Análisis de Algoritmos II

Jesús Alexander Aranda Ph.D Robinson Duque, Ph.D
Juan Francisco Díaz, Ph. D

Universidad del Valle

jesus.aranda@correounivalle.edu.co
robinson.duque@correounivalle.edu.co
juanfco.diaz@correounivalle.edu.co

Programa de Ingeniería de Sistemas
Escuela de Ingeniería de Sistemas y Computación

Overview

Presentación

- De los profesores y el (la) monitor(a)
- De los estudiantes $<$ *Socrative* $>$
- Del curso campusvirtual.univalle.edu.co:
  - Competencias a las que tributa
  - Resultados de aprendizaje
  - Indicadores de logro y sistema de evaluación

Presentación

- De los profesores y el (la) monitor(a)
- De los estudiantes $<$ *Socrative* $>$
- Del curso campusvirtual.univalle.edu.co:
  - Competencias a las que tributa
  - Resultados de aprendizaje
  - Indicadores de logro y sistema de evaluación

Presentación

- De los profesores y el (la) monitor(a)
- De los estudiantes $< \textit{Socrative} >$
- Del curso campusvirtual.univalle.edu.co:
  - Competencias a las que tributa
  - Resultados de aprendizaje
  - Indicadores de logro y sistema de evaluación

Motivación

Considere el siguiente problema:

El problema del cambio de monedas

Entrada: $A = \{a_1, a_2, \ldots, a_n\}$, $a_i \in \mathbb{N}, a_i < a_{i+1}, i \in 1..(n-1)$, y $P \in \mathbb{N}$

Salida:$(m_1, m_2, \ldots, m_n)$, $m_i \in \mathbb{N}, i \in 1..(n-1)$, y $\sum_{i=1}^{n} m_i a_i = P$ y $\sum_{i=1}^{n} m_i$ es mínima.

Considere los siguientes dos algoritmos:

fun *CambioDeMonedas1*(A, P)

- $i = n, p = P$
- **mq** $i > 0$ **haga**
  $m_i = p \div a_i, p = p - m_i * a_i, i = i - 1$
- **return** $(m_1, \ldots, m_n)$

fun *CambioDeMonedas2*(A, P)

- $Sum = \infty, M = (P \div a_1, P \div a_2, \ldots, P \div a_n)$
- **para cada** $(m_1, m_2, \ldots, m_n)$ **tal que**
  $0 \leq m_i \leq M_i \wedge \sum_{i=1}^{n} m_i a_i = P$ **haga**
  si $Sum > \sum_{i=1}^{n} m_i$ **ent** $Sum = \sum_{i=1}^{n} m_i, m^* = m$
- **return** $(m_1^*, \ldots, m_n^*)$

¿Son correctos los dos? ¿Cuál es su eficiencia?

< *Socrative* >

Motivación

Considere el siguiente problema:

El problema del cambio de monedas

Entrada: $A = \{a_1, a_2, \ldots, a_n\}$, $a_i \in \mathbb{N}, a_i < a_{i+1}, i \in 1..(n-1)$, y $P \in \mathbb{N}$

Salida:$(m_1, m_2, \ldots, m_n)$, $m_i \in \mathbb{N}, i \in 1..(n-1)$, y $\sum_{i=1}^{n} m_i a_i = P$ y $\sum_{i=1}^{n} m_i$ es mínima.

Considere los siguientes dos algoritmos:

fun *CambioDeMonedas*1(A, P)

- $i = n, p = P$
- **mq** $i > 0$ **haga**
  $m_i = p \div a_i, p = p - m_i * a_i, i = i - 1$
- **return** $(m_1, \ldots, m_n)$

fun *CambioDeMonedas*2(A, P)

- $Sum = \infty, M = (P \div a_1, P \div a_2, \ldots, P \div a_n)$
- **para cada** $(m_1, m_2, \ldots, m_n)$ **tal que**
  $0 \leq m_i \leq M_i \wedge \sum_{i=1}^{n} m_i a_i = P$ **haga**
  si $Sum > \sum_{i=1}^{n} m_i$ **ent** $Sum = \sum_{i=1}^{n} m_i, m^* = m$
- **return** $(m_1^*, \ldots, m_n^*)$

¿Son correctos los dos? ¿Cuál es su eficiencia?

