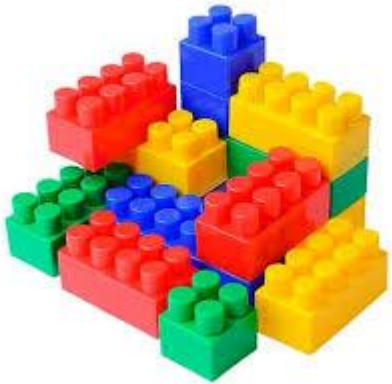
Структури даних



Структура даних – це організована сукупність даних, що виконує завдання адресації і доступу до елементів даних.

Кожна структура даних характеризується набором операцій, які з нею можна виконувати (поведінка структури даних), і стратегією розміщення елементів структури у пам’яті (представлення структури даних). Тому кожна структура даних має:

- зовнішній опис, тобто опис властивостей структури й опис набору операцій;
- внутрішнє представлення, тобто реалізацію операцій, стратегію взаєморозміщення

Будь-яку структуру даних можна описувати на трьох різних рівнях, які перераховують від більш абстрактного до конкретнішого.

- Функціональна специфікація
- Логічний опис
- Фізичне представлення

Процес поділу складної задачі на простіші називають **декомпозицією**. При цьому:

- кожна задача має один і той самий рівень розгляду та має бути розв’язана незалежно від іншої;
- одержані розв’язки можуть бути об’єднані й з їх допомогою можна розв’язати вихідну задачу.

Абстракція – відволікання від чогось несуттєвого для кращого розуміння певного аспекту досліджуваного явища, спосіб приховування деталей реалізації деякого набору функціональних можливостей, метод створення нових понять.

Лінійні структури даних

Масив і динамічний масив

Масив – це структура даних, що складається з елементів одного типу (однієї довжини), розташованих у пам’яті послідовно, і доступ до яких здійснюють обчисленням індексів.

Базові операції з масивом

- *створення масиву довжини n:* O(1) або O(n) - залежить від менеджеру пам’яті
- *зчитування/запис і-го елемента:* O(1)
- *вставка нового елемента у початок/середину/кінець:* O(n)/O(n)/O(1)
- *видалення елемента масиву з початку/середини/кінця :* O(n)/O(n)/O(1)

Для динамічного масиву:

- додавання елемента в масив

?

Зазвичай O(1)
Іноді O(n)

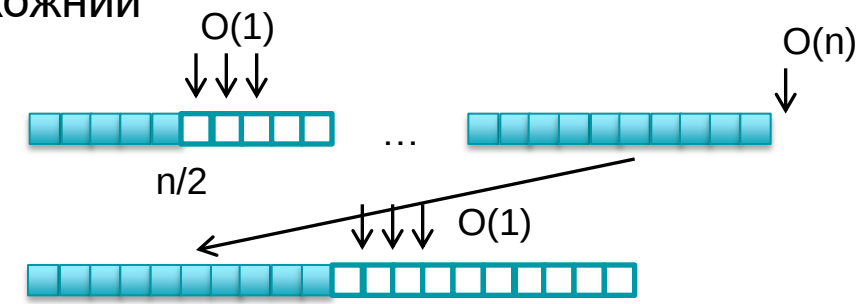
А в результаті ?

Амортизаційний аналіз

Амортизаційний аналіз — метод підрахунку часу, необхідного для виконання послідовності операцій над структурою даних. При цьому час усереднюється за всіма виконуваними операціями і аналізується середня продуктивність операцій в гіршому випадку.

Кількість операцій при додаванні елементів у динамічний масив:

1. вставити елементи з n/2 по n-1 за O(1) кожний
2. виділити нову пам’ять за O(2 * n)
3. скопіювати значения за O(n)
4. звільнити пам’ять за O(1)



Стратегії збільшення довжини масива

адитивна

length = length + a, a>1

$\approx n$

a + 2a + 3a +…+ ka =

= a(1+2+3+…k)=

=a*k*(k+1)/2

O(n^2)

мультиплікативна

length = a * length, a>1

$\approx n$

1 + a + a^2 +…+ a^k =

= (a^{k+1}-1)/(a-1)

O(n)

Алгоритм бінарного пошуку (Binary Search)

1. Знаходимо середній елемент в діапазоні пошуку (на першому кроці діапазоном пошуку є весь массив, left=0, right=n-1) mid = (left + right)/2 ;
2. Середній елемент mid порівнюємо з шуканим key, результатом цього порівняння буде один з трьох випадків:

key<mid. Крайньою правою границею області пошуку стає елемент, що стоїть перед середнім (right ← mid-1);

key>mid. Крайньою лівою границею області пошуку стає наступний за середнім елемент (left ← mid+1);

key=mid. Значення середнього та шуканого елементів співпадають, отже елемент знайдений, робота алгоритму завершується.

