Algorithms
Assignment 1
Augmentation

Hoel Kervadec

January 27, 2015

Contents

1 Description of the problem

The aim of this problem is to create a database to store temperature measurements using an existing datastructure.

The tuples are of the form (t, c), with t being the date of the measurement (in the form YYYYMMDDHHmm), and c being the temperature. We can also note that we won't have two identical measurements.

The data structure must have the following basic operation:

- $Add(D, t, c)$: Add (t, c) to the database D.
- $Delete(D, t, c)$: Remove (t, c) of the database D.
- $Max(D, t)$: Return the maximum temperature at time t.
- $Max(D, t1, t2)$: Return the maximum temperature in $[t1; t2]$.

Each operation should be at worse $\mathcal{O}(\log_2 n)$, with a preprocessing time of $\mathcal{O}(n \log_2 n)$.

2 Self-balancing binary search tree

Self-balancing binary search tree are great for accessing and modifying elements in $\mathcal{O}(\log_2 n)$ time.

However, it requires a way to sort elements between them. We could sort the elements first by date, and then by temperature.

Let's first create a basic type for the datapoints:

```
type DP:
    int t
    int c
```

Let's assume we got an base class `SBBST<E>`. We could create the class `TempDB` extending `SBBST<DP>`. To have Add and Delete working, we would only need to override the functions `compare(DP d1, DP d2)` and `equal(DP d1, DP d2)`.

```
class TempDB extends SBBST<DP>:
    override bool compare(DP d1, DP d2):
        if d1.t == d2.t:
            return d1.c < d2.c

        return d1.t < d2.t


    override bool equal(DP d1, DP d2):
        return d1.t == d2.t and d1.c == d2.c
```

Now, we can add our two functions Max

```
int Max(TempDB D, int t):
    if D == null:
        return -275

    if D.value.t == t:
        return max(D.value.c,
                   Max(D.right, t))

    if D.value.t < t:
        return Max(D.left, t)

    if D.value.t > t:
        return Max(D.right, t)

int Max(TempDB D, int t1, int t2):
    if D == null:
        return -275

    if D.value.t in [t1;t2]:
        return max(D.value.c,
                   Max(D.left, t1, t2),
                   Max(D.right, t1, t2))

    if D.value.t < t1:
        return Max(D.left, t1, t2)

    if D.value.t > t2:
        return Max(D.right, t1, t2)
```

The complexity would be approximately $\mathcal{O}(\log_2 n)$.

3 HashTable

Another way of storing the temperatures would be by using a HashTable. We could use the date as the key, and the values would lists of the temperatures for this time.

We can assume that there is a finite number of locations which does the measurements, m. Thus, the time to add or delete an element would be $\mathcal{O}(1 + m)$.

A way to improve this would be to use binary search trees to store the temperatures. This way, the complexity would become $\mathcal{O}(1 + \log_2 m)$.

Which such a data structure, implementing Max would be really easy to do:

```
int Max(TempDB D, int t):
    % I assume there is a function returning
    % the maximum value of the tree
    return D.get(t).getMax()

int Max(TempDB D, int t1, int t2):
    ts = D.keySet().filter(t => t is in [t1; t2])

    return max([D.get(t).getMax() for t in ts])
```

However, the complexity of the second function would no so great. We could improve it with the next hash optimization.

3.1 Hash optimization

Hash function is always a big question in hashmaps. At the moment, we can use the date as a hash, but with some assumption, we could do far better.

For instance, we can suppose that the measurements are periodic (one per hour, for example). Let's call the period p, in seconds. By converting the time t to a timestamp t', we could use a modulo operation. Also, we can suppose that there is no measurements before the date t_0.

This way, we could produce the hash function:

$$hash(t) = (t' - t'_0)\%p.$$

Now, we could rewrite our second Max function, to use this new property:

```
int Max(TempDB D, int t1, int t2):
    tt1 = to_timestamp(t1)
    tt2 = to_timestamp(t2)

    ts = [tt1 + i*p for i in range(0, (tt2 - tt1)/p)]

    return max([D.get(t).getMax() for t in ts])
```

4 Conclusion

I found different approach for this problem. The first one works only be using the problem informations, while the second one need some assumptions (probably close to reality).