Lecture 14 : 연결 리스트(linked list)의 응용

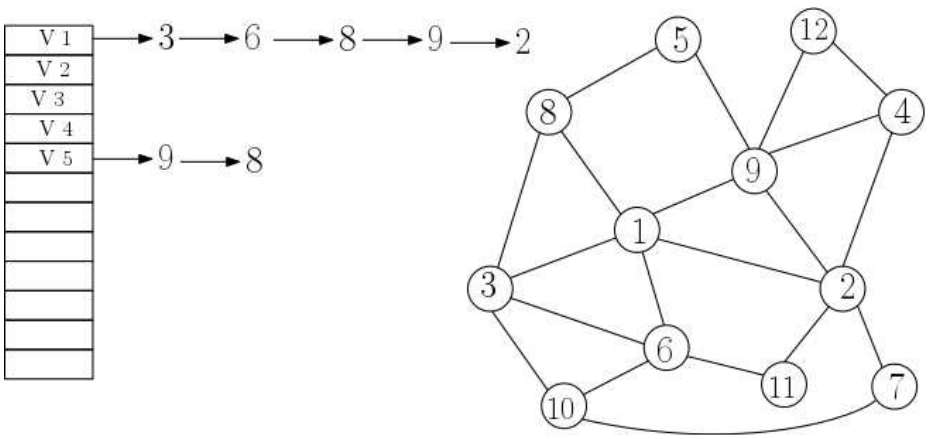
이 장에서는 연결리스트의 다양한 응용문제를 다룬다. 여러분은 제시된 예제에서 왜 다른 자료구조가 아닌 연결리스트를 사용하는 것이 효율적인지를 이해해야 한다. 그리고 그 성능을 실제 수행시간 기준으로 측정해보아야 할 것이다.[1)]

리스트 구조의 가장 큰 덕목은 dynamic 동작을 처리하는데 효율성을 제공한다는 것이다. 즉 삽입 삭제가 자주 사용할 경우 리스트의 경우는 배열과 달리 전체 데이터의 크기와 상관없이 그 앞뒤 원소의 pointer만 조정하면 되기 때문에 $O(1)$의 시간이 걸린다는 것이다. 만일 삽입 삭제와 같은 동작이 별로 없는 경우에는 고정적인 배열만으로도 충분하다. 사전(dictionary)은 고정적인(static) 자료구조의 대표적인 예이다. 사전(dictionary)과 같은 경우에는 배열보다 더 빠른 시간에 원소를 탐색할 수 있는 hash 구조를 이용한다.[2)]

이렇게 자료구조에 변화가 없는 경우에는 미리 가장 효율적인 자료구조를 만들어 놓고 준비를 한다. 이런 작업을 전처리(pre-processing)이라고 하며, 이렇게 만들어진 자료구조는 변화없이 운용된다.

14.1 그래프(Graph)를 표현하는 전형적인 자료구조.

```
vector< list<int> > Graph ;
```



연결 리스트(adjacent list)에서 해야 할 작업

1) edge (x,y) 추가

2) edge (x,y)의 삭제

3) 모든 vertex x의 인접한 vertices를 삭제, print $N(x)$={ }

---

1) 단순히 빠르다 늦다가 아니라 얼마나 더 빠르고 느릴지에 대하여 확인해야 한다.
2) 사전에 새로운 단어가 추가되고 빠지는 경우 10여년 단위로 batch 작업으로 처리된다.

4) vertex x의 삭제 (연결된 모든 edge의 삭제)

