Conglomerado de
Algoritmos y Complejidad
Guía máxima de ejercicios y respuestas

Exponential Complexers

24 de septiembre de 2018

Este material fue preparado con lo más cercano a cariño que es capaz de sentir el *staff* del ramo, siendo consistente con los estándares internacionales de maldad.

.

Aunque se han incluído las soluciones a varios ejercicios, es altamente recomendable que antes de mirarlas haga todo lo posible para resolverlos por su cuenta, ya que sólo eso mejorará su capacidad de razonar sobre algoritmos, lo que finalmente se evaluará en los certámenes, más que su memoria.

.

Tenga presente que en la mayoría de los casos no existen técnicas "mecánicas" que permitan resolver cualquier problema y es posible que llegue al mismo resultado de forma muy diferente a la mostrada, si tiene dudas sobre la correctitud de su método, si encuentra un error en una de las soluciones aquí presentadas o si estima que su solución es mejor, coméntelo con el profesor y ayudantes, para mejorar este material y/o el entendimiento de todos.

.

Muchos de estos ejercicios fueron obtenidos y adaptados de los libros[1] en el material complementario del ramo. Así como de material de semestres pasados y de antiguo ramo Análisis de Algoritmos.

.

Este material aun está en una versión preliminar primitiva, y será mejorado gradualmente con el tiempo, disculpe las molestias.

[1] El útil libro de Jeff Erickson http://jeffe.cs.illinois.edu/teaching/algorithms/

Agradecimientos

. . . al staff de ayudantes que ha cooperado en la evolución del contenido y la (no)reprobación de un flujo constante de estudiantes:

Generación 0

Laura Bermeo y Daniel Quinteros [2015-2]

Generación 1

- Francisco Casas [2016-1 . . . 2017-2]
- Francisco Tobar [2016-1]
- Vicente Lizana [2017-1 . . . 2017-2]
- Rodrigo Escar [2017-2 . . . 2018-2]

Generación 2

-
-

Índice

1. Búsqueda de raíces

Ejercicio 1.
Calcule cuantas iteraciones son necesarias para lograr una precisión de k números decimales usando el método de la bisección cuando se parte con el intervalo $[2,7]$.

Ejercicio 2.
Obtenga el orden de convergencia del método de Newton si x^* es un cero doble,vale decir, $f(x^*) = f'(x^*) = 0$, $f''(x^*) \neq 0$.

Ejercicio 3.
Calcular la convergencia lineal del método de la bisección.

Ejercicio 4.
Se busca resolver el siguiente problema $f(x) = x^3 + x - 1 = 0$ para lo cual se descubren las siguientes iteraciones de punto fijo:

$$x = 1 - x^3 = g(x)$$
$$x = \sqrt[3]{1-x} = g(x)$$
$$x = \frac{1+2x^3}{1+3x^2} = g(x)$$

Comprobar si hay convergencia local en las raíces de $f(x)$, que son los puntos fijos de cada una de estas $g(x)$.

Ejercicio 5.
Demostrar que $1,2,3$ son puntos fijos de la siguiente función:

$$\frac{x^3 + x - 6}{6x - 10}$$

Ejercicio 6.
Encontrar el orden y razón de convergencia local de las siguientes funciones para iteración de punto fijo:

$$g(x) = \frac{2x-1}{x^2} \qquad \text{al punto fijo} x = 1$$
$$g(x) = cos(x) + \pi + 1 \qquad \text{al punto fijo} x = \pi$$
$$g(x) = e^{2x} - 1 \qquad \text{al punto fijo} x = 0$$

Ejercicio 7.
Suponga que utiliza el método de la bisección para obtener una *raíz* de $\frac{1}{x}$, con un intervalo $[-a,a]$ ¿Qué sucede?

Ejercicio 8.
Para calcular $\sqrt{2}$ los *antiguos* babilonios utilizaban, sin saberlo, la siguiente iteración de punto fijo:

$$x^2 = 2$$
$$2x^2 = x^2 + 2$$
$$2x = x + \frac{2}{x}$$
$$x = \frac{x + \frac{2}{x}}{2}$$

Y comenzaban con $x = 1$, sabían que si $1 \leq x < 2$ la respuesta se encontraba entre x y $\frac{2}{x}$, así que sacaban el promedio entre los dos números.

1. Demuestre que este método converge cuadráticamente.
2. Expanda su método para encontrar la raíz de cualquier número n.

Ejercicio 9.
El método *regula-falsi* tiene un radio de convergencia dado por:

$$\frac{e_{n+1}}{e_n} = 1 - f'(x_*)\frac{e_0}{f(x_0)}$$

donde x_0 corresponde al lado del *bracketing* que se mantiene durante las últimas iteraciones (osea, se supone que entre x_0 y x_* la función es convexa).
Identifique el radio de convergencia para la función $f(x) = x^2 - 1$ comenzando con el intervalo $[0,2]$.

Ejercicio 10.
Utilice 2 pasos del método de la bisección para encontrar la raíz de $f(x) = x^3 + x - 1$ en el intervalo $[0,1]$.

Ejercicio 11.
Dada $f(x) = log(x^2) - (x-2)^3$ utilice el método de Newton para encontrar su punto de inflexión ($f'(x) = 0$) en $[2,3]$.

Ejercicio 12.
Encontrar los puntos fijos de $\frac{3}{x}$ y $x^2 - 2x + 2$.

Ejercicio 13.
Ordenar las siguientes iteraciones que convergen a $\sqrt{5}$ de más rápida a menos rápida convergencia:

$$g_1(x_n) = x_{n+1} = \frac{4}{5}x_n + \frac{1}{x_n}$$
$$g_2(x_n) = x_{n+1} = \frac{1}{2}x_n + \frac{5}{2x_n}$$
$$g_3(x_n) = x_{n+1} = \frac{x+5}{x+1}$$

Ejercicio 14.
Demuestre que siendo $x = r$ el punto fijo de $g(x)$, si $g'(r) = 0$ (y $g(r)$ tiene primera y segunda derivada continua) entonces hay un órden de convergencia cuadrático $|g''(r)|/2$ cuando $n \to \infty$.

Búsqueda de raíces- Soluciones

1. El tamaño del intervalo tras n pasos será $\frac{B-A}{2^n}$ y el error máximo $x_n - x_*$ será la mitad de eso (porque siempre colocaremos x_n en la mitad del intervalo en el paso actual.
   Que la precisión sea mayor a k números decimales equivale a que el error sea menor que $0{,}5 \cdot 10^{-k}$. Entonces tenemos que lograr:

$$\begin{aligned} e_n &< 0{,}5 \cdot 10^{-k} \\ \frac{7-2}{2^{n+1}} &< 0{,}5 \cdot 10^{-k} \\ \frac{5}{2^{n+1}} &< 0{,}5 \cdot 10^{-k} \\ 2^{-(n+1)} &< 10^{-k-1} \\ -(n+1) &< \log_2(10^{-k-1}) \\ -n &< \log_2(10^{-k-1}) + 1 \\ n &> (k+1)\log_2(10) - 1 \end{aligned}$$

---

2. La iteración del método de Newton es:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

   Dadas nuestras condiciones sobre la función, expandiendo en serie de Taylor alrededor del cero x^* tenemos:

$$\begin{aligned} f(x) &= f(x^*) + \frac{f'(x^*)}{1!}(x-x^*) + \frac{f''(x^*)}{2!}(x-x^*)^2 + O((x-x^*)^3) \\ &= \frac{f''(x^*)}{2}(x-x^*)^2 + O((x-x^*)^3) \\ f'(x) &= f'(x^*) + \frac{f''(x^*)}{1!}(x-x^*) + O((x-x^*)^2) \end{aligned}$$

   Substituyendo, llamando $e_n = x_n - x^*$ al error y absorbiendo algunas constantes en los $O(\cdot)$ queda:

$$\begin{aligned} e_{n+1} &= e_n - \frac{f''(x^*)e_n^2 + O(e_n^3)}{2f''(x^*)e_n + O(e_n^2)} \\ &= e_n\left(1 - \frac{1}{2}\right) + O(e_n^2) \end{aligned}$$

   O sea,en este caso la convergencia es lineal.

---

3. Si el intervalo inicial era $[A,B]$, tras el paso n vemos que el tamaño del intervalo será $\frac{B-A}{2^n}$ y el error máximo $x_n - x_*$ a lo más será $\frac{\frac{B-A}{2^n}}{2}$ y que la aproximación en el paso n, se coloca en la mitad del intervalo pequeño.Al comparar los errores máximos en n y $n+1$, resulta

$$\begin{aligned} \frac{e_{n+1}}{e_n} &= \frac{\frac{\frac{B-A}{2^{n+1}}}{2}}{\frac{\frac{B-A}{2^n}}{2}} \\ &= \frac{\frac{B-A}{2^{n+2}}}{\frac{B-A}{2^{n+1}}} = \frac{1}{2} \end{aligned}$$

---

4. Cerca de la raiz r, la iteracion de punto fijo se comporta aproximadamente de forma lineal $e_{i+1} \approx Se_i$, donde $S = |g'(r)|$ y e_i es el error de la iteracion i. Sabemos que la raiz real de $f(x)$ es $r \approx 0{,}6823$, entonces:

   - Para el primer caso $g(x) = 1 - x^3$ la derivada es $g'(x) = -3x^2$ y $|g'(r)| \approx 1{,}3966 > 1$ por lo que esta iteración diverge.
   - Para la segunda version $g(x) = \sqrt[3]{1-x}$ la derivada es $g'(x) = -\frac{1}{3\sqrt[3]{1-x^2}}$ y $|g'(r)| \approx 0{,}3786258 < 1$ por lo que esta iteracion si converge localmente.
   - Para la tercera version

     $$g(x) = \frac{1+2x^3}{1+3x^2}$$
     $$g'(x) = \frac{6x(x^3+x-1)}{(3x^2+1)^2}$$

     y $|g'(r)| = 0$ convergiendo de forma veloz.(*Cabe mencionar que esta es la iteracion de Newton para* $f(x)$, $x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$ *que converge cuadraticamente*)

---

5. Resolviendo $g(x) = \frac{x^3+x-6}{6x-10} = x$ podemos encontrar sus puntos fijos:

   $$\frac{x^3+x-6}{6x-10} - x = 0$$
   $$\frac{(x-3)(x-2)(x-1)}{2(3x-5)} = 0$$
   $$x = 1, 2, 3$$

   Verificando que $1, 2$ y 3 son puntos fijos de la iteracion $g(x) = x$

---

6. Una forma rapida de encontrar el orden de convergencia de FPI es revisar las derivadas de g(x)

   - Para la primera revisamos $|g'(1)|$

     $$g'(x) = \frac{2-2x}{x^3}$$
     $$|g'(1)| = 0 \qquad \Longrightarrow \text{Existe algo mejor que convergencia lineal}$$

     Revisamos $|g''(1)|$

     $$g''(x) = \frac{4x-6}{x^4}$$
     $$|g''(1)| = 2 \neq 0 \qquad \Longrightarrow \text{Convergencia cuadratica}$$

   - Para el segundo caso:

     $$g'(x) = -sin(x)$$
     $$|g'(\pi)| = 0 \qquad \Longrightarrow \text{Existe algo mejor que convergencia lineal}$$

     Revisamos $|g''(\pi)|$

     $$g''(x) = cos(x)$$
     $$|g''(\pi)| = 1 \neq 0 \qquad \Longrightarrow \text{Convergencia cuadratica}$$

   - Para el tercer caso:

     $$g'(x) = 2e^{2x}$$
     $$|g'(0)| = 2 \qquad \Longrightarrow \text{Diverge}$$

     0 no es punto fijo, basta con probar cualquier *initial guess* en la localidad de 0

7. El primer guess es $c = (-a + a)/2 = 0$, resultando en una division por 0 en la primera iteracion.

8. Para la a se puede demostrar que se trata del método de Newton!

   a) Se puede verificar que este algoritmo, es en realidad el metodo de Newton, que se sabe tiene convergencia cuadratica.