3. Якщо для перевірки не залишилося жодного елемента, то алгоритм завершується, інакше виконується перехід до пункту 1.

Бінарный пошук (приклад)

a - left, b - right, c - mid
key = 16

крок \ i	1	2	3	4	5	6	7	8	9	a, b, c
1	1	4	9	16	25	36	49	64	81	a=1 b=9 c=5
2	1	4	9	16	25	36	49	64	81	a=1 b=4 c=2
3	1	4	(9)	16	25	36	49	64	81	a=3 b=4 c=3
4	1	4	9	(16)	25	36	49	64	81	a=4 b=4 c=4

Переваги застосування масивів:

• легкість обчислення адреси елемента за його індексом та доступу до всіх елементів;

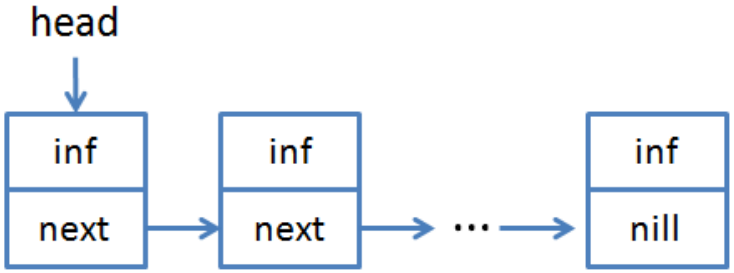
• малий розмір елементів: вони складаються тільки з інформаційного поля.

Недоліки застосування масивів:

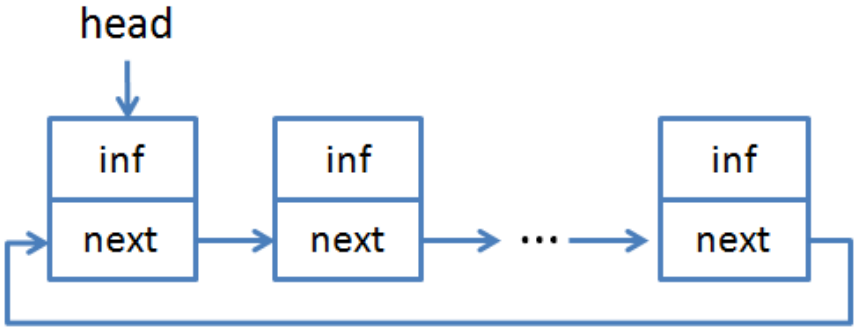
• складність додавання та вилучення елементів у початок та середину масиву;

• для статичного масиву – обмеженість та відсутність динаміки, для динамічного – додаткові накладні витрати на підтримку динамічних властивостей.

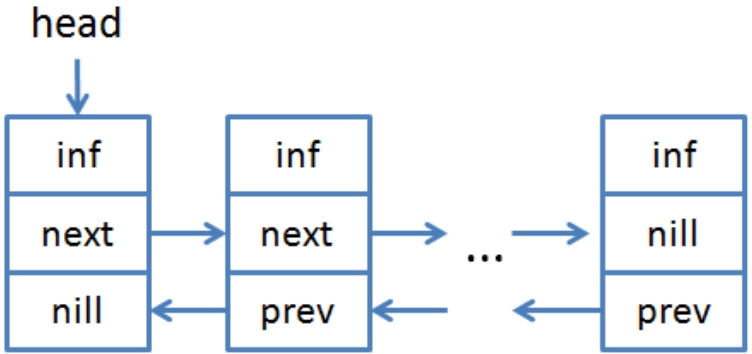
Лінійний односпрямований (однозв'язний) список



