뽑기(순환)

pick 함수 revisited

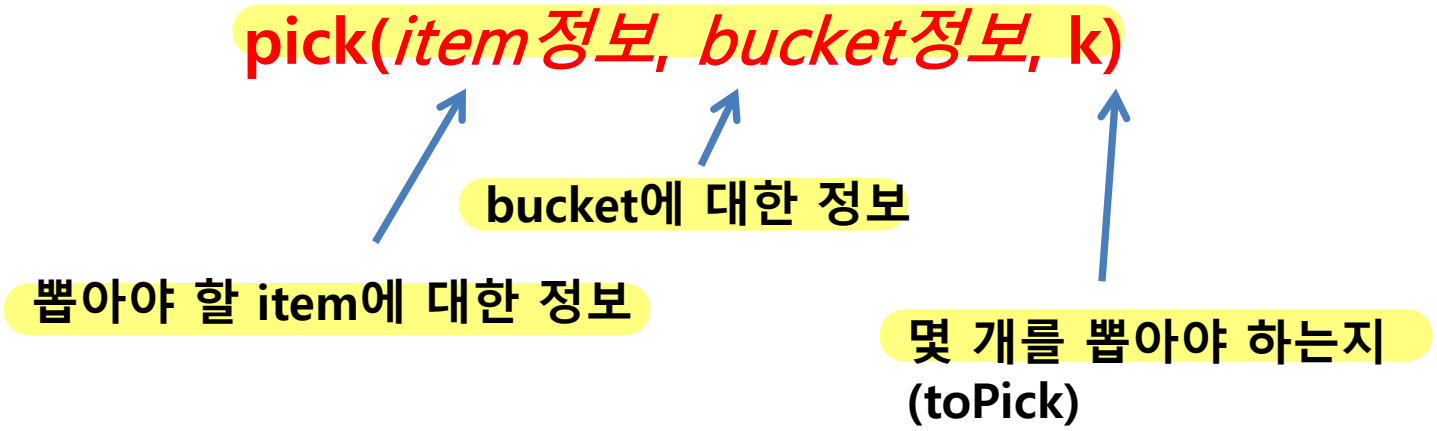
문제

- n 개의 item에서 m개를 뽑고자 할 때 가능한 모든 방법을 출력하는 프로그램을 작성하시오.
  - 뽑기 방법
    - 중복 여부
    - 순서 고려
  - 조합(combination) : 순서와 상관없음 (1. 2) = (2. 1)
  - 순열(permutation) : 순서에 상관있음 (1. 2) ≠ (2. 1)
  - 중복(with repetition) : 같은 item을 여러 번 뽑을 수 있음
    - 중복조합 (1. 1. 2) = (1. 2. 1) = (2. 1. 1)
    - 중복순열 (1. 1. 2) ≠ (1. 2. 1) ≠ (2. 1. 1)

문제의 해결 방법(I)

1. m개를 뽑아서 담을 수 있을 공간을 미리 할당 (bucket)

2. 구현하고자 하는 함수는 모양은 아래와 같다.



문제의 해결 방법(II)

3. pick 함수는 다음과 같이 구현이 될 수 있음.
   - k : 앞으로 뽑아야 할 크기
   - Trivial case (if k = 0) → 앞으로 더 뽑을 것이 없음.
     - m개를 뽑아야 할 문제인데 이미 m개를 다 뽑은 경우.
     - 적절한 일을 (printf) 해주고 return 한다. 무한 호출을 막아준다.
   - Recursive case (if k > 0) → k개를 더 뽑아야 한다.
     - **앞으로 k개를 뽑아야 하므로 일단 1개를 뽑고 같은 함수를 이용하여 k-1개를 더 뽑는다.**
     - 1개를 뽑는 방법은 순열/조합/중복순열/중복조합이냐에 따라 다르다. (★ 요거.)

call pick(*item정보, bucket정보,* k-1) → k=0일 때까지 재귀 호출

함수의 큰 모양

```
pick(   int* items, itemSize
        int* bucket, int bucketSize,
        int k)
{
        if ( k == 0 ) // trivial case
        {
                //적당한 일
                return;
        }
        // k > 0
        for item from candidate items
        {
                buckets[새로 뽑을 곳] = item;
                pick( items, bucket, bucketSize, k-1 );
        }
}
```

