13-дәріс

ДЕРЕКТЕР ҚҰРЫЛЫМЫ РЕТІНДЕГІ АҒАШ. БИНАРЛЫ ІЗДЕУ АҒАШЫ

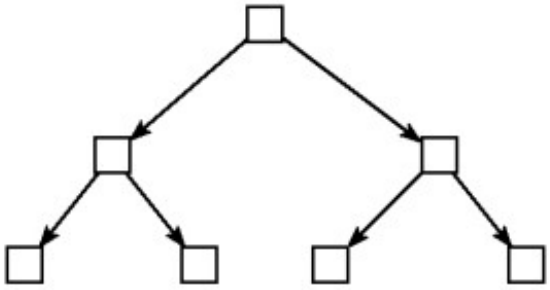
Ағаш бұл элементтерді енгізу және өшіру – қолжетімділікті сызықты құрылымға қарағанда аз уақытта орындайтын деректер құрылымы.

Ағаштар келесі тәсілмен байланысқан түйіндерден тұрады:

– бастапқы түйін түбір болады;

– әрбір түйіннің жеке тармағы болады;

– әр түйіннен тізбектей түйінмен тармақты өту арқылы түбірді алуға болады.

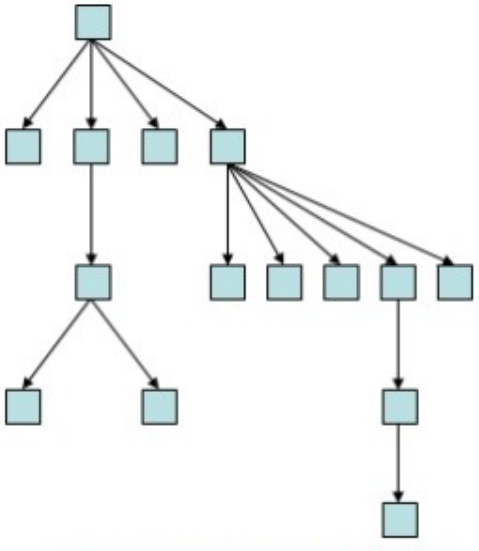


1-сурет. Базалық ағаш

Барлық тармақты түйіндер ***ата-аналық*** деп аталады, ал оған қосылған түйіндерді ***балалар*** деп атайды. Балалары жоқ түйіндерді ***жапырақтар*** деп атаймыз. Ағаштың ***биіктігі*** деп түйіннің барлық деңгейінің санын айтамыз.

1-суреттегі ағаштың биіктігі 3-ке тең және 4 түйін жапырақтан тұрады.

Ағаштар компьютерлік ғылымдардың көптеген қосымшаларында қолданылады. Мысалы, онлайн кітап дүкендерінің категориясын және ішкі категорияларын бейнелеу қажет болсын. Программалаушы оларды файлдық жүйеде бейнелеу үшін ағашты қолданады. Сонымен қатар, ағаштарды компиляторларды жүзеге асыру үшін және файлдарды алгоритмдік сығу үшін қолданылады.

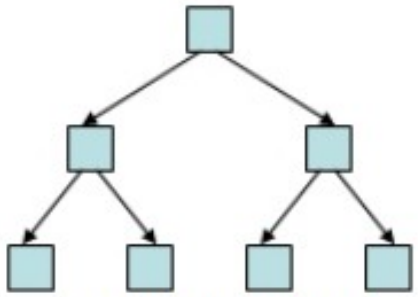


2-сурет. Ағаштардың жалпы құрылымы

Ағаштардың түрлері

Программаларды орындау барысында ағаштардың кейбір сипаттамалары мен қасиеттерін ескере отырып, бірнеше типтерге бөлінеді. Ағаштардың саны бойынша классификациясы бала түйіннің максималды санымен анықталады. Егер балалардың қандай да бір түйіндегі максималды саны шектелмесе, онда мұндай ағаштар жалпы иерархия классификациясына сәйкес келеді.

