1. [10 marks] Implement Dijkstra’s algorithm for shortest paths. It is described in the textbook, section 7.1.1. It has some similarity with Prim’s algorithm; the difference is in the meaning of priorities and how they are changed.

The starter code and test cases are on Blackboard. A few remarks:

- For simplicity, vertices are int’s from 0 to |V| − 1 inclusive.
- An input graph is given as an array of adjacency lists. It is an undirected, connected graph. Weights are non-negative (a requirement of Dijkstra’s algorithm).
- Your algorithm inputs a graph, a start vertex, and a target vertex. It returns a shortest path as a list of vertices, including the start and the target.

2. (a) [5 marks] Perform these union-find operations in the order listed. Just hand in the final picture.
   i. Start with 9 elements: A, B, C, D, E, F, G, H, I.
   ii. Perform a makeSet on each of those elements.
   iii. union(A,B), union(C,D), union(A,C), union(E,F), union(G,E), union(H,I), union(E,H), union(A,E).
   iv. find(I).

   During union, if the two sets have the same size, merge the second set into the first set. Example: after union(A,B), B’s parent is A.

(b) [5 marks] Give a sequence of makeSet and union operations that builds a set such that the longest path from an element to the root consists of 5 elements (i.e., 4 edges). Use the fewest number of elements possible. (Hint: The fewest possible is far more than 5.)

(c) [5 marks] Prove that the number of elements you use in part (b) is the fewest possible.

3. We have seen extendable arrays. This question goes one step further to explore extendable, shrinkable arrays.

Again, we have an array A, and limit remembers how many cells of A are used. The operations are:

- Some read and write operations on the used cells. Omitted.
- add(x): use one more cell and store x there. As we have seen, if this exceeds the current array size, then transfer the data to a new array twice as big.
- remove(): use one fewer cell and ignore its content, i.e., decrease limit by 1. Now, if limit is “low enough”, transfer the data to a new, “smaller” array, in order to free up memory.

The question is: what are good criteria for “low enough” and “smaller”.

(a) [5 marks] Bad idea: When limit ≤ A.length/2, transfer to a new array half as big.

```java
public void remove() {
    if (limit == 0) return;
    limit--;
    if (2*limit <= A.length && A.length >= 2) {
        char[] B = new char[A.length/2];
        for (int i=0; i<limit; i++) {
            B[i] = A[i];
        }
        A = B;
    }
}
```

Give a sequence of 2·m operations consisting of add’s and remove’s that take \( \Theta(m^2) \) total time, with brief explanation why it is \( \Theta(m^2) \).

(b) [10 marks] Good idea: When limit < A.length/4, transfer to a new array half as big.
public void remove() {
    if (limit == 0) return;
    limit--;
    if (4*limit < A.length && A.length >= 2) {
        char[] B = new char[A.length/2];
        for (int i=0; i<limit; i++) {
            B[i] = A[i];
        }
        A = B;
    }
}

Prove that a sequence of \(m\) operations consisting of add’s and remove’s take total time \(O(m)\). You may use any constants for the initial values of \(\text{limit}\) and \(\text{A.length}\) that make your argument easier.

(This is essentially exercise C-1.32 in the textbook.)