1.1 Binary search

A. Binary search recursive

코드 설계

start 와 end의 값을 변경하여 함수를 호출함으로써 범위를 좁혀 간다.

start 와 end의 중간 위치에 찾으려는 숫자가 존재하는 경우 중간 위치를 반환한다.

start 가 end보다 커질 경우에는 값이 존재하지 않음을 의미하므로 프로그램을 종료한다.

코드 구현

```
if(start > end)
```

start가 end보다 커지는 경우는 값이 없는 경우이므로 0을 반환한다.

```
else
{
    int mid = (start + end) / 2;
    if(A[mid] == key)
        return (mid + 1);
    else if(A[mid] > key)
        BinarySearch(A, start, mid - 1, key);
    else
        BinarySearch(A, mid + 1, end, key);
}
```

mid는 중간 값의 위치를 가리키는 변수이다.

만약 중간 값이 찾으려는 숫자인 경우, 중간 값의 위치를 반환하고

중간 값이 찾으려는 숫자보다 큰 경우는 범위를 중간 값 – 1의 위치를 종료 위치로 설정한 재귀함수를 호출한다.

중간 값이 찾으려는 숫자보다 작은 경우는 중간 값 + 1의 위치를 시작 위치로 설정한 재귀함수를 호출한다.

실행

input.txt : 10 11 12 16 18 23 29
33 48 54 57 68 77 84 98

```
>binarysearch_recursive input.txt
찾으시려는 숫자를 입력해 주세요>> 29
29는 7번째에 위치해 있습니다.
찾으시려는 숫자를 입력해 주세요>> 98
98는 15번째에 위치해 있습니다.
찾으시려는 숫자를 입력해 주세요>> 84
84는 14번째에 위치해 있습니다.
```

B. Binary Search Iterative

코드 설계

재귀함수에서 start와 end값을 바꾸어 함수를 호출하였었으므로, start와 end값만 바꾸어 반복문을 실행한다.

코드 구현

```
while(end >= start)
```

재귀 함수의 경우와 마찬가지로, 시작 위치가 끝 위치보다 크게 되면 값이 존재하지 않으므로 시작 위치가 끝 위치보다 작을 동안 반복을 실행한다.

```
{
    int mid = (start + end) / 2;
    if(A[mid] == key)
        return mid + 1;
    else if(A[mid] > key)
        end = mid - 1;
    else
        start = mid + 1;
}
return 0;
```

중간 값이 찾으려는 숫자인 경우, 중간 위치를 반환하고

중간 값이 찾으려는 숫자보다 큰 경우, 끝 값을 중간 위치 – 1로 설정하여 반복문을 실행한다.

중간 값이 찾으려는 숫자보다 작은 경우, 중간 위치 + 1을 시작 값으로 설정하여 반복문을 실행한다.

값이 존재하지 않는 경우에는 0을 반환한다.

실행

input2.txt : 1 2 4 5 7 8 10 11 13 14 16 17
19 20 22 23 25 26
28 29

```
>binarysearch_recursive input2.txt
찾으시려는 숫자를 입력해 주세요>> 1
1는 1번째에 위치해 있습니다.
찾으시려는 숫자를 입력해 주세요>> 20
20는 14번째에 위치해 있습니다.
찾으시려는 숫자를 입력해 주세요>> 28
28는 19번째에 위치해 있습니다.
```

1.2 k-ary search

코드 설계

배열의 크기가 k 보다 큰 경우에는 배열을 k 개로 분할한다. 분할 방식은 다음과 같다.

배열의 크기 + 1 을 k 로 나누어 탐색 위치의 증가량을 정한다.

탐색 위치는 (시작 위치 – 1 + (I * 탐색 위치의 증가량)) (1 <= I < k)

탐색 위치에 찾으려는 값이 존재하는 경우, 탐색 위치를 반환한다.

탐색 위치의 값보다 작은 경우, 시작 위치는 (이전 탐색 위치 + 1)로 설정하고, 끝 위치는 (현재 탐색 위치 – 1)로 설정한 재귀함수를 호출한다.

반복문을 모두 실행한 후, 탐색 위치의 값보다 큰 경우, 시작 위치만 (현재 탐색 위치 + 1)로 설정한 재귀함수를 호출한다. 반복문을 모두 실행한 경우는 탐색 위치의 값보다 큰 경우 또는 값이 존재하지 않는 경우이다.

k 가 배열의 크기보다 큰 경우, 순차적으로 비교를 실행한다.

예)

배열의 크기가 15 일 때

5 개로 나눌 경우

탐색 위치의 증가량 : (15 + 1) / 5 = 3 (탐색 위치의 증가량은 정수형)

분할된 배열 (a 는 탐색 위치)

2 a 2 a 2 a 2 a 3

2 개를 갖는 파트 4 개와 3 개를 갖는 파트 1 개로 분할됨.

4 개로 나눌 경우

탐색 위치의 증가량 : (15 + 1) / 4 = 4

분할된 배열 (a 는 탐색 위치)

3 a 3 a 3 a 3

3 개를 갖는 파트 4 개로 분할됨

코드 구현

```
int dividedNum = (end - start + 2) / k;
```

dividedNum 은 탐색 위치의 증가량이다.

