Структури та абстрактні типи даних

Ніколайчук Михайло,
6-й курс, ФВЕ

Зміст

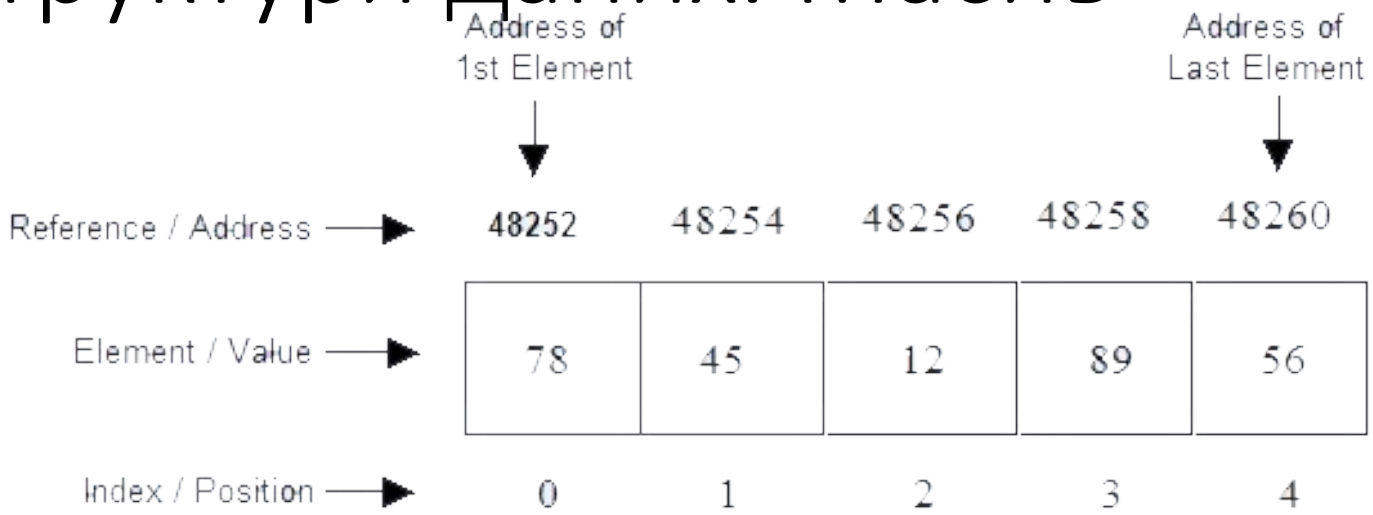
- Атомічні типи даних
- Базові структури даних
  - Масив
  - Зв’язний список
- Що таке ADT (abstract data type)
- Різновиди ADT:
  - Стек, черга
  - Дерева
  - Хеш-таблиця
  - Асоціативні масиви
  - …

Атомічні типи даних

Тип даних	Значення
Boolean	True, False
Integer	-inf, …, -2, -1, 0, 1, 2, …, +inf
Float	-inf, …, -1., -0.5, 0., 0.5, …, +inf
Char	‘a’, ‘b’, ‘c’, …, ‘&’, ‘*’, …

