Applied Algorithms Assignment 2

Fei Fan (Peter) Chen

January 15, 2021

Exercise 1 (10 pts)

a) Let p be the probability of being assigned to your desk and P be the event of being assigned to your desk or not. We assign value of 1 if you are assigned to your desk and value of 0 if you are not. The expected number of people assigned to their desk with replacement is:

$$E[\sum_{i=1}^{n} P] = \sum_{i=1}^{n} E[P] = \sum_{i=1}^{n} \frac{1}{n} = 1$$

b) The probability of someone not being assigned to a particular desk is $1 - \frac{1}{n}$, therefore the probability that zero people are assigned to a particular desk is $(1 - \frac{1}{n})^n$.

c) This is combinatorial:

$$\binom{n}{1} \frac{1}{n} (\frac{n-1}{n})^{n-1} = (\frac{n-1}{n})^{n-1} = (1 - \frac{1}{n})^{n-1}$$

d) The probability of a particular desk having no one assigned approaches $\frac{1}{e}$ and the probability of exactly one person assigned to particular desk approaches also $\frac{1}{e}$.

e) This can be rephrased as every person gets their own seat (or no replacement):

$$\frac{n}{n} \frac{(n-1)}{n} \ldots \frac{1}{n} = \frac{n!}{n^n}$$

Exercise 2 (10 pts)

If we always choose a pivot that is the median of a set, QSelect for any number of rank not equal to the median is then:

$$T(N) = T(N/2) + N$$

Expanding:

$$T(N/2) = T(N/4) + N/2$$

$$T(N/4) = T(N/8) + N/4$$

...

$$T(1) = 1$$

Substituting:

$$T(N) = N + N/2 + N/4... + 1 = 2N - 1$$

Exercise 3 (10 pts)

Consider two vertices s and t in graph g, where all the nodes in $g\backslash\{s, t\}$ are $\{v_1, v_2, ...v_{n-2}\}$. Assume there is an edge from all v_i to s and two edges to t. There are no edges between v_i.

Then the chance of finding the min-cut would require every v_i to not contract with s. On each iteration of Karger's algorithm, there are two possibilities:

- contract with t - where the probability of contracting with node t is $\frac{2(n-i)}{2(n-i)+(n-2)} \leq \frac{2}{3}$ for each iteration i
- contract with s - we will not find the min-cut

Therefore the probability of finding the min-cut is:

$$\frac{2}{n}^{n-2} < \frac{2(n-2)}{2(n-2)+(n-2)} + \frac{2(n-3)}{2(n-3)+(n-2)}.. + \frac{2}{2+(n-2)} < \frac{2}{3}^{n-2}$$

This is of the form $c^{-\alpha n}$ where to get the possibility of a min-cut when there are two nodes left, you would have to run c^{n-2} trials (exponential in n!).

NOTE: I am having a bit of trouble rationalizing this for cases where there are edges between v_i or from v_i to t that grow much faster ($100n$ or n^2) than edges from v_i to s). The proof says that it is the same exponential case where ultimately the need for no edges to s to be touched means the probability multiplied together becomes a decaying exponential (vs. denominators canceling the numerator in the Karger's Algorithm case where s and t are unspecified). But it seems practically we can get that decaying factor to arbitrary high number of 9s depending on the graph where even though the number of trials is exponential, it would still be feasible?

Exercise 4 (20 pts)

See graphs.

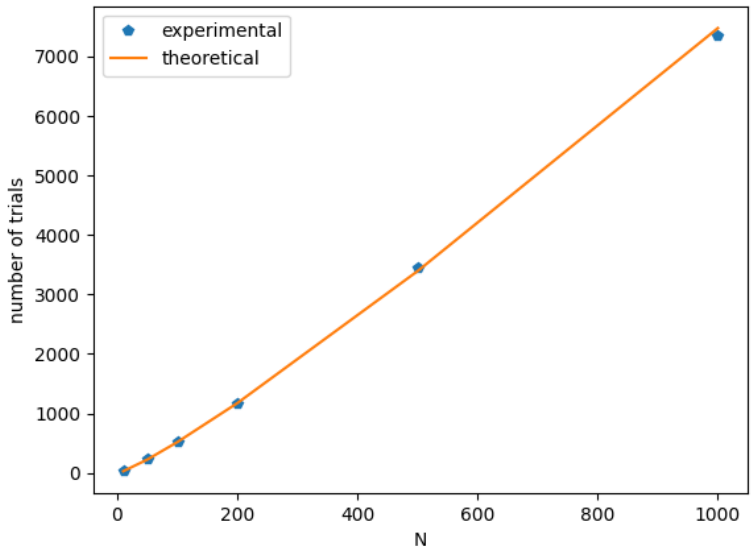

Figure 1: Number of coupons sampled before a set is complete.