(end – start + 1)은 배열의 크기이므로 여기에 1 을 더한 뒤 k 로 나눈다.

```
if(start > end)
    return 0;
```

시작 위치가 끝 위치보다 크게 되면 값이 존재하지 않는 것이므로 0 을 반환한다.

```
if(dividedNum == 0)
{
    int remainder = end - start + 1;
    for(int i = 0; i < remainder; i++)
    {
        if(A[start + i] == key)
            return start + i + 1;
    }
    return 0;
}
```

배열의 크기가 k 보다 작은 경우, 배열을 나눌 수 없으므로 순차적으로 비교를 실행한다.

```
else
{
    int checkNum;                                   // 비교를 실행할 값의 위치
    for(int i = 1; i < k; i++)
    {
        checkNum = (start - 1) + i * dividedNum;
        //cout << A[checkNum] << " "; 탐색 위치의 값
        if (A[checkNum] == key)                     // 비교값이 찾는 값일 경우
            return checkNum + 1;                    // 위치 리턴
        else if(A[checkNum] > key)                  // 비교값이 클 경우
        {                                           // 재귀함수 호출
            return KarySearch(A, k, checkNum - dividedNum + 1, checkNum - 1, key);
        }
    }                                               // 마지막 비교값은 배열의 end위치의 값
    if(key > A[end])        // 배열의 마지막 값보다 큰 경우
        return 0;           // 값을 찾을 수 없음
    else
    {                       // 재귀함수 호출
        return KarySearch(A, k, checkNum + 1, end, key);
    }
}
```

비교 위치의 값과 찾으려는 값을 비교하면서 찾으려는 값이 존재하는 파트를 탐색하여 재귀함수를 호출한다.

실행

실행에 사용한 텍스트 파일은 input.txt 와 input2.txt 로 이전에 사용한 텍스트와 같다.

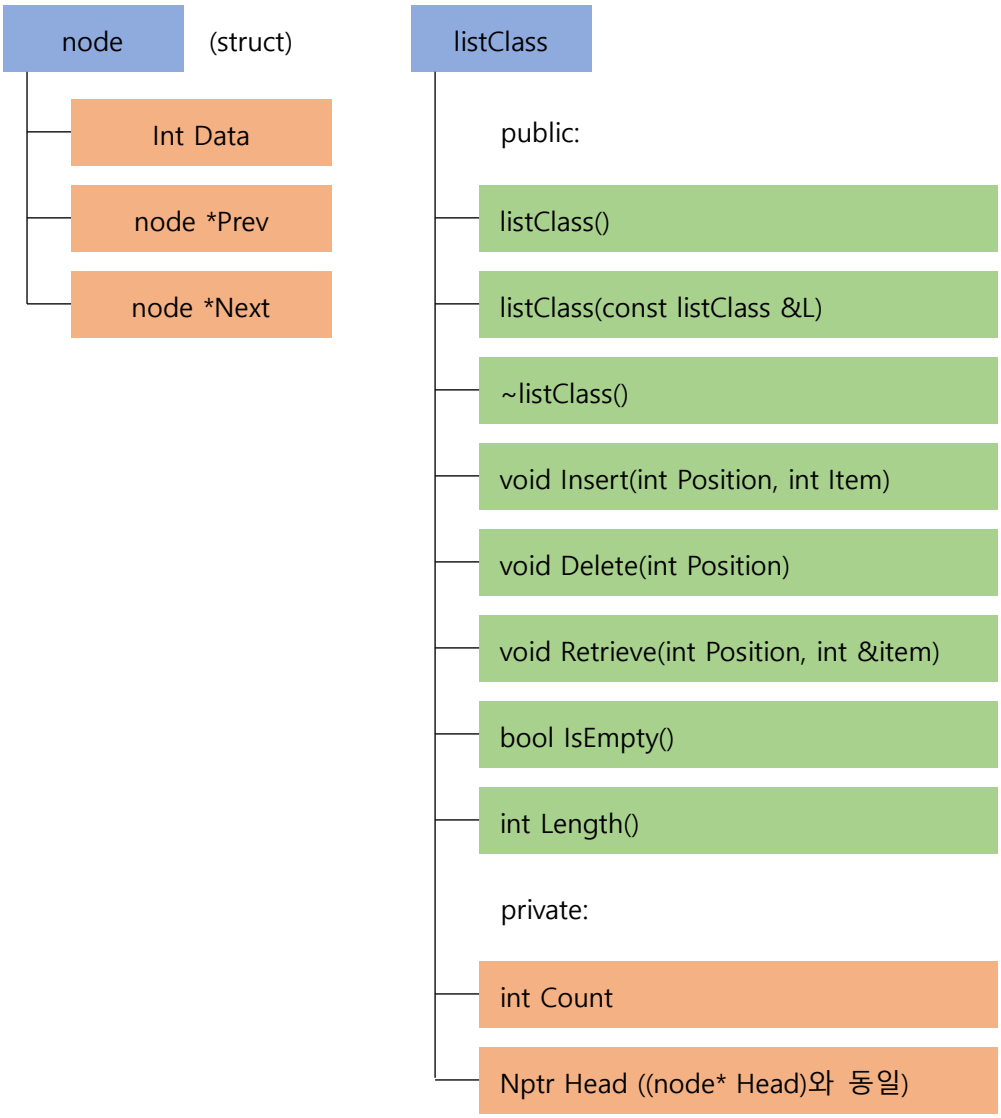
```
>karysearch input.txt
찾으시려는 숫자를 입력해 주세요>> 57
데이터를 몇 개의 파트로 나눌지 입력해 주세요 >> 3
57는 11번째에 위치해 있습니다.
찾으시려는 숫자를 입력해 주세요>> 10
데이터를 몇 개의 파트로 나눌지 입력해 주세요 >> 5
10는 1번째에 위치해 있습니다.

>karysearch input2.txt
찾으시려는 숫자를 입력해 주세요>> 26
데이터를 몇 개의 파트로 나눌지 입력해 주세요 >> 6
26는 18번째에 위치해 있습니다.
```

1.3 이중 연결 리스트 구현

1.3.1 정수형 버전

코드 설계

클래스 계층도



listClass()와 listClass(const listClass &L)에는 Count 와 Head 의 초기화가 필요하다.