   Encontrar $\sqrt{2}$ es equivalente a encontrar la solucion de

   $$x^2 = 2$$

   haciendo busqueda de ceros a

   $$f(x) = x^2 - 2$$

   Con derivada

   $$f'(x) = 2x$$

   Entonces la iteracion de Newton es

   $$x_{n+1} = x_n - \frac{x_n^2 - 2}{2x_n}$$

   Con un poco de algebra...

   $$x_{n+1} = \frac{2x_n^2}{2x_n} - \frac{x_n^2 - 2}{2x_n}$$
   $$x_{n+1} = \frac{x_n^2 + 2}{2x_n}$$
   $$x_{n+1} = \frac{\frac{x_n^2+2}{x_n}}{2}$$
   $$x_{n+1} = \frac{x_n + \frac{2}{x_n}}{2}$$ y este es el metodo de los *antiguos* babilonios

   El metodo de Newton-Raphson tiene convergencia cuadratica.

   Alternativamente, para la FPI $g(x) = \frac{x+\frac{2}{x}}{2}$ se puede comprobar convergencia con su derivada en el punto fijo $\sqrt{2}$:

   $$g'(x) = \frac{1}{2} - \frac{1}{x^2}$$

   Con $r = \sqrt{2}$, $|g'(\sqrt{2})| = 0$ convergiendo al menos linealmente, se verifica ahora la segunda derivada:

   $$g''(x) = \frac{2}{x^3}$$

   con $r = \sqrt{2}$, $|g''(\sqrt{2})| \neq 0$ demostrando convergencia al menos cuadratica.

b) Sabiendo que este es el metodo de Newton, se puede expandir para cualquier raiz cuadrada

Encontrar $\sqrt{x}$ es equivalente a encontrar la solucion de

$$\begin{aligned} x^2 &= r \\ f(x) &= x^2 - r \\ f'(x) &= 2x \end{aligned}$$

Entonces la iteracion de Newton es

$$\begin{aligned} x_{n+1} &= x_n - \frac{x_n^2 - r}{2x_n} \\ x_{n+1} &= \frac{x_n + \frac{r}{x_n}}{2} \end{aligned}$$

---

9. Colocar gráfico...Si graficamos la función vemos que $x_0 = 2$, pues el *braketing* siempre mantendrá ese lado fijo, podemos usar eso para encontrar $f(x_0)$ y e_0:

$$\begin{aligned} \left|\frac{e_{n+1}}{e_n}\right| &= \left|1 - f'(x_*)\frac{e_0}{f(x_0)}\right| \\ &= \left|1 - f'(1)\frac{2-1}{f(2)}\right| \\ &= \left|1 - 2\frac{1}{3}\right| \\ &= \left|\frac{1}{3}\right| \end{aligned}$$

El radio de convergencia es 1/3, mejor que usando bisección.

---

10. Inicialmente $a = 0$ y $b = 1$.Se comprueba que $f(a) = -1$ y $f(b) = 1$, por lo que existe una raíz en dicho intervalo (dado que la función es continua),Se tiene que $c = (0+1)/2 = 0{,}5$ y resulta $f(c) = -0{,}375$ por lo que la solución se encuentra entre c y b, c se convierte en el nuevo a.Se tiene que $c = (0{,}5+1)/2 = 0{,}75$ y resulta $f(c) = 0{,}171875$ por lo que la solución se encuentra entre a y c, c se convierte en el nuevo b.Realizados los dos pasos, se retorna el promedio entre a y b, $c = (0{,}5+0{,}75)/2$.

---

11. Como se busca un 0 de la derivada de $f(x)$, $f'(x)$ sería nuestra función f en el método de Newton. Nuestra iteración de punto fijo queda entonces:

$$\begin{aligned} g(x) &= x - \frac{f'(x)}{(f'(x))'} \\ &= x - \frac{f'(x)}{f''(x)} \end{aligned}$$

Por lo que calculamos $f'(x)$ y $f''(x)$:

$$\begin{aligned} f'(x) &= \frac{2x}{x^2} - 3(x-2)^2 \\ &= \frac{2}{x} - 3(x-2)^2 \\ f''(x) &= -\frac{2}{x^2} - 6(x-2) \end{aligned}$$

Y resulta:

$$g(x) = x - \frac{\frac{2}{x} - 3(x-2)^2}{-\frac{2}{x^2} - 6(x-2)}$$
$$= x + \frac{2x - 3(x-2)^2 x^2}{2 + 6(x-2)x^2}$$

Se elige un x_0 cerca del punto que buscamos, para estar seguros de que no converja a otro punto fuera del intervalo elegimos la mitad del mismo. $x_0 = 2{,}5$ y comenzamos a iterar.

i	x_i
0	2,5
1	2,527472527472527
2	2,505591740706584
3	2,522514171936734
4	2,509093315687682
5	2,519544667396556
6	2,511281645983929
7	2,517740823220339
8	2,512644898862549
9	2,516637059111556
10	2,513491843041918
11	2,515958989472346
12	2,514016980816051
13	2,515541496274079
14	2,514342151287987
15	2,515284103770793
16	2,514543324990736
17	2,515125291814672
18	2,514667716476063
19	2,515027257882626
20	2,514744604486299

Se ve que nos vamos acercando al valor real $x \approx 2{,}51486893843872$.

12. Para encontrar los puntos fijos hay que igualar la función a x:

$$x = \frac{3}{x}$$
$$x^2 = 3$$
$$x = \pm\sqrt{3}$$

La primera función tiene dos puntos fijos, $-\sqrt{3}$ y $\sqrt{3}$.

$$x = x^2 - 2x + 2$$
$$0 = x^2 - 3x + 2$$
$$0 = (x-1)(x-2)$$

La segunda función tiene dos puntos fijos, $x = 1$ y $x = 2$.

13. Se calcula la derivada de cada una de las iteraciones de punto fijo:

$$g_1'(x) = \frac{4}{5} - \frac{1}{x^2}$$
$$g_2'(x) = \frac{1}{2} - \frac{5}{2x^2}$$
$$g_3'(x) = \frac{(x+1)-(x+5)}{(x+1)^2} = \frac{-4}{(x+1)^2}$$

Se evalúan todas en $x = \sqrt{5}$, resultando:

$$|g_1'(\sqrt{5})| = \frac{3}{5}$$
$$|g_2'(\sqrt{5})| = 0$$
$$|g_3'(x)| = \frac{4}{(\sqrt{5}+1)^2} \approx 0{,}381966$$

Se ve que g_2 tiene un órden de convergencia mejor que lineal porque $|g_2'(\sqrt{5})| = 0$, por lo tanto, cerca de la raíz, converge más rápido que las otras funciones. De los otros métodos, g_3 es mejor que g_1 porque tiene un radio de convergencia lineal menor, todos convergen ya que este radio es menor que 1.

---

14. De la serie de Taylor:

$$g(x_n) = g(r) + g'(r)(x_n - r) + \frac{1}{2}g''(r)(x_n - r)^2 + O((x_n - r)^3)$$
$$x_{n+1} = r + g'(r)(x_n - r) + \frac{1}{2}g''(r)(x_n - r)^2 + O((x_n - r)^3)$$
$$x_{n+1} - r = g'(r)(x_n - r) + \frac{1}{2}g''(r)(x_n - r)^2 + O((x_n - r)^3)$$
$$x_{n+1} - r = \frac{1}{2}g''(r)(x_n - r)^2 + O((x_n - r)^3)$$
$$\frac{x_{n+1} - r}{(x_n - r)^2} = \frac{1}{2}g''(r) + O((x_n - r))$$
$$\frac{|x_{n+1} - r|}{|(x_n - r)^2|} = \left|\frac{g''(r)}{2}\right| + |O((x_n - r))|$$

Finalmente, si $n \to \infty$:

$$\frac{|x_{n+1} - r|}{|(x_n - r)^2|} = \frac{|g''(r)|}{2}$$

---

2. Interpolación

Ejercicio 1.
Encontrar una cantidad n (de puntos de Chebyshev) que permita construir un polinomio interpolador $Q_n(x)$para la función $f(x) = \ln\left(\frac{x}{2}\right)$ de tal manera que el error de interpolación para cualquier $x \in [\frac{1}{2}, 2]$ sea:

$$|f(x) - Q_n(x)| \leq K$$

Donde K es una constante positiva cualquiera.
Nota: No es necesario que encuentre una expresión explícita para n.

Ejercicio 2.

Encontrar el polinomio interpolador de los puntos $(-1,1),(2,3)\,y(4,0)$ utilizando diferencias divididas. Suponga que ahora le piden interpolar agregando el punto (a,b). Calcule el nuevo polinomio interpolador reciclando su trabajo anterior.

Ejercicio 3.

Se quiere interpolar, utilizando puntos de Chebyshev, la función $f(x) = e^{2x}$ en el intervalo $[0,r]$, encuentre una cota para el número mínimo de puntos necesarios para que el error sea menor que $\mathfrak{E}$ en dicho intervalo.

Ejercicio 4.

Considere los puntos $(0,0);(1,0);(2,0);(3,0);(4,24)$. Encuentre el polinomio interpolador usando Lagrange y Diferencias Dividas.¿Qué método es más directo en este caso? ¿Por qué?

Ejercicio 5.

Considere los puntos $(-2,0);(-1,0);(0,1);(0,2);(1,3);(2,3)$. Claramente no provienen de ninguna función, pero ¿Es posible utilizar nuestros conocimientos de interpolación polinomial para encontrar una curvaque se ajuste a estos puntos?

Interpolación - Soluciones

1. Primero vemos que si utilizamos n puntos de Chevyshev, la productoria estará acotada por:

$$\left|\prod_{1\le i\le n}(x-x_i)\right| \le \frac{\left(\frac{2-\frac{1}{2}}{2}\right)^n}{2^{n-1}} = \frac{\left(\frac{3}{4}\right)^n}{2^{n-1}} = 2\left(\frac{3}{8}\right)^n$$

Por otro lado, se busca la $n+1$ derivada de $f(x)$:

$$\begin{aligned} f(x) &= \ln\frac{x}{2} \\ f'(x) &= \frac{1}{x} \\ f''(x) &= -\frac{1}{x^2} \\ f'''(x) &= \frac{2}{x^3} \\ f^{(4)}(x) &= -\frac{6}{x^3} \\ &\vdots \\ f^{(n)}(x) &= (-1)^{n-1}\frac{(n-1)!}{x^n} \\ f^{(n+1)}(x) &= (-1)^n\frac{n!}{x^{n+1}} \end{aligned}$$

Fijándose sólo en el valor absoluto vemos que, en el intervalo $[\frac{1}{2},2]$, $f^{(n+1)}(x)$ está acotada por sus valores en los extremos:

$$\begin{aligned} f^{(n+1)}(1/2) \ge f^{(n+1)}(x) \ge f^{(n+1)}(2) \qquad &\text{con } n \text{ par.} \\ f^{(n+1)}(1/2) \le f^{(n+1)}(x) \le f^{(n+1)}(2) \qquad &\text{con } n \text{ impar.} \end{aligned}$$

Ahora nos remitimos a los valores absolutos:

$$\begin{aligned} \left|f^{(n+1)}(2)\right| \le \left|f^{(n+1)}(x)\right| &\le \left|f^{(n+1)}(1/2)\right| \\ \frac{n!}{2^{n+1}} = \le \left|f^{(n+1)}(x)\right| &\le \frac{n!}{(\frac{1}{2})^{n+1}} \\ \frac{n!}{2^{n+1}} = \le \left|f^{(n+1)}(x)\right| &\le 2^{n+1}n! \\ \frac{n!}{2^{n+1}} = \le \left|f^{(n+1)}(\zeta)\right| &\le 2^{n+1}n! \end{aligned}$$

Sabemos entonces que:

$$\left|\prod_{1\le i\le n}(x-x_i)\right| \le \frac{\left(\frac{3}{4}\right)^n}{2^{n-1}} \qquad \text{y} \qquad \left|f^{(n+1)}(\zeta)\right| \le 2^{n+1}n!$$

Por lo tanto, el valor absoluto del error:

$$\left|\frac{1}{(n+1)!}f^{(n+1)}(\zeta)\prod_{1\le i\le n}(x-x_i)\right| \le \frac{1}{(n+1)!}\cdot\frac{\left(\frac{3}{4}\right)^n}{2^{n-1}}\cdot 2^{n+1}n! = \frac{4\left(\frac{3}{4}\right)^n}{n+1}$$

Entonces, bastaría elegir n suficientemente grande para que:

$$\frac{4\left(\frac{3}{4}\right)^n}{n+1} \leq K$$

2.

