Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Computación

FACULTAD DE CIENCIAS
FÍSICAS Y MATEMÁTICAS
UNIVERSIDAD DE CHILE

Matemáticas Discretas para la Computación

Informe de Tarea 1

Autor: Miguel Sepúlveda

Profesor: Gonzalo Navarro

Auxiliares: Ian Letter
Pablo Torres D.

Fecha de realización: 26 de septiembre de 2017
Fecha de entrega: 26 de septiembre de 2017
Santiago, Chile

Índice de Contenidos

1. Pregunta 1

1.1. Correctitud del algoritmo

Se verificará la corrector del algoritmo demostrando la frase .[El] algoritmo devuelve el vuelto correcto para X"haciendo inducción fuerte sobre X. Con $X = 1$, el algoritmo disminuirá j hasta que $j = 1$, pues ese es el máximo j tal que $X - m_j \geq 0$. Así, el algoritmo entrega solo la moneda 1, cambia X por 0, y termina el programa, por lo que se entrega el vuelto correcto.

Para el paso inductivo, veamos que el algoritmo es correcto para $X > 1$. Si $X - m_k < 0$, entonces el algoritmo disminuirá j hasta que $X - m_j \geq 0$; en este caso, j es el mayor índice tal que ocurre esto, y su existencia esta garantizada pues $X > 1$. Si no, entonces $j = n$. Así, el algoritmo entrega la moneda j y cambia X por $X - m_j$, por por hipótesis inductiva, el algoritmo entrega una cantidad de vuelto correcta para $X - m_j$. Así, este vuelto junto con la moneda j constituyen un vuelto correcto para X, por lo que se concluye que el algoritmo es correcto. □

1.2. Vuelto óptimo

Tomando las hipótesis indicadas, es posible demostrar que el algoritmo es óptimo para un vuelto X usando otra vez inducción fuerte sobre X. Si $X = 1$, el algoritmo solo entrega la moneda 1 y se termina, lo cual es el óptimo (0 monedas no pueden sumar 1).

Si $X > 1$, sea j el máximo índice tal que $X - m_j \geq 0$. El algoritmo siempre llega a este índice sin modificar X ya que las monedas están ordenadas de manera creciente, y existe pues $X > 1$ y $m_1 = 1$. Ahora demostraremos que la moneda j debe estar en la solución óptima utilizando una propiedad de la misma:

Propiedad: En un vuelto óptimo no pueden haber más de $\frac{m_{i+1}}{m_i} - 1$ monedas del tipo i.

Demostración: En efecto, de no ser así, primero notemos que $\frac{m_{i+1}}{m_i}$ es un número entero por la hipótesis del enunciado, luego, tomando el mismo vuelto pero intercambiando $\frac{m_{i+1}}{m_i}$ de las monedas i por una moneda $i+1$, y como $m_{i+1} > m_i$, entonces se tendrá una solución con menor cantidad de monedas que el óptimo, lo cual es una contradicción. □

Ahora, notemos que ninguna moneda $i > j$ puede estar en la solución pues $m_j \leq X < m_i$, así que la solución óptima solo puede estar compuesta de las monedas $1 \leq i \leq j$. Si la moneda j no se encuentra en la solución, entonces el mayor valor que puede sumar un vuelto que cumpla la propiedad recién demostrada es $m_j - 1$. En efecto, si tomamos la máxima cantidad de cada tipo de monedas $1 \leq i < j$ tales que cumplan la propiedad, entonces el valor total será:

$$\left(\frac{m_2}{m_1} - 1\right) m_1 + \left(\frac{m_3}{m_2} - 1\right) m_2 + \cdots + \left(\frac{m_{j-1}}{m_{j-2}} - 1\right) m_{j-2} + \left(\frac{m_j}{m_{j-1}} - 1\right) m_{j-1} = m_j - 1$$

La igualdad se debe a que es una suma telescópica. Así, como $X \geq m_j > m_j - 1$, el vuelto óptimo debe tener a la moneda j en él. Así, por hipótesis inductiva, se argumenta que las monedas entregadas por el algoritmo para $X - m_j$ también deben ser parte del óptimo. Se concluye entonces que, con las hipótesis dadas, el algoritmo entrega el vuelto óptimo para todo X. □

Si las monedas no cumplen que m_i divide a m_{i+1} para todo $1 \leq i < k$, entonces podemos tomar $m_2 = 2$, $m_3 = 7$ y $m_4 = 11$, y $X = 21$. El algoritmo entrega las monedas $\{4, 3, 2, 1\}$, pero es posible entregar un vuelto solo con tres monedas de valor 7, por lo que el algoritmo no entrega el óptimo.

