CSCI 136
Data Structures & Advanced Programming

Lecture 31

Fall 2017

Instructors: Bills

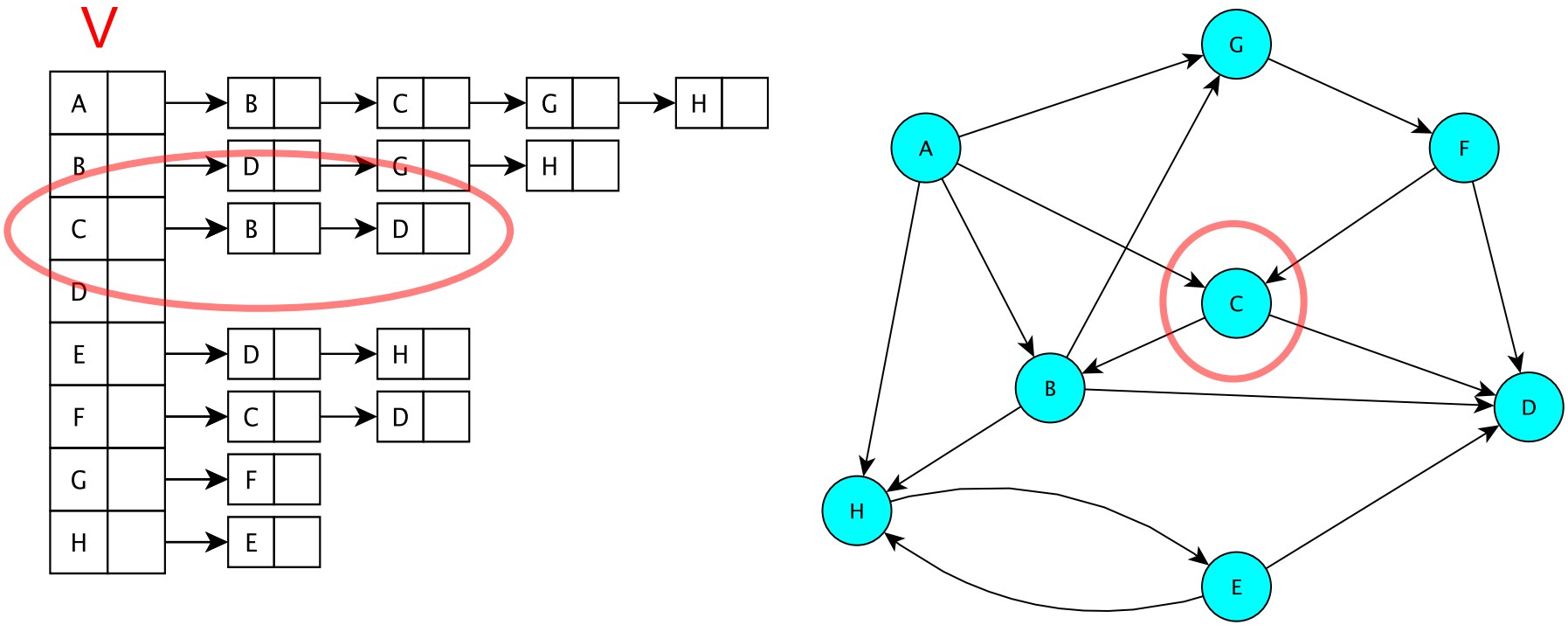
Last Time

- Adjacency Matrix Implementation Details
- Greedy Algorithms for Optimization
  - At each step, make decision that brings you closest to your goal
  - Does not always lead to optimum solution!
- Lab 11 : Exam Scheduling – graph coloring in disguise

