**Двойно свързан списък**

В компютърните науки, двойно свързаният списък е свързана структура от данни, състояща се от множество последователно свързани елементи наречени върхове (nodes на английски). Всеки един връх съдържа две полета наричани връзки, които са референции (указатели) към предишния и следващия връх в поредицата от върхове. Връзките на началните и крайните елементи в двойно свързания списък имат по един специален вид разклонение, служещо за прекратяване обхождането на списъка. Този специален вид разклонение обичайно е празен/кух връх (sentinel node) или “null”. Ако списъкът има само един празен връх, то той е кръгообразно свързан чрез него. Двойно свързаният списък може да бъде представен и като два отделни единично свързани списъка, съставящи се от едни и същи елементи, но в противоположен ред.



//коментар под снимката//Двойно свързан списък, чиито върхове съдържат три полета: елемент със стойност от целочислен тип (integer), връзката към следващия връх(node) и такава към предишното.

Това, което позволява обхождането на списъка в двете посоки, са двете връзки на всеки елемент, към върховете (референциите към съседните елементи). Въпреки че добавянето и премахването на елемент от двойно свързания списък изисква промяната на повече връзки, отколкото същата операция в единично свързания, операциите са по - опростени и потенциално по - ефикасен (за елементите различни от крайните), защото по време на обхождане няма нужда да се взима под внимание връзката към предишното разклонение и няма нужда да обхождаме списъка, за да намерим връзката, която искаме да променим.

Тази концепция е и в основата на техиката за запаметяване в мнемоничната (статия уики) свързана система (наричана още “свързан метод“–) /\*ПРЕПРАТКИ/\*

Номенклатура и имплементация

Първият и последният елемент на двойно свързания списък, наричани съответно “head” (глава) и “tail” (опашка), могат да бъдат достигнати незабавно (т.е. без обхождане на списъка). Те позволяват обхождането на списъка от началото или края. Например обхождане на листа от началото към края, или от края към началото, за намиране на елемент с конкретна стойност. След като намерим конкретен елемент с дадена стойност, той може да се използва като начало за ново обхождане на списъка в двете посоки - към началото на списъка или към неговия край.

Полетата представляващи връзките към съседните елементи в двойно свързаният списък често се наричат “next” и “previous” или “forward” и “backward” съответно предходен и следващ. Указателите, които държат съответните полета най- често са представени като “pointer” (референция), но могат също така да бъдат и адресни отклонения или индекси в масив, в който съществуват елементите, към които се сочи.

**Basic algorithms  
Отворен свързан списък**

**record** *DoublyLinkedNode* {

prev *// A reference to the previous node*

next *// A reference to the next node*

data *// Data or a reference to data*

}

**record** *DoublyLinkedList* {

*DoublyLinkedNode* firstNode *// points to first node of list*

*DoublyLinkedNode* lastNode *// points to last node of list*

}

**Обхождане на списъка.**

Обхождането на двойно свързаният списък може да се направи и в двете посоки . Посоката на обхождане може да бъде променяна многократно. Обхождането често е наричано итериране, но използването на тази терминология е неподходящо, тъй като итерацията има добре дефинирана семантика (например в математиката), която не е аналог на обхождането.

Обхождане от началото до края

node := list.firstNode

**while** node ≠ **null**

<do something with node.data>

node := node.next

Обхождане от края до началото

node := list.lastNode

**while** node ≠ **null**

<do something with node.data>

node := node.prev

**Вмъкване на елемент**  
Вмъкването на елемент преди или след конкретен елемент става чрез следните симетрични функции:

**function** insertAfter(*List* list, *Node* node, *Node* newNode)

newNode.prev := node

newNode.next := node.next

**if** node.next == **null**

list.lastNode := newNode

**else**

node.next.prev := newNode

node.next := newNode

**function** insertBefore(*List* list, *Node* node, *Node* newNode)

newNode.prev := node.prev

newNode.next := node

**if** node.prev == **null**

list.firstNode := newNode

**else**

node.prev.next := newNode

node.prev := newNode

За вмъкването на елемент в началото на празен списък ни трябва функция:

**function** insertBeginning(*List* list, *Node* newNode)

**if** list.firstNode == **null**

list.firstNode := newNode

list.lastNode := newNode

newNode.prev := null

newNode.next := null

**else**

insertBefore(list, list.firstNode, newNode)

Вмъкването на елемент в края на списъка става чрез следната функция:

**function** insertEnd(*List* list, *Node* newNode)

**if** list.lastNode == **null**

insertBeginning(list, newNode)

**else**

insertAfter(list, list.lastNode, newNode)

**Премахване на елемент**Премахването на елемент става по- лесно отколкото вмъкването, но изисква по- специално внимание, когато елементът, който искаме да премахнем, е първият (“headNode”) или последният (“lastNode”) от списъка.

