알고리즘 과제 6. 그래프 순회

제출일 : 2017/05/25

경영학과 201101328　박성환

목차 :

1. 문제 정의

이번 과제는 그래프 데이터 타입을 구현하고 두 순회 알고리즘(DFS, BFS)의 결과를 비교하는 것이다. 따라서 문제는 크게 세 가지로 표현할 수 있다.

1. 그래프는 어떻게 구현할 것인가?
2. DFS(Depth First Search) 알고리즘은 어떻게 구현할 것인가?
3. BFS(Breadth First Search) 알고리즘은 어떻게 구현할 것인가?

이 세 문제에 대한 해결방안을 알아보도록 한다. 또한 과제에서 요구한 대로 이번 그래프는 무방향, 가중 그래프이고 그래프 구현은 배열 방법을 사용하도록 한다.

2. 해결방안

2.1. 그래프 구현

그래프는 정점(vertex)와 엣지(edge)로 이루어져 있다. 따라서 그래프 자료구조는 정점과 엣지를 가지고 있어야 하고 정점은 배열로, 엣지는 2D 매트릭스로 표현한다.

```
typedef struct _Graph {
    int nVertices;
    int nEdges;
    int vertex[MAX_VERTICES];
    int edge[MAX_VERTICES][MAX_VERTICES];
} Graph
```

이 특성들을 init_graph로 초기화한다. 간단한 식이라 생략한다.

내 구현 모델에서는 입력 정점 이름을 배열 번호로 변환하는 get_index라는 헬퍼함수가 존재한다. 사용자는 정점을 인식하기 쉽게 정점에 이름을 붙이기 마련인데 이 과제에서는 알파벳 대문자 글자들로 이름을 지었다. 그런데 배열로 짜면 정점에 접근하기 위해 0 이상의 정수가 필요하고 과제에서는 'A'를 0으로, 이후 알파벳은 1씩 증가하도록 설정했다. 따라서 알파벳 대문자를 0 이상의 정수로 변환하는 함수가 필요했다.

```
int get_index(char v) {
    return v - 'A';
}
```

c에서는 신기하게도 영어 알파벳은 그 자체로 ascii 숫자로도 쓸 수 있다. 이 식을 통해 'A'는 0에서부터 'Z'는 25까지 변환할 수 있다. 함수 자체는 매우 간단하지만 다른 모든 함수들에 유용하게 쓰임을 생각해볼 때 꽤 괜찮은 함수 구현이라고 평가한다.

이 과제에서는 정점과 엣지를 추가하는 함수만 구현한다. 그 코드들은 다음과 같다.

```
void insertVertex(Graph *g, char v) {
    int index = get_index(v);
    if(g->vertex[index1] == 1) {
        printf("Vertex already exists.");
        return;
    }
    g->vertex[index] = 1;
    g->nVertices += 1;
}

void insertEdge(Graph *g, char v1, char v2, int weight) {
    int index1 = get_index(v1);
```

```
    int index2 = get_index(v2);
    if(g->vertex[index1] == 0 || g->vertex[index2] == 0) {
        printf("At least one vertex doesn't exist.\n");
        return;
    }
    if(g->edge[index1][index2] != 0) {
        printf("Edge already exsits.\n");
        return;
    }
    g->edge[index1][index2] = weight;
    g->edge[index2][index1] = weight;
    g->nEdges += 1;
}
```

기본 코드들은 과제 설명서에 있는 그대로이다. 여기서 주목할 부분은 추가할 정점과 엣지를 추가할 때 이미 기존 정점 또는 엣지가 존재할 때의 대응 코드이다. 구현한 그래프 타입은 엣지와 정점의 개수를 추적하는 변수가 설정되어 있다. 그래서 추가하는 코드에서 그 변수들의 값이 하나씩 증가하는 것은 당연하다. 그런데 두 함수의 'if' 부분이 없다면 기존에 정점 또는 엣지가 존재하는데도 개수 변수들은 증가하게 된다. 이런 부분은 에러가 발생하지 않기 때문에 특히 조심해야 한다. 더욱이 엣지를 추가할 때는 엣지가 부속하는 두 정점이 하나라도 존재하지 않거나, 엣지가 이미 존재할 때를 모두 고려해줘야 한다.

마지막으로 과제가 요구한 형식에 맞게 그래프를 출력하는 함수이다.

```
void printGraph(Graph *g) {
    int i, j;
    int count;

    // Vertex section
    printf("\n  V(G) = { ");
    count = 1;
    for(i = 0; i < MAX_VERTICES; i++) {
        if(g->vertex[i] != 0) {
            if(count < g->nVertices) {
                printf("%c, ", i+'A');
            }
            else {
                printf("%c }", i+'A');
            }
            count++;
        }
    }

    // Edge section
    printf("\n  E(G) = { ");
```

```
    count = 1;
    for(i = 0; i < MAX_VERTICES; i++) {
        for(j = 0; j < MAX_VERTICES; j++) {
            if(g->edge[i][j] != 0 && i <= j) {
                if(count < g->nEdges) {
                    printf("(%c,%c), ", i+'A', j+'A');
                }
                else {
                    printf("(%c,%c) }", i+'A', j+'A');
                }
                count++;
            }
        }
    }
    printf("\n\n");
}
```

생각해볼 부분은 다음과 같다. 출력하는 형태의 중간(ex ‘a’, ‘b’),)과 마지막(ex ‘a’, ‘b’) })
부분의 차이를 만들어 주는 것인데 count라는 변수를 두어서 정점과 엣지의 개수의 마지막에
다다르면 출력을 다르게 해서 해결했다. 앞서 살펴본 것처럼 그래프의 정점과 엣지의 개수를
나타내는 변수를 잘 관리해야 함을 알 수 있다.

