10-§. B daraxtlar

10.1. B daraxt ta’rifi

B daraxti (inglizcha *B-tree*) – izlash, qo'shish va o'chirish imkonini beradigan, juda ko‘pshoxli muvozanatlashgan qidiruv daraxti. Tugunlari *n* bo'lgan B daraxti O(logn) balandlikka ega bo‘ladi. Tugun shoxlari soni bittadan bir necha minggacha bo'lishi mumkin (odatda, B-daraxtining shoxlanish darajasi daraxt ishlaydigan qurilma (disklar) xususiyatlari bilan belgilanadi). B-daraxtlar O(logn) ko'p dinamik to'plam amallarini o'z vaqtida bajarish uchun ham ishlatilishi mumkin.

B daraxti birinchi marta 1970-yilda R. Bayer va E. Makkreyt tomonidan taklif qilingan.

B daraxt strukturasi. B daraxti mukammal muvozanatlashgan, ya'ni uning barcha barglarining chuqurligi bir xil. B daraxti quyidagi xususiyatlarga ega (t – bu daraxt parametrlari, B daraxtining *minimal darajasi* deyiladi, 2 dan *kam* emas):

- Ildizdan tashqari har bir tugun hech bo'lmaganda *t-1* kalitni o'z ichiga oladi va har bir ichki tugun kamida avlodli *t* tugunlarga ega. Agar daraxt bo'sh bo'lmasa, ildizda kamida bitta kalit bo'lishi kerak.
- Har bir tugun, ildizdan tashqari, ichki tugunlarda ko'pi bilan *2t - 1* kalitni va ko'pi bilan *2t* avlodni o'z ichiga oladi
- Ildizda daraxt bo'sh bo'lmasa bittadan *2t-1* gacha kalit va balandligi 0 dan katta bo'lsa 2 dan 2t gacha avlodni o'z ichiga oladi.
- Daraxtning har bir tugunida k_1, ..., k_n kalitlari bo'lgan barglardan tashqari n + 1 avlodlari bor. i-avlodda $[k_{i-1}; k_i]$, $k_0 = -\infty$, $k_{n+1} = \infty$ kesmaning kalitlari mavjud.
- Har bir tugunning kalitlari kamaymaydigan tartibda tartiblangan.
- Barcha barglar bir xil darajada.

B daraxtlar disklarda (fayl tizimlarida) yoki boshqa to'g‘ridan-to'g‘ri kiruvchi bo'lmagan saqlash muhitlarida, shuningdek, ma'lumotlar bazalarida foydalanish uchun mo'ljallangan. B daraxtlar qizil-qora daraxtlarga o'xshaydi, lekin ular diskni o'qish/yozish amallarini minimallashtirishda yaxshiroq hisoblanadi.

Daraxtlar - bu dinamik to'plam amallarni bajaradigan ma'lumotlar tuzilmalari. Bunday amallar sifatida elementni qidirish, minimal (maksimal) elementni qidirish, kiritish, o'chirish, avlod-ajdodga o'tish, avlodga o'tish kabilarni keltirish mumkin. Shunday qilib, daraxt oddiy lug‘at sifatida ham, ustivor navbat sifatida ham ishlatilishi mumkin. Daraxtlardagi asosiy amallar uning balandligiga mutanosib vaqtda bajariladi. Muvozanatlashgan daraxtlar ularning balandligini kamaytiradi (masalan, tugunlari bo'lgan ikkilik muvozanatli daraxtning balandligi *logn*).

Bu standart qidiruv daraxtlarida qanday muammo bor? Oldingi mavzularda aytib o'tilgan daraxtlardan biri tasvirlangan ulkan ma'lumotlar bazasini ko'rib chiqaylik. Shubhasiz, bu daraxtlarning barchasini tezkor xotirada saqlay olmaymiz – ma'lumotlarning faqat bir qismini saqlaymiz, qolganlari uchinchi tomon vositasida saqlanadi (masalan, kirish tezligi ancha past bo'lgan qattiq diskda). Qizil-qora yoki Dekart kabi daraxtlar uchinchi tomon vositalariga kirishni talab qiladi. Katta *n* lar uchun bu juda ko'p. Aynan mana shu muammo B daraxtlar hal qilish imkonini beradi.