item 정보

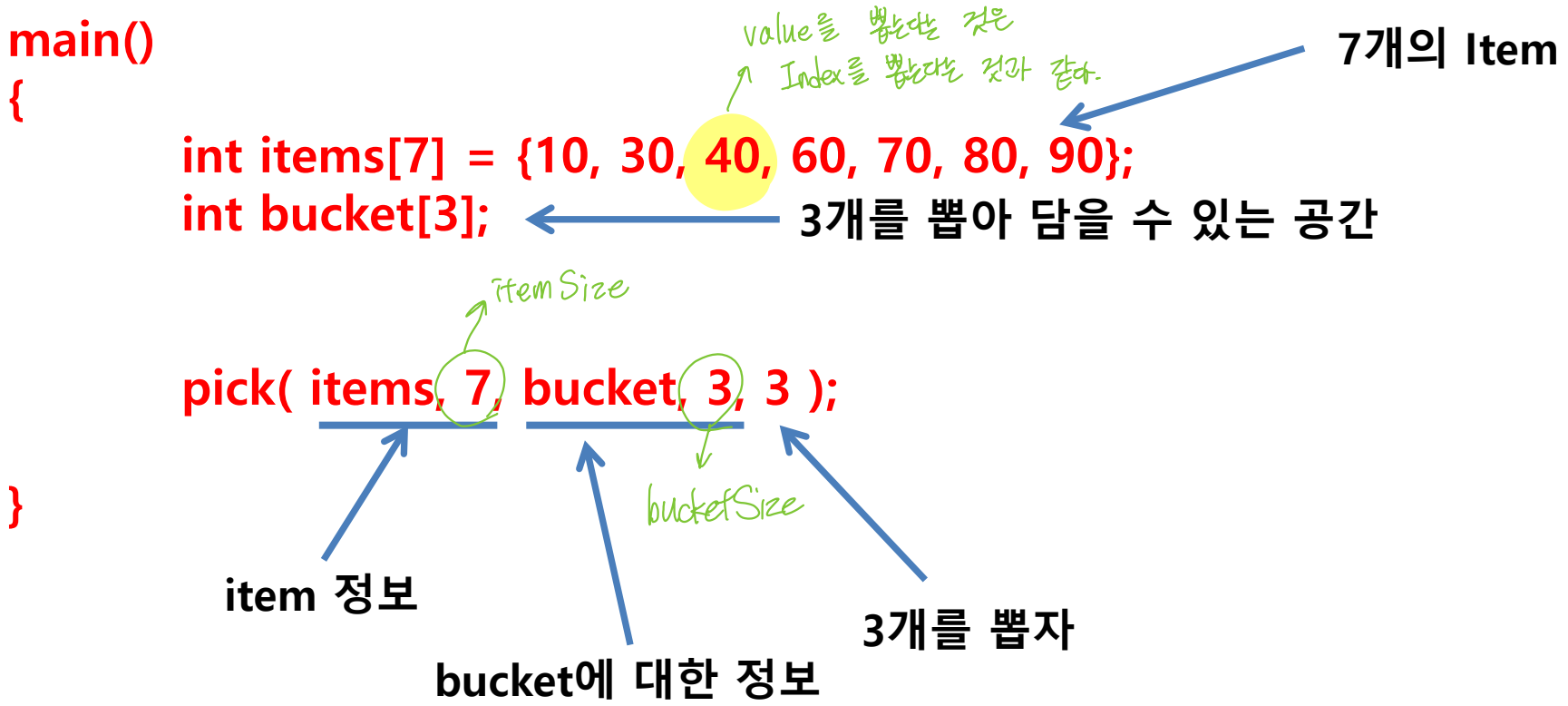
bucket에 대한 정보

몇 개를 뽑아야 하는지

후보 item 중에서 1개를 뽑는다

같은 함수를 이용하여 k-1개를 뽑는다.

Solution의 큰 모양



7개 중에서 3개를 뽑는다!!

#1 조합(Combination)

중복조합(Combination with Repetition)
중복순열(Permutation with Repetition)
순열(Permutation)

#1 조합(Combination)

Let's go back to the power of recursion!!

- 조합 출력하기
  - 0부터 차례대로 번호가 매겨진 n개의 원소 중에서 4개를 골라 출력하는 코드를 작성하시오.
  - item은 0,1,2,3,…,n-1
    (배열을 이용하여 어떤 값을 담을 수 있겠으나,
    예: int items[7] = {**10, 30, 40, 60, 70, 80, 90**}
    인 경우 item정보는
    items와 그 크기인 7이 된다.

    여기서는 문제를 단순화 하기 위하여, item을 int형의 수로 하여 0부터 (그 수-1)까지로 한다. 즉, item정보는 item만으로 표현한다.

  - bucket은 숫자 4개를 담을 수 있는 int형 배열이면 충분.

#1 조합(Combination)

Solution의 큰 모양

```
main()
{
        int n = 8;
        int bucket[4];


        pick( n, bucket, 4, 4 );
}
```

8개의 item (0,1,2,..7을 내포함)

4개를 뽑아 담을 수 있는 공간

item 정보

bucket에 대한 정보

4개를 뽑자

#1 조합(Combination)