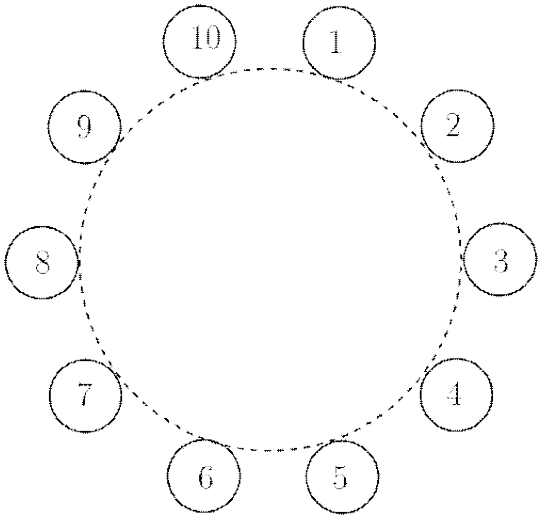
5) vertex x의 인접한 vertices, $N(x)$={ }

6) 모든 edge (x,y)를 출력하기

7) connected component 출력하기

14.2 Josephus Problem : 1 세기쯤에 역사가 파비우스 조셉푸스 (Favius Josephus)는 배를 타고 여행하기를 좋아했다고 한다. 그런데 어떤 탐험 여행 도중 해적을 만나 모두 꼼짝없이 잡히는 신세가 되었다. 조셉푸스와 같이 배를 타고 간 사람의 수는 모두 N명이다. 해적 대장은 성격이 아주 괴팍한 악당인데, 선장은 잡힌 사람들을 원형으로 앉힌 뒤에 돌아가면서 매 두 번째 사람을 바다에 던져 버린다. 이렇게 하여 제일 마지막에 남게 된 한 사람만을 살려준다고 한다. 예를 들어 10명이 붙잡힌 경우를 들어보자. 따라서 이 경우에는 5번의 위치에 앉아있든 사람이 최후의 생존자로 살아남게 된다.

해적이 붙잡힌 사람 10명이 원을 만들어서 갑판 위에 앉아 있다. 1번부터 시작하여 매 2번째 사람을 선택하여 바다에 던져 넣는다. 따라서 바다에 던져지는 순서는 2번, 4번 6, 8, 10번이며, 그 다음에는 3번이다.



14.3 N=10 명인 경우를 위 작업을 수행해보자,

순서	1	2	3	4	5	6	7	8	9	10
빠지는 사람	***2***	***4***	***6***	***8***	***10***	***3***	***7***	***1***	***9***	***5***

여러분이 생각해야 할 문제는 N=10 이 아닌 일반적인 경우에, 예를 들어 N=32 라면 여러분은 과연 몇 번째 자리에 앉아야만 살아남을 수 있는지를 빨리 계산해서 그 자리를 차지하면 된다. 이 문제를 좀 일반화하여 매번 2번째가 아니라 매 k번째 사람을 바다에 던져 넣을 경우 초기 N명에서 시작하여 가장 최후에 살아남는 사람이 누구인지를 계산하는

것이다. 예를 들어 위의 경우에 k=3이라면 처음에 빠지는 사람은 3, 6, 9, 그리고 2, 7이 되어야 한다. 아래 코드를 참조하시오.

```
#include <bits/stdc++.h>
#define mout(msg,lx)  \
     cout<< "\n--" << msg<<": " ; for(auto _w : lx ) \
     cout<< " " << _w ;
using namespace std;

struct Nodex {
   char letter ;
   list<Nodex>::iterator jump   ;
} nx, ny ;

int main(){

   string name="abcdefghijklmnopqrstuvwxyz" ;
   list<Nodex> ml ;
   char xc ;

   vector<char> vn(name.begin(), name.end()) ;
   mout("초기:", vn) ;

   for(auto it = vn.begin(); it != vn.end() ; ++it){
      nx.letter = *it ;
      nx.jump  = ml.end() ;
      ml.push_back( nx ) ;
   }

   // jump iterator(pointer)를 setting 합니다.
   auto jit = ml.begin()  ;
   advance(jit, 7) ; // 앞으로 7칸을 먼저 감
   cout << "\n jit의 초기값 : " << (*jit).letter ;

   for(auto it= ml.begin(); it!= ml.end(); ++it) {
      (*it).jump = jit ;
      ++jit ;
      if (jit == ml.end()) { jit = ml.begin() ; }
   }

   auto bit = ml.begin() ;
   for(auto it= ml.begin(); it!= ml.end(); ++it) {
     cout << "\n (*it).letter =" << (*it).letter ;
     bit = (*it).jump ;
     cout << "  (*bit).letter =" << (*bit).letter ;
   }

   int count = 0 ;
   auto it = ml.begin();
   it = (*it).jump ;
   cout << "\n\n Jump 시작 위치의 letter= " << (*it).letter ;

   while( ++count < 30 ) {
      cout << "\n from " << (*it).letter ;
      it = (*it).jump  ;
      cout << " to " << (*it).letter ;
   }
   return 0;
}
```

