6. 동치 관계(Equivalence Relations)

- 동치 관계
  - reflexive, symmetric, transitive 성질을 만족
  - "equal to"(=) 관계는 동치 관계임.
    - x = x
    - x = y 이면 y = x
    - x = y 이고 y = z 이면 x = z

- **동치 관계를 이용하여 집합 S를 "동치 클래스"로 분할**
  - 동일한 클래스내의 원소 x, y에 대해서는 x ≡ y 관계 성립
  - 예: 0 ≡ 4, 3 ≡ 1, 6 ≡ 10, 8 ≡ 9, 7 ≡ 4, 6 ≡ 8, 3 ≡ 5, 2 ≡ 11, 11 ≡ 0
  - 동치 클래스: {0, 2, 4, 7, 11}; {1, 3, 5}; {6, 8, 9, 10}

동치 클래스 검색 알고리즘

1. 동치항 <i, j>를 입력한 후 저장

2. 원소 0부터 시작하여 0의 동치항 <0, j> 항들을 검색한 후, j를 0과 동일한 클래스에 포함
   - Transitivity 속성에 의해 <j, k> 항의 원소 k도 0과 동일한 클래스에 포함됨
   - 이 과정을 반복하여 0을 포함하는 모든 원소를 발견

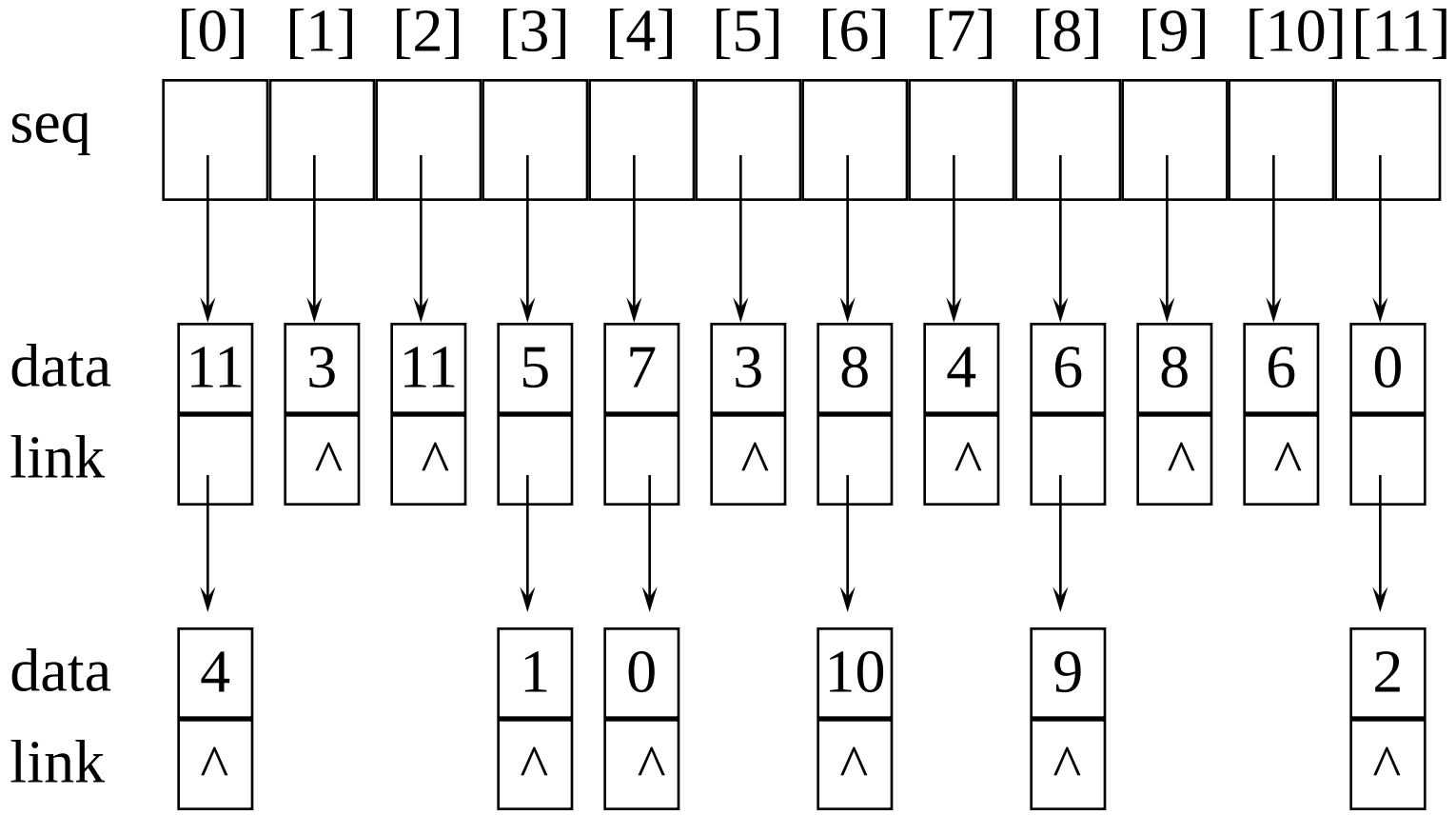
3. 클래스에 포함되지 않은 다른 원소들에 대해 단계 2를 반복

동치 클래스 검색을 위한 자료 구조(1)

- 변수 선언
  - m: 입력된 동치항의 수
  - n: 원소의 수

- 동치항의 구현 방법
  - 배열: pairs[n][n]
    - <i, j>가 입력될 경우, pairs[i][j]와 pairs[j][i]를 1로 설정
    - 원소의 수에 비해 동치항의 수가 적을 경우, 메모리 낭비
  - n개의 연결 리스트: seq[n]
    - <i, j>가 입력될 경우
      - 노드 j를 seq[i]가 가리키는 리스트에 추가
      - 노드 i는 seq[j]가 가리키는 리스트에 추가

예: 동치항의 입력이 완료된 후의 리스트



동치 클래스 검색을 위한 자료 구조(2)

- 일차원 boolean 배열 **out[n]**
  - out[i] = TRUE: 원소 i를 출력하여야 함. (초기값)
  - out[i] = FALSE: i가 이미 출력되어 다시 출력할 필요 없음.

- 리스트 표현을 이용한 동치 클래스 검색
  - Step 1: 동치항을 입력받아 seq[]를 이용한 리스트 구성
  - Step 2: out[i] = TRUE인 첫번째 원소 i, $0 \le i < n$, 를 선택하여 seq[i] 리스트를 스캔하면서 리스트의 원소들을 출력
