Лекция 7

БАЙЛАНЫСҚАН ТІЗІМДЕР

Байланысқан тізімдер – элементтердің сызықты тізбегі. Байланысқан тізімдер мен операцияларды қолданып элементтер тізімін құруға, басқаруға болады. Векторлар да элементтер тізімі болып келеді. Векторлар мен байланысқан тізімдер арасында ұқсастық көп. Мысалы, векторлар мен байланысқан тізімдер деректер құрылымынан элементтерді өшіру және енгізу қызметтерін атқарады. Бірақ амалдарды орындау тиімділігі әртүрлі. Мысалы, алғашқы элементті өшіру амалы векторға қарағанда байланысқан элементтерде тиімді орындалады. Бірақ кездейсоқ элементтерге қолжетімділік векторларда тиімді. Тиімділіктің айырмашылықтары деректер құрылымының жадыда сақталуына тікелей байланысты болады. Векторлар үзіліссіз орналасады, ал байланысқан элементтер кездейсоқ орналасады

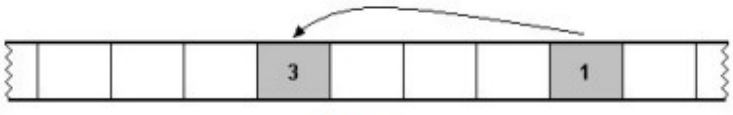
Векторлардың жадыда орналасуы

Байланысқан тізімдер элементтерді жадыда бірнеше араласпаған бөліктерде сақтайды. Яғни бұл араласқан элементтер жадыда кездейсоқ сақталатынын көрсетеді.

Байланысқан тізімдердің жадыда орналасуы

Егер әрбір элемент келесі элементке сілтеме жасайтын болса, онда бірнеше араласпаған жады ұяшықтарында элементтерді сақтауға болады. Алдыңғы суретте элементтер арасындағы сілтеме нұсқауышқа байланысты жүзеге асырылады. Байланысқан тізімдер шамасы 1-ге тең элементтен басталады. Бұл элемент келесі 2-элементке сілтеме жасайды, ал ол сәйкесінше 3-элементке сілтеме жасайды. Сілтемелерді "***linked list***" атауын беретін "***link***" деп белгілейді. Ескеретін жайт, жады тізімдегі әрбір элементке сілтемені қосымша сақтауы тиіс. Бұл - векторлар мен байланысқан тізімдер арасындағы негізгі айырмашылық.

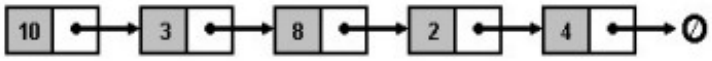
Бір-бірімен араласпаған бөліктерден тұратын элементтерді жадыда сақтау кейбір амалдардың тиімділігіне әсер етеді. Бір-бірімен араласпаған бөліктерден тұратын элементтерді жадыда сақтаудың өзіндік артықшылығы мен кемшіліктері бар. Байланысқан тізімдердің бір артықшылығы: элементтерді сілтемелерін өзгерту арқылы өшіріп енгізе алады. Мысалы, 2-суреттегі екінші элементті оңай өшіре аламыз. Ол үшін бір элемент үш элементіне нұсқайтын сілтемені жаңғыртамыз. Егер екі элементі динамикалық (***using new***) ерекшеленген болса, онда біз ол жадыны босатуымыз керек.



3-сурет

Байланысқан элементтер

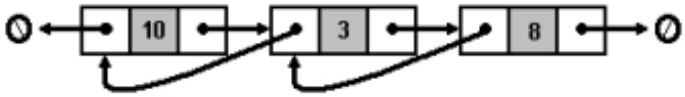
Жоғарыда айтқанымыздай, байланысқан элементтер үшін элементтердің деректерін жадыда сақтау жеткіліксіз. Әрбір элемент келесі элементтің жадыда қай жерде орналасқанын нұсқауы тиіс.



