Homework # 9

데이타구조론 (CSE2003-02)

HW#9 과제 안내

- 일정
  - 게시 : 5/20(목) 13:00
  - 제출 마감 : 5/27(목) 11:00 (delay 없음)
  - 채점 결과 확인 : 5/31(월)
  - 이의신청 마감 : 6/7(월) (이의신청 이메일 : greenlife124@yonsei.ac.kr,
    [데이타구조론HW#9 이의신청])

HW#9 과제 안내

- 설명
  - 모든 코드의 핵심 부분에는 comment를 달아 설명 할 것 (not option!!)
  - Compiler는 visual studio 2019 이상을 사용하여, HW#9_학번_이름 하나의 파일로 압축하여 제출 할 것
- HW#9_학번_이름
  - HW#9_1 > **ArrayGraph.c**, ArrayGraph.h, GraphMain.c
  - HW#9_2 > GraphMain.c, **ListGraph.c**, ListGraph.h
  - HW#9_3 > GraphMain.c, **LinkedQueue.c,** LinkedQueue.h, **LinkedStack.c**, LinkedStack.h, **ListGraph.c**, ListGraph.h

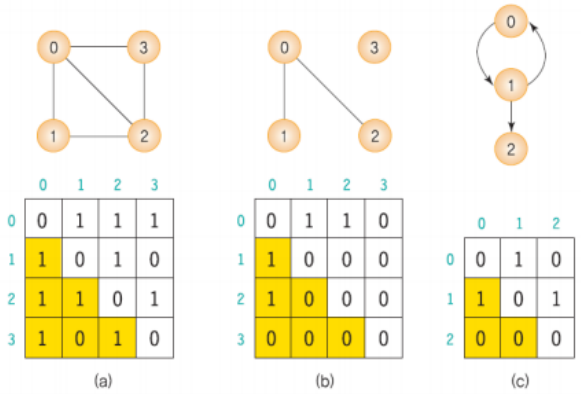
HW#9_1 인접 행렬

- 아래와 같이 실행되도록 **ArrayGraph.c** 작성
  - **insertEdge(), deleteVertex(), deleteEdge()** 작성
- ArrayGraph.h, GraphMain.c제공
- 주의 사항
  - HW#9_1 폴더 안에, 주어진 3개의 파일 모두 존재 해야 함

```
G1의 인접행렬
 0       1       0       1
 1       0       1       1
 0       1       0       1
 1       1       1       0
G1의 인접행렬
 0       1       0       1
 1       0       0       1
 0       0       0       0
 1       1       0       0
G2의 인접행렬
 0       1       0       1
 0       0       1       1
 0       0       0       1
 0       0       0       0
G2의 인접행렬
 0       1       0       1
 0       0       0       1
 0       0       0       0
 0       0       0       0
G3 인접행렬
 0       1       1
 1       0       1
 1       1       0
G3 인접행렬
 0       0       1
 0       0       1
 1       1       0
G4 인접행렬
 0       1       1
 0       0       1
 0       0       0
G4 인접행렬
 0       0       1
 0       0       1
 0       0       0
```

HW#9_1 참고 자료

- 행렬에 대한 **2차원 배열**을 사용하는 순차 자료구조 방법
- 그래프의 두 정점을 연결한 간선의 유무를 행렬로 저장
  - n개의 정점을 가진 그래프 : n x n 정방행렬 M (행, 열 번호가 정점)
  - **if (간선 (i, j)가 그래프에 존재) M[i][j] = 1,**
    그렇지 않으면 M[i][j] = 0
- 인접행렬의 대각선 성분은 모두 0 (자체 간선 불허)

