# Tutorial - Hashing

Prasanta Dutta

# Content

---

- Problem Discussion (Previous year)
- Examples
- Solution
- Implementation

# Problem Discussion

- **X is a certain company** which operates in stock market. They enter huge number of financial transaction - call trades - every day. On the other side of each trade, there is some company **Y, call it a counterparty of X**.

- To each trade of X is assigned a **portfolio number**, which is **not a unique** identifier of the trade, and is not even unique per counterparty. In other words, trades with different counterparties may be assigned the same portfolio, and different trades with the same counterparty may be assigned distinct portfolios as well, according to some internal policy of the company.

- Whenever X enters into a trade, a new line such as **“+ 101 26”** is appended to a log file, where **“+”** indicates that a **new trade** was initiated by X, **“101”** is the **counterparty id**, and **“26”** is the **portfolio** of that trade. At any time during a typical day, however, a counterparty **Y may withdraw** all transactions with X. From X’s standpoint, that means all trades it had entered into with Y that day are now considered void, and a line such as **“- 101”** is logged. Here, the **“-”** sign indicates that a **cancelation** took place, and **“101”** is the id of the **counterparty** the cancelation refers to.

# Objective

- By the end of each day, the company X needs to determine the set of **distinct portfolios** associated to trades that are still active by then, that is, the portfolios of trades that have not been canceled during the day.

- You are required to find an **efficient solution** to this using **Hashing** and print the **hashtables**.

# Some Conditions

1. The **hashing function** is a simple mod function **(K%size)**, where K is the input key and size is the size of hash table (aka hash map).

2. **Collisions** will be handled by **separate chaining**.

3. You may use the following definition to define the hash

```
typedef struct _hashing {
      int key;
      struct _hashing *next;
} hash;
```