Today's Outline

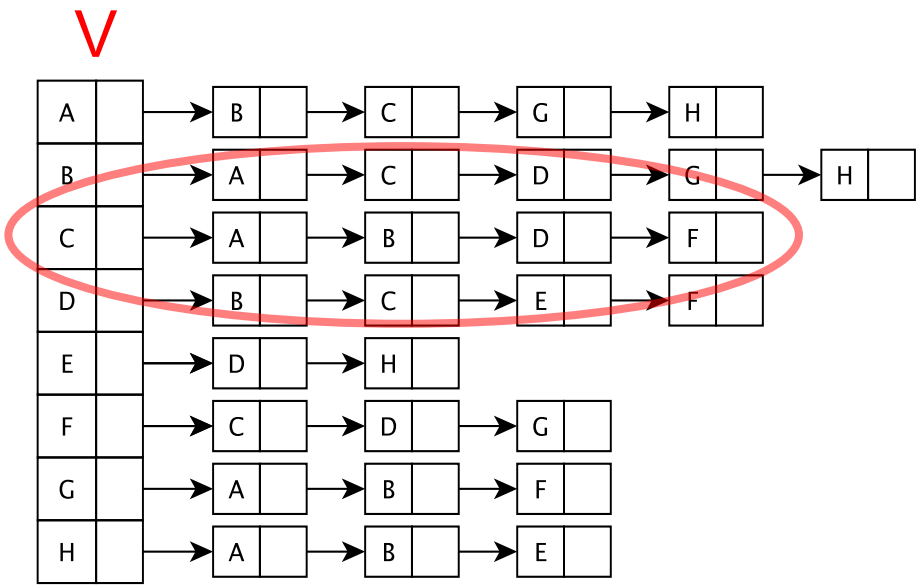
- Adjacency List Implementation Details
- GraphList Time/Space Complexity
- An Important Algorithm: Minimum-cost spanning subgraph

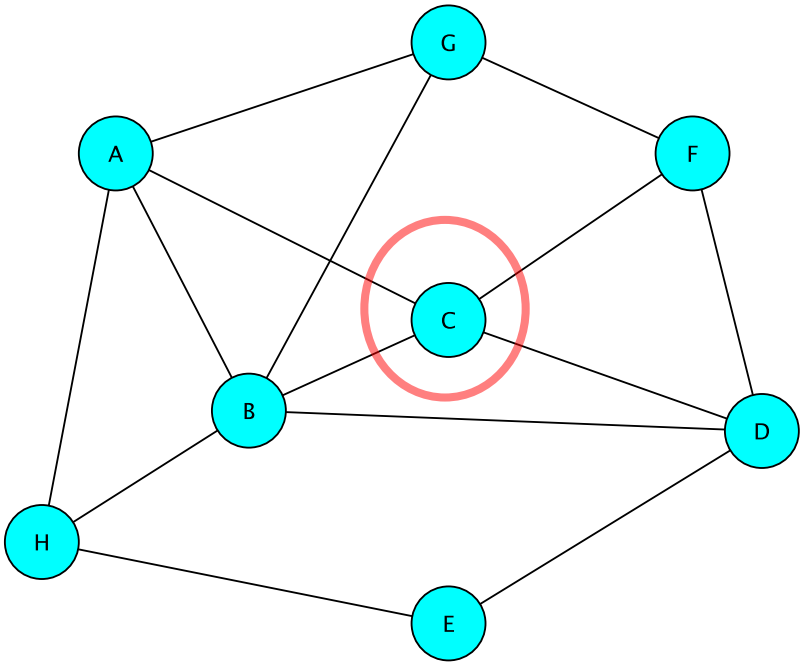
Adjacency List : Directed Graph



The vertices are stored in an array V[]
V[i] contains a linked list of all edges with a given source

Adjacency List : Undirected Graph





The vertices are stored in an array V[]

