CATAM Part II - 17.3 - Hamiltonian Cycles TODO fix wording of last paragraph

Introduction

Our first task is to create a sensible data structure to store our graph in, and also write a function that generates a random graph from $\mathcal{G}(n,p)$. Since I'm coding in python, I've implemented a Graph class, which stores all our graph's information, and will contain various methods pertaining to our graph. It allows for us to generate a random graph or input a vertex and edge set.

Question 1

The simplest, and least efficient way to check if a graph has a Hamiltonian cycle is to check each of the $n!$ individual permutations of the vertex set. We can reduce checking slightly since cyclic permutations are equivalent, sow we only have $(n-1)!$. The hard part here is iterating over all permutations, since theres eventually too many to store in memory. We can overcome this by, say, Heap's Algorithm[1], but our implementation uses the *permutations* iterator from the standard python library *itertools*. It can be found under $naive\_hamiltonian1()$ as a method of the Graph class. We find this does work, but is exceedingly slow for $n > 12$ so unable to generate the results we want.

A slightly smarter simple approach is as follows. We try to generate paths starting at 1, using the minimum possible vertex at each step. If we hit a dead end, we go back and try the next minimum vertex. This eventually gives every path starting at 1, so must give us a cycle if it exists. Otherwise it will run through all possible paths and return to the 1 vertex path, in which case there is no cycle. The algorithm is as follows

First set candidate = [1], and k = 1, and loop the following. k is our track of which position vertex in the path we're looking for:

1) If len(candidate) = n, check if the final vertex is a neighbour of 1, and if so return candidate. If it's not a neighbour 1, define maxcandidate = candidate, remove the last element, and reduce k by 1.

2) Otherwise len(candidate) < n. So we find the minimal vertex we can append to the path that we havn't already tried (by comparing the set of neighbours of the most recent vertex with the k'th vertex in maxcandidate), and add it on to candidate. Set maxcandidate to candidate + [0] to indicate we havn't tried any position k+1 elements yet. Iterate k by 1.

3) The only way the above fails is if there's no possible neighbours that we havn't tried. In this case, set maxcandidate to candidate, remove the most recent vertex, and reduce k by 1.

4) We terminate if we ever get a successful cycle, or if candidate ever returns to [1] (or possibly [], if 1 had no neighbours) after this process, ie if after checking all paths we fail to find a cycle.

Our implementation is $simple_hamiltonian()$. This is just about quick enough, and we get the following data, gathered by $q1()$.

```
* Order=3, Size=3, {1: {2, 3}, 2: {1, 3}, 3: {1, 2}} * has a hamiltonian cycle:
    [1, 2, 3]
* Order=3, Size=2, {1: {2}, 2: {1, 3}, 3: {2}} * has no hamiltonian cycle
* Order=4, Size=3, {1: {2, 3}, 2: {1, 3}, 3: {1, 2}, 4: set()} * has no
    hamiltonian cycle
* Order=6, Size=8, {1: {2, 4}, 2: {1, 4, 5}, 3: {4, 6}, 4: {1, 2, 3}, 5: {2, 6},
    6: {3, 5}} * has a hamiltonian cycle: [1, 2, 5, 6, 3, 4]
```

p/n	4	6	8	10	12	14	16	18	20
0.1	6	0	0	0	0	0	0	0	0
0.3	2	3	5	10	19	35	53	71	85
0.5	14	26	51	76	92	97	99	100	100
0.7	44	75	95	99	99	99	99	100	100
0.9	84	98	99	100	100	100	100	100	100

Table 1: Number of graphs containing Hamiltonian cycles from a selection of 100 taken from $\mathcal{G}(n,p)$

[1] https://en.wikipedia.org/wiki/Heap%27s_algorithm

a/n	4	6	8	10	12	14	16	18	20
0.1	0	0	0	0	0	0	0	0	0
0.55	0	0	0	0	0	0	0	0	0
1	5	2	1	2	0	0	1	1	1
1.45	18	13	14	18	24	20	22	29	32
1.9	36	43	51	66	64	66	62	69	100

