CS 171: Introduction to Computer Science II

Stacks and Queues

Li Xiong
Today

• Quick note on running book code
• Stacks
  – Operations
  – Implementations
Book code

- All book code are available at
  ~cs171000/share/book
- To go to the directory from a lab machine
  `cd ~cs171000/share/book`
- To run a program from the directory (classpath include stdlib.jar)
  `java -cp .:* Selection < tiny.txt`
Stacks and queues

Fundamental data types.
- Value: collection of objects.
- Operations: insert, remove, iterate, test if empty.
- Intent is clear when we insert.
- Which item do we remove?

**Stack.** Examine the item most recently added.  LIFO = "last in first out"

**Queue.** Examine the item least recently added.  FIFO = "first in first out"
Stacks

• A stack stores an array of elements but with only two main operations:
  Push: add an element to the top of the stack
  Pop: remove the top element of the stack.

• Pop always removes the last element that’s added to the stack. This is called LIFO (Last-In-First-Out).
Stacks – A Familiar Example

• A can of tennis balls
  – Imagine the entire can represents an array, and each ball is an element.
  – It only allows access to one element at a time: the last element.
  – If you remove the last element, you can then access the next-to-last element.
  – There is no way to directly access the element at the bottom.
Stacks – Another Example

- A dynamic list of tasks you perform everyday:
  - Imagine you start your day by working on task A.
  - At any time you may be interrupted by a co-worker asking you for temporary help on task B.
  - While you work on B, someone may interrupt you again for help on task C.
  - When you are done with C, you will resume working on B.
  - Then you go back to work on A.
  - Think about the sequence of tasks you perform.
Stacks – Any other examples?
Stack Examples
Stacks

• An element cannot be inserted to or accessed from the middle of the array.
• The only way you modify the elements is through the push and pop operations.

• This capability turns out to be very useful in many programming situations.
• In a computer, the stack is an essential data structure for handling program calls and returns.
Stacks

• Programmer’s tool
  – Arrays are typically used as data storage structures in apps such as a database (e.g. personal records, inventories …)
  – In contrast, stacks are often used as programmer’s tool, and are not typically used for data storage.
**Client, implementation, interface**

Separate interface and implementation.
Ex: stack, queue, bag, priority queue, symbol table, union-find, ....

**Benefits.**

- **Client can't know details of implementation** ⇒
  client has many implementation from which to choose.
- **Implementation can't know details of client needs** ⇒
  many clients can re-use the same implementation.
- **Design:** creates modular, reusable libraries.
- **Performance:** use optimized implementation where it matters.

---

*Client:* program using operations defined in interface.
*Implementation:* actual code implementing operations.
*Interface:* description of data type, basic operations.
Stack API

Warmup API. Stack of strings data type.

```java
public class StackOfStrings {
    StackOfStrings() {
        create an empty stack
    }
    void push(String s) {
        insert a new item onto stack
    }
    String pop() {
        remove and return the item most recently added
    }
    boolean isEmpty() {
        is the stack empty?
    }
    int size() {
        number of items on the stack
    }
}
```

Warmup client. Reverse sequence of strings from standard input.
Stack test client

```java
public static void main(String[] args)
{
    StackOfStrings stack = new StackOfStrings();
    while (!StdIn.isEmpty())
    {
        String item = StdIn.readString();
        if (item.equals("-")) StdOut.print(stack.pop());
        else stack.push(item);
    }
}
```

% more tobe.txt
_to be or not to - be -- that -- -- is_

% java StackOfStrings < tobe.txt
Stack test client

```java
public static void main(String[] args) {
    StackOfStrings stack = new StackOfStrings();
    while (!StdIn.isEmpty()) {
        String item = StdIn.readString();
        if (item.equals("-")) StdOut.print(stack.pop());
        else stack.push(item);
    }
}
```

```bash
% more tobe.txt
to be or not to - be -- that -- -- is
% java StackOfStrings < tobe.txt
to be not that or be
```
Array implementation of a stack.