1.3. Programa

Para hacer el programa pedido, se utiliza el lenguaje `C++`. Se programa la función `Kioskito`, que recibe un entero X, el vuelto a pedir, un arreglo *coins*, que contiene el valor de la moneda i en su índice $i-1$, y k, que es el tamaño de este arreglo. Como se tiene que el algoritmo de este problema entrega el óptimo para el input que se recibirá, entonces se utiliza para programar esta función, con un par de modificaciones.

Se agrega un contador **c**, que se inicializa en 0, aumenta cada vez que se entrega una moneda, y se reinicia cada vez que se disminuye el valor de j, donde ya no se entregarán mas monedas de este tipo. Así, cuando $X - m_j < 0$ (Condición para disminuir j), el contador contendrá el número de monedas j que contendrá el vuelto final. El programa así muestra este valor y luego reinicia el contador y disminuye j a $j-1$.

1.3.1. Modo de uso

El usuario debe ingresar el número de tipos de monedas que hay, luego el programa pedirá ingresar el valor de cada una (El valor de la moneda 1 es siempre 1 y no se pide), y finalmente pide el valor del vuelto que se debe entregar. La cantidad entregada de cada moneda se muestra en pantalla.

2. Pregunta 2

2.1. Observaciones iniciales

Primero que nada, es importante notar que significa que una relación sea un cuasi-orden para su representación como una matriz. Una relación es refleja si y solo si $A_{ii} = 1$, para todo i, pues eso es equivalente con decir que $(a_i, a_i) \in R$, para todo $i \geq n$. Y es transitiva si y solo si para todos todos los índices $i, j, k \geq n$ tales que $A_{ij} = A_{jk} = 1$, entonces $A_{ik} = 1$, pues esto es equivalente a que para todos los elementos a_i, a_j, a_k tales que $(a_i, a_j) \in R$ y $(a_j, a_k) \in R$, se tiene que $(a_i, a_k) \in R$, lo que es exactamente la transitividad.

2.2. `esCuasiOrden`

Para detectar si una función es cuasi-orden, se programa la función `esCuasiOrden` en Python, pues este tiene la biblioteca NumPy que permite trabajar fácilmente con matrices. Esta función recibe una matriz de NumPy y revisa si representa un cuasi-orden.

Para esto, la función recorre la matriz elemento a elemento. Si algun elemento diagonal es 0, entonces la relación no es refleja y devuelve False. Si algún elemento A_{ij} no diagonal es 1, entonces recorre todos los elemento de la columna j, si alguno de estos elementos A_{jk} es 1, entonces para que la relación sea transitiva la casilla A_{ik} debe ser 1. Si esto no ocurre, entonces la relación no es transitiva y regresa False. Si recorre toda la matriz sin terminar, entonces esta si representa un cuasi-orden y la función retorna True, pues la relación que representa cumple todas las propiedades de un cuasi-orden

2.3. Mayor Relación de equivalencia

Para demostrar que $R \cap R^{-1}$ es la mayor relación de equivalencia contenida en R, primero notemos que en efecto está contenida en R, pues es una intersección entre R y otro conjunto. Demostremos ahora que es una relación de equivalencia:

Reflexiva: Como R es refleja, entonces todo elemento $a \in A$ está en R y en R^{-1}, por definición, por lo que se tiene que $(a, a) \in R \cap R^{-1}$. Así, esta es una relación reflexiva.

Simétrica: sea $(a, b) \in R \cap R^{-1}$. Por definición, esto significa que $(a, b) \in R$ y que $(a, b) \in R^{-1}$. Por definición de R^{-1}, entonces $(b, a) \in R^{-1}$ y $(a, b) \in R$, por lo que $(b, a) \in R \cap R^{-1}$. Se concluye entonces que la relación es simétrica.

Transitiva: Sean $a, b, c \in A$ tales que $(a, b) \in R^{-1}$ y $(b, a) \in R \cap R^{-1}$. Por definición, se tiene que (a, b) R, $(b, a) \in R$, $(b, c) \in R$ y $(c, b) \in R$. Por transitividad de R, tenemos que $(a, c) \in R$ y $(c, a) \in R$, por lo que $(a, c) \in R \cap R^{-1}$, así que la relación es transitiva.