< *Socrative* >

Motivación

Considere el siguiente problema:

El problema del cambio de monedas

Entrada: $A = \{a_1, a_2, \ldots, a_n\}$, $a_i \in \mathbb{N}, a_i < a_{i+1}, i \in 1..(n-1)$, y $P \in \mathbb{N}$

Salida:$(m_1, m_2, \ldots, m_n)$, $m_i \in \mathbb{N}, i \in 1..(n-1)$, y $\sum_{i=1}^{n} m_i a_i = P$ y $\sum_{i=1}^{n} m_i$ es mínima.

Considere los siguientes dos algoritmos:

fun *CambioDeMonedas*1(A, P)

- $i = n, p = P$
- **mq** $i > 0$ **haga**
  $m_i = p \div a_i, p = p - m_i * a_i, i = i - 1$
- **return** $(m_1, \ldots, m_n)$

fun *CambioDeMonedas*2(A, P)

- $Sum = \infty, M = (P \div a_1, P \div a_2, \ldots, P \div a_n)$
- **para cada** $(m_1, m_2, \ldots, m_n)$ **tal que**
  $0 \leq m_i \leq M_i \wedge \sum_{i=1}^{n} m_i a_i = P$ **haga**
  si $Sum > \sum_{i=1}^{n} m_i$ **ent** $Sum = \sum_{i=1}^{n} m_i, m^* = m$
- **return** $(m_1^*, \ldots, m_n^*)$

¿Son correctos los dos? ¿Cuál es su eficiencia?

$<$ *Socrative* $>$

Motivación

- Existen problemas en la vida real que tienen soluciones computacionales eficientes (problemas fáciles) y otros para los que aún no se ha encontrado ninguna solución computacional eficiente (problemas difíciles).
- ¿Se pueden caracterizar los problemas fáciles? Complejidad $\mathcal{O}(n^k)$
- ¿Se pueden caracterizar los problemas difíciles? Teoría de NP-completitud
- Toda la evidencia actual sugiere que no son fáciles, pero no está demostrado.
- Si lo anterior es cierto, no podremos solucionar este tipo de problemas de forma eficiente con los computadores actuales (i.e., basados en máquinas de Turing deterministas)

Motivación

- Existen problemas en la vida real que tienen soluciones computacionales eficientes (problemas fáciles) y otros para los que aún no se ha encontrado ninguna solución computacional eficiente (problemas difíciles).
- ¿Se pueden caracterizar los problemas fáciles? Complejidad $\mathcal{O}(n^k)$
- ¿Se pueden caracterizar los problemas difíciles? Teoría de NP-completitud
- Toda la evidencia actual sugiere que no son fáciles, pero no está demostrado.
- Si lo anterior es cierto, no podremos solucionar este tipo de problemas de forma eficiente con los computadores actuales (i.e., basados en máquinas de Turing deterministas)

Motivación

- Existen problemas en la vida real que tienen soluciones computacionales eficientes (problemas fáciles) y otros para los que aún no se ha encontrado ninguna solución computacional eficiente (problemas difíciles).
- ¿Se pueden caracterizar los problemas fáciles? Complejidad $\mathcal{O}(n^k)$
- ¿Se pueden caracterizar los problemas difíciles? Teoría de NP-completitud
- Toda la evidencia actual sugiere que no son fáciles, pero no está demostrado.
- Si lo anterior es cierto, no podremos solucionar este tipo de problemas de forma eficiente con los computadores actuales (i.e., basados en máquinas de Turing deterministas)

Motivación

- Existen problemas en la vida real que tienen soluciones computacionales eficientes (problemas fáciles) y otros para los que aún no se ha encontrado ninguna solución computacional eficiente (problemas difíciles).
- ¿Se pueden caracterizar los problemas fáciles? Complejidad $\mathcal{O}(n^k)$
- ¿Se pueden caracterizar los problemas difíciles? Teoría de NP-completitud
- Toda la evidencia actual sugiere que no son fáciles, pero no está demostrado.
- Si lo anterior es cierto, no podremos solucionar este tipo de problemas de forma eficiente con los computadores actuales (i.e., basados en máquinas de Turing deterministas)