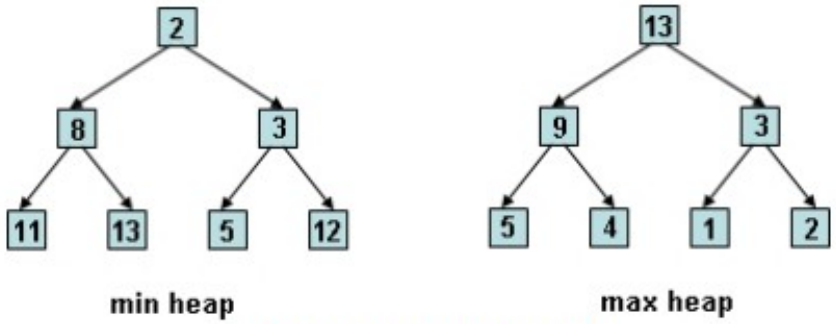
Еншілес түйіндер санымен шектелген ағаштарды жүзеге асыру ағаштардың жеке класын бейнелейді. Олардың ішіндегі кең таралған типі орындалу барысында түйіндер санын 2-ге дейін шектейді. Ағаштардың бұл типі ***бинарлы ағаш*** деп аталады.



3-сурет. Бинарлы ағаш

Бинарлы ағаштарды қосымша ағаш ішіндегі элементтерді реті бойынша классификациялауға болады. ***Төбе*** – бұл өспелі немесе кемімелі реттік нөмірлі элементтерден тұратын бинарлы ағаш, яғни төбенің әрбір деңгейінде түйін нөмірі оның астында орналасқан барлық түйіндердің шамасынан артық немесе кем болады.

Төбелердің екі типі бар: ***мин*** және ***макс*** төбе. Мин төбе ағаштың әр деңгейінде үлкен мәнге ие болатын элементтерден тұрады. Сондықтан ағаштың түбірлі түйіні ең кіші элементке тең болады. Макс төбеде түбірлі каталог барлық уақытта ең үлкен нөмірге тең, ал қалғандары сәйкесінше кемиді.

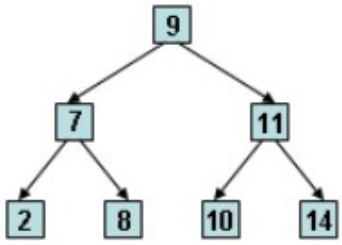


4-сурет. Мин және макс төбелер

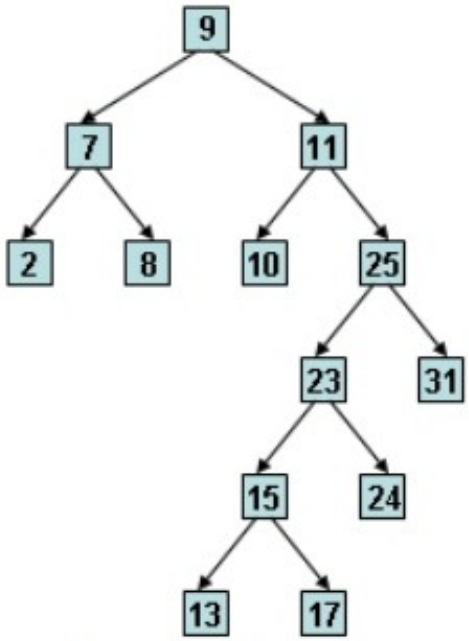
Ағаштар іздеу құрылымы ретінде

Іздеудің бинарлы ағашы

Бинарлы ағаштар элементтерге ағаштардың орналасу ретімен сипатталуы мүмкін, яғни олар элементтерді бекітілген ретпен сақтауы мүмкін. Сондықтан олар жеке элементтерді іздеу барысында тиімді болып табылады. Бинарлы іздеу ағаштары бекітілген ретпен элементтерде сақталады. Түбір элементінен кіші болатын барлық элементтер сол жақ ***child*** түбірінде сақталады. Түбір элементінен үлкен болатын барлық элементтер оң жақ ***child*** тармақталуында сақталады. Бұл қағида ағаштағы барлық түйіндер үшін рекурсивті қолданылады.

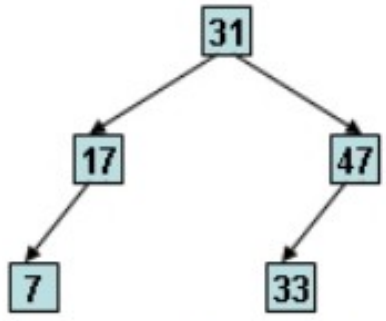


5-сурет. Бинарлы іздеу ағашы



6-сурет. Балансталмаған бинарлы іздеу ағаштары

Барлық бинарлы ағаштардың іздеу жапырақтары бір деңгейде орналаспайды. Бинарлы іздеу ағашы балансталмаған болуы мүмкін. ***Балансталмаған ағаштар*** – тармақтары бір деңгейге ерекшеленуі мүмкін ағаш. Бинарлы іздеу ағаштары элементтерін ағаш өлшеміне көбейтіп және төмендеткенде құрылымдарды қосымшалайды. Кейде бинарлы іздеу ағаштары тек бір ***child*** тармақтан тұруы мүмкін. Мұндай бинарлы іздеу ағаштарын ***толық емес ағаш*** деп атайды.



