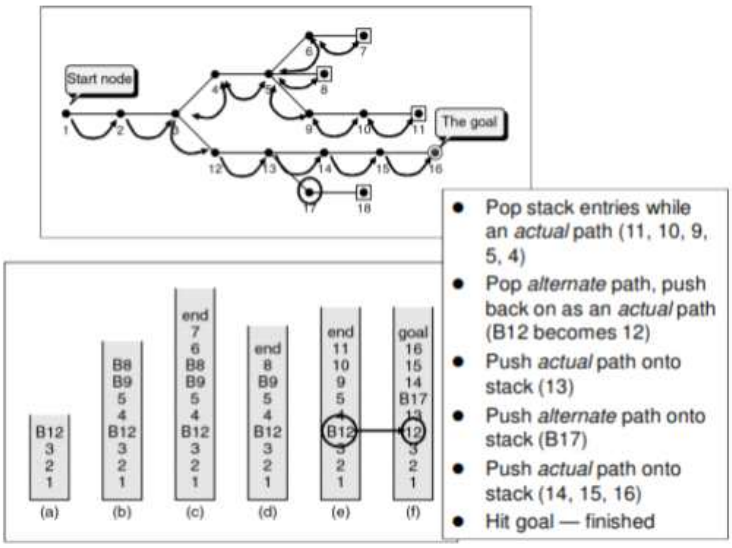
과목명	자료구조 및 알고리즘	분반	x	담당교수	김화성 교수님
학과	전자통신공학과	학번	2016707079	이름	하상천
Goal Seeking					

1, 과제설명 (과제에 대한 설명 및 목표)

다음 Goal seeking 문제의 해결 방법인 아래 Pseudo code를 구현하라. (C 혹은 C++ 사용)

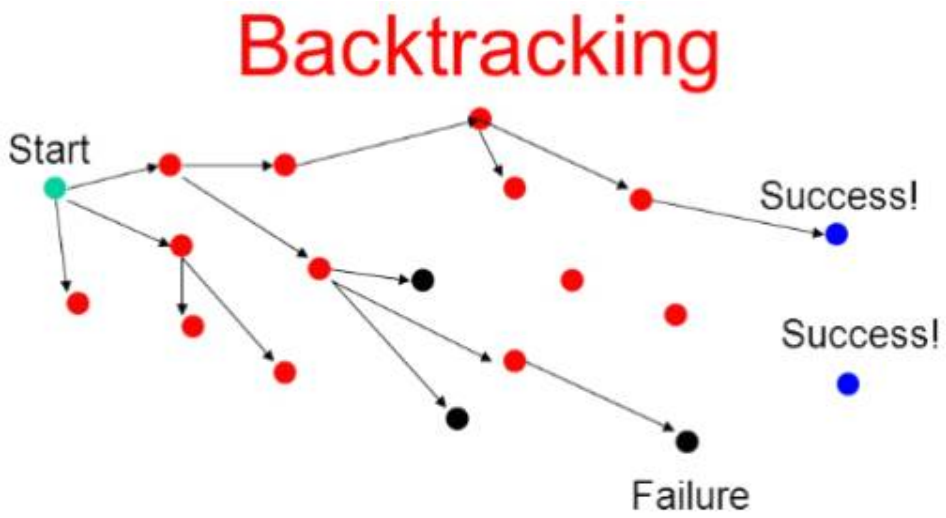
```
Algorithm seekGoal (map)
1 createStack (stack)
2 set pMap to starting point
3 loop (pMap not null AND goalNotFound)
  1  if (pMap is goal)
     1  set goalNotFound to false
  2  else
     1  pushStack (stack, pMap)
     2  if (pMap is a branch point)
        1  loop (more branch points)
           1  create branchPoint node
           2  pushStack (stack, branchPoint)
        2  end loop
     3  end if
     4  advance to next node
  3  end if
4 end loop
5 if (emptyStack (stack))
  1  print (There is no path to your goal)
6 else
  1  print (The path to your goal is:)
  2  loop (not emptyStack (stack))
     1  popStack (stack, pMap)
     2  if (pMap not branchPoint)
        1  print(map point)
     3  end if
  3  end loop
  4  print (End of Path)
7 end if
8 return
end seekGoal
```

2, 이론 (과제와 관련한 일반적인 내용)



스택과 백트래킹을 이용해서 goal을 찾는다.