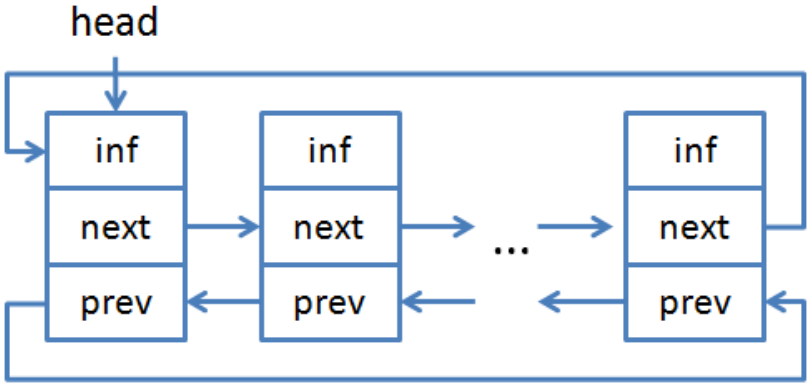
Циклічний однонаправлений список



Лінійний двонаправлений (двобічно зв’язаний) список

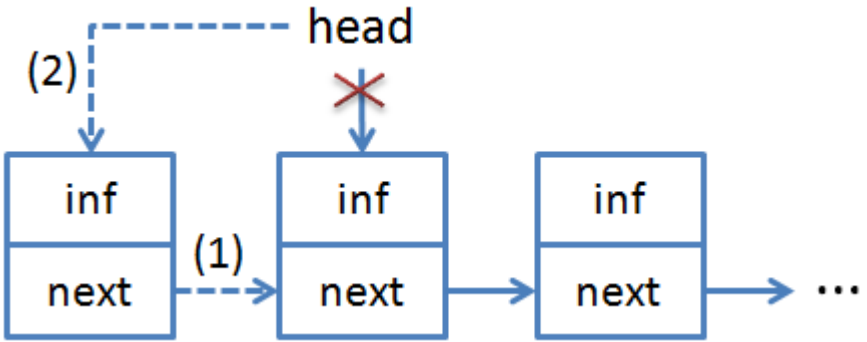


Циклічний двонаправлений список



Операції зі зв'язними списками

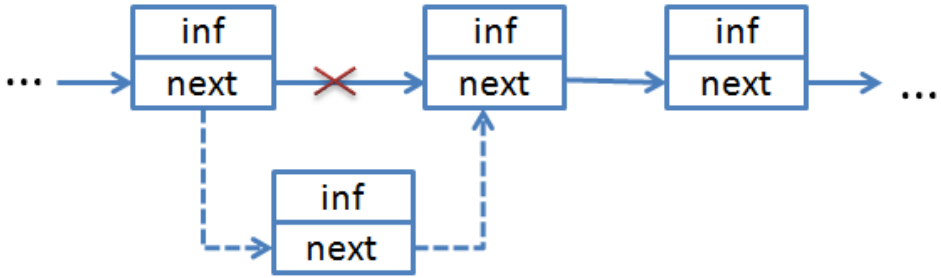
- Вставка елемента в початок списку. Складність O(1).



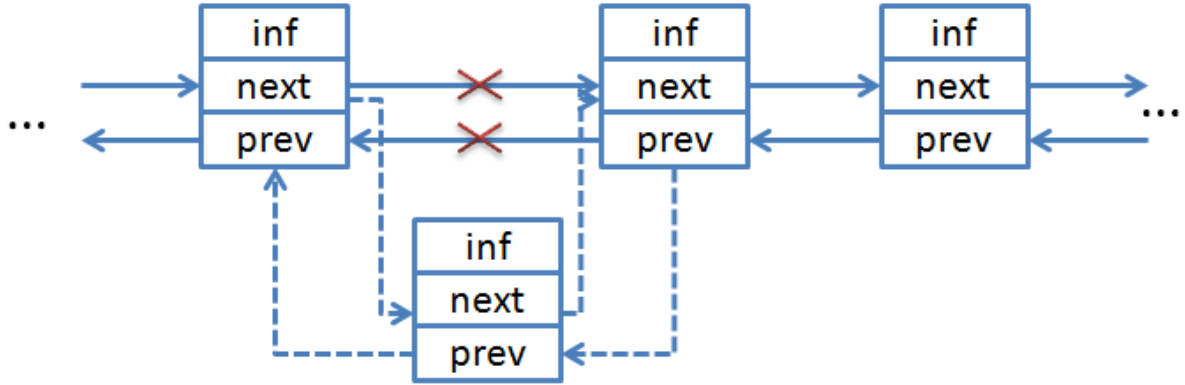
- Вставка елемента в кінець списку. Складність O(n).

