```
    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.

Ustivor navbatlar ko'pincha uyum (kucha) bilan amalga oshirilsa-da, ular konseptual jihatdan uyumlardan farq qiladi. Ustivor navbat - bu "ro'yxat" yoki "karta" ga o'xshash narsa; Ro'yxat bog‘langan ro'yxat yoki massiv yordamida amalga oshirilishi mumkin bo'lganidek, ustivor navbat uyum yoki tartiblanmagan massiv kabi boshqa usullar yordamida amalga oshirilishi mumkin.

Ustivor navbat - bu yozuvlar bir-biri bilan chiziqli taqqoslanadigan kalitlarga (masalan, raqamlar) ega bo'lgan va ikkita amalni realizatsiya qiladigan axborot tizimidir. Bu ikki amal tizimga tasodifiy yozuvni kiritish va yozuv tizimidan eng kichigi bilan tanlov kalit.

Dasturiy ta'minot tizimlarida ustivor navbatlar juda keng tarqalgan va dasturlarning ishlashi to'g‘ridan-to'g‘ri ularni amalga oshirish samaradorligiga bog‘liq.

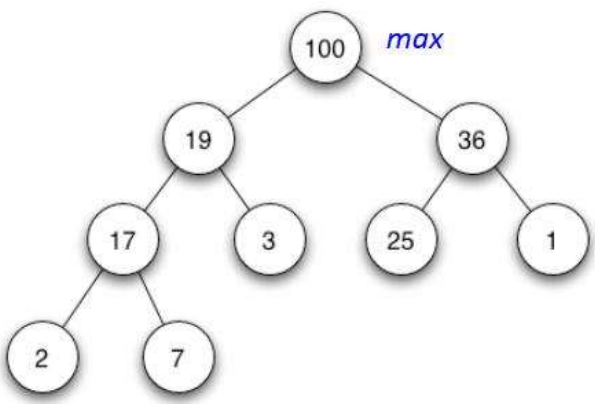
Ustivorda navbatda qo‘llab-quvvatlanadigan amallar quyidagilar hisoblanadi:

1) Insert - navbatga element qo'shish
2) Max - ustivorligi yuqori bo'lgan elementni qaytaradi
3) ExtractMax - navbatdagi eng ustivor elementni olib tashlaydi
4) IncreaseKey - berilgan elementning ustivor qiymatini o'zgartiradi
5) Merge - ikkita navbatni bittaga birlashtiradi

11.1. Binar uyum (kucha) - piramida (binary heap)

Binar uyum (binary heap) bu quyidagi shartlarni qanoatlantiradigan binar daraxtdir:

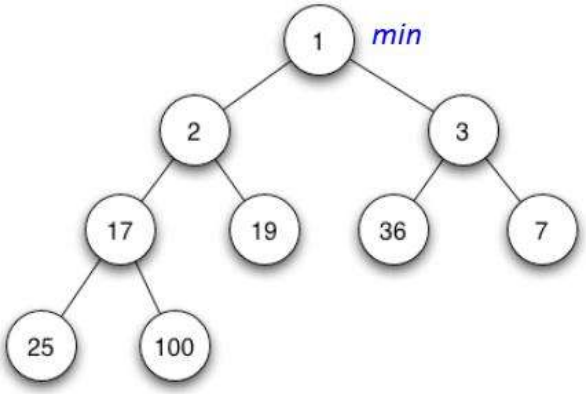
- Har qanday uchning ustivorligi, uning avlodlarining ustivorligidan kichik emas.
- Daraxt to'liq ikkilik daraxt bo‘lishi uchun (complete binary tree) - barcha darajalar chapdan o'ngga to'ldiriladi (oxirgisi bundan mustasno bo‘lishi mumkin).



O'smaydigan piramida

max-heap

Har qanday uchning ustuvorligi avlodlarning ustuvorligidan kichik emas



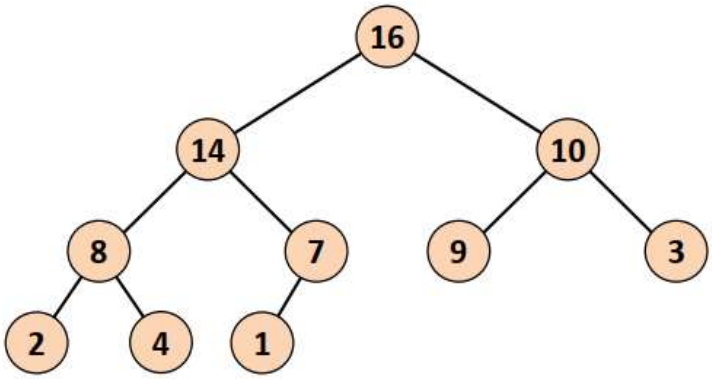
Kamaymaydigan piramida

min-heap

Har qanday uchning ustuvorligi avlodlarning ustuvorligidan katta emas

50-rasm. Binar uyum (kucha)

Massivlar orqali binar uyum (kucha) ni realizatsiya qilish

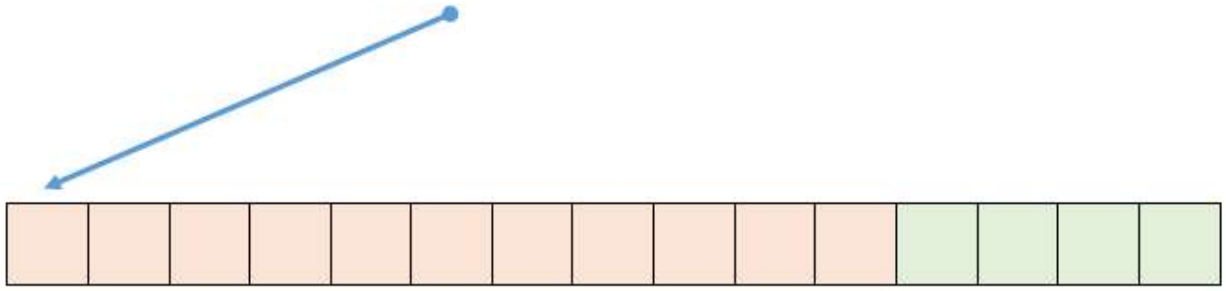


max-heap (10 ta element)

16	14	10	8	7	9	3	2	4	1				

H[1..10] ustuvorliklar (kalitlar) massivi

Daraxtning ildizi H [1] yacheykada saqlanadi - bu maksimal element;



i tugunning ajdod indeksi: $Parent(i)= [i / 2]$;
Chap avlod tugun indeksi: $Left(i) = 2i$;
O‘ng avlod tugun indeksi: Right(i) = $2i + 1$;
$H[Parent(i)] \geq H[i]$.