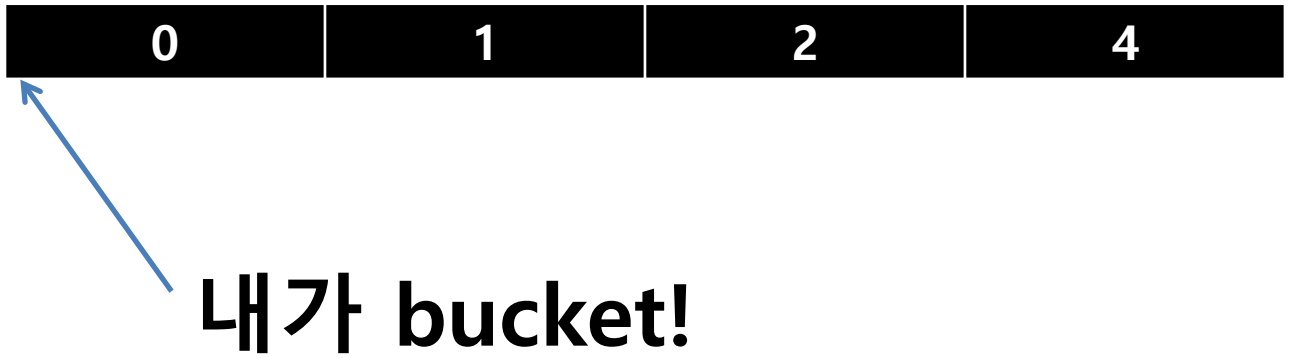
Background of 조합(Combination)

- 조합 : 경우의 수에서 순서를 고려하지 않는 경우
  - 예) 0,1,2, .. 8 에서 3개의 숫자를 조합으로 뽑는 경우.
    - 0,1,2와 1,2,0 그리고 1,0,2 가 한 가지 경우이다.

- 순열 : 순서를 고려하는 경우
  - 예) 0,1,2, .. 8 에서 3개의 숫자를 순열로 뽑는 경우.
    - 0,1,2와 1,2,0 그리고 1,0,2 가 모두 다른 경우이고 별개로 고려 되어야 한다.

Tips for 조합(Combination)

- 뽑을 때 오름차순 (혹은 내림차순) 으로 뽑자.
  - 4개를 뽑는 경우에 항상 오름차순으로 뽑아서 1,0,4,2 로 뽑는 경우를 원천적으로 막자.



함수의 큰 모양

Item 정보

bucket에 대한 정보

몇 개를 뽑아야 하는지

```
pick(   int n,
        int* bucket, int bucketSize,
        int k)
{
        if ( k == 0 ) // trivial case
        {
                //적당한 일 → print bucket (loop를 쓰면 된다)
                return;
        }
        for item from candidate items
        {
                buckets[새로뽑을곳] = item;
                pick( items, bucket, bucketSize, k-1 );
        }
}
```

☆ 오름차순으로 뽑기.

#1 조합(Combination)

Trivial Case

```
if(k == 0 ) {
        for (i = 0; i < bucketSize; i++)
                printf("%d ", bucket[i]);
        printf("₩n");
        return;
}
```

index가 출력됨.

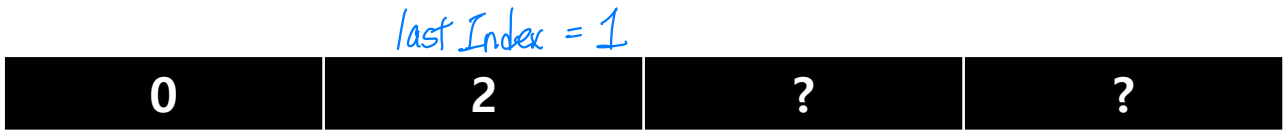
만약에, int item[8] = { 10, 20, 30, 40, ... 80 } 이 전역 변수로 저장되어 있다면,

printf(" %d ", item[bucket[i]]);

index 번호가 아닌 실제 item value를 출력할 수 있다.

#1 조합(Combination)

Recursive Case




```
pick(   int n, (0,1,2,...7)
        int* bucket, int bucketSize (4),
        int k (2))
{
        //... ← trivial case가 여기에

        //조합이기 때문에 가장 마지막에 뽑힌 수보다 큰 수를 뽑는다.
        //마지막에 뽑힌 수는 어디에? lastIndex가 가리키는 곳에 (1)
        // lastIndex = bucketSize – k – 1;
        for item from candidate items (3,4,5,6,7)
        {
                bucket[lastIndex+1] = item;
                pick( items, bucket, bucketSize, k-1 );
        }
}
```

bucketsize: 총 몇 개를 뽑을 것인가?
k: 앞으로 뽑을 것이 몇 개 남았나!
lastIndex: 인덱스 변수이므로 -1.

lastIndex의 다음 칸에는.

그 다음의 item이 한번씩 들어온다.

#1 조합(Combination)

Recursive Case

```
for item from candidate items (3,4,5,6,7)
{
        buckets[lastIndex+1] = item;
        pick( items, bucket, bucketSize, k-1 );
}
```

buckets[2] = 각각의 item이 들어감.

lastIndex ← 1

lastIndex = 2.

