HW#4
Due date: 5/30

1 Ex. 17.2.1 (10%)
Assume that COPY is invoked automatically, after every sequence of $k$ PUSH and POP operations.
Use accounting method to charge the operations PUSH, POP, and COPY.

2 Ex. 17.3.6
a) Write down the pseudocode of queue ENQUEUE and DEQUEUE operations. (5%)
Hint: ENQUEUE operations push elements onto one stack and DEQUEUE operations pop elements off the other stack.
b) Use accounting method to analyze the amortized costs of ENQUEUE and DEQUEUE (5%)
c) Redo b), but this time uses potential method. (5%)

3 a) Ex. 21.3-3 (5%)
b) Ex. 21.4-2 (10%)
First, prove by induction that a node with rank $k$ is the root of a subtree containing at least $2^r$ nodes.
c) Ex. 21.4-4 (5%)

4 Ex. 23.1-6 (5%)
The solution is posted publicly.

The remaining of this homework is to implement Kruskal's and Prim's algorithms. These will be our last two programming exercises in this semester.

5 [Programming exercise] 100%

Implement Kruskal's algorithm, using disjoint-set forests with the union-by-rank and path-compression heuristics.

Test your program on the graph given in lecture note on MST, p.12 and the graph given in book, p. 632.

It is up to you to decide how to represent a graph. You may use adjacency-matrix or adjacency-list representations, or any other representation that is convenient to you. (For example, since the testing graphs are sparse, you may wish to represent each graph as a vector of weighted edges.)

Sample test

```
int main()
{
    cout << "Kruskal's algorithm for graph in lecture\n";
    // define G1 as the graph in lecture
    kruskal(G1);
    cout << "Kruskal's algorithm for graph in book\n";
    // define G2 as the graph in book
    kruskal(G2);
}
```

Sample output

```
Kruskal's algorithm for graph in lecture
(c,f) (g,i) (c,e) (d,h) (f,h) (b,d) (d,g) (a,b)

Kruskal's algorithm for graph in book
(g,h) (c,i) (f,g) (a,b) (c,f) (c,d) (a,h) (d,e)
```

Comment

Since each graph has more than one MST, you may output any one.

6 [Programming exercise] 100%

Implement Prim's algorithm, using binary min-heap to implement min-priority queue.

## Requirement

You shall use the STL container **priority_queue**. Failing to meet this requirement will receive at most 60 grade points.

## Hint

The template class **priority_queue** doesn't support DECREASE_KEY operation and FIND$(v, Q)$ operation that checks if $v \in Q$, where $Q$ is a priority queue. Since Prim's algorithm needs these two operations, you have to write a class that publicly inherits from **priority_queue** and supports both operations.

Note: The template class **priority_queue** has two protected data members **c** and **comp** that can be used by the derived class you define.

## Sample test

```
int main()
{
    cout << "Prim's algorithm for graph in lecture\n";
    // define G1 as the graph in lecture
    prim(G1);
    cout << "Prim's algorithm for graph in book\n";
    // define G2 as the graph in book
    prim(G2);
}
```

## Sample output

```
Prim's algorithm for graph in lecture, starting with vertex a
(a,b) (b,d) (d,h) (h,f) (f,c) (f,e) (d,g) (g,i)

Prim's algorithm for graph in book, starting with vertex a
(a,b) (a,h) (h,g) (g,f) (f,c) (c,i) (c,d) (d,e)
```

## Comment

The starting vertex is up to you.