    - seq[i]의 원소 중에서 out[] 배열의 값이 TRUE인 원소들의 리스트들을 현재 스캔을 완료한 후 스캔하여야 함. (Stack이 필요)
    - Stack을 구현하기 위하여 해당 노드의 link 필드를 stack의 다음 원소를 가리키는 포인터로 변경

Program 4.21: 초기 동치 알고리즘

```
void equivalence ( )
{
        initialize;
        while (there are more pairs)   {
                read the next pair < i, j >;
                process this pair;
        }
        initialize the output;
        do
                output a new equivalence class;
        while ( not done );
}
```

Program 4.22: 수정된 동치 알고리즘

```
void equivalence()
{
    initialize seq[] to NULL and out[] to TRUE;
    while ( there are more pairs )  {
            read the next pair < i, j >;
            put j on the seq[i] list;
            put i on the seq[j] list;
    }
    for (i = 0; i < n; i++)
            if (out[i])    {
                    out[i] = FALSE;
                    output this equivalence class;
            }
}
```

Program 4.23: 최종 동치 알고리즘(1)

```
#include <stdio.h>
#include <alloc.h>
#define MAX_SIZE 24
#define IS_FULL(ptr) (!(ptr))
#define FALSE 0
#define TRUE 1

struct node  {
     int data;                    // 정수형의 데이터가 입력된다고 가정
     struct node *link;
};

void main(void)
{    short int  out[MAX_SIZE];
     struct node *seq[MAX_SIZE], *x, *y, *top;
     int i, j, n;
     printf("Enter the size (<= %d)  ", MAX_SIZE );
     scanf("%d",  &n);
```