```
struct heapnode {
    int key;          /* kalit */
    char *value;      /* qiymat*/
};


struct heap {
    int maxsize;     /* massiv o‘lchami */
    int nnodes;      /* Kalitlar soni */
    struct heapnode *nodes; /* Nodes: [0..maxsize] */
}
```

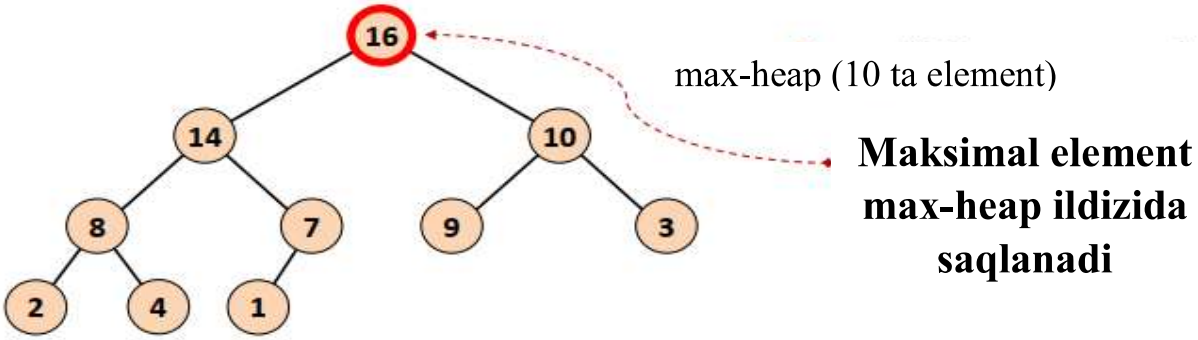
Bo‘sh uyum (kucha) hosil qilish

```
struct heap *heap_create(int maxsize)
{
        struct heap *h;
        h = malloc(sizeof(*h));
        if (h != NULL)
        {
        h->maxsize = maxsize;
        h->nnodes = 0;
    /* Heap nodes [0, 1, maxsize] */
        h->nodes = malloc(sizeof(*h->nodes) * (maxsize + 1));
        if (h->nodes == NULL)
```