Table 2: Number of graphs containing Hamiltonian cycles from a selection of 100 taken from $\mathcal{G}(n, a\log n/n)$

Question 2

Suppose we have n vertices. The worst case is we have no Hamiltonian cycle. Assume each of the $(n-1)!$ paths are checked, only failing at the last step (ie every length n path is Hamiltonian starting at v_0, but not a cycle) We assume each simple operation, eg iterating, looking up a value in a list take time c. For each. At worst, we need about $2n+10$ simple operations per iteration, of which most are spent calculating the new possible vertices to append. This gives a time of $c(2n+10)$ per permutation. So we end up with a total running time of $(n-1)!(2n+10)c = O(n!)$.

For an average case, it'll be useful to know how many hamiltonian cycles we should expect to have in a $\mathcal{G}(n,p)$ graph. This isn't too hard to show:

Theorem. *The expected number of hamiltonian cycles in a random graph $\mathcal{G}(n,1/2)$ is given by $\frac{1}{2}(n-1)!p^n$, for $n \geq 3$.*

Proof. First, K_n has $\frac{1}{2}(n-1)!$ cycles of length n, all of which are Hamiltonian (start at 1, have n-1 choices for the second vertex, n-2 for the third... Factor of 1/2 comes from the undirectedness of the graph). Order these cycles, and let X_i be the event G contains the i'th cycle. Then $\mathbb{E}(X)$ is the expected number of hamiltonian cycles in G, and by linearity of expectation is equal to $\sum_i \mathbb{E}(X_i)$. Now $\mathbb{E}(X_i) = p^n$ since each edge occurs independently with probability p, giving our result. □

Suppose an average graph can be taken from $\mathcal{G}(n,1/2)$. We expect the 2^nth cycle we find to be Hamiltonian. To get from one potential cycle to the next is about constant time independent of n, and is about 6 simple operations. To build our first cycle, we expect it to take at most 2n steps, each of which is about 6 simple operations. So we get a running time of about $(12n+6\times 2^n)c$, at least for n large, where by the above theorem we expect a Hamiltonian cycle to exist in $\mathcal{G}(n,1/2)$.
So the algorithm cannot handle too large an n.

Question 3

Modifying the code slightly, to print the number of graphs G with $\delta(G) < 2$, we find most, at least for $a < 1.45$ for a defined in Table 2, graphs fail to be Hamiltonian because $\delta(G) < 2$. The second range may well have been chosen because of the following theorem, proved in IID Graph Theory.

Theorem. *Let $\omega(n) \to \infty$. If $p = \frac{\log(n)-\omega(n)}{n}$ then G has isolated vertices almost surely. If $p = \frac{\log(n)+\omega(n)}{n}$ then G has no isolated vertices almost surely.*

Proof. Note if $X = \Sigma_A I_A$ is a sum of indicator functions then $\mathrm{Var}(X) = \Sigma_{A,B}\mathbb{P}(A)[\mathbb{P}(B \mid A) - \mathbb{P}(B)]$ simply by expanding.

Now let X be the number of isolated vertices $X = \Sigma_v I_v$ where I_v indicated v being isolated. So

$$\begin{aligned}
\mathrm{Var}(X) &= \Sigma_{u,v}\mathbb{P}(u \text{ isolated})[\mathbb{P}(vu \text{ isolated} \mid uu \text{ isolated}) - \mathbb{P}(v \text{ isolated})] \\
&= (1-p)^{n-1}[1-(1-p)^{n-1}] + n(n-1)(1-p)^{n-1}[(1-p)^{n-2} - (1-p)^{n-1}] \\
&\leq \mathbb{E}(X) + n^2(1-p)^{n-1}(1-p)^{n-2} \\
&= \mathbb{E}(X) + \frac{p}{1-p}(\mathbb{E}(X))^2
\end{aligned}$$

where the first term comes from u and v being the same, and the second term otherwise. Now if $p = \frac{\log(n)+\omega(n)}{n}$

$$\mathbb{E}(X) = \frac{1}{1-p}n(1-p)^n \leq \frac{1}{1-p}ne^{-pn} \to 0$$

So $X = 0$ a.s. by Markov's Inequality. If $p = \frac{\log(n)-\omega(n)}{n}$

$$\mathbb{E}(X) \approx \frac{1}{1-p} ne^{-pn} \to \infty$$

So

$$\frac{\text{Var}(X)}{(\mathbb{E}(X))^2} \leq \frac{1}{\mathbb{E}(X)} + \frac{p}{1-p} \to 0$$

So $X \neq 0$ a.s. by Chebyshev's Inequality.

