Assignment 8 — Approximations

Due: December 13th

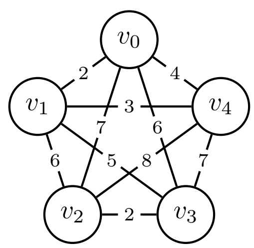
There are numerous problems in computer science that simply cannot be solved efficiently, assuming[1] that $\mathcal{P} \neq \mathcal{NP}$. Since those problems often have real-world applications, it is worthwhile to consider approximating their solutions, effectively trading some amount of correctness for complexity.

Deliverables:

GitHub Classroom:	`https://classroom.github.com/a/sKzMpHhe`
Required Files:	`report.pdf`, `compile.sh`, `run.sh`
Optional Files:	`*.c`, `*.h`, `*.py`, `*.java`, `*.js`, `*.json`, `*.ts`, `*.clj`, `*.kt`, `*.jl`, `*.rs`

Part 1: The Traveling Salesperson Problem

The Traveling Salesperson Problem, or "TSP", asks for the Hamiltonian cycle of minimum weight in a complete, weighted graph. It is one of the most famously difficult problems in computer science and graph theory[2]. In order to make approximation feasible, we restrict the problem to its *metric* case: all edge weights must satisfy the triangle inequality. For example:

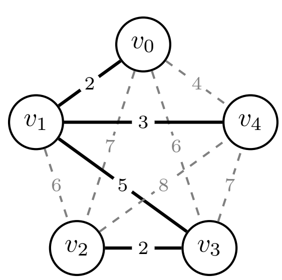


In the above complete, weighted graph, the Hamiltonian cycle of minimum weight is $(v_0, v_1, v_3, v_2, v_4, v_0)$.

Part 2: Approximating Metric TSP

Once restricted to the metric case, there exists a polynomial time approximation algorithm. Note that removing an edge from a Hamiltonian cycle must yield a spanning tree. An MST thus represents a somewhat similar structure to a solution to TSP — but one that is much easier to find. These observations inform a 2-approximation, as follows:

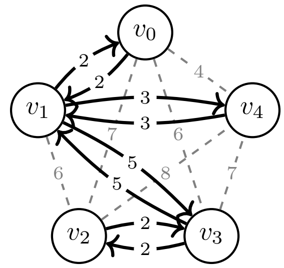
1. Construct an MST, which can be done greedily and requires linearithmic time. In the graph above, there is only one such tree, containing the edges $\{(v_0, v_1), (v_2, v_3), (v_1, v_4), (v_1, v_3)\}$:



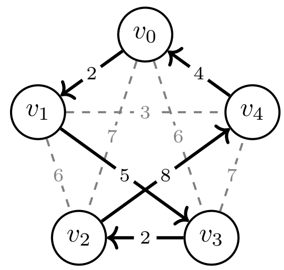
[1]And it certainly appears to be a safe assumption.

[2]Not only is TSP itself $\mathcal{NP}$-Hard, but merely approximating the general case of TSP is also $\mathcal{NP}$-Hard.

2. Certainly, a tree is not a cycle. Explore this tree in depth-first fashion, effectively traversing each edge twice, which requires linear time. From above[3], we have $(v_0, v_1, v_3, v_2, v_3, v_1, v_4, v_1, v_0)$:



3. This cycle is neither Hamiltonian nor valid in the given graph. Bypass any previously explored vertices[4], which requires linear time. In this case, this gives $(v_0, v_1, v_3, v_2, v_4, v_0)$:



In your programming language of choice (see Assignment 1), implement this polynomial time 2-approximation for the metric case of the Traveling Salesperson Problem.

Each input graph will be provided as an edge list: each edge in the graph will be represented by a comma-separated triple consisting of two vertex identifiers and a weight, indicating an edge between the first vertex and the second of that weight. 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 complete, with at least 3 vertices and natural weights satisfying the triangle inequality.

For example, the above graph could be represented as:

```
0, 1, 2
0, 2, 7
0, 3, 6
0, 4, 4
1, 2, 6
1, 3, 5
1, 4, 3
2, 3, 2
2, 4, 8
3, 4, 7
```

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` a Hamiltonian cycle of approximately minimum weight, along with that weight.

For example:

```
>$ ./compile.sh
>$ ./run.sh in1.txt
Hamiltonian cycle of weight 21:
0, 1, 3, 2, 4, 0
```

[3] Assuming ties are broken in ascending numerical order.
[4] With the exception of the first vertex, which must, by convention, appear a second time as the last vertex.

Part 3: Testing

Any approximation naturally has the potential to produce multiple acceptable solutions. After all, it might get lucky[5] and stumble upon the optimal solution, which would certainly satisfy the approximation ratio. But it also can't *always* produce that optimal solution, otherwise it would simply be an optimal algorithm. Prove the following lemmas:

> *Lemma:* There exists a complete, metrically weighted graph $G = (V, E)$ such that your approximation always finds a Hamiltonian cycle of minimum weight.

> *Lemma:* There exists a complete, metrically weighted graph $G = (V, E)$ such that your approximation may find a Hamiltonian cycle of greater than minimum weight.

The given `tsp_tests.py` contains tests capable of compiling and running your implementation[6], parsing its output, and asserting that it has produced an appropriate approximation. It can be invoked from the command prompt:

```
>$ python3 tsp_tests.py
...
----------------------------------------------------------------------
Ran 3 tests in 0.047s

OK
```

You are not required to use Python for this assignment. If you do, you are welcome to make use of the given `weighted_graph.py`, however, you may not modify this code, as it is required by the tests.

Your implementation will be tested using tests written in this manner, so it *must* pass the given `tsp_tests.py`.

Part 4: Submission

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

- `report.pdf` — Pseudocode for an efficient 2-approximation algorithm to approximate Metric TSP, along with proofs of the lemmas given in Part 3.
- `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 approximate Metric TSP, as specified.

Any files other than these will be ignored.

[5] Or, rather, your approximation might make an educated guess that turns out to be correct; it is somewhat misleading to suggest that it merely "got lucky".

[6] These tests will invoke your `compile.sh` and `run.sh` scripts, so they must be run from a UNIX-like command line.