B daraxtlari ham muvozanatlashgan daraxtlardir, shuning uchun standart amallarni bajarish uchun vaqt balandlikka mutanosib. Ammo, boshqa daraxtlardan farqli o'laroq, ular maxsus disk xotirasi bilan ishlash uchun yaratilgan (oldingi misolda - uchinchi tomon tarqatuvchi), aniqrog‘i ular kirish -chiqish turidagi murojaatlarni kamaytiradi.

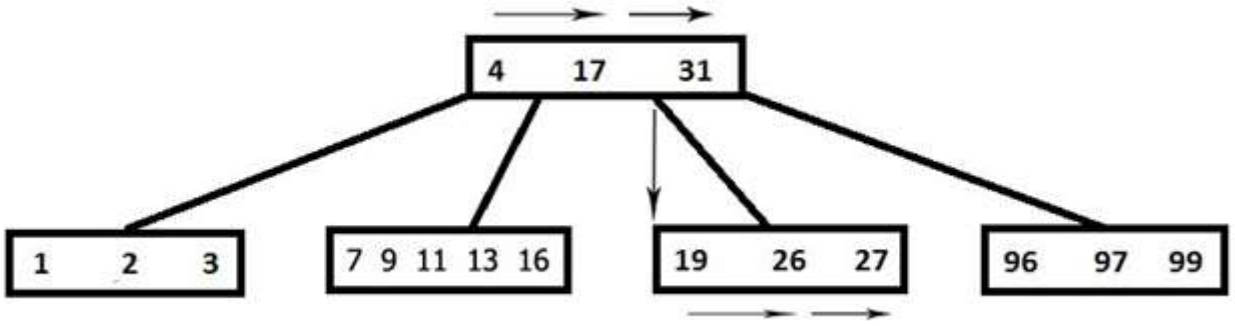
10.2. B daraxtda amallar

B daraxtda izlash algoritmi. Yuqorida aytib o'tilganidek, B daraxti barcha standart qidirish, qo'shish, o'chirish va hokazo amallarni bajaradi.

B daraxtida izlash binar daraxtni qidirishga juda o'xshaydi, faqat bu yerda biz avlodga yo'lni 2 variantdan emas, balki bir nechta variantdan tanlashimiz kerak. Aks holda, farqi bo'lmay qoladi. Quyidagi 46-rasmda 27-kalitni qidirish ko'rsatilgan. Tasvirni ko‘rib chiqaylik (va shunga mos ravishda standart qidirish algoritmi):

- Biz ildiz kalitlarini kerak bo'lguncha o'tamiz. Bu holda 31 ga yetdik.

- Bu kalitning chap tomonidagi avlodga tushamiz.
- 27 dan kichik bo‘lgunga qadar yangi tugunni kalit bo‘yicha izlaymiz. Bunday holda, biz 27 ni topdik va to'xtadik.

