Advanced Programming

Sino-Italian Campus, Tongji University (Shanghai)

Develop an ADT with functions to analyse traffic on an IP network, as follows.

```
int setPacket(int IPnumber, int nBytes);
```

States that the node *IPnumber* has sent *nBytes* bytes

```
int setPacketName(int IPnumber, char* name);
```

States that node *IPnumber* is called *name*

```
void nodesInUsageOrder();
```

Returns the nodes, ordering them from the most used (in term of bytes sent) to the least used, matched by the name.

Example:

```
setPacket(21003 , 60);
setPacket(21015 , 200);
setPacket(22020 , 35);
setPacket(21012 , 200);
setPacket(21003 ,  20);
setPacket(21015 , 120);
setPacket(21003 ,  10);

setPacketName(21003 , “nodeZ”);
setPacketName(21012 , “nodeJ”);
setPacketName(21015 , “nodeA”);
setPacketName(22020 , “nodeW”);
nodesInUsageOrder();
    // should printout  nodeA, 320
                        nodeJ , 200
                         nodeZ, 90
                        nodeW, 35
```

CONSTRAINTS

The time complexity of all functions should be minimal, considering that `setPacket` could be called millions of times per day, `setPacketName` once per day, `nodesInUsageOrder` once per day. In case of conflict between time and space complexity give priority to improve time complexity.

DISCUSSION

The problem has two parts.

1. Managing the triples (IpNumber, name, nBytes), with setPacket and setPacketName. Remember that setPacket is called millions of times per day. SetPacketName once per day, per IpNumber.
2. Ordering IpNumbers on nBytes sent on a day. This is called once per day.

In practice we can immediately write the equation to compute time spent in total, every day

T = t_setPacket * numberOfPacketsSwitchedPerDay + t_setPacket_name * numberOfDifferentNodes + t_nodesInUsageOrder

Clearly t_setPacket is the term that has the biggest influence because numberOfPacketsSwitchedPerDay is in the order of millions. Also t_setPacket_name is important because also numberOfDifferentNodes is a large number. t_nodesInUsageOrder is only called once per day so it should be less important (however it should not be overlooked either).

For the **first part** of the problem. It is better to use one container of triples (and not two separate containers, IpNumber-name and IpNumber-nBytes – this latter approach would increase memory usage without advantages on time).

Let's review our options

- Unbounded array, elements unordered. SetPacket requires to go through all elements, on average has complexity numberOfDifferentNodes/2
- Linked list, elements unordered. Same as above.
- Unbounded array, elements ordered on IpNumber. setPacket is in practice a search, doing it with a convenient search algorithm (ex binary search, because the array is ordered) has complexity ln_2(numberOfDifferentNodes)
- BST with key IpNumber. setPacket is a search on a BST, has complexity ln_2(numberOfDifferentNodes)
- Hash table with key IpNumber. setPacket is a get on a hash table, constant time (if the hash table is correctly sized – in the order of numberOfDifferentNodes)

So the hash table is the best option (provided it is correctly sized)

For the **second part** of the problem. Assuming we have used the hash table, the triples are not ordered in any way. We need ordering on nBytes (NOT on IpNumber), remembering that several nodes can have the same nBytes (so nBytes is not unique). A unique ordering key would allow using a BST (take each element from the hash, and insert it into a BST using nBytes as key. Then do an inOrder trasversal of the BST). But nBytes is not unique. Another option then is to use an orderedArray. Take each element from the hash, and insert it in an array, keeping it ordered on nBytes. At the end trasversing the array from the first element to the last returns elements ordered on nBytes.