Programming-Assignment-2

The sample space Ω contains all possible undirected graphs with n distinct vertices. Since the set of vertices is the same for each graph in Ω, graphs can be distinguished by the set of edges E. Since there are $\binom{n}{2}$ possible distinct edges in a graph with n distinct vertices, and each edge can be chosen to exist independently, there are $2^{\binom{n}{2}}$ possible distinct sets of edges. Thus, the sample space has $2^{\binom{n}{2}}$ distinct elements. Thus $P(G_E) = 2^{-\binom{n}{2}}$ is the probability of sampling a graph defined by the set of edges E from this sample space. *Note: I have interpreted 'distinct' as that every vertex can be identified uniquely (vertices are not identical). And hence any edge between two distinct vertices is identifiable, and thus also becomes a distinct edge, that is why there arrives* $2^{\binom{n}{2}}$ *distinct sets of edges and thereby* $2^{\binom{n}{2}}$ *distinct graphs in the sample set.*

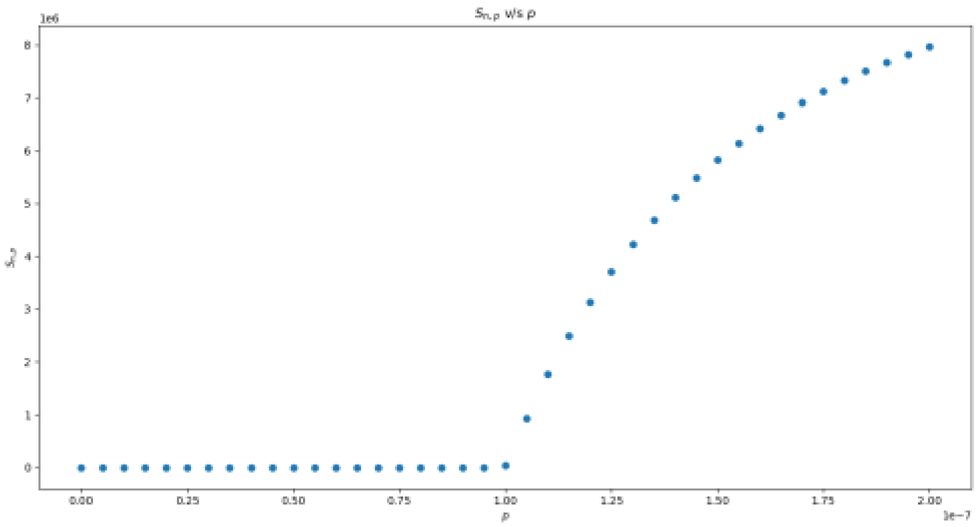
1. Intuitively, I think that $S_{n,p}$ may first increase slowly, then rapidly, and then $S_{n,p}$ would start to saturate and increase much slowly.

   For a random graph G generated using $G(n, p)$ method, we know that every edge exists with a probability p independently of other edges. For a larger p there would be more edges, so more vertices are connected together. It seems reasonable to guess that as we increase p, the size of the largest connected component $S_{n,p}$ should increase.

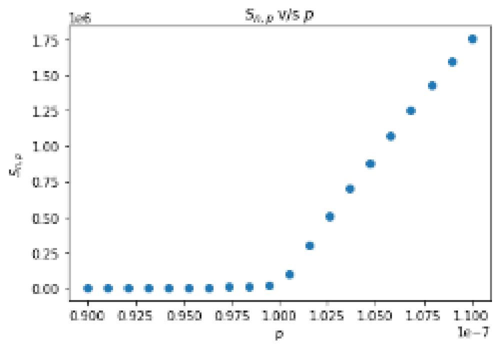
   Expanded thoughts:

