Pontificia Universidad Católica de Chile
Departamento de Ciencia de la Computación
IIC2133 - Estructuras de Datos y Algoritmos

Ayudantía 1

21 de agosto de 2020

Pregunta 1

BogoSort es un algoritmo de ordenación, el cual recibe como parametro un conjunto A de n elementos:

```
BogoSort(A, n):
1:  While (true):
2:    sea B un arreglo vacío
3:    While (A aún tenga elementos):
4:      Extraer un elemento aleatoriamente de A y ponerlo al final de B
5:    If B está ordenado:
6:      return B
7:    Devolver todos los elementos de B a A
```

Demostrar correctitud o no correctitud según sea el caso.

Pregunta 2

Se tiene el siguiente algoritmo para encontrar el menor y mayor elemento en un conjunto A de n elementos **distintos**. Por simplicidad, consideraremos que n **es un número par**.

1. Definimos el conjunto m, que contendrá todos los elementos que no han ganado* una comparación.
2. Definimos el conjunto M, que contendrá todos los elementos que no han perdido* una comparación.
3. Extraemos dos elementos arbitrarios de A, y los comparamos. Ponemos al ganador de la comparación en M, y al perdedor en m
4. Si aun quedan pares de elementos en A, volver al paso 3.
5. Extraemos un elemento arbitrario de m y lo definimos como min. Luego recorremos todo el conjunto m, comparando cada elemento φ con min, guardando el perdedor en min
6. Extraemos un elemento arbitrario de M y lo definimos como max. Luego recorremos todo el conjunto M, comparando cada elemento ψ con max, guardando el ganador en max

Al finalizar el algoritmo, el máximo y el mínimo de A estarán contenidos en las variables min y max respectivamente

* En una comparación entre dos elementos, diremos que el menor "pierde" la comparación, mientras que el mayor "gana" la comparación.

Demostrar correctitud y complejidad del algoritmo.

Solución

Pregunta 1

Correctitud:

Para demostrar que un algoritmo es correcto debemos demostrar dos cosas:

1. Para todo input finito, el algoritmo termina en una cantidad finita de pasos.
2. En caso de terminar, el algoritmo calcula correctamente lo que dice calcular.

El algoritmo es finito

El algoritmo *BogoSort* tiene dos ciclos *while*, en las lineas 1) y 3), por lo cual, si ambos terminan en una cantidad finita de pasos, el algoritmo también será finito.

En primer lugar, el *while* de la línea 3) tiene como condición de termino que el conjunto A este vacío, entrando inicialmente al ciclo con n elementos, pero cada vez que se ejecuta 4), se le quita un elemento aleatoria a A y es agregado al final de B, por lo cual, dado que A es finito con n elementos, son necesarias n ejecuciones de 4) para vaciar A, lo que implica que el ciclo *while* de 3) es finito.

En segundo lugar, el *while* de la línea 1) depende su termino de 5), dado que, si es correcto que B queda ordenado, se ejecuta la línea 6), y se termina el algoritmo, saliendo del ciclo. Por otra parte, Como B siempre es formado como una permutación aleatoria de A, se cumple que la probabilidad de que B este ordenado es $\frac{1}{|P(A)|}$, con $P(A)$ el conjunto de todas las permutaciones de A. Finalmente como $\frac{1}{|P(A)|} < 1$, es imposible garantizar que la condición de 5) se cumplirá, por lo cual el *while* de 1) no es necesariamente finito.

$\therefore$ como el *while* de la línea 1) no es necesariamente finito, el algoritmo *BogoSort* no garantiza termino, implicando que el algoritmo no es correcto.

Pregunta 2

Correctitud:

El algoritmo es finito

Demostración. Para esto debemos demostrar que cada paso es finito:

- Los pasos 1 y 2 son una declaración por lo que no agregan pasos al algoritmo.
- Los pasos 3 y 4 se repiten mientras $|A| > 0$. Ya que inicialmente $|A| = 2k = n$, y cada vez que se ejecuta el paso 3, $|A|$ decrece en 2, eventualmente $|A| = 0$ (luego de k ejecuciones del paso 3). Esto significa que tras una cantidad finita de repeticiones, pasamos al siguiente paso.
- Los pasos 5 y 6 ambos recorren por completo los conjuntos m y M respectivamente, sin repetir elementos. Como ambos conjuntos tienen una cantidad finita de elementos (k), ambos pasos son finitos.