□

Here we're considering a range of values $p \in [\frac{\log(n)-0.9\log(n)}{n}, \frac{\log(n)+0.9\log(n)}{n}]$, so as n grows, the probability of having isolated vertices for $a < 1$ grows tends to 1, which agrees with our results.

Question 4

The *smarter_hamiltonian*(T) method of the Graph class performs the algorithm. To determine what a suitable T would be for various n,p we'll run some trials, and determine a value of T such that we'll find 95% of hamiltonian cycles. We can do so by first using our *simple_hamiltonian*() function to only pick out graphs with hamiltonian cycles, then use, say *smarter_hamiltonian*(1000) to determine a 95'th percentile on T. We'll run 5000 trials, by which I mean we'll test 5000 graphs with hamiltonian cycles for each n,p. We'll only go up to $n = 14$, since *simple_hamiltonian*() is fairly slow for larger n for small p, and won't bother testing $p < 0.3$ since we rarely ever have hamiltonian. We gather the following data using $q4()$.

p/n	**4**	**6**	**8**	**10**	**12**	**14**
0.3	6	15	33	62	80	103
0.5	7	16	27	35	34	33
0.7	7	13	16	17	18	19
0.9	6	8	10	18	14	15

Table 3: 95'th percentile of T required to find a hamiltonian cycle, given one exists, from a selection of 5000 trials in $\mathcal{G}(n,p)$

We find unsurprisingly a larger T is required for small p, and not too large a T is required, making this fairly fast. If we only have T depending on n, setting T to about n^2 should catch almost all Hamiltonian cycles, and is a lot better than the average case complexity of the above.

Question 5

There are 2 steps here. We first need to get P_j to length n-1. Then we need to complete the cycle.

To attain length n-1, we need to add n-1 vertices to our original. The only way to increase the length of P_j is if v_k is joined to a vertex not in P_j. This occurs with probability $1-(1-p)^{n-k-1}$. We can certainly bound the sum of the reciprocals of these by $\frac{n}{p}$, which upper bounds how long we expect it takes us to reach length n-1. Let's, say, double this to be reasonably sure we've reached length n-1.

Once P_j hits length n-1, we may model the time taken to reach a hamiltonian cycle as a geometric distribution with parameter p, since we only need the last and first vertex having an edge between them. We can use the 95th percentile of the CDF for this step, which is $\min\{k, \sum_k (1-p)^{k-1}p > 0.95\}$.

One can verify this gives a T greater than all of the entries of Table 3, and should be reasonable for large n.

In general we expect small graphs with small p to have few Hamiltonian cycles and large p to have many. The algorithm terminates in 2 cases:

1) If v_0 has no neighbours
2) If we find a hamiltonian path
3) If we hit T iterations

Case 1 is unlikely in general, though occurs more for p small than p large. Case 2 occurs more often for p large, and case 3 for p small, so we expect the algorithm to take much longer for small p.

In the average case, we expect the 2^nth length n-1 path to be a cycle. Once we reach length n-1, step 3 is constant time, since all we're doing is a look up and a splice, so we have $O(2^n)$. Before we reach length n-1, we use step 2, which is also constant time. We don't expect to take more than 2 steps to find a neighbour of v_k not in P_j (since we have p=1/2), and step 2 itself is $O(n)$ (since we're reading the set of neighbours, and intersecting it with P_j). If step 2 fails to find a valid neighbour, we go on to the constant time step 3. So this step is in total $O(n)$. So in average case, the algorithm is $O(2^n)$ to find a hamiltonian cycle.