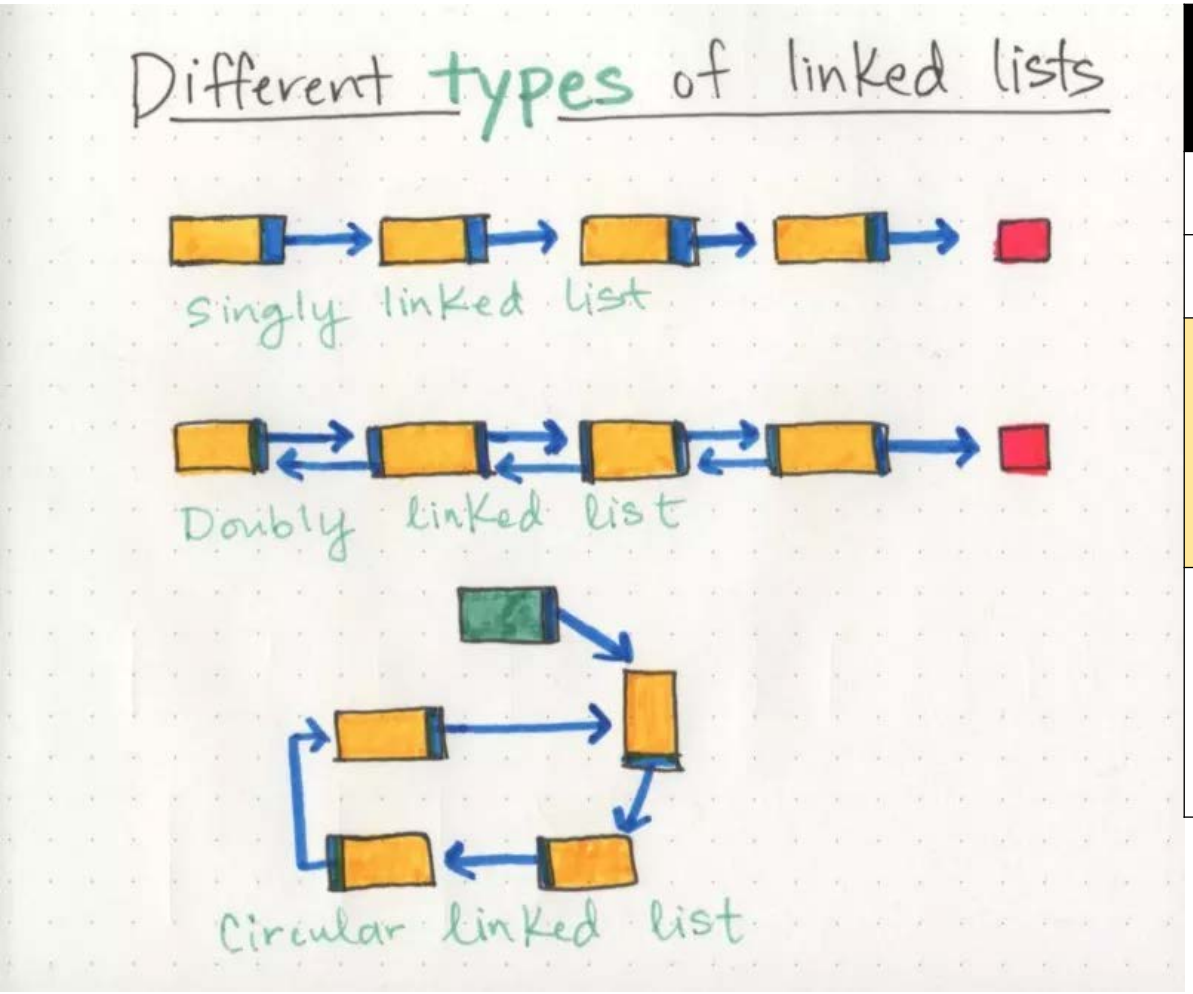
Ми всі добре знаємо що це і як воно працює.

Базові структури даних. Масив



Операція		Алгоритмічна складність
Доступ		O(1)
Пошук		O(N)
Вставка	Початок	O(N)
	Кінець	O(1) амортизовано
	Середина	O(N)
Видалення	Початок	O(N)
	Кінець	O(1)
	Середина	O(N)