이와 같이 좀 더 일반화된 문제로 *Josephus(N, k)*을 생각해볼 수 있다. 앞서 말한 문제와 답은 Josephus(10,2)=5 으로 표시된다. 물론 이 문제는 Dynamic programming을

이용하면 일반화된 공식을 찾을 수 있다. Josephus(N,2)은 Josephus(N/2,2)를 이용해서 풀 수 있다.

Q) N이 짝수일 때 Josephus(2M , 2)을 풀어보자.
Q) N일 홀수일 때 Josephus(2M+1 , 2)을 풀어보자.

물론 이 문제는 리스트를 이용하는 simulation이 아닌 수식을 이용한 계산으로 해결되기도 하며 여러가지 변형이 있다.

14.4 여러분은 아래 Josephus 문제의 답을 STL list를 이용해서 해결해야 한다. 이 문제를 vector를 이용하는 경우와 List를 이용하는 경우로 비교하여 평가해보자.

N	k	Josephus(N, k)
10	*2*	
25	*2*	
140	*2*	
4,567	*2*	
100,327	*2*	
15	*3*	
100	*5*	
1000	*22*	

14.5 Generalized Josephus Problem : 어떤 원형 리스트(circular list)에 원소들이 1부터 N 까지 달려있다. 1부터 시작하여 매 K번째 원소를 제거한다. 이 작업은 마지막 한 노드가 남을 때까지 계속된다. 이 경우 마지막 남은 번호는 유일한데, 그 때의 번호를 Josephus(N, K) 라고 한다. 예를 들어 N=10, K=2 이면 다음의 노드 순서로 전체 구성에서 삭제된다.

Order Eliminated	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Person Eliminated K=2	2	4	6	8	10	3	7	1	9	**5**

Order Eliminated	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Person Eliminated K=5	5	10	6	2	9	8	1	4	7	**3**

따라서 Josephus(10,2)=5 이며, Josephus(10,5)= 3 이 된다. 만일 K=1이라면 1번부터 차례대로 지워지므로 Josephus(N,K=1)은 항상 N이 된다. 여러분은 이것을 실제 제시한 circular list를 이용해서 풀어보시기 바랍니다. 다음은 Josephus(16,3)의 순서를 나타내

고 있다.

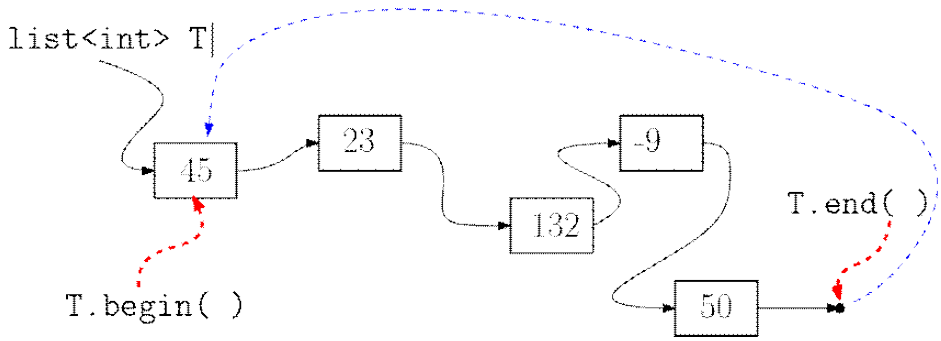
Order Eliminated	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Person Eliminated K=3	4	8	12	16	5	10	15	6	13	3	14	9	7	11	2	1

14.6 Generalized Josephus Problem 2: 이번에는 제일 마지막 원소를 찾아내는 것이 아니라 마지막에서 w번째 원소를 찾아내는 것이다. 이 문제는 매우 복잡한 dynamic programming 식을 요구한다. [3] 이 문제는 Gosephus(N, k, w)로 표시할 수 있다. 즉 위의 예를 이용하면 다음의 결과를 알 수 있다. 즉 $w=1$이면 일반적인 Josephus 문제로 귀납된다.

