# Examenes Finales Algoritmos

Algoritmos (Universidade da Coruña)

**EXAMENES FINALES ALGORITMOS (2012-2019)**

. A partir de la siguiente estructura e datos para la implementación de conjuntos disjuntos:

**tipo**

Elemento = entero;

Conj = entero;

ConjDisj = **vector** [1..N] **de** entero

y del siguiente pseudocódigo para la unión de dos conjuntos:

**procedimiento** Unir (C, raiz1, raiz2) {supone que raíz1 y raíz2 son raíces}

**si** raiz1 < raiz2 **entonces** C[raiz2] := raiz1

**sino** C[raiz1] := raiz2

**fin procedimiento**

a) Escriba el correspondiente pseudocódigo de *Buscar (C, x): Conj* ¸que devuelve el nombre del conjunto (es decir, su representante) de un elemento dado.

b) Razone cuál sería la complejidad computacional de una secuencia de m búsquedas y

n-1 uniones.

SOLUCIÓN

*apartado A*

Acuérdate que hay tres funciones "Unir" y cada una tiene asociado un "Buscar". En este caso, es la función Unir2 y habrá que poner el pseudocódigo de Buscar2.

**función** Buscar (C,x) : Conj

r := x;

**mientras** C[r] <> r **hacer**

r := C[r]

**fin mientras;**

**devolver** r

**fin función**

Si se diera el caso de que nos dan:

**procedimiento** Unir (C, a, b)

i := min (C[a], C[b]);

j := max (C[a], C[b]);

**para** k := 1 **hasta** N **hacer**

**si** C[k] = j **entonces** C[k] := i

**fin para**

**fin procedimiento**

Tendremos que escribirle

**función** Buscar(C, x): Conj

    **devolver** C[x]

**fin función**

Si nos pusieran

**procedimiento** Unir (C,A, raíz1, raíz2) {supone que raíz1 y raíz2 son raíces}

    **si** A[raíz1] = A[raíz2] **entonces**

        A[raíz1] := A[raíz1] + 1;

        C[raíz2] := raíz1

    **sino si** A[raíz1] > A[raíz2] **entonces** C[raíz2] := raíz1

    **sino** C[raíz1] := raíz2

**fin procedimiento**

Tendremos que escribirle

**función** Buscar(C,x) : Conj

    r := x;

    **mientras** C[r] <> r **hacer**

        r := C[r]

    **fin mientras;**

    i := x;

    **mientras** i <> r **hacer**

        j := C[i]; C[i] := r; i := j

    **fin mientras;**

    **devolver** r

**fin función**

*apartado b*

En lo que nos preguntan sería esta la respuesta: O(m*n) ya que por cada una de esas "m" búsquedas, hay "n" uniones.

Si fuera el siguiente Buscar escrito sería: O(m+n^2) ya que por cada una de esas "m" búsquedas habría ese número de búsquedas + "n*n" uniones.

Si fuera el último Buscar escrito sería: O(m*log(n)+n) ya que por cada una de esas "m" búsquedas, hay log(n)+n uniones.

. Diccionario de datos

a) Diseñe, escribiendo en pseudocódigo, los algoritmos InicializarTabal, insertar, buscar y eliminar usando exploración cuadrática de modo que las tres últimas rutinas se ejecutan en u tiempo promedio constante. Use la siguiente declaración de tipos:

**tipo**

ClaseDeEntrada = (legítima, vacía, borrada)

Índice = 0..N-1

Posición = Índice

Entrada = **registro**

Elemento : TipoElemento

Información : ClaseDeEntrada

**fin registro**

TablaDispersión = **vector** [Índice] de Entrada

Si utilizases algún procedimiento auxiliar (distinto de la función hash), refleja también su pseudocódigo.