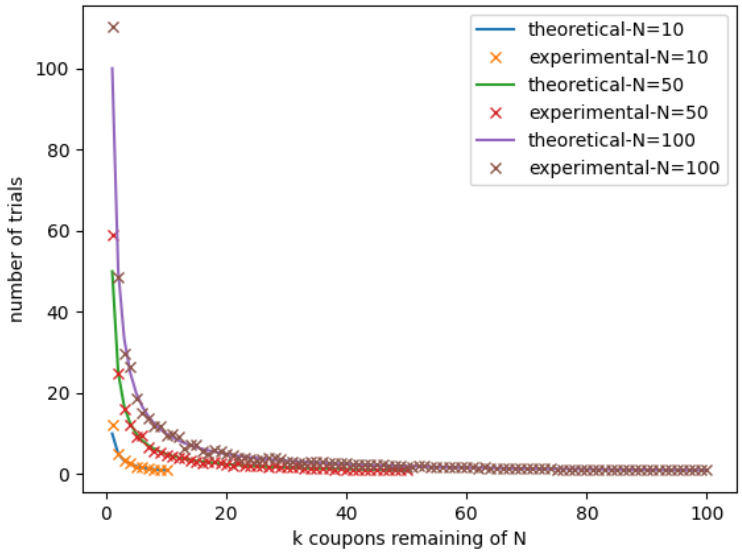

Figure 2: Number of trials to get a new coupon given k of N coupons remaining.

Code

```
import random
import math
import matplotlib.pyplot as plt

class CouponCollector():
  def __init__(self, N):
    """
    CouponCollector simulator
    N - number of coupons
    """
    self.N = N
    self.found = []

  def findNextCoupon(self):
    # technically should return error, but meh
    unfound_coupons = self.N - len(self.found)
    if unfound_coupons <= 0:
      return 0

    trials = 0
    coupon = None
    while coupon is None or coupon in self.found:
      trials += 1
      coupon = int(random.uniform(0, self.N))
    self.found.append(coupon)

    return trials

if __name__ == "__main__":
  random.Random(None)

  # number of couples to try
  N = [10, 50, 100, 200, 500, 1000]
  samples = 100
  N_select = [10, 50, 100] # plot x_n_k as well for these N

  experimental_trials = []
  theoretical_trials = []

  plt.figure()
  for n in N:
    total_trials_experimental = 0
    # theoretical number of trials
    total_trials_theoretical = n * math.log(n) + 0.57 * n
    theoretical_trials.append(total_trials_theoretical)
    # random variable: number of trials needed to get a new coupon when k
```

```
    samples_remain = [ n-i for i in range(n) ]
    x_n_k_theoretical = [ n/k for k in samples_remain ]
    x_n_k_experimental = [0] * n
    for _ in range(samples):
      c = CouponCollector(n)
      for i in range(n):
        experimental = c.findNextCoupon()
        total_trials_experimental += experimental
        if n in N_select:
            x_n_k_experimental[i] += experimental
    for i in range(n):
      x_n_k_experimental[i] = x_n_k_experimental[i] / samples
    avg_experimental_trials = total_trials_experimental / samples
    experimental_trials.append(avg_experimental_trials)
    # Plot the x_n_k
    if n in N_select:
      plt.plot(samples_remain, x_n_k_theoretical, "-", label=f"theoretica
      plt.plot(samples_remain, x_n_k_experimental, "x", label=f"experimen
      plt.ylabel('number of trials')
      plt.xlabel(f'k coupons remaining of N')
      plt.legend()
# Plot the experimental vs theoretical
plt.figure()
plt.plot(N, experimental_trials, "p", label="experimental")
plt.plot(N, theoretical_trials, "-", label="theoretical")
plt.ylabel('number of trials')
plt.xlabel('N')
plt.legend()
plt.show()
```