4-сурет

Суреттен көріп тұрғанымыздай, тізім 5 элементтен тұрады. Бір элементтің деректері келесі элементпен нұсқауыштар арқылы байланысқан. Сонымен қатар, соңғы элементпен байланысқан нұсқауыш нөлдік нұсқауышта орнатылады. Бұл – тізімдегі соңғы элементті білдіретін стандартты әдіс. Байланысқан тізімдегі түйін ақпараттарды тізім түрінде көрсету үшін бір элементтің деректерін сақтайды. 4-суретте түйін тізімдегі келесі элементтің нұсқауышынан тұрады. Мұндай құрылым әрбір түйінде тек бір сілтемеден тұрғандықтан, ***singly*** – ***linked list*** деп аталады.

Кез келген бағытта тізімдерді айналуды жеңілдету үшін әрбір түйінге қосымша нұсқауыш қосуға болады.



5-сурет

5-суретте әрбір түйінге бір нұсқауыш қосылған. Бұл қосылған жаңа нұсқауыш алдыңғы тізімдегі түйінге сілтеме жасайды. Мұндай тізімдерді байланыстыру типі ***doubly-linked list*** деп аталады. ***Doubly-linked list*** тізім бойынша алға және артқа қозғалуға мүмкіндік береді.

Template List Class

Байланысқан тізімдерде ***LinkedList*** деп аталатын құрылған класс тізімді тікелей аралап шығуы тиіс деп тұжырымдайық. Сондықтан біз оны ***singly-linked list*** ретінде жүзеге асыруымыз керек. Сонымен қатар, біз ***LinkedList*** класы элементтерді тізімнің соңына немесе алдыңғы бөлігінен өшіру және енгізу амалдарын орындауы тиіс деп тұжырымдайық. Кластың мүмкіндіктерін кеңейте отырып, класқа ***integers, floats, strings*** және өзге де деректер типін сақтауға мүмкіндік береміз. Сондықтан ол шаблонды класс болуы тиіс.

C++ тілінде жазылған төмендегі программа жоғарыда аталған ерекшеліктердің барлығын орындайды.

```
 1:  template <typename T>
 2:  class LinkedList {
 3:
 4:  public:
 5:      LinkedList();   // Default constructor
 6:      LinkedList(const LinkedList <T>& src);  // Copy
 7:  constructor
 8:      ~LinkedList();  // Destructor
 9:
10:      T& front();  // Element access for front of list
11:      T& back();   // Element access for back of list
12:      int size();  // count of elements in list
13:      bool empty(); // Size > 0
14:
15:    void push_front(const T&);//Insert element at beginning
16:      void push_back(const T&);   // Insert element at end
17:      void pop_front();  // Remove element from beginning
18:      void pop_back();  // Remove element from end
19:      void dump();  // Output contents of list
20:  };
```

Node ұсынысы

Кластарды функция мүшелерімен орындау коды бар болған жағдайда байланысқан тізімде түйіндер қалай бейнеленетінін көрсетеміз. Түйін байланысқан тізімдер құрылымын қолдау үшін қажетті элементтер деректерін және ақпараттарды сақтауы тиіс.

```
 1:   template <typename T>
 2:   class LinkedList {
 3:
 4:   private:
 5:       class Node {
 6:           T data;
 7:           Node* next;
 8:       };
 9:
10:   public:
11:       // rest of public members ...
```

Жоғарыда аталған класс ***singly-linked list***-тегі түйін үшін жеткілікті. Ол мәліметтерді түйін мен нұсқауыштар бойынша келесі түйінге сақтау үшін класс функция мүшелерінен тұрады. Байланысқан тізімдер класының текстінде ***Node*** класы бірнеше функциялар арқылы кеңейтілген болуы тиіс. Біріншіден, біз берілген кластың элементтерін меншіктейтін конструкторды қосамыз. Сонымен қатар ***Node*** класы деректерінің элементтеріне қолжетімді болатын ***LinkedList*** класының ***friend*** операторын енгіземіз.

```
 6: template <typename T>
 7: class LinkedList {
 8:
 9: private:
10:     class Node {
11:         friend class LinkedList<T>;
12:
13:     private:
14:         T data;
15:         Node* next;
16:
17:     public:
18:        Node(T d, Node* n = NULL) : data(d), next(n) {}
19:     ;
```

