Data Structures and Algorithms: Homework #5

Due on May 26, 2015 at 16:20

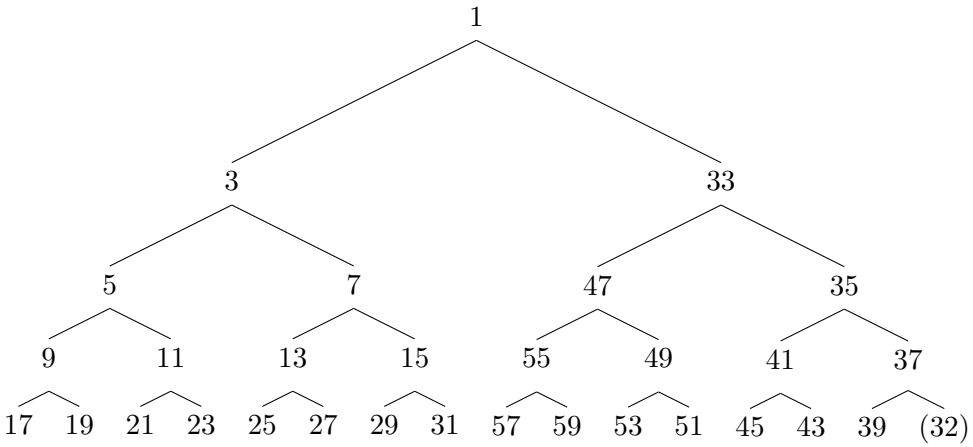
Instructors Hsuan-Tien Lin, Roger Jang

Tim Liou (b03902028)

5.1 Heap and Hash

(1) Complete Exercise R-8.24 of the textbook.

Consider a min heap. The insertion of an entry with key 32 would cause up-head bubbling to proceed all the way up to a child of the root in this heap.



(2) Complete Exercise C-8.4 of the textbook.

Combine each key we want to insert to the stack with a priority variable. These combinations would be insert to the MaxPriorityQueue which compare elements by the priority variable. Note that the basic idea of this method is the fact that every new element is pushed in stack with a priority higher than the current one. Then, MaxPriorityQueue will behave in the LIFO way, which is what we want.

```
1   class Stack {
2       class Element { int priority, Key element; };
3       MaxPriorityQueue<Element> MaxPQ;
4       int top_priority = 0;
5
6       void push(Key key) { MaxPQ.insert(Element(top_priority++, key)); }
7       void pop() { top_priority--; MaxPQ.removeMax() }
8       const Key& top() { return MaxPQ.max(); }
9       int size() { return top_priority; }
10      bool empty() { return (top_priority == 0); }
11  };
```

(3) Complete Exercise C-8.14 of the textbook.

```
1: function FindLeqElemInHeap(heap, key)
2:     while heap.top() ≤ key do
3:         insert heap.top() into the output list, and do heap.removeMin()
4:     end while
5:     return the output list
6: end function
```

Algorithm 1: Compute all the entries in a heap with a key less than or equal to the value

(4) Hash function is everywhere. Use any search engine to study the term MinHash Explain to the TAs what it is and why it is useful. Also, cite the website that you learn the term from.

I learn MinHash from **http://shihpeng.org/tag/minhash/** and **http://en.wikipedia.org/wiki/MinHash**.

MinHash is a technique to estimate how similar two sets are. We can use Jaccard similarity coefficient to find out the similarity between two sets.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

However, it is slow to compute $J(A,B)$ by the the way to check every elements in this two sets. MinHash was designed to compute $J(A,B)$ fast.

Consider a case, $S_1 = \{ b, c \}$, $S_2 = \{ a, b \}$, $S_1 = \{ b \}$

Element	S_1	S_2	S_3
a	0	1	0
b	1	1	1
c	1	0	0
d	0	0	0

And we permutate the element,

Element	S_1	S_2	S_3
c	1	0	0
d	0	0	0
b	1	1	1
a	0	1	0

Let h^k_{min} be a hash function, which k is a permutation of element. In this case, k is $cdba$. And h^k_{min} is defined as the symbol of the first non-zero row. In this case, $h^{cdba}_{min}(S_1) = c$, $h^{cdba}_{min}(S_2) = b$ $h^{cdba}_{min}(S_2) = a$. We can find that the value of $J(A,B)$ is as same as the possibility of $h^k_{min}(A) = h^k_{min}(B)$. The following is the proof.

For any two sets, we have three possible result for every elements.

Type 1 both are zero, that is, both sets do not contain this element.

Type 2 one is zero and the other is one, that is, one set contains this element.

Type 3 both are one, that is, both sets contain this element.

The number of Type 1 is M_1, the number of Type 2 is M_2, and the number of Type 3 is M_3.

$$J(A,B) = \frac{M_3}{M_3 + M_2}$$

The possibility of $h^k_{min}(A) = h^k_{min}(B)$ is

$$\begin{aligned} P(\text{Type 3} \,|\, (\text{Type 2 or Type 3})) &= \frac{\frac{M_3}{M_1+M_2+M_3}}{\frac{M_2+M_3}{M_1+M_2+M_3}} \\ &= \frac{M_3}{M_3+M_2} = J(A,B) \end{aligned}$$

If we choose enough number of k, the possibility of $h^k_{min}(A) = h^k_{min}(B)$ will be closer to $J(A,B)$.

This method would be more efficient than the naive algorithm. The time complexity change from $O(M)$ to $O(1)$ since we don't have to go through all elements. This could be useful for the sets contain lots of elements.

