Equpamiento Usado

Las pruebas se hicieron en un computador marca Compaq deskpro, con procesador celeron de 500 mhz con 64 MB de ram.

Item 1

Planteamiento del Problema

Determinar los coeficientes del n−ésimo polinomio , definido por la recurrencia.

$Hn(x) = 2 x Hn-1(x) - (n-1)Hn-2(x)$

$H0(x) = 1$

$H1(x) = x$

Aplicando la recurrencia, se obtiene:

$H2(x) = 2 x2 - 1$

$H3(x) = 4 x3 - 4x$

$H4(x) = 8 x4 - 14x2 + 3$

Diseñar un algoritmo recursivo A1 que resuelva el problema y repararlo con PD.

ENTREGAR: Informe escrito con los polinomios hasta n = 20 y el tiempo de ejecución de A1 y A1.PD.

Análisis del problema

Al analizar la recurrencia podemos ver la generación de un árbol binario, que realiza una cantidad de llamadas innecesarias, sin embargo se resuelve el problema planteado, al buscar en Internet, nos hemos encontrado con problemas similares como el polinomio de Hermite el que difiere del problema planteado en el segundo termino.

Análisis de la solución

Hemos desarrollado un algoritmo A1 que utiliza un arreglo para mantener los coeficientes y dos arreglos auxiliares, que mantienen los valores obtenidos de las llamadas recursivas, representando uno de ellos las llamadas por el lado izquierdo y el otro por el lado derecho, luego procedemos a realizar la operación entre los valores obtenidos, que finalmente nos da los coeficientes del polinomio planteado.

Al querer reparar el algoritmo A1 se intentó utilizar un tercer arreglo booleano para identificar si se había realizado el calculo del n correspondiente, sin embargo nos encontramos con problemas de distintas naturalezas. Finalmente obtamos cambiar el arreglo de los coeficientes, por una matriz de n x n que mantiene los coeficientes de cada n, haciendo innecesario el recalculo para los nodos que ya fueron calculados, manteniendo un arreglo booleano para este efecto. Llamamos al algoritmo resultante A2.

Arbol Algoritmo A1

4

3 2

2 2 1 0

1 0 1 0

Arbol Algoritmo A2

4

3

2

1 0

Algoritmo A1

```
uses dos,crt;
Const
Max=30;
type
polinomio = array [0..Max] of integer;
Arreglo = Array [0..Max] of Boolean;
cadena=longint;
Archivo=text;
w=string;
Tiempo=Record h,m,s,c : word;
end;
Var
a:Arreglo;
f:Text;
t:polinomio;
```

```
n,kl:integer;
tm:tiempo;
function cs (var tm:tiempo):longint;
begin
with tm do cs:=((h*60+m)*60+s)*100+c;
end;
Procedure Poli1(Var p:polinomio;n:Integer);
var
q,r : polinomio;
i :integer;
Begin
If n = 0 Then
p[0] := 1
Else
If n = 1 Then
Begin
p[0] := 0;p[1] := 1;
End
Else Begin
Poli1(q,n−2);
Poli1(r,n−1);
p[0] := (n−1)*−q[0];
For i := 1 To n−2 Do
begin
p[i] := 2 * r[i−1] − (n−1) * q[i];
{ writeln('p[',i,']=', p[i]);}
```

```
end;
p[n−1] := 2 * r[n−2];
p[n] := 2 * r[n−1];
End; {EndIf};
{EndIf};
End;
Procedure Mostrar(t:polinomio);
Var
i,y:Integer;
Begin
For i:=0 to n Do
Begin
Write(t[i]);
Write(' ');
End;
Writeln(' ');
End;
Procedure Grabar(t:polinomio);
Var
x,i:Integer;
Begin
For i:=0 to n Do
Begin
Write(f,t[i]);
Write(f,' ');
End;
```

```
Writeln(f,' ');

Writeln(f,'Tiempo ',cs(tm));

End;

Begin {Principal}

clrscr;

SetTime(0,0,0,0);

{ Write('N£mero de Coeficientes: '); Read(n);}

Assign(f,'Poli1.Txt');ReWrite(f);

for n:=1 to Max do

begin

Poli1(t,n);

With tm Do

Begin

GetTime(h,m,s,c);

WriteLn('Tiempo ejec.:', cs(tm));

End;

Mostrar(t);

Grabar(t);

end;

Close(f);

{ Readln(w);{Para pausa en pantalla}

End.
```