   > Consider that at some point during the generation of G there are two connected components C_1 and C_2 with n_1 and n_2 vertices. If these components are to be still disconnected on the generation of next edge, then the next edge should not form for all the pairs of vertices $(v_1, v_2)\ \ v_1 \in C_1$ & $v_2 \in C_2$. The edge between v_1 and v_2 does-not exist with probability $(1-p)$ and there are $n_1 \cdot n_2$ such pairs. So, the probability that they will not be connected on the generation of the next edge would be $P_d = (1-p)^{n_1 n_2}$.
   >
   > For a small p the probablity P_d that two given disjoint connected components c_1 and c_2 could have been merged would be very small, so there would be many small disjoint connected components. For a comparatively larger p, intuitively we might expect larger connected components. So, n_1 and n_2 will be greater. The probability that any two arbitrary disjoint connected components remain disjoint in the next generation of an edge $P_d = (1-p)^{n_1 n_2}$ would be less. Somewhere in this stage $S_{n,p}$ should be increasing at a much larger rate.
   >
   > At some point a lot of smaller components would likely become connected, resulting in a very small number of massive connected component and other small disjoint connected components.
   >
   > Intuitively speaking, when $S_{n,p}$ is very large and gets larger, there would be very few vertices that are not already in $S_{n,p}$. The probability that a new vertex (or a disjoint connected component (set of vertices)) is added to $S_{n,p}$ on the next generation of an edge becomes smaller. This would cause a decrease in the rate of increase of $S_{n,p}$.

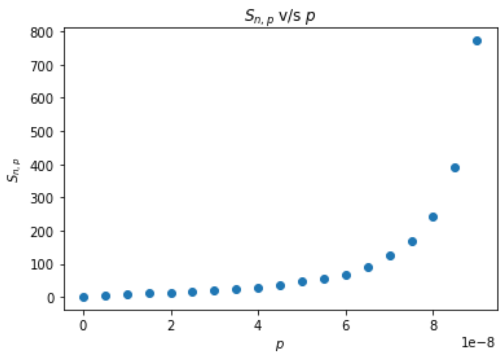
2. The below plot shows how $S_{n,p}$ varies with p in range $[0, 2/n]$. We know that for $p = 0$ there will be no edges, so $S_{n,p=0} = 1$. Forty equidistant points were chosen from the interval $p \in \left[\frac{0.05}{n}, \frac{2}{n}\right]$ and the experiment was repeated 20 times for each point. Then again 20 equidistant points were selected in the interval $\left[\frac{0.9}{n}, \frac{1.1}{n}\right]$ and experiment was run 20 times for each point. These points were used to generate plot-2.
Plot-1 shows how $S_{n,p}$ varies for $p = 0$ to $p = \frac{1}{n}$. The two smaller plots below show how $S_{n,p}$ varies from $p = \frac{0.9}{n}$ to $p = \frac{1.1}{n}$ (plot-2). And $S_{n,p}$ for $p \leq \frac{0.8}{n}$ (plot-3). The computer code to simulate $S_{n,p}$ is attached after the answer of question-3.



Plot-1



Plot-2



Plot-3

3. From the above plots it is clear that $S_{n,p}$ goes through a phase transition at $p = 1/n$. For $p < 1/n$, $S_{n,p}$ increases much slowly but its rate of increase keeps growing. For $p > 1/n$, as $S_{n,p}$ gets larger, its rate of increase w.r.t p decreases. In the interval $p \in \left(\frac{1}{n}, \frac{1.1}{n}\right)$, $S_{n,p}$ seems to increase linearly with p as seen from the plot-2. For $p < \frac{1}{n}$, $S_{n,p}$ shows a somewhat exponential trend as can be seen in plot-3.

