CS23510
Data Structures
Homework 4

2017/11/23 10:10pm

~

2017/12/07 23:59pm

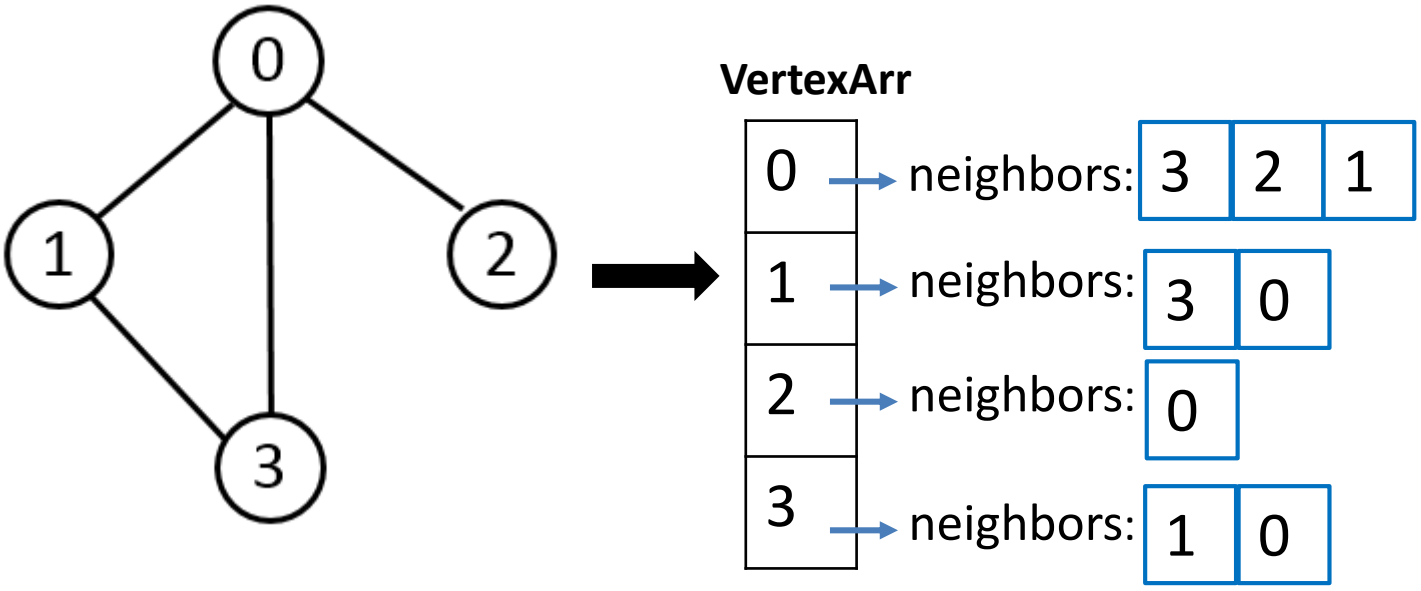
(Hard deadline)

Target

- The target of the homework is to construct an simple undirected graph (No self loops, No multiple edges) by using an adjacency list.

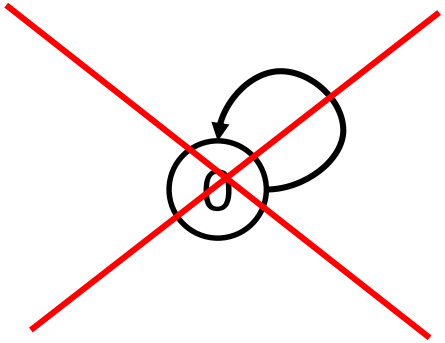
Target

- E.g.

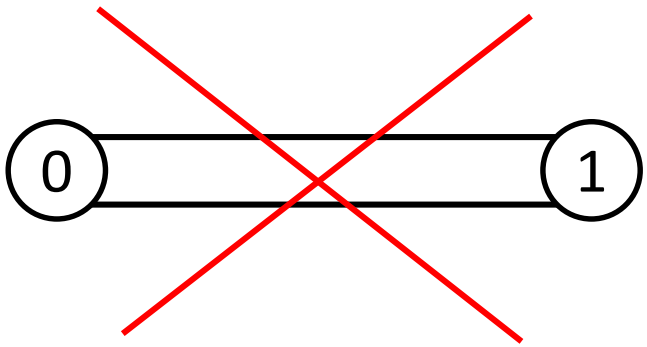

0
1
2
3
VertexArr
0
neighbors:
3
2
1
1
neighbors:
3
0
2
neighbors:
0
3
neighbors:
1
0

Target

- No self loops



- No multiple edges



Target

- Implement these 8 functions
- addEdge(A, B, C)
  - Add an edge between A & B with weight C, if A or B doesn't exist in the graph, create new vertex A or B in this graph too
- deleteEdge(A, B)
  - Delete the edge between A & B, if this edge doesn't exist, then do nothing
- deleteVertex(A)
  - delete the vertex A and all edges that connect to A, if A doesn't exist, do nothing

