Busqueda Binaria y Ternaria

Matias Tealdi[1]

[1]Facultad de Matemática, Astronomía y Física
Universidad Nacional de Córdoba

Training Camp 2012

Contenidos

Contenidos

Qué es y para qué sirve? Ejemplo.

- Dada una lista ordenada de primos menores a 55, decidir si un número r menor a 55 es primo.

Qué es y para qué sirve? Ejemplo.

- Dada una lista ordenada de primos menores a 55, decidir si un número r menor a 55 es primo.
- arr = $\{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53\}$
  assert(arr.size() == 16)

Qué es y para qué sirve? Ejemplo.

- Dada una lista ordenada de primos menores a 55, decidir si un número r menor a 55 es primo.
- arr = $\{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53\}$
  assert(arr.size() == 16)

  Búsqueda Lineal.

  Recorremos elemento por elemento y chequeamos si r esta en la lista.

  Complejidad : O(n)

Qué es y para qué sirve? Ejemplo.

- Dada una lista ordenada de primos menores a 55, decidir si un número r menor a 55 es primo.
- arr = $\{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53\}$
  assert(arr.size() == 16)

Búsqueda Lineal.

Recorremos elemento por elemento y chequeamos si r esta en la lista.

Complejidad : O(n)

Búsqueda Binaria.

Tomamos el intervalo [a..b] nos fijamos si el elemento $[(a+b)/2]$ es menor igual a r. Si lo es, solo me trengo que fijar si r esta en $[a, (a+b)/2]$, sino r tiene q estar en $[(a+b)/2, b]$. Aplico recursivamente la misma idea.

Complejidad : O(log n)

Contenidos

Implementación de Búsqueda Binaria Discreta

```
1  int bsearch(const vector<int> &arr, int r) {
2      int left = 0, right = n;
3      while (right-left>1) {
4          int mid = (left+right)/2;
5          if (arr[mid] <= r) {
6              left = mid;
7          } else {
8              right = mid;
9          }
10     }
11     return left;
12 }
13
14 bool es_primo (const vector<int> &arr, int r) {
15   return arr[bsearch(arr, r)] == r;
16 }
```

Contenidos

Generalización

- Dado un intervalo [a, b], y una propiedad P sobre el intervalo tal que $\forall x, y \ en \ [a, b], \ si \ x < y \ entonces \ P(y) \implies P(x)$, BS nos encuentra el elemento más grande que cumple P o el elemento más chico que no cumple P (el siguiente del más grande que cumple P).

Generalización

- Dado un intervalo [a, b], y una propiedad P sobre el intervalo tal que $\forall x, y \ en \ [a, b], \ si \ x < y \ entonces \ P(y) \implies P(x)$, BS nos encuentra el elemento más grande que cumple P o el elemento más chico que no cumple P (el siguiente del más grande que cumple P).
- Demostración Informal utilizando inducción.
  Supongamos que tenemos un intervalo $[a, b]$ tal que cumple las condiciones dadas arriba. Además supongamos que hay al menos un elemento en $[a, b]$ que cumple P.

Generalización

- Dado un intervalo [a, b], y una propiedad P sobre el intervalo tal que $\forall x, y \; en \; [a, b], \; si \; x < y \; entonces \; P(y) \implies P(x)$, BS nos encuentra el elemento más grande que cumple P o el elemento más chico que no cumple P (el siguiente del más grande que cumple P).
- Demostración Informal utilizando inducción.
  Supongamos que tenemos un intervalo $[a, b]$ tal que cumple las condiciones dadas arriba. Además supongamos que hay al menos un elemento en $[a, b]$ que cumple P.

  Si [a,b] tiene un elemnto, entonces ya tengo el elemento más grande que cumple P.

Generalización

- Dado un intervalo [a, b], y una propiedad P sobre el intervalo tal que $\forall x, y \; en \; [a, b], \; si \; x < y \; entonces \; P(y) \implies P(x)$, BS nos encuentra el elemento más grande que cumple P o el elemento más chico que no cumple P (el siguiente del más grande que cumple P).
- Demostración Informal utilizando inducción.
  Supongamos que tenemos un intervalo $[a, b]$ tal que cumple las condiciones dadas arriba. Además supongamos que hay al menos un elemento en $[a, b]$ que cumple P.

  Si [a,b] tiene un elemnto, entonces ya tengo el elemento más grande que cumple P.

  Si [a,b] tiene tamaño n>1. Sea $mid = (a + b)/2$, si $P(mid)$ vale, entoces como P es monotona estoy seguro que el elemento más grande que vale P esta en [mid, b] de lo contrario estoy seguro de que esta en [a, mid].

Contenidos

Análisis de Complejidad

Sea n = h-l el tamaño del intervalo. complejidad de hacer búsqueda binaria en $[a, b]$ es $log2(n)$.