V[i] contains a linked list of *all* edges incident to a given vertex

Example: Lab 11

Jeannie

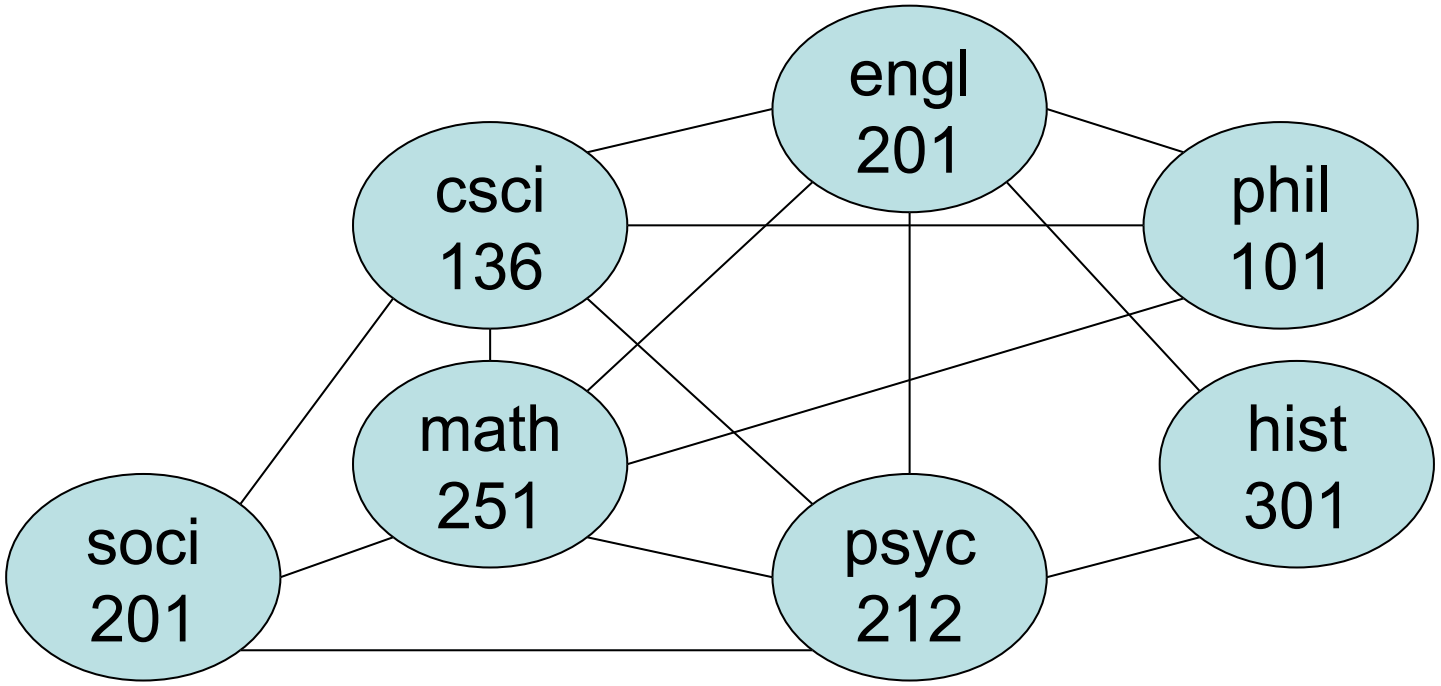
- CSCI 136
- MATH 251
- ENGL 201
- PHIL 101