SOLUCIÓN

Al decirnos exploración cuadrática, lineal o doble sabemos que se trata de dispersión

cerrada (el pseudocódigo es el mismo)

***procedimiento*** *InicializarTabla (D)*

***para*** *i := 0* ***hasta*** *N-1* ***hacer***

*D[i].Información := vacía*

***fin para***

***fin procedimiento***

***función*** *Buscar(Elem, D): Posición*

*i := 0;*

*x = Dispersión(Elem);*

*PosActual = x;*

***mientras*** *D[PosActual].Información <> vacía* ***y***

*D[PosActual].Elemento <> Elem* ***hacer***

*i := i + 1;*

*PosActual := (x + FunResoluciónColisión(x,i)) mod N*

```
        fin mientras
        devolver PosActual
    fin función
    procedimiento Insertar (Elem, D)
        pos = Buscar(Elem, D);
        si D[pos].Información <> legítima
        entonces {Bueno para insertar}
          D[pos].Elemento := Elem;
         D[pos].Información := legítima
    fin procedimiento
    procedimiento Eliminar (Elem, D)
        pos = Buscar(Elem, D);
        si  D[pos].Información := eliminada
    fin procedimiento
```

b) Con la siguiente función *hash*:

hash("a",11) = 8

hash("b",11)=7

hash("c",11)=7

hash("d",11)=7

hash("e",11)=8

hash("f",11)=8

muestre el resultado de insertar las clasves: "a", "b", "c", "d", "e" y "f" (en ese orden) en la siguiente tabla cuando se emplea: 1. exploración lineal, 2. exploración cuadrática, 3.exploración doble usando 5 - (x mod 5) como segunda función (siendo x el resultado de la primera función). Indique asímismo el número total de colisiones que se produce durante las inserciones en cada una de las tres exploraciones.

| Opc. | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | e | f | | | | | | b | a | c | d |
| 2 | c | f | | | d | e | | b | a | | |
| 3 | | e | d | f | | | | b | a | | c |

Para saber las colisiones tenemos que utilizar estas fórmulas dependiendo de que exploración sea:

LINEAL: f(i) = i

CUADRÁTICA: $f(i) = i^2$

DOBLE: $f(i) = i * h_2(x)$

. Escriba el pseudocódigo de un algoritmo voraz que genere un ordenación topológica de los nodos de un grafo dirigido acíclico. Identifique en él los elementos característicos de los algoritmos voraces. Complete su respuesta analizando el algoritmo e identificando su peor caso.

```
función Ordenación topológica (G:grafo) : orden[1..n]
        Grado Entrada [1..n] := Calcular Grado Entrada(G);
        para i := 1 hasta n hacer Número Topológico [i] := 0;
        contador := 1;
        mientras contador <= n hacer
                v := Buscar nodo de rado 0 sin número topológico asignado;
                si v no encontrado entonces
                        devolver error "el grafo tiene un ciclo"
                sino
                        Número Topológico[v] := contador;
                        incrementar contador;
                        para cada w adyacente a v hacer
                                Grado Entrada[w] := Grado Entrada[w] - 1
                fin si
        fin mientras
        devolver Número Topológico
fin función
```

Características de los algoritmos voraces

- Resuelven problemas de optimización: en cada fase, toman un decisión (selección de un candidato), satisfaciendo un óptimo local según la información disponible, esperando así, en conjunto, satisfacer un óptimo global.
- Manejan un conjunto de candidatos C: en cada fase, retiran el candidato seleccionado de C, y si es aceptado se incluye en S, el conjunto donde se construye la solución == candidatos aceptados.
- 4 funciones: S es solución, S es factible, selección para determinar el mejor candidato, objetivo es valorar S.

Análisis: O(n+m) con listas de adyacencia.

Peor caso sería un grafo denso[m->n(n-1)] y visita todas las aristas

Mejor caso: Grafo disperso [m->0, m->n]

. Presente en una tabla los elemenos característicos de los algoritmos voraces que pueda identificar en los algoritmos de Kruskal, Prim y Dijkstra

*algoritmo de Kruskal*

Inicialmente: T vacío. Invariante: (N,T) define un conjunto de componentes conexas. Final: sólo una componente conexta (el a.e.m). Selección: su lema es arista más corta que una ocmponentes conexas distintas. Factible. Estructura de datos: grafo -> aristas ordenadas por peso; árboles -> conjuntos disjuntos (buscar(x), fusionar(A,B)).

*algoritmo de Prim*

Kruskal: bosque que crece hasta convertirse en el a.e.m.

Prim es un único árbol que va creciendo hasta alcanzar todos los nodos.

Inicialización: B={nodo arbitrario} = {1}, T vacío.

