# Esercizi su BST

IASD a.a. 2015-16
A. De Bonis

# Esercizio 1

- Scrivere un algoritmo che prende in input un albero binario e restituisce true se e solo se l'albero e` un albero binario di ricerca.

- L'algoritmo deve utilizzare solo una quantita` costante di spazio aggiuntivo e avere tempo O(n).

# Soluzione 1 esercizio 1

- Un albero binario e` un BST se e solo se per ogni nodo la chiave al suo interno e` > di quelle dei nodi nel sottoalbero sinistro e < di quelle dei nodi nel sottoalbero destro.

- Assumiamo per semplicita` che tutte le chiavi siano positive
- Scrivo una procedura che se invocata su un nodo u restituisce
  - una coppia di due interi che rappresentato la chiave minima e la chiave massima del sottoalbero con radice u, se u e` radice di uno BST
  - la coppia (-1,-1), se u e` radice di un albero che non e` uno BST

# Soluzione 1 esercizio 1

```
1. isBST (Albero):
2.         if(Albero.dimensione==0) return true;
3.          B=isBST_Aux(Albero.radice); //B array di due elementi
4.         return(B[0]!=-1);
5. }
```

# Soluzione 1 esercizio 1

A, AS e AD sono array di due numeri

```
1.  isBST_Aux(Nodo):
2.  If(Nodo.sx!=null){    //caso Nodo ha il figlio sinistro
3.        AS = isBST_Aux(Nodo.sx); //min e max sottoalbero sinistro
4.         If(AS[0]==-1){return AS;}//non fa ricorsione sul figlio destro
5.          }
6.   Else {AS[0]=Nodo.dato.chiave;AS[1]=Nodo.dato.chiave;}//se non ha figlio sinistro
7.   If(Nodo.dx!=null){//Nodo ha il figlio destro
8.           AD= isBSTAux(Nodo.dx);
9.            if(AD[0]==-1){return AD;}
10.  }Else {AD[0]=Nodo.dato.chiave; AD[1]=Nodo.dato.chiave;}//se Nodo non ha il figlio destro
11.   A[0]=AS[0];A[1]=AD[1];
12.  If(Nodo.dato.chiave<AS[1] || Nodo.dato.chiave>AD[0] ) {A[0]=-1; A[1]=-1;}
13.  return A;
```

# Soluzione 2 esercizio 1

- L'algoritmo ricorsivo effettua una visita inorder che prende in input un nodo u e una chiave k che rappresenta la chiave del nodo visiato prima di u.
- Se il nodo u ha un figlio sinistro, la chiave k sara` aggiornata con quella dell'ultimo nodo visitato nella visita del sottoalbero sinistro prima di confrontarla con la chiave di u.
- L'algoritmo ricorsivo restituisce
  - la chiave di u se u non ha figlio destro
  - la chiave dell'ultimo nodo visitato nella visita del sottolabero destro se u ha il figlio destro.
  - -1 se scopre che l'albero radicato in u non e` un BST (per semplicita` assumiamo che le chiavi siano dei numeri positivi)

# Soluzione 2 esercizio 1

```
1. isBST (Albero):
2.         if(Albero.dimensione==0) return true;
3.          nodo=isBST_Aux(Albero.radice,null);
4.         return(nodo!=-1);
5. }
```

# Soluzione 2 esercizio 1

```
1.  isBST_Aux (u,k):
2.        if(u.sx!=null) {//se c'e` il figlio sinistro aggiorniamo k
3.           k=isBST_Aux(u.sx,k);
4.           if(k!=-1) return -1;}
5.        if(k>u.dato.chiave) return -1;
6.        if(u.dx!=null){
7.             result= isBST_Aux(u.dx,u.dato.chiave);
8.              if(result!=-1) return -1;}
9.       else result=u.dato.chiave;
10.      return result;
```

# Esercizio 2

- Scrivere un algoritmo che, data una chiave k, restituisce il numero di chiavi minori o uguali k nello BST.