Gosephus(16,3,1) = 1,
Gosephus(16,3, 3) = 11,
Gosephus(16,3, 5) = 7

14.7 [매우 중요] STL List에서의 순환

STL list에서 iterator를 계속 증가시키면 다음 처음으로 되돌아 온다. 즉 STL list의 기본적인 구조는 circular list이다. 그런데 이 순환 과정에서 아주 유의해야 할 점이 있다. 아래는 list<int> T의 구조이며 현재는 그 원소가 [45, 23, 132, -9, 50]으로 지정된 상태이다.



만일 it가 위 리스트의 iterator라고 할 때 it = T.begin( ), *it는 45를 지정한다. 그 다음 it++ ; 한 다음 다시 그것이 가리키는 원소를 가지고 오면 23을 가리킨다. 그런데 그림과 같이 it=T.end( )[4]는 원소가 아니라 제일 마지막 null point를 가리키며 이 상태에서 다시 ++it를 수행하면 다시 첫 원소를 지시한다. 즉 ++it를 반복하여 *it를 출력하면 한번은 null value를 출력하게 된다.

이런 이유로 인하여 STL에서는 T.rbegin(), T.rend( )를 제공한다. 즉 제일 마지

3) 이 문제는 조환규 교수가 ICPC East-Asia 본선에서 출제한 적이 있다. 예를 들어 N=10000 일 때 마지막에서 10번째 제거되는 원소를 찾는 것이다. jump step $k=2$ 이다. Gosephus(10000,2,3) 으로 형식화할 수 있다. 수학에 재주가 있는 학생이라면 한번 도전해보기를 권한다. 이중 dynamic programming 식으로 해결된다. 2021

4) *T.end( )를 "망통"으로 부르기로 한다. null보다도 훨씬 더 귀에 쏙쏙 들어오지 않습니까?

막 원소를 출력하기 위해서는 *T.end( )가 아니라 *T.rbegin( )를 찾아야 한다. 위 Josephus 문제를 해결할 때 이점을 주의해야 한다. 아래 코드를 꼼꼼히 살펴보기를 권한다. 아래 코드를 보고 어떤 값이 출력되는지 설명하시오.

```
#include <bits/stdc++.h>
using namespace std;

int main () {

    list<int> mylist   = { 101, 102, 103, 104, 105, 106 };
    list<int>::iterator itb, ite, itt ;

    itb = mylist.begin() ;
    ite = mylist.end() ;

    cout <<"\n 1. *itb = " << *itb ;
    cout <<"\n 2. *ite = " << *ite ;

    mylist.push_front(99) ;
    cout <<"\n 3. *mylist.begin() = " << *mylist.begin()  ;
    cout <<"\n 4. *itb = " << *itb ;
    cout <<"\n 5. *ite = " << *ite ;
    cout <<"\n 6. *(++itb) = " << *(++itb) ;
    cout <<"\n 7. *(--ite) = " << *(--ite) ;
  return 0;
}
```

List와 vector iterator의 차이점을 잘 이해하고 있어야 한다. vec_iter를 vector의 iterator, list_iter를 list의 iterator라고 하자. 다음 표를 보면서 각 의미를 이해해야 한다.

표현	의미
*vec_iter;	vec_iter이 지시하고 있는 원소의 값
vec_iter++;	vec_iter다음 원소의 위치(원소크기 무관)
vec_iter+= k;	vec_iter에서 k번째 다음의 원소 위치
list_iter++ ;	list_iter이 지시하는 다음 원소
list_iter-- ;	list_iter이 지시하는 바로 이전 원소
list_iter+=k;	list_iter의 위치에서 k번째 앞 원소? (그러나 이 동작은 지원하지 않음, 오류) 왜? 리스트는 바로 앞,전 원소외에는 알 수가 없어요... 알 수가...
advance(it,k)	list에서 k번째 앞 뒤 원소를 지시하기

따라서 list에서의 iterato의 jump 작업을 위하여 특별한 함수 advance( )를 제공하고 있다. advance(it,k)를 이용해서 k번째 앞 뒤의 원소로 쉽게 옮아갈 수 있다. 그러나 실제로는 이 작업은 it++나 it--를 k번 반복하는 것에 불과하다.

