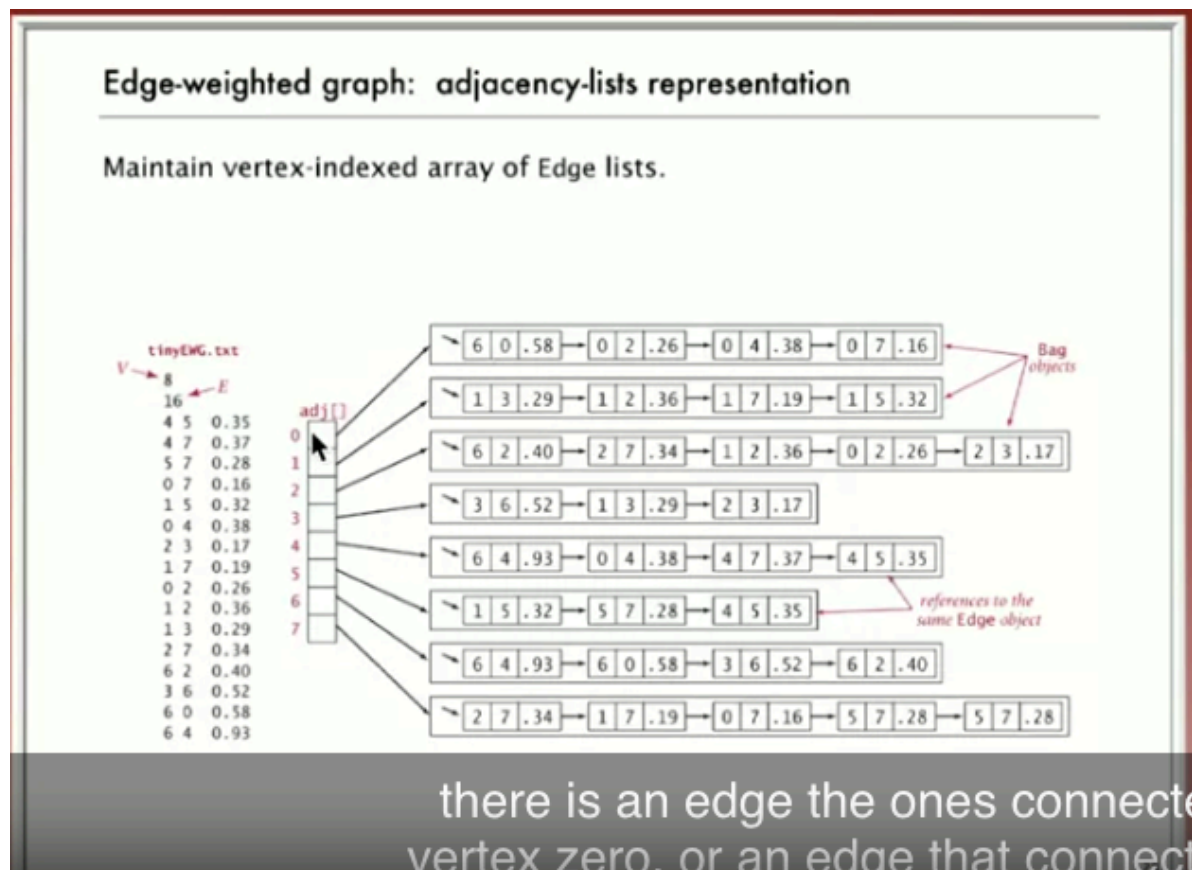


# Graph Studies

## Graph API

```
public class Edge implements Comparable<Edge>
    Edge(int v, int w, double weight) //constructor
    int either() //return the start point
    int other(int v) //return other vertex
    int compareTo(Edge that) //比较权重
```

维护一个数组adj[EdgeList]



## 图的表示, 面向对象相关

有一个整数V, 然后维护一个长度为V的集合.

## Edge-weighted graph: adjacency-lists implementation

```
public class EdgeWeightedGraph
{
    private final int V;
    private final Bag<Edge>[] adj;
```

← same as Graph, but adjacency lists of Edge instead of integers

```
    public EdgeWeightedGraph(int V)
    {
        this.V = V;
        adj = (Bag<Edge>[]) new Bag[V];
        for (int v = 0; v < V; v++)
            adj[v] = new Bag<Edge>();
    }
```

← constructor

```
    public void addEdge(Edge e)
    {
        int v = e.either(), w = e.other(v);
        adj[v].add(e);
        adj[w].add(e);
    }
```

← add edge to both adjacency lists

```
    public Iterable<Edge> adj(int v)
    { return adj[v]; }
```

## MST的表示

给一个图, 然后返回iterator MST的节点或者权重

Minimum spanning tree API

Q. How to represent the MST?

```
public class MST
{
    MST(EdgeWeightedGraph G)  // constructor
    Iterable<Edge> edges()    // edges in MST
    double weight()           // weight of MST
}
```

tinyEWG.txt

V → 8  
E → 16

V	W	E
4	5	0.35
4	7	0.17
5	7	0.28
0	7	0.16
1	5	0.32
0	4	0.18
2	3	0.17
1	7	0.19
0	2	0.26
6	2	0.40

MST edge (black)

non-MST edge

```
% java MST tinyEWG.txt
0-7 0.16
1-7 0.19
0-2 0.26
2-3 0.17
5-7 0.28
4-5 0.35
6-2 0.40
```

So for example if we take a, an example with our tiny edge-weighted graph,

9:27 / 11:15

## Kruskal's Algorithm(无像连通图)

首先sort所有的边(使用priorityQueue), 然后找到最短的, 看两个点是否标记, 没标记, 就把两个都标记. 入栈

## Prim's Algorithm

## MST用priority queue

minimum-cost spanning tree 最小生成树

## Prim's algorithm: lazy implementation

Press **esc** to exit full screen

```
public class LazyPrimMST
{
    private boolean[] marked; // MST vertices
    private Queue<Edge> mst; // MST edges
    private MinPQ<Edge> pq; // PQ of edges

    public LazyPrimMST(WeightedGraph G)
    {
        pq = new MinPQ<Edge>();
        mst = new Queue<Edge>();
        marked = new boolean[G.V()];
        visit(G, 0);

        while (!pq.isEmpty())
        {
            Edge e = pq.delMin();
            int v = e.either(), w = e.other(v);
            if (marked[v] && marked[w]) continue;
            mst.enqueue(e);
            if (!marked[v]) visit(G, v);
            if (!marked[w]) visit(G, w);
        }
    }
}
```

← assume G is connected

← repeatedly delete the min weight edge  $e = v-w$  from PQ

← ignore if both endpoints in T

← add edge e to tree

← add v or w to tree

52

先找一个节点标记mark, 然后用priorityQueue把这个节点所有的边enqueue进去, 然后根据priorityQueue拿出来最小的, 然后再把拿出来的节点标记mark, 把他的边enqueue. 如果poll的两个顶点都用过就continue.  $n - 1$ 条边的时候停止.

[union find](#)

[link](#)解释的很好