# Soluzione 1 per l'esercizio 2

- Assumiamo che ciascun nodo u contenga il campo dim che rappresenta il numero di nodi del sottoalbero con radice u

# Soluzione 1 per l'esercizio 2

```
1. Rango (Albero,k):
2. IF(Albero.dimensione==0)
3.     return null;
4. ELSE
5.   return Rango_Aux(Albero.radice, k);
```

# Soluzione 1 per l'esercizio 2

```
1.  Rango_Aux (Nodo,k):
2.  IF(Nodo == null)
3.      Return 0;
4.  IF(k==Nodo.dato.chiave)
5.    return 1+Dimensione(Nodo.sx);
6.  IF(k<Nodo.dato.chiave)
7.    return Rango_Aux(Nodo.sx, k);
8.  ELSE{
9.        r=Rango_Aux(Nodo.dx,k);
10.       Return r+1+Dimensione(u.sx);
11. }
```

# Soluzione 1 esercizio 2

1. Dimensione(u):
2. IF(u==null) Return 0;
3. Else Return u.dim;
4. }

# Soluzione 2 esercizio 2

Non assumiamo l'esistenza del campo dim nei nodi

```
1. Rango (Albero,k):
2. IF(Albero.dimensione==0)
3.     return null;
4. ELSE
5.   return Rango_Aux(Albero.radice, k);
```

# Soluzione 2 esercizio 2

```
1.  Rango(Nodo,k):
2.          if (Nodo==null) return 0;
3.           chiaveNodo = Nodo.dato.chiave;
4.            c=0;
5.            if (k<=chiaveNodo ) {
6.                  c= Rango(Nodo.sx,k);
7.                  if(k==chiaveNodo)c =c+1;
8.           }
9.              if (k>chiaveNodo)
10.                {
11.                 c=Rango(Nodo.sx,k);
12.                 c=c+1;
13.                 c=c+Rango(Nodo.dx,k);
14.                 }
15.       return c;
```

# Esercizio 4.8

- Scrivere lo pseudocodice di un algoritmo che trova l'elemento di rango m di uno BST. Si assuma che ogni nodo u contenga il campo u.dim = dimensione del sottoalbero radicato in u

# Soluzione esercizio 4.8

```
1. ElementoDiRango (Albero,m):
2. IF(Albero.dimensione==0||m>Albero.radice.dim)
3.     return null;
4. ELSE
5.   return ElDiRangoAux(Albero.radice, m);
```

# Soluzione esercizio 4.8

```
1. ElDiRangoAux(u, m):    //caso u==null non si puo` verificare
2. IF(m==u.sx.dim+1)return u.dato;
3. IF(u.sx!=null&& m<=u.sx.dim)
4.       return ElDIRangoAux(u.sx,m);
5. IF(u.dx!=null)
6.       return ElDIRangoAux(u.dx,m-u.sx.dim-1);
7. Else return null; //non c'e` l'elemento di rango m (in realta` non si verifica mai)
```

# Esercizio 4.10 a

Modificare l'algoritmo di inserimento dello BST in modo che funzioni anche quando gli elementi possono avere chiavi uguali

# Soluzione esercizio 4.10 a

1. Inserisci(Albero,e):
2. Return InserisciInAlbero(Abero.radice, e);

# Soluzione esercizio 4.10 a

1. InserisciInAlbero( u, e ):
2. if(u==null){
3. u = NuovoNodo();
4. u.dato = e; u.sx = u.dx = null; }
5. else if (e.chiave <= u.dato.chiave) { u.sx = InserisciInAlbero( u.sx, e ); }
6. else if (e.chiave > u.dato.chiave) { u.dx = InserisciInAlbero( u.dx, e );
7. }

8. return u;

# Esercizio 4.10 b

- Scrivere lo pseudocodice dell'algoritmo Conta che conta il numero di occorrenze della chiave k in uno BST

# Soluzione esercizio 4.10 b

1. Conta (Albero,k):
2. IF(Albero.dimensione==0)
3. return null;
4. ELSE
5. return ContaAux(Albero.radice, k);

# Soluzione esercizio 4.10 b

1. ContaAux(u, k):
2. IF(u==null) return 0;
3. IF(k==u.dato.chiave)return ContaAux(u.sx,k)+1;
4. IF(k<u.dato.chiave) return ContaAux(u.sx,k);
5. Else return ContaAux(u.dx,k);