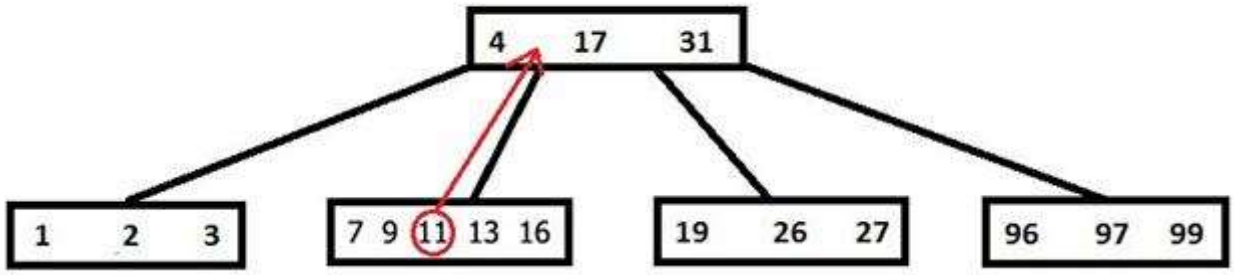


46-rasm. a) B daraxtda izlash algoritmining bajarilishi

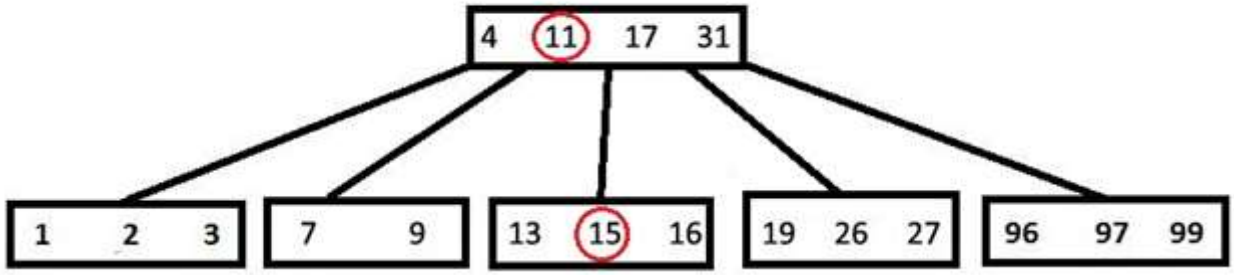
Izlash amali O(t*logt n) vaqtida bajariladi, bu yerda *t* - minimal daraja. Bu yerda disk amallarini faqat O (logt n) da bajarishimiz muhim qismidir.

B daraxtlarda element qo‘shish. Izlashdan farqli o'laroq, qo'shish usuli ikkilik daraxtga qaraganda ancha murakkab, chunki yangi barg yaratish va unga kalit qo'yish mumkin emas, chunki B daraxtining xususiyatlari buziladi. Kalitni allaqachon to'ldirilgan bargga kiritish mumkin emas. Tugunni ikkiga bo'lish amali kerak, agar barg to'ldirilgan bo'lsa, unda *2t-1* kalit bor edi. 2 ga *t-1* ga bo'lamiz undan kam kalitlar va oxirgi t-1 ajdod tuguniga o'tkaziladi. Shunga ko'ra, agar avlod-ajdod tuguni ham to'lgan bo'lsa, yana bo'lishimiz kerak va shunga o'xshash ildizgacha (agar ildiz ajralgan bo'lsa, unda yangi ildiz paydo bo'ladi va daraxt chuqurligi oshadi). Oddiy ikkilik daraxtlar singari, joylashtirish ham ildizdan barggacha bir o'tishda amalga oshiriladi.Har bir iteratsiyada (yangi kalit uchun pozitsiyani qidirishda - ildizdan barggacha) o'tadigan barcha to'ldirilgan tugunlarni ajratamiz (bargni ham o'z ichiga olgan holda). Shunday qilib, agar natijada tugunni ajratish zarur bo'lsa, uning avlod-ajdodi to'ldirilmaganligiga ko‘rishimiz mumkin.

Quyidagi 47-rasmda xuddi o'sha daraxt qidirilmoqda (t = 3). Faqat hozir biz "15" kalitini qo'shamiz. Yangi kalitning o'rnini qidirib,



a)



b)

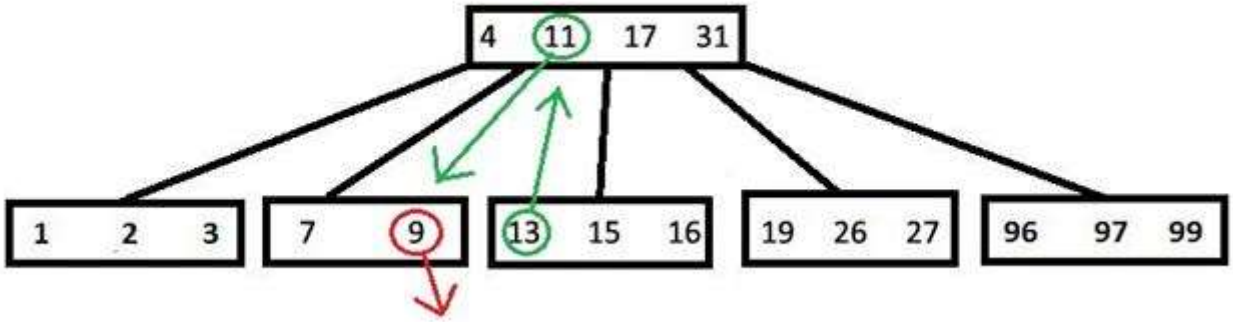
47-rasm. B daraxtlarda element qo‘shish algoritmining bajarilishi