~listClass()에서 리스트를 비워줘야 한다.

Insert 함수에서는 Position 값을 검사한 후, 값이 범위 내에 있으면 Item 을 Position 위치에 삽입한다,.

Delete 함수에서는 Position 값을 검사한 후, Position 위치의 값을 삭제한다.

Retrieve 함수에서는 Position 값을 검사한 후, Position 위치의 값을 item 변수에 대입한다.

IsEmpty 함수에서는 Count 값을 검사하여 0 일 경우 true 를 반환하고 0 이 아닐 경우 false 를 반환한다.
Length 함수에서는 Count 값을 반환한다.

코드 구현
listClass() 함수에서는 Count 를 0 으로 설정하고 Head 포인터를 NULL 로 설정한다.

listClass(const listClass &L) 함수에서는 L 의 Count 를 복사하고, 노드들을 순차적으로 복사한다.

Insert() 함수에서는 Position 값이 1 보다 크고 Count + 1 보다 작거나 같은지 검사한 후, 조건을 만족할 경우 새 노드를 생성하여 Position 위치에 대입한다.
Position 이 1 일 경우 Head 포인터가 새 노드를 가리키도록 변경해야 하며 Position 이 Count + 1 일 경우 새 노드의 Next 는 NULL 로 설정해야 한다.

Delete() 함수에서는 리스트가 비었는지 검사한 후, Position 값을 검사하고 조건을 만족할 경우 Position 위치의 노드를 제거한다.
Position 이 1 일 경우 Head 포인터는 다음 노드를 가리키도록 설정해야 하며 Position 이 Count 일 경우 삭제할 노드의 이전 노드의 Next 값을 NULL 로 설정해야 한다.

Retrieve(int Position, int & item) 함수에서는 Position 값을 검사하고 조건을 만족할 경우 Position 위치의 값을 item 에 대입한다.
Position 이 1 일 경우 Head 포인터가 가리키는 노드의 Data 를 대입하고, 1 이 아닌 경우에는 임시 노드 포인터를 만들어 Position 위치로 간 후, 임시 노드 포인터가 가리키는 노드의 Data 를 대입한다.

IsEmpty() 함수에서는 return 을 통해 (Count == 0) 을 반환한다.

Length() 함수에서는 Count 값을 반환한다.

실행

값 삽입

```
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 1
Insert Position and Number >> 1 1
List item : 1
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 1
Insert Position and Number >> 2 3
List item : 1 3
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 1
Insert Position and Number >> 2 2
List item : 1 2 3
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 1
Insert Position and Number >> 5 5
Position out of Range
List item : 1 2 3
```

Position 을 5 로 설정한 경우 범위를 벗어나므로 오류 메시지가 표시된다.

값 검색

```
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 3
Retrieve Position >> 0
Position out of Range
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 3
Retrieve Position >> 1
item : 1
List item : 1 2 3
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 3
Retrieve Position >> 4
Position out of Range
```

Position 을 0 또는 4 로 설정한 경우 범위를 벗어나므로 오류 메시지가 표시된다.

값 제거

```
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 2
Delete Position >> 4
Position out of Range
List item : 1 2 3
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 2
Delete Position >> 1
List item : 2 3
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 2
Delete Position >> 1
List item : 3
```

리스트 복사

```
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 4
List1 item : 1 2 3
New List item : 1 2 3
```

빈 리스트인지 확인

```
List item : 1 2 3
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 5
Not Empty
```

리스트를 비우고 확인

```
List item :
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 5
Empty
```

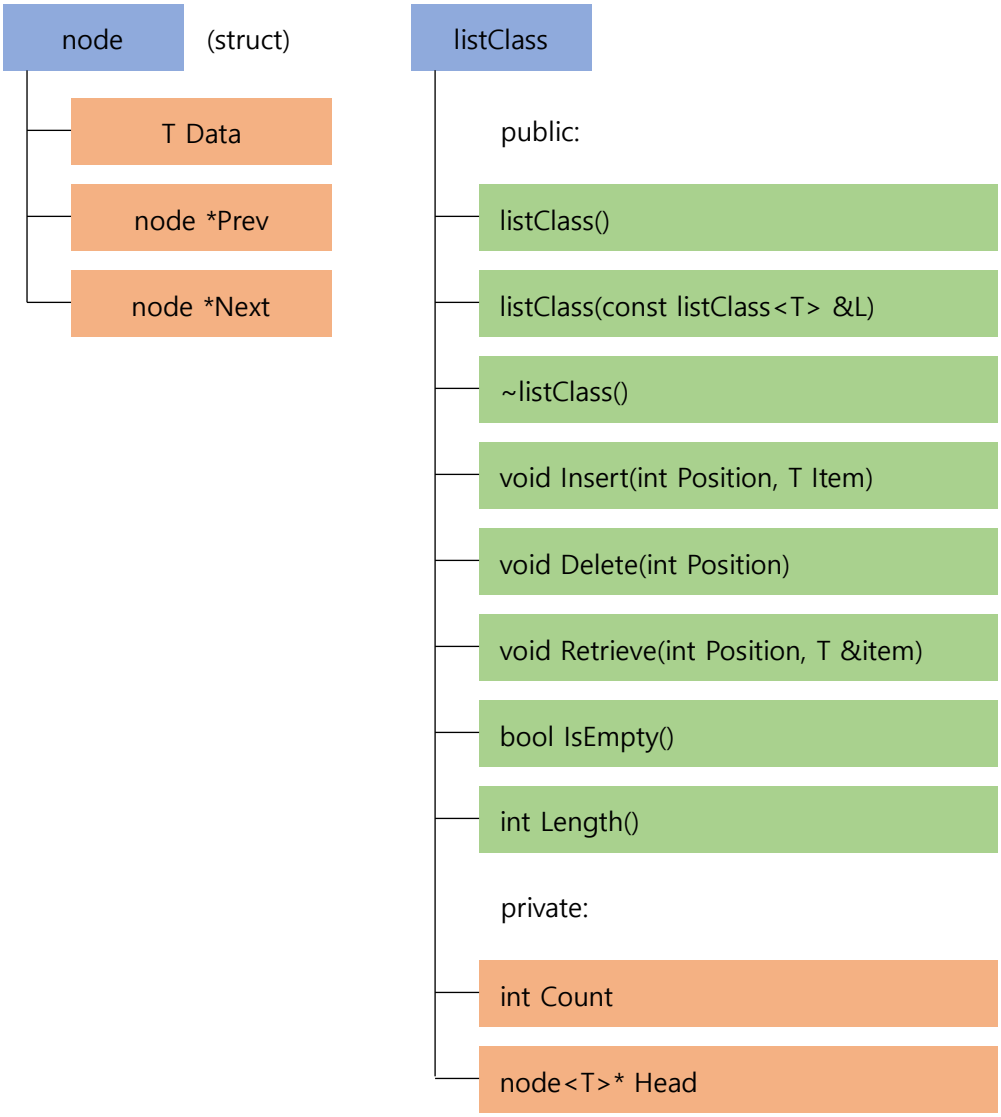
리스트 길이 확인

```
List item : 1 2 3
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 6
3
```

