<자료구조 6차 과제 - Graph 보고서>

201720736 소프트웨어학과 이균

1. Data Structure

1. **static int** __parse_command(**char** *command, **int** *nr_tokens, **char** *): 도시들이 입력으로 들어올 때 도시의 개수만큼 입력이 들어온다. 이때 도시들은 공백으로 구분되어 들어온다. 이때 공백 문자를 기준으로 도시 문자열의 입력을 파싱하여 저장하기 위해서__parse_command함수를 선언하였다. 추후 도시간의 연결을 확인할 때에도 이 함수를 사용하여 -문자를 기준으로 ₩0을 삽입하여 문자열을 파싱하여 연결되는 도시들을 분리해내었다.

   char command[MAX_COMMAND] = {'\0'}; : 입력되는 도시 문자열을 저장하기 위해서 사용하였다.
   char *tokens[MAX_NR_TOKENS] = { **NULL** }; 파싱한 토큰을 저장하기 위해 사용하였다.
   int nr_tokens = 0; : 파싱한 토큰의 개수를 저장한다.

2. **typedef struct** node *nodePointer;
   ```
   typedef struct node {
      int vertex;
      char cityName[10];
      nodePointer link;
   }Node;
   Node adjListsArr[MAX_VERTICES];
   nodePointer avail = NULL;
   ```

   adjacency list를 만들기 위해서 선언하였다. adjListArr 는 들어오는 순서에 따른 vertex와 이름인 cityName, link field를 가진다. 추후 사용하지 않는 노드를 재사용하기 위해 avail을 사용하여 메모리를 재활용하였다.
   따라서 입력에 맞게 파싱된 도시들을 adjListsArr[i].vertex와 adjListsArr[i].cityName에 추가해주고, 입력에 맞게 BST를 만든 후에 도시 연결 입력이 들어오면 연결된 node를 adjListsArr[i]에 추가해주면서 adjacency list를 만들었다.

3. **typedef struct** treeNode* treePointer;

   ```
   typedef struct treeNode {
      char key;
      int index;
      char cityName[10];
      treePointer leftChild;
      treePointer rightChild;
   } treeNode;
   treePointer root = NULL;
   treePointer insert(treePointer node, char* str, int
   index){}
   treePointer newNode (char* str, int i){}
   treePointer search(treePointer root, char* str){}
   ```

   tree 자료들을 구조체로 선언해주었다. **char** key;는 BST에서 크기를 비교하기 위해 도시의 첫 글자 알파벳을 저장하기위한 key값이고, **int** index;는 입력순서에 맞는 index값으로 사용하였다. 또한 도시 이름을 위한 **char** cityName[10];, child node를 위한 treePointer leftChild; treePointer rightChild;를 선언하였다.

트리의 root를 나타내도록 treePointer root = NULL;을 사용하였고, 입력으로 들어오는 도시들을 insert와 newNode함수를 통해 BST로 만들었다.
처음 도시이 개수만큼 도시의 입력이 들어오면 insert()함수를 통해서 BST를 만들고(이때 알파벳 순서로 크기비교를 한다), 이후 도시간의 연결이 입력으로 들어오면 search()함수를 통해 BST를 search하면서 BST에 입력된 도시가 없으면 잘못된 입력이고, BST에서 찾은 노드들이 이미 adjacency list에 만들어져있는지를 check_dup에서 확인한다.

4. **int** check_dup(treePointer ptr0, treePointer ptr1){}:
중복을 확인하기 위한 함수이다.
우선 도시들의 연결이 입력으로 들어오면 그 연결에 맞는 adjacency list를 만들게 된다. 이후 똑같은 연결이 들어온다면 adjacency list에서 이미 연결이 추가되어있을 것이므로 list를 스캔하며 입력과 똑같은 연결이 있는지 체크하면 된다. 따라서 만약 도시의 연결이 mn-kl인데 똑같은 입력인 mn-kl이 들어온다면 이미 adjacency list에 mn과 kl이 연결된 상황일 것이고, 두번째 입력에 대해서 리스트를 스캔하며 adjacency list의 vertex와 입력된 도시의 index를 비교해서 중복성을 체크한다.

