**[과제]Reservoir Sampling / DGIM Algorithm**

**빅데이터 최신기술**

20203066 박민희

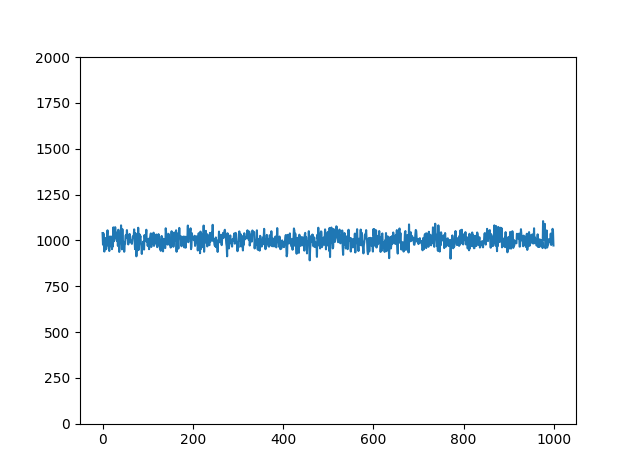
**Reservoir Sampling**

1.비복원 추출

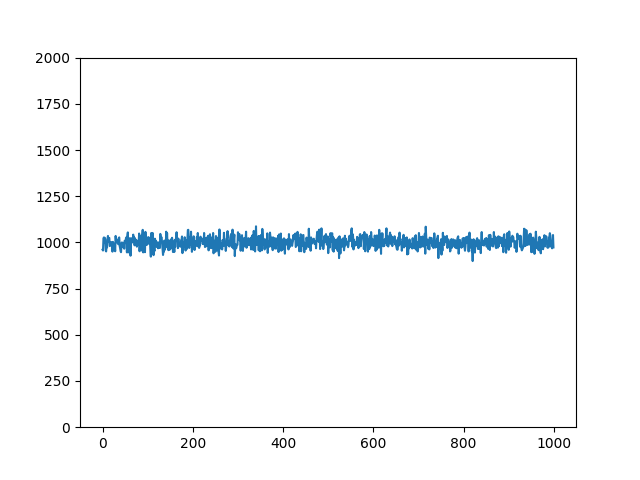
|  |
| --- |
| import random  import matplotlib.pyplot as plt  arr = [0 for i in range(1000)]  class Reservoir:  def \_\_init\_\_(self, k): # 생성자 만드는거  self.sampled = [] # sampling한 것을 담을 리스트  self.k = k # 초기화  self.cnt = 0 # 지금 들어오는 아이템이 몇번째인지  def put(self, item): #스트림에서 아이템하나가 들어오면 어떻게 처리할것인지  if self.cnt < self.k:  self.sampled.append(item)  arr[item] += 1  else:  r = random.randint(0, self.cnt)  if r < self.k: # 랜덤으로 뽑은 r가 k보다 작으면  arr[self.sampled[r]] -= 1  self.sampled[r] = item # sampled[r] 에다가 item 넣기  arr[item] += 1  self.cnt += 1  for i in range(10000):  reservoir = Reservoir(100) # <--요기 괄호안에 있는게 그 추출할 크기  for j in range(1000):  reservoir.put(j)  #print(reservoir.sampled)  ##0~999까지 100개를 10000번  plt.plot(arr) #그래프  plt.ylim([0, 2000])  plt.show() |

2.복원 추출

|  |
| --- |
| import random  import matplotlib.pyplot as plt  arr = [0 for i in range(1000)]  class Reservoir:  def \_\_init\_\_(self, k):  self.sampled = []  self.k = k  self.cnt = 0  def put(self, item):  if self.cnt < self.k:  self.sampled.append(item)  arr[item]+=1  else:  r = random.randint(0, self.cnt)  if r<self.k: #랜덤으로 뽑은 r가 k보다 작으면  arr[self.sampled[r]]-=1  self.sampled[r] = item #sampled[r]에 item넣기  arr[item]+=1  self.cnt +=1  for i in range(10000):  for j in range(100):  reservoir = Reservoir(1)  for k in range(1000):  reservoir.put(k)  #print(reservoir.sampled)  plt.plot(arr) #그래프  plt.ylim([0, 2000])  plt.show() |



