3rd Marathon of Parallel Programming
SBAC'08

For all the problems, the input data will be available in a file called **problem_input** and the output should be written to a file named **problem_output**. For all problems, a sequential implementation is given, and it is against the output of those implementations that the output of your programs will be compared to decide if your implementation is correct. You can modify the program in any way you see fit, except when the problem description states otherwise. You must submit your source code. You can submit a compiling and or an execution script too. The program to execute should have the name of the problem. You can submit as many solutions to a problem as you want. Only the last submission will be considered.

All teams will be given access to the target machine during the marathon. From time to time, such access will be interrupted and the organization will measure the submissions received, and partial results will be published.

The execution time of your program will be measured running it with **time program** and taking the user CPU time given. Each program will be executed at least twice with the same input and only the smaller time will be taken into account. The sequential program given will be measured the same way. You will earn points in each problem, corresponding to the division of the sequential time by the time of your program. The team with the most points at the end of the marathon will be declared the winner.

1 Median of a set

The median is described as the number separating the higher half of a sample, a population, or a probability distribution, from the lower half. The median of a finite list of numbers can be found by arranging all the observations from lowest to highest value and picking the middle one. If there is an even number of observations, the median is the mean of the two middle values.

For example:

1,2,3,4,5 : The median is 3.
1,5,9,2,8,4: The median is 4.5.

Compute the median of a given set of values.

2 Prefix sum

A prefix sum p_i from a set $S = 1, 2, 3, ..., n$ is such as $p_i = 1 + 2 + ... + i$, where $i \leq n$. Calculate all the prefix sums from $1 \leq i \leq n$.

Hint: using a tree to pre-process the result values might be helpful.

3 The Sieve of Eratosthenes

The Sieve of Eratosthenes is an algorithm to find all the prime numbers up to a specified number. To do that, it starts from the first unmarked number k and marks all numbers after (and including) k^2 that are divisible by k. It does the same to all unmarked numbers in the list so that, at the end, all the unmarked numbers are prime.

Make a parallel version for this algorithm.

4 Loyd's tile puzzle

Loyd's tile puzzle is a sliding puzzle that consists of a grid of numbered squares with one square missing, and the labels on the squares jumbled up. If the grid is 3×3, the puzzle is called the 8-puzzle. If the grid is 4×4, the puzzle is called the 15-puzzle. The goal of the puzzle is to unjumble the squares by only making moves which slide squares into the empty space, in turn revealing another empty space in the position of the moved piece.

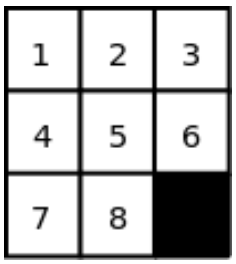

Figura 1: A solved 8-puzzle.

One way to solve this puzzle is to use a branch-and-bound algorithm. In it we create a decision tree with the possible movements in a given moment and decide to go further in one branch using some utility function.

In this case the utility function choses the branch that has a lower manhattan distance value and a lower height in the tree (i. e., has used less movements so far). The manhattan distance is given by the number of "hops" needed for a tile to get to it's final position using only vertical and horizontal movements.

Make a parallel version of this branch-and-bound algorithm to solve a $n - puzzle$.

5 Binary Heaps and Priority Queues

Heaps are a common basis for constructing priority queues and thus are used by many algorithms and in many contexts. Some algorithms, such as Dijkstra Shortest Path, depend upon specialized heaps with tight complexities for insertion, update and element removal.

Binary heaps are the most common implementation of heaps. While it is perfectly acceptable to implement a binary heap using a traditional binary tree structure, it is more commonly implemented using a plain array. If constructed this way, the heap property

(that states that every node is smaller than or equal to its children node), can be formally state as for a heap H with n elements as:

$$\forall i, 1 \leq i < n, H[i] \leq min\{H[2i], H[2i+1]\} \tag{1}$$

Being a building block for many algorithms, a parallel implamentation of a binary heap can speed up the execution of many well known serial and parallel algorithms.

Turn a serial implementation of a heap (functions `PopFromHeap()` and `AddToHeap()`) in a parallel one. Your input is discribed in the comments of the `HandleHeapInput()` function. The result of you program should be the list of elements removed from the heap during `HandleHeapInput()` execution.