Selección: arista más corta que parte de B.

Invariante: T define en todo momento un a.e.m del subgrafo (B,A)

Final: B = N

*algoritmo de Dijkstra*

Teorema: dijkstra encuentra los caminos mínimos desde el origen hacia los demás nodos del grafo. Demostración es por inducción.

$T(n) = \Theta(n^2)$

Mejora: si el grafo es disperso ($m << n^2$), utilizar listas de adyacencia -> ahorro en para anidado: recorrer la lista y no fiila o columna de L.

. Compare la ordenación por fusión con la ordenación rápida desde el punto de diseño del algoritmo (técnica de diseño, fases de la técnica utilizada, estrategias para la mejora del tiempo de ejecución...) así como de su complejidad (relaciones de recurrencia según cada caso, que se justificarán y se resolverán, teorema necesario para resolverlas...).

*fusión*

Mergesort, o bien ordenación por intercalación. Utiliza un algoritmo de Fusión de un vector cuyas mitades están ordenadas para obtener un vector ordenado. El procedimiento Fusión es lineal (n comparaciones). Ordena con el algoritmo de divide y vencerás:

- Divide el problema en 2 mitades, que se resuelven recursivamente; Fusiona las mitades ordenadas en un vector ordenado.

Como mejora: ordenación por inserción para vectores pequeños: n < umbral, que se determina empíricamente.

Análisis del algoritmo: T(n) = Θ(nlogn). Podría mejorarse la complejidad espacial ( = 2n; vector auxiliar) -> el algoritmo adecuado es el quicksort.

*ordenación rápida*

Paradigma de Divide y Vencerás, Con respecto a Fusión:

- más trabajo para construir las subinstancias, pero trabajo nulo para combinar las soluciones.

Se selecciona el pivote con el objetivo de obtener una partición lo más balanceada posible. Usar el primer valor del vector si la entrada es aleatoria está bien pero es una elección muy desafortunada con entradas ordeandas o parcialmente ordenadas -> $O(n^2)$ para no hacer nada.

Usa un valor elegido al azar (pivote aleatorio): es más seguro, evita el peor caso detectado antes, pero depende del generador de números aleatorios.

Análisis del algoritmo: es un pivote aleatorio y sin umbral

- Peor caso: $T(n) = O(n^2)$
- Mejor caso: T(n) = O(nlogn)

. Considerando un sistema monetario M = {v1,v2,...,vm}, se dispone de una función Monedas que utiliza la técnica de Programación Dinámica para encontrar el número mínimo de monedas para pagar la cantidad n, calculando para ello una tabla T con todos los resultados intermedios (solución óptima para pagar la cantidad j con las monedas v1,..,vi). El resultado encontrado puede corresponder a una o varias configuraciones (conjuntos de monedas que suman n).

Se plantea el diseño de una función Composición que, a partir de la tabla T ya construida por la función Monedas, devuelva una configuración posible de la solución, especificando el conjunto de monedas que la componen:

a) Proponga un ejemplo del programa, presentando su tabla T y sus posibles soluciones.

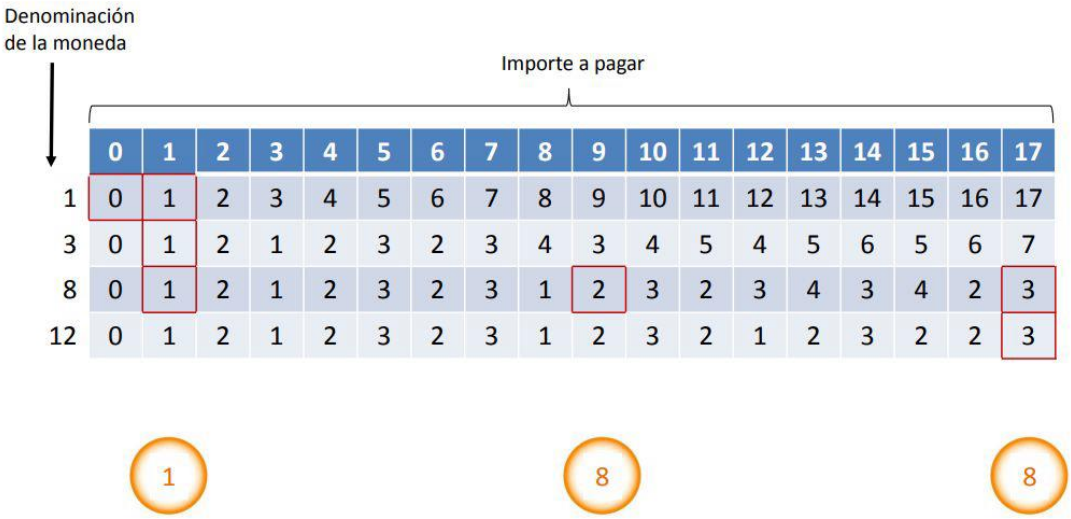
b) Proponga un tipo de datos adecuado para la salida de la función Composición, que ilustrará con el ejemplo anterior.