Операції зі зв'язними списками

- Вставка елемента в позицію курсора. Складність O(1).



однозв'язний список



двобічно зв’язаний список

Операції зі зв'язними списками

- *Перемістити курсор до першого елемента* Складність O(1)
- *Перемістити курсор до останнього елемента* Складність O(n)
- *Перемістити курсор на крок вправо* Складність O(1)
- *Перемістити курсор на крок вліво:*
  - *у випадку однозв'язного списка* Складність O(n)
  - *у випадку двобічно зв’язаного списка* Складність O(1)
- *Пошук у списку* Складність O(n)

Переваги зв'язного представлення даних:

• розмір структури обмежується тільки доступним об'ємом машинної пам'яті;

• при зміні логічної послідовності елементів структури потрібно не переміщення даних в пам'яті, а тільки корекція покажчиків.

Основні недоліки зв'язного представлення даних:

• робота з покажчиками вимагає, як правило, більш високої кваліфікації від програміста;

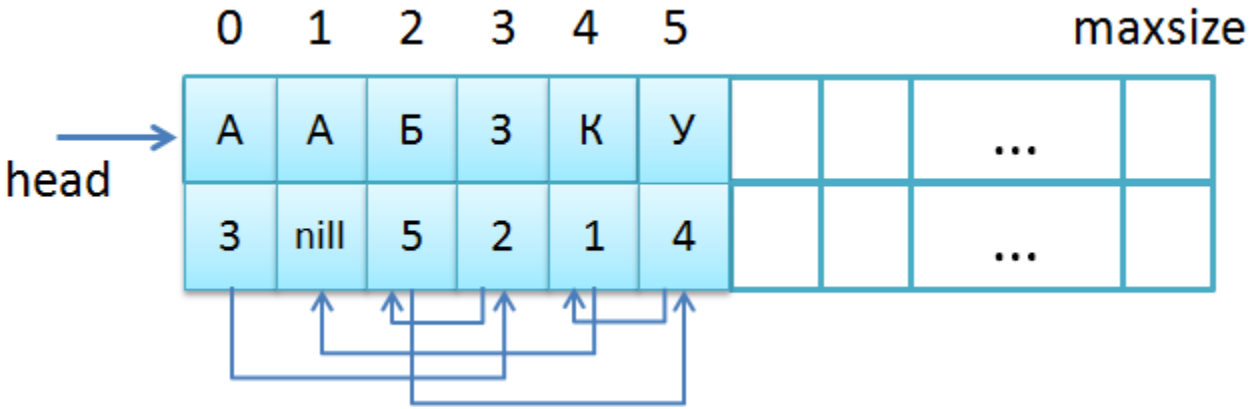
• на поля покажчиків витрачається додаткова пам'ять;

• доступ до елементів зв'язної структури може бути менш ефективним за часом.

Абстрактні типи даних (АТД)

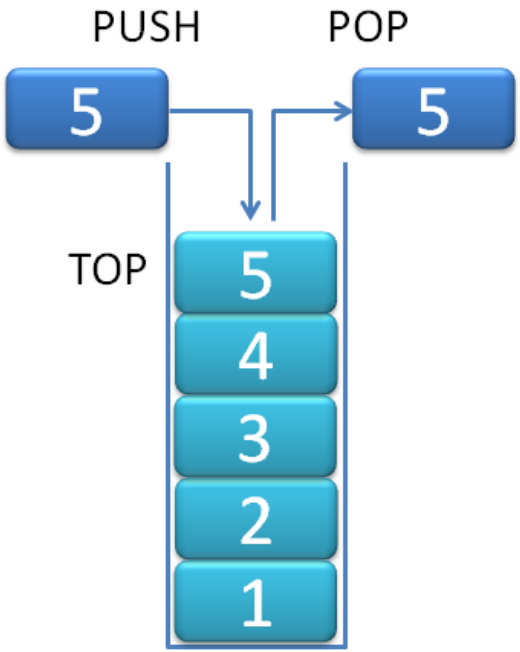
Абстрактний тип даних (АТД) - це тип даних, який надає для роботи з елементами цього типу певний набір функцій, а також можливість створювати елементи цього типу за допомогою спеціальних функцій. Вся внутрішня структура такого типу прихована від розробника програмного забезпечення - в цьому і полягає суть абстракції. АТД визначає набір функцій, незалежних від конкретної реалізації типу, для оперування його значеннями.