1.3.2 template 버전

코드 설계

클래스 계층도



정수형 버전과 동일한 구조로 작성하되, 데이터 타입만 T 로 바꾸어 구현한다.

실행

Int 리스트에 item 삽입

```
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 1
int List : 1, double List : 2, string List : 3 >> 1
Insert Position and item >> 1 1
List item : 1
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 1
int List : 1, double List : 2, string List : 3 >> 1
Insert Position and item >> 2 2
List item : 1 2
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 1
int List : 1, double List : 2, string List : 3 >> 1
Insert Position and item >> 3 3
List item : 1 2 3
```

Double 리스트에 item 삽입

```
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 1
int List : 1, double List : 2, string List : 3 >> 2
Insert Position and item >> 1 1.1
List item : 1.1
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 1
int List : 1, double List : 2, string List : 3 >> 2
Insert Position and item >> 2 2.2
List item : 1.1 2.2
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 1
int List : 1, double List : 2, string List : 3 >> 2
Insert Position and item >> 3 3.3
List item : 1.1 2.2 3.3
```

String 리스트에 item 삽입

```
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 1
int List : 1, double List : 2, string List : 3 >> 3
Insert Position and item >> 1 a
List item : a
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 1
int List : 1, double List : 2, string List : 3 >> 3
Insert Position and item >> 2 b
List item : a b
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 1
int List : 1, double List : 2, string List : 3 >> 3
Insert Position and item >> 3 c
List item : a b c
```

값 검색

```
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 3
int List : 1, double List : 2, string List : 3 >> 1
Retrieve Position >> 1
item : 1
List item : 1 2 3
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 3
int List : 1, double List : 2, string List : 3 >> 2
Retrieve Position >> 2
item : 2.2
List item : 1.1 2.2 3.3
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 3
int List : 1, double List : 2, string List : 3 >> 3
Retrieve Position >> 3
item : c
List item : a b c
```

값 삭제

```
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 2
int List : 1, double List : 2, string List : 3 >> 1
Delete Position >> 3
List item : 1 2
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 2
int List : 1, double List : 2, string List : 3 >> 2
Delete Position >> 2
List item : 1.1 3.3
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 2
int List : 1, double List : 2, string List : 3 >> 3
Delete Position >> 1
List item : b c
```