$$\begin{array}{c|l} -1 & 1 \\ & \quad \frac{3-1}{2-(-1)} = \frac{2}{3} \\ 2 & 3 \qquad\qquad \frac{\frac{-3}{2}-\frac{2}{3}}{4-(-1)} = \frac{-13}{30} \\ & \quad \frac{0-3}{4-2} = \frac{-3}{2} \\ 4 & 0 \end{array}$$

$$P_2(x) = 1 + \frac{2}{3}(x-(-1)) + \frac{-13}{30}(x-(-1))(x-2)$$

Agregar otro punto es agregar un piso mas a la piramide:

$$P_3(x) = P_2(x) + \frac{\frac{\frac{\mathbf{b-0}}{\mathbf{a-4}} - \frac{\mathbf{-3}}{\mathbf{2}}}{\mathbf{a-2}} - \frac{\mathbf{-13}}{\mathbf{30}}}{\mathbf{a-(-1)}}(x-(-1))(x-2)(x-4)$$

por:

$$\begin{array}{c|l} -1 & 1 \\ & \quad \frac{3-1}{2-(-1)} = \frac{2}{3} \\ 2 & 3 \qquad\qquad \frac{\frac{-3}{2}-\frac{2}{3}}{4-(-1)} = \frac{-13}{30} \\ & \quad \frac{0-3}{4-2} = \frac{-3}{2} \qquad\qquad \frac{\frac{\mathbf{b-0}}{\mathbf{a-4}} - \frac{\mathbf{-3}}{\mathbf{2}}}{\mathbf{a-2}} - \frac{\mathbf{-13}}{\mathbf{30}} \\ 4 & 0 \qquad\qquad \frac{\frac{\mathbf{b-0}}{\mathbf{a-4}} - \frac{\mathbf{-3}}{\mathbf{2}}}{\mathbf{a-2}} \\ & \quad \mathbf{\frac{b-0}{a-4}} \\ \mathbf{a} & \mathbf{b} \end{array}$$

---

3. Tomando la fórmula del error de interpolación:

$$E(x) = f(x) - Q_n(x) = \frac{1}{(n+1)!} \cdot f^{(n+1)}(\zeta) \cdot \prod_{1 \leq j \leq n} (x - x_j)$$

Sabemos que por tratarse de puntos de chebyshev:

$$\left| \prod_{1 \leq j \leq n} (x - x_j) \right| \leq \frac{(r-0)^n}{2^{2n-1}} = \frac{r^n}{2^{2n-1}}$$

Ahora debemos encontrar una cota para:

$$f^{(n+1)}(x) = 2^{n+1} e^{2x}$$

Como esta función es positiva y creciente en el intervalo $[0, r]$, su máximo estará en $x = r$, por lo tanto:

$$|f^{(n)}(x)| \leq |f^{(n)}(r)| = 2^n e^{2r}$$

Entonces, el error en dicho intervalo, está acotado por:

$$\begin{aligned} |E(x)| &= \left|\frac{1}{n!}\right| \cdot \left|f^{(n)}(\zeta)\right| \cdot \left| \prod_{1 \leq j \leq n} (x - x_j) \right| \\ &\leq \frac{1}{n!} 2^n e^{2r} \frac{r^n}{2^{2n-1}} \\ &= \frac{e^{2r}}{(n+1)!} \frac{r^n}{2^{n-1}} \end{aligned}$$

Simplemente hay que elegir un n tal que:

$$\frac{e^{2r}r^n}{n!\cdot 2^{n-1}} \leq \mathfrak{E}$$

---

4. **Diferencias Divididas**:
   Construimos la pirámide:

0	0				
		0			
1	0		0		
		0		0	
2	0		0		1
		0		4	
3	0		12		
		24			
4	24				

   obteniendo el polinomio $x(x-1)(x-2)(x-3)$. **Lagrange**:
   Obtenemos inmediatamente el polinomio:

$$0\cdot L_0 + 0\cdot L_1 + 0\cdot L_2 + 0\cdot L_3 + 24\cdot\frac{(x-3)(x-2)(x-1)x}{4!} \quad = \quad x(x-1)(x-2)(x-3)$$

   Lagrange resulta más directo ya que la mayor parte de las imágenes son 0, por lo que solo hay que calcular uno de los polinomios de Lagrange.

---

5. Efectivamente. Normalmente encontramos $P(x) \sim f(x)$, siendo f una función evaluada en los puntos de interpolación, sin embargo podemos hacer lo mismo conuna curva paramétrica $c(x,y) = \langle x(s), y(s)\rangle \sim \langle P_x(s), P_y(s)\rangle$. Solo necesitamos interpolar dos veces. En este caso podemos tomar un parámetro s equiespaciado de que va de 0 a 5, usando diferencias divididas para $P_x(s)$:

0	-2					
		1				
1	-1		0			
		1		$\frac{-1}{6}$		
2	0		$\frac{-1}{2}$		$\frac{1}{8}$	
		0		$\frac{1}{3}$		$\frac{-1}{20}$
3	0		$\frac{1}{2}$		$\frac{-1}{8}$	
		1		$\frac{-1}{6}$		
4	1		0			
		1				
5	2					

   Lo que nos da el polinomio:

$$P_x(s) = -2 + s\Big(1 + (s-1)(s-2)\Big(-\frac{1}{6} + (s-3)\Big(\frac{1}{8} - \frac{1}{20}(s-4)\Big)\Big)\Big)$$

Lo mismo para $P_y(s)$:

$$
\begin{array}{cccccc}
0 & 0 & & & & \\
 & & 0 & & & \\
1 & 0 & & \frac{1}{2} & & \\
 & & 1 & & \frac{-1}{6} & \\
2 & 1 & & 0 & & \frac{1}{24} \\
 & & 1 & & 0 & \\
3 & 2 & & 0 & & \frac{-1}{24} \\
 & & 1 & & \frac{-1}{6} & \\
4 & 3 & & \frac{-1}{2} & & \\
 & & 0 & & & \\
5 & 3 & & & &
\end{array}
$$

Lo que nos da el polinomio:

$$P_y(s) = s(s-1)\Big(\frac{1}{2} + (s-2)\Big(-\frac{1}{6} + (s-3)\Big(\frac{1}{24} - \frac{1}{60}(s-4)\Big)\Big)\Big)$$

---

3. Cuadratura

Ejercicio 1.
Utilizar interpolación gaussiana con 3 puntos para obtener:

$$\int_{-2}^{3} 2^x dx$$

Ejercicio 2.
Suponga una formula de cuadratura de la forma:

$$\int_{-1}^{1} f(x)dx = a_{-1}f(-1) + a_1 f(1) + b_{-1}f'(-1) + b_1 f'(1)$$

1. Plantee ecuaciones para los coeficientes del polinomio interpolador de maximo grado posible en terminos de $f(-1), f(1), f'(-1), f'(1)$
2. Evalúe la integral para el polinomio interpolador, use las ecuaciones anteriores para hallar los coeficientes.

Cuadratura - Soluciones

1. WIP.

2. *a*) Hay cuatro coeficientes, podemos aspirar a una fórmula exacta para polinomios cúbicos:

$$p(x) = c_0 + c_1 x + x_2 x^2 + c_3 x^3$$
$$p'(x) = c_1 + 2x_2 x + 3c_3 x^2$$

Los hacemos coincidir con los valores de la función:

$$f(-1) = p(-1) = c_0 - c_1 + c_2 - c_3$$
$$f(1) = p(1) = c_0 + c_1 + c_2 + c_3$$
$$f'(-1) = p'(-1) = c_1 - 2c_2 + 3c_3$$
$$f'(1) = p'(1) = c_1 + 2c_2 + 3c_3$$

b) Integramos el polinomio interpolador exacto:

$$\int_{-1}^{1} p(x)dx = \left(c_0 x + \frac{1}{2}c_1 x^2 + \frac{1}{3}c_2 x^3 + \frac{1}{4}c_3 x^3\right)\Big|_{-1}^{1} = 2c_0 + \frac{2}{3}c_2$$

De las ecuaciones de los coeficientes:

$$2c_0 + 2c_2 = f(-1) + f(1)$$
$$4c_2 = f'(1) - f'(-1)$$

Queda:

$$\int_{-1}^{1} f(x)dx = f(-1) + f(1) + \frac{f'(-1) - f'(1)}{3}$$

---

4. Algoritmos voraces

Ejercicio 1.
Plantee un algoritmo voraz para encontrar el camino más corto desde un vértice dado de un grafo a cada uno de los otros. ¿Cuánto es el tiempo de ejecución de su algoritmo en términos del número de arcos E y el número de vértices V?

Ejercicio 2.
Se tiene un texto formado por n palabras $w_1, w_2, \ldots, w_n$, el largo en caracteres de la palabra i es l_i. Se desea justificar este texto en líneas de largo L, sin sobrepasar este ancho. Para cada palabra que se pone en la línea(salvo la última) se agrega un espacio que la separa de la anterior.

1. Plantee un algoritmo voraz que resuelva este problema
2. Obtenga la complejidad de su algoritmo

Ejercicio 3.
Se tiene una colección de tareas, con tareas i de duración d_i. Interesa ordenar la ejecución de las tareas de forma de minimizar:

$$\sum_{1 \leq i \leq n} (\text{espera del cliente } i)$$

Acá lo que espera el cliente j, si se ejecutan las tareas en orden de índice creciente, es simplemente:

$$\sum_{1 \leq k \leq j} d_k$$

Plantee un algoritmo voraz para este problema,y demuestre que da el orden óptimo.

Ejercicio 4.
En el lejano pais de "*Horstlandia*", donde la maldad es infinita, utilizan el sistema "*VonBrand Coins*" con denominaciones $\{1,4,7,13,28,52,91,365\}$. Por decreto constitucional del *Doctor Supremo* se exige que el vuelto se entregue con la menor cantidad de "*coins*", fallar a esto se castiga con una eternidad en *MAT021*.
Demuestre que el algoritmo voraz de entregar lo más que se puede de la máxima denominación sin usar va a enviar a mucha gente a *MAT021*.

Ejercicio 5.
La Libra Esterlina viene en denominaciones de $\{1,2,5,10,20,50\}$. Teniendo una dotacion infinita de cada una de estas monedas, plantee un algoritmo voraz para entregar cierta cantidad n usando la menor cantidad de monedas.

Ejercicio 6.
Se tiene dos conjuntos de valores reales positivos $\langle a_1, a_2, ..., a_n \rangle$ y $\langle b_1, b_2, ..., b_n \rangle$, tal que:

$$a_1 \geq a_2 \geq ... \geq a_n \quad \text{y}$$
$$b_1 \geq b_2 \geq ... \geq b_n$$

Se busca maximizar:

$$\sum_{i=1}^{n} a_i b_{\phi(i)}$$

Donde $\phi : [1..n] \rightarrow [1..n]$ es una biyección. Dicho de otra forma, se busca hacer un matching entre a's y b's que maximice la suma de las multiplicaciones entre los pares.Demostrar que el algoritmo voraz de emparejar a_1 y b_1 (osea, los más grandes de cada conjunto) en cada paso es óptimo.

Ejercicio 7.

1. Una flota de barcos se reparten en la costa del mar, cada barco, en el eje x ocupa un intervalo de la forma $[a,b]$, se tiene un arma que dispara un rayo en un punto x que destruye todos los barcos cuyo intervalo contenga dicho x.
   - Describa un algoritmo voraz que destruya toda la flota con la cantidad mínima de disparos.
   - Demuestre su optimalidad.
2. Ahora se tiene una serie de globos representados como círculos de diferente radio en todo el plano, y un arma fija en el origen que puede disparar rayos hacia cualquier ángulo.
   - Describa como podría modificar el algoritmo del ejercicio anterior para resolver
   - Demuestre que este nuevo algoritmo no gasta más de un rayo más que la solución óptima.

Ejercicio 8.
Una string de paréntesis balanceada puede encontrarse en alguna de las siguientes formas:

- Una string vacía.
- Una string (x) donde x es una string balanceada.
- Una string xy donde x e y son strings balanceadas.

1. Describir un algoritmo voraz para encontrar la substring balanceada de mayor largo en una string de paréntesis como:

$$((()()())))()(()()()$$

2. Demostrar su optimalidad.

Ejercicio 9.
En el país de *Otakar* existen muchas ciudades que cultivan arandanos, *Otakar* no tiene una red de internet pero quiere implementar una, no tienen dinero para conectar todas las ciudades con todas, pero es necesario conectar la ciudades que producen arandanos usando la menor cantidad de cable posible. Diseñe e implemente un algoritmo (voraz por que no hay dinero para pagar algo mejor) que permita determinar como conectar a la red sus ciudades *arandaneras* con la menor cantidad de cable posible.