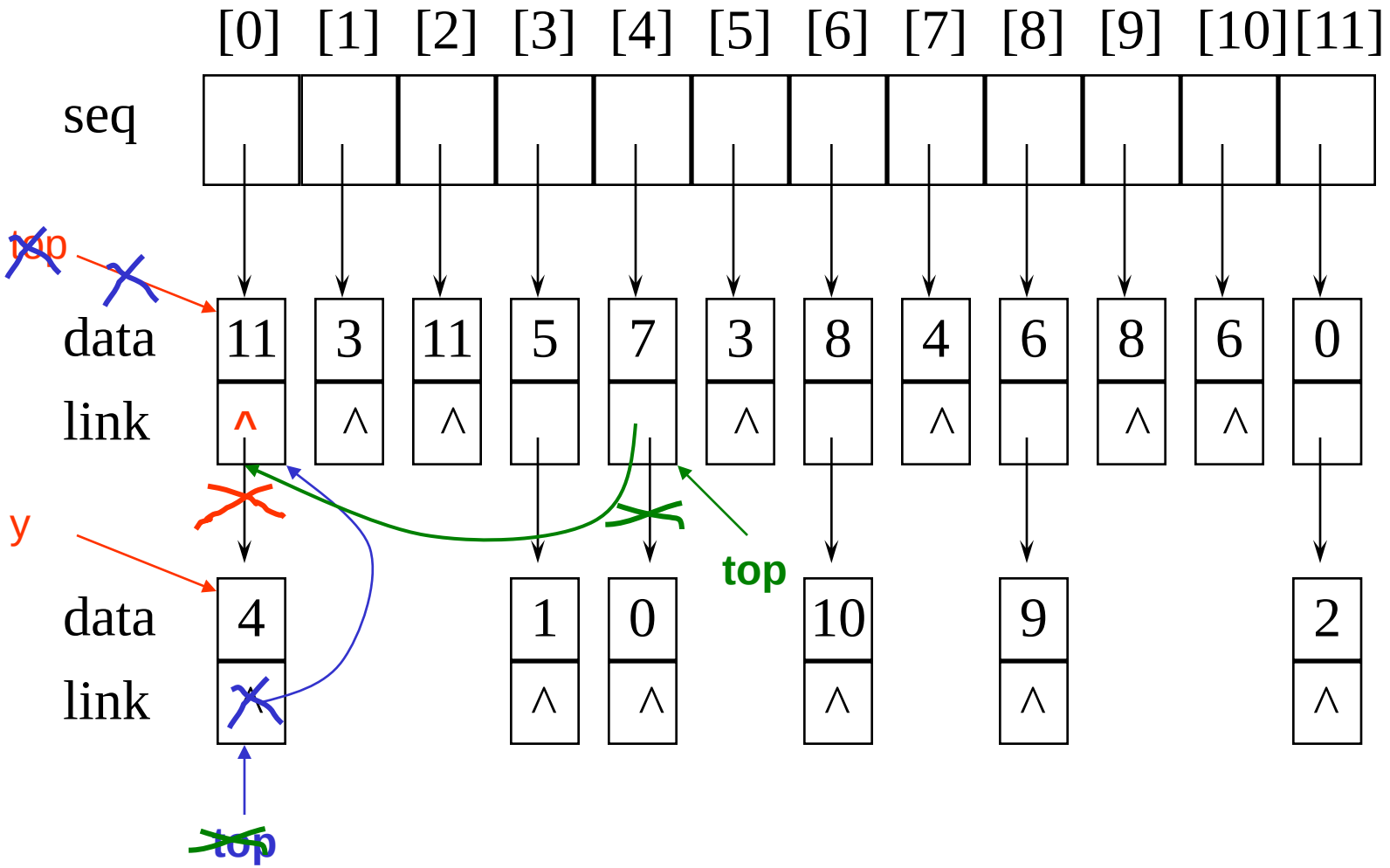
Program 4.23: 최종 동치 알고리즘(2)

```
    for (i = 0; i < n; i++)  // seq[]와 out[] 배열을 초기화
    {     out[i] = TRUE;    seq[i] = NULL;  }
/* Phase 1: Input the equivalence pairs: */
    printf("Enter a pair of numbers (-1 -1 to quit): ");
    scanf("%d%d", &i, &j);
    while (i >= 0)   {        // 음수가 입력되면 리스트 생성 종료
          x = (struct node *) malloc(sizeof(struct node));
          if (x == NULL)   {
                fprintf(stderr, "The memory is full"); exit(1);
          }
          x→data = j;   x→link = seq[i];   seq[i] = x;   // j를 i 리스트의 앞에 추가
          x = (struct node *) malloc(sizeof(struct node));
          if (x == NULL)   {
                fprintf(stderr, "The memory is full"); exit(1);
          }
          x→data = i;   x→link = seq[j];   seq[j] = x;      // i를 j 리스트의 앞에 추가
          printf("Enter a pair of numbers (-1 -1 to quit): ");
          scanf("%d%d", &i, &j);
    }
```