Algoritmo A2

```
Program Polinomio2;

Uses

Dos,Crt;
```

```
Const
Max=20;
Type
Matriz = Array [0..Max,0..Max] of longint;
Arreglo = Array [0..Max] of Boolean;
Tiempo = Record
h,m,s,d:Word;
End;
Var
A:Matriz;
B:Arreglo;
n:Integer;
h:Integer;
w:char;
T:Tiempo;
Procedure Grabar(a:Matriz);
Var
f:Text;
i,j:Integer;
Begin
Assign(f,'poli2.Txt');ReWrite(f);
For i:=0 to Max Do
Begin
For j:=0 to Max Do
Begin
Write(f,a[i,j]);
```

```
Write(f,' ');
End;
Writeln(f,' ');
End;
Close(f);
End;
Procedure IniMatriz(var a:Matriz;var b:Arreglo);
Var
i,j:Integer;
Begin
For i:=0 to Max Do
b[i]:=False;
For i:=0 to Max Do
For j:=0 to Max Do
a[i,j]:=0;
End;
Procedure Mostrar(a:Matriz);
Var
i,j:Integer;
Begin
For i:=0 to n Do
Begin
For j:=0 to n Do
Begin
Write(a[i,j]);
Write(' ');
```

```
End;
Writeln(' ');
End;
End;
Procedure Suma(x,y,z:Integer);
Var
k:integer;
Begin
For k:=(z−1) Downto 0 Do
a[z,k+1]:=2 * a[x,k];
For k:=0 to (z−2) Do
a[z,k]:=a[z,k] − (z−1) * a[y,k];
End;
Procedure poli2(var a:Matriz;var b:Arreglo;n,h:Integer);
Begin
If Not b[n] Then
Begin
If n=0 Then
a[n,0]:=1
Else
Begin
If n = 1 Then
Begin
a[n,0]:=0;
a[n,1]:=1;
End
```

```
Else
Begin
Poli2(a,b,n−1,h−1);
Poli2(a,b,n−2,h−2);
Suma(n−1,n−2,n);
End;
End;
b[n]:=true;
writeln('p[',n,']',a[n,0]);
End;
End;
Begin {* Main *}
clrscr;
SetTime(0,0,0,0);
IniMatriz(a,b);
Write( 'inggrese Coeficientes: '); Read(n);
h:=n;
poli2(a,b,n,h);
Mostrar(a);
Grabar(a);
With T Do
Begin
GetTime(h,m,s,d);
End;
Readln(w);{Para pausa en pantalla}
End.
```

Tiempos de Ejecución

En cuanto a los resultado de los tiempos de ejecución, en el caso del algoritmo A1 obtuvimos tiempos mayores a 0 (cero) solo a contar del n=18, para mostrar el gráfico se decidió realizar la prueba hasta n=30. Para el algoritmo A2 solo se obtuvieron tiempos 0 (cero) por lo que no se realizo el análisis correspondiente, sin embargo de esto podemos deducir que al no realizar las llamadas innecesarias el algoritmo es considerablemente mas rápido.

Conclusión

De lo anterior podemos concluir que : en ciertos problemas es trivial realizar el arreglo con PD, no siendo , para nosotros este el caso, ya que tardamos en encontrar una estructura de datos adecuada, pero al encontrarla el algoritmo es considerablemente mas rápido.

Item 2

Planteamiento del Problema

Diseñar un algoritmo de decisión para determinar si un número de la forma 6k−1 ó 6k+1 es primo o no. Usar este algoritmo para diseñar un algoritmo de optimización que determine la densidad de primos en rango 1 a un millón, esto es

Entre 1 y diez 4 primos 40% 1 dígitos 2..7

Entre 1 y cien 25 primos 25% 2 dígito 11.97

Entre 1 y mil 168 primos 16,8% 3 dígito 101..997

Entre 1 y diez mil 1229 primos 12,3% 4 dígito 1009..9973

Entre 1 y cien mil 9592 primos 9,6% 5 dígito 10007..99991

Entre 1 y un millón ¿??? Primos ¿% 6 dígito xxxxxx..yyyyyy

Completar la información y producir un gráfico de barras; indicar el tiempo de ejecución; estimar mediante regresión la cantidad y comparar con el resultado verdadero.

Entregar : informe escrito con el desarrollo, los datos y el gráfico.

Análisis del Problema

Al analizar este problema, hemos visto que se trata de un algoritmo relativamente sencillo que debe intentar realizar una división exacta de un entero determinado, sin embargo al crecer este entero, no encontramos con una cantidad de divisiones bastante grande que hace que los tiempos sean bastante grandes, analizaremos la complejidad temporal a través de las pruebas del algoritmo desarrollado.