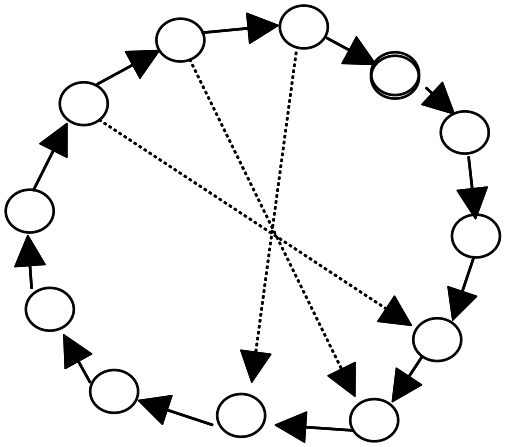
```
typedef list<int>::iterator Lit ;

int main(){
    list<int> X = { 0, 1,2,3,4,5,6,7,8,9} ;

    Lit Xit ;


    for(int i= 0; i<= 20 ; i++ ){
       Xit = X.begin() ;   // 항상 처음으로 고정
       advance(Xit,i) ;
       cout << "\n i=" << i << ", *Xit= " << *Xit ;
    }

    cout << "\n 거꿀로 돌려 봅니데이. \n " ;
   for(int i= 0; i<= 20 ; i++ ){
       Xit = X.begin() ;   // 항상 처음으로 고정
       advance(Xit,-i) ;
       cout << "\n Backward i=" << i << ", *Xit= " << *Xit ;
    }
}
```

대응되는 동작으로는 distance( )가 있다. 이 함수는 동일한 list에서 사용되는 두 iterator의 간격, 즉 그 사이에 몇 개의 노드가 있는지를 돌려주는 매우 유용한 함수이다.

```
int main(){

    string mystr = "PNU_ALL" ;
    list<char> mylist ( mystr.begin(), mystr.end() );

    outlist(" mylist[]= ", mylist ) ;

    auto it1 = mylist.begin() ;
    auto it2 = mylist.end( ) ;

    int gap ;
    gap = distance(it1, it2 ) ;
    cout << "\n gap= " << gap ;

    return 0;
}
// [P, N, N, +, A, L, L] end()
//  0  1  2  3  4  5  6,    7
```

14.8 리스트를 활용하다보면, 단순한 Linked list외에 다양한 노드의 추가 정보가 필요한 경우가 있다. 예를 들어 각 노드에서 다음 노드 (next, 또는 it++)로도 갈 수 있지만 몇 개의 노드를 미리 점프할 수 있도록 준비하면 여러 작업에서 편할 수 있다.

14.9 앞의 문제와 연관이 되어있다. 모든 노드는 그 다음 노드와 동시에 전체에서 N/2 번째 앞 서있는 노드를 가질 수 있도록 하여 일반적인 iterator로는 it++를 통하여 다음 노드, jump를 통하여 한번에 +(N/2) 번째로 바로 뛰어갈 수 있도록 하면 좋을 것이다. 이것을 위한 자료 구조를 만들고 각 노드에서 한번은 next, 한번은 jump를 하면 모든 노드를 출력하는 프로그램 nodejump.cpp를 작성해보자.



14.10 Card(List) Cut

- N장의 카드가 순서대로 제시된다. 첫 i번째 위치의 카드를 c_i라고 표현한다.
- N장 카드 중에서 i번째에서 $j\,(i \le j)$번째 카드까지의 연속된 묶음을 C[i:j]로 표현한다.[5)]
- 이 특정한 sublist를 들어내서 카드 deck의 제일 뒤로 옮긴다.
  이것은 cut(i,j)라고 한다. 이 작업을 위한 코드를 작성하시오.

