Amortized Analysis of vector::push_back

Andreas Klappenecker

Vector

In C++, a vector is a sequence of elements that can be accessed by an index, but - unlike an array - it does not have a fixed size.

```
vector<int> = v; // start with an empty vector

v.push_back(1); // v = [1] and capacity = 1

v.push_back(2); // v = [1,2] and capacity = 2

v.push_back(3); // v = [1,2,3] and capacity = 4
```

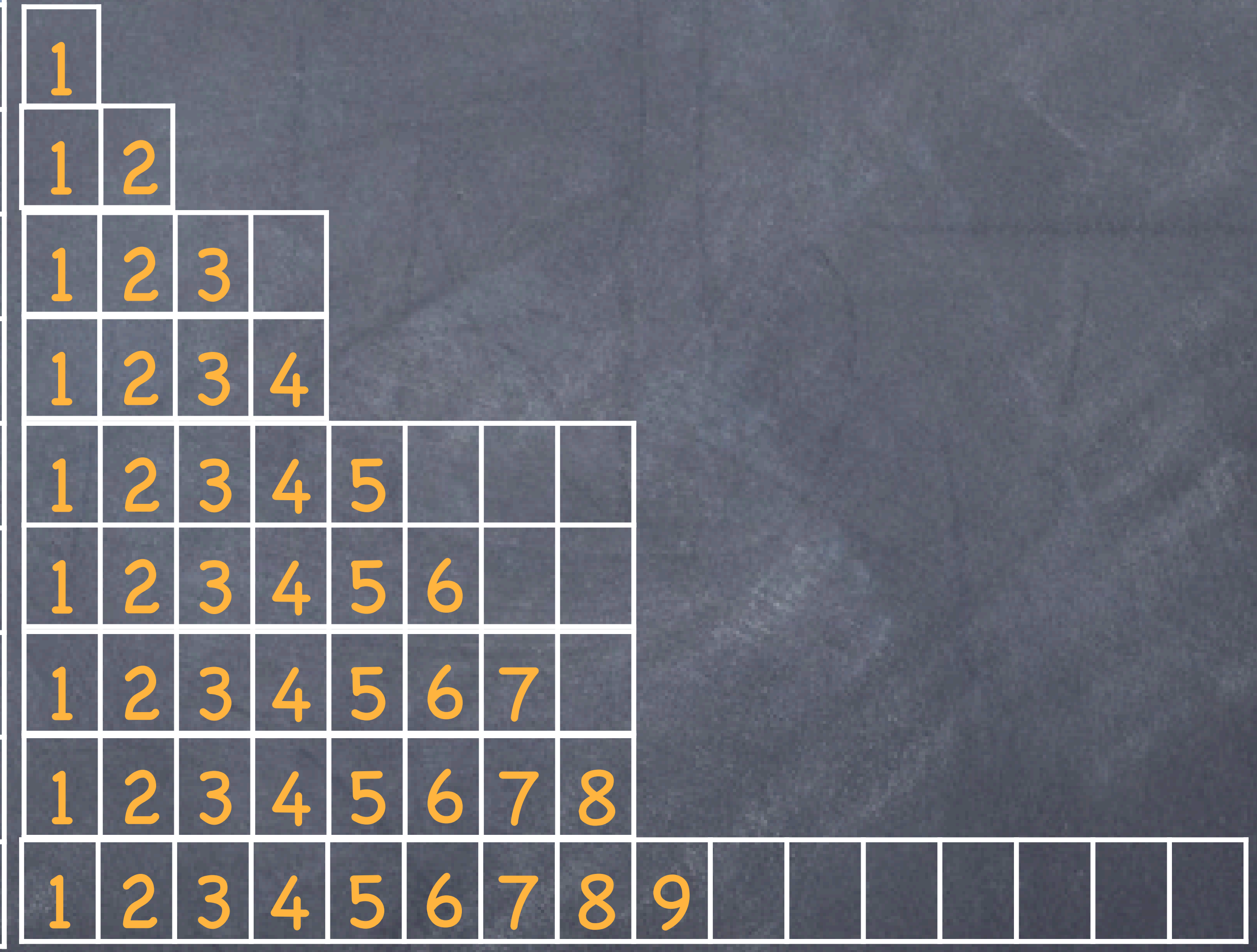
Simplified C++ Vector

```
template<class T>
void vector<T>::push_back(const T& val) {

  if (capac == 0) reserve(1);
  else if (sz==capac) reserve(2*capac); // grow
  alloc.construct(&elem[sz], val); // add val at end
  ++sz;  // increase size

}
```

```
template<class T>
void vector<T>::reserve(int newalloc) {

  if(newalloc <= capac) return;

  T* p = alloc.allocate(newalloc);
  for(int i=0; i<sz; ++i)
    alloc.construct(&p[i],elem[i]); // copy
  // deallocation omitted ...
  elem = p;
  capac = newalloc;

}
```

Costs

Operation	Capacity	Cost
push_back(1)	1	1
push_back(2)	2	1 + 1
push_back(3)	4	2 + 1
push_back(4)	4	1
push_back(5)	8	4 + 1
push_back(6)	8	1
push_back(7)	8	1
push_back(8)	8	1
push_back(9)	16	8 + 1



Aggregate Analysis

Cost for the i-th push_back

$$c_i = \begin{cases} 1 + 2^k & \text{if } i - 1 = 2^k \text{ for some } k \\ 1 & \text{otherwise} \end{cases}$$

Thus, n push_back operations cost

$$T(n) = \sum_{i=1}^{n} c_i \leq n + \sum_{i=0}^{\lfloor \lg n \rfloor} 2^i = n + 2n - 1 = 3n - 1.$$

Amortized costs: $T(n)/n = (3n-1)/n < 3$.

Accounting Analysis

Suppose we charge an amortized cost of 3.

- Adding the value at the end of the vector costs 1,
- and 2 are left over to pay for future copy operations.

If the table doubles, the stored credit pays for the move of

- an old item (in the lower half of the vector)
- the item itself (in the upper half of the vector)

Example

We assume that the lower half of the vector has used up all stored credit (which is a tiny bit too pessimistic).

Since the elements in the upper half of the vector can pay for the move of every element in the lower half, we never go into the red!

Potential Method

We can define a function Φ from the set of vectors to the real numbers by defining

Φ(v) = 2*v.size() - v.capacity()

We have

- Initially: v.capacity() = 0 and v.size() = 0.
- $c_i' = 1 + \Phi_i - \Phi_{i-1} = 1 + 2$ if i^{th} operation doesn't cause growth

Potential Method

If the i^{th} operation does cause growth, then

$capac_i = 2*capac_{i-1}$, $sz_{i-1} = capac_{i-1}$, $sz_i = capac_{i-1} + 1$

Therefore,

$c_i' = capac_{i-1}+1 + \Phi_i - \Phi_{i-1}$

$= capac_{i-1}+1 +(2*(capac_{i-1}+1)-2*capac_{i-1}) - (2*capac_{i-1} - capac_{i-1})$

$= 3$

Summary

The amortized time of vector::push_back is constant.

References

B. Stroustrup: Programming - Principles and Practice Using C++, Addison Wesley, 2009.

Cormen, Leiserson, Rivest, Stein: Introduction to Algorithms, 3rd edition, MIT press, 2009.