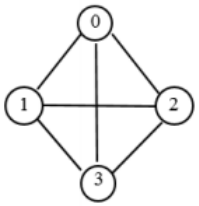


HW#9_2 인접 리스트

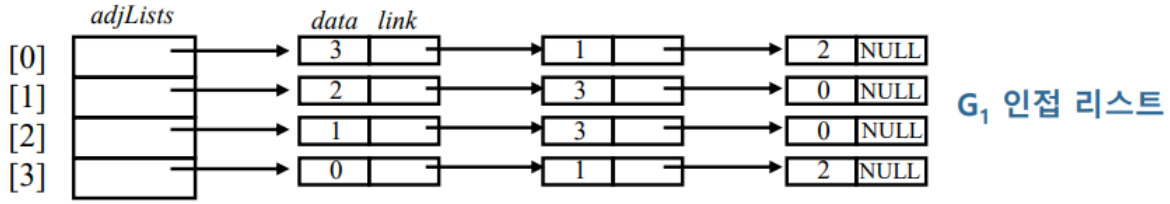
- 아래와 같이 실행되도록 **ListGraph.c** 작성
  - **insertEdge(), deleteVertex(), deleteEdge()** 작성
- ArrayGraph.h, GraphMain.c제공
- 주의 사항
  - HW#9_2 폴더 안에, 주어진 3개의 파일 모두 존재 해야 함

```
G1의 인접 리스트
정점 0의 인접리스트 -> 3  -> 1
정점 1의 인접리스트 -> 3  -> 2  -> 0
정점 2의 인접리스트 -> 3  -> 1
정점 3의 인접리스트 -> 2  -> 1  -> 0
G1의 인접 리스트
정점 0의 인접리스트 -> 3  -> 1
정점 1의 인접리스트 -> 3  -> 0
정점 2의 인접리스트
정점 3의 인접리스트 -> 1  -> 0
G2의 인접 리스트
정점 0의 인접리스트 -> 3  -> 1
정점 1의 인접리스트 -> 3  -> 2
정점 2의 인접리스트 -> 3
정점 3의 인접리스트
G2의 인접 리스트
정점 0의 인접리스트 -> 3  -> 1
정점 1의 인접리스트 -> 3
정점 2의 인접리스트
정점 3의 인접리스트
G3 인접 리스트
정점 0의 인접리스트 -> 2  -> 1
정점 1의 인접리스트 -> 2  -> 0
정점 2의 인접리스트 -> 1  -> 0
G3의 인접 리스트
정점 0의 인접리스트 -> 2
정점 1의 인접리스트 -> 2
정점 2의 인접리스트 -> 1  -> 0
G4 인접 리스트
정점 0의 인접리스트 -> 2  -> 1
정점 1의 인접리스트 -> 2
정점 2의 인접리스트
G4의 인접 리스트
정점 0의 인접리스트 -> 2
정점 1의 인접리스트 -> 2
정점 2의 인접리스트
```

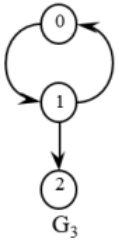
HW#9_2 참고 자료



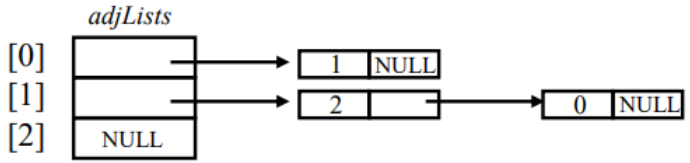
G_1



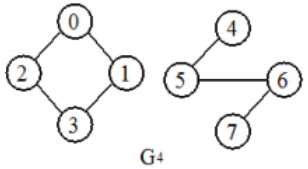
G_1 인접 리스트



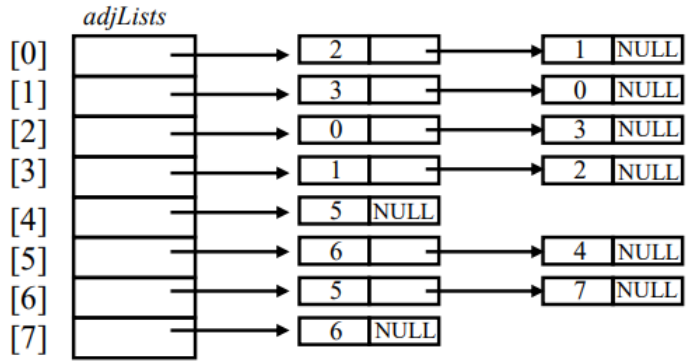
G_3



G_3 인접 리스트



G_4



G_4 인접 리스트

HW#9_3 그래프 탐색 구현

- 아래와 같이 실행되도록 **LinkedQueue.c** 작성
  - **isEmptyQueue(), enqueue(), dequeue()** 작성
- 아래와 같이 실행되도록 **LinkedStack.c** 작성
  - **isEmpty(), push(), pop()** 작성
- 아래와 같이 실행되도록 **ListGraph.c** 작성
  - **insertEdge(), dfs_iter(), dfs_recur(), bfs()** 작성
- GraphMain.c, LinkedQueue.h, LinkedStack.h, ListGraph.h 제공
- 주의 사항
  - HW#9_3 폴더 안에, 주어진 7개의 파일 모두 존재 해야 함