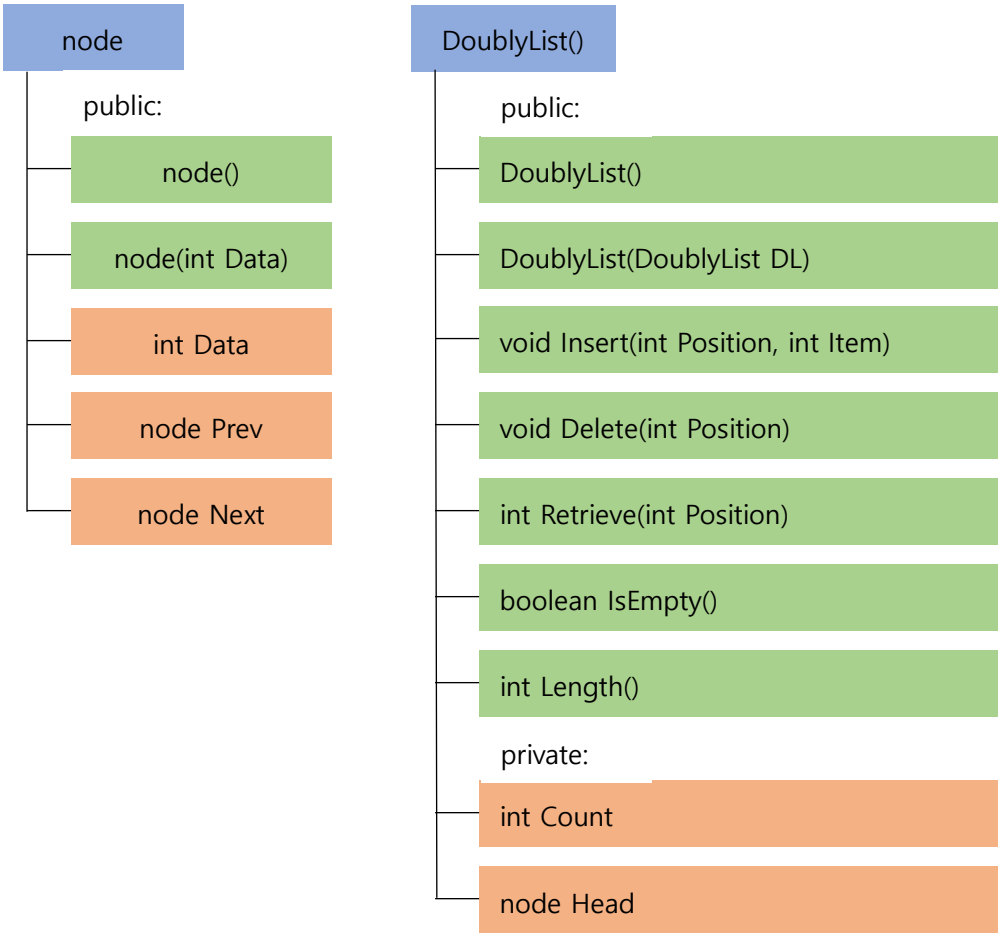
리스트 복사

```
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 4
int List : 1, double List : 2, string List : 3 >> 1
New int list : 1 2
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 4
int List : 1, double List : 2, string List : 3 >> 2
New double list : 1.1 3.3
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 4
int List : 1, double List : 2, string List : 3 >> 3
New string list : b c
```

1.3.3 java 버전

코드 설계

클래스 계층도



Struct 가 존재하지 않으므로 class 로 노드를 만든다.
소멸자는 제거한다.
포인터가 존재하지 않으므로 Retrieve 의 반환형을 int 형으로 설정하고 노드 포인터들을 node 로 변경한다.

코드 구현
코드의 구성은 동일하지만 자바의 특성에 맞추어 '->'를 통해 포인터가 가리키는 값의 요소에 접근하던 코드를 '.'을 통해 클래스의 요소에 접근한다.

실행
값 삽입

```
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 1
Insert Position and Number >> 1 1
List item : 1
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 1
Insert Position and Number >> 2 3
List item : 1 3
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 1
Insert Position and Number >> 2 2
List item : 1 2 3
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 1
Insert Position and Number >> 5 5
Position out of Range
List item : 1 2 3
```

값 검색

```
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 3
Retrieve Position >> 1
item : 1
List item : 1 2 3
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 3
Retrieve Position >> 2
item : 2
List item : 1 2 3
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 3
Retrieve Position >> 3
item : 3
List item : 1 2 3
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 3
Retrieve Position >> 4
Position out of Range
```

값 제거

```
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 2
Delete Position >> 1
List item : 2 3
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 2
Delete Position >> 3
Position out of Range
List item : 2 3
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 2
Delete Position >> 1
List item : 3
```

리스트가 비었는지 확인

```
List item : 1 2 3
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 5
Not Empty
```

리스트를 비우고 확인

```
List item :
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 5
Empty
```

리스트 길이

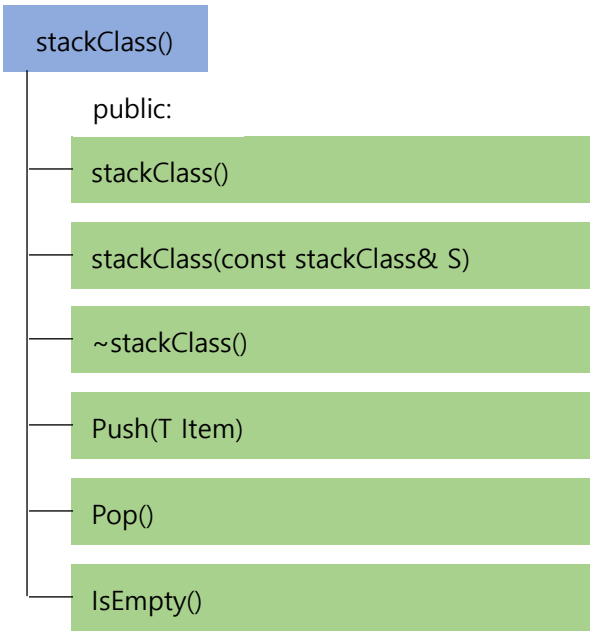
```
List item : 1 2 3
Insert : 1, Delete : 2, Retrieve : 3, Copy List : 4, Is Empty? : 5, Length : 6, Exit : 0 >> 6
3
```

1.3.4 스택과 큐 구현

A. 스택 구현

코드 설계

클래스 계층도



stackClass()함수와 stackClass(const stackClass& S)함수에서 ListClass 를 주어진 자료형으로 만들고 ~stackClass()함수에서 ListClass 를 제거한다.

Push(T Item)함수에선 stack 에 Item 을 쌓고 Pop()함수에선 stack 의 Item 을 꺼낸다.

IsEmpty()함수에서는 ListClass 의 IsEmpty()함수를 통해 스택이 비었는지를 확인한다.

값을 삽입하고 빼는 위치는 1 로 고정한다.

코드 구현

stackClass()함수에서 ListClass 를 주어진 자료형으로 만든다.

stackClass(const stackClass& S)함수에서 S 의 ListClass 의 자료형으로 새 ListClass 를 만들고 생성했던 복사 생성자를 통해 리스트를 복사한다.