De esto se concluye que la relación es de equivalencia. Para demostrar que es la mayor relación de equivalencia contenida en R, tomamos una relación de equivalencia $\overline{R} \subset R$, la cual existe pues $R \cap R^{-1}$ es una de ellas. Esta relación no puede ser vacía pues de ser así no seria refleja. Así, tomamos un elemento $(a, b) \in \overline{R} \subset R$. Como $\overline{R}$ es de equivalencia, es simétrica, por lo que $(b, a) \in \overline{R} \subset R$, pero esto significa que $(a, b) \in R^{-1}$, por

lo que se tiene que $(a, b) \in R \cap R^{-1}$. Se concluye entonces que $\overline{R} \subset R$, es decir, que toda relación de equivalencia contenida en R está contenida también en $R \cap R^{-1}$, que es exactamente lo que queriamos demostrar. □

2.4. `mayorEquivalencia`

La función `mayorEquivalencia` recibe una matriz representante de un cuasi-orden R, y entrega la mayor la relación de equivalencia contenida en esta. Para esto solo basta crear la matriz representante para $R \cap R^{-1}$, pues en la sección anterior se demostró que esta es la relación buscada. Así, el programa crea una nueva matriz del mismo tamaño que R llena de ceros, recorre la matriz de R, y si encuentra un uno en una casilla A_{ij} y un uno en la casilla A_{ji} (Lo que significa que $(a_i, a_j) \in R$ y $(a_i, a_j) \in R^{-1}$, entonces coloca un 1 en la casilla A_{ij} de la matriz recién creada. Cuando el programa termina de recorrer esta matriz, devuelve la creada y el programa se termina.

2.5. Modo de uso

El programa pide primero el número de elementos del conjunto donde se define la relación, y luego se deben ingresar los índices de todos los pares que se encuentran en ella. Después de ingresar el último par el usuario debe ingresar un 0. El programa entonces mostrará en pantalla si la relación corresponde a un cuasi-orden o no, y si lo es, muestra en pantalla la matriz representante mayor relación de equivalencia con el cuasi-orden.

3. Pregunta 3

3.1. Descomposición en números de Fibonacci

Para demostrar que todo número natural se puede descomponer en una suma de números de Fibonacci donde no hay dos de estos consecutivos, se utilizará inducción fuerte para demostrar que n se puede descomponer de esta manera.

Para el caso base, con $n = 1$, basta tomar $F_0 = 1$. Para el paso inductivo, tomando $n > 1$, podemos definir un conjunto que contenga las diferencias entre n y todos los números de Fibonacci menores que n, y como $n > 1$, entonces este conjunto es no vació, y todos sus elementos son naturales. Por el principio del buen orden, existe un número de Fibonacci tal que esta diferencia es mínima, llamémoslo F_k. Así, tendremos que existe un entero $c \geq 0$ tal que $n = F_k + c$. Si $c = 0$, n es un número de Fibonacci y se tiene lo pedido, en el caso contrario, por hipotesis inductiva c puede ser descompuesto en una suma de números de Fibonacci donde no hay dos números de Fibonacci consecutivos.

Como F_k es el máximo número de Fibonacci tal que $n \geq F_k$, entonces $F_{k+1} > n = F_k + c$. Por esto, c no puede contener a F_{k+1} en su descomposición, y tampoco a F_{k-1}, pues de ser así, entonces existe un entero $c' \geq 0$ tal que $c = F_{k-1} + c'$, así, $F_{k+1} > n = F_k + F_{k-1} + c' = F_{k+1} + c'$, lo que es una contradicción. Así, la descomposición de c no contiene a los vecinos de F_k. Se concluye entonces que n se puede descomponer de la forma que buscábamos demostrar. □

3.2. Función `fibonacci`

La función `fibonacci` se programó en Python debido a la facilidad que entrega al trabajar con arreglos de tamaño dinámico. El programa recibe un número natural **n** y muestra la representación binaria pedida en el enunciado en pantalla; para esto, se utiliza un algoritmo avaro que toma el mayor número de Fibonacci F_k que sea menor a **n**, lo incluye, y cambia n por $n - F_K$, este funciona pues es el usado durante la demostración anterior.