14.11 Card(List) Shuffling

Shuffle은 2개의 list La, Lb를 대상으로 수행한다. 방법은 두 리스트의 원소를 다음과 같은 순서로 섞은 것을 의미한다.

$$La = [\, a_1,\ a_2,\ a_3, \ldots\ a_i, a_{i+1}, \ldots\ a_n \,]$$
$$Lb = [\, b_1, b_2,\ b_3, \ldots\ b_i \ldots b_{m-1},\ b_m \,]$$
$$\text{Shuffle(La, Lb)} = [\, a_1, b_1, a_2, b_2, \ldots\ldots a_i,\ b_i, a_{i+1}, b_{i+1}, \ldots . b_n \,]$$

14.12 함수의 표현: 음이 아닌 정수 계수와 지수의 항(term)으로 구성된 다항식 함수가 있다. 아래 $f(x)$는 한 예이다.

$$f(x) = 4x^2 - 3x^7 - x + 2x^5 - x^4 - 7x + 2$$

각 항은 $c_i \cdot x^{e_i}$으로 표현된다. 계수인 c_i는 0이 아닌 정수, 지수 e_i는 0을 포함한 양의 정수이다. 여러분은 이렇게 표현된 함수를 모두 더한 새로운 함수를 "**표준형 (canonical form)" 함수로** 출력해야 한다. 표준형이란 서로 다른 지수의 항이 **지수의 내림차순으로 재정리된 형**을 말한다. 단 계수가 0인 $c_i = 0$ 항은 표현되지 않는다. 위에서 제시된 함수 $f(x)$를 표준형으로 바꾸면 다음과 같다.

$$f(x) = -3x^7 + 2x^5 - x^4 + 4x^2 - 8x + 2$$

---

5) 주의해야 한다. python의 list L에서 L[i:j]는 index i부터 j를 넘지 않는 sublist, 즉 $j-1$번째 카드까지의 sublist를 의미한다.

[입출력] 표준 입력 파일의 첫 줄에는 제시된 함수의 개수 $N \le 10$ 이 주어진다. 그 다음에는 N개 함수의 정보가 이어서 나타난다. 함수 정보 첫 줄에는 해당 함수의 항(term) 개수 t_i가 제시된다. 다음 이어지는 t_i개 줄에는 각 항의 계수와 지수 c_i e_i가 두 정수로 주어진다.

입력 함수의 계수와 지수의 범위는 $-100 \le c_i \le 100$, $0 \le e_i \le 1{,}000$ 이다. 단 입력은 표준형이 아니며 같은 지수의 항도 존재할 수 있다. 여러분은 제시된 다항식을 모두 더한 결과를 표준형 함수로 만들어 출력해야 한다. 출력 형식은 다음과 같다.

항의 개수 k를 첫 줄에 출력하고 이어 지수의 내림차순으로 각 항의 계수와 지수, 두 정수를 한 줄에 하나씩 순서대로 출력한다. 단 $f(x)=0$는 첫 줄에 **'1'** 다음 줄에 **'0 0'**로 출력한다.

[예제]

stdin	**stdout**
3 // 3개의 함수 3 // 함수는 3 항 -1 1 // $-x+5+x^2$ 5 0 1 2 2 // 2개 항, $2x^5-x^2$ 2 5 -1 2 1 // 1개의 항, $-2x^4$ -2 4	4 // 4개의 항(terms) 2 5 -2 4 -1 1 5 0 // $2x^5-2x^4-x+5$

14.13 위 다항식 덧셈을 활용하여 다항식 곱셈 코드를 만들어 보자. 자료구조는 앞서 사용한 것과 동일하다.