~stackClass()함수에서는 동적으로 생성된 ListClass 를 제거한다.

Push(T Item)함수에서는 ListClass 의 맨 처음에 Item 을 삽입한다.

Pop()함수에서는 ListClass 의 Retrieve(int position, T& item)함수를 통해 값을 구한 뒤, Delete(int Position)함수를 통해 맨 앞의 값을 제거한다.

IsEmpty() 함수에서는 ListClass 의 IsEmpty()함수를 통해 스택이 비었는지를 확인한다.

실행

int 스택 값 대입

```
Push : 1, Pop : 2, Copy Stack : 3, Is Empty? : 4, Exit : 0 >> 1
Select stack (int : 1, double : 2, string : 3, Exit : 0) >> 1
Push int >> 1
Push : 1, Pop : 2, Copy Stack : 3, Is Empty? : 4, Exit : 0 >> 1
Select stack (int : 1, double : 2, string : 3, Exit : 0) >> 1
Push int >> 2
Push : 1, Pop : 2, Copy Stack : 3, Is Empty? : 4, Exit : 0 >> 1
Select stack (int : 1, double : 2, string : 3, Exit : 0) >> 1
Push int >> 3
```

double 스택 값 대입

```
Push : 1, Pop : 2, Copy Stack : 3, Is Empty? : 4, Exit : 0 >> 1
Select stack (int : 1, double : 2, string : 3, Exit : 0) >> 2
Push double >> 1.1
Push : 1, Pop : 2, Copy Stack : 3, Is Empty? : 4, Exit : 0 >> 1
Select stack (int : 1, double : 2, string : 3, Exit : 0) >> 2
Push double >> 2.2
Push : 1, Pop : 2, Copy Stack : 3, Is Empty? : 4, Exit : 0 >> 1
Select stack (int : 1, double : 2, string : 3, Exit : 0) >> 2
Push double >> 3.3
```

string 스택 값 대입

```
Push : 1, Pop : 2, Copy Stack : 3, Is Empty? : 4, Exit : 0 >> 1
Select stack (int : 1, double : 2, string : 3, Exit : 0) >> 3
Push string >> a
Push : 1, Pop : 2, Copy Stack : 3, Is Empty? : 4, Exit : 0 >> 1
Select stack (int : 1, double : 2, string : 3, Exit : 0) >> 3
Push string >> b
Push : 1, Pop : 2, Copy Stack : 3, Is Empty? : 4, Exit : 0 >> 1
Select stack (int : 1, double : 2, string : 3, Exit : 0) >> 3
Push string >> c
```

스택 복사

```
Push : 1, Pop : 2, Copy Stack : 3, Is Empty? : 4, Exit : 0 >> 3
Items of int Stack :
3 2 1
Items of double Stack :
3.3 2.2 1.1
Items of string Stack :
c b a
```

스택을 Pop

```
Push : 1, Pop : 2, Copy Stack : 3, Is Empty? : 4, Exit : 0 >> 2
Pop one item : 1, Pop all items : 2 (Exit : 0) >> 2
Select stack (int : 1, double : 2, string : 3, Exit : 0) >> 1
3 2 1
Push : 1, Pop : 2, Copy Stack : 3, Is Empty? : 4, Exit : 0 >> 2
Pop one item : 1, Pop all items : 2 (Exit : 0) >> 1
Select stack (int : 1, double : 2, string : 3, Exit : 0) >> 3
Item : c
Push : 1, Pop : 2, Copy Stack : 3, Is Empty? : 4, Exit : 0 >> 2
Pop one item : 1, Pop all items : 2 (Exit : 0) >> 1
Select stack (int : 1, double : 2, string : 3, Exit : 0) >> 3
Item : b
Push : 1, Pop : 2, Copy Stack : 3, Is Empty? : 4, Exit : 0 >> 2
Pop one item : 1, Pop all items : 2 (Exit : 0) >> 2
Select stack (int : 1, double : 2, string : 3, Exit : 0) >> 2
3.3 2.2 1.1
```

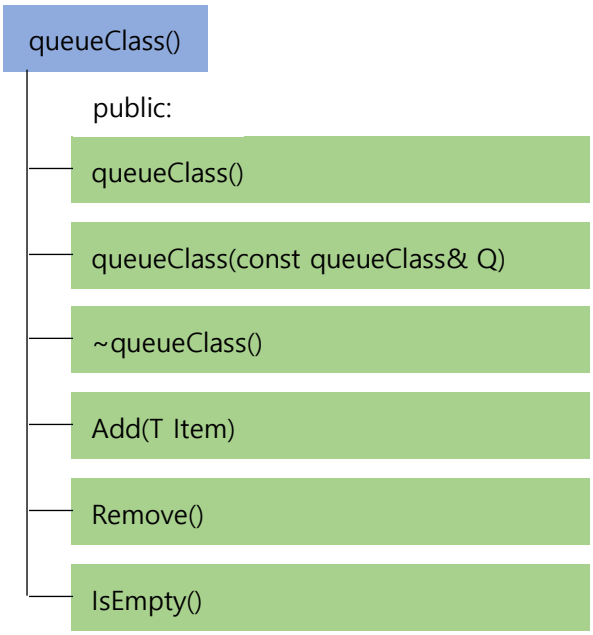
스택이 비었는지 검사

```
Push : 1, Pop : 2, Copy Stack : 3, Is Empty? : 4, Exit : 0 >> 4
Select stack (int : 1, double : 2, string : 3, Exit : 0) >> 1
Yes
Push : 1, Pop : 2, Copy Stack : 3, Is Empty? : 4, Exit : 0 >> 4
Select stack (int : 1, double : 2, string : 3, Exit : 0) >> 2
Yes
Push : 1, Pop : 2, Copy Stack : 3, Is Empty? : 4, Exit : 0 >> 4
Select stack (int : 1, double : 2, string : 3, Exit : 0) >> 3
No
```