c) Proponga un pseudocódigo de la función composición.

d) Determine su complejidad.

Solucion

a) Constuir la tabla con la que podría determinarse en programación dinámica la manera óptima de pagar una cantidad de 17 unidades de valor con un mínimo de monedas, sabieno que el sistema monetario considerado está constituido por monedas de 1,3,8 y 12 unidades de valor. Indicar la solución al problema dibujando una traza en la tabla anterior para justificar cómo se obtiene.




| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| 3 | 0 | 1 | 2 | 1 | 2 | 3 | 2 | 3 | 4 | 3 | 4 | 5 | 4 | 5 | 6 | 5 | 6 | 7 |
| 8 | 0 | 1 | 2 | 1 | 2 | 3 | 2 | 3 | 1 | 2 | 3 | 2 | 3 | 4 | 3 | 4 | 2 | 3 |
| 12 | 0 | 1 | 2 | 1 | 2 | 3 | 2 | 3 | 1 | 2 | 3 | 2 | 1 | 2 | 3 | 2 | 2 | 3 |

b) y c

c)

```
función monedas (n): número de monedas
    const v[1..m]=[1,4,6]          {denominaciones de las monedas}
    {se construye una tabla c[1..m, 0..n]}
    para i := 1 hasta m hacer c [i,0] := 0;
    para i := 1 hasta m hacer
        para j := 1 hasta n hacer
            si i = 1 y j < v[i] entonces c[1,j] := infinito
            sino si i = 1 entonces c[1,j] := 1 + c[1, j-v[1] ]
            sino si j < v[i] entonces c[i,j] := c[i-1,j]
            sino c[i,j] := min ( c[i-1, j], 1 + c[i, j-v[i] ] )
            fin si
        fin para
    fin para;
    devolver c[m,n]
fin función
```

```
const M = [1, 2, 5, 10, 20, 50, 100, 200] {denominaciones}
función Devolver cambio (n) : conjunto de monedas
    S := conjunto vacio;              {la solución se construye en S}
    ss := 0;                              {suma de las monedas de S}
    mientras ss <> n hacer           {bucle voraz}
        x := mayor elemento de M : ss + x <= n;
        si no existe tal elemento entonces devolver "no hay soluciones"
        S := S U {una moneda de valor x};
        ss := ss + x;
    fin mientras;
    devolver S
fin función
```

d) Análisis: T(n) = Θ(mn)

Problema: conjunto de monedas. Θ(m+c[m,n])

PODRIAN PREGUNTAR PARA LA MOCHILA

```
función Mochila 1 ( w[1..n], v[1..n], W): objetos[1..n]
    para i := 1 hasta n hacer
        x[i] := 0;            {la solución se construye en x}
    peso := 0;
    {bucle voraz:}
    mientras peso < W hacer
        i := el mejor objeto restante; {1}
        si peso+w[i] <= W entonces
            x[i] := 1;
            peso := peso+w[i]
        sino
            x[i] := (W-peso)/w[i];
            peso := W
        fin si
    fin mientras;
    devolver x
fin función
```

Análisis: inicialización Θ(n).