Computer-Code to generate the data:

```
import math
from queue import Queue
import random
import multiprocessing as mp
from csv import writer
from union_find_ADT import UnionFind
import matplotlib.pyplot as plt


def worker_naive(n,p,queue):
    uf = UnionFind(n)
    for i in range(n):
        for j in range(i+1,n):
            if random.random()<p:
                uf.union(i,j)
    # find Snp and tally of sizes
    max_size = 1; sizes={}
    for size in uf.itter_sizes():
        if size not in sizes: sizes[size]=0
        sizes[size]+=1
        max_size = max(max_size, size)
    # return [p, max_size, uf.num_sets, sizes]
    queue.put([p, max_size, uf.num_sets, sizes])


def worker(n, p, queue):
    # ref: Batagelj V, Brandes U. Efficient generation of large random
    # networks. Physical Review E. 2005 Mar 11;71(3):036113.
    # union-find to compute connected components
    # while making the graph.
    uf = UnionFind(n)
    w = -1; v = 1
    lp = math.log(1.0 - p)
    while v < n:
        lr = math.log(1.0 - random.random())
        w = w + 1 + int(lr / lp)
        while w >= v and v < n:
            w = w - v
            v = v + 1
        if v < n:
            uf.union(v,w)

    # find Snp and tally of sizes
    max_size = 1; sizes={}
    for size in uf.itter_sizes():
        if size not in sizes: sizes[size]=0
        sizes[size]+=1
        max_size = max(max_size, size)
    # return [p, max_size, uf.num_sets, sizes]
    queue.put([p, max_size, uf.num_sets, sizes])


def watcher(queue:Queue, fname:str):
    with open(fname, 'w') as f:
        print('saving in', fname)
        csv_writer = writer(f)
        csv_writer.writerow(['p', 'max_size', 'num_components', 'sizes'])

        num_finished = 0
        while 1:
            m = queue.get()
            if m == 'kill':
                print('\nall done', flush=True)
                break
            csv_writer.writerow(m)
            f.flush()

            plt.scatter(m[0],m[1])
            plt.title('$S_{n,p}$ v/s $p$')
            plt.xlabel('$p$')
            plt.ylabel('$S_{n,p}$')
            plt.draw()
            plt.pause(0.00001)

            num_finished += 1
            print(f'\r {num_finished}', end = '', flush=True)
        plt.show()


if __name__ == '__main__':
```

```
    mp_manager = mp.Manager()
    queue = mp_manager.Queue()

    n = 1E7
    num_points = 40
    num_repeats = 20
    num_parallel = 20

    # vary p from (0, 2/n]
    fname = 'outputs.csv'
    probs = [2*(i+1)/(n*num_points) for i in range(num_points)]

    # vary p in 0.9/n to 1.1/n
    # fname = 'outputs-mid.csv'
    # probs = [0.9/n + 0.2*x/(n*(num_points-1)) for x in range(num_points)]

    num_parallel = min(num_parallel, num_repeats)
    pool =  mp.Pool(num_parallel)
    listener = pool.apply_async(watcher, (queue,fname))

    queue.put([0.0, 1, n, {1:n}]) # for p=0
    for p in probs:
        for r in range(0,num_repeats,num_parallel):

            jobs = []
            for i in range(num_parallel):
                job = pool.apply_async(worker, (int(n),p,queue))
                jobs.append(job)
            for job in jobs:
                job.get()

    queue.put('kill')
    pool.close()
    pool.join()
```

Code to generate the plots:

```
import pandas as pd
import matplotlib.pyplot as plt

data = pd.read_csv('outputs.csv')

# scatter plot each run
plt.scatter(data['p'], data['max_size'])
plt.title('$S_{n,p}$ v/s $p$')
plt.xlabel('$p$')
plt.ylabel('$S_{n,p}$')
plt.show()

# plots avg. stat
data = data.groupby('p').mean()
plt.scatter(data['max_size'].index, data['max_size'])
plt.title('$S_{n,p}$ v/s $p$')
plt.xlabel('$p$')
plt.ylabel('$S_{n,p}$')
plt.show()

# for p <= 0.9/n
tempdata = data.loc[data['max_size'].index <= 0.9E-7]
plt.scatter(tempdata['max_size'].index, tempdata['max_size'])
plt.title('$S_{n,p}$ v/s $p$')
plt.xlabel('$p$')
plt.ylabel('$S_{n,p}$')
plt.show()

# for p=0.9/n to 1.1/n
tempdata = pd.read_csv('outputs-mid.csv').groupby('p').mean()
plt.scatter(tempdata['max_size'].index, tempdata['max_size'])
plt.title('$S_{n,p}$ v/s $p$')
plt.xlabel('$p$')
plt.ylabel('$S_{n,p}$')
plt.show()
```