Model for Boltzmann TSP
(modifications inspired Ref. 1)

Instructions
`python anneal.py`
or, if you want to set the starting temperature, hamiltonian, and bias:
`python anneal.py 500 1.0 -0.15`

Notes
The Network:
- A network created and stored within an n x m matrix, where n = # cities and m = # epochs required to travel to them (n + 1)
- Each network node contained an n x m weight matrix

```
    epochs
[ 1 0 0 0 0 1 ]
[ 0 0 1 0 0 0 ]
[ 0 1 0 0 0 0 ] cities
[ 0 0 0 0 1 0 ]
[ 0 0 0 1 0 0 ]
```
Figure 1. A state diagram describing a hamiltonian-complete tour of a traveling saleman over 5 cities

Weight matrix:
- Generally, the weight matrix for a given node, (n_i,e_j) for city 'i' and epoch 'j' followed the principles here:
    > discouraged the activation of n_i in epoch, e_k where k != j (any other time epochs) with one exception: tour completion
    > discourage other node activation within the same time epoch
    > encouraged the activation of itself in order to promote at least on active node in any epoch
    > discourage long distances travelled between any two nodes
    > encourage the active node in the final epoch to activate the same city in the first (and vice versa)
- For any single node, these procedures were followed:
    > All weights were initialized to 0
    > Weights of any adjacent node (in epoch +/- 1) i is = -100*e^(-w)
    > All other nodes within the same epoch received a static weight for breaking the hamiltonian (0.5)
    > Weights of adjacent nodes of the same city receive the same static weight for breaking the hamiltonian (0.5)
    > Self-connections for each node uses a static bias (-0.2)

Annealing process:
- The approach aimed at starting with a high temperature, annealing to a valid hamiltonian tour at that temperature, then reducing the temperature. To avoid breaking hamiltonian cycles and saving

computational time, a swapping mechanism was implemented to swap the active states from any two epochs. Ultimately, the ideal route is D->A->B->E->C->D with a total travel distance of 66:

    > Starting temperature: 2000
    > Final temperature: 1
    > Temperature decrement value: log_10(T) if T > 10, else 0.01
    > The probability of any node changing value: (1 / size of weight matrix) * log10(T) * 1 / (1 + e^-(dE/T))
    > The minimum temperature/configuration were tracked during the process

Comments:

- Configurable parameters were generally maintained in the weight matrix, but probability function used here (sigmoid) allowed you to expand/contract the search space with the cost of computational time (high temperatures leads to half of the nodes activating, which is not a complete hamiltonian tour)
- Most of the annealing computations are spent attempting to find a hamiltonian using a stochastic search. I think these two phases can be separated: search for a hamiltonian and then search for an annealing solution (see below).
- The convergence time can be reduced drastically if instead of randomnly selected one node and proposing that it change its state, first create a hamiltonian tour randomnly and then during the annealing simulation, propose that two different nodes swap. This way, there is no barrier when breaking out of a high-distance route to a lower distance route. With the old method, you would first have to break out of a high-distance route by breaking the hamiltonian and then reconverging to a lower distance route.

Completed:

- Implement a 'swapping' mechanism during the annealing process instead of random node selection, in order to avoid breaking hamiltonian

References:

- [1] Aarts E.H.L., Korst J.H.M. (1987) Boltzmann machines and their applications. In: de Bakker J.W., Nijman A.J., Treleaven P.C. (eds) PARLE Parallel Architectures and Languages Europe. PARLE 1987. Lecture Notes in Computer Science, vol 258. Springer, Berlin, Heidelberg