# CS473/573 Algorithms I
# Assignment #1

Kernighan-Lin (KL) algorithm [1] is a famous heuristic algorithm for partitioning (dividing) the vertices of a graph into two balanced parts while minimizing the cutsize (the number of edges crossing between the two parts). The balance criteria is defined such that the number of vertices in each part differ at most by 1 (assuming unit-weight vertices). Read the original paper by Kernighan and Lin [1] in order to understand the algorithm. Consider the following pseudocode of a simplified KL:

---

**function** SIMPLER-KL($G(V, E)$)
  Split $V$ into two balanced disjoint sets $A$ and $B$
  **repeat** ▷ Each repeat loop corresponds to a pass
    Compute $D$ values (move gain values) for all $a \in A$ and $b \in B$
    ▷ Move gain values refers to the gain of moving vertex $a$ from one part to another part
    Let $L_g, L_a, L_b$ be empty lists
    $i \leftarrow 0$
    **for** $n = 1$ to $|V|/2$ **do** ▷ Assuming $n$ is even
      Find $a \in A$ and $b \in B$ such that the exchange gain $g = D_a + D_b$ is maximized.
      Remove $a$ and $b$ from further consideration in this pass through locking.
      $i \leftarrow i + 1$
      $L_g[i] \leftarrow g$; $L_a[i] \leftarrow a$; $L_b[i] \leftarrow b$
      Update $D$ values of affected elements in $A - \{a\}$ and $B - \{b\}$
    Find $k$ which maximizes $g_{max} = \sum_{j=1}^{k} L_g[j]$
    **if** $g_{max} > 0$ **then**
      Exchange $L_a[1], L_a[2]..L_a[k]$ with $L_b[1], L_b[2]..L_b[k]$
  **until** $g_{max} \leq 0$

---

Note that the difference between this simplified version and the original algorithm is the use of a more relaxed definition of exchange gain . That is, in the original KL definition the exchange gain is defined as $g = D_a + D_b - 2$ if there is an edge $(a, b) \in E$, whereas $g = D_a + D_b$ otherwise. In the simplified KL, however, we assume that $g = D_a + D_b$ for the sake of easier implementation.

You are asked to do the following tasks as individuals:

1. Implement the above simplified version of the KL algorithm in **C** considering the following two approaches for finding an exchange with maximum gain:

a) Maintain the vertices in the two parts as binary heaps and perform extract max operation on each heap at each iteration (extract vertices with maximum D values). What is the asymptotic worst-case running time of the algorithm?

b) Go over each part in order to find the maximum $D$ values at each iteration. What is the asymptotic worst-case running time of the algorithm?

Your program should take the name of a graph file as input, and prints three values as output (one line, space or tab separated): initial cutsize, final cutsize, and runtime for both a) and b) on the same line. The input graph file is stored as list of edges, where each line represents an edge as "SRC DST", where SRC and DST are receptively the source and destination vertex ids.

2. Compare your program against Python's `NetworkX` library implementation of KL in terms of final cutsize and runtime for the following graphs which can be obtained from SuitSparse Matrix Collection[1]:
   - `Erdos02`
   - `com-DBLP`
   - `rgg_n_2_20_s0`

   You may use either graphical plots or tables for the comparison. Which runs faster and why ?

3. Use GNU profiler[2] (gprof) to analyze the actual running time of your program and to see where most of the time is spent. Compare the asymptotic running times of the steps of your algorithm with the actual runtimes, observe any inefficiencies (for instance, if an $O(n)$-time operation takes much more time than an $O(n^2)$ operation) and try to improve the overall performance by focusing on improving the most time-consuming operations/steps .

4. Prepare a report of **maximum two pages** that contains:
   - a brief explanation of your implementation
   - discuss which of 1.a) or 1.b) is more efficient in case the maximum degree (max edges per vertex) of the input graph is limited to a constant $C$.
   - the comparison requested in task #3
   - your conclusions and improvements done to the code after profiling
   - and any other code-based/asymptotic improvements you have performed and would like to highlight.

## Submission rules and guidelines

- You will submit your code and report in one zipped file, called SURNAME_FIRSTNAME.zip, which contains the following:
  - Your source code files (.c and .h files).
  - Makefile to be used to compile and build your program.

[1] `https://sparse.tamu.edu/`
[2] `https://ftp.gnu.org/old-gnu/Manuals/gprof-2.9.1/html_node/gprof_toc.html`

- report.pdf
- [Optional] README.txt file that contains any (potential) technical issues.
- Nothing else!, no graph input files nor IDE-related configuration files.

by **December 7th, 2020 17:30** as an email to `nabil.abubaker@bilkent.edu.tr`. Every day of late submission will cost 25pts.

- Use GNU Make[3] tool to automate the process (you should be familiar with this from CS-201/202). Your executable is called KL (**not kl nor KL.exe**) and takes only one argument. Example compile and run:

```
> make                    #This should generate a binary executable file called KL
> ./KL input_graph.mtx    #runs the code with input graph file
> 1530 440 3.6s 1530 440 2.8s        #output (initial_cut final_cut runtime)
```

  Before submission, make sure that your code can be compiled on a Unix-like environment using `gcc`. If you use Windows OS, you can either use Cygwin or newly added windows subsystem for Linux (WSL) feature of Windows 10.

- The Matrix Market (.mtx) files has the following format for storing an undirected, unweighted graphs:

```
% some comment lines
50 50 230 %#Vertices #Vertices #Edges
1 3 %SRC DST
3 5
...
```

- The grading will be done as follows. A script will be used to compile and run your code automatically. If your code does not compile, you might loose a large portion of the grade up to 50%. Your program must give an output in the same format above, even if the code is incomplete you can print something like "`1530 440 3.6s -1 -1 -1`" to tell the grader there's an incomplete part of the code (part 1.b in this case). After the results are collected, the grader will open both the report and the code together to assess your efforts. Make sure the report contains useful information for easier interpretation of the code (e.g., "I use doubly-linked lists to store X for efficiently computing Y").

## FAQ

- *Q: Can I use C++ instead of C ?*
  A: No, only codes written in C are accepted .

- *Q: Can I use external libraries for heaps, lists ..etc?*
  A: No, libraries outside of the built-in C standard libraries are not allowed. The only exception is that you can use an external library to read Matrix Market format (.mtx) files, but those are not recommended.

[3] `https://www.gnu.org/software/make/`

- *Q: The Matrix Market format has the following header line* `#Vertices #Vertices #Edges`*, I got number of vertices and number of edges, but what is the second #vertices?*
  A: The Matrix Market format is used to store sparse matrices and stores the graph the graph as an adjacency matrix. Usually the first line is `#rows #columns #nonzeors`, but since in case of graphs `#rows = #columns=#vertices`.

## References


[1] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," in The Bell System Technical Journal, vol. 49, no. 2, pp. 291-307, Feb. 1970, doi: 10.1002/j.1538-7305.1970.tb01770.x.