CS213/293 Data Structure and Algorithms 2023

Lecture 3: Stack

Instructor: Ashutosh Gupta

IITB India

Compile date: 2023-08-09

Topic 3.1

Stack

Stack

Definition 3.1
Stack is a container where elements are added and deleted according to the last-in-first-out (LIFO) order.

- Addition is called pushing
- Deleting is called popping

Example 3.1

- *Stack of papers in a copier*
- *Undo-redo features in editors*
- *Back button on Browser*

Interface of stack

Reference: https://en.cppreference.com/w/cpp/container/stack

Stack supports four interface methods

- `stack<T> s` : allocates new stack s
- `s.push(e)` : Pushes the given element e to the top of the stack.
- `s.pop()` : Removes the top element from the stack.
- `s.top()` : accesses the top element of the stack.

Some support functions

- `s.empty()` : checks whether the stack is empty
- `s.size()` : returns the number of elements

Axioms of stack

Let `s1` and `s` be stacks.

- `Assume(s1 == s); s.push(e); s.pop(); Assert(s1==s);`
- `s.push(e); Assert(s.top()==e);`

`Assume(s1 == s)` means that we assume that the content of s1 and s are the same.
`Assert(s1 == s)` means that we check that the content of s1 and s are the same.

Exercise: action on the empty stack

Exercise 3.1

Let `s` be an empty stack in C++.

- What happens when we run `s.top()`?
- What happens when we run `s.pop()`?

Ask ChatGPT.

Commentary: Answer: `s.top()` will cause a segmentation fault. `s.pop()` will not cause any error and exit without any effect.

Topic 3.2

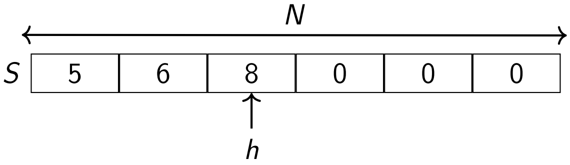
Implementing stack

Array-based stack

Let us look at a simplified array-based implementation of an array of integers.

The stack consists of three variables.

- N specifies the currently available space in the stack
- S is the integer array of size N
- h is the position of the head of the stack



Implementing stack

```
class arrayStack {
  int   N = 2;      // Capacity
  int*  S = NULL; // pointer to array
  int   h = -1;    // Current head of the stack
public:
  arrayStack() { S = (int*)malloc(sizeof(int)*N ); }
  int  size()  { return h+1;  }
  bool empty() { return h<0;  }
  int  top()   { return S[h]; } // On empty stack what happens?
  void push(int e) {
    if( size() == N ) expand(); // Expand capacity of the stack
    S[++h] = e;
  }
  void pop() { if( !empty() ) h--; }
```

Commentary: The behavior of the above implementation may not match with the behavior of the C++ stack library. To ensure segmentation fault in top() when the stack is empty one may use the following code. `if( empty() ) return *(int*)0; else return S[t];`

Implementing stack (expanding when full)

```
private:
  void expand() {
    int new_size = N*2; // We observed the growth in our lab!!
    int* tmp = (int*) malloc( sizeof(int)*new_size );//New array
    for( unsigned i =0; i < N; i++ ) {// copy from the old array
      tmp[i] = S[i];
    }
    free(S);        // Release old memory
    S = tmp;        // Update local fields
    N = new_size; //
  }
};
```

Efficiency

All operations are performed in $O(1)$ if there is no expansion to stack capacity.

What is the cost of expansion?

Topic 3.3

Why exponential growth strategy?

Growth strategy

Let us consider two possible choices for growth.

- Constant growth: `new_size = N + c` (for some fixed constant c)
- Exponential growth: `new_size = 2*N`

Which of the above two is better?

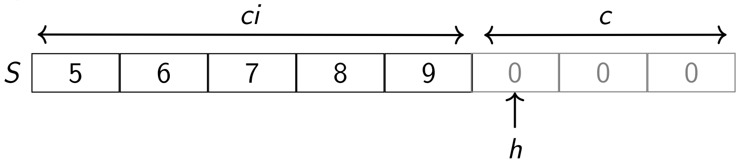
Analysis of constant growth

Let us suppose initially $N = 0$ and there are n consecutive pushes.

