Pontificia Universidad Católica
Departamento de Ciencias de la Computación

Tarea 1: Sistemas Distribuidos

Profesor: Yadran Eterovic Solano

Ayudante: Orlando Vásquez Herrera

"MPI CONTEST"

Fecha de entrega: 15 de Septiembre 2014

Objetivos

En esta **tarea-concurso** deberás utilizar tus conocimientos de Programación Distribuida mediante Paso de Mensajes (MPI) para construir las versiones distribuidas de los algoritmos de ordenación descritos en las preguntas 1 y 2 y e implementar programas distribuidos que resuelvan los problemas descritos en las preguntas 3 y 4.

Herramientas a utilizar y restricciones

Para realizar Programación distribuida mediante Paso de Mensajes utilizarán *Python* con *mpi4py* un paquete utilizado para realizar comunicación entre procesos. **No se podrá utilizar ningún otro package, con excepción de numpy.**

Concurso

Para cada una de las preguntas que se detalla a continuación se describirá **la llamada al método**, el **input** que se te entregará, lo que se espera que haga el algoritmo y el **output.**

Pregunta 1: *Bucket Sort* (1 pto.)

Bucket Sort es un algoritmo de ordenación que asume que los n elementos a ser ordenados se encuentran en el intervalo (a, b).

El intervalo (a, b) es dividido en m sub-intervalos de igual tamaño, los cuales se denominan *buckets*. Cada elemento es ubicado en el *bucket* apropiado. Como los elementos son uniformemente distribuidos en el intervalo (a, b), el número de elementos en cada *bucket* es aproximadamente n/m. Luego el algoritmo ordena los elementos en cada bucket.

La llamada al método será:

$$\underline{\boldsymbol{bucket\ sort}(elements, \#process)}$$

donde $elements$ es la lista de números que se desea ordenar y $\#process$ es el número de procesos que ordenarán (**int**).

El output que se espera es una lista con los mismos valores del input pero ordenados. Considera el índice 0 del arreglo como el de menor valor.

Cada uno de los procesos debe recibir una parte del arreglo a ordenar, los procesos deben intercambiar los números entre ellos de modo que cada uno de ellos obtenga los elementos de su

bucket. Finalmente cada proceso debe ordenar los elementos que posee. El arreglo ordenado corresponderá a la concatenación "adecuada" de los arreglos de cada proceso.

Pregunta 2: *Sample Sort* (1 pto.)

Sample Sort es un algoritmo que implementa una mejora sobre Bucket Sort. La idea básica de este algoritmo es la siguiente:

A partir de la secuencia de n elementos que se desea ordenar se selecciona una muestra de tamaño s y el rango de los *buckets* es determinado ordenando la muestra y escogiendo $m-1$ elementos de los resultados. Estos elementos (llamados *separadores*) dividen la muestra en m *buckets*. Luego de haber definido los *buckets* el algoritmo se comporta de la misma manera en que ordena *Bucket Sort.*

La llamada al método será:

$$\boldsymbol{sample\_sort}(elements, \#process, s, m)$$

donde $elements$ es la lista que se desea ordenar y $\#process$ es el número de procesos que ordenarán (**int**).

El output que se espera es una lista con los mismos valores del input ordenados, considere el índice 0 del arreglo como el de menor valor.

Para este algoritmo primero debes seleccionar la muestra de s elementos, y ordenarlos de manera distribuida. A continuación seleccionan $m-1$ elementos y realizan la misma ordenación descrita en la pregunta 1.

Los siguientes dos problemas requerirán investigación de parte de Uds.

Pregunta 3: *Coloración de un grafo sparse* (2 ptos.)

El problema consiste en dado un grafo $G=(V,E)$ no dirigido y conexo, cuyos nodos tienen pocas conexiones, encontrar la coloración mínima posible, es decir, utilizar el menor número de colores) de tal manera que dos nodos que estén comunicados por una arista tengan distinto color.

La llamada al método será:

$$\boldsymbol{sparse\_graph\_coloring}(vector, \#process)$$

donde $vector$ corresponde al vector de adyacencia (del tipo ndarray), el cual solo posee valores booleanos, $vector[s]=1$ si el nodo que representa al proceso esta conectado con s, 0 en el caso contrario (esta matrix es simétrica ya que el grafo es no dirigido) y $\#process$ es el número de procesos que se utilizarán. Para este caso $\#process$ será igual al número de nodos

El **output** corresponderá a una tupla (x,y) donde x es el número de colores utilizados e y es una lista que representará la coloración. Esta lista tendrá largo el número de nodos del grafo y la posición i será el color considerando valores enteros desde 0 a $x-1$.

Para este algoritmo se espera que el cálculo de número máximo de colores a utilizar y la coloración se realicen de manera distribuida. Para resolver este problema se espera los procesos se comuniquen con sus procesos vecinos para encontrar la coloración.