Peor caso: O(nlogn)

Mejor caso: T(n)

. Escriba el pseudocódigo del algoritmo de ordenación rápida con pivote aleatorio y justifique el análisis de su complejidad.

```
procedimiento OrdenarAux (V[izq..der])
        si izq+UMBRAL <= der entonces
          x := {num. aleatorio en el rango [izq..der]}
          pivote := V[x];
          intercambiar (V[izq], V[x]);
          i := izq + 1;
          j := der;
          mientras i <= j hacer
                mientras i <= der y V[i] < pivote hacer
                        i := i + 1
                fin mientras;
                mientras V[j] > pivote hacer
                        j := j - 1
                fin mientras;
                si i <= j entonces
                        intercambiar (V[i], V[j]);
                        i := i + 1;
                        j := j - 1;
                fin si
        fin mientras;
        intercambiar (V[izq], V[j]);OrdenarAux(V[izq..j-1]);OrdenarAux(V[j+1..der];
        fin si
```

```
fin procedimiento

procedimiento Ordenación Rápida (V[1..n])
        OrdenarAux(V[1..n]);
        si (UMBRAL > 1) entonces
                Ordenación por Inserción (V[1..n])
        fin si
fin procedimiento
```

.Dado el siguiente algoritmo:

*función Kruskal (G = [N,A]): tipo_salida(1)*

*Organizar los candidatos a partir de A: (2)*

*n := [N]; T := conjunto vacío; peso := 0;*

*inicializar n conjuntos, cada uno con un nodo de N;*

*repetir*

*Seleccionar y extraer un candidato a; (3)*

*ConjuntoU := Buscar(a.nodo1); ConjuntoV := Buscar(a.nodo2)*

*si ConjuntoU <> ConjuntoV entonces*

*Fusionar (ConjuntoU, ConjuntoV);*

*T := T U [a];*

*peso := peso + a.peso;*

*fin si*

*hasta [T] = n-1;*

*devolver <T, peso>*

*fin función*

a) Precise el tipo de datos de la salida de la función ([1]).
b) Reescriba las instrucciones {2} {3} introduciendo el uso de un montículo, decisión que justificará en su respuesta.
c) Enuncia de forma precisa el problema que resuelve este algoritmo.
d) Concrete los elementos característicos de la técnica voraz que correspondan a este ejemplo.
e) Análisis detallado de la complejidad.

Solución

a) Tipo Árbol
b) nose

c) encontrar un árbol recubridor mínimo en un grafo conexo y ponderado. Busca un subconjunto de aristar que, formando un árbol, incluyen todos los vértices y donde el valor de la suma de todas las aristas del árbol es mínimo.

d) Inicialmente: T vacío. Invariante: (N,T) define un conjunto de componentes conexas. Final: sólo una componente conexta (el a.e.m). Selección: su lema es arista más corta que una ocmponentes conexas distintas. Factible. Estructura de datos: grafo -> aristas ordenadas por peso; árboles -> conjuntos disjuntos (buscar(x), fusionar(A,B)).

e) Análisis: |N| = n ^ |A| = m

ordenar A: O(mlogm) = O(mlogn): n-1 <= m <= n(n-1)/2

inicializar n conjuntos disjuntos: Θ(n)

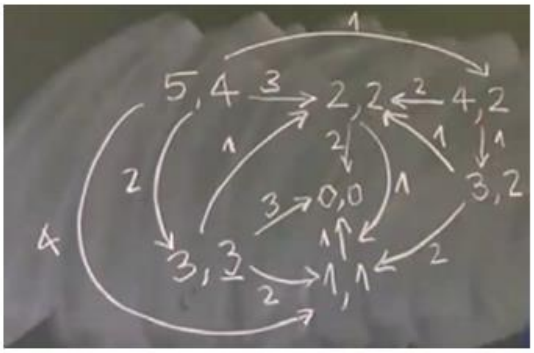
2m buscar (peor caso) y n-1 fusionar (siempre): 2(mα(2m,n)) = O (mlogn)

resto: O(m) peor caso

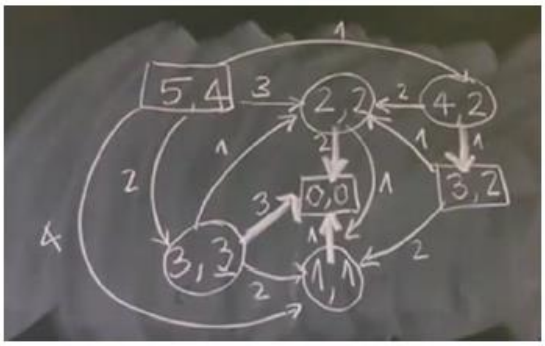
T(n) = O(mlogn)

. Represente mediante un grafo decorado todas las situaciones de juego que podrían alcanzarse a partir de un montón de 5 palillos papra la variante del juego de Nim vista en clase.