After every cth push, there will be an expansion operation.

Therefore, the expansion operation at $(ci + 1)$th push will

- allocate memory of size $c(i + 1)$
- copy ci integers



Cost of ith expansion: $c(2i + 1)$.

Commentary: We are assuming that allocating memory of size k costs k time, which may be more efficient in practice. Bulk memory copy can also be sped up by vector instructions.

Analysis of constant growth(2)

For n pushes, there will be n/c expansions.

The total cost of expansions:

$$c(1 + 3 + ... + (2\frac{n}{c} + 1)) = c(n/c)^2 \in O(n^2)$$

Non-linear cost!

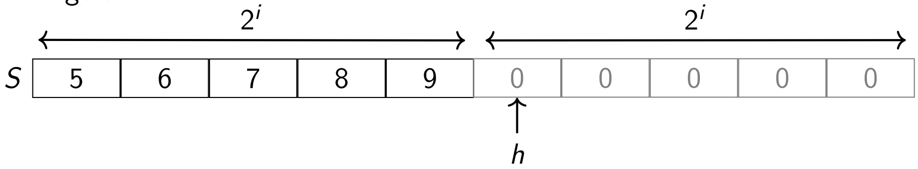
Analysis of exponential growth

Let us suppose initially $N = 1$ and there are $n = 2^r$ consecutive pushes.

The expansion operations will only occur at $2^i + 1$th push, where $i \in [0, r - 1]$.

The expansion operation at $2^i + 1$th push will

- allocate memory of size 2^{i+1}
- copy 2^i integers



Cost of the expansion: $3 * 2^i$.

Analysis of exponential growth(2)

For 2^r pushes, the last expansion would be at $2^{r-1} + 1$.

The total cost of expansions:

$$3(2^0 + ... + 2^{r-1}) = 3 * (2^r - 1) = 3 * (n - 1)$$

Linear cost! The average cost of push remains $O(1)$.

Exercise 3.2

Why double? Why not triple? Why not 1.5 times? Is there a trade-off?

Topic 3.4

Applications of stack

Stacks are everywhere

Stack is a foundational data structure.

It shows up in a vast range of algorithms.

Example: matching parentheses

Problem:
Given an input text check if it has matching parentheses.

Examples:

- "{a[sic]tik}"✓
- "{a[sic}tik}"✗

```
bool parenMatch(string text ) {
  std::stack<char> s;
  for(char c : text ) {
    if( c == '{' or c == '[' ) s.push(c);
    if( c == '}' or c == ']' ) {
      if( s.empty()       ) return false;
      if( c-s.top() != 2 ) return false;
      s.pop();
    }
  }
  if( s.empty() ) return true;
  return false;
}
```

Topic 3.5

Problems

Use of stack

Exercise 3.3

The span of a stock's price on ith day is the maximum number of consecutive days (up to ith day) the price of the stock has been less than or equal to its price on day i.

Example: for the price sequence 2 4 6 3 5 7 *of a stack, the span of prices is* 1 2 3 1 2 6*.*

Give a linear-time algorithm that computes s_i for a given price series.

Flipping Dosa

Exercise 3.4

There is a stack of dosas on a tava, of distinct radii. We want to serve the dosas of increasing radii. Only two operations are allowed: (i) serve the top dosa, (ii) insert a spatula (flat spoon) in the middle, say after the first k, hold up this partial stack and flip it upside-down and put it back. Design a data structure to represent the tava, input a given tava, and to produce an output in sorted order. What is the time complexity of your algorithm?
This is also related to the train-shunting problem.

Commentary: Milind notes!

Exponential growth

Exercise 3.5

a. Do the analysis of performance of exponential growth if the growth factor is three instead of two? Does it give us better or worse performance than doubling policy?
b, Can we do the similar analysis for growth factor 1.5?

End of Lecture 3