```
        {
        free(h);
        return NULL;
        }
return h;
}
```

Uyumni o‘chirish

```
void heap_free(struct heap *h)
{
free(h->nodes);
free(h);
}
void heap_swap(struct heapnode *a, struct heapnode *b)
{
struct heapnode temp;
temp = *a;
*a = *b;
```

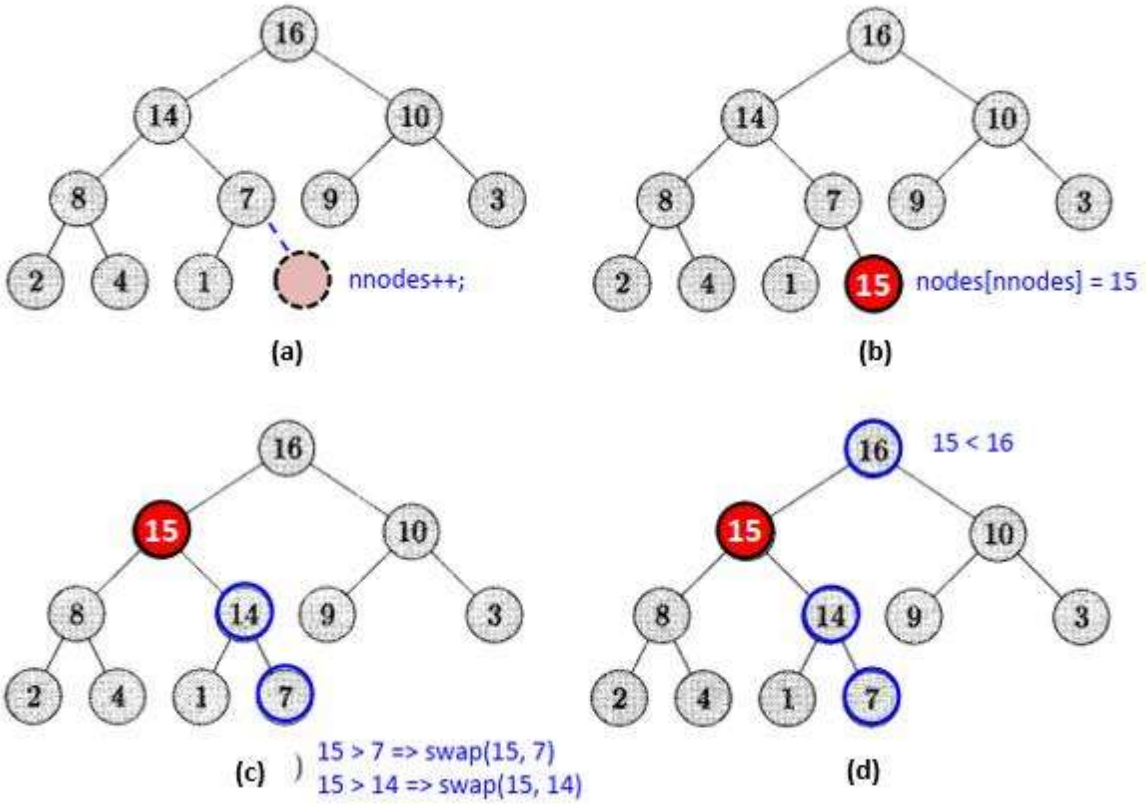


16	14	10	8	7	9	3	2	4	1				

51-rasm. Maksimal elementni izlash

