CS600 WebCampusAdvanced Algorithm Design and Implementation

Homework 7

1.
Solution:
The input number is n(number of vertices), m(number of edges),w(the interval).
When the input number n is settled, the range of m is [n-1, n*(n-1)]. Because if we generate graph from a series of random non-cycle paths. If a path has 1 edge in all, and it points to itself, then it becomes a cyclic. So I limit the m range to [n-1, n*(n-1)]. In order to avoid the overflow I restrict the interval to be no larger than INT_MAX/m.
I use the permutation function to generate random paths. And my function will generate m/(n-1) random paths each of which has (n-1) edges, using the permutation function, and then generate random weight for each edge. But if I found that a edge's weight already exists, I do not generate the weight again and use the existent value, record it instead. After I generated the paths using permutation, if the total number of edges do not reach m, I will randomly generate the remain edges, take them as random paths that have 1 edge each.
See code in random_graph()

2.
Solution:
I modified the DFS to determine whether a specific edge completes a negative-weight cycle.
int graph::check_circle_for_e(int** m, int v,int i,int j)
I use a array cir_recorder[n] to store data so as to output a cycle.
If we want to know if edge(u,v) completes a negative-weight circle in graph G, the adjacent matrix of G is m. And use a nude structure to record the 'color' of each node. Record vertice u in cir_recorder[0], set its color to be grey. DFS visit from vertice v, and store new vertice into cir_recorder by judging if its color is white. When we meet a grey color vertice, it means a circle generates. Go back through the cir_recorder to see from which place the circle start, output the circle, and compute the weight. When each DFS is over, reset the vertice to

be white, so that, it could be reached by other vertices later. The function will discover all the cycle for (u,v) and return the result. There three possible result: no cycle, a positive-weight and a negative cycle.

3.
Solution:
int graph::bellman_ford(int** m, int s, int n)
The function will check an adjacent matrix of a graph with n vertices, to see whether a negative-weight cycle exist from the specific vertices s.
About the running time:
Ini_single_source will take running time: n
The process of all the relax will have running time: (n-1)*n*n, and do relax (n-1)*m times
The process of computing will have worst running time: n*n and do the comparison m times.
The run time for the code using adjacent matrix will be:
$n+(n-1)*n^2+n^2$
So is: $O(n^3)$

4.
Solution:
floyd_warshall(int ** m, int n)
The function will produce the shortest weight matrix d and the predecessor matrix p using the adjacent matrix of a graph with n vertices. It will only output the final matrixes d and p.

Parellel sorting
5
Solution:
In parel folder, main.cc

6
Solution:

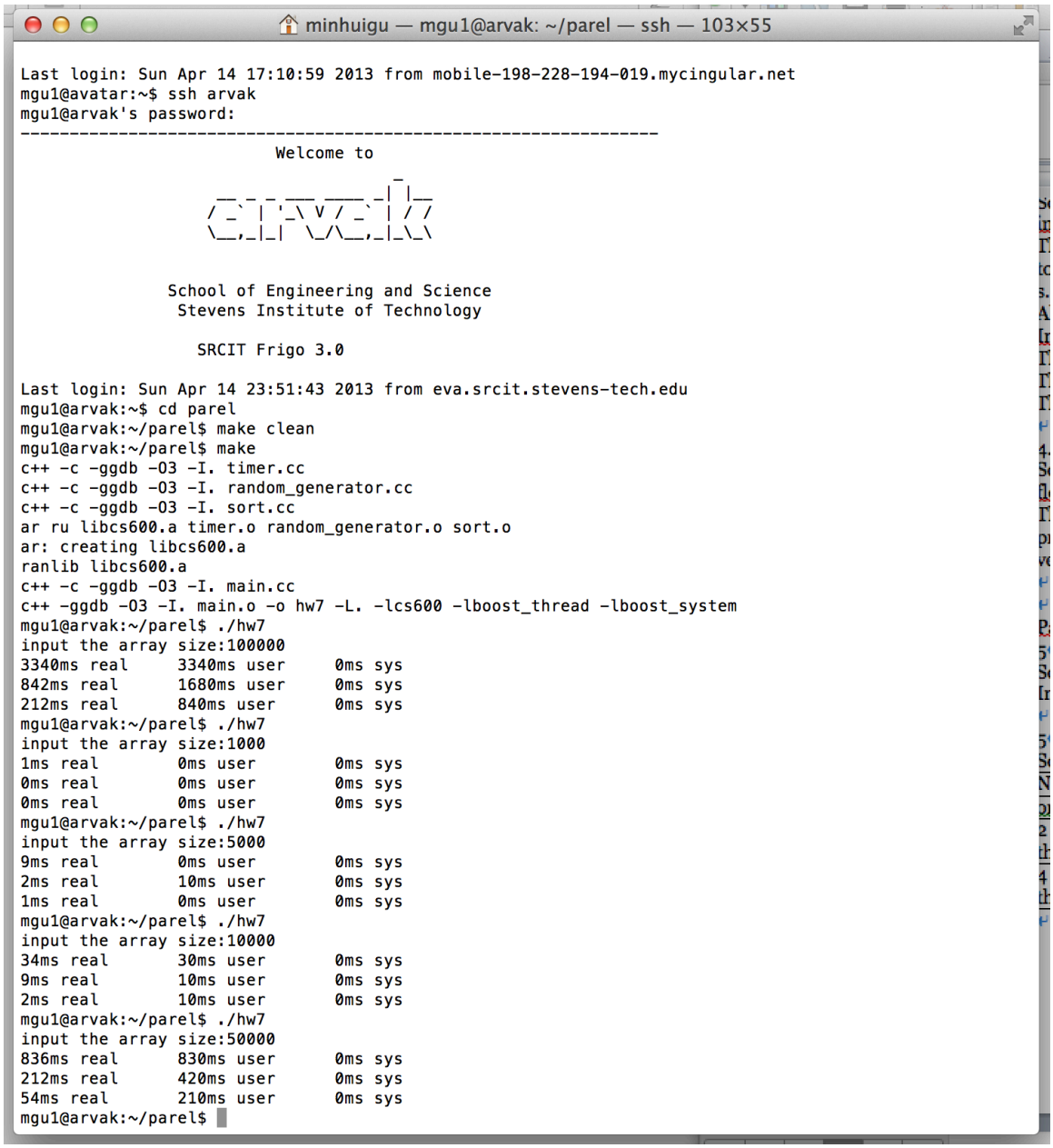
N=	1000	5000	10000	50000	100000	1000000
Original	1	9	34	836	3340	335347
2 threads	0	2	9	212	842	83769

