Государственный Университет Молдовы

Факультет Математики и Информатики

Департамент Информатики

Лабораторная работа №3

“Динамические сструктуры данных”

Проверил: М.Кроитор

Выполнил: Журба Дарья

Группа: IA2102

Кишинев 2023

Оглавление

[Односвязный список. 2](#_Toc134800326)

[Двусвязный список 3](#_Toc134800327)

[Кольцевой список 5](#_Toc134800328)

[Стэк 7](#_Toc134800329)

[Очередь: 8](#_Toc134800330)

# Односвязный список.

**Односвязный список** - это структура данных, состоящая из узлов, каждый из которых содержит какую-то информацию и ссылку на следующий узел в списке. Первый узел называется головным (head), а последний - хвостовым (tail) узлом.

Каждый узел односвязного списка содержит два поля: поле данных (data) и ссылку на следующий узел в списке (next). Ссылка на следующий узел указывает на позицию памяти, где хранится следующий узел. Если ссылка на следующий узел равна NULL, то это означает, что данный узел является последним в списке.

Односвязный список может быть использован для реализации многих других абстрактных структур данных, таких как стек, очередь, хеш-таблицы и деревья.

Одним из преимуществ использования односвязного списка является его эффективность при вставке и удалении элементов из списка, так как для этого не нужно перемещать другие элементы, как в случае массивов. Однако, поиск элемента в односвязном списке занимает больше времени, чем в массиве, так как для этого нужно перебирать элементы по порядку.

class SveazniiSpisok

public function

\_construct(Sarr, $key) {

$list = new Spisok();

foreach (Sarr as $index =› $value) {

$list-›dobavlenie($value-›$key, $index);

}

return $list;

7

class Uzel

{

public $value;

public $n = NULL;

public function \_construct ($value)

$this-›value = $value;

}

class Spisok

{

public $current = NULL;

public function dobavlenie ($value)

$node = new Uzel ($value);

$this-›dobavlenieUzla($node, $this-›current);

public function dobavlenieUzla(Uzel $node, &subList)

{

if (is\_null ($subList))

{

$subList = $node;

}

else

{

$this-›dobavlenieUzla($node, $subList-›next);

}

return $this;

public function vivod($sublist) {

if ($sublist != NULL) {

print\_r($sublist-›value

" ");

$this-›vivod($sublist-›next);

}

}

public function naititlement ($sublist, $item) {

if ($sublist-›value != $item && $sublist !== NULL) { return $this-›naititlement($sublist-›next, $item);

} else {

return $sublist;

public function udalenie($item, $sublist, $next) {

if ($sublist != NULL) {

if ($next-›value == $item) {

• $sublist-›next = $next-›next;

} elseif ($next != NULL) {

if ($next-›value != Sitem) §

$this-›udalenie($item, $sublist-›next, $next-›next);

# **Двусвязный список**

**Двусвязный список** - это структура данных, аналогичная односвязному списку, но в каждом узле двусвязного списка есть две ссылки: на предыдущий и следующий узлы в списке. Это позволяет двигаться по списку в обе стороны, а не только в одном направлении, как в случае с односвязным списком.

Каждый узел двусвязного списка содержит три поля: поле данных (data), ссылку на предыдущий узел (prev) и ссылку на следующий узел (next). Если ссылка на предыдущий или следующий узел равна NULL, то это означает, что данный узел находится либо в начале, либо в конце списка соответственно.

Одним из преимуществ использования двусвязного списка является возможность быстрого перемещения в списке в обе стороны, что делает его особенно удобным для некоторых операций, таких как удаление последнего элемента списка или итерация по списку в обратном порядке. Однако, двусвязный список требует больше памяти, чем односвязный список, из-за необходимости хранения двух ссылок в каждом узле.

class DvusveazniiSpisok

public function construct (Sarr, $key) {

$list = new Spisok();

foreach (Sarr as $index =› $value) { $list-›dobavlenie($value-›Skey);

Done

return $list;

class Uzel

public $value;

public $n = NULL;

public $p = NULL;

public function \_construct ($value = NULL)

{

$this-›value = $value;

class Spisok §

public $head = NULL;

public function dobavlenie (Svalue)

Suzel = new Uzel ($value);

$this-›dobavlenieUzla(Suzel);

public function dobavlenieUzla(Uzel Suzel)

if (is\_null ($this-›head)) { $this-›head = $uzel;