Target

- degree(A)
  - return the degree of vertex A, return 0 if A doesn't exist
- isExistPath(A, B)
  - return true(bool) if there is at least one path between A & B, else return false(bool), if A or B doesn't exist in the graph, return false
- deleteGraph()
  - delete all vertices and edges in the graph
- number_of_component()
  - return the number of the components, if this graph doesn't have any vertex, then return 0

Target

- isExistCycle()
  - return true(bool) if there is at least one cycle in any component, else return false(bool)

Target

- “Vertex” - the data structure of the vertex.
- “Neighbor” - the data structure about a neighbor of a node.
- “GraphOperations” - it specifies the functions to be implemented.
- “Implement” - your implementation.

Class - Vertex

- A label (n = 0, 1, 2, …., 99) represents an unique vertex in the graph

```
class Vertex {
public:
    //--------------------------------
    // label of the vertex
    //--------------------------------
    int label;
    int v_degree;
    //--------------------------------
    // record the neighbor vertex
    //--------------------------------
    std::list<Neighbor> neighbors;

    Vertex(){};
    Vertex(const int label)
    {
        this->label = label;
        v_degree = 1;
    };
    ~Vertex(){};
};
```

Class - Neighbor

- This class is going to help you maintain the information of a neighbor.
- Weight: the weight of the edge between this vertex and the neighbor

```
class Neighbor{
public:
    int label;
    int weight;

    Neighbor(){};
    Neighbor(const int label)
    {
        this->label = label;
    };
    Neighbor(const int label, const int weight)
    {
        this->label = label;
        this->weight = weight;
    };

    ~Neighbor(){};

};
```

Class - GraphOperations

```
class GraphOperations
{
public:
    //------------------------------------------------------------------------
    // a member to create an adjacencyList
    //------------------------------------------------------------------------
    std::list<Vertex> VertexArr;

    //------------------------------------------------------------------------
    //add an edge between the A and the B with weight
    //if A or B doesn't exist in the graph,
    //create new A or B in this graph too
    //------------------------------------------------------------------------
    virtual void addEdge(const int label_1, const int label_2 , const int weight) = 0;

    //------------------------------------------------------------------------
    //delete an edge between A and B
    //if this edge doesn't exist, then do nothing
    //------------------------------------------------------------------------
    virtual void deleteEdge(const int label_1, const int label_2) = 0;

    //------------------------------------------------------------------------
    //delete the vertex A and all edges that connect to A, if A doesn't exist, do nothing
    //------------------------------------------------------------------------
    virtual void deleteVertex(const int label) = 0;

    //------------------------------------------------------------------------
    //return the degree of vertex A, return 0 if A doesn't exist
    //------------------------------------------------------------------------
    virtual int degree(const int label) = 0;
```

Class - GraphOperations

```
    //------------------------------------------------------------------------
    //return true(bool) if there is at least one path between A & B, else return false(bool),
    //if A or B doesn't exist in the graph, return false
    //------------------------------------------------------------------------
    virtual bool isExistPath(const int label_1, const int label_2) = 0;
    //------------------------------------------------------------------------
    //delete all vertexes and edges in the graph
    //------------------------------------------------------------------------
    virtual void deleteGraph() = 0;
    //------------------------------------------------------------------------
    //return the numbers of the component, if this graph doesn't have any vertex, then return 0
    //------------------------------------------------------------------------
    virtual int number_of_component() = 0;

    //------------------------------------------------------------------------
    //return true(bool) if there is at least one cycle in any component, else return false(bool)
    //------------------------------------------------------------------------
    virtual bool isExistCycle() = 0;
};
```

Class - Implement

```
class Implement : public GraphOperations
{
public:
    void addEdge(const int label_1, const int label_2 , const int weight);
    void deleteEdge(const int label_1, const int label_2);
    void deleteVertex(const int label);
    int degree(const int label);
    bool isExistPath(const int label_1, const int label_2);
    void deleteGraph();
    int number_of_component();
    bool isExistCycle();
};
```

std::list

- We use std::list in this homework
- Here are some often used methods of list
  - begin: Return iterator to beginning
  - end: Return iterator to end
  - erase/remove: Delete elements
  - push_back: Add element at the end
- Website for more details about usage of list
  - http://www.cplusplus.com/reference/list/list/

Judge

- Use partial online judge to submit your code and test
- https://acm.cs.nthu.edu.tw/problem/11695/
  You have to **#include “function.h”**

```
#include "function.h"
#include ......

void Implement::addEdge(const int label_1, const int label_2, const int weight){
  .......
}

......
```

Submission

Submit your **code.cpp** to iLMS system BEFORE the deadline.