4 threads	0	1	2	54	212	20980

Insertion sort: $O(n^2)$
We suppose the following analysis use the same n:
In my modified 2 thread sort, two thread work on the array simultaneously. Each thread: $O((n/2)^2)$
In my modified 4 thread sort, for thread work on the array simultaneously. Each thread: $O((n/4)^2)$

Modified 2 thread insertion-based merge sort:
$O(n^2/4)+O(n)$.
Modified 2 thread insertion-based merge sort:
$O(n^2/16)+2*O(n/2)+O(n)$.

It means, for a large n, O(Original insertion) =
4*O(modified_2_thread) = 16*O(modified_4_thread)


minhuigu — mgu1@arvak: ~/parel — ssh — 103×55
Last login: Sun Apr 14 17:10:59 2013 from mobile-198-228-194-019.mycingular.net
mgu1@avatar:~$ ssh arvak
mgu1@arvak's password:
Welcome to
School of Engineering and Science
Stevens Institute of Technology
SRCIT Frigo 3.0
Last login: Sun Apr 14 23:51:43 2013 from eva.srcit.stevens-tech.edu
mgu1@arvak:~$ cd parel
mgu1@arvak:~/parel$ make clean
mgu1@arvak:~/parel$ make
c++ -c -ggdb -O3 -I. timer.cc
c++ -c -ggdb -O3 -I. random_generator.cc
c++ -c -ggdb -O3 -I. sort.cc
ar ru libcs600.a timer.o random_generator.o sort.o
ar: creating libcs600.a
ranlib libcs600.a
c++ -c -ggdb -O3 -I. main.cc
c++ -ggdb -O3 -I. main.o -o hw7 -L. -lcs600 -lboost_thread -lboost_system
mgu1@arvak:~/parel$ ./hw7
input the array size:100000
3340ms real 3340ms user 0ms sys
842ms real 1680ms user 0ms sys
212ms real 840ms user 0ms sys
mgu1@arvak:~/parel$ ./hw7
input the array size:1000
1ms real 0ms user 0ms sys
0ms real 0ms user 0ms sys
0ms real 0ms user 0ms sys
mgu1@arvak:~/parel$ ./hw7
input the array size:5000
9ms real 0ms user 0ms sys
2ms real 10ms user 0ms sys
1ms real 0ms user 0ms sys
mgu1@arvak:~/parel$ ./hw7
input the array size:10000
34ms real 30ms user 0ms sys
9ms real 10ms user 0ms sys
2ms real 10ms user 0ms sys
mgu1@arvak:~/parel$ ./hw7
input the array size:50000
836ms real 830ms user 0ms sys
212ms real 420ms user 0ms sys
54ms real 210ms user 0ms sys
mgu1@arvak:~/parel$