Projektovanje algoritama

L07. Strukture podataka. Stabla (BST)

Osnovne strukture podataka

Za ponoviti:

- LIFO (Last-In-First-Out)
- FIFO (First-In-First-Out)

Stack

- STACK-EMPTY (S) $T(n) = \theta(1)$
- PUSH (S, x) $T(n) = \theta(1)$
- POP (S) $T(n) = \theta(1)$

Queue

- ENQUEUE (Q, x) $T(n) = \theta(1)$
- DEQUEUE (Q) $T(n) = \theta(1)$

Linked List

- LIST-SEARCH (L, k) $T(n) = \theta(n)$
- LIST-INSERT (L, x) $T(n) = \theta(1)$
- LIST-DELETE (L, x) $T(n) = \theta(1)$

Stablo (Rooted Tree)

- Kako implementirati sve ranije navedene strukture?
  - STACK
  - QUEUE
  - LINKED LIST
- Implementacija binarnog stabla
- Implementacija stabla sa proizvoljnim brojem naslednika
  - Levi pokazivač na dete, desni pokazivač na brata/sestru

Binary Search Tree (BST)

Osobina BST

Ako je *x* čvor stabla:

- Svi čvorovi *y* u levom podstablu zadovoljavaju: *y.key <= x.key*
- Svi čvorovi *y* u desnom podstablu zadovoljavaju: *y.key >= x.key*

Binary Search Tree (BST) – šetnja

```
INORDER-TREE-WALK(x)
  if x != NIL
    INORDER-TREE-WALK(x.left)
    print x.key
    INORDER-TREE-WALK(x.right)
```

$$T(n) = \theta(n)$$

Binary Search Tree (BST) – pretraga

```
TREE-SEARCH(x,k)
  if x == NIL or k == x.key
    return x
  if k < x.key
    return TREE-SEARCH(x.left, k)
  else
    return TREE-SEARCH(x.right, k)
```

$$T(n) = \theta(\lg n)$$

Iterativna realizacija je mnogo efikasnija!

Binary Search Tree (BST) – min/max

```
TREE-MINIMUM (x)
  while x.left != NIL
    x = x.left
  return x
```

$$T(n) = \theta(\lg n)$$

```
TREE-MAXIMUM (x)
  while x.right != NIL
    x = x.right
  return x
```

$$T(n) = \theta(\lg n)$$

Binary Search Tree (BST) – sledeći element

```
TREE-SUCCESSOR (x)
  if x.right != NUL
    return TREE-MINIMUM(x.right)
  y = x.p
  while y != NIL and x == y.right
    x = y
    y = y.p
  return y
```

$$T(n) = \theta(\lg n)$$

Binary Search Tree (BST) – dodavanje elementa

```
TREE-INSERT (T,z)
  y = NIL
  x = T.root
  while x != NIL
    y = x
    if z.key < x.key
      x = x.left
    else
      x = x.right
  z.p = y
  if y == NIL
      T.root = z                    // empty
  elseif z.key < y.key
    y.left = z
  else
    y.right = z
```

$$T(n) = \theta(\lg n)$$

Binary Search Tree (BST) – prenos elementa

```
TRANSPLANT(T,u,v)
  if u.p == NIL
    T.root = v
  elseif u == u.p.left
    u.p.left = v
  else
    u.p.right = v
  if v != NIL
    v.p = u.p
```

$$T(n) = \theta(1)$$

Binary Search Tree (BST) – uklanjanje elementa

```
TREE-DELETE(T,z)
  if z.left == NIL
    TRANSPLANT(T,z,z.right)
  elseif z.right == NIL
    TRANSPLANT(T,z,z.left)
  else
    ...
```

$$T(n) = \theta(\lg n)$$

```
y = TREE-MINIMUM(z.right)
  if y.p != z
    TRANSPLANT(T,y,y.right)
    y.right = z.right
    y.right.p = y
  TRANSPLANT(T,z,y)
  y.left = z.left
  y.left.p = y
```