Motivación

- Existen problemas en la vida real que tienen soluciones computacionales eficientes (problemas fáciles) y otros para los que aún no se ha encontrado ninguna solución computacional eficiente (problemas difíciles).
- ¿Se pueden caracterizar los problemas fáciles? Complejidad $\mathcal{O}(n^k)$
- ¿Se pueden caracterizar los problemas difíciles? Teoría de NP-completitud
- Toda la evidencia actual sugiere que no son fáciles, pero no está demostrado.
- Si lo anterior es cierto, no podremos solucionar este tipo de problemas de forma eficiente con los computadores actuales (i.e., basados en máquinas de Turing deterministas)

Motivación

Como ingenieros de sistemas, podemos encontrarnos con diversos problemas computacionales que:

- Son reales, importantes, y de optimización
- Son fáciles de entender y, aún, de especificar, pero . . .
- No somos capaces de solucionarlos eficientemente con un computador
- Los algoritmos naturales que resuelven el problema son ineficientes (e.g., consumen mucha memoria o tardan mucho o ambas cosas)

En este curso estudiaremos cómo reconocer si el problema al que nos enfrentamos es difícil y, en caso de ser de optimización cómo construir soluciones prácticas, es decir, eficientes en tiempo de ejecución, cuando el problema lo permite, utilizando algunas técnicas de optimización.

Motivación

Como ingenieros de sistemas, podemos encontrarnos con diversos problemas computacionales que:

- Son reales, importantes, y de optimización
- Son fáciles de entender y, aún, de especificar, pero ...
- No somos capaces de solucionarlos eficientemente con un computador
- Los algoritmos naturales que resuelven el problema son ineficientes (e.g., consumen mucha memoria o tardan mucho o ambas cosas)

En este curso estudiaremos cómo reconocer si el problema al que nos enfrentamos es difícil y, en caso de ser de optimización cómo construir soluciones prácticas, es decir, eficientes en tiempo de ejecución, cuando el problema lo permite, utilizando algunas técnicas de optimización.

Motivación

Como ingenieros de sistemas, podemos encontrarnos con diversos problemas computacionales que:

- Son reales, importantes, y de optimización
- Son fáciles de entender y, aún, de especificar, pero ...
- No somos capaces de solucionarlos eficientemente con un computador
- Los algoritmos naturales que resuelven el problema son ineficientes (e.g., consumen mucha memoria o tardan mucho o ambas cosas)

En este curso estudiaremos cómo reconocer si el problema al que nos enfrentamos es difícil y, en caso de ser de optimización cómo construir soluciones prácticas, es decir, eficientes en tiempo de ejecución, cuando el problema lo permite, utilizando algunas técnicas de optimización.

Motivación

Como ingenieros de sistemas, podemos encontrarnos con diversos problemas computacionales que:

- Son reales, importantes, y de optimización
- Son fáciles de entender y, aún, de especificar, pero ...
- No somos capaces de solucionarlos eficientemente con un computador
- Los algoritmos naturales que resuelven el problema son ineficientes (e.g., consumen mucha memoria o tardan mucho o ambas cosas)

En este curso estudiaremos cómo reconocer si el problema al que nos enfrentamos es difícil y, en caso de ser de optimización cómo construir soluciones prácticas, es decir, eficientes en tiempo de ejecución, cuando el problema lo permite, utilizando algunas técnicas de optimización.

Motivación

Como ingenieros de sistemas, podemos encontrarnos con diversos problemas computacionales que:

- Son reales, importantes, y de optimización
- Son fáciles de entender y, aún, de especificar, pero ...
- No somos capaces de solucionarlos eficientemente con un computador
- Los algoritmos naturales que resuelven el problema son ineficientes (e.g., consumen mucha memoria o tardan mucho o ambas cosas)

En este curso estudiaremos cómo reconocer si el problema al que nos enfrentamos es difícil y, en caso de ser de optimización cómo construir soluciones prácticas, es decir, eficientes en tiempo de ejecución, cuando el problema lo permite, utilizando algunas técnicas de optimización.