tugallangan tugunga duch kelamiz (7, 9, 11, 13, 16). Algoritmga amal qilib, uni ajratdik - bu holda "11" avlod-ajdod tuguniga o'tadi va manba 2 ga bo'linadi. Keyin "15" tugmasi ikkinchi "ajratish" tuguniga kiritiladi. Bu holda B daraxtining barcha xususiyatlari saqlanib qolgan.

Element qo'shish amali ham O (t logt n) vaqtda bajariladi. Shunga qaramay, biz diskdagi amallarni faqat O (h) da bajaramiz, bu yerda h - daraxt balandligi.

Element o‘chirish. Kalitni B daraxtidan olib tashlash, unga element qo‘shishdan ham murakkab hisoblanadi. Buning sababi shundaki, ichki tugunni olib tashlash daraxtni umuman qayta tiklashni talab qiladi. Element qo‘shishga o'xshab, biz B daraxtning xususiyatlarini saqlaganligimizni tekshirishimiz kerak, faqat bu holda biz kalitlarning t-1 bo'lishini kuzatishimiz kerak (ya'ni, agar bu tugundan kalit o'chirilsa, tugun mavjud bo'la olmaydi). O'chirish algoritmini ko'rib chiqamiz:

1) Agar bargdan o'chirish sodir bo'lsa, unda qancha kalit borligini tekshirish kerak. Agar t-1 dan ko'p bo'lsa, biz shunchaki o'chirib tashlaymiz va boshqa hech narsa qilishimiz shart emas. Aks holda, agarda t-1 dan ortiq kalitlarni o'z ichiga oluvchi qo'shni barg bo'lsa (uning yonida joylashgan va avlod-ajdodi bir xil bo'lsa), biz qo'shni tugunning qolgan kalitlari orasidagi ajratuvchi bo'lgan bu qo'shnidan kalitni tanlaymiz. Bu kalit k_1 bo'lsin. Avval tugunni va uning qo'shnisini ajratuvchi asosiy tugundan k_2 kalitini tanlaymiz. Kerakli kalitni manba tugundan olib tashlaylik (o'chirilishi kerak edi), bu tugunga k_2 ni tushiring va ajdod tugunidagi k_2 o'rniga k_1 qo'ying. Tushunarli bo'lishi uchun quyida "9" tugmasi o'chirilgan rasm (48 -rasm) ko'rsatilgan. Agar bizning tugunning barcha qo'shnilarida t-1 kalitlari bo'lsa. Keyin biz uni qo'shnisi bilan birlashtiramiz, kerakli kalitni o'chirib tashlaymiz va bu ikkita "sobiq" qo'shnilar uchun ajratuvchi bo'lgan avlod-ajdod tugunining kaliti, yangi tashkil etilgan tugunimizga o'tamiz (bu, albatta, undagi mediana bo'ladi).

