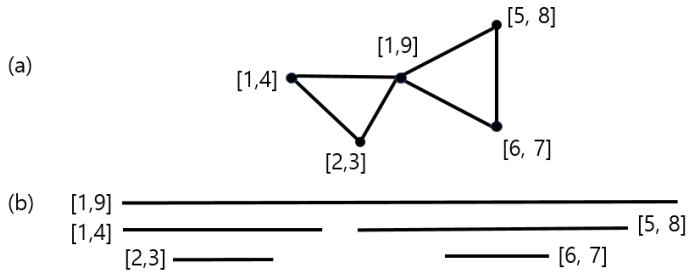
숙제 3 : Interval graph

<Due : 2020-06-06 (토) 23:59>

Interval graph (구간 그래프) 는 graph 의 한 종류로서 각각의 vertex 가 closed interval (닫힌 구간) [a, b] 로 주어지며, **vertex a = [a1, a2] 와 b= [b1, b2] 가 있을 때 interval A와 B에 공통된 원소가 있다면 (즉 A∩B≠∅ 이면) vertex a와 b는 adjacent 하다**. 예를 들어 아래 그림은 (b) 의 interval 들을 vertex 로 가지는 interval graph (a) 를 보여주고 있다.



이번 과제에서는 interval graph 를 adjacency list 를 통해 구현하고, 몇가지 연산들을 구현해 보는 것을 목표로 한다.

1) Graph 의 각각의 vertex (interval) 는 다음과 같은 구조체로 정의되며, 해당 구조체의 leftmost 와 rightmost 멤버에 interval의 양 끝점이 저장된다. 예를 들어 vertex [a, b] 를 저장할 때 interval 의 leftmost 멤버에 a, rightmost 멤버에 b가 들어가게 된다.

```
typedef struct interval {
    int leftmost;
    int rightmost;
}interval;
```

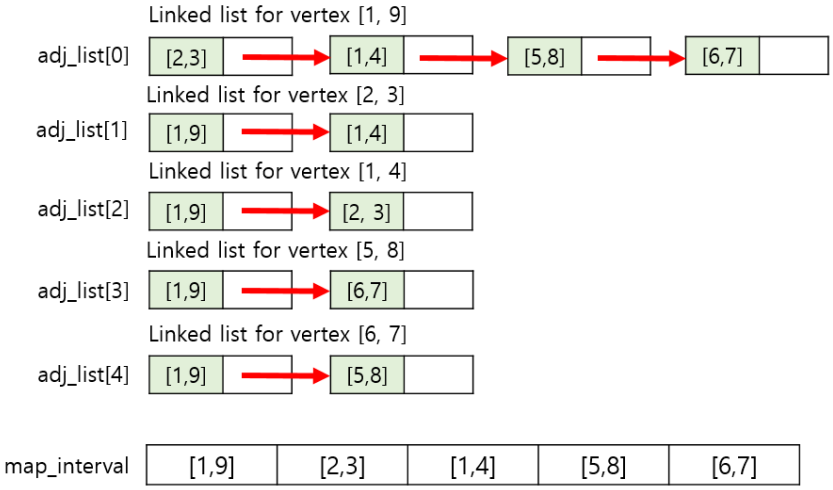
여기서 leftmost 와 rightmost 는 언제나 양수 (1 이상의 integer) 이며, **한 vertex (interval) 의 leftmost 와 rightmost 가 동일할 수 있다는 점에 주의하자 (예: [1,1])**.

2) Adjacency list 는 다음과 같이 정의한다. 강의 자료와 마찬가지로 n 개의 vertex가 있다 하면 Adjacency list 는 n개의 linked list 로 정의되며, 각각의 linked list 의 node와 그에 따른 그래프의 구조체는 다음과 같이 정의된다 (강의자료와는 다르게 node의 item 이 interval 형식이 되는 것에 주의하자).

```
typedef struct node {
    interval item;
    struct node* next;
}node;

typedef struct Graph {
    int num_vertices;     //number of vertices
    node *adj_list[MAX_VERTICES];
   interval map_interval[MAX_VERTICES];
}Graph;
```

Graph 가 가질 수 있는 최대 vertex 개수를 저장하는 MAX_VERTICES 는 헤더 파일 graph.h 에 정의 되어 있다. (채점 시 변경 예정). 또한 Graph 구조체에서 강의 슬라이드와는 다르게 interval 구조체의 1차원 array map_interval 이 정의되어 있으며, 만약 각각의 adj_list[i] 가 interval $[a_i, b_i]$ 에 대한 adjacency list 를 저장하고 있다면 map_interval[i].leftmost = a_i, map_interval[i].rightmost =b_i 가 된다. **이때 map_interval [0].... map_interval [num_vertices-1] 에는 모두 해당하는 interval 들이 저장되어 있어야 한다 (즉 어떤 경우에도 array 중간에 빈 공간이 존재해선 안된다)**. 예를 들어 위 graph 의 adjacency list 와 map_interval array 는 다음과 같이 저장된다.