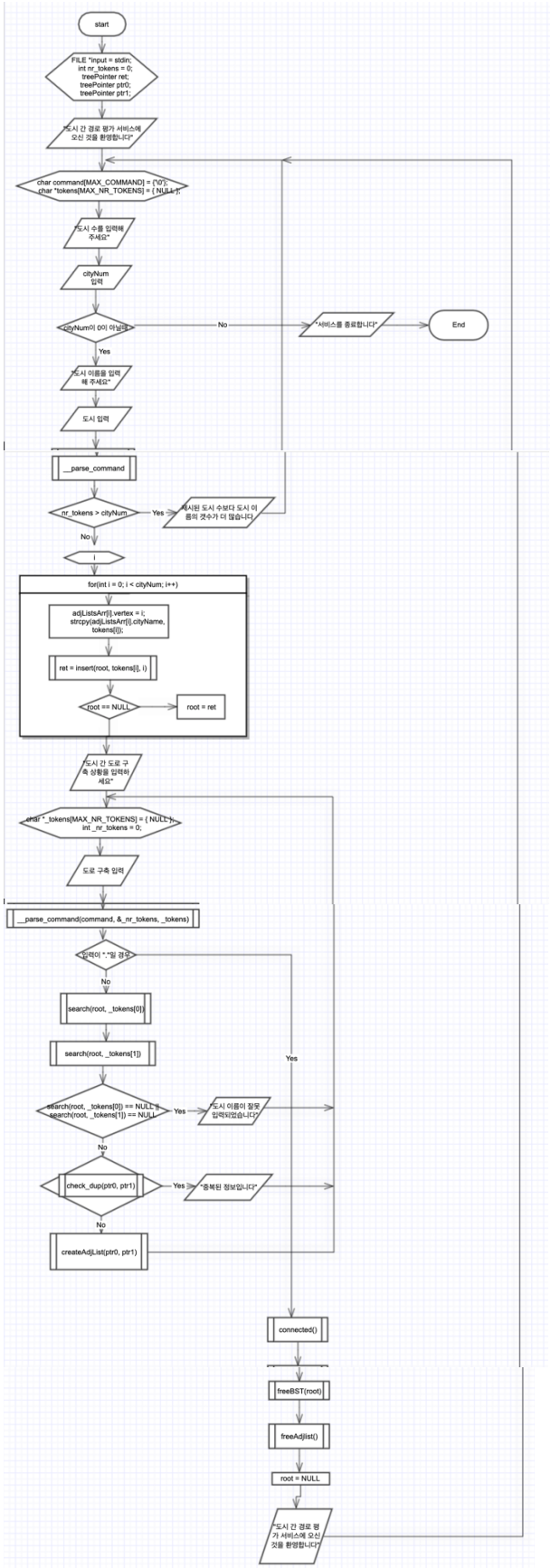
```
while(temp && temp1){
 if(temp->vertex == ptr0->index && temp1->vertex == ptr1->index){
    return TRUE;
 }
 temp1 = temp1->link;
}
```

예를들어 도시의 입력이 mn-kl이라면 temp->vertex는 adjacency list에서 cityname이 mn인 노드의 vertex일 것이고, temp1의 링크를 계속 증가하면서 리스트를 스캔하며 BST에서 찾은 노드의 index와 같은지 비교하였다.

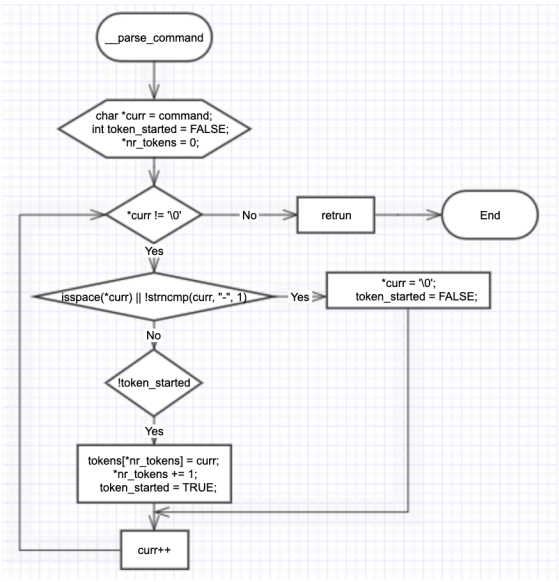
5. createAdjList(ptr0, ptr1);
```
void connected(){}
Node visitedArr[MAX_VERTICES];
nodePointer lastConnected = NULL;
Node visitedArr[MAX_VERTICES];
int cnt = 0;
void dfs(int v){}
short int visited[MAX_VERTICES];
```

