

4C2

끝