Data Structures and Algorithm Analysis

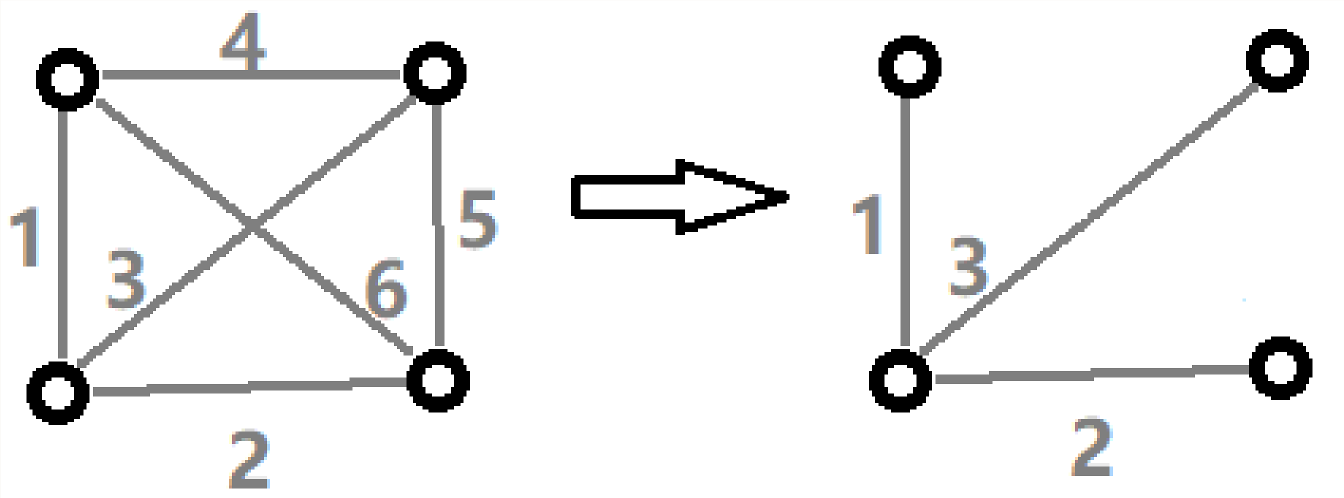
Lab 14, Minimal spanning tree

Contents

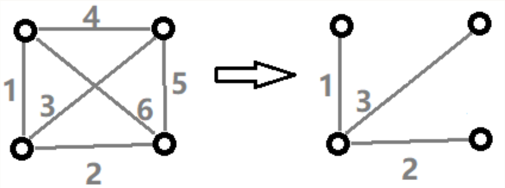
- Prim’s algorithm
- Kruskal’s algorithm

Recall minimal spanning tree

Given a graph and associate each edge with a weight, what is a minimal spanning tree?



Recall minimal spanning tree



A minimal spanning tree of a graph is a tree such that:

- Contains all the vertices of the original graph.
- Contains a subset of edges from the original graph.
- The edges connect all vertices.
- The edges form a tree (hence no cycle).
- The sum of weights of the tree is minimal.

Basic procedure of finding minimal spanning tree

The basic procedure is simple:

Step 1: find an edge satisfing the following property:

- You can divide the graph into two parts, there is no edge selected between the two parts.
- This edge is the has the minimal weight between the two parts.

Step 2: Add this edge into the minimal spanning tree. If there is V-1 edges in the tree, stop. Otherwise go to step 1.

Represent a graph with weights in JAVA

First we need to modify the Graph definition in JAVA we have introduced last week. We must add weights to the edges.

```
public interface WeightedGraph<W> {
  // return number of vertices.
  int size();

  // add an edge between v1 and v2, 0 <= v1, v2 <= size()-1
  void addEdge( int v1, int v2, W weight );

  // return all edges that vertex v is connected to.
  Iterable<Edge<W>> adjacency( int v );

  // return the edge from v1 to v2, null if not exist.
  Edge<W> getEdge( int v1, int v2 );
}
```

Graph with adjacency matrix

We should modify our adjacency matrix implementation and add weights.

```
public class GraphAdjacency<W> implements WeightedGraph<W> {

  private int num;
  private Edge<W>[][] adj;

  public GraphAdjacency( int verticesNumber ) {
      num = verticesNumber;
      adj = new Edge[num][num];
  }

...
```

Again be aware of the generic array problem when creating the 2d array.

Graph with adjacency list

We also modify our adjacency list graph implementation and add weights.

```
public class GraphAdjList<W> implements WeightedGraph<W> {

  private int num;
  private LinkedList<Edge<W>>[] adjList;

  public GraphAdjList( int verticesNumber ) {
      num = verticesNumber;
      adjList = new LinkedList[num];
      for( int i = 0; i < adjList.length; i ++ )
          adjList[i] = new LinkedList<>();
  }

...
```

Just modify the corresponding methods in this class.

Implement Kruskal's algorithm

- Put all edges in a priority queue.
- Retrive the edges one by one.
- If the two vertices are connected, do nothing,
- otherwise add the edge in the minimal spanning tree.
- If the mst have V-1 edges, stop.

In order to know whether two vertices are connected, you need to use union find.

Implement Kruskal's algorithm

```java
static LinkedList kruskal( WeightedGraph<W> graph ) {

  PriorityQueue<Edge<W>> minPQ = new PriorityQueue<>();
  for( int i = 0; i < graph.size(); ++ i )
      minPQ.addAll(graph.adjacency(i));

  UnionFind uf = new UnionFind(graph.size());

  LinkedList<Edge<W>> mst = new LinkedList<>();

  while( mst.size() < graph.size()-1 && !minPQ.isEmpty() ) {
      Edge<W> e = minPQ.poll();
      if( uf.isConnected(e.from, e.to) )
          continue;
      mst.add(e);
      uf.union(e.from, e.to);
  }
  return mst;
}
```

Prim's algorithm (Lazy)

- Select a vertex as the tree.
- At each step, select the minimal edge that "expends" the tree.
- If the mst have V-1 edges, stop.

This time you don't need the union find. But you still need the priority queue.

Prim’s algorithm (Lazy)

```
static LinkedList primLazy( WeightedGraph<W> graph ) {

  PriorityQueue<Edge<W>> minPQ = new PriorityQueue<>();
  boolean[] visited = new boolean[graph.size()];

  visited[0] = true;
  for( Edge<W> e : graph.adjacency(0) )
      minPQ.add(e);

  LinkedList<Edge<W>> mst = new LinkedList<>();

  while( mst.size() < graph.size()-1 && !minPQ.isEmpty() ) {
      Edge<W> e = minPQ.poll();
      if( visited[e.to] )
          continue;
      visited[e.to] = true;
      mst.add(e);
      for( Edge<W> edge : graph.adjacency(e.to) )
          minPQ.add(edge);
  }
  return mst;
```

Prim’s algorithm (eager)

The eager version of the prim is an improvement of the lazy one. It reduces the size of the priority queue by not putting useless edges in it.

- Record the min distance from the tree to each vertex.
- Edge with large weight do not add to the tree.
- Duplicate edge do not add to the tree.

In this version you need a different version of priority queue too. This pq should support changing the weight of an element with the given index.