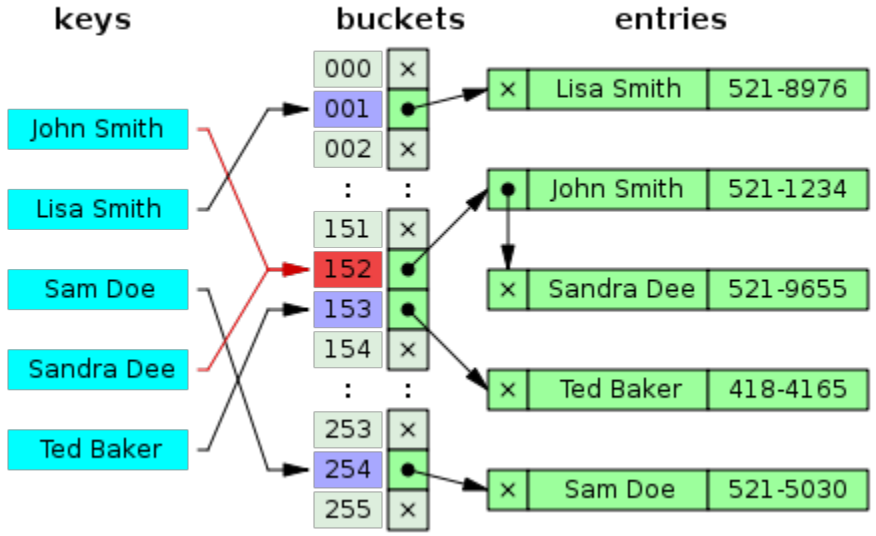
Базові структури даних. Список

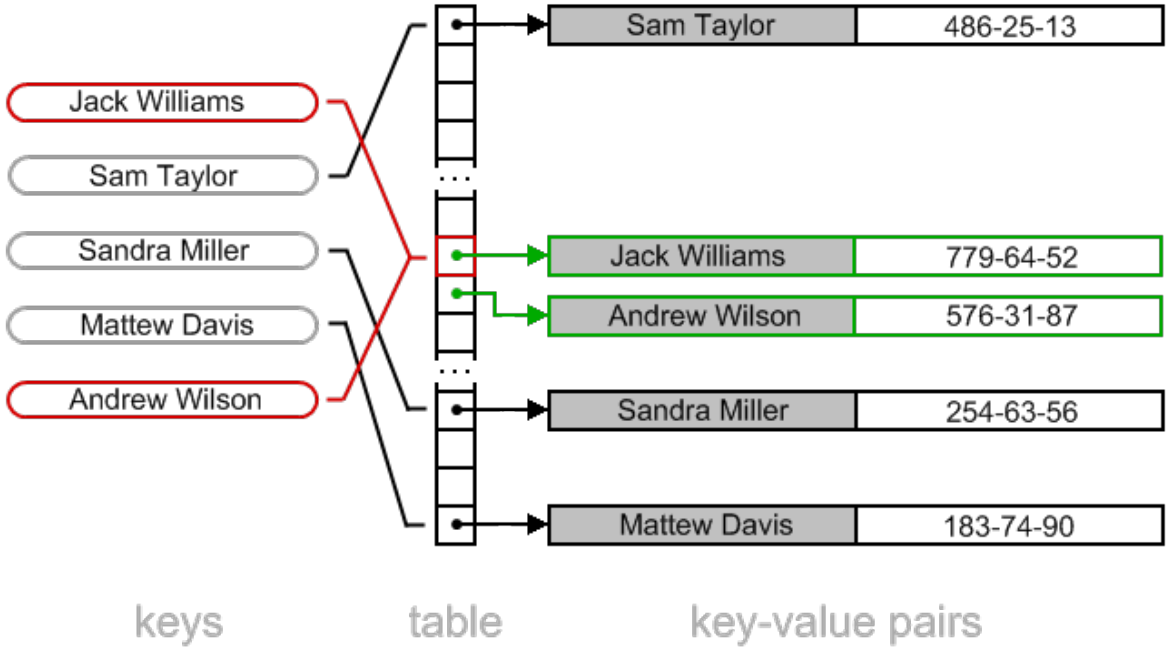


Операція		Алгоритмічна складність
Доступ		O(N)
Пошук		O(N)
Вставка	Початок	O(1)
	Кінець	O(1)*
	Середина	O(N)
Видалення	Початок	O(1)
	Кінець	O(1)*
	Середина	O(N)

*якщо кінець відомий, інакше O(N)

