1092 演算法 Homewoek 2

tags: **Homework**

題目

17-2 Making binary search dynamic

只需實作insert、search及print

> Binary search of a sorted array takes logarithmic search time, but the time to insert anew element is linear in the size of the array. We can improve the time for insertion by keeping several sorted arrays.
>
> Specifically, suppose that we wish to support SEARCH and INSERT on a set of n elements. Let k = $\lceil \lg(n+1) \rceil$, and let the binary representation of n be $< n_{k-1}, n_{k-2}, \ldots, n_0 >$. We havek sorted arrays $A_0, A_1, \ldots, A_{k-1}$, where for $i = 0, 1, \ldots, k-1$, the length of array A_i is 2^i. Each array is either full or empty, depending on whether $n_i = 1$ or $n_i = 0$, respectively. The total number of elements held in all k arrays is therefore $\sum_{i=0}^{k-1} n_i 2^i = n$. Although each individual array is sorted, elements in different arrays bear no particular relationship to each other.

a. Describe how to perform the SEARCH operation for this data structure. Analyze its worst-case running time.

> The A_{k-1} isn't have relationship for each other.
> Search each array the worst-case time is $k = \lceil lg(n+1) \rceil \rightarrow O(\lg(n))$, and search each sorted array elements using binary search the worst-case time is $O(\lg(n))$, So the worst-case time of SEARCH operation is $O(\lg(n)) * O(\lg(n)) = O(\lg^2(n))$

b. Describe how to perform the INSERT operation. Analyze its worst-case and andamortized running times.

The worst-case time is $O(n)$. If now A_{k-1} need to merge to A_k, then merge two sorted array need $O(m)$, merge $\sum_{i=0}^{k} A_i = O(km) = O(n)$.

The andamortized time is $O(\lg(n))$. Merge array need 2^k, Insert m times $T = \sum_{i=0}^{k-1} \frac{m}{2^i} 2^{i+1} \leq O(mk) \leq O(m \lg(n))$ time, then each m insertion need $\frac{O(m \lg(n))}{O(m)} = O(\lg(n))$.

C++ Code

```cpp
#include <cmath>
#include <iostream>
#include <iterator>
#include <list>
#include <vector>

using namespace std;

template <typename T>
class DynamicBinarySearchArray {
private:
  typedef typename vector<T>::const_iterator cvt_iterator;
  typedef typename vector<T>::iterator vt_iterator;
  typedef typename list<vector<T>>::iterator lvt_iterator;
  list<vector<T>> data;
  size_t size;

  void merge(const vector<T> &x, vector<T> &y) const {
    y.reserve(y.size() << 1);

    cvt_iterator first1 = x.begin();
    vt_iterator first2 = y.begin();
    for (; first1 != x.end() && first2 != y.end(); ++first2)
      if (*first1 <= *first2)
        y.insert(first2, *first1++);

    y.insert(first2, first1, x.end());
  }

public:
  DynamicBinarySearchArray<T>() : size(0) {}

  void insert(const T &val) {
    lvt_iterator prev_it = data.insert(data.begin(), vector<T>{val});
    lvt_iterator now_it = next(prev_it);

    while (now_it != data.end())
      if (prev_it->size() == now_it->size()) {
        merge(*prev_it, *now_it);
        data.erase(prev_it);
      } else {
        prev_it = now_it;
        now_it = next(now_it);
      }

    ++size;
  }

  template <typename... Args>
  void insert(T first, Args... val) { insert(first), insert(val...); }

```

```
52    bool search(const T &val) const {
53      for (const vector<T> &sub_arr : data)
54        if (*lower_bound(sub_arr.begin(), sub_arr.end(), val) == val)
55          return true;
56
57      return false;
58    }
59
60    void print() { cout << *this; }
61
62    friend ostream &operator<<(std::ostream &stream, const DynamicBinarySearch
63      vector<T> result;
64      result.reserve(bsa.size);
65      for (const vector<T> &x : bsa.data)
66        bsa.merge(x, result);
67
68      cout << "{ ";
69      for (int w = 0; w < result.size(); ++w)
70        cout << result[w] << (w + 1 == result.size() ? " " : ", ");
71
72      cout << "}";
73
74      return stream;
75    }
76  };
```