Si corremos el algoritmo veremos que en cada paso el tamaño del intervalo se divide en 2.
n , n/2 , n/4, n/8 ... hasta llegar a 1

La cantidad de veces que se ejecuta el ciclo es $log2(n)$

Contenidos

Implementación de Búsqueda Binaria Discreta

```
1  int bsearch( ... ) {
2      int left = 0, right = n;
3      while (right-left>1) {
4          int mid = (right+left)/2;
5          if (P(mid)) {
6              left = mid;
7          } else {
8              right = mid;
9          }
10     }
11     return left;
12 }
```

Implementación recursiva de Búsqueda Binaria Discreta

```
1 int bsearch_rec(int left, int right, ...) {
2     if (right-left <= 1) return left;
3     int mid = (right+left);
4     if (P(mid))
5         return bsearch_rec(mid, right, ...);
6     else
7         return bsearch_rec(left, mid, ...);
8 }
```

Ejemplos

- n input, decir si n es primo menor igual a 55.
  BS usando $P(x) : x <= n$
  retornar si $arr[bsearch(L, n)] == n$

Ejemplos

- n input, decir si n es primo menor igual a 55.
  BS usando $P(x) : x <= n$
  retornar si $arr[bsearch(L, n)] == n$
- *lowerBound*(y): dado arreglo ordenado de n elementos, devolver la posición del primer elemento no menor a y (mayor igual).
  $P(x) : x < y$ — (valor de r en la BS)
  retornar $arr[bsearch(L, n) + 1]$ si $bsearch(L, n) + 1 < n$, otro caso no existe.

Ejemplos

- n input, decir si n es primo menor igual a 55.
  BS usando $P(x) : x <= n$
  retornar si $arr[bsearch(L, n)] == n$
- *lowerBound*(y): dado arreglo ordenado de n elementos, devolver la posición del primer elemento no menor a y (mayor igual).
  $P(x) : x < y$ — (valor de r en la BS)
  retornar $arr[bsearch(L, n) + 1]$ si $bsearch(L, n) + 1 < n$, otro caso no existe.
- *upperBound*(y): dado arreglo ordenado de n elementos, devuelve la posición del primer elemento mayor a y.
  $P(x) : x <= y$
  retornar $arr[bsearch(L, n) + 1]$ si $bsearch(L, n) + 1 < n$, otro caso no existe.

Ejemplos

- Dado n cajas tal que en cada caja entran c[i] elementos. Dados r elementos, distribuir los r elementos tal que la cantidad de elementos en la caja que más tiene sea mínimo. Devolver este valor.

Ejemplos

- Dado n cajas tal que en cada caja entran c[i] elementos. Dados r elementos, distribuir los r elementos tal que la cantidad de elementos en la caja que más tiene sea mínimo. Devolver este valor.

  $P(x)$ = puedo poner los r elementos de tal modo que cada caja no tiene más de x elementos.

Implementación

```
1
2  assert(r > 0 && sum_i(c[i]) >= r);
3  int left = 0, right = r;
4  while(left<right-1) {
5      mid = (left+right)/2;
6      if( P(mid, r) ) right = mid;
7      else left = mid;
8  }
9  return right;
10
11 bool P(int x, int r) {
12     int cnt = 0;
13     for(int i=0 ; i<n ; i++) cnt += min(x, c[i]);
14     return cnt >= r;
15 }
```

- Complejidad del algoritmo : O(n log(n)), ya que en cada ciclo de la BS hago n pasos.

Contenidos

Generalización

- Dadas una función f continua creciente o decreciente, en un intervalo $[a, b]$
  Es decir,
  creciente, si para todo $x \leq y \implies f(x) \leq f(y)$ o,
  decreciente, si para todo $x \leq y \implies f(y) \leq f(x)$

  Dado y, supongamos que existe x en $[a, b]$ tal que $f(x) = y$,
  BS sirve para encontrar el valor de x tal que $f(x) = y$

Generalización

- Dadas una función f continua creciente o decreciente, en un intervalo $[a, b]$
  Es decir,
  creciente, si para todo $x \leq y \implies f(x) \leq f(y)$ o,
  decreciente, si para todo $x \leq y \implies f(y) \leq f(x)$

  Dado y, supongamos que existe x en $[a, b]$ tal que $f(x) = y$,
  BS sirve para encontrar el valor de x tal que $f(x) = y$
- Puede haber error de precisión de punto flotante.

Contenidos

Código

```
1 double left = a, right = b;
2 while(right-left > EPS) {
3     double mid = (left+right)/2;
4     if( f(mid) < y) left = mid;
5     else right = mid;
6 }
7 return (left+right)/2;
```

Código

```
1 double left = a, right = b;
2 /*x entre 30 y 100, aprox log(h-l) + 30 si el EPS es 10E-9*/
3 for(t=0 ; t<x ; t++) {
4     double mid = (left+right)/2;
5     if( f(mid) < y) left = mid;
6     else right = mid;
7 }
8 return (left+right)/2;
```

Contenidos

Algunas custiones sobre números reales

Como los números reales son infinitos, esta claro que la computadora no los puede representar a todos. Por ello, las operaciones con punto flotante pueden tener errores de precisión.