0	2	3	?	pick(items, bucket, bucketSize, 1);
0	2	4	?	pick(items, bucket, bucketSize, 1);
0	2	5	?	pick(items, bucket, bucketSize, 1);
0	2	6	?	pick(items, bucket, bucketSize, 1);
0	2	7	?	pick(items, bucket, bucketSize, 1);

#1 조합(Combination)

Recursive Case

```
lastIndex = bucketSize - k - 1; // 가장 최근에 뽑힌 수가 저장된 위치 index

if (bucket_size == k ) → 아직 하나도 뽑지 않았다.
        smallest = 0; → 0 부터 오름차순으로 뽑는다.
else
        smallest = bucket[lastIndex] + 1; → 마지막 bucket 안의 value 보다 1 큰 값을 smallest 로 정함.

for(item = smallest; item < n; item++) // candidate items
{                                       ↳ smallest ... n 까지 모두 후보 item 이 된다.
        bucket[lastIndex + 1] = item;
        pick(n, bucket, bucketSize, k - 1);
}
```

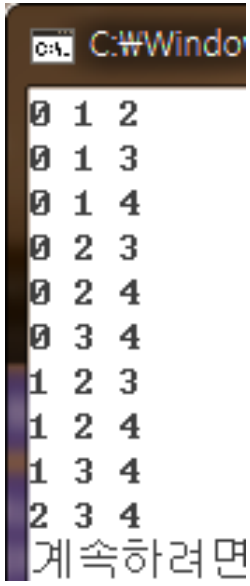
#1 조합(Combination)

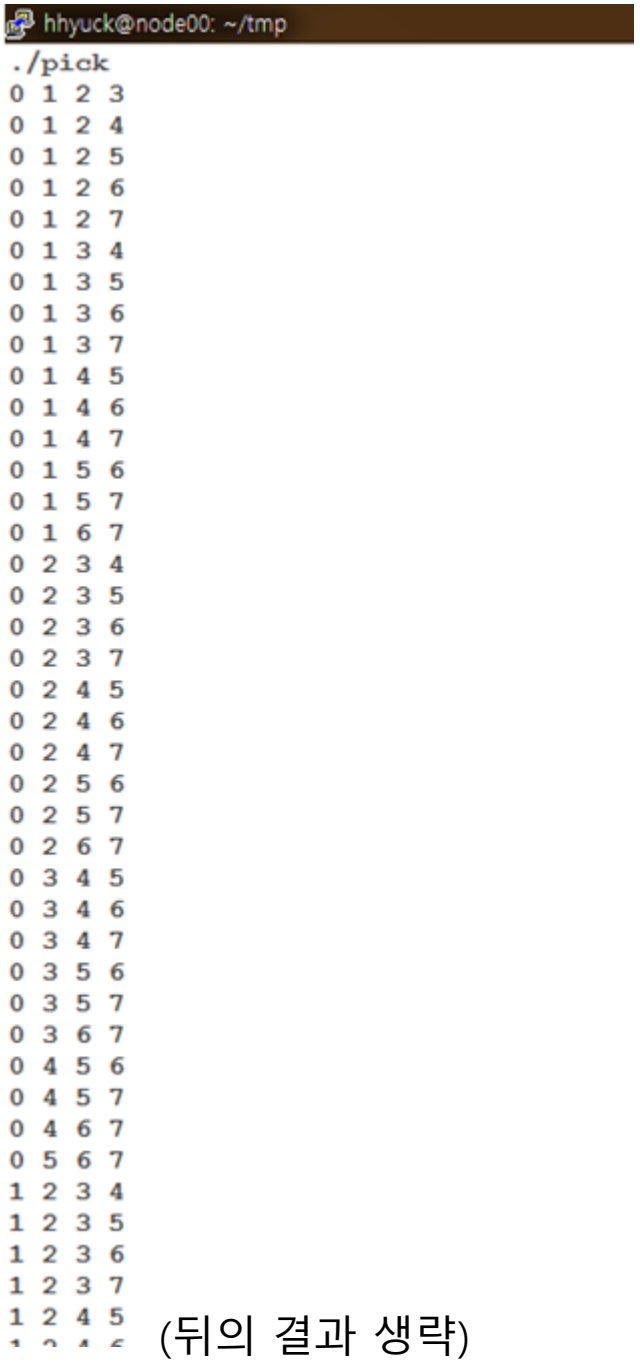
```
void pick( int n, int* bucket, int bucketSize, int k ) {

        int i, lastIndex, smallest, item;
        if(k == 0 ) {
                for (i = 0; i < bucketSize; i++)
                        printf("%d ", bucket[i]);
                printf("₩n");
                return;
        }

        lastIndex = bucketSize - k - 1; // 가장 최근에 뽑힌 수가 저장된 위치 index

        if (bucketSize == k )
                smallest = 0;
        else
                smallest = bucket[lastIndex] + 1;

        for(item = smallest; item < n; item++) {
                bucket[lastIndex + 1] = item;
                pick(n, bucket, bucketSize, k - 1);
        }
}
```

#1 조합(Combination)

```
main()
{
    int n = 8;
    int bucket[4];

    pick( n, bucket, 4, 4 );
}
```

```
main()
{
    int n = 5;
    int bucket[3];

    pick( n, bucket, 3, 3 );
}
```