3. 알고리즘 및 자료구조 설계 내용 기술



백트래킹이란 어떤 노드의 유망성을 점검하고 유망하지 않으면 그 노드의 부모 노드로 돌아간 후 다른 자손의 노드를 검색하는 것을 말한다. 즉, 스택에 자식노드를 넣기 전에 유망한지(해답이 될 가능성이 있는지) 확인하고 스택에 넣음을 말한다.

4. 소스코드 설명 (직접 작성한 소스코드 중에 핵심 부분을 발췌하여 설명)

```
typedef struct tTreeNode {  //이진트리 노드의 구조체
        element data;
        struct tTreeNode *left, *right;
        boolean key;
} TreeNode;
typedef struct {  //스택의 구조체
        TreeNode **stack;  //TreeNode의 포인터형을 삽입할 배열
        int max_size;  // 최대사이즈
        int top;
} Stack;
```

Stack의 구조체와 TreeNode의 구조체를 선언한 내용이다. **stack을 사용한 이유는 TreeNode의 포인터형을 배열에 삽입할 것이기 때문에 이중포인터를 사용하였다.

```
TreeNode* CreateTree(TreeNode *left, element item, TreeNode *right)
{
        TreeNode *pNewNode = (TreeNode*)malloc(sizeof(TreeNode));
        if (pNewNode == NULL)
                return NULL;
        pNewNode->data = item;
        pNewNode->left = left;
        pNewNode->right = right;
        pNewNode->key = false; // false로 초기화 해준다.
        return pNewNode;
}
```

TreeNode의 size 만큼 동적 메모리를 할당 받고, TreeNode 포인터 형으로 형변환하여 pNewNode에 대입하였다. parameter로 받은 left, item, right를 대입하고, key 값을 false로 초기화 해주었다.
key 값은 branch point인지 아닌지를 알려주는 flag이다.

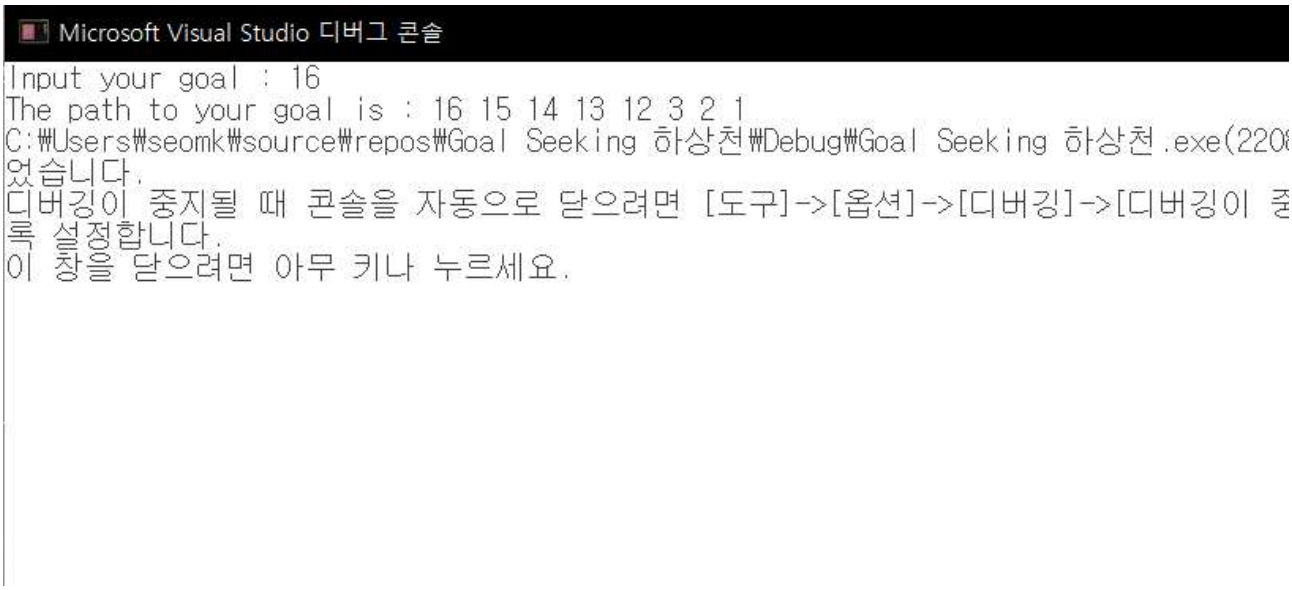
```
if (pMap == NULL) {
                        while (true) {
                                if (isEmpty(pStack)) {
                                        printf("Goal is not founded.");
                                        finish = true; //while문 탈출을 위해 true로 바꾸어주고 break문을 통해 가까운 반복문 탈출
                                        break;
                                }
                                pMap = pop(pStack); // branch point까지 노드를 하나씩 꺼내준다.
                                if (pMap->key == true) {
                                        pMap->key = false; // goal path를 출력하기 위해 key를 false로 하여 다시 stack에 삽입
                                        push(pStack, pMap);
                                        pMap = pMap->left; // branch point에서 반대쪽 노드로 이동
                                        break;
                                }
                    }
}
```