Análisis de la solución

Hemos construido un algoritmo que en primera instancia intenta realizar la división exacta por 2 excluyendo inmediatamente a todos los números pares, de esta división se excluido el 2 que obviamente es primo, finalmente si el numero en cuestión ha pasado esta prueba realizamos un ciclo, intentando la división desde 3 en delante, con un incremento de 2 en cada pasado.**Desarrollo del algoritmo:**

```
uses dos,crt;
Const Max=1000000;
type
cadena=longint;
Archivo=text;
w=string;
Tiempo=Record h,m,s,c : word;
end;
Var
f:archivo;
c1,c,n:cadena;
tm:tiempo;
k:boolean;
function cs (var tm:tiempo):longint;
begin
with tm do cs:=((h*60+m)*60+s)*100+c;
end;
function primo(n:cadena):boolean;
var
d:longint;
f:longint;
begin
if ((n mod 2 ) = 0) and (n<>2) then
begin
primo:= false;
writeln('factor : ',n div 2);
```

```
end
else
begin
d:=3;
f:=n div d;
while ((d<f) and (d*f<>n)) do
begin
d:=d+2;
f:=n div d;
if (d*f=n) then
writeln('factor : ',d,' ',f)
end;
primo:=(d*f<>n)
end;
end;
Begin {Principal}
clrscr;
SetTime(0,0,0,0);
c:=0;
c1:=0;
Assign(f,'Primo_dat.Txt');ReWrite(f);
for n:=1 to Max do
begin}
if k then
begin
writeln(n);
```

```
c:=c+1;

end;

writeln(' C : ',c);

With tm Do

Begin

GetTime(h,m,s,c);

WriteLn('Tiempo ejec.:', cs(tm));

End;

Close(f);

End.
```

Resultados (tiempos de ejecución)

N	n	Nro. Primos	Primero	Ultimo	Porcentaje	Tiempo CS
1 − 10	10	4	1	7	40.00%	**0**
1 − 100	100	25	11	97	25.00%	**0**
1 − 1000	1000	168	101	997	16.80%	**0**
1 − 10000	10000	1229	1009	9973	12.29%	**16**
1 − 100000	100000	9592	10007	99991	9.59%	**637**
1 − 1000000	1000000	78498	100003	999983	7.85%	**5311**

Factores :

N 1001 = 7 * 143

N 1003 = 17 * 59

N 1007 = 19 * 53

N 100001 = 11 * 9091

N 100002 = 2 * 50001 , 3 * 33334 , 6 * 16667

¿Cuánto demora la siguiente fila?

Basándonos en el modelo de regresión calculado la siguiente fila se demora:

1,426,806,702,175,960,000,000 CS

es decir aproximadamente **452,437 Millones de años**

Conclusión

En el problema de los numero primos , nos encontramos con un algoritmo bastante sencillo, podemos observar que la densidad de primos en el rango va disminuyendo a medida que crece el n, y que hasta un millón, el algoritmo es sumamente rápido, en comparación con lo que demora, según lo hemos calculado a través del modelo de regresión, el rango siguiente. De las dos observaciones anteriores podemos deducir que en rangos superiores es extremadamente costoso encontrar los primos, además hemos investigado que la seguridad, es decir los algoritmos de criptografía se basan en números primos de los rangos superiores a los que hemos analizado en el problema anterior.

Item 3

Planteamiento del Problema

Reparar el algoritmo Existe del ejercicio 3 de la prueba 2 con programación dinámica, explicar y justificar detalladamente.

- Calcular la complejidad temporal del peor caso
- Realizar pruebas de Ejecución.

Análisis del Problema

Para reparar el algoritmo de la prueba 2 nos basaremos en las conclusiones que se adoptaron en dicha prueba.

En aquel entonces llegamos a la conclusión que para los peores casos (este ocurre cuando no existen valores que sumen la cantidad k, o la suma de todos los valores del arreglo sea k) el algoritmo se comporta de manera exponencial, lo que quiere decir, se demora demasiado tiempo, y obedece a la siguientes ecuación.

$$T = 2^n$$

De acuerdo al análisis visto en la prueba 2, se mostró el recorrido del algoritmo existe, con el cúal pudimos deducir, como funciona éste.

Este algoritmo genera un árbol binario, los siguientes son ejemplos de posibles arboles que se pueden generar con el arreglo a=(20,15,18,11,17,).

K=20, n=1 k=20, n=2 k=35,n=2 k=15,n=2

20 20 35 15

20 35 20 15 0

K=20, n=3 k=35, n=3 k=53,n=3

20 35 53

20

35 53 35

20 35 20

38 35 20

53

K=18, n=3

18

18

18 3

K=57,n=4

57

57 46

57 39 46 28

57 42 39 24 46 31 28 13

Analizando detenidamente el último árbol visto (k=57 y n=4), en cual se muestra un tipo de peor caso (el cual es que el algoritmo no encuentre valores que sumen k), nos podemos dar cuenta que se hacen muchas llamadas demás, y una forma de ahorrarnos llamadas, es dejar en un arreglo auxiliar, la sumatoria de los a[n], es decir dejar aux[n] = a[n]+a[n−1]+.a[1], para luego comparar el aux[n] con el k, si k>aux[n] entonces no es necesario hacer otra llamada, y nuestro algoritmo debiera disminuir el tiempo de ejecución.