Program 4.23: 최종 동치 알고리즘(3)

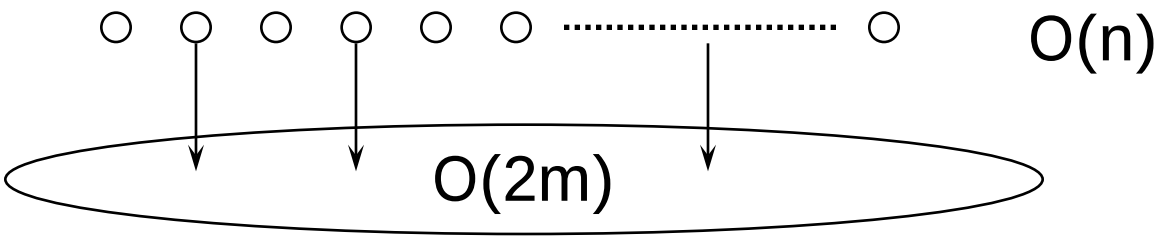
```
    for (i = 0; i < n; i++)       /* Phase 2: output the equivalence classes */
        if (!out[i])          continue;
        printf(" \  n New class: %5d ", i );      // 새로운 클래스 출력 시작
        out[i] = FALSE;                           // i를 출력하였음.
        x = seq[i]; top = NULL;                   // 스택 초기화
        for ( ; ; )    {              // 클래스의 나머지 원소를 찾자
                while (x)   {         // 리스트를 스캔
                        j = x→data;
                        if (out[j])   {     // j가 아직 출력되지 않았다면...
                                printf("%5d", j );   out[j] = FALSE;    // j를 출력한 후,
                                y = x→link; x→link = top; top = x; x = y;  // push()
                        }
                        else x = x→link;
                }
                if (!top)     // 현재 클래스의 모든 원소를 출력하였음.
                        break;
                x = seq[top→data]; top = top→link;          // pop()
}       }
```

예: 스택의 삽입과 삭제 과정



동치 알고리즘의 분석

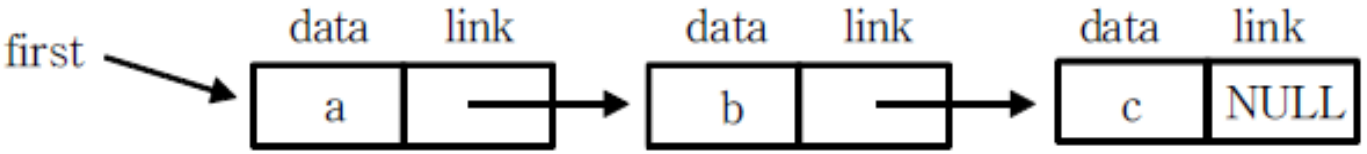
- m: 입력된 동치항의 수, n: 원소의 수
- 초기화 및 동치항의 입력 단계: O(m+n)
- 동치 클래스 출력 단계: O(m+n)
  - 각 노드는 기껏해야 한번 스택에 저장
  - 2*m 개의 노드가 존재하며, for loop는 n 번 실행



- 전체적인 시간 복잡도 = **O(m+n)**

문제 1

- data 필드와 link 필드를 갖는 노드들로 구성된 아래 연결 리스트에서 data 필드가 b인 노드를 삭제하는 C 프로그램의 부분은? 단, first는 리스트를 가리키는 포인터이며, temp는 first와 동일한 타입의 포인터이다. [10]



① temp = first; first = first->link; free(temp);
② first = first→link; temp = first; free(temp);
③ temp = first→link; first→link = first→link→link; free(temp);
④ first→link = first→link→link; temp = first→link; free(temp);
⑤ temp = first→link; free(temp); first→link = first→link→link;

