CS6135 VLSI Physical Design Automation

Homework 2: Two-way Min-cut Partitioning

Due: 23:59, October 29, 2019

1. Introduction and Goal

Let *C* be a set of cells and *N* be a set of nets. Each net connects a subset of cells. The two-way min-cut partitioning problem is to partition the cell set into two groups *A* and *B*. The cost of a two-way partitioning is measured by the cut size, which is the number of nets having cells in both groups. In this homework assignment, you are asked to implement **FM ALGORITHM** to solve the problem of two-way min-cut partitioning. Note that any form of plagiarism is strictly prohibited. If you have any problem, please contact TA.

2. Problem Description

The two-way min-cut partitioning problem is defined as follows:

1. Input:

- ✓ A netlist for a circuit (.nets)
- ✓ The size of each cell (.cells)

2. Objective: Partition the circuit in two sub-circuits *A* and *B*, such that the cut size

is minimized under the constraint of $|area(A) - area(B)| < \frac{n}{10}$, where $area(A)$

is the sum of all cell sizes in *A*, $area(B)$ is the sum of all cell sizes in *B*, and *n* is the sum of all cell sizes in the circuit.

3. Input

1. The *.cells* file

This input file specifies a list of cells. Each cell statement starts with its cell name (unordered) and size (a positive integer).

2. The *.nets* file

This input file specifies a list of nets. Each net statement starts with the keyword "NET" and the name of the net. The cells that are connected by the net are listed between a pair of braces following the net name.

Example:

.cells	.nets
c 2 1	NET n1 { c2 c3 c4 }
c 3 2	NET n2 { c3 c7 }
c 4 1	NET n3 { c3 c5 c7 }
c 7 2	NET n4 { c1 c3 c5 c7 }
c 5 1	NET n5 { c2 c4 c8 }
c 1 1	NET n6 { c4 c6 }
c 8 2	NET n7 { c2 c6 c8 }
c 6 2	
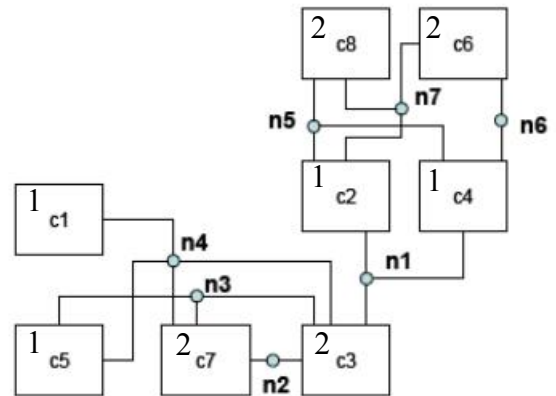
P.S. We will give a testcase generator coded by ruby. It's applicable to ruby v1.9.3 and beyond.

```
$ ruby genTestcase.rb
Usage: ruby genTestcase.rb [filename] [#cells] [maxCellSize] [#nets] [maxDegree]
========== Start Generating ==========
filename: test
#cells: 48579
maxCellSize: 29
#nets: 97943
#maxDegree: 27
==========  End Generating  ==========
```

```
$ ruby genTestcase.rb test2 10000 10 10000 10
========== Start Generating ==========
filename: test2
#cells: 10000
maxCellSize: 10
#nets: 10000
#maxDegree: 10
==========  End Generating  ==========
```