**function** remove(*List list,* Node *node)*

**if** node.prev == **null**

list.firstNode := node.next

**else**

node.prev.next := node.next

**if** node.next == **null**

list.lastNode := node.prev

**else**

node.next.prev := node.prev

**Вальо:**

**// дублаж с горните редове**

**Премахване на елемент (node)**

**Премахванто на елемент(node) е по-лесно от вмъкването му, но изисква особен подход ако елемента(node) за премахване е първият(firstNode) или последният(lastnNode). :**

**function remove(List list, Node node)**

**if node.prev == null**

**list.firstNode := node.next**

**else**

**node.prev.next := node.next**

**if node.next == null**

**list.lastNode := node.prev**

**else**

**node.next.prev := node.prev**

**Нежелан ефект , от предходния пример, е довеждането до стойност null на firstNode и lastNode. Ако се вгледаме по- внимателно, можем да забележим, че няма нужда от отделни функции "removeBefore" и "removeAfter", защото в двойно свързаният списък можем просто да използваме "remove(node.prev)" или "remove(node.next)". Това също така ни гарантира, че елементът, който се отстранява съществува. Ако елемента не съществува в списъка, ще възникне грешка (“Exception”), която ще трябва да се обработи.**

**Кръгови двойно свързани списъци**

**Обхождане на списъка:**

**Ако приемем че, "someNode" е елемент в списъка (който не е празен), следващият пример ще обходи списъка използвайки "someNode" за отправна точка, като тя може да бъде който и да е елемент от списъка.**

**Напред:**

**node := someNode**

**do**

**do something with node.value**

**node := node.next**

**while node ≠ someNode**

**Назад :**

**node := someNode**

**do**

**do something with node.value**

**node := node.prev**

**while node ≠ someNode**

**Обърнете внимание, че проверката за прекратяване на обхождането се извършва, след като той се е изпълнил поне веднъж. Това е нужно в случаите, когато списъкът съдържа единствено елемента "somenode", който е и нашата отправна и крайна точка.**

**Вмъкване на елемент (node)**

**Следната функция вмъква node след зададен елемент в кръговия двойно свързан списък:**

**function insertAfter(Node node, Node newNode)**

**newNode.next := node.next**

**newNode.prev := node**

**node.next.prev := newNode**

**node.next := newNode**

**За да вмъкнем елемент преди някой друг („insertBefore”), можем отново да използваме функцията „insertAfter(node.prev, newNode)”.**

**Вмъкване на елемент в празен списък изисква използването на по сложна функция.**

**function insertEnd(List list, Node node)**

**if list.lastNode == null**

**node.prev := node**

**node.next := node**

**else**

**insertAfter(list.lastNode, node)**

**list.lastNode := node**

**За да вмъкнем елемент в началото, можем да използваме „insertAfter(list.lastNode, node)”.**

**Премахването на елемент (node), става чрез следната функция, която се съобразява с това, че елементите в списъка намаляват:**

**function remove(List list, Node node)**

**if node.next == node**

**list.lastNode := null**

**else**

**node.next.prev := node.prev**

**node.prev.next := node.next**

**if node == list.lastNode**

**list.lastNode := node.prev;**

**destroy node**

**Сложни конструкции**

**Несиметричен двойно свързан списък**

Като концепция несиметричният двойно свързан списък се намира някъде между единично свързания списък и обикновения двойно свързан списък. Той споделя някои характеристики на единично свързания (има еднопосочно обхождане) и други на двойно свързания списък (леснота на модификацията).

Несиметричният двойно свързан списък представлява списък, при който всеки указател към предишен връх сочи не към предишния връх (node), а към самия указател. Докато от това води до малка разлика за елементите (тъй като той сочи към офсет в предишния връх), то сменя head node-a (“главата“) на списъка: това позволява на първия връх да модифицира firstNode указателя лесно.

Докато даден връх е част от списъка, неговия указател не е със стойност null.

**//Трябва да намерим някъде обяснение на български, което да пресъздадем, че май нищо няма да вденем от това обяснение за вмъкването от оригиналната статия на английски... //**

**Вмъкване на елемент**

За да вмъкнем node преди друг node, трябва да насочим указателя, който сочи към предишния node на новия node към предишния съществуващ връх. След това нагласяме указателя на новия node към nextNode да сочи към съществуващия последващ node, чийто указател към prevNode го нагласяме да сочи към новия node.

**function** insertBefore(*Node* node, *Node* newNode)

**if** node.prev == **null**

**error** "The node is not in a list"

newNode.prev := node.prev

atAddress(newNode.prev) := newNode

newNode.next := node

node.prev = addressOf(newNode.next)

**function** insertAfter(*Node* node, *Node* newNode)

newNode.next := node.next

**if** newNode.next != **null**

newNode.next.prev = addressOf(newNode.next)

node.next := newNode

newNode.prev := addressOf(node.next)

**Премахване на елемент**

За да премахнем node, просто модифицираме указателя към предишен node, посочен от следващия node, независимо дали даденият node е първи в списъка.

**function** remove(*Node* node)

atAddress(node.prev) := node.next

**if** node.next != **null**

node.next.prev = node.prev

**destroy** node