```
struct heapnode *heap_max(struct heap *h
{
if (h->nnodes == 0)
   return NULL;
return &h->nodes[l];
}
```

Binar uyum (kucha) ga element qo‘shish. Ustuvorligi 15 ga teng bo‘lgan elementni joylashtirish



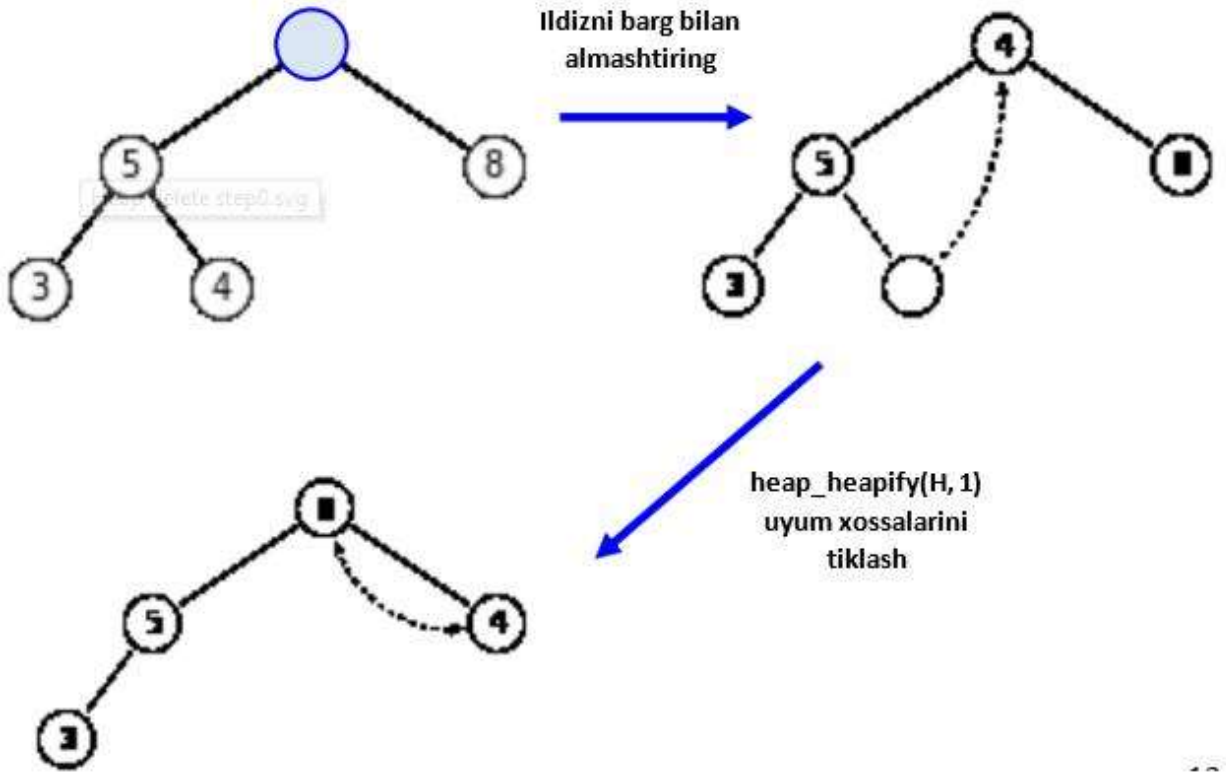
52-rasm. Binar uyum (kucha) ga element qo‘shish

Binar kuchaga element joylashtirish

```
int heap_insert(struct heap *h, int key, char *value)
{
        if (h->nnodes >= h->maxsize) {
return -1;
}
h->nnodes++;
h->nodes[h->nnodes].key = key;
h->nodes[h->nnodes].value = value;
for (int i = h->nnodes; i > 1 &&
        h->nodes[i].key > h->nodes[i / 2].key; i = i/2)
        {
                heap_swap(&h->nodes[i], &h->nodes[i / 2]);
        }
return 0;
```

}

Maksimal elementni o‘chirish



53-rasm. Maksimal elementni o‘chirish

```
struct heapnode heap_extract_max(struct heap
{
if (h->nnodes == 0)
return (struct heapnode){0, NULL};
struct heapnode maxnode = h->nodes[l];
h->nodes[l] = h->nodes[h->nnodes];
h->nnodes--;
heap_heapify(h, 1);
return maxnode;
}
```

Uyum xususiyatlarini tiklash

```
void heap_heapify(struct heap *h, int index)
{
    for (;;) {
```

```
int left = 2 * index;
int right = 2 * index + 1;
int largest = index;
if (left <= h->nnodes &&
h->nodes[left].key > h->nodes[index].key)
{ largest = left; }
if (right <= h->nnodes && h->nodes[right].key > h->nodes[largest].key)
    { largest = right; }
if (largest == index)
    break;
heap_swap(&h->nodes[index], &h->nodes[largest]);
index = largest;
    }
}
```

Kalit qiymatini oshirish