Así, lo primero que hace la función crear una lista con los primeros dos números de Fibonacci, y genera en este arreglo el resto de los números de Fibonacci hasta el primero mayor a n, utilizando programación dinámica. Luego, inicializa un arreglo donde irá el número binario, con cada dígito en un índice.

Luego, se inicia el loop descrito anteriormente, con una variable auxiliar, llamemosla i, que índica el último número de Fibonacci revisado en la lista: Si $n - F_i \geq 0$, el algoritmo pone un 1 en la posición que le corresponde a F_i, y cambia n por $n - F_i$. Del caso contrario, F_i no se incluye, por lo que se coloca un 0 en su posición, y se descarta de la lista. Finalmente, se muestra el arreglo completo en pantalla. El programa se encuentra en los anexos.

4. Anexos

4.1. P1.cpp

[fragile]

```
#include <stdio.h>

 void kioskito(int X, int coins[], int k)
{
  int j = k;
  int c = 0;
  do
  {
    if (X - coins[j{-1}] >= 0){
      c++;
      X = X-coins[j{-1}];
    }
    else // Se dejan de entregar monedas del mismo valor, se dice cuantas se entregaro
      // Y se resetea el contador
    {
      printf("%d moneda%s de %d peso%s\n", c,
         c==1 ? "" : "s", coins[j{-1}], coins[j{-1}]==1 ? "" : "s");
      if(!X) break; //Si X es 0, terminar
      c = 0;
      j--;
    }
  } while(true);
  return;
}

int main()
{
  int k;
  printf("Ingrese número de monedas: ");
  scanf("%d", &k);
  int coins[k];
  coins[0] = 1;
  int i;
  for(i = 1; i<k; i++)
  {
    printf("Ingrese valor de moneda numero %d: ", i+1);
    scanf("%d", &coins[i]);
  }
  int X;
  printf("Ingrese el valor del vuelto\n");
  scanf("%d", &X);
  printf("Para dar el vuelto se usan\n");
  kioskito(X, coins, k);
}
```

4.2. P2.py

```
#!/usr/bin/env python
import numpy as np
def esCuasiOrden(R) :
   n = np.size(R,0)
   for i in range(0, n) :
      for j in range(0, n) :
         # Primero, revisar si es refleja
         if i==j and R[i,i] == 0:
            return False
         # Si la casilla es 1, revisar la columna de j
         # para ver con quien se relaciona
         elif R[i,j] == 1 :
            for k in range(0, n) :
               if R[j,k]==1 and R[i,k] ==0:
                  print(i," se relaciona con ",j, " y ", j," se relaciona con",k, "
    pero ",i," no se relaciona con ",k)
                  return False
   return True

def mayorEquivalencia(R) :
   n = np.size(R,0)
   ansR = np.zeros((n,n), dtype=np.int)
   for i in range(0, n):
      for j in range(0, n):
         if R[i,j] == 1 and R[j,i] == 1 :
            ansR[i,j] = 1
   return ansR

n = input("Ingrese el numero de elementos del conjunto: ")
print("Ingrese los indices de los pares de elementos \
 que pertenecen a la relacion. Ingrese un 0 para terminar")
R = np.zeros((n,n), dtype=np.int) #Crea array de tamano nxn llena de 0s
while(True):
   i = input("Primer índice: ")
   if i == 0:
      break
   j = input("Segundo índice: ")
   R[i-1,j-1] = 1
if esCuasiOrden(R):
   print("R es es cuasi-orden")
   print("La matriz representante de la mayor relación de equivalencia en R es")
   maxR = mayorEquivalencia(R)
   print(np.transpose(maxR)) # Transpone la matriz para que se vea
               # de la manera estandar
else:
   print("R no es cuasi-orden")
```

4.3. P3.py

```
#!/usr/bin/env python
def fibonacci(n) :
   fib = [1, 1]
   endOfList = len(fib){-1}
   while fib[endOfList] < n :
      fib.append(fib[endOfList]+fib[endOfList{-1}])
```

```
7        endOfList+=1
8     ans = []
9     while endOfList >= 0 :
10       if n - fib[endOfList] >= 0 :
11          ans.append(1)
12          n -= fib[endOfList]
13       else:
14          ans.append(0)
15       endOfList-=1
16    for dig in ans:
17       print dig,
18
19 n = input("Ingrese numero: ")
20 fibonacci(n)
```