[문제] 음이 아닌 정수 계수와 지수의 항(term)으로 구성된 다항식 함수가 있다. 아래 $f(x)$는 한 예이다.

$$f(x)=4x^2-3x^7-x+2x^5-x^4-7x+2$$

각 항은 $c_i \cdot x^{e_i}$으로 표현된다. 계수인 c_i는 0이 아닌 정수, 지수 e_i는 0을 포함한 양의 정수이다. 여러분은 이렇게 표현된 함수를 모두 더한 새로운 함수를 "**표준형(canonical form)" 함수로** 출력해야 한다. 표준형이란 서로 다른 지수의 항이 **지수의 내림차순으로 재정리된 형**을 말한다. 단 계수가 0인 $c_i=0$ 항은 표현되지 않는다. 위에서 제시된 함수 $f(x)$를 표준형으로 바꾸면 다음과 같다.

$$f(x)=-3x^7+2x^5-x^4+4x^2-8x+2$$

[입출력] 표준 입력 파일의 첫 줄에는 제시된 함수의 개수 $N \leq 10$ 이 주어진다. 그 다음에는 N개 함수의 정보가 이어서 나타난다. 함수 정보 첫 줄에는 해당 함수의 항(term) 개수 t_i가 제시된다. 다음 이어지는 t_i개 줄에는 각 항의 계수와 지수 c_i e_i가 두 정수로 주어진다.

입력 함수의 계수와 지수의 범위는 $-100 \leq c_i \leq 100$, $0 \leq e_i \leq 1{,}000$ 이다. 단 입력은 표준형이 아니며 같은 지수의 항도 존재할 수 있다. 여러분은 제시된 다항식을 모두 더한 결과를 표준형 함수로 만들어 출력해야 한다. 출력 형식은 다음과 같다.

항의 개수 k를 첫 줄에 출력하고 이어 지수의 내림차순으로 각 항의 계수와 지수, 두 정수를 한 줄에 하나씩 순서대로 출력한다. 단 $f(x)=0$는 첫 줄에 **'1'** 다음 줄에 **'0 0'**로 출력한다.

[예제]

stdin	**stdout**
3 // 3개의 함수 3 // 함수는 3 항 -1 1 // $-x+5+x^2$ 5 0 1 2 2 // 2개 항, $2x^5-x^2$ 2 5 -1 2 1 // 1개의 항, $-2x^4$ -2 4	4 // 4개의 항(terms) 2 5 -2 4 -1 1 5 0 // $2x^5-2x^4-x+5$

14. 14 다음은 과제물로 제시될 새로운 문제이다.

[문제] n개의 좌석이 그림-1(a)과 같이 원형(circular)으로 배치된 카페 <코로나>가 영업 중에 있다. 즉 1번 좌석은 마지막 n번 좌석과 인접해 있다. 이 카페에 많은 사람이 들어오고 나간다. 그런데 코로나 바이러스 확산을 막기 위해서 카페에서는 각 입장 고객의 바로 앞뒤에 앉은 사람을 모두 기록해두려고 한다. 만일 확진자가 이 카페를 다녀간 경우에 이 자료로 근접 접촉자를 추적하기 위해서이다.

자신 앞쪽의 고객은 반시계방향으로 진행할 때 처음 만나게 되는 손님이며 뒤쪽 고객은 시계방향으로 진행할 때 처음 만나게 되는 고객이다. 이 문제에서 좌석 번호와 손님의 번호는 같은 것으로 본다.

그림-1에 나타난 예를 들어 설명해보자. (b)의 경우 4번 고객(좌석)의 앞 사람(좌석)은 22번이며 뒷사람은 6번이다. 그리고 17번 고객의 앞 고객은 15번, 뒤 인접 고객은 22번이다. 만일 카페에 단 1명만 있으면 그 유일한 손님의 앞뒤 고객은 바로 자신이 된다.