• Use array $s[]$ to store $N$ items on stack.
• $\text{push}()$: add new item at $s[N]$.
• $\text{pop}()$: remove item from $s[N-1]$. 

```
<table>
<thead>
<tr>
<th>s[]</th>
<th>to</th>
<th>be</th>
<th>or</th>
<th>not</th>
<th>to</th>
<th>be</th>
<th>null</th>
<th>null</th>
<th>null</th>
<th>null</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>
```

$N$ capacity = 10
Stack: array implementation

```java
public class FixedCapacityStackOfStrings {
    private String[] s;
    private int N = 0;

    public FixedCapacityStackOfStrings(int capacity) {
        s = new String[capacity];
    }

    public boolean isEmpty() {
        return N == 0;
    }

    public void push(String item) {
        s[N++] = item;
    }

    public String pop() {
        return s[--N];
    }
}
```

- Use to index into array; then increment N
- A cheat (stay tuned)
- Decrement N; then use to index into array
Stack: Array implementation

• Underflow: what happens if pop from an empty stack?
  – Throw exception

• Overflow: what happens if push to a full stack?
  – Use resizing array
Stack: resizing-array implementation

**Problem.** Requiring client to provide capacity does not implement API!

**Q.** How to grow and shrink array?

**First try.**

- `push()`: increase size of array $s[]$ by 1.
- `pop()`: decrease size of array $s[]$ by 1.
Stack: resizing-array implementation

**Problem.** Requiring client to provide capacity does not implement API!

**Q.** How to grow and shrink array?

**First try.**

- `push()`: increase size of array `s[]` by 1.
- `pop()`: decrease size of array `s[]` by 1.

**Too expensive.**

- Need to copy all item to a new array.
- Inserting first $N$ items takes time proportional to $1 + 2 + \ldots + N \sim \frac{N^2}{2}$.

```
  infeasible for large N
```

**Challenge.** Ensure that array resizing happens infrequently.
Stack: resizing-array implementation

Q. How to grow array?
A. If array is full, create a new array of **twice** the size, and copy items.

```java
public ResizingArrayStackOfStrings()
{
    s = new String[1];
}

public void push(String item)
{
    if (N == s.length) resize(2 * s.length);
    s[N++] = item;
}

private void resize(int capacity)
{
    String[] copy = new String[capacity];
    for (int i = 0; i < N; i++)
        copy[i] = s[i];
    s = copy;
}
```

**Consequence.** Inserting first $N$ items takes time proportional to $N$ (not $N^2$).
Stack: Array Implementation

• What’s the cost of pushing/adding to a stack of size N?
  – Case 1: array resizing not required
  – Case 2: array resizing required
Stack: amortized cost of adding to a stack

Cost of inserting first $N$ items. $N + (2 + 4 + 8 + \ldots + N) \sim 3N$.

- 1 array accesses per push
- $k$ array accesses to double to size $k$
  (ignoring cost to create new array)

![Graph showing cost of push operations](image-url)
Stack: resizing-array implementation

Q. How to shrink array?

First try.

- **push()**: double size of array $s[]$ when array is full.
- **pop()**: halve size of array $s[]$ when array is one-half full.
Stack: resizing-array implementation

Q. How to shrink array?

First try.

• push(): double size of array $s[]$ when array is full.
• pop(): halve size of array $s[]$ when array is one-half full.

*Too expensive in worst case.*

• Consider push-pop-push-pop-... sequence when array is full.
• Each operation takes time proportional to $N$.
Stack: resizing-array implementation

Q. How to shrink array?

Efficient solution.

- **push()**: double size of array $s[]$ when array is full.
- **pop()**: halve size of array $s[]$ when array is one-quarter full.

```java
public String pop()
{
    String item = s[--N];
    s[N] = null;
    if (N > 0 && N == s.length/4) resize(s.length/2);
    return item;
}
```

Invariant. Array is between 25% and 100% full.
Stack resizing-array implementation: performance

**Amortized analysis.** Average running time per operation over a worst-case sequence of operations.

**Proposition.** Starting from an empty stack, any sequence of $M$ push and pop operations takes time proportional to $M$.