Data structure. Hash-table

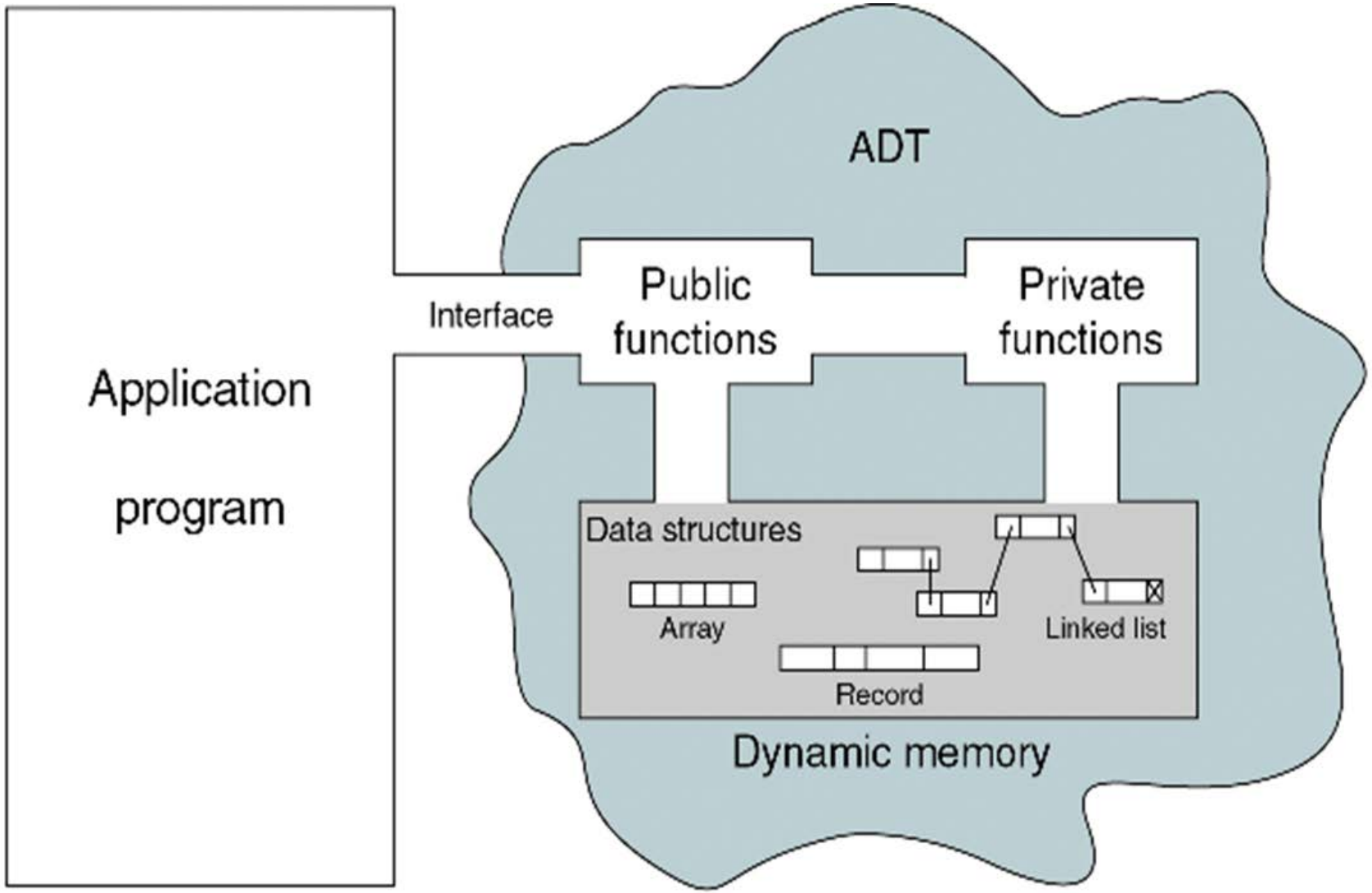




Операція	Алгоритмічна складність			
	Linked list		Probing	
	Mean	Worst	Mean	Worst
Доступ	O(1)	O(N)	O(1)	O(N)
Пошук	O(1)	O(N)	O(1)	O(N)
Вставка	O(1)	O(N)*	O(1)	O(N)
Видалення	O(1)	O(N)	O(1)	O(N)

*при дозволених дублікатах можна отримати O(1)

Abstract Data Type

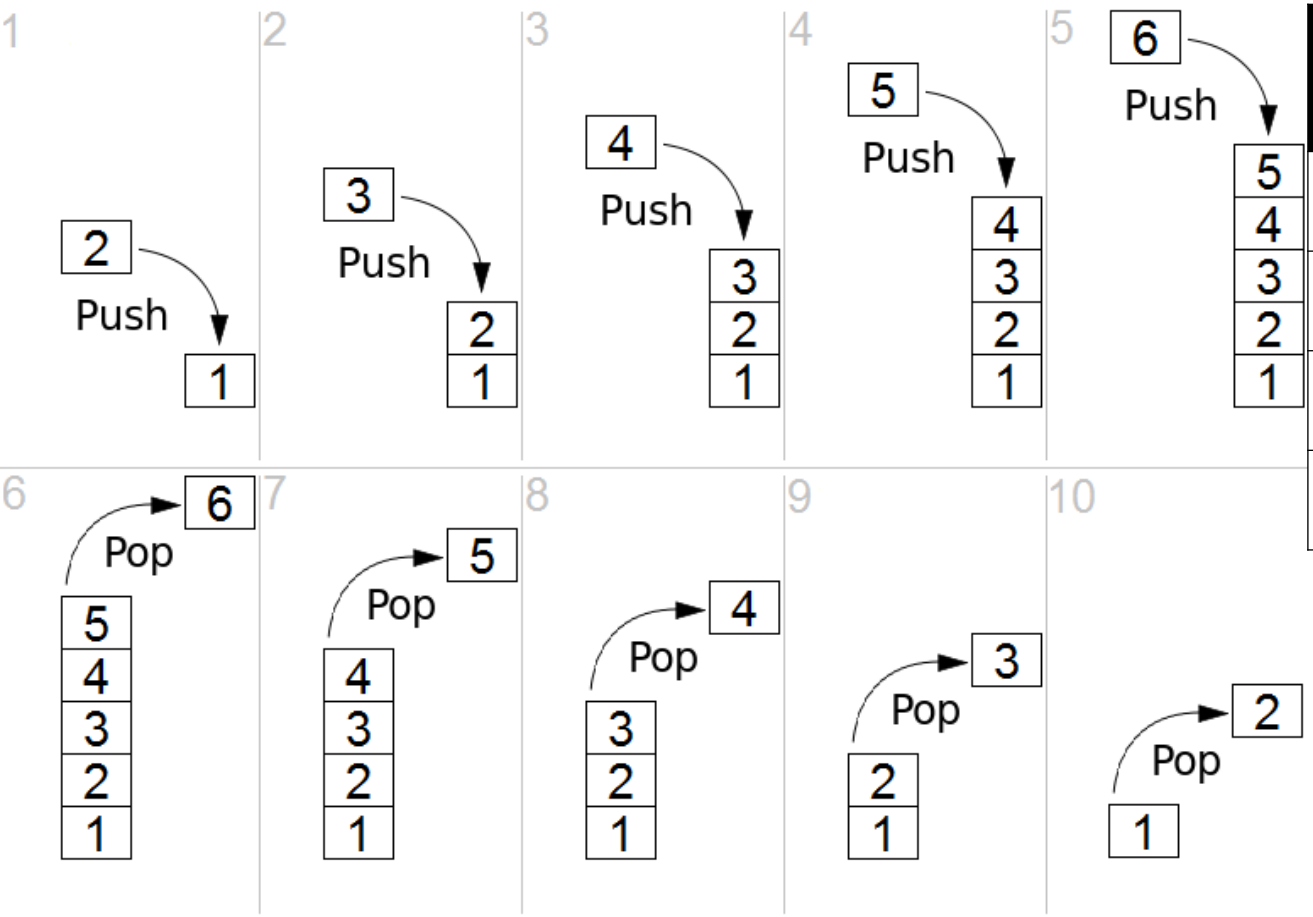


Abstract Data Type Model

У нас є:

- Набір методів для роботи з даними;
- Гарантії на складність по часу/пам’яті

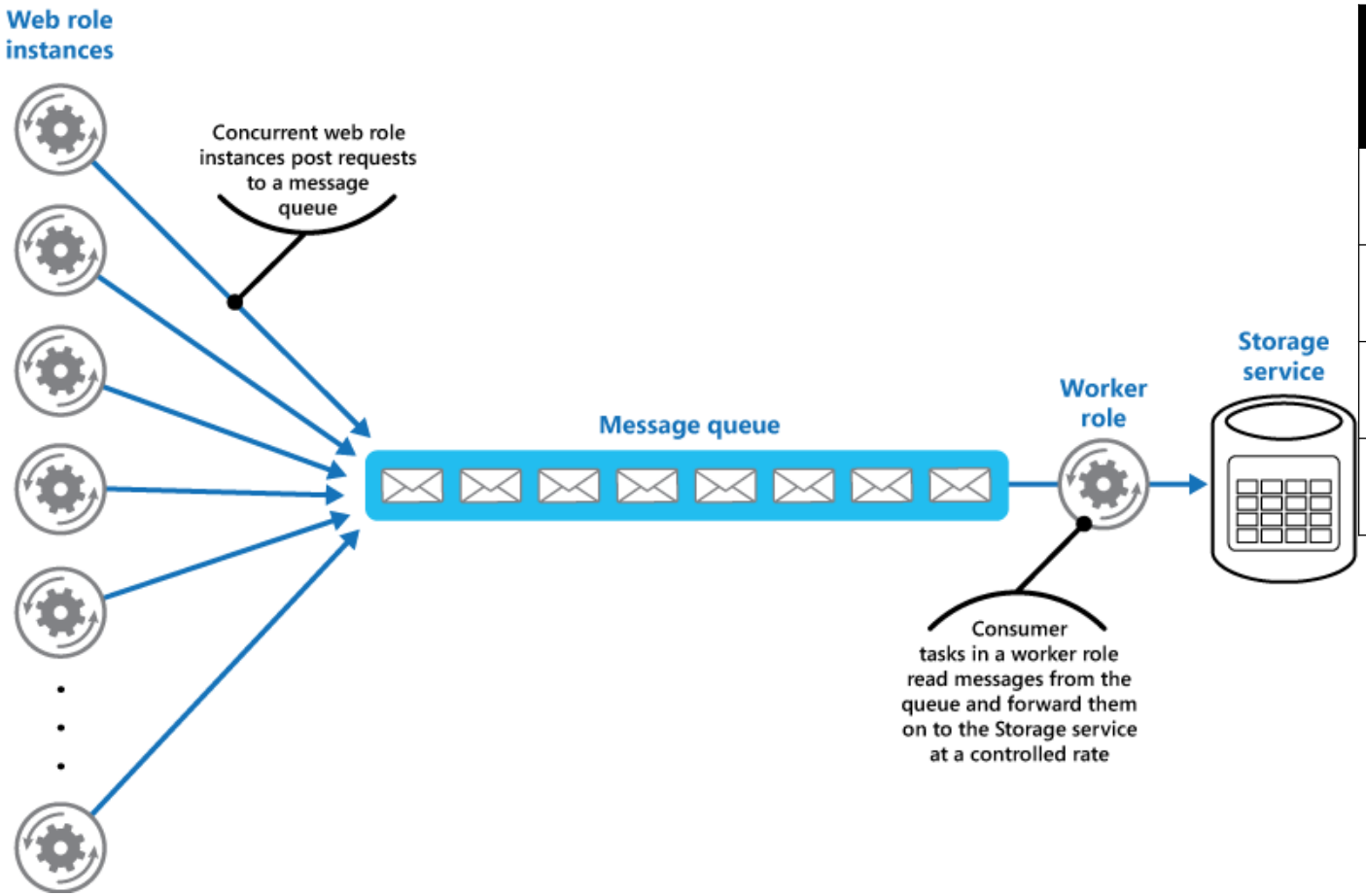
Abstract Data Type. Stack



Операція	Алгоритмічна складність
Доступ	O(N)
Пошук	O(N)
Вставка	O(1)
Видалення	O(1)

Зазвичай використовується на низькому рівні. Інколи корисний у деяких високорівневних алгоритмах (наприклад, перевід виразу з інфіксної в постфіксну нотацію)