<비복원 추출>

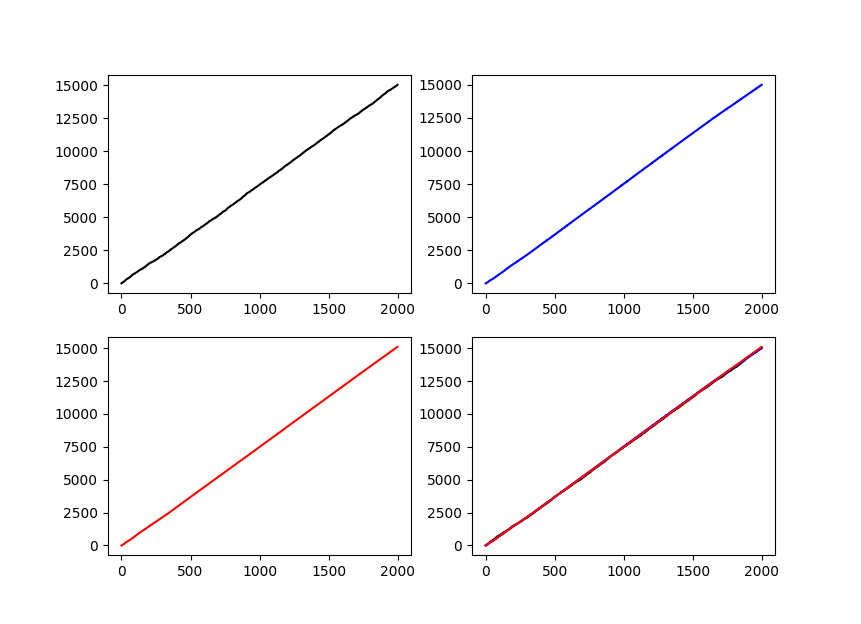


<복원 추출>

Reservoir Sampling에서 복원 추출과 비복원 추출을 시각화 하여 그래프로 그렸을 때 각 그래프 의 모든 값이 약 950에서1050사이의 값으로 그려진 것으로 보인다. 1000개의 숫자중 100개를 뽑는 시행을 10000회 반복하므로, 예상치는 1000(=)이였다. 복원추출과 비복원 추출 모두 그래프에서 예상치와 각 숫자가 추출된 횟수가 비슷하게 추출된 것을 확인할 수 있다.