<table>
<thead>
<tr>
<th></th>
<th>best</th>
<th>worst</th>
<th>amortized</th>
</tr>
</thead>
<tbody>
<tr>
<td>construct</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>push</td>
<td>1</td>
<td>N</td>
<td>1</td>
</tr>
<tr>
<td>pop</td>
<td>1</td>
<td>N</td>
<td>1</td>
</tr>
<tr>
<td>size</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

order of growth of running time for resizing stack with N items
doubling and halving operations
Parameterized stack

We implemented: `StackOfStrings`.

We also want: `StackOfURLs, StackOfInts, StackOfVans, ...`

Attempt 1. Implement a separate stack class for each type.
• Rewriting code is tedious and error-prone.
• Maintaining cut-and-pasted code is tedious and error-prone.

@#$*! most reasonable approach until Java 1.5.
Parameterized stack

We implemented: StackOfStrings.
We also want: StackOfURLs, StackOfInts, StackOfVans, ....

Attempt 2. Implement a stack with items of type object.
- Casting is required in client.
- Casting is error-prone: run-time error if types mismatch.

```java
StackOfObjects s = new StackOfObjects();
Apple a = new Apple();
Orange b = new Orange();
s.push(a);
s.push(b);
a = (Apple) (s.pop());
```
Parameterized stack

We implemented: StackOfStrings.
We also want: StackOfURLs, StackOfInts, StackOfVans, ....

Attempt 3. Java generics.
• Avoid casting in client.
• Discover type mismatch errors at compile-time instead of run-time.

Guiding principles. Welcome compile-time errors; avoid run-time errors.
Generic stack: array implementation

```java
public class FixedCapacityStackOfStrings {
    private String[] s;
    private int N = 0;

    public StackOfStrings(int capacity) {
        s = new String[capacity];
    }

    public boolean isEmpty() {
        return N == 0;
    }

    public void push(String item) {
        s[N++] = item;
    }

    public String pop() {
        return s[--N];
    }
}
```

```java
public class FixedCapacityStack<Item> {
    private Item[] s;
    private int N = 0;

    public FixedCapacityStack(int capacity) {
        s = new Item[capacity];
    }

    public boolean isEmpty() {
        return N == 0;
    }

    public void push(Item item) {
        s[N++] = item;
    }

    public Item pop() {
        return s[--N];
    }
}
```

@#$^! generic array creation not allowed in Java
Generic stack: array implementation

```java
public class FixedCapacityStackOfStrings {
    private String[] s;
    private int N = 0;

    public ..StackOfStrings(int capacity) {
        s = new String[capacity];
    }

    public boolean isEmpty() {
        return N == 0;
    }

    public void push(String item) {
        s[N++] = item;
    }

    public String pop() {
        return s[--N];
    }
}

the way it is

```java
public class FixedCapacityStack<@Item> {
    private Item[] s;
    private int N = 0;

    public FixedCapacityStack(int capacity) {
        s = (Item[]) new Object[capacity];
    }

    public boolean isEmpty() {
        return N == 0;
    }

    public void push(Item item) {
        s[N++] = item;
    }

    public Item pop() {
        return s[--N];
    }
}

the ugly cast
```
Generic data types: autoboxing

Q. What to do about primitive types?

Wrapper type.
- Each primitive type has a wrapper object type.
- Ex: `Integer` is wrapper type for `int`.

Autoboxing. Automatic cast between a primitive type and its wrapper.

Syntactic sugar. Behind-the-scenes casting.

```java
Stack<Integer> s = new Stack<Integer>();
s.push(17); // s.push(new Integer(17));
int a = s.pop(); // int a = s.pop().intValue();
```

Bottom line. Client code can use generic stack for any type of data.
Stack: Resizing Array Implementation

• ResizingArrayStack.java
Today

• Quick note on running book code
• Stacks

• Coming up
  – Applications using stack
  – Queues