4. Output

Report the cells in each group and the cut size. **Please strictly follow the output format or your result will be treated as illegal.** You can run the "verifier" to check whether your result is legal or not.
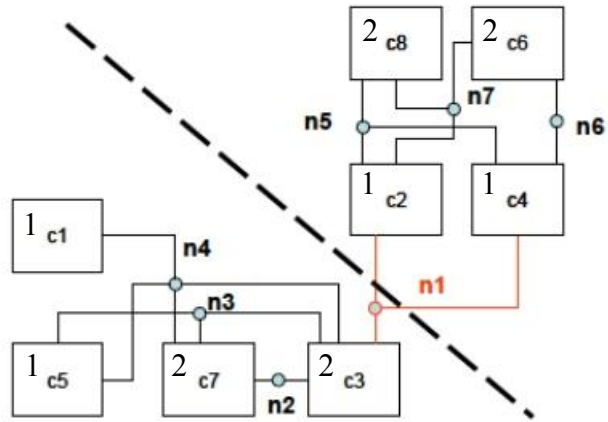
Output Format:

```
cut_size #
A #Cell_in_groupA
cell_ID
…
B #Cell_in_groupB
cell_ID
```

2

...

...

Example:

```
cut_size 1
A 4
c1
c3
c5
c7
B 4
c2
c4
c6
c8
```


5. Language/Platform
- Language: C/C++
- Platform: Unix/Linux

6. Report
Your report should contain the following content, and you can add more as you wish. The more the better.

(1) Your name and student ID

(2) How to compile and execute your program, and give an execution example. If you implement parallelization, please let me know how to execute it with single thread.

(3) The final cut size and the runtime of each testcase

P.S. You could use the command `time` to measure runtime.

e.g., `$ /usr/bin/time -p ./main`

`$ echo "alias time '/usr/bin/time -p'" >> ~/.tcshrc`

Re-log in then `$ time ./main` is equal to `$ /usr/bin/time -p ./main`

(4) Runtime = $T_{IO} + T_{computation}$. For each case, please analyze your runtime and find out how much time you spent on I/O and how much time you spent on the computation (FM Algorithm).

(5) The details of your implementation containing explanations of the following questions:

    I.   Where is the difference between your algorithm and FM Algorithm described in class? Are they exactly the same?

II. Did you implement the bucket list data structure?
   ✓ If so, is it exactly the same as that described in the slide? How many are they?
   ✓ If not, why? You had a better data structure? Or, is bucket list useless?
III. How did you find the maximum partial sum and restore the result?
IV. Please compare your results with the top 5 students' results from last year and show your advantage either in runtime or in solution quality. Are your results better than them?
   ✓ If so, please express your advantages to beat them.
   ✓ If not, it's fine. If your program is too slow, then what do you reckon the bottleneck of your program is? If your solution quality is inferior, what do you think that you could do to improve the result in the future?
V. What else did you do to enhance your solution quality or to speed up your program?
VI. What have you learned from this homework? What problem(s) have you encountered in this homework?
VII. If you implement parallelization, please describe the implementation details and provide some experimental results.
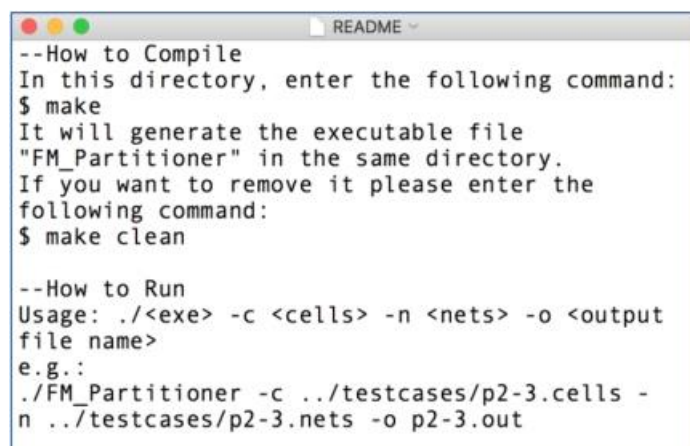
7. Required Items

Please compress `HW2/` (using `tar`) into one with the name `CS6135_HW2_{StudentID}.tar.gz` before uploading to iLMS.

