Treap-uri

Pătrașcu Adrian-Octavian
Grupa 131

Cuprins

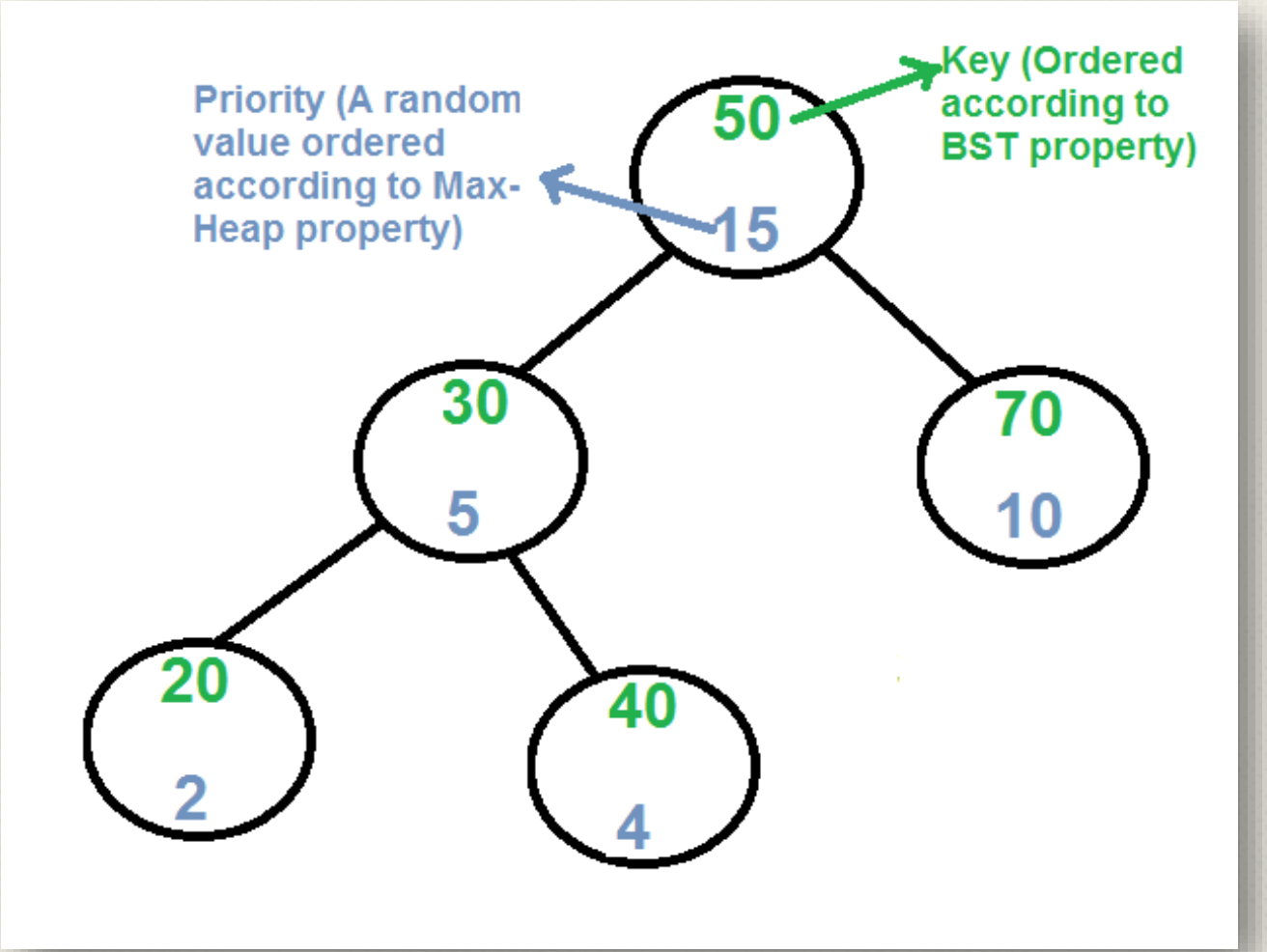
- Ce este un treap?
- Motivație (Definiții + discuție)

Operații:

1. Rotații pe arbori
2. Inserție O(logN)
3. Căutare O(logN)
4. Ștergere O(logN)

Treap

- Structură de date arborescentă care menține simultan proprietatea de arbore binar de căutare (ABC) și cea de max-heap.
- Puțin mai formal, fie (X_i, Y_i) nodurile arborelui. Treap-ul asigură structură de ABC pentru toți X_i, iar în Y_i, structura de max-heap.



De ce ne plac treap-urile?