Arbol de Llamados para n=4 y k=57 reparado.

57

Aux[64]

57 46

Aux[3]=53

57 46 28

Aux[2]=35

57 46 28 13

Aux[1]=20

Como se puede apreciar la cantidad de llamadas, disminuyó de 15 a 10, para n=4 ahorrandose 5 llamadas, pero cuando n sea más grande, si se dán cuenta las llamadas que se puede ahorrar, corresponde a los nodos de la parte derecha del árbol.

También podemos agregar otra manera de ahorrar llamados, la cual es aplicando la misma técnica, pero ahorrando llamadas por los nodos de la izquierda, lo cual sería, si aux[n] >0 entonces se debiera hacer la misma pregunta que la de la solución anterior (k>Aux[n]), también ahorrando llamadas en forma considerable.

Arbol de Llamados para n=4 y k=57 2da. reparación

57

Aux[64]

57 46

Aux[3]=53

57 28

Aux[2]=35

57 13

Aux[1]=20

Observación: La solución propuesta funciona para valores del arreglo mayores que cero.

Algortimo en lenguaje pascal

```
function existe_d(var a : arreglo;n:longint;k :numero) : boolean;
var hay : boolean;
begin
if (k<0) then
existe_d:=false
else if k=0 then
existe_d:=true
else if n=1 then
begin
aux[n]:=a[n];
existe_d:=a[n]=k;
end
else
begin
if (aux[n]=0) or (k<=aux[n]) then
```

```
hay:=existe_d(a,n−1,k)

else

hay:=false;

aux[n]:=aux[n−1]+a[n];

if (not hay) and (k−a[n]<=aux[n−1]) then

hay := existe_d(a,n−1,k−a[n]);

existe_d:=hay;

end{if};

end;
```

a) Se realizaron varias pruebas, con las pd aplicadas, y el tiempo peor que antes se había definido que ocurría en dos instancias:

- Cuando no existen valores que sumen la cantidad k
- La suma de todos los valores del arreglo sea k

Pasó a ser solo el item 2.

Con respecto a la cantidad de llamadas por ejemplo en el peor caso del algoritmo no reparado, para un arreglo de n=30 y cuya sumatoria era 404, de 1,073,741,824 paso a ser 87, lo cual es demasiado considerable.

Ejecutamos los peores tiempos n desde 1 hasta 16, arrojando la siguientes resultados:

N	**k**	**Llamadas**
1	20	1
2	35	3
3	53	6
4	64	9
5	81	12
6	89	15
7	105	18
8	118	21
9	128	24
10	153	27
11	159	30
12	171	33
13	185	36
14	189	39
15	196	42
16	217	45

La fórmula que indica los llamados varió de :

$T = 2^n$
pasó a ser : $T = 3n - 3$

b) Veamos algunos otras pruebas:

a=(20,15,18,11,17,8,16,13,10,25,6,12,14,4,7,21)

$a[n] = 217$

K	**N**	**Existe**	**Existe con PD**	**Resultado**
57	16	57	45	True
217	16	65535	45	True
218	16	65535	16	False
18	16	18	18	True
1	16	31	31	False
35	16	17	17	True
1000	16	65535	16	False
28	16	32	32	True
46	16	24	23	True

De esta muestra podemos reafirmar que el tiempo peor sucederá cuando el la suma de todos los elementos sea k.

Además podemos observar que incluso los tiempos que no se consideraban peores también han mejorado.

Conclusión

Podemos concluir que el algoritmo propuesto mejora el tiempo de respuesta de una manera tremendamente considerable, y que la instancia del tiempo peor varió de dos a una sola, no sabemos si existe, algún algoritmo más eficiente que el propuesto.

Prueba #3

Análisis de Algoritmos

20

20

20

15

15

15

20

20

$$Y = 73.1258 * e^{0.000C044}$$

20

18

18

15

15

18

15

15

20

20

20

20

20

20

20

15

15

15

20

20

11

18

18

15

15

15

15

20

20

20

20

20

20

20

20

20

11

18

18

15

15

15

20

20

20

20

11

18

18

15

15

20

20

UNIVERSIDAD TECNOLOGICA METROPOLITANA
FACULTAD DE INGENIERIA
DEPARTAMENTO DE
INFORMATICA Y COMPUTACION