Semana 9

Pedro O. Pérez M., PhD.

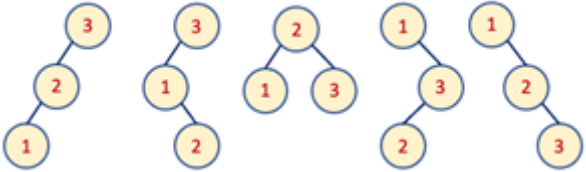
Análisis y diseño de algoritmos avanzados
Tecnológico de Monterrey

pperezm@tec.mx

10-2021

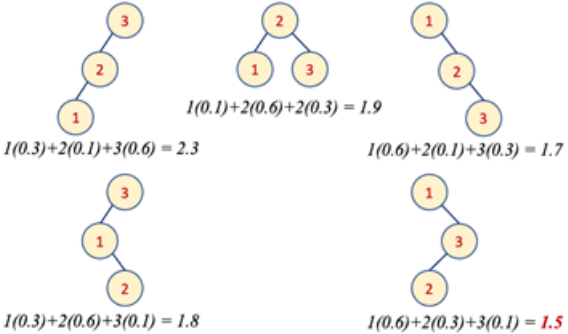
Definición

Como recordarán, un árbol binario de búsqueda (Binary Search Tree, BST), es un árbol binario que almacena valores comparables únicos. Los valores son almacenados de tal forma que para cada nodo, los valores de los nodos de su subárbol izquierdo son menores y todos los valores del subárbol derecho son mayores a él. La altura del árbol es la cantidad máxima de comparaciones que se tienen que hacer como máximo para buscar un dato.



El problema del BST óptimo es dadas n llaves, cada llave con su probabilidad de ser buscadas, encontrar la estructura del árbol que minimice el tiempo promedio de búsqueda. El tiempo promedio de búsqueda de un BST se calcula sumado la cantidad de comparaciones que se requieren para llegar a cada nodo multiplicado por su probabilidad.

Por ejemplo, si se tuvieran 3 datos con las siguientes probabilidades, el 1 con probabilidad de 0.6, el 2 con probabilidad de 0.1 y el 3 con probabilidad de 0.3.



Algoritmo de Gilbert & Moore

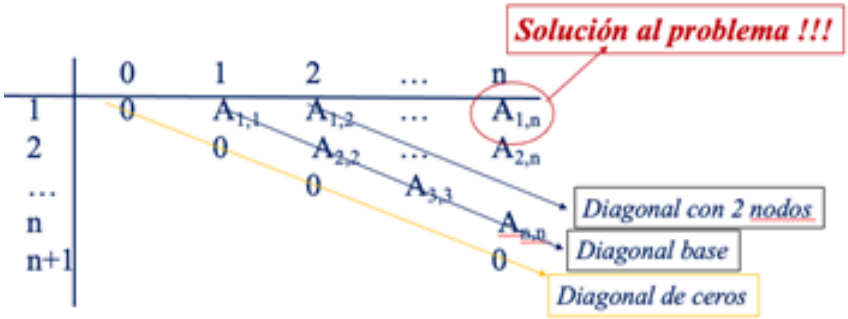
El algortimo propuesto por Gilbert & Moore en 1959, utiliza la técnica de programación dinámica para resolver este problema. El algoritmo utiliza una matriz A de dimensiones $(n+1)x(n+1)$ donde n es la cantidad de datos a almacenar, teniendo los reglones 1 hasta $n+1$, y las columnas 0 hasta n. En cade celda, $A[i][j]$, se almacena el mínimo promedio de búsqueda de los nodos i hasta j. Lo que se desea es encontrar el tiempo mínimo promedio de búsquedas de todo el árbol, el cual quedará almacenado en la celda $A[i][n]$.

1. Al inicio, se coloca en la celda $A[i][i]$ la probabilidad, p_i, en la búsqueda del i-ésimo elemento. Este se considera el caso base, un árbol de un solo nodo.
2. Partiendo de esto, hay que encontrar la solución general. Asumiendo que k es el nodo raíz del árbol óptimo desde i hasta j, esto es, k tiene que ser un valor entre i y j inclusive. Los subárboles del nodo raíz k serán BST óptimos que tienen un tiempo mínimo promedio previamente calculado y almacenados en:
   - El subárbol izquierdo tiene las llaves desde i hasta $k-1$, y su valor del tiempo mínimo promedio estará almacenado en $A[i][k-1]$.
   - El subárbol derecho tiene las llaves desde $k+1$ hasta j, y su valor del tiempo mínimo promedios estará almacenado en $A[k+1][j]$.