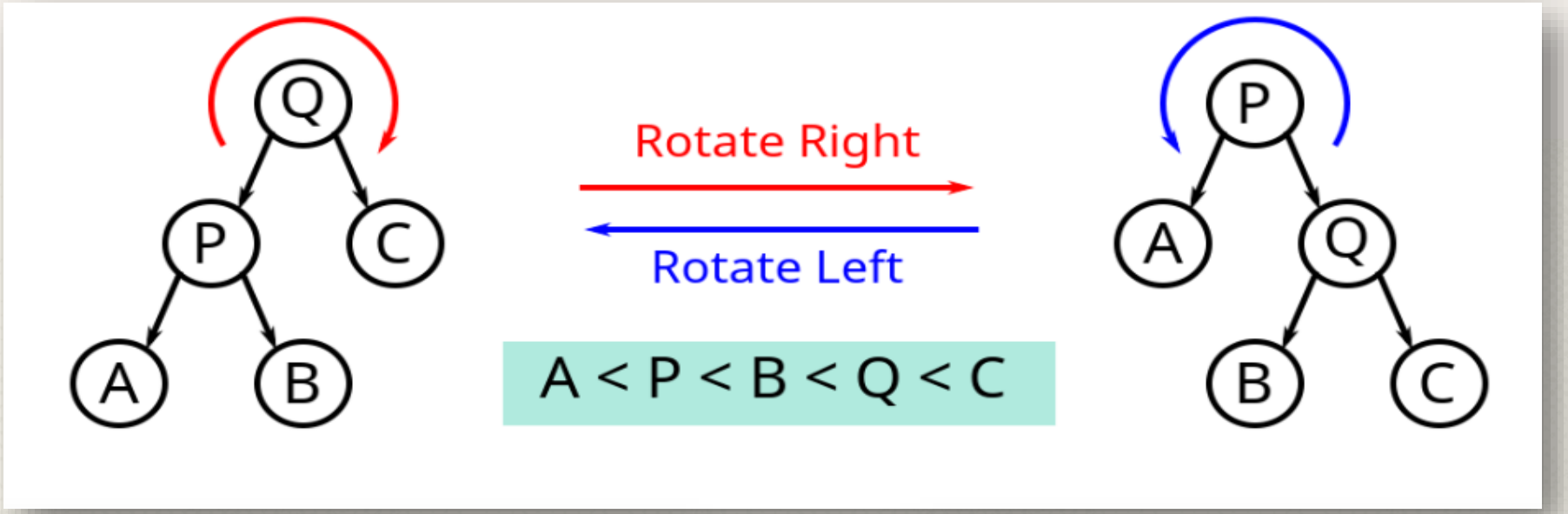
- Sunt ușor de implementat
- Sunt mai rapizi decât arborii roșu-negru (ARN) și decât skip-list-urile [1.1] , [1.2]
- Cu puține modificări, permit abordarea multor tipuri de query-uri și update-uri (sume, cmmdc, rotații etc pe intervale)

```
struct Treap
{
    int key, p;
    Treap *left, *right;
};
```

```
Treap* New_node(int x)
{
    Treap* nod = new Treap;
    nod->key = x;
    nod->p = rand();
    nod->left = nod->right = NULL;
    return nod;
}
```


Arbore
binar de
căutare + max-heap
Treap
imgflip.com

Rotații pe arbori




```
Treap* rot_right(Treap* y)
{
    Treap *x = y->left;
    y->left = x->right;
    x->right = y;
    return x;
}
```

```
Treap* rot_left(Treap* y)
{
    Treap *x = y->right;
    y->right = x->left;
    x->left = y;
    return x;
}
```

Inserție

- Inserăm, recursiv, nodul ca într-un ABC.
  Odată inserat, la fiecare pas înapoi în recursivitate, rotim în sens invers subarborele cu rădăcina în nodul curent, în funcție de tipul fiului (stâng/drept).

```
void insert(Treap* &node, int key)
{
    if(node==NULL)
        node= New_node(key);    ///in curs aceste doua if-uri sunt invers si nu merge
    if(node->key==key)
        return;
    else
    {
        if(key<node->key)
        {
            insert(node->left, key);
            if(node->p < node->left->p)
                node= rot_right(node);
        }
        else
        {
            insert(node->right, key);
            if(node->p < node->right->p)
                node= rot_left(node);
        }
    }
}
```

Căutare

- Exact ca la un ABC, anume:

1) Dacă valoarea căutată este mai mică decât cea pe care o caut, atunci cobor în subarborele stâng.
2) Analog, dacă valoarea căutată este mai mare.

```
Treap* Search(Treap* node, int key)
{
    if(node->key == key)
        return node;
    if(key < node->key)
        Search(node->left, key);
    else
        Search(node->right, key);
}
```

Ștergere

- Există trei cazuri în care se poate afla un nod pe care vrem să-l ștergem:

1. E frunză -> Îl ștergem direct.
2. Are numai un fiu -> Fiul ia locul nodului respectiv
3. Are doi fii -> Rotim în locul rădăcinii subarborelui făcut din nodul curent fiul cu prioritatea cea mai mare. Repetăm algoritmul până când ne aflăm în unul dintre primele două cazuri.

```
Treap* Delete(Treap* node, int key)
{
    if(!node)
        return node;
    if(key < node->key)
        node->left = Delete(node->left, key);
    else if(key > node->key)
        node->right = Delete(node->right, key);
    else if(!node->left || !node->right)
    {
        Treap* aux = (node->left ? node->left : node->right);
        delete node;
        node = aux;
    }
    else if(node->left->p > node->right->p)
    {
        node = rot_right(node);
        node->right = Delete(node->right, key);
    }
    else
    {
        node = rot_left(node);
        node->left = Delete(node->left, key);
    }
    return node;
}
```

Bibliografie

- [1.1] pinporelmundo: Skip Lists compared with Treaps and Red-Black Trees
- [1.2] 7.2 Treap: A Randomized Binary Search Tree (opendatastructures.org) (La secțiunea 7.2.1)
- [2] Treapuri (infoarena.ro)
- [3] Treap | Set 2 (Implementation of Search, Insert and Delete) – GeeksforGeeks
- Despre rotiri de arbori: Balanced binary search tree rotations – YouTube
- Implementarea operațiilor: Treap - Pastebin.com

Final