Para solucionar este inconveniente, muchas veces se utiliza valores constantes muy pequeños a los que llamamos epsilon. Este valor nos permite darle un margen al error.

Por ejemplo : Supongamos x, y dos números double y EPS = 10E-9
x < y, deberiamos hacer x < y-EPS
x == y, deberiamos hacer fabs(x-y) < EPS
El resto de las operaciones las podemos armar en función de estas dos.

Problemas que puede haber con manejo de números reales.

- Por ejemplo, en la BS, es posible que ((a+b)/2 == a) e incluso que (a/2+b/2) < a. En este caso, si utilizamos un EPS muy pequeño, puede ocurrir que nunca se haga falsa la codición del while.
- Que $(a1 + a2) + a3 \; != \; a1 + (a2 + a3)$
- Que $(a1 + a2)/2 \; != \; a1/2 + a2/2$

Ejemplos

- Encontrar la raíz cuadrada de r.
  Si bien la búsqueda binaria no es el método más preciso de encontrar la raíz cuadrada de un número, sirve de ejemplo en este caso.

  Sabemos que la función de raíz cuadrada es una función monotona creciente, por lo tanto solo tenemos que aplicar el algoritmo de BS con el predicado $P(x) = x * x < r$.

Contenidos

Concepto

- Es una técnica para encontrar el máximo o mínimo en un intervalo de un función que es estrictamente creciente en un intervalo y luego estrictamente decreciente o viceversa.
- Se puede aplicar en funciones continuas o discretas.
- En funciones discretas es importante verificar que la función cumpla que sea estrictamente creciente y luego estrictamente decreciente. (no dos valores consecutivos iguales).

Contenidos

Generalización

- Asumamos que estamos buscando el máximo de una función f, y sabemos que el máximo esta en el intervalo $[A, B]$. Para poder aplicar TS, debe existir algún valor x tal que

$$\forall a, b \; con \; A <= a < b <= x, \; tenemos \; f(a) < f(b)$$

$$\forall a, b \; con \; x <= a < b <= B, \; tenemos \; f(a) > f(b)$$

Contenidos

Algoritmo

```
1  double TS(double A, double B) {
2      double left = A, right = B;
3      while(abs(right-left) < EPS) {
4          double lt = (2.*left + right) / 3;
5          double rt = (left + 2.*right) / 3;
6          if(f(lt) < f(rt)) left = lt;
7          else right = rt;
8      }
9      return (left+right)/2;
10 }
```

- Tiene el mismo problema de precisión que la BS continua, se soluciona del mismo modo.

Contenidos

Complejidad

En cada paso del ciclo dividimos el intervalo en 3 y achicamos el nuevo intervalo a 2/3 del intervalo inicial.
Por lo tanto generamos una secuencia de intervalos de la forma

$n, (2/3) * n, (4/9) * n, ..(2^j/3^j) * n$

Buscamos que el $(2/3)^j$ sea tan chico como EPS/n

Por lo tanto la coplejidad de BT es $O(log_{2/3}(n)) = O(log(n))$

Ejemplos

Supongamos que granjero John viven en una casa en la posición (xh, yh) con yh>0, y quiere regar su árbol que está en la posición (xa,ya), con ya>0. Para poder regarlo, John camina hasta el río que se encuentra en el eje x, toma agua y se dirige hasta el árbol.

Cual es la miníma distancia que tiene que recorrer granjero John para regar su árbol.

Algoritmo

```
1  #define EPS 1E-9
2  int main () {
3      double xh, yh, xa, ya;
4      cin >> xh >> yh;
5      cin >> xa >> ya;
6      if(xh < xa) swap(xh, xa), swap(yh, ya);
7
8      double left = xh, right = xa;
9      while(fabs(right-left) > EPS) {
10         double pl = (2*left + right) / 3.;
11         double pr = (left + 2*right) / 3.;
12         if(f(pl, xh, yh, xa, ya) < f(pr, xh, yh, xa, ya)-EPS)
13             right = pr;
14         else left = pl;
15     }
16     cout << f((left+right)/2., xh, yh, xa, ya) << endl;
17     cout << sqrt( sqr(xh-xa) + sqr(yh+ya) ) << endl;
18     return 0;
19 }
```

Algoritmo

```
1 #define sqr(A) ((A)*(A))
2 double f(double x, double xh, double yh, double xa, double ya) {
3     return sqrt(sqr(xh-x) + sqr(yh)) + sqrt(sqr(xa-x) + sqr(ya));
4 }
```