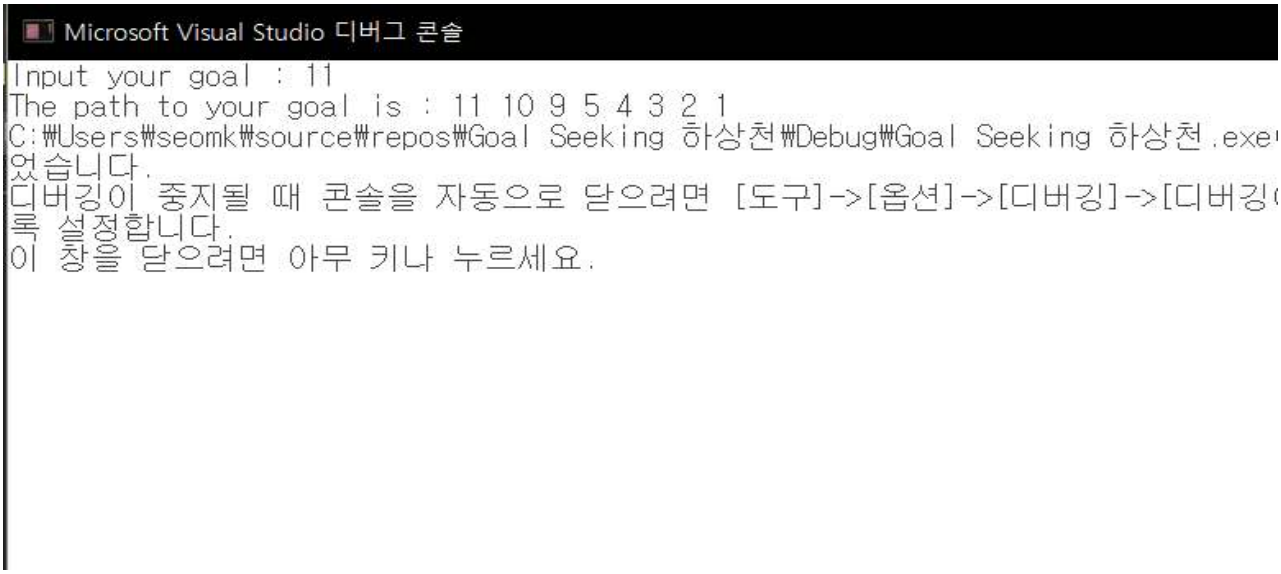
pMap이 NULL이라면 반복문을 통해 stack에서 pop을 해준다. 먼저 stack이 비었다면, goal을 못 찾은 것이기 때문에 finish를 true로 바꾸어주고, break를 통해 가까운 while문을 탈출하고 finish를 통해 또 다른 while문을 탈출한다. stack이 비어있지 않다면 노드를 하나씩 꺼내서 pMap에 삽입하고, pMap의 key가 true이면 즉, branch point 이면 pMap의 key를 false로 해서 다시 stack에 삽입해주고 pMap을 pMap->left로 삽입한다.