- Demuestre que su algoritmo encuentra el optimo
- Implemente su algoritmo

Algoritmos voraces - Soluciones

1. WIP.
2. WIP.
3. WIP.
4. Para representar $n = 37$, el algoritmo elegiria: $[25, 9, 1, 1, 1]$ 5 monedas, siendo que el optimo es 4 monedas: $[25, 4, 4, 4]$. Hemos encontrado un caso en el que se incumple la propiedad *Greedy Choice* puesto que la eleccion voraz $\hat{p} = 9$ no esta en la solucion optima $\Pi^\star = \{25, 4, 4, 4\}$
5. El algoritmo consiste en agregar la moneda k de mayor valoración tal que $k \leq n$, luego se sigue, resolviendo el problema para $n-k$. Cuando llegamos n sea igual a alguna moneda, agregamos dicha moneda y terminamos.Demostramos las 3 propiedades:

   Elección voraz Supongamos que existe una solución $\Pi_n^\star$ óptima que no elige la moneda de mayor valor m (tal que $m \leq n$), entonces, poseé un subconjunto de monedas que suman m (por los tipos de monedas que hay), si reemplazamos ese subconjunto por la moneda m, tendríamos una solución mejor, pero $\Pi_n^\star$ era óptima, contradicción.

   Estructura inductiva: Tras elegir la moneda m, resulta el subproblema P_{n-m} de representar $n-m$ con monedas, El conjunto de monedas $\Pi_{n-m} \cup \{m\}$, es solución factible para P_n ya que suman m y no hay restricciones externas al resolver P_{n-m} (la moneda elegida no afecta para nada cómo resolveremos P_{n-m}).

   Sub-estructura óptima: Debemos demostrar que $\Pi_n = \Pi_{n-m}^\star \cup \{m\}$, es óptima, osea, no se puede resolver el problema P_n con menos monedas que $|\Pi_{n-m}^\star| + 1$.
   Supongamos lo contrario, entonces es posible reemplazar un subconjunto de monedas de $\Pi_{n-m}^\star$ por otro subconjunto de menos o igual cantidad de monedas de manera que la suma de estas monedas aumente en m.
   Si hacemos este cambio queda una solución para P_n con $|\Pi_{n-m}^\star|$ monedas, por lo visto en Greedy Choice, existe una solución igual de óptima que tiene una moneda m, si sacamos esa moneda, tendríamos una solución para P_{n-k} que requeriría $|\Pi_{n-m}^\star| - 1$ monedas y por lo tanto $\Pi_{n-m}^\star$ no sería óptima. contradicción.
6. - **Elección voraz**: Cualquier solución óptima Π^* que no tenga $a_1 b_1$, tendrá que tener $a_i b_1$ y $a_1 b_j$, en dicha solución podemos hacer un swap, intercambiando $a_i b_1$ y $a_1 b_j$ por $a_1 b_1$ y $a_i b_j$, en dicho caso la diferencia en la sumatoria será:

     $$a_1 b_1 + a_i b_j - a_i b_1 - a_1 b_j$$

     Este valor es positivo ya que:

     $$\begin{aligned} &(a_1 - a_i) \geq 0 \wedge (b_1 - b_j) \geq 0 \\ &\qquad \Rightarrow (a_1 - a_i)(b_1 - b_j) \geq 0 \\ &\Rightarrow a_1 b_1 + a_i b_j - a_i b_1 - a_1 b_j \geq 0 \end{aligned}$$

     Por lo tanto, o la solución Π^* que no incluye $a_1 b_1$ y es óptima no existe (en caso de que la expresión siempre sea positiva) o existe una solución óptima que incluye $a_1 b_1$.
   - **Sub-estructura inductiva**: Tras hacer el match $a_1 b_1$ sacamos a_1 y b_1 del problema, quedando un problema P' en que a cualquier solución Π' (si vemos las soluciones como conjuntos de pares) se le puede agregar $a_1 b_1$ para formar una solución factible de P.
   - **Sub-estructura óptima**: De elección voraz se obtuvo que existe una solución óptima que incluye $a_1 b_1$ para P y la mejor forma posible de elegir los otros pares sería la solución óptima a P' dada por Π'^*, como el criterio de optimalidad es al suma de los pares, la única forma de maximizar esta suma es maximizando los sumandos y por lo tanto, $\Pi = \{a_1 b_1\} \cup \Pi'^*$ debe óptima para P.

(No existe la posibilidad de que una solución a P' diferente aumente más $a_1 b_1 + \sum_{i=2}^{n} a_i b_{\phi(i)}$, pues dicha solución no puede aumentar otra cosa que $\sum_{i=2}^{n} a_i b_{\phi'(i)}$ y si pudiera hacerlo Π'^* ya no sería óptima)z.

7. WIP.
8. WIP.
9. WIP.

5. Programación dinámica

Ejercicio 1.

Se desea diseñar un sistema formado por una serie de dispositivos D_i conectados en serie.
El problema es que los elementos constituyentes de estos dispositivos no son totalmente fiables, siendo r_i la probabilidad de que un elemento i funcione correctamente.
Para solucionar este problema se colocan m_i copias de cada dispositivo D_i en paralelo, así, la probabilidad de que la etapa i falle, será aproximadamente $(1 - r_i)^{m_i}$.
La probabilidad de que el circuito completo funcione bien es el producto de las probabilidades del funcionamiento correcto de toda las etapas.
Considerando que un dispositivo de tipo i tiene un costo c_i, plantee un método eficiente para encontrar el mejor circuito (encontrar lo números m_i que den la máxima confiabilidad del sistema) que se puede hacer con un costo no mayor que C.

Ejercicio 2.

Plantee un método de solución al siguiente problema de inversiones:
Se cuenta con 30 millones de pesos a invertir.
Se han planteado diez proyectos, tal que el proyecto i requiere una inversión de I_i millones de pesos, y tiene un retorno al cabo de dos años estimado en R_i.
El capital que no se invierta en proyectos se puede colocar en el mercado financiero a una tasa del 10 % de interés anual.
¿Cuál es el conjunto óptimo de proyectos a financiar?

Ejercicio 3.

Plantee un algoritmo basado en programación dinámica para encontrar el subrango $a[i..j]$ de un arreglo $a[1..N]$ que tenga la máxima suma. ¿Que complejidad tiene su algoritmo?

Ejercicio 4.

El *problema de asignación de máquinas* considera dos máquinas iguales y un conjunto de tareas (con tiempos de ejecución enteros) a ser efecuadas en ellas.
Describa un algoritmo que encuentre el tiempo mínimo necesario para completar todas las tareas.

Ejercicio 5.

Dado un arreglo a de n elementos describa un algoritmo basado en programación dinámicapara calcular la subsecuencia creciente más larga(LIS), la secuencia de índices $x_1, x_2, \ldots, x_k$ más larga tal que $a[x_1] \leq a[x_2] \leq \ldots \leq a[x_k]$. Por ejemplo, para $\langle 90, 10, 22, 9, 33, 21, 50, 41, 60, 80 \rangle$ la subsecuencia creciente más larga es $\langle 10, 22, 33, 50, 60, 80 \rangle$ (posiciones $2, 3, 5, 7, 9, 10$).

1. De la recurrencia subyacente.
2. Demuestre que se cumplen las suposiciones de programación dinámica.
3. Esboce un algoritmo iterativo, indicando en particular el orden en que se llena la estructura.
4. Indique cómo extraer la solución de su estructura.
5. Indique la complejidad de su algoritmo.

Pista: Considere la secuencia más larga que termina con $a[i]$.

Ejercicio 6.

Le dan a usted una pieza de tela de tamaño $X \times Y$, y una lista de n posibles productos que se pueden hacer (cuantas veces se quiera), cada uno genera una ganancia de g_i y requiere un rectángulo de tela de $a_i \times b_i$ para su fabricación. Ud. cuenta con una máquina que puede cortar cualquier rectángulo en dos piezas, ya sea vertical o horizontalmente.

1. Diseñe un algoritmo de programación dinámica que determine la máxima ganancia que se puede sacar de un trozo de tela.
2. Defina la fórmula recursiva del algoritmo.
3. Determine el orden de resolución de los subproblemas.
4. Demuestre que el algoritmo es óptimo.

Ejercicio 7.

Recuerde la definición del coeficiente binomial:

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

Que tiene algunas propiedades que siempre se nos olvidan:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

$$\binom{n}{0} = 1$$

$$\binom{k}{k} = 1$$

Con $n, k \in \mathbb{N}$:

1. Describa las desventajas de usar la definición para calcular $\binom{n}{k}$
2. Determine la complejidad de calcular $\binom{n}{k}$ por definición.
3. Describa un algoritmo *"eficiente"* para calcular $\binom{n}{k}$
4. Determine la complejidad de su algoritmo eficiente para calcular $\binom{n}{k}$

Ejercicio 8.

Imagine una grilla bidimensional de $n \times k$(si, una matriz, otra vez ¬¬) sobre la que se mueve un robot llamado "Robi". Robi empieza en la casilla $[0,0]$ y tiene como objetivo llegar a la casilla $[n,k]$. El robot se mueve vertical y horizontalmente una casilla a la vez. Sobre el tablero se han arrojado monedas al azar (como maximo una por celda). Usted es muy avaro y obliga a Robi a recolectar la mayor cantidad de monedas mientras se mueve desde una esquina a la otra, pero también es perezoso asi que va a escribir un programa para calcular el recorrido.

	0	1	2	3	4
0				$	
1		$			
2				$	
3		$			

Programación dinámica - Soluciones

1. WIP.
2. WIP.
3. WIP.
4. WIP.
5. Paso a paso.

   a) Considerando la subsecuencia creciente más larga que llega hasta $a[i]$, hay dos casos según este último:

   Es menor que todos los anteriores: En este caso, la secuencia es únicamente este elemento.

   Es mayorque algunos de los anteriores: En este caso,es la secuencia más largaque termina en un elemento anterior menor a $a[i]$ junto con $a[i]$.

   Esto da la recurrencia:

   $$\text{lis}(i) = \begin{cases} 1 & \text{si } a[i] \leq a[j] \text{ para } 1 \leq j \leq i-1 \\ 1 + \max\limits_{\substack{1 \leq j \leq i-1 \\ a[j] < a[i]}} \{\text{lis}(j)\} & \end{cases}$$

   Además de el valor de $\text{lis}(i)$ debemos registrar el valor de j que da lugar al máximopara poder reconstruir la secuencia.

   b) Las suposiciones son:

   Casos exhaustivos: Consideramos todas las alternativas para el último elemento:Siempre lo incluimos, aún si es en una secuencia de largo uno.

   Subestructura inductiva: Agregar un nuevo elemento no puede interferircon las secuencias crecientes que terminanen cada uno de los elementos existentes.

   Subestructura óptima: Si la secuencia creciente más larga incluye a $a[n]$,la secuencia creciente que termina en el elemento anterior en ellaclaramente debe ser de largo máximo.

   c) La idea es llenar un arreglo con el largo de la secuencia más larga que llega a $a[i]$,y paralelamente registrar el valor de j que da lugar a ese máximo. Es claro que la recurrencia de 5*a* que podemos ir llenando este arreglo ordenadamente desde $i = 1$ hasta $i = n$, solo haremos uso de valores calculados anteriormente. Conviene ir registrando el máximo hallado hasta el momento durante el proceso de llenado, este es LIS de la secuencia.

   d) Para obtener la secuencia de lo registrado,buscamos el elemento del arreglodefinido en el punto 5*c* el índice que da el máximo, y seguimos los índices j que referencian las secuencias previas.

   e) Recorrer el arreglo para hallar j según lo esbozado por la recurrencia de 5*a* tiene costo $O(i)$, como esto se repite para cada uno de los n elementos el total de esta fase es $O(n^2)$. Construir la secuencia creciente máxima de los datos registradosclaramente es $O(n)$. En total, el tiempo de ejecución es $O(n^2)$.

---

6. *a*) En primer lugar hay que notar que hay dos formas posibles de ubicar un rectángulo $a_i \times b_i$ en un rectángulo de tela, llámese en forma de paisaje o en forma de retrato, siempre conviene hacerlo en una esquina (da lo mismo cual). Para cada una de estas dos formas de hacerlo, si el rectángulo del producto cabe, existen dos formas de cortar el rectángulo más grande (horizontal o verticalmente), esto resulta en hasta 4 posibles formas de recortar el rectángulo por cada producto i, cada una de estas aportará p_i a la ganancia y dejará 2 rectángulos (pueden tener un largo 0).