6 PageRank

PageRank is a well-known link-analysis algorithm used mostly on Web search engines. Intuitivelly, it sees each link from a given page u to another page v as a vote from u in v. The more votes a given page receives, greater is its importance and thus greater is the value of its votes.

Given a webpage u. Let B_u be te set of pages pointing (having links) to u and N_v the number of links departing (outlinks) from u to other web pages. Formally, u's PageRank, $PR(u)$ is is defined as follows:

$$PR(u) = (1-d) + d \sum_{v \in B_u} \frac{PR(v)}{N_v} \tag{2}$$

While this is a recursive formula, the PageRank value for millions of pages can be eficiently calculated in matter of hours using a iterative algorithm. One is provided in serial form for you. Although it can some iterations of the algorithm the calculated aproximate values to converge, they do converge. One form of stablishing that the aproximate values have converged is calculating the L1-Norm of the difference between two consecutive iteractions, defined as follow:

$$\sum_{u \in Pages} |PR_{old}(u) - PR_{new}(u)| \tag{3}$$

If the calculated norm is smaller then a given threshold, defined as `PAGERANK_RESIDUAL_LIMIT` in the sequential implementation, the algorithm has converged.

Your objective is to calculate the PageRank value of a series of pages whose linking information is given as input. The L1-Norm of the difference between your result and the output of the sequential algorithm should be smaller than `PAGERANK_RESIDUAL_LIMIT`.

7 Finding the roots of a forest

Let F be a forest consisting of a set of rooted directed trees. The forest F is specified by an array P of length n such that $P(i) = j$ if (i, j) is an arc in F; that is, j is the parent of i in a tree of F. For simplicity, if i is a root, we set $P(i) = i$. *The problem is to determine the root* $S(j)$ *of the tree containing he node* j *for each* j *between* 0 *and* $n-1$.

Your input will consist of P and the expected output is S's contents.

8 Clustering Coefficient for a Graph

The clustering coefficient of a vertex in a graph quantifies how close the vertex and its neighbors are to being a clique (complete graph). Duncan J. Watts and Steven Strogatz introduced the measure in 1998 to determine whether a graph is a small-world network.

Let $G = V, R$ be a graph such with a set of vertices V and a set of edges E between them. An edge $e_{i,j}$ connects vertex i with vertex j.

The neighbourhood N for a vertex v_i is defined as its immediately connected neighbours as follows:

$$N_i = \{v_j : e_{ij} \in E \vee e_{ji} \in E\} \tag{4}$$

The clustering coefficient C_i for a vertex v_i is then given by the proportion of links between the vertices within its neighbourhood divided by the number of links that could possibly exist between them. For a directed graph, e_{ij} is distinct from e_{ji}, and therefore for each neighbourhood N_i there are $k_i(k_i - 1)$ links that could exist among the vertices within the neighbourhood (k_i is the total (in + out) degree of the vertex). Thus, the *clustering coefficient for directed graphs* is given as

$$C_i = \frac{|\{e_{jk}\}|}{k_i(k_i - 1)} : v_j, v_k \in N_i, e_{jk} \in E. \tag{5}$$

An undirected graph has the property that e_{ij} and e_{ji} are considered identical. Therefore, if a vertex v_i has k_i neighbours, $\frac{k_i(k_i-1)}{2}$ edges could exist among the vertices within the neighbourhood. Thus, the "'clustering coefficient for undirected graphs"' can be defined as

$$C_i = \frac{2|\{e_{jk}\}|}{k_i(k_i - 1)} : v_j, v_k \in N_i, e_{jk} \in E. \tag{6}$$

The *clustering coefficient for the whole system* is given by Watts and Strogatz as the average of the clustering coefficient for each vertex:

$$\bar{C} = \frac{1}{n} \sum_{i=1}^{n} C_i. \tag{7}$$

Your problem is to calculate the average clustering coefficient for a undirected graph. Your input will consist in the number of nodes in this graph, a list of its edges along with their weights. Your sole output should be the average clustering coefficient for this undirected graph.