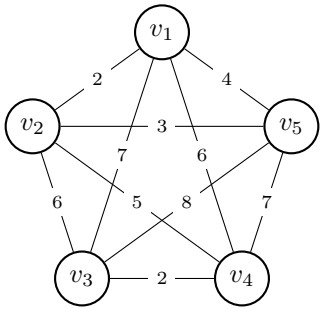
Assignment 5 — Approximations

There are numerous problems in computer science for which an efficient solution is not known. Since those problems often have real-world applications, rather than settling for solving them inefficiently, we can instead approximate their solutions, effectively trading some amount of correctness for improved time complexity.

1 The Traveling Salesperson Problem

The Traveling Salesperson Problem asks for, given a complete, weighted graph, the Hamiltonian cycle of minimum weight. This is a famously difficult problem in computer science[1]. In order to make approximate feasible, we consider *metric TSP*, the restriction to graphs for which the triangle inequality holds[2]. For example, given:

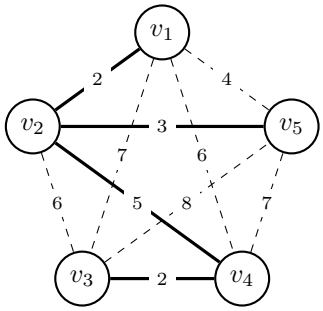


Here there are numerous Hamiltonian cycles of minimum weight. One such cycle is $(v_1, v_2, v_4, v_3, v_5, v_1)$ having a weight of 21.

2 Approximating Metric TSP

When TSP is restricted to the metric case, there then exists a relatively simple 2-approximation that runs in polynomial time:

1. First, construct an MST, which can be done in polynomial time with Kruskal's algorithm. An MST is a decent starting point for an approximation, since its weight is a lower bound on that of the optimal cycle. For example, in the graph above, we get:

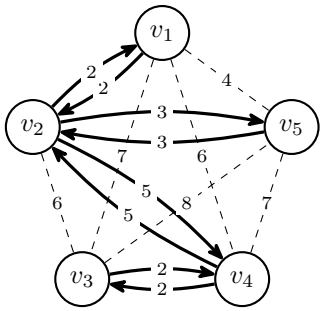
   

   The tree contains the edges $\{(v_1, v_2), (v_3, v_4), (v_2, v_5), (v_2, v_4)\}$ with a weight of 12.

2. Of course, a tree is not a cycle. In order to construct a cycle from the MST, double the tree edges, effectively allowing each edge to be traversed twice. This can be done as part of a depth-first search of the MST. For example:

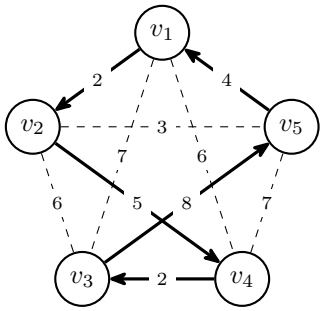
[1] Not only can we not solve it efficiently, we can't approximate it efficiently.

[2] If $w(u, v)$ is the weight of the edge from u to v, we require $w(u, v) \leqslant w(u, x) + w(x, v)$ for all vertices u, v, and x.



We get the cycle $(v_1, v_2, v_4, v_3, v_4, v_2, v_5, v_2, v_1)$ whose total weight is twice that of the tree, 24.

3. But, the approximate cycle is currently not Hamiltonian (it repeats vertices). To fix this, skip any vertices that have already appeared[1]. Because the edge weights form a metric (in particular, the triangle inequality holds), skipping vertices will not increase the weight of the cycle. For example:



In the above graph, we start at v_1 and follow the cycle to v_2, v_4, and v_3. At this point, we cannot proceed as the next vertex in the cycle is one we've already seen. So, we skip until we get to v_5. Because the edge weights satisfy the triangle inequality, skipping vertices will never result in a heavier cycle (and in this case, going directly from $v_3 \to v_5$ resulting in saving 3 versus $v_3 \to v_4 \to v_2 \to v_5$). We then skip v_2 and get to the end v_1. This results in the Hamiltonian cycle—and approximate solution to the problem—of $(v_1, v_2, v_4, v_3, v_5, v_1)$.

In your programming language of choice, implement the polynomial time 2-approximation for the metric version of TSP.

Each input will be provided as a complete edge list with weights. Each edge in the graph will be represented by a space 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 positive integers starting from 1 (i.e. they begin at 1 and contain no "gaps"). You may also assume that the graph will be complete, that all weights will be natural numbers, and that the weights between vertices form a metric.

For example: the above graph could be represented as:

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

[1]With the exception of the first vertex which must appear again as the last vertex.

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 test_files/in1.txt
Hamiltonian cycle of weight 21:
1, 2, 4, 3, 5, 1
```

3 Testing

Any approximation naturally has the potential to produce multiple correct solutions. It's even possible to get lucky and stumble upon the optimal solution (as we did in the example above). The given `tsp_tests.py` contain tests capable of compiling and running your implementation, parsing its output, and asserting that it has produced an appropriate approximation. These tests can be invoked from the command prompt:

```
$ python3 tsp_tests.py
...
----------------------------------------------------------------------
Ran 3 tests in 0.151s
OK
```

While the tests have been written in Python, there is no obligation for your implementation to be written in Python. Similarly, there is no obligation for your implementation to use the given `graph.py` (though you are welcome to do so), however, you should not modify this file, as it is required by the tests.

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

4 Submission

The following items must be demonstrated/presented to me in lab on the day of the deadline.

- *none*

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

- `compile.sh` — A bash script to compile your submission (even if it does nothing).
- `run.sh` — A bash script to run your submission.
- `*.py` or `*.java` — Your source code in Python or Java of a working program.