(뒤의 결과 생략)

조합(Combination)

#2 중복조합(Combination with Repetition)

중복순열(Permutation with Repetition)

순열(Permutation)

#2 중복조합

Combination with Repetition

- 중복조합
  - 논리 흐름은 Combination과 같다.
  - 다른 점은 bucket에 새로운 item을 뽑을 때 오름차순(or 내림차순)인데, 같은 것도 뽑을 수 있게 한다.
  - 즉, combination에서는 마지막에 뽑은 item보다 큰 것을 뽑았지만
  - **Combination with repetition 에서는 크거나 같은 것을 뽑는다.** → 오름차순으로 뽑지만, 마지막에 뽑은 것을 또 뽑을 수 있다.

#2 중복조합

```
void pick( int n, int* bucket, int bucketSize, int k ) { //중복 조합

        int i, lastIndex, smallest, item;
        if(k == 0 ) {
                for (i = 0; i < bucket_size; i++)
                        printf("%d ", bucket[i]);
                printf("₩n");
                return;
        }

        lastIndex = bucketSize - k - 1; // 가장 최근에 뽑힌 수가 저장된 위치 index

        if (bucketSize == k )
                smallest = 0;
        else
                smallest = bucket[lastIndex]; // 이부분만 combination과 차이

        for(item = smallest; item < n; item++) {
                bucket[lastIndex + 1] = item;
                pick(n, bucket, bucketSize, k – 1);
        }

}
```

↳ 크거나 같은 것을 뽑을수 있기 때문에 +1 하지 않음.

↳ bucket [lastIndex] + 1.

#2 중복조합

```
main()
{
    int n = 5;
    int bucket[3];

    pick( n, bucket, 3, 3 );
}
```

```
C:₩Windov
0 0 0
0 0 1
0 0 2
0 0 3
0 0 4
0 1 1
0 1 2
0 1 3
0 1 4
0 2 2
0 2 3
0 2 4
0 3 3
0 3 4
0 4 4
1 1 1
1 1 2
1 1 3
1 1 4
1 2 2
1 2 3
1 2 4
1 3 3
1 3 4
1 4 4
2 2 2
2 2 3
2 2 4
2 3 3
2 3 4
2 4 4
3 3 3
3 3 4
3 4 4
4 4 4
계속하려면
```

```
main()
{
    int n = 8;
    int bucket[4];

    pick( n, bucket, 4, 4 );
}
```

```
hhyuck@node0
0 0 0 0
0 0 0 1
0 0 0 2
0 0 0 3
0 0 0 4
0 0 0 5
0 0 0 6
0 0 0 7
0 0 1 1
0 0 1 2
0 0 1 3
0 0 1 4
0 0 1 5
0 0 1 6
0 0 1 7
0 0 2 2
0 0 2 3
0 0 2 4
0 0 2 5
0 0 2 6
0 0 2 7
0 0 3 3
0 0 3 4
0 0 3 5
0 0 3 6
0 0 3 7
0 0 4 4
0 0 4 5
0 0 4 6
0 0 4 7
0 0 5 5
0 0 5 6
0 0 5 7
0 0 6 6
0 0 6 7
0 0 7 7
0 1 1 1
0 1 1 2
```

(뒤의 결과 생략)

조합(Combination)
중복조합(Combination with Repetition)
#3 중복순열(Permutation with Repetition)
순열(Permutation)

#3 중복순열

Permutation with Repetition

- 중복순열
  - 논리 흐름은 비슷하다.
  - 다른 점은 bucket에 새로운 item을 뽑을 때 매번 전체 아이템 중에서 뽑는다.
  - **Permutation with repetition 에서는 매번 전체 아이템 중에서 뽑는다.**

↳ candidate item 이 항상 전체 아이템이 된다.