B. 큐 구현

코드 설계

클래스 계층도



queueClass()함수와 queueClass(const queueClass& Q)함수에서 ListClass 를 주어진 자료형으로 만들고 ~queueClass()함수에서 ListClass 를 제거한다.
Add (T Item)함수에선 queue 에 Item 을 넣고 Remove()함수에선 queue 의 Item 을 꺼낸다.
IsEmpty()함수에서는 ListClass 의 IsEmpty()함수를 통해 스택이 비었는지를 확인한다.
값을 삽입하고 빼는 위치는 1 로 고정한다.

코드 구현

queueClass()함수에서 ListClass 를 주어진 자료형으로 만든다.
queueClass(const queueClass& Q)함수에서 Q 의 ListClass 의 자료형으로 새 ListClass 를 만들고 생성했던 복사 생성자를 통해 리스트를 복사한다.
~queueClass()함수에서는 동적으로 생성된 ListClass 를 제거한다.
Add(T Item)함수에서는 ListClass 의 맨 뒤에 Item 을 삽입한다.
Remove()함수에서는 ListClass 의 Retrieve(int position, T& item)함수를 통해 값을 구한 뒤, Delete(int Position)함수를 통해 맨 앞의 값을 제거한다.
IsEmpty() 함수에서는 ListClass 의 IsEmpty()함수를 통해 스택이 비었는지를 확인한다.

실행

int 큐 값 대입

```
Add : 1, Remove : 2, Copy Queue : 3, Is Empty? : 4, Exit : 0 >> 1
Select queue (int : 1, double : 2, string : 3, Exit : 0) >> 1
Push int >> 1
Add : 1, Remove : 2, Copy Queue : 3, Is Empty? : 4, Exit : 0 >> 1
Select queue (int : 1, double : 2, string : 3, Exit : 0) >> 1
Push int >> 2
Add : 1, Remove : 2, Copy Queue : 3, Is Empty? : 4, Exit : 0 >> 1
Select queue (int : 1, double : 2, string : 3, Exit : 0) >> 1
Push int >> 3
```

double 큐 값 대입

```
Add : 1, Remove : 2, Copy Queue : 3, Is Empty? : 4, Exit : 0 >> 1
Select queue (int : 1, double : 2, string : 3, Exit : 0) >> 2
Push double >> 1.1
Add : 1, Remove : 2, Copy Queue : 3, Is Empty? : 4, Exit : 0 >> 1
Select queue (int : 1, double : 2, string : 3, Exit : 0) >> 2
Push double >> 2.2
Add : 1, Remove : 2, Copy Queue : 3, Is Empty? : 4, Exit : 0 >> 1
Select queue (int : 1, double : 2, string : 3, Exit : 0) >> 2
Push double >> 3.3
```

string 큐 값 대입

```
Add : 1, Remove : 2, Copy Queue : 3, Is Empty? : 4, Exit : 0 >> 1
Select queue (int : 1, double : 2, string : 3, Exit : 0) >> 3
Push string >> a
Add : 1, Remove : 2, Copy Queue : 3, Is Empty? : 4, Exit : 0 >> 1
Select queue (int : 1, double : 2, string : 3, Exit : 0) >> 3
Push string >> b
Add : 1, Remove : 2, Copy Queue : 3, Is Empty? : 4, Exit : 0 >> 1
Select queue (int : 1, double : 2, string : 3, Exit : 0) >> 3
Push string >> c
```

큐 복사

```
Add : 1, Remove : 2, Copy Queue : 3, Is Empty? : 4, Exit : 0 >> 3
Items of int Queue :
1 2 3
Items of double Queue :
1.1 2.2 3.3
Items of string Queue :
a b c
```

큐 Remove 함수

```
Add : 1, Remove : 2, Copy Queue : 3, Is Empty? : 4, Exit : 0 >> 2
Remove one item : 1, Remove all items : 2 (Exit : 0) >> 2
Select queue (int : 1, double : 2, string : 3, Exit : 0) >> 1
1 2 3
Add : 1, Remove : 2, Copy Queue : 3, Is Empty? : 4, Exit : 0 >> 2
Remove one item : 1, Remove all items : 2 (Exit : 0) >> 1
Select queue (int : 1, double : 2, string : 3, Exit : 0) >> 3
Item : a
Add : 1, Remove : 2, Copy Queue : 3, Is Empty? : 4, Exit : 0 >> 2
Remove one item : 1, Remove all items : 2 (Exit : 0) >> 1
Select queue (int : 1, double : 2, string : 3, Exit : 0) >> 3
Item : b
```

큐가 비었는지 확인

```
Add : 1, Remove : 2, Copy Queue : 3, Is Empty? : 4, Exit : 0 >> 4
Select queue (int : 1, double : 2, string : 3, Exit : 0) >> 1
Yes
Add : 1, Remove : 2, Copy Queue : 3, Is Empty? : 4, Exit : 0 >> 4
Select queue (int : 1, double : 2, string : 3, Exit : 0) >> 2
No
Add : 1, Remove : 2, Copy Queue : 3, Is Empty? : 4, Exit : 0 >> 4
Select queue (int : 1, double : 2, string : 3, Exit : 0) >> 3
No
```

1.4 calculator 구현

코드 설계

스택을 두 개 만들어 각각 연산자와 숫자를 저장한다. 닫는 괄호가 나오면 스택에서 숫자 두개와 연산자 하나를 받아와서 연산을 실행한다. 함수를 입력할 시, 값을 계산하여 스택에 쌓는다.