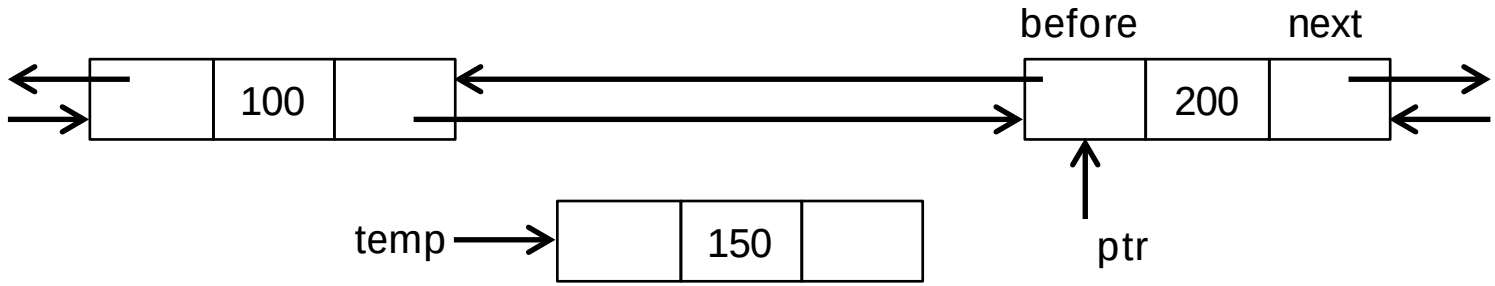
문제 2

- 다음은 원형 연결 리스트(circular linked list)에서 data 필드 값이 0보다 작은 노드들의 개수를 반환하는 함수이다. A와 B에 들어갈 내용은 무엇인가? [10]

```
struct node { intdata; struct node *next; };
int compute_sum(struct node *ptr) {
    struct node *p = ptr→next;
    int count = (ptr→data < 0) ? 1 : 0;
    for ( A )
        if ( B )     count++;
    return count;
}
```

문제 3

- 다음은 이중 연결 리스트(doubly linked list)에서 temp가 가리키는 노드를 ptr이 가리키는 노드의 왼쪽에 삽입하는 알고리즘의 일부이다. A와 B에 들어갈 문장은 무엇인가? 단, ptr의 이전 노드와 다음 노드는 NULL이 아니다. [10]




```
temp→next = ptr;
A ;
B ;
ptr→before = temp;
```

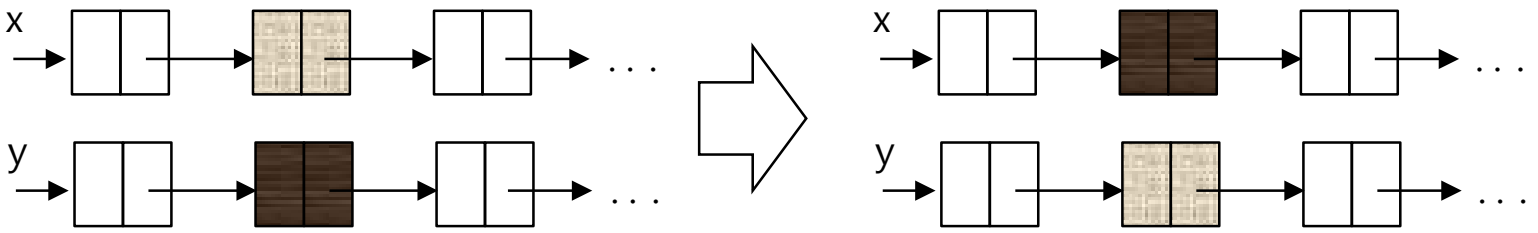
문제 4

- 이중 연결 리스트에서 노드 p의 다음(next) 노드가 q이고 노드 q의 이전(prev) 노드가 p인 경우, 인접한 p, q 노드의 위치를 서로 바꾸기 위하여 아래 연산들이 필요하다. 어떤 순서로 실행되어야 하는가? (단, p, q 노드는 모두 연결 리스트의 처음 노드나 마지막 노드가 아니라고 가정한다) [10]

① p→prev = q;	② p→prev→next = q;	③ p→next = q→next;
④ q→next = p;	⑤ q→next→prev = p;	⑥ q→prev = p→prev;

문제 5

- 두 개의 단순 연결리스트 x, y의 후위 노드(음영부분)들을 서로 스왑(swap)하고자 한다. 후위 노드들의 next는 NULL이 아니라고 가정할 때, 아래 A와 B의 box를 채우시오. [10]




```
struct node {      int   data;     struct node *next;     };
struct node *swap_next_node(struct node *x, struct node *y) {
        struct node *tmp;
        tmp = x→next→next;
        A ;
        tmp = x→next;
        B ;
    }
```