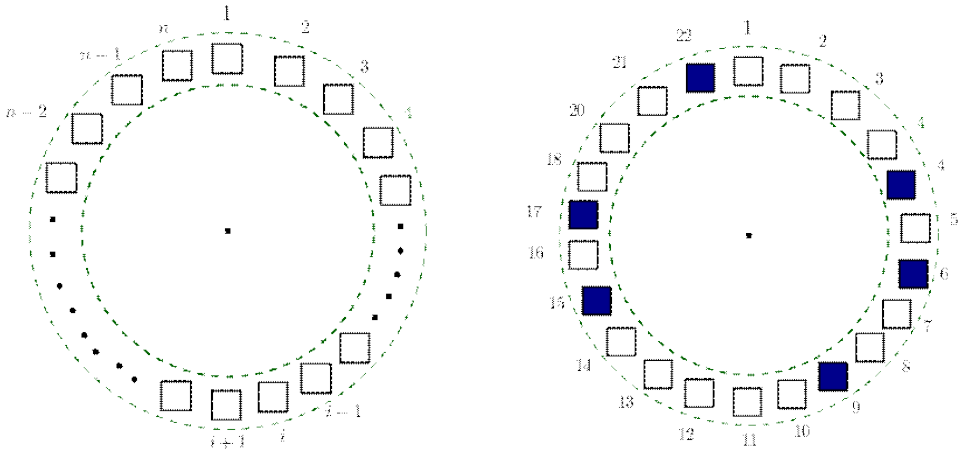


그림-1 (a) 카페에 배치된 n개 좌석. (b) 22개 좌석의 상황. 파란색은 고객.

[입출력] k명의 고객이 이 카페에 입출입한 기록이 있다. i번째 고객의 번호는 양의 정수 u_i로 표시된다. 나타난 횟수가 홀수 번이면 입장, 짝수 번인 경우는 퇴장을 의미한다. 즉 678이 처음 나타나면 678번 좌석에 착석한 상황이며 다시 678이 나타나면 이는 678번이 퇴장한 상황을 의미한다. 즉 어떤 번호가 짝수 번 나타나면 이 번호에 해당하는 사람이 카페를 떠나갔음을 의미한다. 여러분은 각 고객이 입장하여 자신의 번호에 해당하는 자리에 앉을 때 때마다 그 고객의 앞뒤 고객(좌석) 번호를 출력해야 한다. 단 퇴장할 때는 출력할 필요는 없다.

입력 파일 첫 줄에는 카페에 준비된 좌석의 개수 n과 출입 고객의 기록(번호)의 전체 개수 k가 나온다. 이 범위는 $10 \le n \le 2^8 = 1{,}000{,}000,\ 5 \le k \le 1000$ 이다. 즉 최대 좌석은 100만개이다. 그리고 이어지는 k개의 각 줄에는 출입 고객의 번호가 제시된다. 고객은 반드시 자신의 번호와 같은 좌석에 앉는다. 즉 $u_s = 1234$ 고객은 반드시 1234번 자리에 앉아야 한다.

여러분은 각 고객 u_i가 **입장할 때마다** 그 고객의 번호 u_i와 그 앞 뒤에 앉은 두 명의 좌석(고객) 번호, 즉 3개의 숫자를 한 줄에 순서대로 출력한다. 손님이 카페를 나가는 상황에서는 번호를 출력할 필요가 없다. 단 카페에 1명만 있는 경우 그 고객의 앞뒤 고객은 그 자신이 된다.

[예제] 아래 stdin에는 여덟 번의 출입기록이 표시되어 있다. 실제 출입한 사람은 모두 7명이다. 3번 고객은 들어와서 곧 나갔다. 따라서 여러분은 입장한 일곱 사람이 앉은 자리의 앞뒤 고객 번호(자리번호)를 찾아서 한 줄에 순서대로 출력해야 한다.

입력 stdin	출력 stdout
22 8 //n=22,k=8 4 9 15 6 22 3 // 3 입장 3 // 3 퇴장 17	4 4 4 9 4 4 15 9 4 //15번 앞뒤 9,4 6 4 9 22 15 4 3 22 4 17 15 22

[제한조건] 프로그램의 이름은 **`cafe.{py, c,cpp,java}`**이다. 최대 제출 횟수는 15회, 데이터 당 제한시간은 1초이다. 과제 마감시간은 9월 25일(금요일) 저녁 10시이며, 22일 화요일부터 제출할 수 있다. NESPA에 연습용 Open data에 공개되면 그것으로 확인해볼 수 있다. 이번 문제의 **허용 코드 길이는 최대 2500 byte이다.**