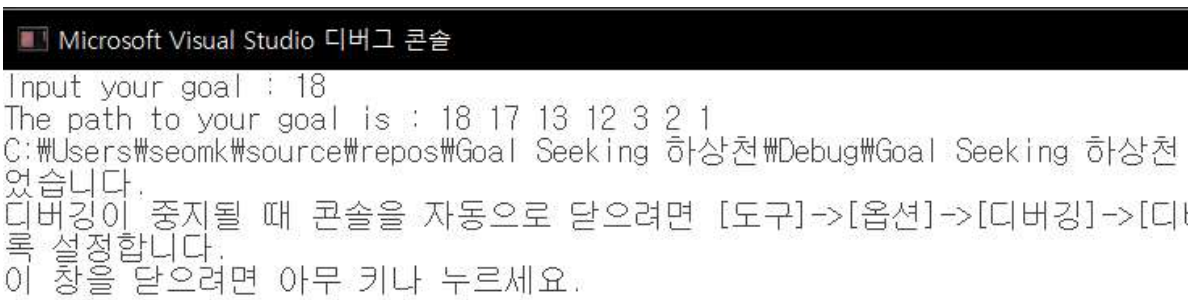
```
if (goalFound) { // goal을 찾았다면
                printf("The path to your goal is : ");
                while (true) {
                        if (!isEmpty(pStack)) { // stack이 비어있지 않다면 pop을 통해 goal path 출력
                                printf("%d ", pop(pStack)->data);
                        }
                        else
                                break;  // stack이 비어있다면 반복문 탈출
                }
        }
```

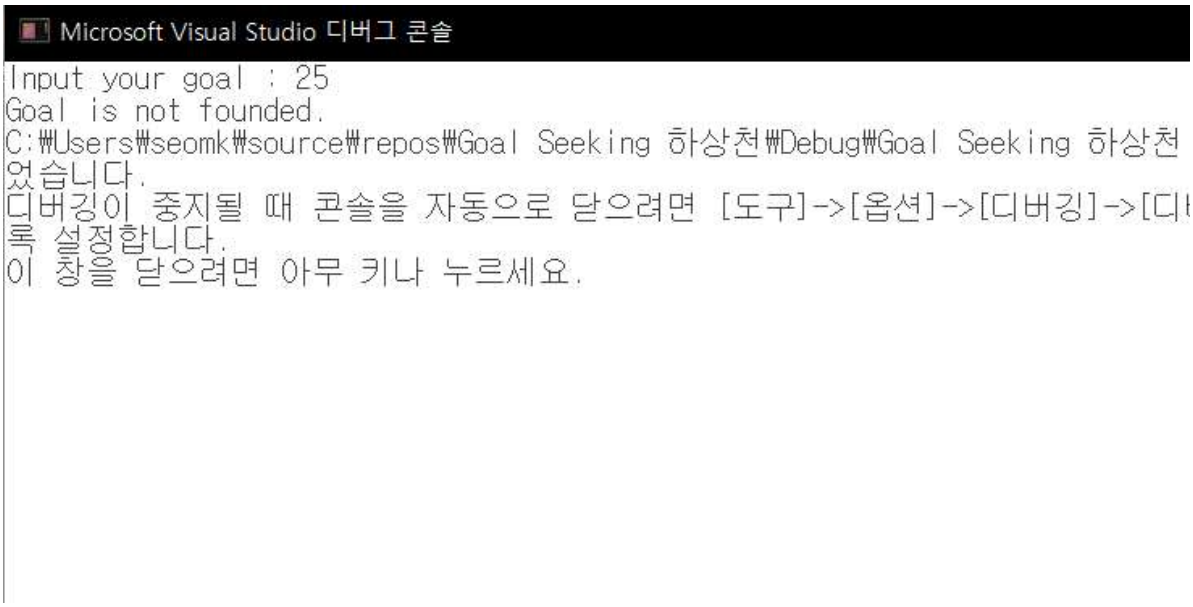
goal을 찾았다면, 반복문을 통해 goal path를 출력해준다. isEmpty 함수를 통해 stack이 비어있는지 확인하고, 비어있다면 break를 통해 반복문을 탈출하고 비어있지 않다면 pop 함수를 통해 스택에서 노드를 하나씩 꺼내서 data를 출력해준다.