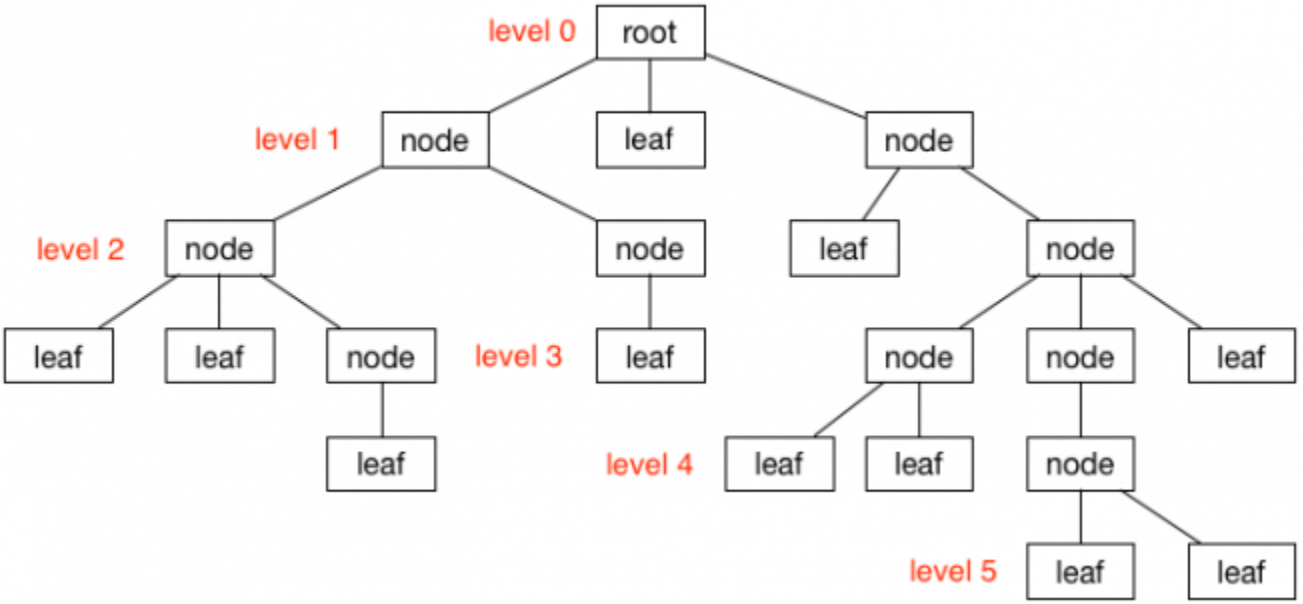
Abstract Data Type. Queue



Операція	Алгоритмічна складність
Доступ	O(N)
Пошук	O(N)
Вставка	O(1)
Видалення	O(1)

Корисний, коли у вас завдання для обробки у піку приходять швидше, ніж машина здатна виконати.

Abstract Data Type. Trees



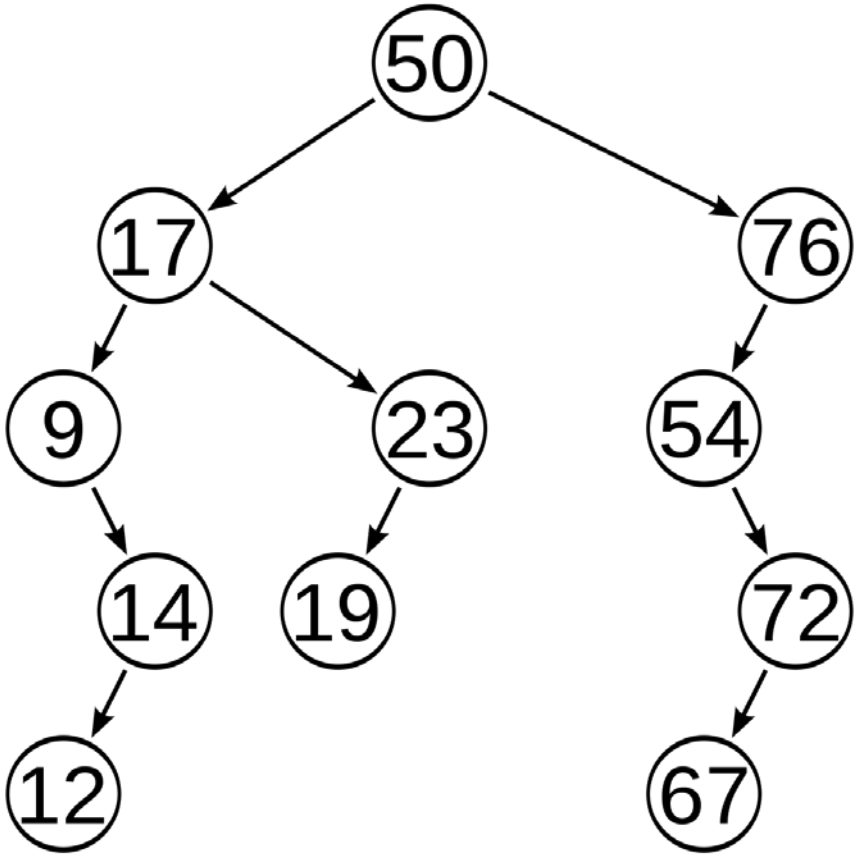
Дерево – зв’язний граф без циклів.