   Entonces el problema $P(a,b)$ de obtener la ganancia máxima del rectángulo $a \times b$ se puede reducir a encontrar la más conveniente de $4p_i$ opciones.

   Los problemas a resolver entonces, son los $P(a,b)$ hasta $a \leq X$ y $b \leq Y$. Ahora, puesto que el problema $P(b,a)$ es equivalente a $P(a,b)$, podemos definir $P(b,a) = P(a,b)$, si $b \leq a$ para así tener que resolver menos problemas. También podemos homogeneizar el problema (no tener que poner condiciones) si definimos $P(a,b) = -\infty$ para $a < 0 \vee b < 0$ y, luego, $P(a,b) = 0$ para $a = 0 \vee b = 0$.

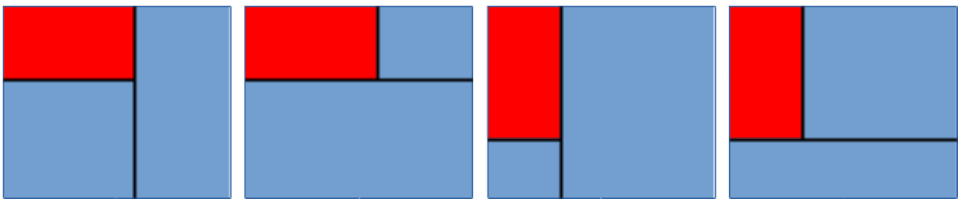

Figura 1: Existen 2 maneras diferentes (que valgan la pena) de colocar un rectángulo $a_i \times b_i$ en uno más grande y para cada una, dos maneras de realizar los cortes necesarios.

b) La fórmula para los $P(a,b)$ no definidos es la siguiente:

$$\begin{aligned} P(a,b) = \max_{i \in \{1..n\}} \{p_i + \max\{&P(a-a_i,b) + P(a_i,b-b_i),\\ &P(a,b-b_i) + P(a-a_i,b_i),\\ &P(a-b_i,b) + P(b_i,b-a_i),\\ &P(a,b-a_i) + P(a-b_i,a_i)\}\} \end{aligned}$$

c) Podemos utilizar el mismo orden que se usa en el problema de la *Edit distance*, haciendo a y b.

d) WIP.

---

7. *a*) La multiplicacion de factoriales rapidamente causa **overflow** (17! excede un `int32`)

b) $X!$ realiza $X-1$ multiplicaciones, tenemos:

- $n-1$ multiplciaciones en el numerador
- 1 resta, $n-k$, k multiplicaciones y una multiplicacion mas en el denominador ($1+n-k+k+1 = n+2$)
- 1 division

Con un total de $\Theta(2n+2)$

c) Vamos por paso:

1) Nos definimos la recurrencia desde las propiedades mencionadas:

$$\begin{aligned} &\texttt{CB}(n,k) = \texttt{CB}(n-1,k-1)\ \forall n > k\\ &\texttt{CB}(s,0) = 1 \forall 0 \le s \le n\\ &\texttt{CB}(s,s) = 1 \forall 0 \le s \le k \end{aligned}$$

2) De la recurrencia nos generamos la tabla:

$$\begin{vmatrix} \texttt{CB}(0,0) & \texttt{CB}(0,1) & \cdots & \texttt{CB}(0,k)\\ \texttt{CB}(1,0) & \texttt{CB}(1,1) & \cdots & \texttt{CB}(1,k)\\ \vdots & \vdots & \ddots & \vdots\\ \texttt{CB}(n-1,0) & \texttt{CB}(n-1,k-1) & \cdots & \texttt{CB}(n-1,k)\\ \texttt{CB}(n,0) & \texttt{CB}(n,1) & \cdots & \texttt{CB}(n,k) \end{vmatrix}$$

3) Implementado en el siguiente codigo:

Listing 1: ¿Que tiene de malo este código?

```
1  #! /bin/env python3
2  import numpy as np
3  def binom(n, k):
4      assert(n >= k) # Comprueba que estamos definidos
```

```
    bc = np.zeros((n+1,k+1)) # Inicializa la tabla
    # Casos base
    bc[:,0] = 1
    np.fill_diagonal(bc, 1)
    # El calculo....
    for i in range(1, n+1):
        for j in range(1, k+1):
            bc[i, j] = bc[i-1, j-1] + bc[i-1, j]
    return bc[n, k]
```

d) Un for de k iteraciones dentro de un for de n iteraciones: $\Theta(n \times k)$

8. Sea $\mathrm{F}(i,j)$ el número de monedas que llevamos recolectadas hasta la casilla $[i,j]$. Como el robot se mueve en movimientos horizontales y verticales por casillas adyacentes (Robi no sabe saltar), tenemos que elegir entre el recorrido que viene desde $\mathrm{F}(i-1,j)$(desde la izquierda) ó $\mathrm{F}(i,j-1)$(desde arriba):

   $$\mathrm{F}(i,j) = c_{i,j} + \max\{\mathrm{F}(i-1,j), \mathrm{F}(i,j-1)\}$$

   Donde $c_{i,j}$ es el dinero que nos aporta la moneda (0 si no hay moneda, 1 si la hay (por que todas las monedas son iguales aquí)). Nos definimos la tabla:

   $$\begin{vmatrix} F(0,0) & \cdots & F(n-1,0) & F(n,0) \\ F(0,1) & \cdots & F(n-1,1) & F(n,1) \\ \vdots & \ddots & \vdots & \vdots \\ F(0,k-1) & \cdots & F(n-1,k-1) & F(n,k-1) \\ F(0,k) & \cdots & F(n-1,k) & F(n,k) \end{vmatrix}$$

   Esto nos guarda la cantidad de monedas obtenidas, para recuperar el recorrido necesitamos guardar informacion de "de donde viene el F" que tomamos, no lo vamos a hacer por que enrealidad es otro problema.

```
#! /bin/env python3
import numpy as np
def robi(C):
    # Matriz que lleva la cuenta
    M = np.zeros(C.shape, dtype=int)
    # Conteo de monedas
    for r in range(C.shape[0]):
        for c in range(C.shape[1]):
            M[r, c] = np.max([M[r-1, c], M[r, c-1]]) + C[r,c]
    return M

# Tiramos monedas sobre un tablero de 5x6
tablero = np.random.randint(2, size=(6, 5), dtype=int)
print("Tablero:\n", tablero)
robi(tablero)
```

9. WIP.

6. Backtracking

Ejercicio 1.

El problema del *Closed Knight's path* consiste en encontrar una ruta en un tablero de ajedrez tal que un caballo recorra el tablero completo (cada uno de los casilleros una sola vez) volviendo al punto inicial. Escriba un algoritmo que determine tal camino en un tablero de 8x8.
Hint: El camino siempre existe en un tablero de 8x8

Ejercicio 2.

El *Problema de las cuatro reinas* consiste en encontrar posiciones para cuatro reinas en un tablero de 4×4 tal que ninguna pueda atacar a las demás. Muestre cómo puede obtener una solución a este problema, si es que existe.

Ejercicio 3.

Describa un algoritmo basado en backtracking para determinar si hay una asignación de valores a las variables booleanas x_i que hacen que la fórmula lógica $F(x_1, x_2, \cdots x_N)$ tome el valor verdadero.

Ejercicio 4.

Esboce un algoritmo para enumerar las permutaciones de $[1, n]$ que son desarreglos, vale decir, tales que $\pi(i) \neq i$ para todo $i \in [1, n]$.

Ejercicio 5.

Implemente en pseudocódigo un algoritmo de backtracking para encontrar un camino (lista de nodos) entre uno nodo y otro en un grafo dirigido.

Ejercicio 6.

Implemente en pseudocódigo un algoritmo de backtracking para resolver un sudoku.

Ejercicio 7.

Tiene una matriz `booleana` de tamaño $N \times M$, en la cual al cambiar el estado de una celda (de 0/False a 1/True o viceversa) todas las celdas en esa fila y columna tambien cambian de estado.
Desarrolle un algoritmo de Backtracking para que dada como entrada una matriz arbitraria, entregue la serie de celdas que se deben cambiar de estado para que toda la matriz quede nula (matriz de ceros).
HINT: El orden no importa

Ejercicio 8.

Dado un arreglo A[1..n] que tiene digitos enteros, queremos generar todas las permutaciones existentes.

Ejercicio 9.

Dado un multiconjunto de enteros, ¿existe un subconjunto no vacio cuya suma sea un numero X? Por ejemplo: para el conjunto $\{-7, -3, -2, 5, 8\}$ y con objetivo $X = 0$ la respuesta es **sí** puesto que $\{-3, -2, 5\}$ suman 0. Plantee un algoritmo de backtracking para responder si o no.

Backtracking - Soluciones

1.
```
Caballo (solucion, k) :
|--solucion <- solucion + [ k ]
|--Si todos los cuadrados han sido visitados :
|--|--retorna solucion
|--si no :
|--|--Para m cada movida desde k :
|--|--|--si m es legal :
|--|--|--|--P <- Caballo ( solucion , m) :
|--|--|--|--|--|--retorna P:
|--|--retorna Nulo
```

2. WIP.

3. WIP.

4. Aplicando directamente la idea de backtracking resulta: Se llama originalmente con argumento 1.

```
free ← {1,2,...,n}
function derangements (k)
|
if k = n then
|   for i ← 1 to n do
|   |   print (d [i ])
|   end
else
|   for i ∈ free do
|   |   free ← free ∖ {i}
|   |   d[k] ← i
|   |   derangements(k + 1)
|   |   free ← free ∪ {i}
|   end
end
```

5.
```
CaminoA(path,grafo,fin):
-si path[-1]== fin:
--retornar path
-sino:
--S'= todos los arcos (path[-1],i) del grafo
---tal que i no está en path
--mientras |S'| >0:
---ai= algun elemento de S'
---borrar ai de S'
---sol = CaminoA(path+[ai],grafo,fin)
---si sol no es NULA:
----retornar sol
--retornar NULA
```
Sin embargo, para que esto funcione, tenemos que llamar inicialmente al algoritmo con un camino que incluye el nodo inicial, llamémoslo ini: CaminoA([ini], grafo, fin)

6.
```
Sudoku(matrx):
-si no quedan celdas en blanco:
```

```
--retornar matrx
-sino:
--S'= todos los números que se pueden poner
---en la siguiente celda vacía
--mientras |S'| >0:
---ai= algun elemento de S'
---borrar ai de S'
---matrx'= matrx con ai en la siguiente
-----celda vacía.
---sol = Sudoku(matrx')
---si sol no es NULA:
----retornar sol
--retornar NULA
```

7. WIP.

8. Nos aprovechamos de las funciones *nesteadas* para guardar todas las soluciones que vamos encontrando

```python
#! /bin/env python3
def all_permutations(A):
    all_sol = []

    def perm(A, sol):
        if len(A) < 1:
            all_sol.append(sol)
        else:
            for i in range(len(ls)):
                perm(A[0:i] + A[i+1:], sol + [A[i]])

    perm(A, [])
    return all_sol

all_permutations([0,1,2])
```

9. Algo bien simple, tomamos el primer elemento y lo probamos recursivamente contra los demas, al elegir el primer elemento $\hat{p}$ quedan dos opciones:

   - Existe un subset que contiene a $\hat{p}$ y suma X
   - Existe un subset que no contiene a $\hat{p}$ y suma X

Listing 2: ¿Como puede hacer que retorne el subset encontrado?

```python
#! /bin/env python3
def subset_sum(A, X):
    if sum(A) == X:
        # Si estan sumando X...
        return True
    elif len(A) == 0:
        # Conjunto vacio no permitido
        return False
    else:
        # Existe un subset que incluye A[0] o no lo incluye y suma X
        return subset_sum(A[1:], X-A[0]) or subset_sum(A[1:], X)
```

7. Diseño de algoritmos

Ejercicio 1.

Suponga que se tiene un arreglo con las alturas de n edificios, se dice que el edificio en la posición i tiene *vista al mar* si todos los edificios en posiciones $j < i$ tienen menor altura que ese.
Suponga que se quiere obtener un arreglo de n *bools* que indica para cada edificio si tiene *vista al mar* o no. Escriba en pseudocódigo un algoritmo con tiempo de ejecución $O(n)$ para lograr esto.

Ejercicio 2.

El problema del *Convex Hull* consiste en encontrar el menor polígono convexo capaz de contener un determinado conjunto de puntos. Esto es, dado un conjunto de puntos, encerrarlos en un polígono cuyos vértices correspondan a la menor cantidad posible de puntos de este conjunto. Diseñe un algoritmo para resolver este problema y analice su complejidad, trate de llevarlo al menos a $O(n^2)$.