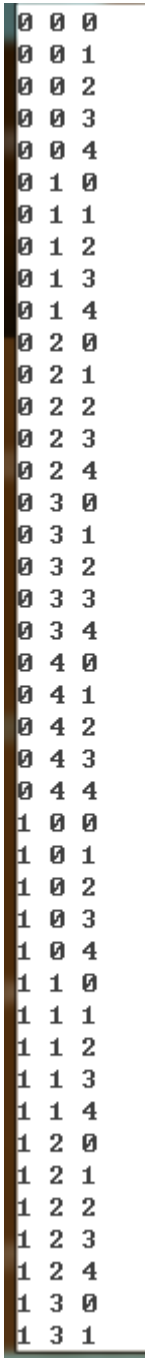
#3 중복순열

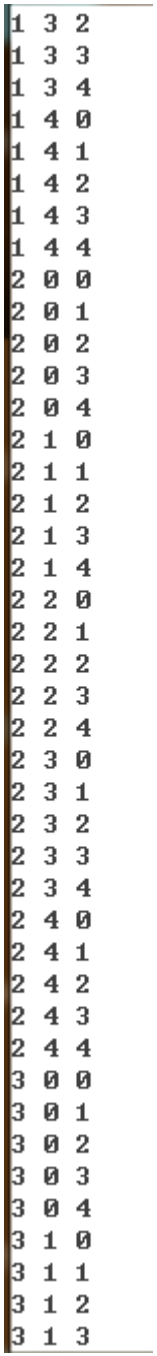
```
void pick( int n, int* bucket, int bucketSize, int k ) { //중복 조합

        int i, lastIndex, smallest, item;
        if(k == 0 ) {
                for (i = 0; i < bucketSize; i++)
                        printf("%d ", bucket[i]);
                printf("₩n");
                return;
        }

        lastIndex = bucketSize - k - 1; // 가장 최근에 뽑힌 수가 저장된 위치 index

        smallest = 0; // 이부분만 차이
          ↳ 매번 하나도 뽑지 않은 것처럼. 전체 아이템이 smallest 가 된다.
        for(item = smallest; item < n; item++) {
                bucket[lastIndex + 1] = item;
                pick(n, bucket, bucketSize, k - 1);
        }

}
```

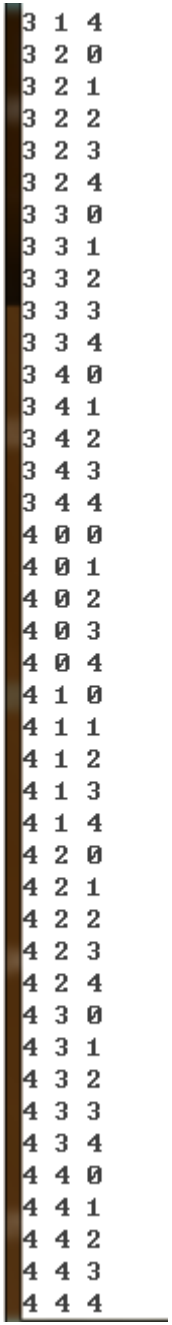
조합 / 중복조합에서의 smallest 계산.

```
if (bucketSize = k)
    smallest = 0;
else
    smallest = bucket[lastIndex] + 1. (조합)
    smallest = bucket[lastIndex] (중복조합)
```

하나도 뽑지 않았으면 0부터 뽑는다.








```
main()
{
    int n = 8;
    int bucket[4];

    pick( n, bucket, 4, 4 );
}
```

```
main()
{
    int n = 5;
    int bucket[3];

    pick( n, bucket, 3, 3 );
}
```

```
hhyuck@node00:
0 0 0 0
0 0 0 1
0 0 0 2
0 0 0 3
0 0 0 4
0 0 0 5
0 0 0 6
0 0 0 7
0 0 1 0
0 0 1 1
0 0 1 2
0 0 1 3
0 0 1 4
0 0 1 5
0 0 1 6
0 0 1 7
0 0 2 0
0 0 2 1
0 0 2 2
0 0 2 3
0 0 2 4
0 0 2 5
0 0 2 6
0 0 2 7
0 0 3 0
0 0 3 1
0 0 3 2
0 0 3 3
0 0 3 4
0 0 3 5
```

(뒤의 결과 생략)

조합(Combination)
중복조합(Combination with Repetition)
중복순열(Permutation with Repetition)

#4 순열(Permutation)

#4 순열(Permutation)

Permutation

- 이건 좀 어렵다.
- 순열
  - 논리 흐름은 비슷하다.
  - 다른 점은 bucket에 새로운 item을 뽑을 때 bucket에 존재하지 않는 아이템 중에서 뽑는다.
  - **Permutation에서는 안 뽑힌 아이템 중에서 뽑는다.**

(2. 0. ~)
↳ 2. 0을 제외한 다른 숫자를 뽑는다.

#4 순열(Permutation)

```
void pick( int n, int* bucket, int bucketSize, int k ) { //중복 조합

        int i, lastIndex, smallest, item;
        if(k == 0 ) {
                for (i = 0; i < bucket_size; i++)
                        printf("%d ", bucket[i]);
                printf("₩n");
                return;
        }

        lastIndex = bucketSize - k - 1; // 가장 최근에 뽑힌 수가 저장된 위치 index
        smallest = 0;
        for(item = smallest; item < n; item++) {
                int j = 0; int flag = 0;
                for(j=0; j <= lastIndex; j++ )
                        if ( bucket[j] == item )  flag = 1;
                if( flag == 1 ) continue;

                bucket[lastIndex + 1] = item;
                pick(n, bucket, bucketSize, k - 1);
        }
}
```

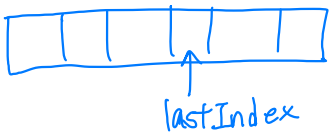
→ 오름차순이 아니라는 코드

중복을 허용하지 않음을 나타내는 코드

item이 나왔는지를 점검하고, 나오지 않았으면 그것을 bucket에 넣고 재귀로 pick을 부른다.