또한 graph.c 에는 adjacency list 를 이루는 각각의 linked list 에서 node 를 추가하고 지울 수 있는 add_node 및 delete_node 함수가 구현되어 있으며, 해당 함수는 자유롭게 사용할 수 있다.

3) 본 과제에서는 simple, undirected, unweighted graph 만 고려한다.

4) 2번 과제와 마찬가지로 error 메시지를 출력하는 print_error() 함수가 미리 정의되어 있다.

5) Stack 과 Queue 는 array-based implementation 으로 stack.h 와 queue.h 에 정의되어 있으며 해당 헤더 파일들에 정의된 연산들은 (연산의 의미는 강의 자료 참조) 자유롭게 사용할 수 있다. create() 와 isEmpty() 연산의 경우 강의 슬라이드와 이름이 약간 다르므로 해당 파일에서 확인할 것.

<구현해야 하는 함수들>

주의 : 1 번을 제외하고는 G의 인자로 NULL 이 들어가는 경우는 고려하지 않는다.

1. void construct (Graph *G, char * file) : file 에 있는 정보를 읽어 와서 file에 있는 interval들을 vertex로 가지는 interval graph 의 adjacency list G 를 생성하는 함수. 여기서 입력으로 받는 file은 다음과 같이 주어진다.

i) 맨 윗줄에 총 vertex의 개수, 그리고 그 아래로 한 줄 마다 각각의 vertex 들이 저장되어 있다.
ii) 각각의 vertex (Interval) 들의 **left 나 right mostpoint 는 i) 공백 문자 , ii) 콤마 (,), iii) new line 문자 셋 중 하나로 구분되며, 특히 leftmost point와 rightmost point 사이는 콤마 (,) 로 구분된다** (두 point 사이의 공백문자 크기는 자유롭게 결정할 수 있다).

iii) 파일이 주어진 형식에 맞지 않으면 (vertex 수만큼 interval이 주어지지 않거나, vertex 수가 MAX_VERTICES 를 넘을 때, 혹은 형식에 맞지 않은 interval이 있을 때) print_error 를 호출한 뒤 G 는 NULL 로 설정한다.

예를 들어 1페이지에 있는 graph를 저장하는 file은 다음과 같다.

(Hint : string.h 에 정의된 함수들을 자유롭게 쓸 수 있으므로 strtok 함수를 활용하자)

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

2. void print_graph (Graph *G) : 현재 저장된 interval graph G에 대한 map_interval array 와 G 에 해당하는 adjacency matrix 를 출력해주는 함수 (G 는 adjacency list 로 저장되어 있다). 예를 들어 1페이지의 그래프에 대해 print_graph 함수를 호출하면 다음과 같이 출력한다.

List 0 = [1 , 9]

List 1 = [2 , 3]

List 2 = [1 , 4]

List 3 = [5 , 8]

List 4 = [6 , 7]

0 1 1 1 1

1 0 1 0 0

1 1 0 0 0

1 0 0 0 1

1 0 0 1 0

3. int adjacent (Graph* g, interval v, interval u) : Vertex v와 u 가 G에서 adjacent 한지 확인하는 함수 (adjacent 하면 1, 아니면 0을 return). v 또는 u 가 G 상에 존재하지 않으면 print_error 함수를 호출하고 0을 return 한다.

4. int degree (Graph* g, interval v) : Interval graph G에서 vertex v 의 degree를 return 하는 함수. Vertex v 가 G 상에 존재하지 않으면 print_error 함수를 호출하고 0을 return 한다.

5. void add_vertex (Graph* G, interval v) : Vertex v를 G에 추가하는 함수. **v 를 추가한 뒤에도 G는 interval graph 의 성질을 만족해야 한다 (즉 새로 추가한 vertex에 맞게 edge들을 추가해야 함).** v 가 이미 G에 있으면 print_error 함수를 호출하고 아무 변경도 하지 않는다.

6. void delete_vertex (Graph* G, interval v) : Vertex v 를 G 에서 제거 하는 함수. add_vertex 와 마찬가지로 **v 를 제거한 뒤에도 G는 interval graph 의 성질을 만족해야 한다 (즉 제거한 vertex에 맞게 edge들을 제거해야 함).** v 가 G 에 없으면 print_error 함수를 호출하고 아무 변경도 하지 않는다.

5번과 6번의 경우 Adjacency list 와 vertex 의 관계를 보여주는 map_interval array를 함께 바꿔주어야 하는 점에 주의하자.

7. int connectivity (Graph* G) : Interval graph G가 connected 한지 check 하는 함수 (connected 하면 1을 아니면 0을 return 한다). Connected 의 정의는 Chap 7 강의 슬라이드를 참조할 것 (hint : graph traversal 을 이용하자).