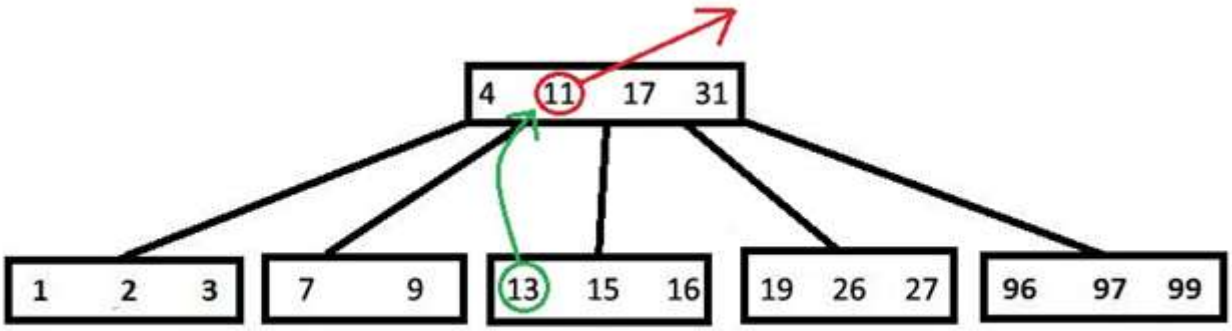


48-rasm. B daraxtda element o‘chirish algoritmining bajarilishi

2) Endi x ichki tugunidan k kalitini olib tashlashni ko'rib chiqaylik. Agar k kalitidan oldingi avlod tugunida t -1 dan ortiq kalitlar bo'lsa, biz k_1 ni topamiz - bu tugunning pastki daraxtida k. Uni o'chirib tashlaymiz (Algoritmni rekursiv tarzda ishga tushiramiz). Manba tugundagi k ni k_1 bilan almashtiramiz. Agar k kalitidan keyingi avlod tugunida t-1 dan ortiq kalit bo'lsa, biz ham xuddi shunday ishni bajaramiz. Agar ikkalasida ham (keyingi va oldingi avlod tugunlarida) t-1 kalitlari bo'lsa, biz bu avlodlarni birlashtiramiz, k-ni ularga o'tkazamiz, so'ngra k-ni yangi tugundan olib tashlaymiz (biz algoritmimizni rekursiv tarzda ishga

tushiramiz). Agar ildizning oxirgi 2 avlodi birlashsa, ular ildizga aylanadi va oldingi ildiz olib tashlanadi. Rasm quyida ko‘rsatilgan (49-rasm), bu yerda "11" ildizdan chiqariladi (keyingi tugunda t-1 avlodlari ko'p bo'lgan holat).

O'chirish jarayoni O (t logt n) qo'shilishi bilan bir xil vaqtni oladi va disk operatsiyalari faqat O (h) talab qilinadi, bu yerda h - daraxt balandligi.



49-rasm. B daraxtda element o‘chirish algoritmining bajarilishi

10.3. B-daraxtni realizatsiya qilish

```
/* B-daraxtning minimum darajasi*/
#define T 2        /* 2-3-4 B-tree */
struct btree {
int leaf;
int nkeys;
int *key;
int *value;
struct btree **childj
};
```

B-daraxtni hosil qilish

```
struct btree *btree_create()
{
struct btree *node;
node = malloc(sizeof(*node));
node->leaf = 1;
node->nkeys = 0;
node->key = malloc(sizeof(*node->key) *
```

```
        2 * T - 1);
node->value = malloc(sizeof(*node->value)
        2 * T - 1);
node->child = malloc(sizeof(*node->child)
2 * T);

        return node;
        }
```

B daraxtda element izlash

```
        void btree_lookup(struct btree *tree, int key,
        struct btree **node, int *index)
        {
        int i;
        for (i = 0; i < tree->nkeys && key > tree->key[i]; ) {
i++;
        }
        if (i < tree->nkeys && key == tree->key[i]) {
        *node = tree;
        *index = i;
return;
        }
        if (!tree->leaf) {
        /* Disk read tree->child[i] */
btree_lookup(tree, key, node, index);
        } else {
        *node = NULL;
        }
```