(1) `src/` contains all your sources code, your `Makefile` and `README`.
   ➢ `README` must contain how to compile and execute your program.
     An example of `README` is like the following figure:

```
--How to Compile
In this directory, enter the following command:
$ make
It will generate the executable file
"FM_Partitioner" in the same directory.
If you want to remove it please enter the
following command:
$ make clean

--How to Run
Usage: ./<exe> -c <cells> -n <nets> -o <output
file name>
e.g.:
./FM_Partitioner -c ../testcases/p2-3.cells -
n ../testcases/p2-3.nets -o p2-3.out
```
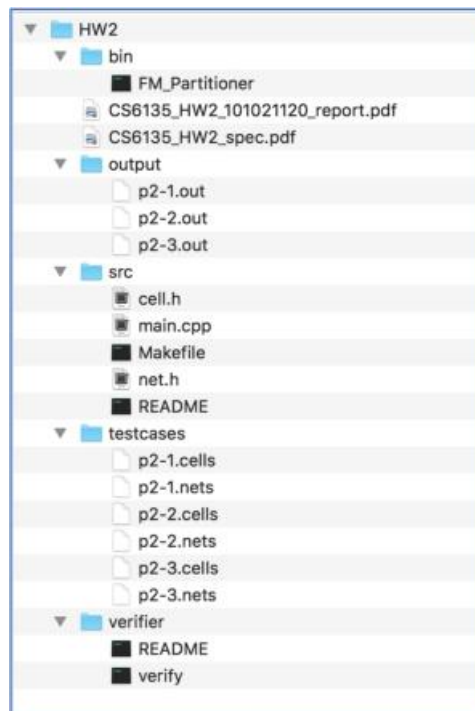
(2) `output/` contains all your outputs of testcases for comparison.
(3) `bin/` contains your compiled executable file.
(4) `CS6135_HW2_{STUDENT_ID}_report.pdf` contains your report.

The file structure would be like the following figure:

You can use the following command to compress your directory on a workstation:

```
$ tar -zcvf CS6135_HW2_{StudentID}.tar.gz directory
```

For example:

```
$ tar -zcvf CS6135_HW2_108062501.tar.gz HW2/
```

8. Grading

✓ 80%: The solution quality (final cut size) and the runtime of each testcase, hidden
   testcases included. Note that this part will be executed with single thread version.

✓ 20%: The completeness of your report

✓ 5% (bonus): Parallelization.

P.S. Using C++11

The C++11 standard is implemented in GCC 4.8.1 and beyond, yet the default version
of gcc in our CAD servers (ic21~ic29) is 4.1.2.

You need to `source /tools/linux/gnu/setup_toolkit.csh` to link gcc to
gcc 4.8.5.

```
[vlsipda01@ic21 ~]$ g++ --version
g++ (GCC) 4.1.2 20080704 (Red Hat 4.1.2-55)
Copyright (C) 2006 Free Software Foundation, Inc.
本程式是自由軟體；請參看來源程式碼的版權宣告。本軟體沒有任
何擔保；
包括沒有適銷性和某一專用目的下的適用性擔保。
[vlsipda01@ic21 ~]$ source /tools/linux/gnu/setup_toolkit.c
sh
[vlsipda01@ic21 ~]$ g++ --version
g++ (GCC) 4.8.5
Copyright (C) 2015 Free Software Foundation, Inc.
本程式是自由軟體；請參看來源程式碼的版權宣告。本軟體沒有任
何擔保；
包括沒有適銷性和某一專用目的下的適用性擔保。
```

However, it's too much work and too forgettable that you need to source it every time when you log in on a server.

Instead, you can create a shell resource file called `.tcshrc` in the root folder and put the source command in it. Just enter the following command once, re-log in, and then it won't never bother you anymore.

```
$ echo "source /tools/linux/gnu/setup_toolkit.csh" >> ~/.tcshrc
```

P.S. hMETIS

If you have time, you can learn how to run hMETIS and compare your results with those generated by hMETIS.

(http://glaros.dtc.umn.edu/gkhome/metis/hmetis/overview)