Ejercicio 3.

Suponga que tiene un arreglo de enteros tal que varios enteros se repiten, esboce un algoritmo para eliminar los enteros duplicados que corra en $O(n \log n)$.

Ejercicio 4.

Dados dos arreglos de tamaño n y m, ordenados, esboce un algoritmo de complejidad $O(\log k)$ para encontrar el k elemento más pequeño en la unión de ambos, para cualquier valor de $k \leq n + m$.

Ejercicio 5.

Un programador novato inventó el método de ordenamiento del siguiente listado:

Listing 3: really-bad.c – a sorting algorithm

```
1  static inline void swap(double *a, double *b){
2          double tmp;
3          tmp = *a; *a = *b; *b = tmp;
4  }
5
6  void sort(double a[], int n) {
7          int i;
8          for(i = 0; i < n - 1; i++){
9                  if(a[i] > a[i + 1]){
10                         swap(&a[i], &a[i + 1]);
11                         i = 0;
12                 }
13         }
14 }
```

Nótese que cada vez que intercambia `a[i]` con `a[i + 1]` repone `i` al comienzo del arreglo.

1. Demuestre (informalmente) que este programa realmente ordena.
2. Halle el orden del tiempo de ejecución mínimo y máximo del programa.
3. Evalúe el número promedio de comparaciones entre elementos y el número promedio de intercambios al ordenar arreglos de n elementos.

Ejercicio 6.

Un problema popular en entrevistas para trabajo es determinar rápidamente (tiempo mejor que $O(n)$) en un arreglo ordenado que contiene elementos en pares, salvo un único elemento que no está en un par, cuál es el elemento que no se repite.

Pista: Un rango de elementos repetidos debe tener largo par.

Diseño de algoritmos - Soluciones

1. WIP.
2. A continuación, como referencia, el algoritmo de **Jarvis March**:

   - Se define `orientation(p, q, r)` como una operación que retorna si los puntos p, q y r se encuentran en sentido horario o anihorario.
   - Tomo el punto de más a la izquierda a.
   - Encuentro el único punto b para el cual `orientation(a, b, r)` retorne antihorario para todo otro punto r.
   - Tomo este como mi nuevo a y repito la operación hasta llegar al punto inicial.

   Complejidad: $O(nh)$, siendo n la cantidad de puntos y h la cantidad de vertices del *Convex Hull*. El peor de los casos es cuando todos los puntos deben pertenecer al *Convex Hull*, con complejidad $O(n^2)$.
3. WIP.
4. WIP.
5. WIP.
6. WIP.

8. Dividir y conquistar

Ejercicio 1.
Analice la complejidad de los siguientes algoritmos:

- Cada problema se divide en 8 subproblemas, cada uno con la mitad de los datos de entrada. Unir los subproblemas toma un tiempo de $1000n^2$.
- Cada problema se divide por la mitad, y la unión toma un tiempo de $10n$.
- Cada problema se divide por la mitad, pero unir las soluciones toma $\frac{n^2}{2}$.

Ejercicio 2.
¿Cuántas veces imprime la siguiente función? Puede asumir que n es una potencia de 2.

Listing 4: Javascript

```
function f(n){
    if( n > 1 ){
        console.log("Still going...");
        f(n/2);
        f(n/2);
    }
}
```

Ejercicio 3.
Dado un arreglo de elementos distintos ordenados $A[1,...,n]$, diseñar un algoritmo que detecta si existe algún elemento i tal que $A[i] = i$ en $O(\log n)$.

Ejercicio 4.

Suponga que se un arreglo de n elementos *igualables* pero no *comparables*, osea, que no se puede preguntar por relaciones de *menor* o *mayor*, pero si de *igual*. Se dice que un elemento tiene *mayoría* si este elemento se repite más de $n/2$ veces. Diseñe un algoritmo que corra en $O(n)$, demuestre que funciona y que esa es su complejidad.
Hint:

- Agrupe los elementos en pares, para obtener $n/2$ pares.
- Observar cada par: si los dos elementos son diferentes, descartarlos, si son iguales, quedarse sólo con uno de ellos.

Ejercicio 5.

El método tradicional para multiplicar dos matrices de $N \times N$ demanda $O(n^3)$ operaciones.
Descomponiendo las matrices de $N \times N$ en 4 matrices de $N/2 \times N/2$, y usando una fórmula que permite evaluar el producto de dos matrices de 2×2 en 7 multiplicaciones, Strassen obtuvo un algoritmo más eficiente aplicando esta idea recursivamente.
Analice el algoritmo de Strassen.

Ejercicio 6.

Usando dividir y conquistar, derive un algoritmo para encontrar simultáneamente el mínimo y el máximo de los elementos de un arreglo de N elementos.
Derive el orden de complejidad de su algoritmo.
¿Vale la pena complicarse la vida así?

Ejercicio 7.

Plantee y analice un método basado en dividir y conquistarpara obtener la suma máxima de un rango de los elementos de un arreglode N elementos. Por ejemplo, en el arreglo

```
31 -41 59 26 -53 58 97 -93 -23 84
```

la suma máxima se obtiene entre el tercer elemento (59) y el séptimo (97).

Ejercicio 8.

La manera "obvia" de calcular el número de Fibonacci $F_n$es usar directamente la recurrencia:

$$F_{n+2} = F_{n+1} + F_n \qquad F_0 = 0, F_1 = 1$$

Explique cómo calcular cuántas llamadas a la función Fse hacen para calcular F_n con esta estrategia.

Ejercicio 9.

Para cierto problemacuenta con tres algoritmos alternativos:

Algoritmo A: Resuelve el problema dividiéndolo en cinco problemasde la mitad del tamaño,los resuelve recursivamentey combina las soluciones en tiempo lineal.

Algoritmo B: Resuelve un problema de tamaño $n$resolviendo recursivamente dos problemas de tamaño $n-1$y combina las soluciones en tiempo constante.

Algoritmo C: Divide el problema de tamaño $n$en nueve problemas de tamaño $n/3$,resuelve los problemas recursivamentey combina las soluciones en tiempo $O(n^2)$.

¿Cuál elije si n es grande,y porqué?

Ejercicio 10.

En un arreglo a se dice que la posición i es un *mínimo local* si $a[i]$ es menor a sus vecinos,o sea si $a[i-1] > a[i]$ y $a[i] < a[i+1]$.Decimos que 0 es un mínimo local si $a[0] < a[1]$,y similarmente $n-1$ si $a[n-2] > a[n-1]$ (los extremos tienen un único vecino). Dado un arreglo a de n números distintos,diseñe un algoritmo eficiente basado en dividir y conquistarpara hallar un mínimo local (pueden haber varios). Justifique su algoritmo, y derive su complejidad aproximada.

Ejercicio 11.

Plantee y analice un método basado en dividir y conquistarpara obtener la multiplicación de dos números binarios de $n$dígitos en un tiempo menor a la solución directa ($O(n^2)$).

Dividir y conquistar - Soluciones

1.
   - Tenemos la recurrencia $T(2n) = 8T(n) + 1000n^2$. En base a esto sacamos:
     - $a = 8$
     - $b = 2$
     - $\log_b a = 3$
     - $f(n) = O(n^c)$, con $c = 2$.

     Como $\log_b a > c$ nos queda $T(n) = \Theta(n^{\log_b a}) = \Theta(n^3)$.
   - Tenemos la recurrencia $T(2n) = 2T(n) + 10n$. En base a esto sacamos:
     - $a = 2$
     - $b = 2$
     - $\log_b a = 1$
     - $f(n) = \Theta(n^{\log_b a} \log^\alpha n)$, con $\alpha = 0$.

     Como $\alpha > -1$ nos queda $T(n) = \Theta(n^{\log_b a} \log^{\alpha+1} n) = \Theta(n \log n)$.
   - Tenemos la recurrencia $T(2n) = 2T(n) + \frac{n^2}{2}$. En base a esto sacamos:
     - $a = 2$
     - $b = 2$
     - $\log_b a = 1$
     - $f(n) = \Omega(n^c)$, con $c = 2$.

     Primero comprobamos que exista k tal que $af(\frac{n}{b}) < kf(n)$ y $k < 1$. En este caso:

     $$2 \cdot \frac{1}{2}\left(\frac{n}{2}\right)^2 < k\frac{n^2}{2} \quad \Rightarrow \quad k > \frac{1}{2}$$

     se cumple para k en el intervalo $(\frac{1}{2}, 1)$. Como $\log_b a < c$ nos queda $T(n) = \Theta(f(n)) = \Theta(n^2)$.

---

2. Primero modelamos la recurrencia de la cantidad de veces que imprime el algoritmo

   $$T(1) = 0 \qquad (\rightarrow t_0 = 0)$$
   $$T(2n) = 1 + 2T(n) \qquad (\rightarrow t_{k+1} = 1 + 2t_k)$$

   Definimos la función generatriz ordinaria $A(z)$ para resolver esta recurrencia.

   $$A(z) = \sum_{k \geqslant 0} t_k z^k$$
   $$\frac{A(z) - t_0}{z} = 1 + 2 \cdot A(z)$$
   $$A(z) = \frac{z}{1-2z} = \sum_{k \geqslant 0} 2^k z^{k+1} = \sum_{k \geqslant 1} 2^{k-1} z^k$$
   $$t_k = [z^k]\, A(z) = 2^{k-1}; \quad k \geqslant 1$$

   Como $k = \log_2 n$:

   $$T(n) = 2^{\log_2 n - 1} = \frac{n}{2}$$

---

3. WIP.

---

4. WIP.

---

5. WIP.

```
function F (n)
|
if n ≤ 1 then
|  return 1
else
|  return F (n- 1) + F (n- 2)
end
```

6. WIP.
7. WIP.
8. Lo sugerido en la pregunta corresponde a: Llamemos t_n al número de llamadas a F al invocar $F(n)$. Se ve que si $n = 0$ o $n = 1$ se hace una llamada a F, si $n \geq 2$ se hacen $t_{n-1} + t_{n-2}$ llamadas recursivas. O sea, tenemos la recurrencia:

   $$t_{n+2} = t_{n+1} + t_n \qquad t_0 = t_1 = 1$$

   Para resolver esta recurrencia,recurrimos a funciones generatrices:

   $$T(z) = \sum_{n \geq 0} t_n z^n$$

   Aplicando las propiedades de funciones generatrices ordinarias:

   $$\frac{T(z) - t_0 - t_1 z}{z^2} = \frac{T(z) - t_0}{z} + T(z) + \frac{1}{1-z}$$

   Substituyendo los valores iniciales y despejando tenemos:

   $$\begin{aligned} T(z) &= \frac{1 - z + z^2}{(1-z)(1-z-z^2)} \\ &= \frac{2}{1-z-z^2} - \frac{1}{1-z} \end{aligned}$$

   Sabiendo que la función generatriz de los números de Fibonacci es:

   $$F(z) = \frac{z}{1-z-z^2}$$

   al ser $F_0 = 0$ resulta la función generatriz de los F_{n+1}:

   $$\frac{F(z) - F_0}{z} = \frac{1}{1-z-z^2}$$

   lo que nos permite concluir:

   $$t_n = 2F_{n+1} - 1$$

9. Veamos cada algoritmo por turno.

   a) El tiempo de ejecución cumple:

   $$T_A(2n) = 5T_A(n) + cn$$

   Por el teorema maestro,con $a = 5$, $b = 2$, $d = 1$, estamos en el caso $a > b^d$:

   $$T_A(n) = \Theta(n^{\log_2 5})$$

b) La recurrencia es:

$$T_B(n) = 2T_B(n-1) + c \quad T_B(0) = \ldots$$

Usando funciones generatrices, definimos:

$$g(z) = \sum_{n\geq 0} T_B(n) z^n$$

Tenemos por las propiedades del caso:

$$\frac{g(z) - T_B(0)}{z} = 2g(z) + \frac{c}{1-z}$$
$$g(z) = \frac{T_B(0)}{1-2z} + \frac{cz}{(1-z)(1-2z)}$$

Usando el truco para fracciones parciales:

$$\frac{z}{(1-z)(1-2z)} = \frac{A}{1-z} + \frac{B}{1-2z}$$
$$A = \lim_{z\to 1} \frac{z}{1-2z}$$
$$= -1$$
$$B = \lim_{z\to 1/2} \frac{z}{1-z}$$
$$= 1$$

y queda:

$$\begin{aligned} T_B(n) &= [z^n]g(z) \\ &= T_B(0)\cdot 2^n - c + c\cdot 2^n \\ &= (T_B(0)+c)\cdot 2^n - c \\ &= \Theta(2^n) \end{aligned}$$

Más fácil: el valor de A, B es irrelevante, mientras sean diferentes de cero.Directamente vemos el resultado.

c) La recurrencia es:

$$T_C(3n) = 9T_C(n-1) + cn^2 \quad T_C(0) = \ldots$$

Se aplica el teorema maestro,con $a = 9$, $b = 3$, $d = 2$, o sea es el caso $a = b^d$:

$$T_C(n) = \Theta(n^2 \log n)$$

En resumen:

$$T_A(n) = \Theta(n^{\log_2 5})$$
$$T_B(n) = \Theta(2^n)$$
$$T_C(n) = \Theta(n^2 \log n)$$

Es claro que $\log_2 5 > 2$, para n muy grande gana C. Podría ser que para valores "razonables" de n tenga ventaja A. El algoritmo B está fuera de discusión para n grande.