도시의 연결이 입력으로 들어온다면 그 연결에 맞는 adjacency list를 만들어야 한다. 따라서 createAdjList를 이용하였다.
우선 사용하지 않는 노드의 저장을 위한 avail이 남아있는지의 여부에 따라 메모리를 reuse하게 하였고, list를 추가할 때 맨 앞부분에 추가하는 방식으로 코드를 구성하였다. 따라서 O(1)의 시간이 걸리도록 하였다.
도시들이 몇 개의 component로 분리되는지 확인하기 위해서 도시의 연결성이 끊기는지 아닌지 확인해야 한다. 따라서 adjacency list에서 dfs를 이용하면 연결이 끊기는지 안닌지 확인할 수 있다. 이때 visitedArr[i]에 연결된 노드끼리 저장하게 하였다. 만약 연결이 완전하다면 visitedArr[0]에 연결된 노드들이 쭉 이어지게 될 것이고, 만약 두개로 연결이 끊긴다면 visitedArr[1]에 두번째 그래프 component들의 연결이 나타나게 되는 방식으로 visitedArr를 만들었다. 이때 만약 그래프의 연결이 최대로 끊어지는 경우는 연결이 하나도 없는 경우이므로 최대 입력인 20개의 노드가 모두 연결되지 않는 경우이다. 따라서 Node visitedArr[MAX_VERTICES];이러한 방식으로 선언해주었다. 따라서 연결이 끊어진 경우에는 visitedArr가 가리키는 첫번째 링크들끼리 연결을 추천한다고 출력하면 된다. 또한 연결이 끊어졌을 때 visitedArr[i]의 링크를 증가하면서 쭉 출력을 해주면 나눠서 출력을 할 수 있다. lastConnected는 visitedArr에 링크를 연결하기 위해서 선언하였다. Adjacency list를 만들때 맨 앞에 노드를 이어붙였는데 노드를 순서대로 출력해주기 위해 리스트에 순서대로 저장을 하기 위해서 리스트의 마지막에 저장하고 이를 가리키기 위해 선언하였다.