} else {

$temp = $current = $this-›head;

while (lis\_null (Scurrent-›next)) {

$temp = $current;

$current = $current-›next;

}

Suzel-›prev = $temp;

Scurrent-›next = Suzel;

public function udalenie($index)

$temp = $current = $this-›head;

$i = 0;

if (Sindex < 0) $index = 0;

if (Sindex == 0) {

$current-›next-›prev = NULL;

$this-›head = $current-›next;

} else {

while (lis\_null(Scurrent)) {

if ($i == $index) {

break;

$temp = $current;

$current = current-›next;

}

if (is \_null ($current-›next))

$current-›next = new Uzel();

$temp-›next = $current-›next;

scurrent-›next-›orev-=stemo:

Done

public function vivod()

{

Scurrent = $this-›head;

while (Scurrent-›next != NULL) { print\_r(Scurrent-›value $current = $current-›next;

}

print\_r($current-›value . "In");

public function posik($value)

{

if (is \_null (Sthis-›head)) { return NULL;

] else {

$current = $this-›head;

while ($current-›next != NULL) {

if (Scurrent-›value == $value) {

return $current;

$current = $current-›next;

if (Scurrent-›value return $current;

== Svalue) {

return NULL;

}

# **Кольцевой список**

**Кольцевой список** - это структура данных, которая представляет собой связный список, в котором последний элемент списка ссылается не на NULL, а на первый элемент, образуя тем самым замкнутый цикл.

Каждый узел в кольцевом списке содержит два поля: поле данных (data) и ссылку на следующий узел в списке (next). В отличие от обычного связного списка, последний узел в кольцевом списке ссылается не на NULL, а на первый узел в списке. Таким образом, при обходе списка можно начинать с любого узла и продолжать обход до тех пор, пока не вернемся к начальному узлу.

Одним из преимуществ использования кольцевого списка является возможность бесконечной итерации по списку, так как нет явного конца списка, как в случае с обычным связным списком. Это может быть полезно в некоторых алгоритмах, таких как круговая очередь или реализация циклического буфера.

Однако, кольцевой список может быть более сложным в реализации, чем обычный связный список, и может привести к бесконечному циклу, если не были предприняты соответствующие меры предосторожности при итерации по списку. Кроме того, кольцевой список требует больше памяти, чем обычный связный список, из-за необходимости хранения ссылки на первый узел списка в последнем узле.

class KolicevoiSpisok

public function construct (Sarr, $key) {

$list = new Spisok();

foreach (Sarr as $index => $value) {

$list-›dobaviti ($value->$key, $index);

return $list;

class Uzel

public $value;

public $next = NULL;

public function \_\_construct ($value)

$this-›value = $value;

class Spisok {

public $head = NULL;

public function dobaviti($value)

$node = new Uzel ($value);

$this-›dobavitiUzel($node);

public function dobavitiuzel (Uzel $node)

$current = $this-›head;

$node-›next = $node;

if (!is\_null ($current)) {

while (Scurrent-›next != $this-›head) { $current = $current-›next;

Scurrent-›next = $node;

$node-›next = $this-›head;

} else {

$this-›head = $node;

public function vivod()

$current = $this-›head;

while (Scurrent-›next != $this-›head)

print\_r($current-›value.'

Scurrent = $current -›next;

print\_r (Scurrent-›value. "In");

public function udaliti (Sindex)

$temp = $current = $this-›head;

if (Scurrent-›next == $current) { $this-›head = NULL;

} else

$i = 0;

if (Sindex < 0) $index = 0;

(Sindex == 0) {

while ($current-›next != $this-›head) { $current = $current-›next;

Scurrent-›next = $this-›head-›next;

$this-›head - $this-›head-›next;

} else {

while ($current-›next !- $this-›head) {

$temp = Scurrent;

$current = $current -›next;

if ($i -= $index) {

break;

7

$temp-›next = $current-›next;

public function poisk($value)

if (is \_null ($this-›head))

- return NULL;

$current = $this-›head;

while (Scurrent-›next != $this-›head) { if (Scurrent-›value == $value) {

return scurrent:

}

$current = $current-›next;

if (Scurrent-›value == $value) {

return $current;

return NULL

# Стэк

**Стэк** - это абстрактная структура данных, которая используется для хранения и организации данных в соответствии с принципом "последним пришел - первым ушел" (англ. last-in, first-out, LIFO).