Pregunta 4*: Camino más corto* (2 ptos.)

Debes desarrollar un programa MPI que calcule el camino más corto desde un vértice fuente s a todos los otros vértices en el grafo $G = (V, E)$ dirigido usando p procesadores.

La llamada al método será:

$$\boldsymbol{shortest\_paths}(vector, s, \#process)$$

Donde $vector$es el vector de adyacencia (del tipo matrix de *numpy*), la cual solo posee valores enteros no negativos, los cuales representan la distancia entre los nodos. Si $vector[r] = 0$ el nodo representado por el proceso y r no están conectados, s es el índice del nodo fuente y $\#process$ es el número de procesos que se utilizarán (**int**). Para este caso $\#process$ será igual al número de nodos

El **output** corresponderá a una lista que en la posición i tendrá la distancia más corta entre el nodo fuente s e i.

Para esta pregunta se espera que encuentres la distancia a cada nodo desde el nodo source de manera distribuida. La idea del algoritmo es que las distancias mínimas pueden ser encontradas de manera expansiva en el grafo, es decir, encontrar aquellos que son vecinos del nodo source, luego los vecinos de los vecinos y así sucesivamente hasta que se han calculado las distancias mínimas hacia todos los nodos.

Entregable

Lo que deberán entregar el día 15 de Septiembre del 2014 será un archivo con extensión $.py$ (utilizado por Python) con nombre $[TuApellido]\_[TuNombre]\_Tarea1.py$ con una clase llamada $[1era\ letra\ de\ Tu\ Nombre]\_[Tu\ Apellido]$ que implemente los 4 programas requeridos (**recordar que el primer argumento de los métodos de una clase es self**) siguiendo las pautas ya señaladas. Pueden crear más métodos dentro de la clase.

Para la entrega se habilitará un buzón de tareas en SIDING. En caso de atraso pueden enviar la tarea al mail **iic2523sd22014@gmail.com** pudiendo optar principalmente a nota máxima 4.0 sin posibilidad de participar en el concurso.

Evaluación

La evaluación de la tarea se compone de 2 partes:

- La mitad del puntaje de cada pregunta (detallado ya anteriormente) corresponderá a la implementación y respuesta del programa, es decir, si se implementa el programa/algoritmo y la soluciones/respuestas que entrega son correctas, se obtendrá la mitad del puntaje de la pregunta.

Si el algoritmo/programa no retorna la respuesta correcta el puntaje para la pregunta i (P_i) se encontrara en el intervalo $\left[0, \frac{PT_i}{2}\right[$.

- La segunda parte medirá el performance del programa/algoritmo. Para esto se realizará una competencia entre aquellos que hayan obtenido la mitad de puntaje destinado a la correctitud del algoritmo.
  El puntaje para la pregunta i (P_i) para aquellos que pasen a etapa estará dada por la siguiente formula:

$$P_i = PT_i\left(\frac{1}{2} + \frac{(n_i - x)}{2n_i}\right)$$

Donde:
PT_i : Puntaje total de la pregunta i, como se mencionó anteriormente las pregunta 1 y 2 tienen 1 punto cada una y 3 y 4 2 puntos cada una.
n_i: Es el número de preguntas i que lograron pasar a la segunda etapa
x: Es la posición del alumno en el ranking de la pregunta i. El ranking va de 0 a $n_i - 1$ donde la posición 0 indica aquella solución/algoritmo con el mejor performance.

Considerando los puntos anteriores la nota de la Tarea 1 se calculará como sigue:

$$NT1 = 1 + \sum P_i$$

El ganador del concurso será aquel que mayor nota tenga en la tarea.

Recompensas

De modo que te concentres en realizar tu tarea de la mejor manera posible y obtener buenos resultados la persona que gane el concurso obtendrá un premio.

El premio aún no está definido del todo pero lo más probable es que corresponda a una de las siguientes opciones:

- 100 dólares en Amazon
- 1 libro

Recorreciones

La fecha y el horario destinada a recorregir las tareas se definirá una vez se hayan publicado los resultados de las competencias y se realizará de manera presencial. Por temas de respetar los resultados de la competencia podrán pasar por esta etapa solo aquellas preguntas que hayan obtenido un puntaje $< \frac{PT_i}{2}$. No obstante, el puntaje máximo posible de obtener en esta instancia corresponderá a $\frac{PT_i}{2}$. La adición, modificación y eliminación de cualquier línea de código conllevará la penalización de 0.1 ptos por línea, considere esto cuando desee recorregir.

Integridad Académica

Como toda tarea de este departamento, está sujeta a la política de integridad académica de esta universidad por lo cual cualquier conducta impropia detectada será sancionada.