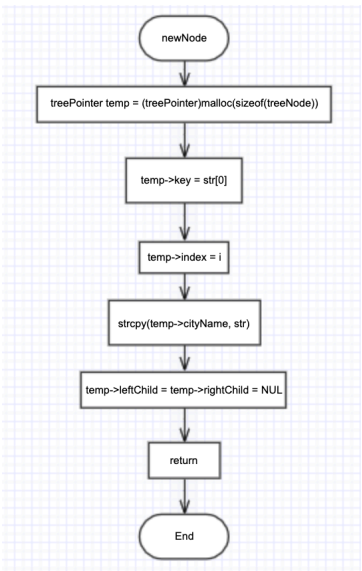
2. Flow Chart



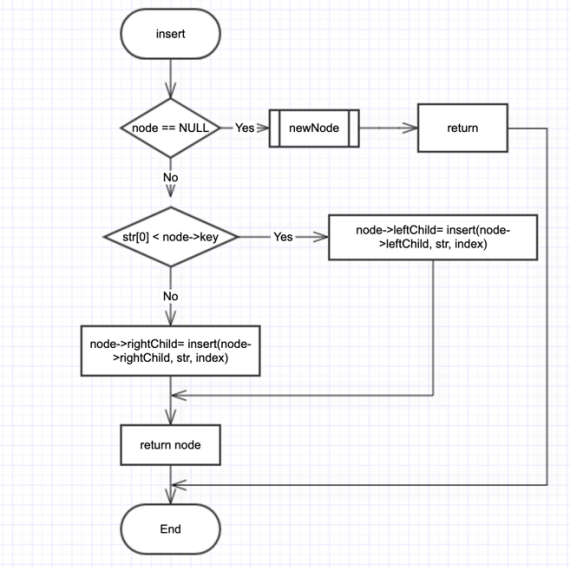
main 함수



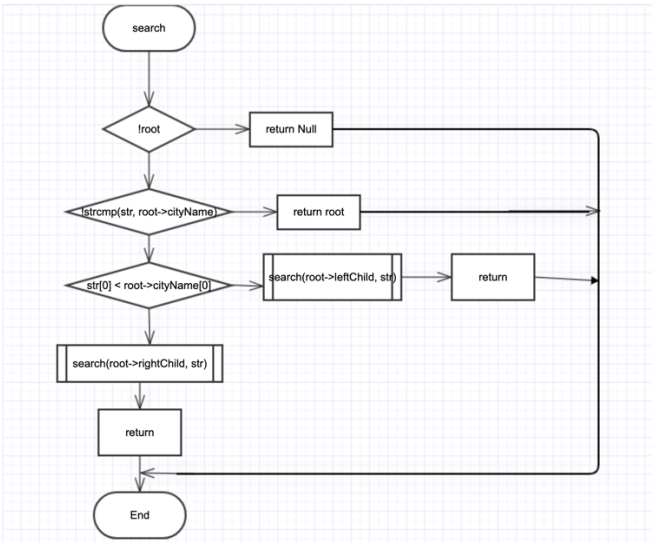
__process_command 함수