Duane

- PSYC 212
- ENGL 201
- HIST 301
- CSCI 136

Andrea

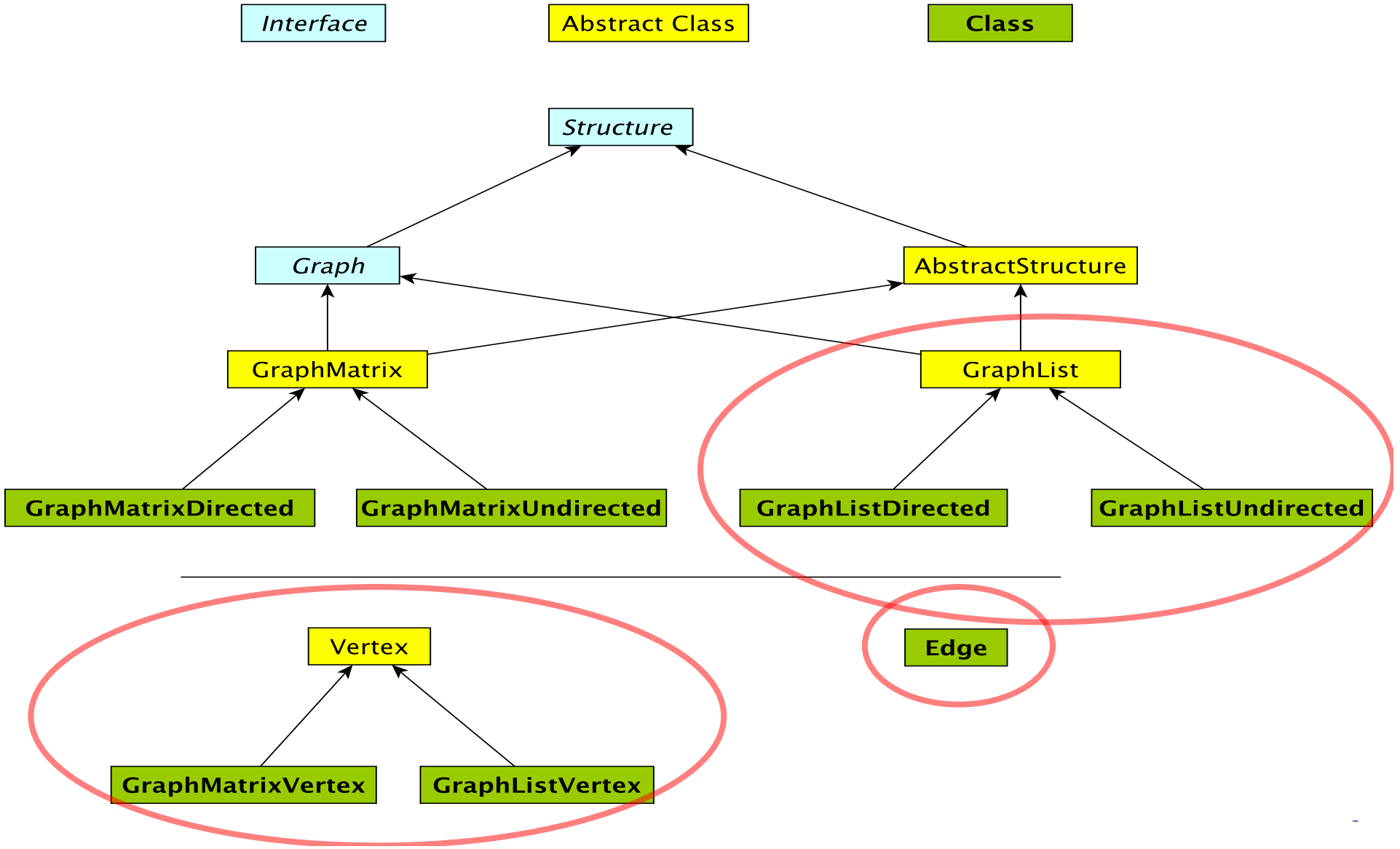
- SOCI 201
- CSCI 136
- MATH 251
- PSYC 212



GraphList: Big Picture

- Maintain an *adjacency list* of *edges* at each vertex (no adjacency matrix)
  - Keep only outgoing edges for directed graphs
- Support both directed and undirected graphs (GraphListDirected, GraphListUndirected)

Graph Classes in structure5

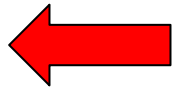


Vertex and GraphListVertex

- We use the same `Edge` class for all graph types
- We extend `Vertex` to include an `Edge` list
- `GraphListVertex` class adds to `Vertex` class:
  - A `Structure` to store edges adjacent to the vertex

    protected Structure<Edge<V,E>> adjacencies; // adjacent edges
    - adjacencies is created as a SinglyLinkedList of edges
  - Several methods

```
public void addEdge(Edge<V,E> e)
public boolean containsEdge(Edge<V,E> e)
public Edge<V,E> removeEdge(Edge<V,E> e)
public Edge<V,E> getEdge(Edge<V,E> e)
public int degree()
// and methods to produce Iterators...
```



GraphListVertex

```
public GraphListVertex(V label){
        super(label); // init Vertex fields
        adjacencies = new SinglyLinkedList<Edge<V,E>>();
}

public boolean containsEdge(Edge<V,E> e){
        return adjacencies.contains(e);
}

public void addEdge(Edge<V,E> e){
        if (!containsEdge(e)) adjacencies.add(e);
}

public Edge<V,E> removeEdge(Edge<V,E> e) {
        return adjacencies.remove(e);
}
```

GraphListVertex Iterators

```
// Iterator for incident edges
public Iterator<Edge<V,E>> adjacentEdges() {
        return adjacencies.iterator();
}

// Iterator for adjacent vertices
public Iterator<V> adjacentVertices() {
        return new GraphListAIterator<V,E>
            (adjacentEdges(), label());
}
```

GraphListAIterator creates an Iterator over *vertices* based on the Iterator over *edges* produced by `adjacentEdges()`

GraphListAIterator

GraphListAIterator is a class with two instance variables:

```
    protected Iterator<Edge<V,E>> edges;
    protected V vertex;

public GraphListAIterator(Iterator<Edge<V,E>> i, V v) {
        edges = i;
        vertex = v;
}

public V next() {
        Edge<V,E> e = edges.next();
        if (vertex.equals(e.here()))
            return e.there();
        else { // could be an undirected edge!
            return e.here();
}
```