Реалізація абстрактного списку на базі масиву



Стек

Стек - лінійна структура даних, в якій доступ до елементів організований за принципом LIFO (англ. Last In - First Out, «останнім прийшов - першим вийшов»).



Основні операції зі стеком:
- додавання елемента (Push);
- видалення елемента (Pop);
- читання верхнього елементу (Top).

Додаткові операції зі стеком:
- перевірка стека на наявність елементів;
- підрахунок кількості елементів;
- пошук елемента і т. п.

Стек в реальному житті

Реалізація стека на основі масиву

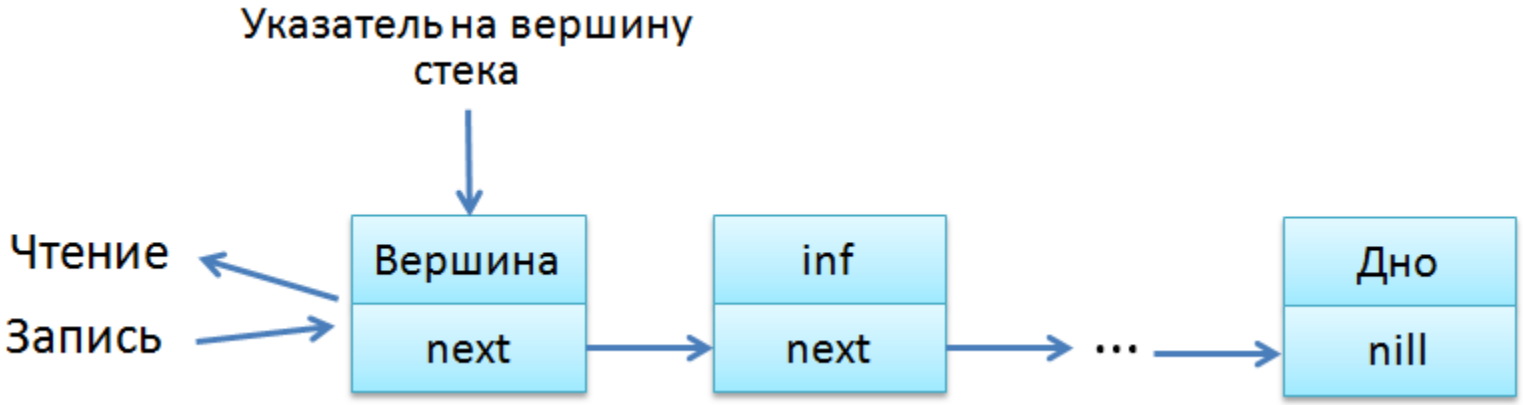
Реалізація стека на основі масиву

Елементи стека зберігаються в масиві так: елемент на дні стека розташовується на початку масиву. Індекс елемента на вершині стека зберігається в спеціальній змінній, яку зазвичай називають покажчиком стека (Stack Pointer або просто SP). Коли стек порожній, покажчик стека містить значення мінус одиниця.

При додаванні елемента покажчик стека спочатку збільшується на одиницю, потім в чарунку масиву з індексом, що міститься в покажчику стека, записується елемент.

Під час вилучення елемента зі стека вміст чарунки масиву з індексом, що міститься в покажчику стека, повертається (може не повертатися, в залежності від реалізації) в якості результату операції, потім покажчик стека зменшується на одиницю.

Реалізація стека на основі зв'язного списку



Приклад використання стека

Правильна дужкова послідовність (ПДП):

1.Порожня послідовність – ПДП;

2.Якщо А – ПДП, то (А), [А], {А} – також ПДП;

3.Якщо А, В – ПДП, то АВ – також ПДП.

[( ) ( )] – правильна дужкова послідовність;

[( ]) ( ] – неправильна дужкова послідовність.

Перевірка правильності дужкової послідовності:

1.Створюємо стек.

2.Якщо зустрічається відкриваюча дужка, то поміщаємо її в стек.

3.Якщо зустрічається закриваюча дужка, то беремо елемент зі стека і порівнюємо типи.

4.В кінці дужкової послідовності стек повинен бути порожній.

Черга

Черга - лінійна структура даних, в якій доступ до елементів організований за принципом FIFO (англ. First In - First Out, «першим прийшов - першим вийшов»).