이미 뽑혔는지를 검사

lastIndex 까지 루프를 돌며 검사한다.



flag=1 이면 아래의 코드 실행하지 않고, item+1 해서 중복이 나오지 않을 때까지 bucket을 검사한다.


```
0 1 2
0 1 3
0 1 4
0 2 1
0 2 3
0 2 4
0 3 1
0 3 2
0 3 4
0 4 1
0 4 2
0 4 3
1 0 2
1 0 3
1 0 4
1 2 0
1 2 3
1 2 4
1 3 0
1 3 2
1 3 4
1 4 0
1 4 2
1 4 3
2 0 1
2 0 3
2 0 4
2 1 0
2 1 3
2 1 4
2 3 0
2 3 1
2 3 4
2 4 0
```

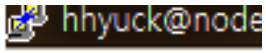
```
2 4 1
2 4 3
3 0 1
3 0 2
3 0 4
3 1 0
3 1 2
3 1 4
3 2 0
3 2 1
3 2 4
3 4 0
3 4 1
3 4 2
4 0 1
4 0 2
4 0 3
4 1 0
4 1 2
4 1 3
4 2 0
4 2 1
4 2 3
4 3 0
4 3 1
4 3 2
```

계속하려면

→ candidate 를 뽑을 때, 기존의 bucket 에 그 데이터가 있는지 확인하고 없으면 뽑는다.

```
main()
{
    int n = 5;
    int bucket[3];

    pick( n, bucket, 3, 3 );
}
```

```
main()
{
    int n = 8;
    int bucket[4];

    pick( n, bucket, 4, 4 );
}
```



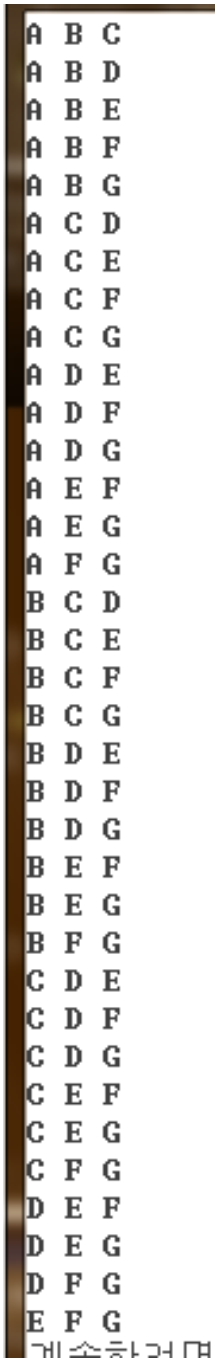

```
0 1 2 3
0 1 2 4
0 1 2 5
0 1 2 6
0 1 2 7
0 1 3 2
0 1 3 4
0 1 3 5
0 1 3 6
0 1 3 7
0 1 4 2
0 1 4 3
0 1 4 5
0 1 4 6
0 1 4 7
0 1 5 2
0 1 5 3
0 1 5 4
0 1 5 6
0 1 5 7
0 1 6 2
0 1 6 3
0 1 6 4
0 1 6 5
0 1 6 7
```

(뒤의 결과 생략)

LAB(뽑기-공뽑기)

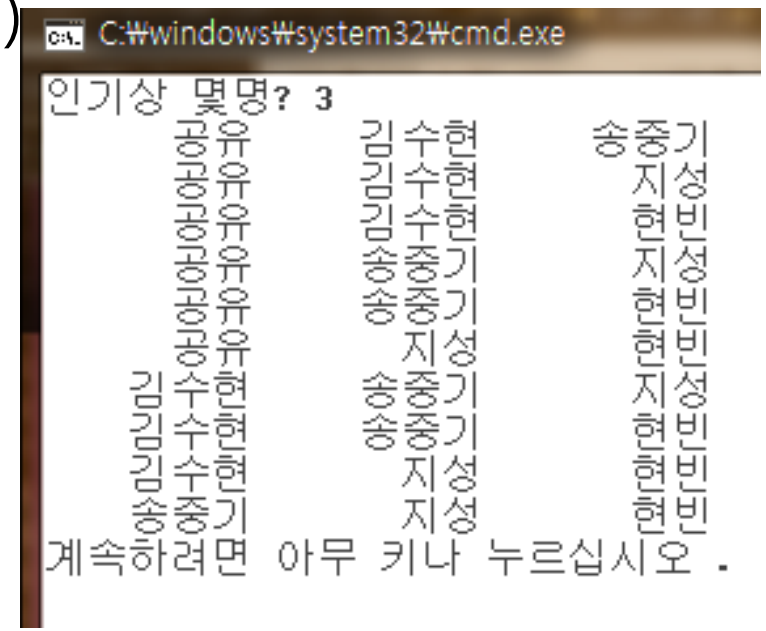
- ‘A’, ‘B’, ‘C’, ‘D’, ‘E’, ‘F’, ‘G’의 번호가 매겨져 있는 공 7개에서 3개를 뽑는 경우를 구하려 한다.