Жоғарыдағы листинг ***Node*** класын *LinkedList* класының шегінде толығымен анықтайды. ***Node*** типті объектілерді қайта-қайта анықтамас үшін арнайы осылай жасалған. Бұл листингте ***Node*** класын анықтау конструкторды қосады және деректер элементтерінің ***private*** және ***public*** айқын операторларын енгізеді. Сонымен қатар, ***friend*** операторы ***LinkedList*** класы объектілерінің ***private*** элементтеріне қол жетімді болады.

```
 6:  template <typename T>
 7:  class LinkedList {
 8:
 9:  private:
10:      class Node {
11:          friend class LinkedList<T>;
12:
13:      private:
14:          T data;
15:          Node* next;
16:
17:      public:
18:         Node(T d, Node* n = NULL) : data(d), next(n) {}
19:      ;
20:
21:      Node* head;  // Beginning of list
22:      Node* tail;  // End of list
23:      int count;   // Number of nodes in list
```

Функцияның қарапайым мүшелері

Төмендегі листинг ***front, back, size*** және ***empty*** функция мүшелері бойынша конструкторларды жүзеге асырады.

```
30: // Default constructor
31: LinkedList(void) : head(NULL), tail(NULL), count(0)
32: {}
33:
34: // Returns a reference to first element
35: T& front(void) {
36:     assert (head != NULL);
37:     return head->data;
38: }
39:
40: // Returns a reference to last element
41: T& back(void) {
42:     assert (tail != NULL);
43:     return tail->data;
44: }
45:
46: // Returns count of elements of list
47: int size(void) {
48:     return count;
49: }
50:
51: // Returns whether or not list contains any elements
52: bool empty(void) {
53:     return count == 0;
    }
```

Үнсіздік конструкторы деректердің үш элементін head, tail және count меншіктеуі тиіс. Екі нұсқауыш head және tail алдымен ешқайда нұсқамайтынын білдіру үшін нөлдік нұсқауыш ретінде жазылған. Count деректерінің элементтері тізімде элементтердің жоқтығын білдіру үшін 0 ретінде меншіктеледі. front and back функциясының мүшелері сілтемені тізімдегі алғашқы және соңғы элементтерге қайтарады. Соңында тізімдегі элементтердің өлшемі size қайтарады, ал empty функция мүшесі тізімде қандай да бір элементтердің бар-жоғын көрсетеді.

Inserting and Removing from the Beginning

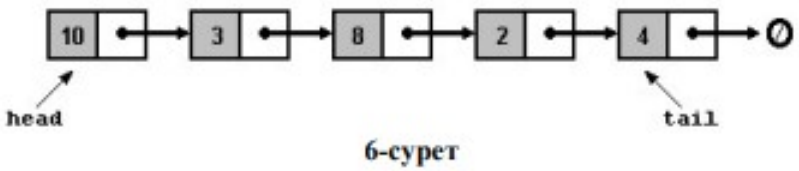
Тізімнің басына элементті орнату - салыстырмалы түрде тікелей процесс.

Келесі мәселелерден тұрады:

1. Node жаңа объект құру.
2. Жаңа түйінді тізім басымен байланыстыру.
3. Жаңа тізімді бейнелеу үшін head нұсқауышын позициялау.
4. Элементтер санын бірге арттыру.

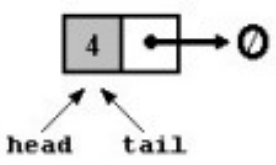
Тізім соңына нұсқауышты орнатуымыз қажет болғандықтан, бос орынға тізімге элемент енгізгенде ерекше жағдайлар туындауы мүмкін. Мұндай жағдайда біз енгізген элемент тек тізім басындағы элемент қана емес, сонымен қатар тізім соңындағы элемент те болып табылады. Мұндай жағдайды ***шекаралық шарт*** деп атайды.

Шекаралық шарт нүкте немесе басқаша өңделуі тиіс алгоритмдегі жағдай болып табылады. 6 және 7-суреттер осы амалды бейнелейді.



6-сурет

Бұл сурет жоғарғы бөлігінде нұсқауышы бар бес байланысқан элемент тізімін бейнелейді.



7-сурет

Бұл түйін (7-сурет), бір жағынан, тек бір ғана элементтен тұрады. Мұндай жағдайда жалғыз элемент тізімнің бастапқы және соңғы элементі болып табылады.