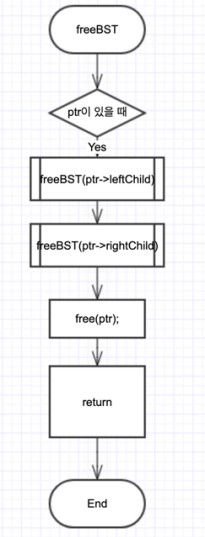
newNode 함수



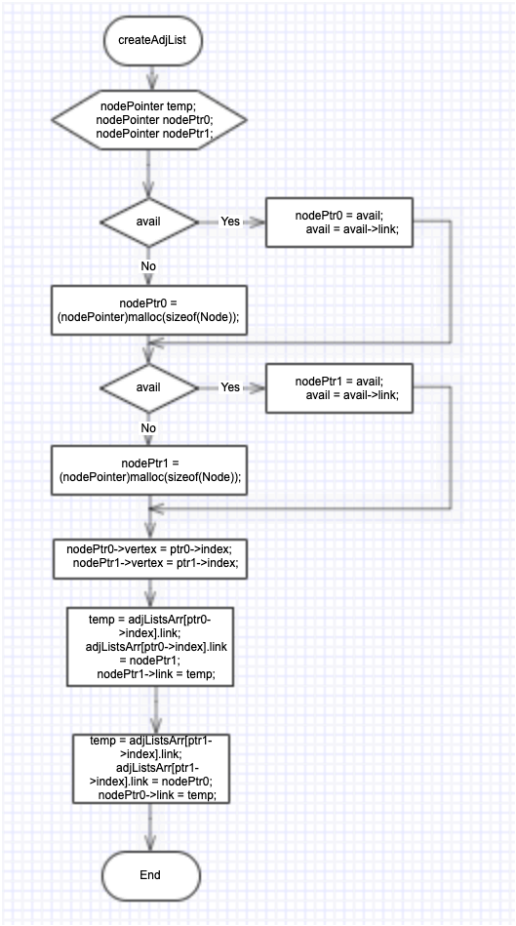
insert 함수



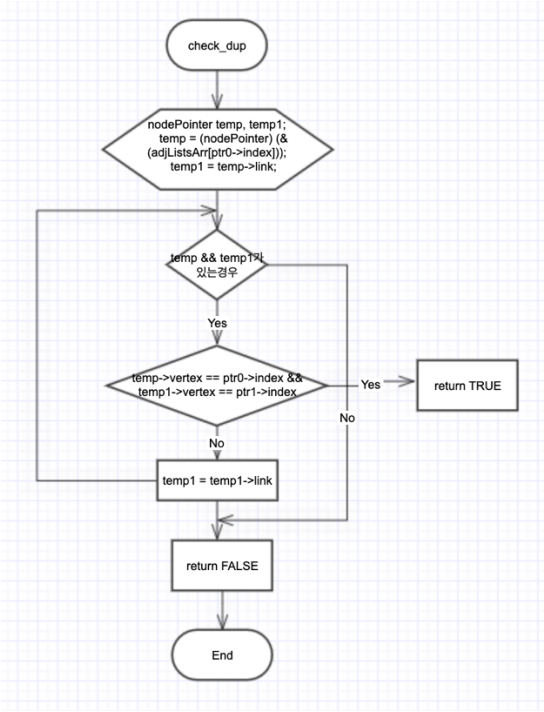
search 함수



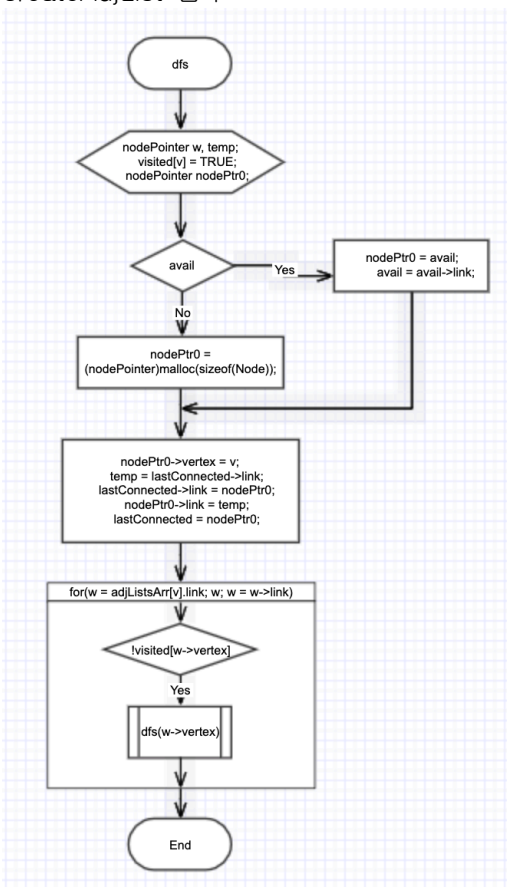