B daraxtda element qo‘shish

```
        struct btree *btree_insert(struct btree *tree,
        int key, int value)
        {
        struct btree *newroot;
        if (tree == NULL) {
        tree = btree_create();
tree->nkeys = 1;
tree->key[0] = key;
```

```
tree->value[0] = value;
return tree;
        }
        if (tree->nkeys == 2 * T - 1) {
        newroot = btree_create(); /* Create empty root */
        newroot->leaf = 0;
        newroot->child[0] = tree;
        btree_split_node(tree, newroot, 0);
        return btree_insert_nonfull(newroot, key, value);
        }
        return btree_insert_nonfull(tree, key, value);
        }

        B-daraxtda tugunni ajratish

        void btree_split_node(struct btree *node,
        struct btree *parent, int index)
        {
        struct btree *z;
int i;
        z = btree_create();
z->leaf = node->leaf;
z->nkeys = T - 1;
for (i = 0; i < T - 1; i++) {
        z->key[i] = node->key[T + i];
z->value[i] = node->value[T + i];
        }
        if (!node->leaf) {
        for (i=0; i<T; i++)
        z->child[i] = node->child[i + T]j
        }
        node->nkeys = T - 1;
        /* Ajdod tugunga mediana kalitni kiritish */
for (i = parent->nkeys; i >= 0 && i <= index + lj i--)
parent->child[i + 1] = parent->child[i];
parent->child[index + 1] = z;
        for (i = parent->nkeys - 1; i >= 0 && i <= index; i--)
parent->key[i + 1] = parent->key[i];
parent->value[i + 1] = parent->value[i];
```

```
        }
        parent->key[index] = node->key[T - 1];
parent->value[index] = node->value[T - 1];
parent->nkeys++;

        struct btree *btree_insert_nonfull(
        struct btree *node, int key, int value)
        {
        int i;
        i = node->nkeys;
if (node->leaf) {
        for (i = node->nkeys - 1; i > 0 &&
key < node->key[i]; i--)
        {
        node->key[i + 1] = node->key[i]
        }
        node->key[i + 1] = key;
node->nkeys++;
        } else {
        for (i = node->nkeys - 1; i > 0 &&
key < node->key[i]; )
        {
        i--;
        }
        i++;
        if (node->child[i]->nkeys == 2 * T - 1) {
        btree_split_node(node->child[i], node, i);
if (key > node->key[i])
i++J
        }
        node = btree_insert_nonfull(node->child[i],
key, value);
        }
        return node;
        }
        int main()
        {
        struct btree *tree;
        tree = btree_insert(NULL,       3, 0);
```

```
    tree = btree_insert(tree,  12, 0);
    tree = btree_insert(tree,  9, 0);
    tree = btree_insert(tree,  18, 0);
    return 0;
}
```

Mavzu yuzasidan savollar:

1. B daraxt nima
2. B daraxtda izlash qanday amalga oshiriladi?
3. B daraxtda element o‘chirish qanday amalga oshiriladi?
4. B daraxtda element qo‘shish qanday amalga oshiriladi?
5. B-daraxt ma’lumotlar strukturasi qo‘llaniladigan sohalarga qaysilar kiradi?

Mustaqil ishlash uchun masalalar:

1. B daraxtida element olib tashlash funksiyasini yozing va uni daraxtda qo‘llang
2. B-daraxtda element qo‘shish funksiyasini optimallashtiring

11-§. Ustivor navbatlar

Ko'pgina ilovalar kalitlarga ega bo'lgan elementlarni qayta ishlashni talab qiladi, lekin ular to'liq tartibda va birdaniga hammasi emas. Ko'pincha, biz bir qator narsalarni to'playmiz, so'ngra eng katta kalit bilan ishlov beramiz, keyin ko'proq narsalarni to'playmiz, so'ngra hozirgi eng katta kalit bilan ishlov beramiz va hokazo. Bunday muhitda tegishli ma'lumotlar turi ikkita amalni qo'llab-quvvatlaydi: maksimal miqdorni o‘chirish va joylashtirish. Bunday ma'lumotlar turi **ustivor navbat** deb nomlanadi.

Ustivor navbatlar odatdagi navbat yoki stek ma'lumotlar tuzilmasiga o'xshash abstrakt ma'lumotlar turi bo'lib, unda har bir element qo'shimcha ravishda bog‘liq bo'lgan "ustivorlikka" ega. Ustivor navbatda yuqori ustivor element past ustivor elementdan oldin xizmat qiladi.