코드 구현

함수

bool is_operator(string k)	연산자를 입력했는지 확인하는 함수
bool is_func(string k)	함수를 입력했는지 확인하는 함수
double calc(double x, double y, string op)	계산하는 함수

연산이 괄호로 감싸여 있지 않을 경우의 오류 처리를 한다. 함수를 입력했을 경우, 각각의 함수에 대해 연산을 실행하여 값을 스택에 넣는다. 닫는 괄호가 나올 시, 스택에서 숫자 두 개와 연산자 하나를 받는다. 먼저 꺼낸 숫자를 x, 나중 숫자를 y 라고 할 시 연산은 y op x 가 된다. 연산 결과는 다시 스택에 넣는다.

실행

```
>> ( 1 + ( ( 5 + 8 ) * ( 10 + 7.5 ) ) )
228.5
>> ( ( 2 + sqrt ( 10.0 ) ) * ( log ( 2.0 ) + 5 ) )
27.5977
>> ( ( 2 + pow ( 5.0 , 3.0 ) + log ( 7.0 ) ) / 3.0 )
42.5486
```

```
>> ( 5 + 6
Invalid input
>> pow ( 6 7 ) + log ( 5 )
Invalid input
>> ( 4 + 2 ) * 10 )
Invalid input
```

1.5 깊이 우선 탐색: depth-first search 구현

코드 설계

모든 edge 의 수를 구해 배열을 만든 후, 배열에 edge 를 저장한다. 이 배열을 갖고 깊이 우선 탐색을 실행한다.

코드 구현

함수

int GetPairNum(ifstream& fin)	모든 edge 의 수를 구하는 함수

공백과 줄의 수를 구해서 edge 의 수를 계산한다.

int GetNodeNum(int ary[][2], int arySize)	모든 노드의 수를 구하는 함수

노드를 저장하는 리스트를 생성하여 다음 노드가 저장되어 있는지를 확인하여 노드 개수를 구함. 저장되어 있지 않으면 노드를 저장.

void PrintAry(int ary[][2], int arySize)	모든 edge 를 출력하는 함수

int (*MakeAry(ifstream& fin, int& size))[2]	edge 를 ary 로 만드는 함수

파일을 읽어서 존재하는 edge 들을 동적으로 할당한 배열에 저장하여 반환한다.

void DFS(int ary[][2], int arySize, int start, int end)	깊이 우선 탐색을 하는 함수

edge 가 저장되어 있는 배열과 배열 크기, 시작 노드와 끝 노드를 인자로 받는다.
처음 실행하면 스택을 만들어서 시작 노드를 스택에 넣는다. 그 뒤, 스택이 빌 때까지 다음 작업을 하는 반복문을 실행한다.

1. 스택의 값을 얻은 후 값이 목적지인지를 확인한다. 만약 목적지일 경우, 시작점에서 목적지로의 경로를 출력하고 반복을 종료한다.
2. 스택에서 얻은 값을 시작점으로 하는 edge 들을 검색하여 도착점들을 얻고, 도착점이 방문하지 않은 노드일 경우 방문한 노드로 설정한 뒤, 스택에 쌓는다.

이 함수에서는 경로를 출력하기 위해 스택 3 개를 추가로 생성하였다.
각 스택이 하는 일은 다음과 같다.
스택 1 : 기본 스택에서 Pop 한 모든 노드를 저장한다. 이 노트가 경로를 구하는 스택이 된다.
스택 2 : 지나온 모든 노드를 저장한다. 만약 edge 를 검색하여 얻은 도착점이 방문했던 노드일 경우, 스택 1 의 Top 노드와 이 스택의 Top 노드의 값이 같아질 때까지 스택 1 에 Pop 을 실행한다.
스택 3 : 값을 출력할 때, 스택 1 의 경로를 Pop 하면 순서가 반대로 출력되므로 이 스택에 Push 한 후 Pop 을 하여 정상적인 순서로 경로가 출력되도록 한다.

실행

sample_graph.txt :

```
1 2 7
2 3 5
3 4
4 2
5 6 7
7 8
8 9
9 2
```

sample_graph2.txt :

```
10 11 12
11 14
12 13
13 11
14 15 16
16 13 17
```

sample_graph3.txt :

```
1 2 4
2 3 4
3 6
4 9
5 7
6 5
7 6 8
8 9 10
```

```
>DFS sample_graph.txt
enter starting and ending vertices (Exit : 0 0)>> 1 6
path length: 4
1 2 5 6
enter starting and ending vertices (Exit : 0 0)>> 5 1
Not Found
enter starting and ending vertices (Exit : 0 0)>> 1 9
path length: 4
1 7 8 9
```

```
>DFS sample_graph2.txt
enter starting and ending vertices (Exit : 0 0)>> 10 17
path length: 5
10 11 14 16 17
enter starting and ending vertices (Exit : 0 0)>> 16 15
path length: 5
16 13 11 14 15
enter starting and ending vertices (Exit : 0 0)>> 11 10
Not Found
```

```
>DFS sample_graph3.txt
enter starting and ending vertices (Exit : 0 0)>> 1 10
path length: 8
1 2 3 6 5 7 8 10
enter starting and ending vertices (Exit : 0 0)>> 2 9
path length: 3
2 4 9
enter starting and ending vertices (Exit : 0 0)>> 4 8
Not Found
```