freeBST함수



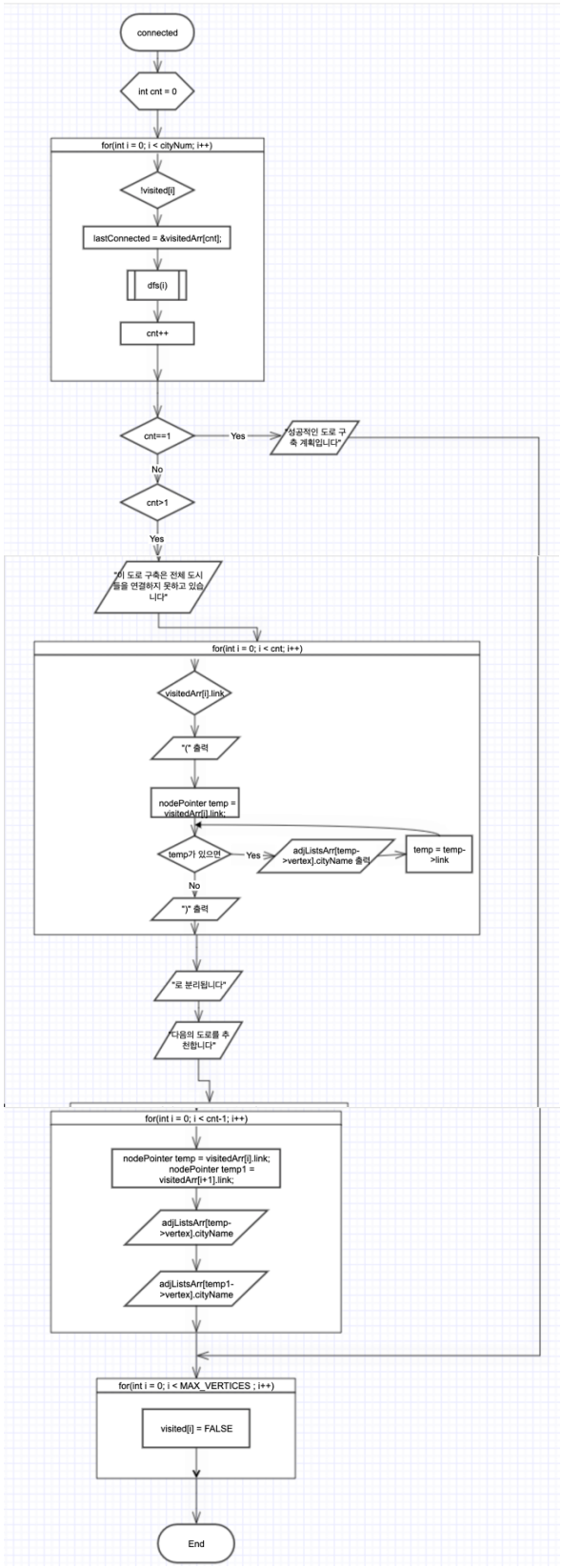
createAdjList 함수



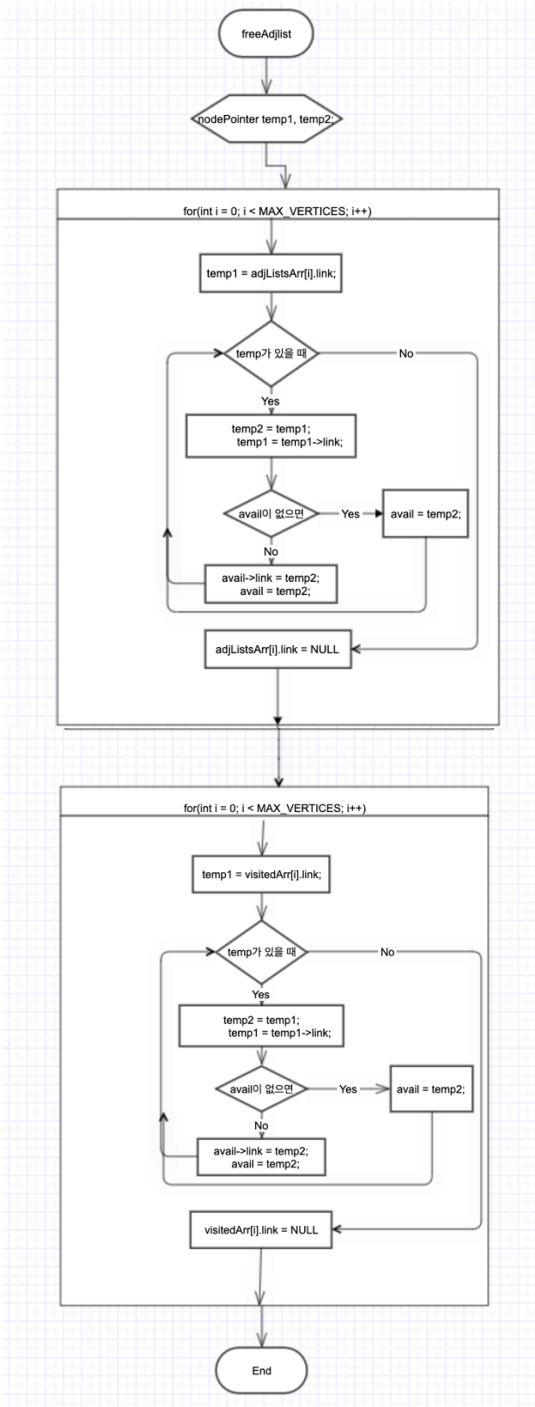
check_dup 함수



dfs 함수



connected 함수



freeAdjlist 함수