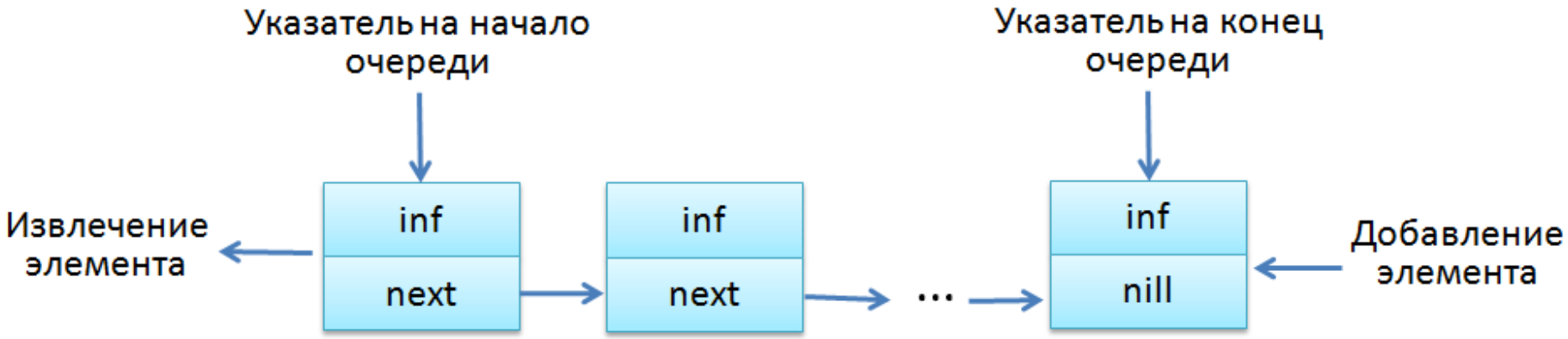
Основні операції над чергою:

додавання елемента (Enqueue);
видалення елемента (Dequeue);
зчитування першого елемента черги.

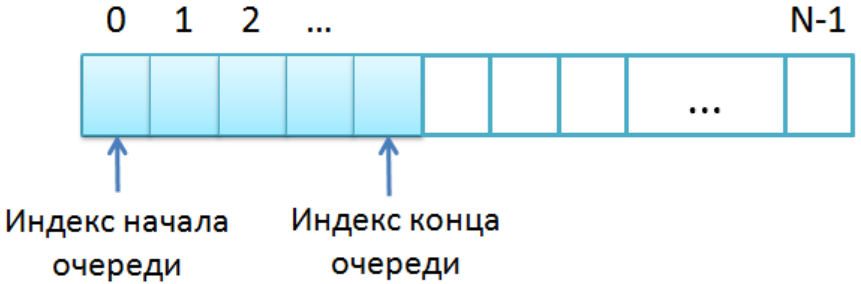
Додаткові операції над чергою:

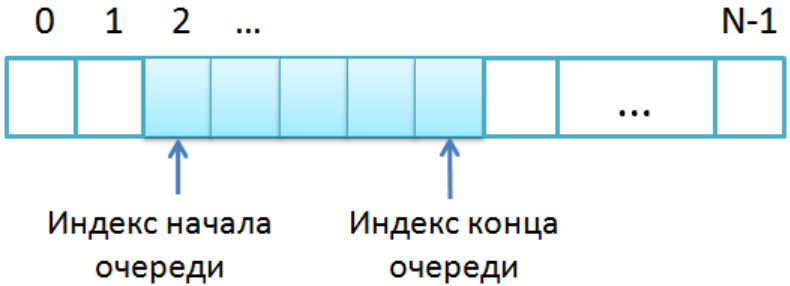
перевірка черги на наявність елементів;
підрахунок кількості елементів;
пошук і т. п.

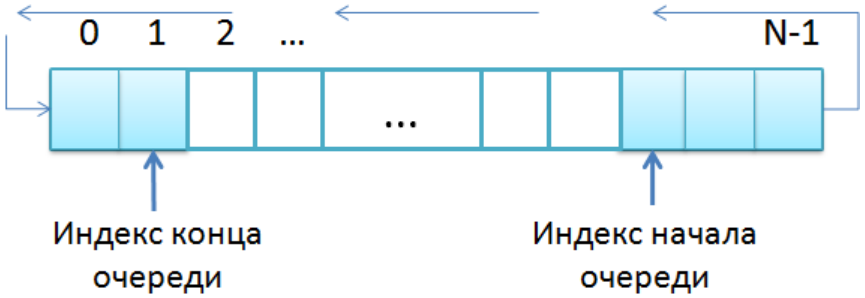
Реалізація черги на основі зв'язного списку



Реалізація черги на основі масиву







Реалізація черги на основі масиву

Крім масиву, при реалізація черги зберігаються змінні: індекс початку черги, індекс кінця черги, число елементів черги. Елементи черги містяться у масиві від індексу початку до індексу кінця.