1. Item과 Bucket에 대해서 고민하라.
2. 조합/중복조합/중복순열/순열인지 생각하라.



LAB(뽑기-인기상)

- 배우들 중에서 n명을 뽑아서 인기상을 주려 한다.
  - 배우를 공유, 김수현, 송중기, 지성, 현빈 중에서 선택한다고 하자. 어떤 경우가 가능한가?
  - 입력 : 3 (← 인기상 몇 명?)
  - 출력

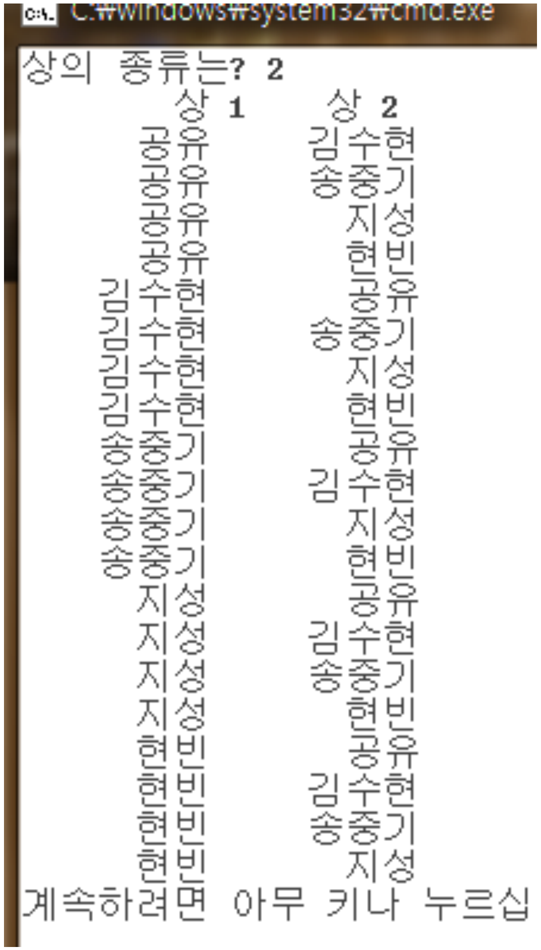


5명의 배우 ↑ ↳ 입력받은 n

1. Item과 Bucket에 대해서 고민하라.
2. 조합/중복조합/중복순열/순열인지 생각하라.

LAB(뽑기-연기상)

- 배우들 중에서 n명을 뽑아서 최우수연기상, 우수연기상을 주려 한다. 1명은 단 하나의 상만 받을 수있다.
  - 배우를 공유, 김수현, 송중기, 지성, 현빈 중에서 선택한다고 하자. 어떤 경우가 가능한가?
  - 입력 : 2 (← 상의 종류는?)
  - 출력



1. Item과 Bucket에 대해서 고민하라. (Item ↑ 5명의 배우, Bucket → 입력받은 n)
2. 조합/중복조합/중복순열/순열인지 생각하라.

HW(뽑기-4진수)

- 4진법으로 만들 수 있는 n자리의 수를 모두 나열하는 프로그램을 작성하시오.
  - 입력 : 3 ← 3자리의 수
  - 출력 : 000, 001, …. 333 으로 나오면 된다.

↱ 0.1.2.3 ↱ 3개

  - 1. Item과 Bucket에 대해서 고민하라.
  - 2. 조합/중복조합/중복순열/순열인지 생각하라.

↳ 001. 100 의 경우가 다르고, 숫자가 중복해서 나올 수 있음.

HW(뽑기-수식나열)

- 1부터 n까지 연속되어 있는 수와 +/-를 이용하여 만들 수 있는 모든 수식을 그 결과와 함께 나열하시오.
  - 입력 : 2 ← 1부터 2
  - 출력 :
    - +1+2 = 3
    - +1-2 = -1
    - -1+2 = 1
    - -1-2 = -3

↳ 숫자를 뽑는 것이 아니라. 부호를 n개 뽑는 것이다.

~1 ~2 ~3 ··· ~n

+, - 를 n개 뽑는다.

부호의 순서를 고려해야 한다.

↑ +, - ↑ n개

  - Item과 Bucket에 대해서 고민하라.
  - 조합/중복조합/중복순열/순열인지 생각하라.

HW(뽑기-세배돈)

- 1000, 5000, 10000원 짜리 지폐로 세뱃돈을 주고 싶다. 주고 싶은 세뱃돈을 입력하면 3가지 지폐들을 이용하여 세뱃돈을 만들 수 있는 방법을 나열하시오.
  - 입력 : 6000 ← 6천원
  - 출력 :
    - 1000 1000 1000 1000 1000 1000
    - 5000 1000
  - Item과 Bucket에 대해서 고민하라.
  - 조합/중복조합/중복순열/순열인지 생각하라.

0원짜리 지폐를 추가해서
6개를 중복조합으로 뽑는다.
6000 > 원인 지폐는 출력하지 않고,
0원짜리 지폐는 출력하지 않는다.