---

10. Para dividir y conquistarnos conviene dividir el arreglo en mitades iguales. En consecuencia, consideremos el elemento medio, $m = \lfloor n/2 \rfloor$ y sus vecinos. Hay tres posibilidades:

a) Si $a[m-1] > a[m]$ y $a[m] < a[m+1]$, hemos hallado un mínimo local.Retorne m.

b) Si $a[m-1] > a[m] > a[m+1]$, tiene que haber un mínimo local en la segunda mitad del arreglo(en el peor caso,todos los elementos disminuyen, y el último es mínimo local).

c) En forma similar,si $a[m-1] < a[m] < a[m+1]$, tiene que haber un mínimo local en la primera mitad del arreglo (en el peor caso,todos los elementos aumentan, y el primero es mínimo local).

Esto es similar a búsqueda binaria. La complejidad cumple:

$$T(n) = T(n/2) + 1$$

y el teorema maestro dice que $T(n) = \Theta(\log n)$.

---

11. Para encontrar una solución consideremos dividir los números porla mitad (tamaño $n/2$) y trabajar con estos sub-numeros resultantes.

$$x = x_1 \cdot 2^{\frac{n}{2}} + x_0$$
$$y = y_1 \cdot 2^{\frac{n}{2}} + y_0$$
$$xy = x_1 y_1 \cdot 2^n + (x_0 y_1 + x_1 y_0) \cdot 2^{\frac{n}{2}} + x_0 y_0$$

Unir las soluciones consiste en realizar 3 sumas y 2 multiplicaciones, cada una incluyendo una cantidad de dígitos que depende linealmente de n.Por lo tanto, la operación *merge* ocupa tiempo lineal ($O(n)$). Utilizamos el teorema maestro, modelando el problema de la siguiente manera:

$$T(2n) = 4T(n) + kn$$

sin embargo, esto sigue arrojando un tiempo $O(n^2)$.
Podemos reescribir la suma de en medio para tratar de disminuir la cantidad desubproblemas.

$$x_0 y_1 + x_1 y_0 = (x_0 + x_1)(y_0 + y_1) - x_0 y_0 - x_1 y_1 \Rightarrow xy = x_1 y_1 \cdot 2^n + ((x_0 + x_1)(y_0 + y_1) - x_0 y_0 - x_1 y_1) \cdot 2^{\frac{n}{2}} + x_0 y_0$$

Tengo ahora solo 3 subproblemas que calcular recursivamente.

$$T(2n) = 3T(n) + kn = O(\log_2 3)$$

---

9. Grafos y búsqueda en grafos

Ejercicio 1.
El juego de escaleras y serpientes consiste en una serie de celdas numeradas de 1 a $n \times n$ por las que se avanza con una ficha que se puede mover hasta k celdas más adelante en cada turno, algunas celdas tienen *escaleras* entre la celda a y la b tal que $a < b$, si la ficha cae en la celda a se mueve a la b; otras celdas tienen *serpientes* que funcionan de la misma manera sólo que $b < a$. Describa un algoritmo que compute la menor cantidad de movimientos necesarios para llegar desde la primera a la última celda con la ficha.

Ejercicio 2.
El *radio* de un árbol es la máxima distancia entre una hoja y la raíz.
Escriba un algoritmo que dado un grafo cualquiera, encuentre el *spanning tree* de mínimo radio.

Ejercicio 3.
Un grafo no dirigido $G = (V, E)$ se dice *bipartito* si se puede particionar V en conjuntos disjuntos no vacíos V_1 y $V_2 = V - V_1$ con las propiedades:

- Ningún par de vértices en V_1 son adyacentes en G

- Ningún par de vértices en V_2 son adyacentes en G

Dé un algoritmo que determine si un grafo es bipartito. Si lo es, su algoritmo deberá entregar una partición de V que cumpla las propiedades dadas.

Ejercicio 4.
Para los siguientes problemas en grafos, indique qué tipo de búsqueda usaría para resolverlos, indicando la razón de su preferencia:

- Encontrar componentes conexos del grafo
- Determinar el camino más corto(en término de arcos)entre dos vértices dados
- Encontrar los ciclos de un grafo

Ejercicio 5.
Describa un algoritmo para encontrar los *componentes conexos*de un grafo, vale decir, grupos de vértices que están conectados entre sí.
Explique el diseño de su algoritmo.

Ejercicio 6.
A veces se requieren árboles recubridores "livianos" que cumplen condiciones especiales.Por ejemplo,buscamos el árbol recubridor de mínimo costodel grafo $G = (V, E)$ con la condición que los vértices en $U \subset V$ son todos hojas. Plantee un algoritmo eficiente para resolver este problema, y demuestre que es correcto.
Pista: ¿Qué queda si de la solución óptima se eliminanlos vértices en U?

Ejercicio 7.
En su publicación original, Kruskal propuso otro algoritmo para hallar un árbol recubridor mínimo(*minimal spanning tree* en inglés), conocido como *reverse delete*: Dado un grafo conexo, sucesivamente se elimina un arco de mayor costo que no desconecta el grafo. Demuestre que este algoritmo es correcto.

Ejercicio 8.

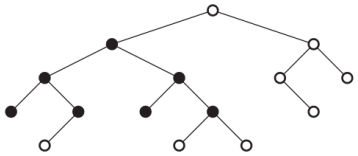

Figura 2: Un árbol binario con un máximo subárbol completo

Esboce un algoritmo recursivoque dado un árbol binarioentregue la raíz y la altura de un máximo subárbol binario completo.En el ejemplo de la figura, uno de ellos está marcado en negro,su altura es 2.
Pista: Considere un recorrido en postorden.

Grafos y búsqueda en grafos - Soluciones

1. WIP.
2. WIP.
3. WIP.
4. WIP.
5. WIP.
6. WIP.
7. WIP.
8. La idea es que el mayor subárbol completoque tiene la raíz en un nodo dadoestá formado por los máximos subárboles completos con raíz en sus hijos, podados al mínimo de sus dos alturas, junto a este nodo. Mientras calculamos éstos en un recorrido en postorden,registramos en variables globales los máximos hallados hasta el momento. Resulta natural asignar "altura" -1 a los punteros nulos, da altura 0 para las hojas en forma automáticamente. Esto da el algoritmo siguiente:

```
maxtree ← nil
maxheight ← −1

procedure subtree (root, height)
|
if root = nil then
|   height ← −1
else
|   subtree(root− > left, hl)
|   subtree(root− > right, hr)
|   height ← 1 + min(hl, hr)
|   if height > maxheight then
|   |   maxheight ← height
|   |   maxtree ← root
|   end
end
```

---

10. Union-find

Ejercicio 1.
Demuestre que después de una cantidad arbitraria de operaciones unión (sin *Path Compression*), la complejidad en el peor caso de *find* o de *union* es $O(\log n)$.

Ejercicio 2.
Suponga que se tiene una secuencia de bits $X[1..n]$, los cuales son inicialmente 1, sobre la que se pueden realizar sólo las siguientes operaciones:

- **Lookup**(i): Entrega el bit en la posición i.
- **Blacken**(i): Deja en 0 el bit en la posición i, no se puede aplicar en el último bit, cuando $i = n$.
- **NextWhite**(i): Entrega la posición $j \leq i$ del siguiente bit que vale 1 en la secuencia.

Describa una implementación en que el costo de cada operación sea mejor que $O(n)$ y demuestre que ese es el caso.

Ejercicio 3.

Suponga que se tiene un bosque de árboles que representan una partición del conjunto $\{1,2,...,n\}$ sin conocer cómo están construidos los árboles. Se busca construir un arreglo $label[1..n]$ tal que $label[i] = Find(i)$.

- ¿Cuál es la complejidad del peor caso, si Ud. decide llenar todas las entradas de $label$ usando $find$ y no hay *path-compression*? Muestre los tipos de árboles en que se daría.
- Muestre que la complejidad del peor caso en la misma situación anterior pero con *path-compression* es $O(n)$.
- Muestre cómo lograr esto en complejidad $O(n)$ usando búsqueda en amplitud o búsqueda en profundidad.

Union-find - Soluciones

1. La complejidad de una operación *find* está determinada por la altura del árbol que representa el conjunto, por lo tanto hay que demostrar que esta altura no supera $O(\log n)$.

   En primer lugar hay que recordar que el *rank* de un nodo representa la altura que tiene un nodo (cuando no hay *Path Compression*).

   El *rank* de un nodo sólo sube cuando se realiza una operación *union* entre dos árboles de igual *rank*.

   Para crear un árbol con raíz *rank* 1 es necesario unir dos árboles con raíz *rank* 0, para crear un árbol con raíz *rank* 2, es necesario unir dos árboles con raíz *rank* 1, y así sucesivamente.

   Entonces, es posible demostrar por inducción, que se necesitan al menos 2^n nodos para lograr un árbol con raíz *rank n*.

   - **Caso base**: Es fácil ver que se cumple para $n = 0$, pues para tener un árbol raíz *rank* 0 se necesita tener un nodo (la raíz).
   - **Hipótesis inductiva**: Supongamos que se cumple que un árbol raíz *rank p* tiene al menos 2^p nodos.
   - **Tesis inductiva**: La única forma de lograr un árbol raíz *rank* $p+1$ es uniendo dos árboles raíz *rank p*, por hip. inductiva, cada uno tiene al menos 2^p nodos, por lo que el nuevo árbol tiene al menos 2^{p+1} nodos, lo que ser quería demostrar.

   Como un árbol no puede tener más de n nodos (este sería el caso en que todo el universo esté en el mismo árbol), la altura máxima que puede tener un árbol es $\log_2 n$, y esto limita el costo de una operación *find*.

   Por otro lado, la operación *union* consta de dos operaciones *find* y el resto es $O(1)$, por lo tanto esta también es $O(\log n)$.

2. WIP.
3. WIP.

11. Hashing

WIP.

12. Análisis amortizado

Ejercicio 1.
Suponga una secuencia de operaciones numeradas $1,2,3,\ldots$ tal que la operación i tiene costo 1 si i no es una potencia de 3, mientras el costo es $i+1$ si i es una potencia de 3. ¿ Cuál es una cota ajustada del costo amortizado de las operaciones?

Ejercicio 2.
Un arreglo dinámico (que se utiliza como *stack*) tiene L elementos que entran en un espacio de memoria de tamaño S, cuando se inserta un elemento nuevo y $L = S$, entonces se tiene que duplicar S la operación es costosa pues se tienen que ubicar todos los L elementos en una posición de memoria nueva.Por otro lado, cuando L disminuye mucho y queda que $L \leq S/4$, también se reubican todos los L elementos, en una posición de memoria con la mitad del tamaño, osea, S disminuye a la mitad.

- Si sólo hay operaciones de eliminación ¿Cuál es el costo amortizado de estas?
- ¿El orden de complejidad anterior sigue siendo una cota si hay operaciones de inserción y eliminación y estas ocurren al azar?

- Si el tamaño del arreglo se redujera a la mitad cuando, después de una eliminación, resultara $L \leq S/2$ en vez de $L \leq S/4$ ¿Qué desventajas tendría esto?