(5) Describe an algorithm to find out the position that the two strings differ efficiently. Briefly discuss and justify the time complexity of your algorithm.

```
1:  function FindDiffPostion(s1, s2)
2:      start ← 0
3:      end ← the length of string - 1
4:      middle ← ⌊(start + end) / 2⌋
5:      while start ≠ end do
6:          if postfixHash(s1, end - middle) ≠ postfixHash(s2, end - middle) then
7:              start ← middle + 1
8:              middle ← ⌊(start + end) / 2⌋
9:          else
10:             end ← middle
11:             middle ← ⌊(start + end) / 2⌋
12:         end if
13:     end while
14:     return start
15: end function
```

Algorithm 2: Find the position that the two strings differ efficiently

The time complexity of this algorithm is $O(\log n)$, which n is the length of two strings, because we check whether the hash values of the last half of two strings are the same. If they are same, it

means the postion that two strings differ is in first half. Otherwise, the position is in the last half. After one loop in **while**, the length of string we need to check becomes half. This implies that the time complexity of this algorithm is $O(\log n)$.

(6) Construct a perfect hash function that is efficiently computable for the following 32 standard keywords in C. You need to explain why the hash function is perfect and why it is efficiently computable to get the full bonus.

Create a table translating characters to integers. The following are the magic numbers and a magic function.

```
int HashCode(string s)
{
    int CharTable[26] = {17, 17, 4, 1, 2, 17, 15, 16, 15, 0,
                         17, 17, 5, 16, 10, 0, 0, 15, 5, 8,
                         12, 15, 15, 0, 0, 0};
    return CharTable[(s.front() - 'a')] + CharTable[(s.back() - 'a')] +
        s.length() - 8;
}
```

Result:

```
else: 0
double: 1
case: 2
enum: 3
signed: 4
do: 5
continue: 6
static: 7
default: 8
const: 9
short: 10
struct: 11
void: 12
unsigned: 13
while: 14
char: 15
extern: 16
volatile: 17
int: 18
switch: 19
sizeof: 20
goto: 21
float: 22
```

```
auto: 23
typedef: 24
union: 25
if: 26
for: 27
long: 28
return: 29
register: 30
break: 31
```

This function is efficiently conputable because the time complexity is simply $O(1)$. The only problem is that it is not trivial to find out these magic numbers.

5.2 Distributed System

(1) Finish/rewrite the BinomialHeap class, and describe how you test whether the data structure is correct.

I wrote a simple cpp file to test this data structure.

```
#include <iostream>
#include "binomial_heap.h"

#define ArraySize(A) (sizeof((A)) / sizeof(*(A)))

int main()
{
    BinomialHeap<int> bh, bh2;

    int input[] = {1, 29, 33, 234, 234, 92, 11};
    int input2[] = {2, 30, 31, 34, 92, 0, 111};

    std::cout << "****************" << std::endl;
    for (unsigned int i = 0; i < ArraySize(input); i++)
    {
        bh.insert(input[i]);
        std::cout << "insert [" << input[i] << "] to bh,\t"
                  << "bh.size = " << bh.size << std::endl;
    }
    std::cout << "****************" << std::endl;

    std::cout << "****************" << std::endl;
```

```
23      for (unsigned int i = 0; i < ArraySize(input2); i++)
24      {
25          bh2.insert(input2[i]);
26          std::cout << "insert [" << input[i] << "] to bh2,\t"
27                    << "bh2.size = " << bh2.size << std::endl;
28      }
29      std::cout << "****************" << std::endl;
30
31      std::cout << "****************" << std::endl;
32      std::cout << "bh.size = " << bh.size << std::endl;
33      std::cout << "bh2.size = " << bh2.size << std::endl;
34      bh.merge(bh2);
35      std::cout << "Merge bh2 to bh" << std::endl;
36      std::cout << "bh.size = " << bh.size << std::endl;
37      std::cout << "bh2.size = " << bh2.size << std::endl;
38      std::cout << "****************" << std::endl;
39
40      std::cout << "****************" << std::endl;
41      while (bh.size)
42      {
43          std::cout << "bh.top() is " << bh.top() << " and do bh.pop()" << std::endl
                ;
44          bh.pop();
45      }
46      std::cout << "bh.size = " << bh.size << std::endl;
47      std::cout << "****************" << std::endl;
48
49      return 0;
50  }
```

I can check the correctness of this data structure by the output.

```
1   ****************
2   insert [1] to bh, bh.size = 1
3   insert [29] to bh, bh.size = 2
4   insert [33] to bh, bh.size = 3
5   insert [234] to bh, bh.size = 4
6   insert [234] to bh, bh.size = 5
7   insert [92] to bh, bh.size = 6
8   insert [11] to bh, bh.size = 7
9   ****************
10  ****************
11  insert [1] to bh2, bh2.size = 1
12  insert [29] to bh2, bh2.size = 2
```

```
insert [33] to bh2, bh2.size = 3
insert [234] to bh2, bh2.size = 4
insert [234] to bh2, bh2.size = 5
insert [92] to bh2, bh2.size = 6
insert [11] to bh2, bh2.size = 7
****************
****************
bh.size = 7
bh2.size = 7
Merge bh2 to bh
bh.size = 14
bh2.size = 0
****************
****************
bh.top() is 234 and do bh.pop()
bh.top() is 234 and do bh.pop()
bh.top() is 111 and do bh.pop()
bh.top() is 92 and do bh.pop()
bh.top() is 92 and do bh.pop()
bh.top() is 34 and do bh.pop()
bh.top() is 33 and do bh.pop()
bh.top() is 31 and do bh.pop()
bh.top() is 30 and do bh.pop()
bh.top() is 29 and do bh.pop()
bh.top() is 11 and do bh.pop()
bh.top() is 2 and do bh.pop()
bh.top() is 1 and do bh.pop()
bh.top() is 0 and do bh.pop()
bh.size = 0
****************
```