7-сурет. Толық емес бинарлы іздеу ағашы

Бинарлы іздеу ағаштарын қолдану

Бинарлы іздеу ағаштарының деректер құрылымы реттелген элементтер жиынынан тұрады. Бинарлы іздеу ағаштары белгілі бір есептерді басқа деректер құрылымына қарағанда тиімді орындауы мүмкін. ***Мысалы***, элементтерді сұрыптаудың тура ретімен қарау. Шеңбер бойымен іздеу бинарлы іздеу ағаштарын қолдануда оңай орындалатын есептердің бірі болып табылады. Бинарлы іздеу ағашының интерфейсін қарастырайық.

1-листинг BSTree класының декларациясынан тұрады. Бұл – элементтерді орналастыру, өшіру және қолжетімділік операциясынан тұратын бинарлы іздеу ағашының класы. Ол, сонымен қатар BSTree сақталған элементтер санын қайтаратын әдістен тұрады.

```
 1: template <typename T>
 2: class BSTree {
 3:
 4: protected:
 5:     BSTNode<T> *root;    // root of tree
 6:     int count;        // size of tree
 7:
 8: public:
 9:    BSTree() : root(NULL), count(0) {}
10:    BSTree(const BSTree&);
11:    virtual ~BSTree() {if (root) delete root;}
12:
13:    virtual int size() const { return count; }
14:    virtual bool insert( const T& x );
15:    virtual const T* const search( const T& x );
16:    virtual bool remove( const T& x );
17:
18: protected:
19:    virtual BSTNode<T>* copy_tree(BSTNode<T>* nodep);
20:    virtual bool insert_helper(BSTNode<T> *&nodep, const T &x);
21:    virtual  const  T*  const  search_helper(BSTNode<T>  *  nodep,
    const T &x);
22:    virtual bool remove_helper(BSTNode<T> *&nodep, const T &x);
23:    virtual     BSTNode<T>*     remove_leftmost_child(BSTNode<T>
    *&nodep);
24:
25: };
```

1-листинг. The BSTree interface

```
1:  template <typename T>
2:  const T* const BSTree<T>::search_helper(BSTNode<T> *nodep,
    const T &x) {
3:      if (nodep == 0) {
4:          return NULL;
5:      }
6:
7:      if (x == nodep->data) {
8:          return &(nodep->data);
9:      }
10:
11:     if (x < nodep->data) {
12:         return search_helper(nodep->left, x);
13:     }
14:     else {
15:         return search_helper(nodep->right, x);
16:     }
17: }
```

2-листинг. Internal use of operators of type *T*

BSTree класы ***template class*** болып табылады. Бұл әртүрлі деректер типін сақтайтын кластардың көшірмесін құруға мүмкіндік береді. Бірақ бұл класты қолданбас бұрын қолжетімділік, енгізу, өшіру элементтерін орындау үшін ==, < және > операторларын қолдану қажет.

Төбелерді қолдану

Төбелер кез келген минималды және максималды жиынына тиімді қолжетімділікті қамтамасыз ету үшін қолданылады. Бұл функция приоритетті кезекпен жабдықтау үшін қолданылады. Приоритетті кезек ***push, pop, size*** және ***empty*** әдістерімен жабдықталғандықтан регулярлы кезекті құрайтын деректер құрылымынан тұрады. Кезек приоритетінің қарапайым кезектерден ерекшелігі ***pop*** приоритет кезегі орнатылған

приоритеттерден тәуелді өшіреді. ***STL priority_queue*** адаптерімен жабдықтайтын кезек приоритеті жиі қолданылады.

Using the Tree Based STL Containers

STL set контейнері элементтерді реттелген формада сақтайды. ***Addition, removal*** және ***access*** есептері стандартты ***STL set*** арқылы кепілденеді. Бекітілген ретпен элементтерді қою ***BSTree*** кезінде операцияларды бұзбайды.

Set класының объектілерін хабарлау ***STL*** контейнерлерінің басқа хабарламаларына ұқсас болып келеді. Программалаушы деректер типін шаблон парметрлері ретінде анықтайды. ***Set*** контейнерге қолжетімді болу үшін программалаушы ***set*** кітапханасын қосуы тиіс.