Ejercicio 3.
Para guardar vectores N-dimensionales se utiliza un arreglo *principal* de memoria de tamaño muy grande (siempre suficiente), este arreglo tiene asociada una matriz también de tamaño muy grande que se utilizará para guardar las distancias entre cada par de vectores.Existen dos operaciones:

- **Insertar** un vector en la siguiente posición del arreglo principal.
- **Obtener la matriz de distancias**, para lo cuál primero se calculan las celdas de la matriz de distancias y luego se entrega un puntero, los resultados de las celdas calculadas se dejan para que la próxima vez que se haga esta operación no sea necesario calcularlos.

Encuentre el costo amortizado por operación de una secuencia de estas operaciones y demuéstrelo.

Ejercicio 4.
La estructura de datos *Ordered stack* almacena una secuencia de elementos y tiene las siguientes dos operaciones:

- **Ordered push**(x): Se eliminan todos los elementos menores que x al principio de la secuencia y luego coloca x, también al principio de la secuencia.
- **Pop**: elimina el primer elemento de la secuencia y luego lo retorna.

Suponiendo que esta estructura se implementa con una lista enlazada, probar que, al comenzar con una estructura vacía, el costo de amortizado de cualquier operación es $O(1)$.

Ejercicio 5.
Suponga que se tiene un número en binario, inicialmente 0, sobre el que se realiza la operación de sumar 1, básicamente cambiando todos los 1 que haya desde la derecha a la izquierda por 0 hasta encontrar el primer 0 y cambiarlo por 1. ¿Cuál es el costo amortizado de esta operación si se quiere contar de 0 a m?
Nota: Puede serle más fácil si asume, sin pérdida de generalidad, que $m = 2^k$.

Ejercicio 6.
Suponga que se se aplicará una serie de n operaciones sobre una estructura de datos, si $f(k)$ denota el tiempo de ejecución de la operación k, determine el costo amortizado resultante de estas.

1. $f(k)$ es el mayor entero i tal que 2^i divide k.
2. $f(k)$ es la mayor potencia de 2 que divide k.
3. $f(k) = k$ si k es un cuadrado perfecto, $f(k) = 1$, en caso contrario.
4. $f(k) = k$ si k es un número de Fibonacci, $f(k) = 1$ en caso contrario.
5. Siendo T un árbol binario completo que almacena los enteros de 1 a n, $f(k)$ es la cantidad de ancestros del nodo k.
6. Siendo T un árbol binario arbitrario, $f(k)$ es el largo del camino entre los nodos $k-1$ y k.

Ejercicio 7.
Una **cola** se implementa con una lista enlazada, donde se almacenan los punteros al inicio y al final (de manera que las operaciones **Push**(x) y **Pop**() tienen complejidad $O(1)$.

- Si se agrega la operación **MultiPop**(r) que elimina r elementos al frente de la cola y entrega el último eliminado, demuestre que cualquier secuencia de n operaciones **Push**(x) y **MultiPop**(r) sigue teniendo costo amortizado $O(1)$.
- Si se agrega ahora la operación **MultiPush**(x, r), que agrega r copias de x al final de la lista, demuestre que cualquier secuencia de operaciones **MultiPop**(r) y **MultiPush**(x, r) tiene costo amortizado $O(r)$.
- Describa una estructura de datos que permita realizar secuencias arbitrarias de **MultiPush** y **MultiPop**(r), que no ocupe más que una cantidad constante de memoria por cada elemento en la cola.
- Si se crea la operación **Decimate**() que recorre la lista enlazada eliminando todos los elementos en posiciones múltiplos de 10 a partir del inicio, demostrar que, comenzando con una cola vacía, el costo amortizado de una secuencia de n operaciones **Push**(x), **Pop**() y **Decimate**() tiene complejidad $O(1)$.

Ejercicio 8.
Se tiene una **hashtable** en la que se puede insertar o eliminar elementos en $O(1)$, para asegurarse de que la tabla es suficientemente grande y que no gaste mucha memoria, se utilizan las siguientes reglas de reconstrucción:

- Si después de una inserción la tabla está más de $3/4$ llena, los elementos se mudan a una nueva tabla con el doble del tamaño.
- Si después de una eliminación la tabla está menos de $1/4$ llena, los elementos se mudan a una nueva tabla con la mitad del tamaño.

Demostrar que para cualquier secuencia de inserciones o eliminaciones el costo amortizado por operación sigue siendo $O(1)$.

Ejercicio 9.
Una QUEUE se implementa mediante dos STACK.La idea es que ENQUEUE pone un objeto en el tope del primer stack; DEQUEUE saca el elemento del tope del segundo, si éste está vacío transfiere todos los elementos del primero al segundo (vía POP y PUSH). Suponiendo que las operaciones sobre un STACK(PUSH, POP, EMPTY) todas tienen costo 1, use análisis amortizado para demostrar que las operacionessobre QUEUE tienen costo amortizado $O(1)$.
Pista: Una función potencial $\Psi(D)$ es el número de elementosen el primer stack.

Ejercicio 10.
Cierto algoritmo requiere trabajar con matrices cuadradas. Lamentablemente no se puede predecir cuál es el tamaño final requerido, por lo que ocasionalmente necesita extenderlas. Proponga un esquema eficiente de extensión de las matrices y obtenga el costo amortizado al extender las matrices desde 1×1 hasta $n \times n$.

Análisis amortizado - Soluciones

1. Sospechamos que el costo de hacer n operaciones es $O(n)$ pues se parece al caso de un arreglo de tamaño dinámico.

   Para demostrar eso, lo que se debe demostrar es que aumentando nuestro saldo una cantidad $O(n)/n = O(1)$ fija de monedas en cada operación, siempre se tendrán monedas para pagar las operaciones, incluso las caras.

   Probamos con un número $O(1)$ de monedas que parezca lógico, por ejemplo, 3 (osea, que cada operación de inserción se paga a sí misma y aporta 2 más para que hayan suficientes monedas para la operación grande).

   Entonces, cuando hay una inserción el saldo aumenta en 2 y cuando i es una potencia de 3, el saldo aumenta en $(-i+2)$ (en realidad disminuye).

   Demostraremos que la operación ($i = 3^k$) se puede pagar con las operaciones entre $(3^{k-1}+1)$ y (3^k), pues estas generan:

   $$\begin{aligned} &3(3^k - 3^{k-1}) && \text{monedas} \\ &= 2 \cdot 3^k && \text{monedas} \end{aligned}$$

   Descontando las $3^k - 3^{k-1} - 1$ necesarias para pagar las inserciones:

   $$\begin{aligned} &2 \cdot 3^k - 3^k + 3^{k-1} + 1 && \text{monedas} \\ &= 3^k + 3^{k-1} + 1 && \text{monedas} \end{aligned}$$

   Las que son suficientes para pagar la operación de inserción, que cuesta 3^k+1, incluso sobran monedas.

   Nota: En este caso sobraron monedas y podríamos haber hecho que sobraran aun más diciendo, por ejemplo, que cada operación cuesta 9001 (seguiría siendo $O(1)$ y la demostración seguiría siendo válida), pero generalmente queda más clara cuando elegimos la cantidad exacta para que no sobre, en este caso habrían sido 5/2 monedas, este radio (con el que no sobran monedas) entrega una razón entre el costo amortizado y el costo de la operación simple.

2. WIP.
3. WIP.
4. WIP.
5. WIP.
6. WIP.
7. WIP.
8. WIP.
9. WIP.
10. WIP.

13. Generatrices multivariadas y método simbólico

Ejercicio 1.
Podemos definir secuencias binarias que no contienen 1 seguidos mediante:

$$\mathscr{S} = SEQ(\{0, 10\}) \times \{\epsilon, 1\}$$

1. ¿Cuántas secuencias de éstas hay de largo n? Use el método simbólico para responder a esta pregunta.
2. Halle una fórmula para el número promedio de 1 en las secuencias de largo n usando el método simbólico con la clase $\mathscr{Z}$ contando largos y $\mathscr{U}$ contando unos (o sea, 10 es representado por $\mathscr{Z}^2 \times \mathscr{U}$, el largo es 2 y hay 1 uno). Use un paquete de álgebra simbólica para obtener los 10 primeros valores.

Generatrices multivariadas y método simbólico - Soluciones

1. *a*) De la descripción de la clase obtenemos directamente:

$$\begin{aligned} S(z) &= \frac{1}{1-(z+z^2)} \cdot (1+z) \\ &= \frac{1+z}{1-z-z^2} \end{aligned}$$

Vemos que $S(z)$ es la suma de las funciones generatrices de los números de Fibonacci F_n y F_{n+1}, o sea:

$$\begin{aligned} s_n &= F_n + F_{n+1} \\ &= F_{n+2} \end{aligned}$$

Podemos confirmarlo:

$$\begin{aligned} \frac{F(z)-F_0-F_1 z}{z^2} &= \frac{1}{z^2}\left(\frac{z}{1-z-z^2} - z\right) \\ &= \frac{1+z}{1-z-z^2} \end{aligned}$$

b) Acá la descripción de la clase es algo más compleja:

$$\mathcal{S} = SEQ(\mathcal{Z} + \mathcal{Z}^2 \times \mathcal{U}) \times (\mathcal{E} + \mathcal{Z} \times \mathcal{U})$$

Lo que nos lleva directamente a:

$$\begin{aligned} S(z,u) &= \frac{1}{1-(z+z^2 u)} \cdot (1+zu) \\ &= \frac{1+zu}{1-z-z^2 u} \end{aligned}$$

De acá:

$$\begin{aligned} S_u(z,u) &= \frac{z}{(1-z-z^2 u)^2} \\ S_u(z,1) &= \frac{z}{(1-z-z^2)^2} \end{aligned}$$

Nos interesan los valores:

$$\frac{[z^n]S_u(z,1)}{[z^n]S(z,1)} = \frac{[z^n]S_u(z,1)}{F_{n+2}}$$

El programa Maxima adjunto calcula los valores:

$$\frac{1}{2}, \frac{2}{3}, 1, \frac{5}{4}, \frac{20}{13}, \frac{38}{21}, \frac{71}{34}, \frac{26}{11}, \frac{235}{89}, \frac{35}{12}$$

---

14. Algoritmos aleatorizados

Ejercicio 1.

Suponga que tiene un algoritmo de Monte Carlo, que siempre se ejecuta en tiempo T pero que responde correctamente solo con probabilidad $\frac{2}{3}$. Nótese que siempre responde "sí" o "no", pero en ambos casos la respuesta está errada 1/3 de las veces. Explique cómo calcular el número de corridas k para que la probabilidad de error sea menos de ϵ.

Pista: Use variables indicadoras X_i con $X_i = 1$ si la corrida i retorna el resultado correcto, $X_i = 0$ en caso contrario; buscamos que el promedio de ellas (mayoría en k corridas) sea al menos 1/2.

Algoritmos aleatorizados - Soluciones

1. Corremos el algoritmo hasta tener una probabilidad menor que ϵ de que la mayoría de el resultado equivocado. Siguiendo la pista, tenemos la variable aleatoria $X = X_1 + \cdots + X_n$, con los $0 \leq X_i \leq 1$ para todo $1 \leq i \leq n$. La mayoría queda expresada por el valor de X/n, si es mayor de $1/2$, gana el "correcto". Es claro que $E[X] = 2n/3$, interesa n tal que:

   $$\Pr[X/n < 1/2] < \epsilon$$

   Es aplicable la cota de Chernoff:

   $$Pr[X < \mathbb{E}[X]/c] < \mathrm{e}^{-\beta(1/c)\mathbb{E}[X]}$$

   Esto da:

   $$\frac{n}{2} = \frac{1}{c} \cdot \frac{2n}{3}$$
   $$c = \frac{4}{3}$$

   de donde:

   $$Pr[X < 1/2] < \mathrm{e}^{-\beta(3/4)\cdot 2n/3}$$

   O sea:

   $$\epsilon \leq \mathrm{e}^{-\beta(3/4)\cdot 2n/3}$$
   $$\ln\epsilon \leq -\beta(3/4) \cdot 2n/3$$
   $$n \geq \frac{3\ln\epsilon}{-2\beta(3/4)}$$
   $$= 39,807 \ln\frac{1}{\epsilon}$$

   Para $\epsilon = 10^{-k}$:

   $$n \geq 39,807 \cdot k \ln 10$$
   $$= 91,660k$$

---

15. Comprensión de la materia

Ejercicio 1.
Defina *concisamente* lo que se entiende por:

1. Dividir y conquistar
2. Programación dinámica
3. Algoritmo voraz
4. Búsqueda heurística

Indique a qué situaciones es aplicable cada una de estas estrategias.

Comprensión de la materia - Soluciones

1. WIP.