Основные операции, которые можно выполнить со стеком, это добавление нового элемента на вершину стека (push) и удаление элемента с вершины стека (pop). При добавлении нового элемента он помещается на вершину стека, при удалении элемента он извлекается с вершины стека.

Стек можно реализовать на основе массива или связного списка. В случае реализации на основе массива стек имеет фиксированный размер и может приводить к переполнению, если количество элементов превышает максимальный размер стека. В случае реализации на основе связного списка, стек может иметь произвольный размер, но потребляет больше памяти, так как каждый элемент списка должен содержать ссылку на следующий элемент.

Стек используется во многих алгоритмах и программах, например, в компиляторах, интерпретаторах, алгоритмах поиска и сортировки данных, и во многих других областях.

class Stack {

protected $queue;

protected $limit;

public function

\_construct($limit = 10, $initial = array()) {

$this-›queue = $initial;

$this-›limit $limit;

public function push ($item) {

if (count ($this-›queue) < $this-›limit) { array\_unshift($this-›queue, Sitem);

public function pop() {

return array shift($this-›queue);

public function top() {

return current ($this-›queue);

구

public function out () {

$temp

= [1;

while ($this-›queue) {

$element = $this-›pop();

print\_r ($element);

Stemp[] = $element;

public function search($item, $key) {

$temp = New Stack(50);

while (Sthis-›queue) §

$element = $this-›pop();

$temp-›push ($element);

if (Selement-›$key == $item) { while (Stemp-›queue) {

$this-›push($temp-›pop ());

- unset ($temp) ;

return $element;

while ($temp-›queue) {

$this-›push ($temp-›pop ()) ;

unset ($temp);

return false;

public function delete($item, $key) {

$temp = New Stack(50);

while ($this-›queue) {

Selement = $this->pop);

if (Selement-›$key == $item) {

break;}

$temp-›push (Selement);

}

while (Stemp-›queue)

$this-›push (Stemp-›pop());}

unset (Stemp);

}

}

# Очередь:

**Очередь -** это абстрактная структура данных, которая используется для хранения и организации данных в соответствии с принципом "первым пришел - первым ушел" (англ. first-in, first-out, FIFO).

Основные операции, которые можно выполнить со очередью, это добавление нового элемента в конец очереди (enqueue) и удаление элемента из начала очереди (dequeue). При добавлении нового элемента он помещается в конец очереди, при удалении элемента он извлекается из начала очереди.

Очередь также можно реализовать на основе массива или связного списка. В случае реализации на основе массива, очередь имеет фиксированный размер и может приводить к переполнению, если количество элементов превышает максимальный размер очереди. В случае реализации на основе связного списка, очередь может иметь произвольный размер, но потребляет больше памяти, так как каждый элемент списка должен содержать ссылку на следующий элемент.

Очередь используется во многих алгоритмах и программах, например, в компьютерных сетях, системах управления задачами, алгоритмах поиска и сортировки данных, и во многих других областях.

class Queue

protected $queue;

protected $limit;

public function

\_construct($limit = 10, $initial = array ()) {

$this-›queue = $initial;

$this-›limit = $limit;

public function push($item) {

if (count ($this-›queue) < $this-›limit) { array\_unshift(Sthis-›queue, Sitem);

public function pop() {

return array\_pop ($this-›queue);

public function top() {

return current ($this-›queue);

구

public function out () {

$temp = [];

while ($this-›queue) {

$element - $this-›pop();

print\_r(Selement);

$temp[] = $element;

public function search(sitem, $key) {

$temp = New Queue(50);

while ($this-›queue) {

$element = $this-›pop ();

$temp->push (Selement);

if (Selement-›$key == $item) { while (Stemp-›queue) {

$this-›push($temp-›pop ());

}

unset ($temp);

return $element;

while ($temp-›queue) {

$this-›push($temp-›pop());

unset ($temp);

return false;

public function delete($item, $key) {

$temp = New Queue (50);

while ($this-›queue) {

$element = $this-›pop ();

if (Selement -›$key == $item)

break;

$temp-›push($element);

}

while ($temp-›queue) {

$this-›push ($temp-›pop ());

unset ($temp);}