8. int two_connectivity(Graph* G) : Interval graph G가 2-connected 한지 check 하는 함수 (2-connected 하면 1을 , 아니면 0을 return 한다). 2-connected 의 정의는 **G에서 임의의 vertex (즉 아무 vertex) 하나를 제거해도 여전히 G가 connected 라면 2-connected 이다**. (예를 들어 1 페이지에 있는 graph는 vertex [1,9] 를 제거하면 connected 하지 않으므로 2-connected 가 아니다) 만약 G가 2-connected 가 아니라면 G에서 어떤 vertex를 제거했을 때 G가 connected 가 안되는지 출력한다 (해당 vertex 는 하나만 출력한다).

예) CUT VERTEX = [1, 9]

9. int two_edge_connectivity (Graph* G) : Interval graph G가 2-edge-connected 한지 check 하는 함수 (2-edge-connected 하면 1을, 아니면 0을 return 한다). 2-edge-connected 의 정의는 **G에서 임의의 edge (즉 아무 edge) 하나를 제거해도 여전히 G가 connected 라면 2-edge-connected 이다** (예를 들어 1 페이지에 있는 graph는 2-connected 는 아니지만 2-edge-connected 이다).

만약 G가 2-edge-connected 가 아니라면 함수 안에서 G에서 어떤 edge 를 제거했을 때 G가 connected 가 안되는지 출력한다. (아래 출력 예시는 edge ([1,2], [2, 3]) 을 제거했을 때 G가 disconnected 되는 경우의 단순 예시이며, 1페이지의 graph 의 경우 2-edge-connected 이기 때문에 아무것도 출력하지 않는다. 또한 해당 edge는 하나만 출력한다).

예) CUT EDGE = ([1,2], [2,3])

10. int shortest_path (Graph *G, interval v, interval u) : Vertex v에서 u로 가는 shortest path 의 경로를 출력한 후 해당 path의 길이를 return 한다. 예를 들어 vertex [0,1] 에서 [2,3] 까지의 shortest path 의 길이가 2 이고 해당하는 경로가 [0,1] -> [1,2]->[2,3] 라면 다음과 같이 출력한다 (본 예제와 main 에 있는 예제를 정확히 확인하여 공백 문자에서 문제가 발생하지 않도록 주의하자). Vertex v나 u가 G에 없거나 vertex v에서 u 사이에 path가 존재하지 않으면 print_error() 함수를 호출하고 종료한다. 예를 들어 1페이지의 graph G 에서 v = [1, 4], u= [6, 7] 일 때, shortest_path(G, v, u) 을 호출하면 다음과 같이 출력된다 (해당 path가 여러 개 존재할 경우 그 중 하나만 출력한다).

Shortest path from [1, 4] to [6, 7] : [1, 4] [1, 9] [6, 7]

8, 9, 10 번의 경우 반드시 main 의 주석에 주어진 출력 결과와 비교하여 출력 양식 (spacing 등) 을 정확히 지킬 것.

3. 주의 **(이 중 하나라도 어기면 0점)**

- 주어진 파일은 다음과 같은 형식으로 되어 있음

1. graph.h 에서 정의한 함수들을 모아 둔 헤더파일.

2. **graph.c : graph.h 에서 정의한 함수들의 세부 구현 (숙제에서 제출해야 할 파일).**

3. stack.h stack.c : Array-based Stack 구현 파일

4. queue.h, queue.c : Array-based queue 구현 파일

5. main.c : 테스트용 main file (결과값이 주석에서 언급한 답과 일치하는지 확인할 것.

6. 리눅스, 맥 환경에서 작업하는 학생들을 위한 Makefile.

- 숙제 제출은 due 전까지 e-campus의 과제탭의 **과제물 제출 – 파일 첨부를 이용**하여, 오직 **graph.c 파일만을 학번_graph.c 파일로 업로드** 할 것. (예 : 2019000000_graph.c). **그 외 어떠한 파일도 받지 않음**.

- 숙제의 delay는 받지 않음.

- 문제의 skeleton code 에서 주어진 헤더 파일 외의 **어떠한 헤더 파일도 include 하지 말 것**.

- 주어진 코드에 정의된 함수 및 전역번수만을 구현 및 사용 하고, **자기가 따로 함수나 전역변수를 정의하여 graph.c 에 추가하지 말 것 (**채점 시 graph.c 제외 원래 주어진 파일들로 채점한다는 것을 잊지 말것). **또한 모든 C 함수는 반드시 표준 C함수 (standard C) 함수 및 constant 들만 사용할 것** (Visual studio 에서만 정의된 C 함수를 사용시 0점).

Standard C 헤더 파일 및 함수, constant 목록은 (https://en.wikipedia.org/wiki/C_standard_library) 에서 확인 가능

- 채점 시 GCC 7.3.0 (컴파일 시 과제에서 주어진 Makefile 이용), 및 Visual studio 2019 에서 채점할 예정이며, 둘 중 한곳에서 컴파일이 안될 시 무조건 0점.

- 제출하기 전에 예제로 주어진 main.c 파일을 컴파일 및 실행해서 실제로 올바르게 출력되는지 확인할 것.