CC40A Auxiliar
Algoritmos Aleatorios

Profesor: Ricardo Baeza Yates
Prof. Auxiliar: Rodrigo Paredes

Dept. de Ciencias de la Computación, Universidad de Chile.

1. Preliminares

Básicamente los algoritmos aleatorios tienen las siguientes propiedades:

- Toman decisiones aleatorias.
- Tienden a comportarse siempre como en el caso promedio, o dicho de otro modo, su comportamiento es "independiente" de la entrada.

Los algoritmos aleatorios son especialmente útiles cuando el problema a resolver tiene varias soluciones, calcular en forma determinística una solución es difícil, pero verificar la solución es fácil. También son útiles para protegerse de los casos duros de un problema, por ejemplo, antes de ordenar con QuickSort, podemos tomar la secuencia, desordenarla al azar y luego ordenar la secuencia desordenada, con esto en promedio evitamos el peor caso de $O(n^2)$.

2. Calcular π utilizando un algoritmo aleatorio

Consideremos la Figura 1 donde se muestra una circunferencia de radio a, circunscrita dentro de un cuadrado de lado $2a$.

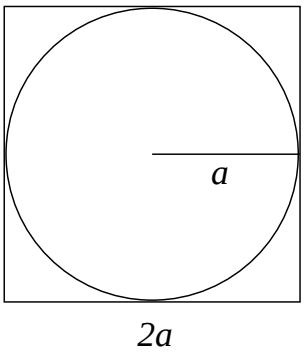

Figura1. Antes de colocar puntitos al azar

El área del cuadrado es $4a^2$ y el área de la circunferencia es πa^2, luego la razón entre las áreas de la circunferencia y el cuadrado es $\pi/4$.

Si generamos puntos al azar (X, Y), con X e Y siguiendo una distribución uniforme $U(0, 2a)$, y contamos la frecuencia de los puntos que caen dentro de la circunferencia (que llamaremos $p_{\odot}$) y los dividimos por la cantidad total de puntos ($p_{\square}$), tendremos una estimación de π según la siguiente fórmula:

$$\pi \approx 4 \cdot \frac{p_{\odot}}{p_{\square}}$$

En la Figura 2 se muestra cuando se han agregado 55 puntos al azar, segun una distribución uniforme. Según la fórmula, el valor estimado para π en esta iteración es 3.12727...

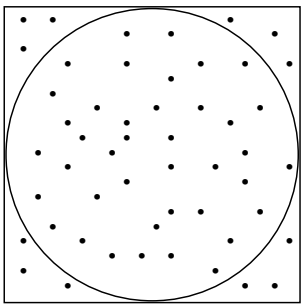

Figura2. Después de colocar 55 puntitos al azar, según una distribución uniforme

3. Buscar un buen estudiante

La idea de este problema es encontrar un buen estudiante, vale decir, un estudiante mejor que el del medio. La primera solución que viene a la cabeza es ordenar el conjunto y sacar el máximo ($O(n \log n)$), con este, estamos seguros que es mejor que el del medio. Ahora, para sacar el máximo basta con recorrer la lista completa y sacar el máximo ($O(n)$). Si somos un poco más observadores, basta con revisar hasta la mitad más un alumno; el máximo de ellos, necesariamente es mejor que el alumno del medio, pero esto también tiene costo ($O(n)$).

Que pasa si tomamos un alumno al azar, cuál es su probabilidad de que sea mejor que el del medio. Claramente es de 1/2.

Si ahora tomo dos alumnos al azar, cuál es la probabilidad de que al menos uno de ellos sea mejor que el del medio. Esto es más fácil verlo al revés, cuál es la probabilidad de que los dos sean menores que el del medio, y luego con esta probabilidad calculamos la primera:

$\mathbb{P}$(al menos uno de los dos sea mejor que el del medio) $= 1 - \mathbb{P}$(los dos son peores que el del medio)

La segunda probabilidad es $1/2 \cdot 1/2 = (1/2)^2$, luego la primera es $1 - (1/2)^2$.

Si tomamos un conjunto de tamaño k, tenemos que la probabilidad de que los k sean más malos que el del medio es $(1/2)^k$, luego la probabilidad de que uno de ellos sea mejor que el del medio es

$$\mathbb{P}(k) = 1 - \left(\frac{1}{2}\right)^k$$

Por último, si tomamos el máximo del conjunto de tamaño k, sabemos que este elemento tiene probabilidad $1 - (1/2)^k$ de ser mejor que el del medio.

Si $k = 2$ esta probabilidad vale 0.75, pero con $k = 10$, esta probabilidad es aproximadamente 0.999023, y con $k = 20$, la probabilidad es de 0.999999046 lo que ya es suficientemente bueno para casi cualquier fin práctico.

Este es un ejemplo de algoritmo tipo Monte Carlo.

4. Buscar una secuencia creciente de largo $\ln n$

El problema consiste en dada una secuencia de números, encontrar una subsecuencia (no necesariamente consecutiva) de elementos crecientes de largo al menos $\ln n$.

Una manera simple de hacer esto es buscar el máximo de un arreglo y guardar la subsecuencia de los elementos intercambiados al buscar el máximo. Observación: la cantidad de intercambios que se hacen en promedio para encontrar el máximo en un arreglo de tamaño n es al menos $\ln n$, y viene de resolver la recurrencia: $T(n) = \frac{1}{n} + T(n-1) = \sum_{i \le n} \frac{1}{i} = H_n \approx \ln n$.

Para hacer esto en forma aleatoria hay varias opciones, una solución tipo Montecarlo es:

```
MCSubSecCreciente (Arreglo A)
  n ← |A|
  inicio ← Random(0, n/4) // con esto estoy usando un arreglo de tamaño 3n/4,
      y la secuencia será casi de tamaño ln n (en realidad es ln n + ln 3 − ln 4)
  subsec ← {A[inicio]}
  max ← A[inicio]
  for i ← inicio ... N do
    if A[i] > max then
      subsec ← subsec ∪ A[i]
      max ← A[i]
  return subsec
```

y una solución tipo Las Vegas es:

```
LVSubSecCreciente (Arreglo A)
  while (1)
    subsec ← MCSubSecCreciente(A)
    if (|subsec| ≥ ln n) return subsec
```