Ya que el algoritmo está compuesto de puros pasos finitos, el algoritmo es finito. □

El algoritmo entrega el resultado correcto.

Demostración. Para esto demostraremos por separado que tanto el mínimo como el máximo se calculan correctamente.

Ya que A sufre modificaciones a lo largo del algoritmo, definimos el conjunto B como el A original provisto al algoritmo.

En primer lugar, sabemos que si $|B| > 0$, entonces existe un mínimo y un máximo en B. Esto se asume como correcto ya que $A = \varnothing$ es un input inválido para el algoritmo.

Para demostrar que este algoritmo entrega el resultado correcto, primero es necesario demostrar que luego de los pasos 3 y 4, el mínimo está en m y el máximo está en M.

Demostraremos cada uno por separado.

Por demostrar: $min(B) \in m$

Demostración. Lo demostraremos por contradicción.

Asumamos que el $min(B) \notin m$. Por lo tanto, $min(B) \in M$.

Esto significa que en el paso 3, cuando extrajimos $min(B)$ y lo comparamos, este ganó la comparación. Por lo tanto, existía otro número $z \in B$ tal que $z < min(B)$. Por definición de mínimo, $\forall b \in B(min(B) \leq b)$ y por lo tanto, llegamos a una contradicción $\rightarrow\leftarrow$. □

Por demostrar: $max(B) \in M$

Demostración. La demostración para este caso es análoga. □

Solo una vez habiendo demostrado esto es que tiene sentido buscar el mínimo y el máximo en m y M respectivamente (pasos 5 y 6). Demostraremos cada uno por separado.

Por demostrar: Al final del paso 5, $min \equiv min(B)$

Sea $k = |\mathbf{m}|$.

Por inducción

BI: Si $k = 1$, se retorna el mismo elemento como menor, por lo que el algoritmo es correcto para el caso base.

HI: Al recorrer un un conjunto con k elementos con este algoritmo min es el menor valor del mismo.

TI: Al recorrer un conjunto con $k + 1$ elementos, min es el menor valor del mismo.

Para recorrer el conjunto de $k + 1$ elementos, primero se recorre un subconjunto de k elementos. Por hipótesis de inducción se obtiene el mínimo de este subconjunto, llamemoslo m. Luego, frente al elemento $k + 1$ se tienen dos casos:

1. El elemento $k + 1$ es menor que m, por lo que el valor mínimo se actualizaría, y por transitividad $k + 1$ sería mínimo del conjunto de $k + 1$ elementos.
2. El elemento $k + 1$ es mayor que m, se mantiene el valor mínimo. Y ya que m es menor que $k + 1$, se cumple que se encontró el mínimo del conjunto de $k + 1$ elemento.

$\therefore$ Se puede concluir que se retorna el valor mínimo del conjunto $\mathbf{m}$.

Al igual que antes, la demostración de que al final del paso 6, $max \equiv max(B)$ es análoga.

De esta forma podemos decir que el algoritmo es correcto.

Complejidad

Por lo que vimos en la demostración de correctitud, se realizan $\frac{n}{2}$ comparaciones, quedando la misma cantidad en $\mathbf{m}$ y $\mathbf{M}$.

Después para obtener el mínimo y máximo en cada uno de los conjuntos se realizan $\frac{n}{2} - 1$ comparaciones, ya que inicialmente se extrae un elemento de cada conjunto.

De esta forma, la cantidad de comparaciones hechas en función del largo del conjunto original es: $T(n) = \frac{n}{2} + \left(\frac{n}{2} - 1\right) + \left(\frac{n}{2} - 1\right) = 3 \cdot \frac{n}{2} - 2 \in \Theta(n)$

□