```
int heap_increase_key(struct heap *h, int index, int key)
{
if (h->nodes[index].key > key)
    return -l;
h->nodes[index].key = key;
for ( ; index > 1 && h->nodes[index].key > h->nodes[index / 2].key; index = index
/ 2)
{
        heap_swap(&h->nodes[index], &h->nodes[index / 2]);
}
return index;
}
```

Binar kucha bilan bilan ishlash

```
int main()
{
        struct heap *h;
        struct heapnode node;
        h - heap_create(100);
        heap_insert(h,      16, "16");
        heap_insert(h,      14, "14");
        heap_insert(h,      10, “10");
```

```
    heap_insert(h,      8, "8");
    heap_insert(h,      7, "7");
    heap_insert(h,      9, “9");
    heap_insert(h,      3, "3");
    heap_insert(h,      2, "2");
    heap_insert(h,      A, “4”);
    heap_insert(h,      1, "1");
    node = heap_extract_max(h);
    printf(“Item: %d\n", node.key);
    int i = heap_increase_key(h, 9, 100);
    heap_free(h);
    return 0;
}
```

11.2. Uyum (kucha)larni saralash (Heap-Sort)

Heapsort (Heapsort, "Heap sorting") - n elementlarni saralashda O(nlogn) amallarda eng yomon, o'rtacha va eng yaxshi (ya'ni kafolatlangan) holda ishlaydigan saralash algoritmi. Ishlatiladigan qo'shimcha xotira miqdori massiv kattaligiga bog‘liq emas (ya'ni O (1)).

Ushbu saralashni pufaksimon saralashning rivojlantirilgan ko‘rinishi deb qarash mumkin.

Eng yomon vaqt - $O(n\log(n))$

Eng yaxshi vaqt - $O(n\log(n))$

O‘rtacha vaqt - $O(n\log(n))$

Heap-Sort algoritmini realizatsiya qilish (C++)

```
#include <iostream>
#include <time.h>

using namespace std;

int main()
{
   srand(time(NULL));
   int const n = 100;
   int a[n];
```

```
for ( int i = 0; i < n; ++i )
   {
      a[i] = rand()%1000;
      cout << a[i] << " ";
   }
//massivni to'ldirish
//-----------Saralash-----------//
//O'sish bo'yicha saralash.
int sh = 0; //смещение
bool b = false;
for(;;) //Sikl cheksiz davom etadi
{
b = false;
for ( int i = 0; i < n; i++ )
{
   if( i * 2 + 2 + sh < n )
   {
   if( ( a[i + sh] > /*<*/ a[i * 2 + 1 + sh] ) || ( a[i + sh] > /*<*/ a[i * 2 + 2 +
sh] ) )
   {
      if ( a[i * 2 + 1 + sh] < a[i * 2 + 2 + sh] )
      {
      swap( a[i + sh], a[i * 2 + 1 + sh] );
      b = true;
      }
      else if ( a[i * 2 + 2 + sh] < a[ i * 2 + 1 + sh])
          {
             swap( a[ i + sh], a[i * 2 + 2 + sh]);
             b = true;
          }
   }
   // oxirgi ikki element uchun qo'shimcha tekshirish
         // ushbu tekshiruv yordamida siz piramidani saralashingiz mumkin
         // faqat uchta elementdan iborat
      if( a[i*2 + 2 + sh] < /*>*/ a[i*2 + 1 + sh] )
      {
      swap( a[i*2+1+sh], a[i * 2 +2+ sh] );
               b = true;
      }
```

```
        }
        else if( i * 2 + 1 + sh < n )
            {
               if( a[i + sh] > /*<*/ a[ i * 2 + 1 + sh] )
               {
                  swap( a[i + sh], a[i * 2 + 1 + sh] );
                  b = true;
               }
            }
      }
      if (!b) sh++;
      if ( sh + 2 == n ) break;
      }  //Saralash tugatildi

      //Natijani chiqarish
      cout << endl << endl;
      for ( int i = 0; i < n; ++i ) cout << a[i] << " ";


     // getch();
      return 0;
   }
```

Mavzu yuzasidan savollar:

1. Ustivor navbat nima?
2. Heap-Sort algoritmi haqida gapiring
3. Uyum tushunchasi.
4. Binar kucha bilan ishlash?
5. Ustivor navbat ma’lumotlar strukturasi qo‘llaniladigan sohalarga qaysilar kiradi?

Mustaqil ishlash uchun masalalar:

1. Masalalarda Heap-Sort algoritmini qo‘llang.
2. Binar uyumga element qo‘shish dasturini yozing