На практиці один із найпоширеніших типів даних. Приклади дерев:

- Червоно-чорне дерево
- AVL дерево
- АСД

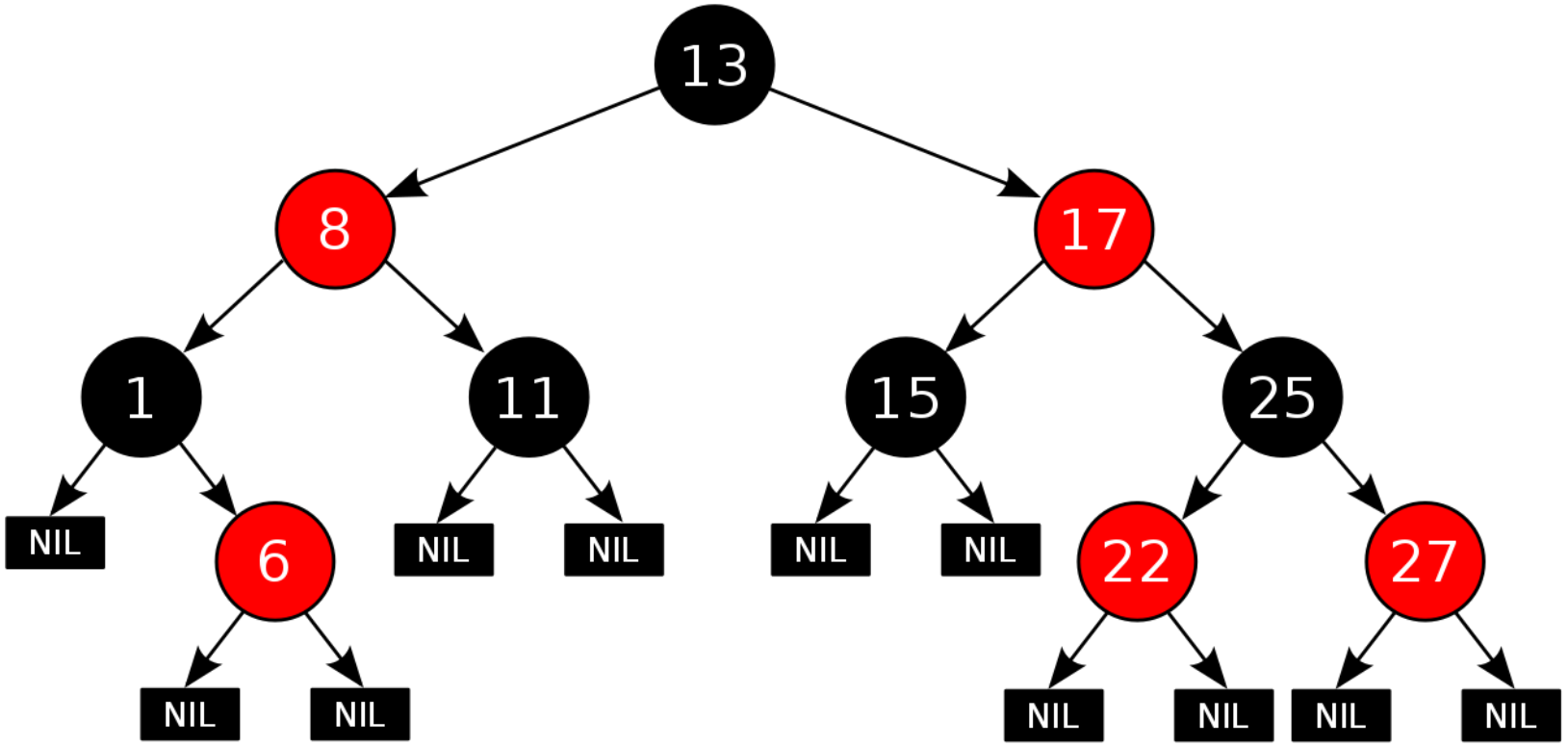
Для зберігання даних використовуються самобалансуючі дерева. Це дерева, які при заповненні намагаються зберегти мінімальну висоту.

Abstract Data Type. Binary Search Tree



Операція	Алгоритмічна складність	
	Середня	Найгірша
Доступ	O(lnN)	O(N)
Пошук	O(lnN)	O(N)
Вставка	O(lnN)	O(N)
Видалення	O(lnN)	O(N)