5. 실행결과 및 설명 (실행 결과를 캡쳐하여 첨부한 후 설명)









goal을 사용자에게 입력받고, goal을 찾았다면 goal path를 출력해주고 goal을 못 찾았다면 찾지 못했다고 출력해주었다.

(그림을 문서에 포함, 글자처럼 취급 옵션, 잉크 절약과 잘 보이게 하기위해 그림 반전)

6. 고찰 (과제를 진행하면서 배운점 이나, 시행 착오 내용, 기타 느낀점)

이번 과제는 goal seeking 문제의 해결방법인 Pseudo code를 C언어나 C++을 이용해서 구현하는 것이었다. goal seeking 문제는 Depth-First Traversal과 관련이 있는 것 같았다. 그래서 백트래킹과 스택을 이용해서 문제를 풀어도 되고, 재귀함수를 통해서 문제를 풀어도 됐지만 Pseudo code처럼 구현하기 위해 백트래킹과 스택을 이용하였다. 그리고 스택을 만들 때 이중포인터를 이용해서 배열에 Node의 포인터형을 삽입하였다. 하지만 처음에는 실행화면이 제대로 나오지 않아서 한 줄씩 코드를 실행하여 결과값을 확인하였다. 이 과정을 통해 잘 못된 점을 찾아내고 코드를 재작성하여 실행화면이 올바르게 나왔을 때 많은 뿌듯함을 느끼는 것 같다. 저번 과제 때 느꼈던 것처럼 주석이나 변수의 이름을 가독성 좋게 작성하려고 노력하였다.