**DGIM Algorithm**

|  |
| --- |
| import random  import matplotlib.pyplot as plt  class Bucket: #박스 만드는 클래스  def \_\_init\_\_(self,start,end):  self.start = start  self.end = end  def \_\_repr\_\_(self):  return f"({self.start},{self.end})"  class Bucket2: #박스 만드는 클래스  def \_\_init\_\_(self,start,end,psum):  self.start = start  self.end = end  self.psum = psum  def \_\_repr\_\_(self):  return f"({self.start},{self.end},{self.psum})"  class DGIM:  def \_\_init\_\_(self):  self.bucket\_tower = [[]]#상자들을 저장하는 공간  self.ts =0 #timestamp  def put(self, bit):  if bit == 1: #1이 들어오면  b = Bucket(self.ts, self.ts) #노랑 박스를 만들어  self.bucket\_tower[0].insert(0,b) #0번자리에 b를 넣어  layer = 0  while len(self.bucket\_tower[layer]) >2:  if len(self.bucket\_tower) <= layer+1:  self.bucket\_tower.append([])  b1 = self.bucket\_tower[layer].pop()  b2 = self.bucket\_tower[layer].pop()  b1.end = b2.end #합치기  self.bucket\_tower[layer+1].insert(0,b1)    layer +=1  self.ts +=1  # 0010100010[10101]0[101]000[1][1]0  # s e  def count(self, k): #가장 최근 k개에 몇개의 1이 있었는지  s = self.ts - k  cnt = 0  #검사  for layer, buckets in enumerate(self.bucket\_tower):  for bucket in buckets:  if s <= bucket.start: # 중간지점인지 아닌지  cnt += (1<<layer)#중간지점이 아닌경우  elif s <= bucket.end: #중간지점인경우  cnt += (1<<layer) \* (bucket.end -s +1)//(bucket.end - bucket.start+1)#비례하는거 계산  return cnt  else:  return cnt  return cnt  class DGIM2:  def \_\_init\_\_(self):  self.bucket\_tower = [[]]  self.ts = 0 # timestamp  def put(self, bit):  #if bit == 1: # 1이 들어오면  b\_ = Bucket2(self.ts, self.ts,bit) # 노랑 박스를 만들어  self.bucket\_tower[0].insert(0, b\_) # 0번자리에 b를 넣어  layer = 0  while len(self.bucket\_tower[layer]) > 2:  if len(self.bucket\_tower) <= layer + 1:  self.bucket\_tower.append([])  b1 = self.bucket\_tower[layer].pop()  b2 = self.bucket\_tower[layer].pop()  #b1이랑 b2의 누적합이 2\*\*layer보다 작거나 같으면  if(b1.psum + b2.psum) <= 2\*\*layer:  b1.end = b2.end #원래대로 합쳐주고  b1.psum += b2.psum #누적값 바꿔주고  self.bucket\_tower[layer + 1].insert(0, b1) #layer+1에 b1넣고  layer += 1  else: #b1이랑 b2의 누적합이 2의 레이어제곱보다 크다면  self.bucket\_tower[layer + 1].insert(0, b1) #b1만 layer+1에 넣고  self.bucket\_tower[layer].insert(1,b2) #위에서 pop했던 b2 다시 넣어주기  layer += 1  self.ts += 1  def count(self, k):  s = self.ts - k  cnt = 0  #검사  for layer, buckets in enumerate(self.bucket\_tower):  for bucket in buckets:  if s <= bucket.start: #중간지점인지 아닌지 cnt += bucket.psum  elif s <= bucket.end: #중간지점인경우  cnt += bucket.psum \* (bucket.end -s +1)//(bucket.end - bucket.start+1) #비례하는거 계산  return cnt  else:  return cnt  return cnt  dgim = DGIM()  bitstream = []  #첫번째 방법에 필요한 객체랑 배열들  a=[]  b=[]  c=[]  d=[]  dgima = DGIM()  dgimb = DGIM()  dgimc = DGIM()  dgimd = DGIM()  #요기 밑에는 스트림 랜덤으로 받는부분  for i in range(10000): #10000개의 정수  prob = random.randrange(16) #0~15까지 랜덤으로 난수(정수) 생성  bitstream.append(prob)  a.append(((1 << 3) & prob) >> 3) #1000  b.append(((1 << 2) & prob) >> 2) #0100  c.append(((1 << 1) & prob) >> 1) #0010  d.append((1 & prob)) #0001  #첫번째 방법  #각각 비트 쪼갠거를 dgim에 넣은거지  for i in a:  dgima.put(i)  for i in b:  dgimb.put(i)  for i in c:  dgimc.put(i)  for i in d:  dgimd.put(i)  #두번째 방법  dgim2 = DGIM2()  for i in bitstream:  dgim2.put(i)  realsum = [] #실제합  sum1 = [] #첫번째합  sum2 = [] #두번째합  for i in range(2000):  realsum\_ = 0  for j in range(i):  realsum\_ += bitstream[j]  realsum.append(realsum\_)  sum1.append(dgima.count(i)\*8 + dgimb.count(i)\*4 + dgimc.count(i)\*2 + dgimd.count(i))  sum2.append(dgim2.count(i))  #print(realsum[i],sum1[i],sum2[i]) #값 출력  plt.plot(realsum,'k') #검은색  plt.plot(sum1,'b') #파랑색  plt.plot(sum2,'r') #빨간색  plt.show() |



왼쪽위는 실제합, 오른쪽위는 첫번째 방법으로 구한 합,

왼쪽 아래는 두번째 방법으로 구한 합, 오른쪽 아래는 세 그래프를 합친 그래프이다.

코드는 세 그래프를 합친 그래프만 출력되게 되어있고 위의 사진은 임의로 4개의 그래프를 보인것이다.

DGIM Algorithm에서 실제의 합과 제시된 두가지 방법으로 구현한 합을 겹쳐보았을 때, 그래프를 확인 해보니 어떤 것이 더 정확한지는 따지기 힘들었다.

여러 번 시행하여 각 구간을 확대 해본 결과 첫번째 방법으로 구현한 합과 두번째 방법으로 구현한 합이 번갈아 가면서 실제합의 그래프와 더 가깝게 나타났기 때문이다.