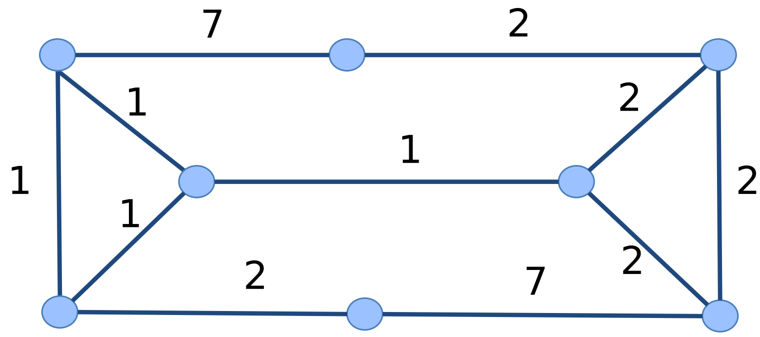
INTRO TO APPROXIMATION, CLASS 2

the travelling salesman has come to us

From the last time:

Exercise one Simulate a biased coin with the probability of heads equal to p using a fair coin. It remains to deal with the case of arbitrary $p \in (0, 1)$ (even irrational), while keeping the expected number of flips very small.

Exercise two Consider the following graph:



1. Find the shortest Hamiltonian cycle in this graph.
2. What is the optimal solution to the graph TSP problem on this graph?
3. What solution is found by a run of the Christofides' algorithm?

Exercise three Find an infinite class of graphs showing that the algorithm for metric TSP that uses a DFS traversal of the minimum spanning tree (and shortcuts) is no better than a 2-approximation.

More precisely, for infinitely many n's construct a graph G_n with n vertices so that

$$\frac{\mathrm{ALG}(G_n)}{\mathrm{OPT}(G_n)} \to 2$$

for $n \to \infty$ where $\mathrm{ALG}(G_n)$ is the cost of the algorithm's solution and $\mathrm{OPT}(G_n)$ is the optimum cost.

Exercise four Consider a connected, directed graph G with edge lengths. Before studying the TSP problem on directed graphs, we can search for a more general structure – a subgraph $P \subseteq G$ of minimum total length such that P contains all the vertices and every vertex has exactly one entering and one exiting edge. This problem is called Minimum Directed Cycle Cover.

- You can use a straightforward total unimodularity argument, if you know what that is from Optimization methods.
- If you are not faimilar with total unimodularity, you can use a direct argument. You can for instance make use of the fact that minimum-weight perfect matching can be found in polynomial time. (Remember, Christofides' algorithm also uses this fact.)

Exercise five Is TSP solvable in polynomial time? One could suggest a dynamic programming algorithm as follows:

1. Create table $d[i, x, y]$ where the meaning of the entry is „best walk from x to y in i steps“.
2. Set $d[0, x, x] = 0$ and $d[0, x, y] = \infty$.
3. For every length $i \in \{1, 2, 3, \ldots, n\}$:
4. For every pair of vertices a, b:
5. Visit neighbors and set $d[i, a, b] = \min_{x \text{ neighbor of } a}(d(a, x) + d[i-1, x, b])$.
6. Set $d[i, b, a] = d[i, a, b]$.
7. Return the minimum value $d[n, v, v]$ over all v.

Analyze this algorithm.