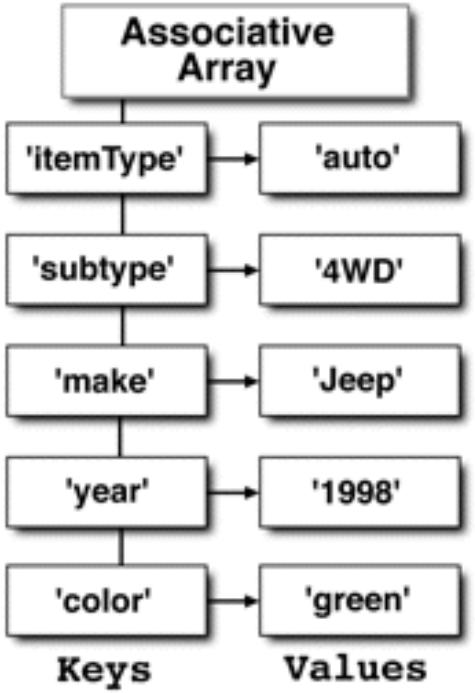
Abstract Data Type. Red-Black Tree

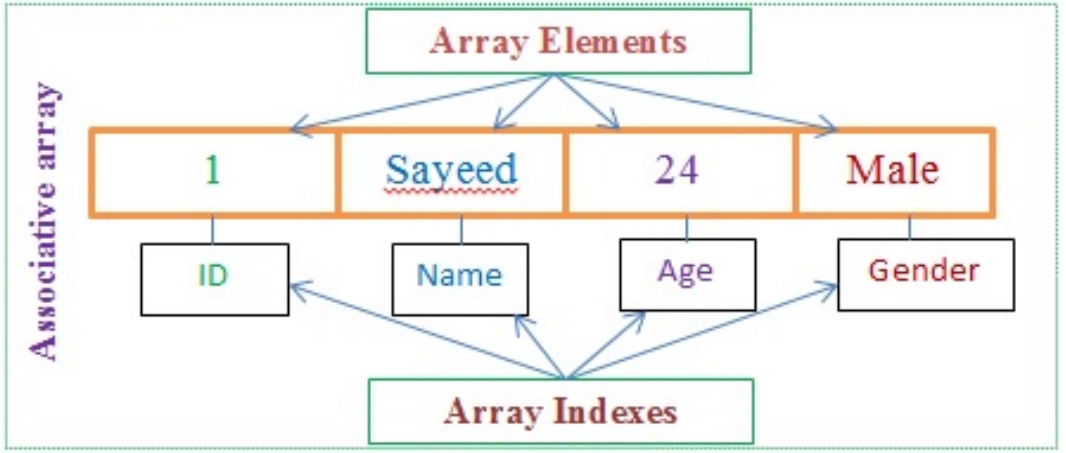


Кожна нода додатково містить один біт (умовно «колір»), що використовується для рівномірного заповнення дерева.

Операція	Алгоритмічна складність
	Середня/Найгірша
Доступ	O(lnN)
Пошук	O(lnN)
Вставка	O(lnN)
Видалення	O(lnN)

Abstract Data Type. Associative array.





Underlying data structure	Lookup		Insertion		Deletion		Ordered
	average	worst case	average	worst case	average	worst case	
Hash table	O(1)	O(*n*)	O(1)	O(*n*)	O(1)	O(*n*)	No
Self-balancing binary search tree	O(log *n*)	O(log *n*)	O(log *n*)	O(log *n*)	O(log *n*)	O(log *n*)	Yes

Приклад стеку

```
void reverse(string & x)
{
   stack<char> s;
   const int n = x.length();
   //Put characters from x onto the stack
      for(int i=0; i<n; ++i)
      s.push(x[i]);
   //take characters off of stack and put them back into x
      for(int i=0; !s.empty(); ++i, s.pop())
      x[i]=s.top();
}
```

Приклад черги

```
#include <iostream>
#include <queue>

using namespace std;

int main ()
{
   queue <string> names; /* Declare a queue */
   names.push ("Danny"); /* Add some values to the queue */
   names.push ("Kayleigh"); /* Much like vectors */
   names.push ("Joe"); /* This basically does the same thing */

   cout << "There are currently " << names.size () << " people in the queue"
<< endl
        << "The person at the front of the queue is " << names.front () << endl
        << "The person at the back of the queue is " << names.back () << endl
<< endl;

   cout << names.front () << " has been served!" << endl;
   names.pop ();
   cout << "There are currently " << names.size () << " people in the queue"
<< endl
        << "The person at the front of the queue is " << names.front () << endl
        << names.back () << " is still at the back!" << endl;
   cin.get ();
   return EXIT_SUCCESS;
}
```

Приклад асоціативного масиву

```
#include <iostream>
#include <map>

bool fncomp (char lhs, char rhs) {return lhs<rhs;}

struct classcomp {
  bool operator() (const char& lhs, const char&
rhs) const
  {return lhs<rhs;}
};

int main ()
{
  std::map<char,int> first;

  first['a']=10;
  first['b']=30;
  first['c']=50;
  first['d']=70;

  std::map<char,int> second
(first.begin(),first.end());

  std::map<char,int> third (second);

  std::map<char,int,classcomp> fourth;
// class as Compare

  bool(*fn_pt)(char,char) = fncomp;
  std::map<char,int,bool(*)(char,char)> fifth
(fn_pt); // function pointer as Compare

  return 0;
}
```

Висновки

Ми розглянули основні структури даних. Що не було розглянуто:

- Безліч дерев:
  - Купа (частковий випадок бінарного дерева)
  - Не бінарні пошукові дерева
  - Application specific дерева (такі як AST)
- Графи
- Специфічні списки та масиви (такі як списки з пропусками чи розріджені матриці)
- Ряд інших