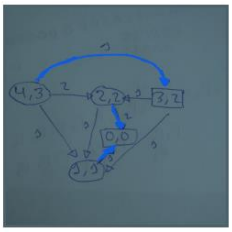
*5 palillos, 4 jugadas posibles*




Grafo sin decorar




Grafo decorado

Con 4 palillos 

. Cálculo de un coeficiente binomial C(n,k) utilizando la técnica de Programación Dinámica de forma que las necesidades de memoria sean mínimas:

a) Proponga un ejemplo concreto, partiendo del triángulo de Pascal, para explicar el algoritmo.

b) Determine su complejidad temporal y espacial.

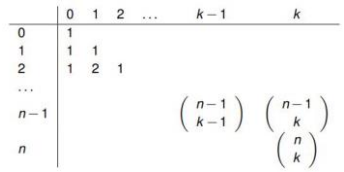
función C{n,k}:valor

si k = 0 ó k = n entonces devolver 1

sino devolver c(n-1,k-1) + C(n-1,k)

fin si

fin función

a) 

| | 0 | 1 | 2 | ... | $k-1$ | $k$ |
|---|---|---|---|---|---|---|
| 0 | 1 | | | | | |
| 1 | 1 | 1 | | | | |
| 2 | 1 | 2 | 1 | | | |
| ... | | | | | | |
| $n-1$ | | | | | $\binom{n-1}{k-1}$ | $\binom{n-1}{k}$ |
| $n$ | | | | | | $\binom{n}{k}$ |

b) T(n) = Θ(nk) y la complejidad espacial también.

. A partir de lasiguiente estructura d edatos para la implementación de un árbol binario de búsqueda:

tipo

Parbol = ^Nodo

Nodo = registro

Elem : TipoElemento

Izq, Der: Parbol

fin registro

ABB = Parbol;

a) Diseñe, escribiendo su pseudocódigo, un recorrido del árbol en orden de nivle (es decir, todos los nodos de profundidad p se procesan antes que cualquier nodo con profundidad p = 1) de modo que esta rutina se ejecute en tiempo lineal. El procesamiento de cada nodo consiste en su visualización. Refleje en el diseño todos los procedmientos y estructuras de datos necesarias.

b) Determine, justificando su respuesta sobre el pseudcódigo, la O de cada operación.

SOLUCION

a) tipo Cola = registro

Cabeza_de_cola, Final_de_cola : 1..Tamaño_maximo_de_cola

Tamaño de cola: 0..Tamaño_maximo_de_cola

Vector_de_cola : vector [1..Tamaño_maximo_de_cola]

de TipoElemento

fin registro

procedimiento Crear_cola {C} O(1)

C.TamañoDeCola := 0;

C.CabezadeCola := 1;

C.Final de cola := Tamño maximo de cola

fin procedimiento

función colaVacia (C): test

devolver C.tamañodecola = 0

fin función

Procedimiento incrementar (x)

si x = tamañoMaximoDeCola entonces x := 1

sino x := x+1

fin procedimiento

procedimiento InsertarEnCola (x,C)

si C.TamañoDeCola = TamañoMaximoDeCola entonces

erroR Cola Llena

sino

C.tamañoDeCola := C.tamaño de cola + 1;

incrementar(C.finaldecola);

C.vector de cola (C.finaldeCola) := x;

fin procedimiento

. Como quedaría el siugiente árbol, que representa un conjunto, tras buscar el elemento 8 usando la técnica de compresión de caminos?

TIPOS DE TÉCNICAS

- unión por rango : 2 árboles hacemos que la raíz q va a apuntar a la otra sea la del árbol con menos nodos. Mantenemos el rango de los árboles.
- compresión de caminos:operación encontrar. Hace que los nodos recorridos en el camino de búsqueda pasen a apuntar directamente a la raíz. No se modifica la info del rango.