Assignment 5 — Greedy Algorithms

Due: November 8th

A greedy algorithm is one that assumes it can repeatedly make locally optimal choices, never having to rethink them, yet always arriving at a globally optimal solution. When applicable, this approach can drastically reduce the amount of work required to solve a problem.

Deliverables:

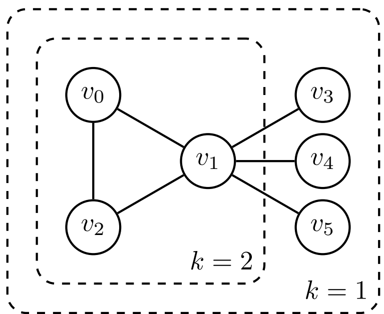
GitHub Classroom: `https://classroom.github.com/a/Mpz6zkQX`

Required Files: `compile.sh`, `run.sh`

Optional Files: `*.c`, `*.h`, `*.py`, `*.java`, `*.js`, `*.json`, `*.ts`, `*.clj`, `*.kt`, `*.jl`, `*.rs`

Part 1: k-Cores

A *k-core* is a maximal connected subgraph within which every vertex has degree at least k. They have applications in graph-based problems as a computationally inexpensive way of identifying the more densely connected regions of a graph. For example:



In the above graph, the vertices $\{v_0, v_1, v_2\}$ form the 2-core. Note that, although the graph contains a vertex v_1 of degree 5, there is no 5-core — there is no subgraph within which every vertex has degree 5. The vertex v_3 has degree 1, so it cannot be in a 5-core, and once v_3 is removed, v_1 only has degree 4.

In your programming language of choice (see Assignment 1), design and implement a greedy algorithm to find k-cores. To help you get started, note the following observations:

- A graph's k-cores form a nested hierarchy: any vertex in a k-core must also be in a $(k-1)$-core. In other words, a k-core is the result of removing[1] zero or more vertices from a $(k-1)$-core.
- If a vertex must be removed to create a core, then it also cannot be in any higher cores: if its degree is less than k, such that it is not in a k-core, then it cannot possibly be in a $(k+1)$-core.

Think carefully about what greedy heuristic your algorithm will employ and what data structures it will use[2]. Given a graph $G = (V, E)$, you should be able to find[3] *all* of its k-cores with complexity $O(|V| + |E|)$.

Each input graph will be provided as an edge list: each edge in the graph will be represented by a comma-separated pair of vertex identifiers, indicating an edge between the first vertex and the second. You may assume that vertex identifiers are contiguous natural numbers — they begin at 0, and there will be no "gaps" in the identifiers used. You may further assume that the graph will be simple, but it may be disconnected.

[1] This is called *pruning*: greedily removing elements that cannot possibly be part of some desired solution.
[2] See also `https://en.wikipedia.org/wiki/Bucket_queue`
[3] Formatting and printing the found k-cores will likely take slightly longer.

For example, the above graph could be represented as:

```
0, 1
0, 2
1, 2
1, 3
1, 4
1, 5
```

Your program must accept as a command line argument the name of a file containing an edge list as described above, then print to `stdout` the k-cores according to the following format:

- Every non-empty k-core must be printed, starting with $k = 1$. For each value of k, all vertices that appear in any k-core should be grouped together (even if they would be in different subgraphs).
- Each grouping of k-cores must appear as a single comma-separated line of vertices. The vertices must be sorted in ascending order: vertices with lower identifiers must appear first.

For example:

```
>$ ./compile.sh
>$ ./run.sh in1.txt
Vertices in 1-cores:
0, 1, 2, 3, 4, 5
Vertices in 2-cores:
0, 1, 2
```

You may further assume that, since the graph is simple, the maximum degree of any vertex (and, thus, the maximum possible value of k) is $|V| - 1$. Your program will be tested using `diff`, so its printed output must match *exactly*.

Part 2: Greedy Stays Ahead

Note that, in this problem, the quantity being optimized is the size of each k-core: by definition, a k-core must be maximal. To support the optimality of your greedy approach, prove the following lemma:

> *Lemma:* Suppose your algorithm returns the k-cores $X = (x_1, x_2, \ldots, x_{n-1})$ and there exist optimal k-cores $\textsc{Opt} = (y_1, y_2, \ldots, y_{n-1})$, where x_i, y_i are sets of vertices in i-cores. For all $i < n$, $y_i \subseteq x_i$.

That is, given the above graph, the optimal k-cores are $\textsc{Opt} = (\{v_0, v_1, v_2, v_3, v_4, v_5\}, \{v_0, v_1, v_2\}, \emptyset, \emptyset, \emptyset)$, in order from the 1-core to the 5-core.

Part 3: Submission

The following files are required and must be pushed to your GitHub Classroom repository by the deadline:

- `report.pdf` — Pseudocode for an efficient, greedy algorithm to find k-cores, along with proof of the lemma given in Part 2 and analysis of complexity for the pseudocode.
- `compile.sh` — A Bash script to compile your submission (even if it does nothing), as specified.
- `run.sh` — A Bash script to run your submission, as specified.

The following files are optional:

- `*.c`, `*.h`, `*.py`, `*.java`, `*.js`, `*.json`, `*.ts`, `*.clj`, `*.kt`, `*.jl`, `*.rs` — The source code of a working program to find k-cores, as specified.

Any files other than these will be ignored.