2.2. DFS

DFS는 깊이 우선 탐색으로 LIFO 방식의 스택을 사용하는 방법과 재귀함수를 사용하는
방법이 있는데 c로 스택을 구현해본 적이 없어서 배열 스택 방법을 사용했다. 스택 코드는
대단하게 어렵지는 않아 보고서에는 적지 않는다.

탐색 알고리즘 구현은 다음과 같다.

```
void depthFirstSearch(Graph *g, char v) {
    int visited[MAX_VERTICES];
    int index = get_index(v);
    int value;
    int i;
    Stack s;
    init_stack(&s);
    push(&s, index);

    for(i = 0; i < MAX_VERTICES; i++) {
        visited[i] = 0;
    }

    while(!is_stack_empty(&s)) {
        value = pop(&s);
        if(visited[value] == 0) {
```

```
            visited[value] = 1;
            printf("%c ", value+'A');
            for(i = MAX_VERTICES - 1; i >= 0; i--) {정점
                if(g->edge[value][i] != 0) push(&s, i);
            }
        }
    }
    printf("\n");
}
```

코드는 수업 시간에 사용한 코드에서 크게 다르지 않다. 고민한 부분은 PPT 유사코드의 '인접한 정점들' 이라는 부분을 어떻게 만들어야 할지 고민이었다. 결과적으로는 for문을 써서 다 훑어보는 방법을 선택했는데 효율성에 의문이 든다. 정점이 많아질수록 효율이 떨어질 것 같은데 더 고민해봐야 할 것 같다.

2.2. BFS

BFS는 너비 우선 탐색으로 FIFO인 queue를 사용해 구현할 수 있다. 이 구현에는 circular buffer 또는 ring buffer를 사용했다. 코드는 역시 붙이지 않는다. 바로 탐색 알고리즘 코드를 살펴보도록 한다.

```
void breadthFirstSearch(Graph *g, char v) {
    Queue q;
    init_queue(&q);
    int value = get_index(v);
    int visited[MAX_VERTICES];
    int i;

    for(i = 0; i < MAX_VERTICES; i++) {
        visited[i] = 0;
    }
    enqueue(&q, value);

    while(!is_queue_empty(&q)) {
        value = dequeue(&q);
        if(visited[value] == 0) {
            visited[value] = 1;
            printf("%c ", value+'A');
            for(i = 0; i < MAX_VERTICES; i++) {
                if(g->edge[value][i] != 0) enqueue(&q, i);
            }
        }
    }
    printf("\n\n");
```

```
}
```

위의 DFS와의 차이는 거의 없고 stack이 queue으로 대체되었다. 이번 과제에서는 그래프의 정점 최대 크기를 26으로 했는데 지금 살펴보니 stack과 queue는 최대 개수를 100으로 했다. 같게 했어도 에러가 발생했을 것 같지는 않다.

3. 테스트

과제에서 요구한 대로 결과가 안정적으로 출력되었다.

```
과제 6-1. 배열로 그래프 구현 201101328 박성환

V(G) = { A, B, C, D, E, F }
E(G) = { (A,B), (A,D), (A,E), (B,C), (B,D), (C,D), (C,E), (D,F), (E,F) }

Depth First Search   : A B C D F E
Breadth First Search : A B D E C F
```

4. 결론

과제에서 요구한 그래프의 기본 구현, DFS 탐색, BFS 탐색이 모두 구현되었다. 그래프를 배열을 사용해서 구현했는데 배열 방법이 훨씬 쉬운 것으로 생각된다. 방법이 쉬운데 모든 케이스를 커버 못한다는 것은 그만큼 단점이 있다는 것이고 그 중 하나가 처음 설정한 정점의 최대 크기를 쉽게 바꾸지 못한다는 것이다. 같은 코드를 파이썬으로 구현했을 때 이 점을 생각해서 최대 크기를 초기화 이후에 늘릴 수 있는 함수를 구현했다. 배열로 구현했음에도 파이썬에서는 그래프의 크기를 증가시킬 수 있었는데 c에서는 그것이 안 되는 것이 아쉬웠다. 아마 자유도가 높은 c에서는 어렵지만 메인 메모리를 잘 건드리면 배열의 크기를 늘릴 수 있을 것 같기도 한데 지금 내가 다룰 수 있는 부분은 아닌 것 같다. 이 점이 해결되면 많은 알고리즘을 배열로 구현하지 말아야 할 이유가 하나 없어지는 것일테다.

그래프의 정점과 엣지를 추가하는 부분에서 입력 받은 정점이나 엣지가 존재할 때의 대응 코드의 중요성을 역설했는데 이 부분은 많이 유감이다. 왜냐하면 개인적으로 자신 있는 파이썬으로 코드를 짤 때는 '사용자는 믿을 사람이 못 된다'는 생각 아래 사용자의 모든 잘못된 입력에 대응하는 코드를 짜는 것을 좋아한다. 그런데 이 과제에서는 엣지나 정점의 존재는 고려했으면서도 정작 정점에 'a'나 '정점AAAA', '저도정점이에요^^'와 같은 입력에 대응하는 코드를 넣지 못했다. 보고서를 쓰면서 개인적으로 굉장히 찜찜했는데 c에 대한 지식이 아직 부족하다는 것을 고려해주셨으면 한다.

마지막으로 그래프는 추가 공부할 이유가 다분한 분야라고 생각된다. 인맥, 관계도, 생물 계통수 등 사회, 과학 다양한 분야를 수량화, 데이터화할 때 필수적이라고 생각된다. 이런 영역에 관심이 있고 그래프는 반드시 수업 ppt의 모든 부분을 구현하고 추가적으로 공부하도록 하겠다.