Ahora bien, el cálculo del subárbol desde i hasta j, considerando a k como la raíz, es:

- La probabilidad de la raíz, p_k.
- La probabilidad de los nodos desde i hasta $k-1$, para bajarlos de 1 nivel, ya que ahora serán el subárbol izquierdo de k.
- La probabilidad de los nodos desde $k+1$ hasta j, para bajarlos 1 nivel, ya que ahora serán el subárbol derecho de k.

$$A[i][j] = \min_{i \leq k \leq j} (A[i][k-1] + A[k+1][j]) + \sum_{a=i}^{j} p_a$$


Solución al problema !!!
0 1 2 … n
1 0 $A_{1,1}$ $A_{1,2}$ … $A_{1,n}$
2 0 $A_{2,2}$ … $A_{2,n}$
… 0 $A_{3,3}$
n $A_{n,n}$
n+1 0
Diagonal con 2 nodos
Diagonal base
Diagonal de ceros

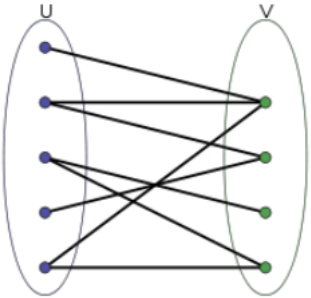
Procedure 1 GILBERT_MOORE

Input: N : *Integer*, Pi : *Vector*

Generate a $(N+1)x(N+1)$ matrix called A
for $i \leftarrow 1$ **to** N **do**
$A[i][i-1] \leftarrow 0$, $A[i][i] \leftarrow P[i]$
end for
for $diag \leftarrow 1$ **to** $N-1$ **do**
for $i \leftarrow |$ **to** $N - diag$ **do**
$j \leftarrow i + diag$, $minimum \leftarrow \infty$
for $k \leftarrow i$ **to** j **do**
$minimum \leftarrow \min(A[i][k-1] + A[k+1][j], minimum)$
end for
$minimum \leftarrow minimum + \sum_{a=i}^{j} p_a$
end for
end for
return $A[1][N]$

Definición

Un grafo bipartita (o bigrafo) $G = (V, E)$ es un grafo cuyos vértices pueden ser divididos en dos conjuntos disjuntos R y S tal que cada arco conecta a un vértice en R con un vértice en S.



Procedure 2 BIGRAPH

Input: G : *Graph*

Q : *Queue*, *Color* : *Array*, *isBipartite* : *boolean*

$INIT(Color, -1)$

$isBipartite \leftarrow true$

$vertex \leftarrow$ some vertex in G

$Color[vertex] \leftarrow 1$

$Q.enqueue(vertex)$

while Q is not empty **do**

NEXT SLIDE

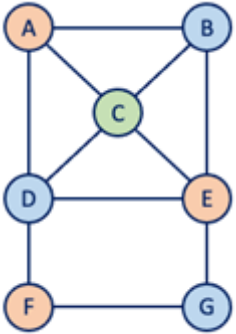
end while

return *isBipartite*

```
u ← Q.dequeue()
for each (u, v) incident in u do
  if Color[v] = −1 then
    Color[v] ← 1 − Color[u]
    Q.enqueue(v)
  else
    if Color[v] = Color[u] then
      isBipartite ← false
    end if
  end if
end for
```

Definición

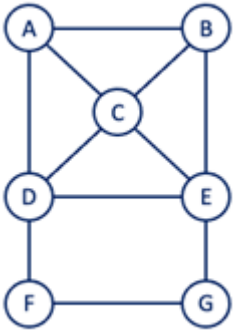
El coloreo de grafos es un problema en donde se va coloreando (etiquetando) a cada vértice de un grafo no dirigido, con la restricción de los vértices adyacentes no tengan el mismo color o etiqueta. Esto es que cada vértice tenga un color diferente a los de sus vecinos. Esta idea puede servir para solucionar problemas de selección para grupos, equipos, etc. donde los vértices son las personas u objetos y las restricciones se vean como arcos que los une.



Algoritmo de Welsh Powell

El algoritmo de coloreo de grafos propuesta por Welsh & Powell en 1967, es un algoritmo ávido que consiste en obtener el grado de cada vértice (cantidad de arcos que llegan a él) y ordenar en forma descendente. Se empieza colorear el de mayor grado, y se va verificando en forma ordenada si se puede colorear con el mismo color a algún otro vértice. Una vez terminando esta iteración, se continua con el siguiente vértice de mayor grado, utlizando la misma estrategia, hasta llegar al último.

1. Grado 4: C
2. Grado 3: A B C E
3. Grado 2: F G



Procedure 3 WELSH_POWELL

Input: G : *Graph*
  Color : *Array*, *colorAssigned* : *Integer*
  Sort vertices in G descending by degree, *sortedV*
  colorAssigned $\leftarrow 0$
  for each v in *sortedV* **do**
    if has v NOT been colored? **then**
      colorAssigned $\leftarrow 1$, *Color*$[v]$ $\leftarrow$ *colorAssigned*
      for each remaining vertex, u, in sortedV **do**
        if u has not been colored and is not adjacent to a node with *assignedColor* **then**
          Color$[u]$ $\leftarrow$ *colorAssigned*
        end if
      end for
    end if
  end for
  return *Color*