7. 전체 소스코드 (글자크기 9에 줄간격을 120%로 유지하고 한 줄이 너무 길지 않게 작성)

```
#include<stdio.h>
#include<stdlib.h>
#define true 1 //보기 쉽게 true, false를 1, 0 으로 정의
#define false 0
typedef int boolean;  //가독성을 높이기 위해 int 자료형을 boolean 자료형으로 쓰겠다.
typedef int element;
typedef struct tTreeNode {  //이진트리 노드의 구조체
        element data;
        struct tTreeNode *left, *right;
        boolean key;
} TreeNode;
typedef struct {  //스택의 구조체
```

```
        TreeNode **stack;  //TreeNode의 포인터형을 삽입할 배열
        int max_size;  // 최대사이즈
        int top;
} Stack;

TreeNode* CreateTree(TreeNode *left, element item, TreeNode *right)
{
        TreeNode *pNewNode = (TreeNode*)malloc(sizeof(TreeNode));
        if (pNewNode == NULL)
                return NULL;
        pNewNode->data = item;
        pNewNode->left = left;
        pNewNode->right = right;
        pNewNode->key = false; // false로 초기화 해준다.
        return pNewNode;
}
TreeNode* MakeTree() {  //트리를 만들어준다.
        TreeNode *eighteen = CreateTree(NULL, 18, NULL);
        TreeNode *seventeen = CreateTree(NULL, 17, eighteen);
        TreeNode *sixteen = CreateTree(NULL, 16, NULL);
        TreeNode *fifteen = CreateTree(NULL, 15, sixteen);
        TreeNode *fourteen = CreateTree(NULL, 14, fifteen);
        TreeNode *thirteen = CreateTree(seventeen, 13, fourteen);
        TreeNode *twelve = CreateTree(NULL, 12, thirteen);
        TreeNode *eleven = CreateTree(NULL, 11, NULL);
        TreeNode *ten = CreateTree(NULL, 10, eleven);
        TreeNode *nine = CreateTree(NULL, 9, ten);
        TreeNode *eight = CreateTree(NULL, 8, NULL);
        TreeNode *seven = CreateTree(NULL, 7, eight);
        TreeNode *six = CreateTree(NULL, 6, seven);
        TreeNode *five = CreateTree(nine, 5, six);
        TreeNode *four = CreateTree(NULL, 4, five);
        TreeNode *three = CreateTree(twelve, 3, four);
        TreeNode *two = CreateTree(NULL, 2, three);
        TreeNode *root = CreateTree(NULL, 1, two);

        return root;
}
boolean isEmpty(Stack *s) {
        return (s->top == -1);
}
boolean isFull(Stack *s) {
        return (s->top == s->max_size - 1);
}
```

```
Stack* CreateStack(int size) {      //Stack 동적 메모리 할당, size 만큼 stack 동적 메모리할당
        Stack *pStack = (Stack*)malloc(sizeof(Stack));
        if (pStack == NULL)
        {
                return NULL;
        }
        pStack->stack = (TreeNode**)malloc((size) * sizeof(TreeNode));
        if (pStack->stack == NULL) {
                free(pStack);
                return NULL;
        }
        pStack->max_size = size;
        pStack->top = -1;
        return pStack;
}
void push(Stack *pStack, TreeNode* item) {
        if (isFull(pStack)) {
                printf("Stack is full");
                return;
        }
        pStack->stack[++pStack->top] = item;  // TreeNode 포인터 형 스택에 삽입
}
TreeNode* pop(Stack *pStack) {
        if (isEmpty(pStack)) {
                printf("Stack is empty");
                exit(0);
        }
        return pStack->stack[pStack->top--]; // TreeNode 포인터형을 스택에서 꺼내온 후 리턴해준다.
}
void DestroyStack(Stack *pStack) {
        free(pStack->stack);
        free(pStack);
}
void seekGoal(TreeNode* map, element goal) {
        Stack* pStack = CreateStack(50); // 여유있게 stack의 사이즈를 50으로 해서 만들어준다.
        boolean goalFound = false; //while문 탈출을 위한 flag
        boolean finish = false; //while문 탈출을 위한 flag
        TreeNode * pMap = map;

        while (!goalFound) {
                if (pMap == NULL) {
                        while (true) {
                                if (isEmpty(pStack)) {
```

```
                                        printf("Goal is not founded.");
                                        finish = true; //while문 탈출을 위해 true로 바꾸어주고 break문을 통해 가까운 반복문 탈출
                                        break;
                                }
                                pMap = pop(pStack); // branch point까지 노드를 하나씩 꺼내준다.
                                if (pMap->key == true) {
                                        pMap->key = false; // goal path를 출력하기 위해 key를 false로 하여 다시 stack에 삽입
                                        push(pStack, pMap);
                                        pMap = pMap->left; // branch point에서 반대쪽 노드로 이동
                                        break;
                                }
                        }
                        if (finish == true) { // while문 탈출
                                break;
                        }
                }
                else {
                        if (pMap->data == goal) { //goal을 찾았다면
                                goalFound = true;  //while문 탈출을 위해 true로 변경
                                push(pStack, pMap); // goal path를 출력하기 위해 goal도 stack에 삽입
                                break; //가까운 반복문 탈출
                        }
                        else {
                                if (pMap->left != NULL && pMap->right != NULL) {
                                        pMap->key = true;  //  branch point이면 key를 true로 대입
                                        push(pStack, pMap);
                                }
                                else {
                                        push(pStack, pMap);
                                }
                        }
                        pMap = pMap->right; // 다음 노드로 연결해준다.
                }

        }

        if (goalFound) { // goal을 찾았다면
                printf("The path to your goal is : ");
                while (true) {
```

```
                        if (!isEmpty(pStack)) { // stack이 비어있지 않다면 pop을 통해 goal path 출력
                                printf("%d ", pop(pStack)->data);
                        }
                        else
                                break;  // stack이 비어있다면 반복문 탈출
                }
        }
        DestroyStack(pStack); //동적 메모리 반납
}
int main() {
        TreeNode *tTreeNode = MakeTree(); // 트리를 만들어준다.
        int goal;
        printf("Input your goal : ");
        scanf("%d", &goal); // goal을 입력받는다.
        seekGoal(tTreeNode, goal); // goal을 찾아준다.
}
```

(글자크기는 10으로 유지하고 줄간격도 160%를 유지할 것)