При додаванні нового елементу в кінець черги індекс кінця спершу збільшується на одиницю, потім новий елемент записується в чарунку масиву з цим індексом. Аналогічно, при видаленні елементу з початку черги вміст чарунки масиву з індексом початку черги запам'ятовується як результат операції, потім індекс початку черги збільшується на одиницю.

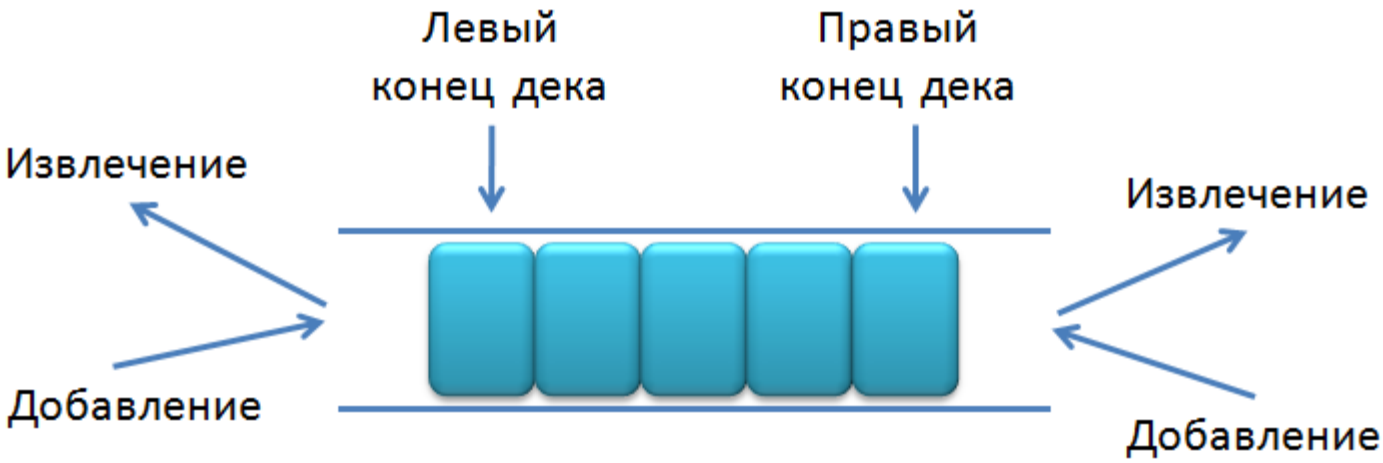
Як індекс початку черги, так і індекс кінця при роботі рухаються зліва направо. Ключова ідея реалізації черги полягає в тому, що масив “зациклюється в кільце”. Вважається, що за останнім елементом масиву слідує його перший елемент.

Реалізація черги на основі двох стеків

```
Процедура enqueue(x):
        S1.push(x)
Функція dequeue():
        якщо S2 пуст:
        якщо S1 пуст:
              повідомити про помилку: черга пуста
        доки S1 не пустий:
                S2.push(S1.pop())
        повернути S2.pop()
```

Дек

Дек (від англ. Deque - double ended queue; черга з двома кінцями) - лінійна структура даних, в якій операції включення та виключення елементів можуть виконуватися як з одного, так і з іншого кінця послідовності.



Основні операції над деком:

додавання елемента в лівий кінець дека;
додавання елемента в правий кінець дека;
видалення елемента з лівого кінця дека;
видалення елемента з правого кінця дека;
читання першого елемента зліва;
читання першого елемента справа.

Додаткові операції над деком:

перевірка дека на наявність елементів;
підрахунок кількості елементів;
пошук і т. п.

Окремі варіанти дека: реєстр та архів

– дек з обмеженим входом (реєстр) - з кінця дека можна тільки видаляти елементи;

– дек з обмеженим виходом (архів) - в кінець дека можна тільки додавати елементи.