CATAM Part II - 17.3 - Hamiltonian Cycles

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_hamiltonian*1() 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.

This is 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 $(n-1)!(2n+10)c = O(n!)$.

Now for an average case, supposing an average graph can be taken from $\mathcal{G}(n, 1/2)$. Supposing such an average graph has k Hamiltonian cycles, from Table 1 it seems reasonable to assume $k = 1/2$. So in the case we have a Hamiltonian cycle, we expect to find it in $(n-1)!/2$ permutations, with each prior permutation failing in $n/2$ steps. This gives a running time of $(n-1)!(3nc+d)/4$. In the other case where we don't have a Hamiltonian cycle, we similarly get $(n-1)!(3nc+d)/2$, so expect a running time of $3(n-1)!(3nc+d)/4 = O(n!)$.

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

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

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.

Question 5

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 longer for small p.