GraphList (Abstract base class)

- To implement `GraphList`, what data structures do we need?
  - (Maintain an *adjacency list* of *edges* at each vertex)
- `GraphListVertex` class
  - Instance vars: `label`, `visited` flag, linked list of edges
- "Array V[]" of `GraphListVertex`
  - Lies! We actually use a Map from V to GraphListVertex:

    Map<V,GraphListVertex<V,E>> dict; // label -> vertex
- Do we need a free list like `GraphMatrix`?
- Do we need to know |V| ahead of time?

GraphList

```
protected Map<V,GraphListVertex<V,E>> dict;
protected boolean directed;

protected GraphList(boolean dir){
      dict = new Hashtable<V,GraphListVertex<V,E>>();
      directed = dir;
}

public void add(V label) {
      if (dict.containsKey(label)) return;
      GraphListVertex<V,E> v = new
            GraphListVertex<V,E>(label);
      dict.put(label,v);
}
```

```
public Edge<V,E> getEdge(V label1, V label2) {
      Edge<V,E> e = new Edge<V,E>(get(label1),
                                  get(label2), null, directed);
      return dict.get(label1).getEdge(e);
}
```

(in GraphListVertex)

```
public Edge<V,E> getEdge(Edge<V,E> e) {
      Iterator<Edge<V,E>> edges = adjacencies.iterator();
      while (edges.hasNext()) {
            Edge<V,E> adjE = edges.next();
            if (e.equals(adjE))
                  return adjE;
      }
      return null;
}
```

GraphListDirected

- GraphListDirected (GraphListUndirected) implements the methods requiring different treatment due to (un)directedness of edges
  - addEdge, remove, removeEdge, …

- (We will only look at GraphListDirected in class)

```
// addEdge in GraphListDirected.java
// first vertex is source, second is destination
 public void addEdge(V vLabel1, V vLabel2, E label) {
    // first get the vertices
    GraphListVertex<V,E> v1 = dict.get(vLabel1);
    GraphListVertex<V,E> v2 = dict.get(vLabel2);
    // create the new edge
    Edge<V,E> e = new Edge<V,E>(v1.label(), v2.label(), label, true);
    // add edge only to source vertex linked list (aka adjacency list)
    v1.addEdge(e);
}
```

```
public V remove(V label) {
   //Get vertex out of map/dictionary
   GraphListVertex<V,E> v = dict.get(label);

   //Iterate over all vertex labels (called the map "keyset")
   Iterator<V> vi = iterator();
   while (vi.hasNext()) {
        //Get next vertex label in iterator
        V v2 = vi.next();

        //Skip over the vertex label we're removing
        //(Nodes don't have edges to themselves...)
        if (!label.equals(v2)) {
            //Remove all edges to "label"
            //If edge does not exist, removeEdge returns null
            removeEdge(v2,label);
        }
    }
    //Remove vertex from map
    dict.remove(label);
    return v.label();
}
```

```
public E removeEdge(V vLabel1, V vLabel2) {
   //Get vertices out of map
   GraphListVertex<V,E> v1 = dict.get(vLabel1);
   GraphListVertex<V,E> v2 = dict.get(vLabel2);

   //Create a "temporary" edge connecting two vertices
   Edge<V,E> e = new Edge<V,E>(v1.label(), v2.label(), null, true);

   //Remove edge from source vertex linked list
   e = v1.removeEdge(e);
   if (e == null) return null;
   else return e.label();
}
```

Efficiency Revisited

- Assume Map operations are O(1) (for now)
  - |E| = number of edges
  - |V| = number of vertices
- Runtime of add, addEdge, getEdge, removeEdge, remove?
- Space usage?
- Conclusions
  - Matrix is better for dense graphs
  - List is better for sparse graphs
  - For graphs "in the middle" there is no clear winner

Efficiency : Assuming Fast Map

	Matrix	GraphList
add	O(1)	O(1)
addEdge	O(1)	O(1)
getEdge	O(1)	O(|V|)
removeEdge	O(1)	O(|V|)
remove	O(|V|)	O(|V|+|E|)
space	$O(\|V\|^2)$	O(|V|+|E|)