Lab Assignment-8
(Multithreading and Design Patterns)

IIIT-Delhi. 3rd November. Due by 23:59pm on 4th November 2017

Instructor: Vivek Kumar

Note that this is a mandatory lab assignment. No extensions will be provided. Any submission after the deadline will not be evaluated. If you see ambiguity or inconsistency in a question, please seek a clarification from the teaching staff.

Plagiarism: All submitted homeworks are expected to be the result of your individual effort. You should never misrepresent someone else's work as your own. In case any plagiarism case is detected, it will be dealt as per IIITD policy for plagiarism.

Problem Description:

Pascal's triangle is a triangular array of the binomial coefficients. The entry in n^{th} row and k^{th} column in a Pascal's triangle is denoted by C(n, k). The value C(n, k) can be recursively calculated using the following standard formula for binomial coefficient:

$$C(n, k) = C(n-1, k-1) + C(n-1, k)$$
$$C(n, 0) = C(n, n) = 1$$

A recursive sequential implementation of computation kernel in Pascal's triangle application is as show below:

```
int recurse(int n, int k) {
        if (n == 0 || k == 0 || n == k) {
                return 1;
        }

        int left = recurse(n - 1, k - 1);
        int right = recurse(n - 1, k);

        return (left + right);
}
```

Deliverables:

1) You have to use this sequential implementation of Pascal's triangle and parallelize it by using thread pool.

2) You will notice that your above parallel program will take ages to complete for large values of n and k. You have to resolve this issue in your parallel implementation by using a Flyweight design pattern.

3) Calculate the speedup of your parallel program by executing it with thread pool sizes = 1, 2 and 3. Choose the value of n and k such that your parallel implementation doesn't quickly complete its execution without using Flyweight design pattern that you implemented above.

Solution

Parallel Pascal using Flyweight pattern and thread pool:

```
import java.util.*;
import java.util.concurrent.*;

public class Pascal extends RecursiveTask<Long> {
   private long n, k;
   public Pascal(long _n, long _k) { n=_n; k=_k; }

   private static final boolean useFlyweight = true;

   private final static Map<String, Long> instances = new HashMap<String, Long>();

   private static long getPartialResult(long n, long k) {
      String key = n + "," + k;
      if(!instances.containsKey(key)) {
         return -1;
      }
      else return instances.get(key);
   }

   private static synchronized void addPartialResult(long n, long k, long r) {
      String key = n + "," + k;
      if(!instances.containsKey(key)) {
         instances.put(key, r);
      }
   }

   /**
    * Pascal's Triangle --- Computes (n C k)
    * C(n,k) = C(n - 1, k - 1) + C(n - 1, k)
    */
   public Long compute() {
```

```
        if(useFlyweight) {
            long result = getPartialResult(n, k);
            if(result >= 0) {
                return result;
            }
        }

        if (n == 0 || k == 0 || n == k) {
            return 1L;
        }

        Pascal left = new Pascal(n-1, k-1);
        Pascal right = new Pascal(n-1, k);

        left.fork();
        long right_result = right.compute();
        long left_result = left.join();

        long result = left_result + right_result;
        if(useFlyweight) {
            addPartialResult(n, k, result);
        }

        return result;
    }

    public static void main(String[] args) {
        long n = args.length > 0 ? Long.parseLong(args[0]) : 30;
        long k = args.length > 1 ? Long.parseLong(args[1]) : 10;
        int threads = args.length > 2 ? Integer.parseInt(args[2]) : 1;

        Pascal p = new Pascal(n,k);
        ForkJoinPool pool = new ForkJoinPool(threads);

        long start = System.currentTimeMillis();
        long result = pool.invoke(p);
        long time = System.currentTimeMillis() - start;
        double secs = ((double)time) / 1000.0;
        System.out.println("Time = " + secs + " secs. Result = "+result);
    }
}
```