HW#9_3 실행 화면

```
G1 인접리스트
정점 0의 인접리스트 -> 1  -> 2
정점 1의 인접리스트 -> 0  -> 3  -> 4
정점 2의 인접리스트 -> 0  -> 4
정점 3의 인접리스트 -> 1  -> 6
정점 4의 인접리스트 -> 1  -> 2  -> 6
정점 5의 인접리스트 -> 6
정점 6의 인접리스트 -> 3  -> 4  -> 5

G1 DFS iterative version:       0       1       3       6       4       2       5
G1 DFS recursive version:       0       1       3       6       4       2       5
G1 BFS: 0       1       2       3       4       6       5

G2 인접리스트
정점 0의 인접리스트 -> 1  -> 2
정점 1의 인접리스트 -> 0  -> 3  -> 4
정점 2의 인접리스트 -> 0  -> 5  -> 6
정점 3의 인접리스트 -> 1  -> 7
정점 4의 인접리스트 -> 1  -> 7
정점 5의 인접리스트 -> 2  -> 7
정점 6의 인접리스트 -> 2  -> 7
정점 7의 인접리스트 -> 3  -> 4  -> 5  -> 6

G2 DFS iterative version:       0       1       3       7       4       5       2       6
G2 DFS recursive version:       0       1       3       7       4       5       2       6
G2 BFS: 0       1       2       3       4       5       6       7
```

HW#9_3 참고 자료

```
int isEmptyQ(Queue* Q){
    return Q->front == NULL;
}
```

Algorithm	void enqueue(Queue* Q, element x)

```
enqueue(Q, x)
    newNode.data ← x
    newNode.link ← NULL
    if (isEmptyQ(Q)) then
        Q.front ← newNode
    else
        Q.rear.link ← newNode
    endif
    Q.rear ← newNode
end enqueue()
```

Algorithm	element dequeue(Queue* Q)

```
dequeue(Q)
    if (isEmptyQ(Q)) then return ERROR
    else
        temp ← Q.front
        e ← temp.data
        Q.front ← temp.link
        delete temp
        if (Q.front = NULL) then Q.rear = NULL
        return e
    endif
end dequeue()
```

HW#9_3 참고 자료

```c
int isEmpty(Stack* S){
    return S->top == NULL;
}
```

```c
void push(Stack* S, element x) {
    stackNode* newNode = (stackNode*)malloc(sizeof(stackNode));
    newNode->data = x;
    newNode->link = S->top;
    S->top = newNode;
}
```

```c
element pop(Stack* S) {
    stackNode* temp; element e;
    if( isEmpty(S) ){
        printf("[ERROR] Stack is EMPTY!!\n");
        return ERROR;
    } else {
        temp = S->top;
        e = temp->data;
        S->top = temp->link;
        free(temp);
        return e;
    }
}
```

HW#9_3 참고 자료

Algorithm void dfs_iter(Graph* G, int v)

```
dfs(G, v)
   S ← createStack()
   G.visited[v] ← TRUE
   push(S, v)
   visit(v)
   while (not isEmpty(S)) do
      v ← pop(S)
      w ← v의 인접 정점
      while 인접정점이 있으면 do
         if w가 방문되지 않았으면 then
            push(S, v)
            G.visited[w.vertex] ← TRUE
            visit(w.vertex)
            v ← w.vertex
            w ← v의 인접 정점
         else
            w ← w.link
end dfs()
```

Algorithm void dfs_recur(Graph* G, int v)

```
dfs(G, v)
   visit(v)
   visited[v] ← true
   for all w ∈ (v 인접 정점) do
      if w가 아직 방문되지 않았으면 then
         pred[w] ← v
         dfs(G, w)
end dfs()
```

Algorithm void bfs(Graph* G, int v)

```
bfs(G, v)
   Q ← createQueue()
   G.visited[v] ← TRUE
   enqueue(Q, v)
   visit(v)
   while (not isEmpty(Q)) do
      v ← dequeue(Q)
      for all w ∈ (v의 인접 정점) do
         if w가 방문되지 않았으면 then
            enqueue(Q, w.vertex)
            G.visited[w.vertex] ← TRUE
            visit(w.vertex)
end dfs()
```