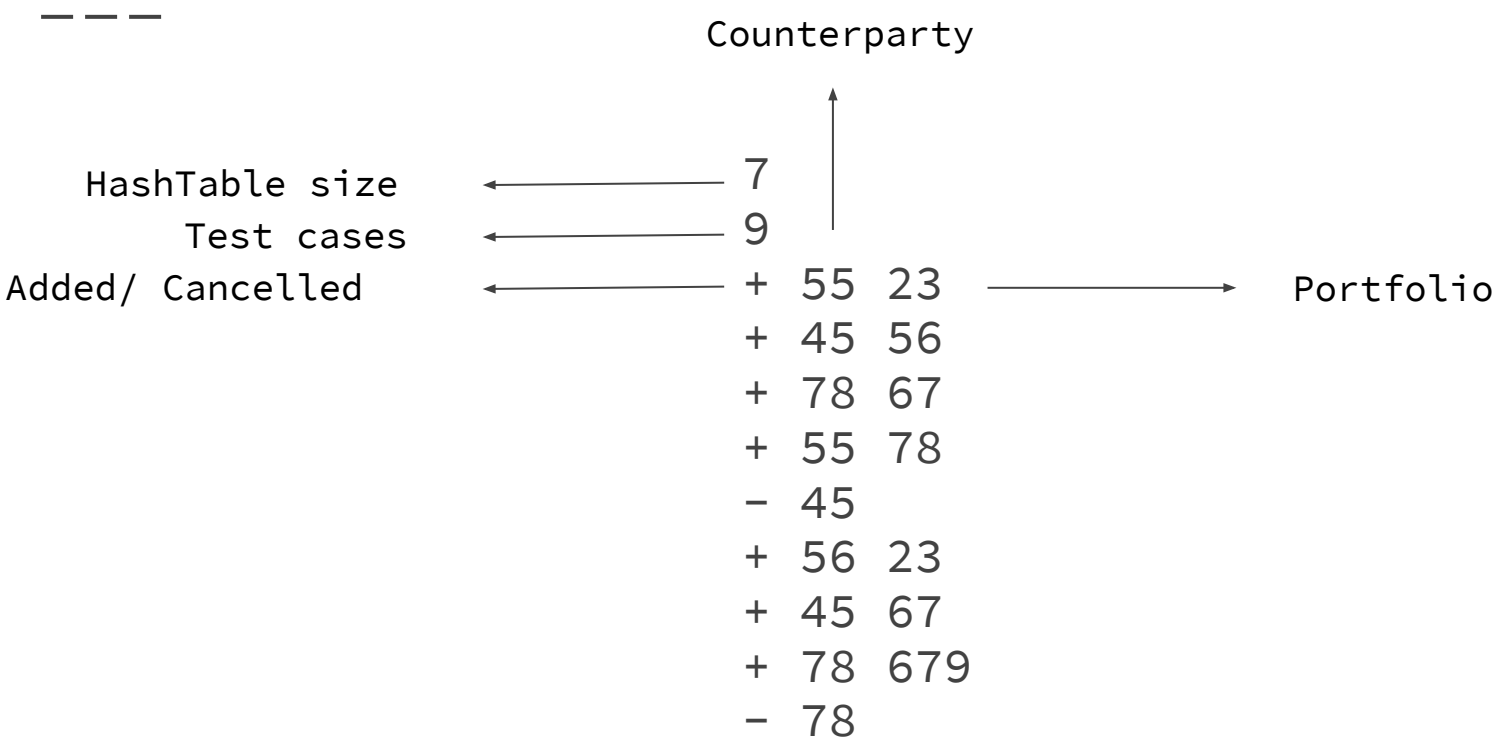
# The log file

| | | |
|---|---|---|
| + | 101 | 26 |
| + | 2 | 25 |
| + | 101 | 25 |
| + | 3005 | 4550 |
| - | 101 | |
| + | 3005 | 26 |
| + | 4 | 184 |
| + | 101 | 4550 |
| - | 2 | |

Table 1: Sample log file.

Tell me the answer...

# Input format



# Solution 1 : Two mirroring hash maps

# Data structure - Hash Table

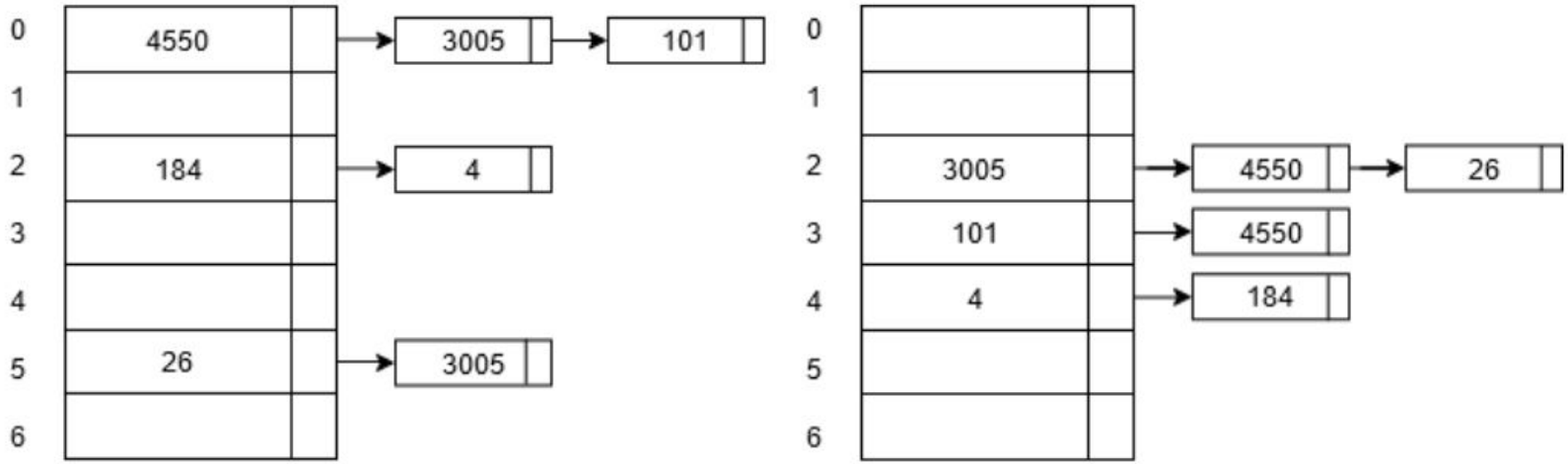
---

- You will have to implement two hash maps -

- By-portfolio hash map
  - The keys are portfolio numbers
  - Values are list containing the counterparties

- By-counterparty hash map
  - Keys are the counterparties
  - Values are list containing the portfolios

# Visual Demonstration

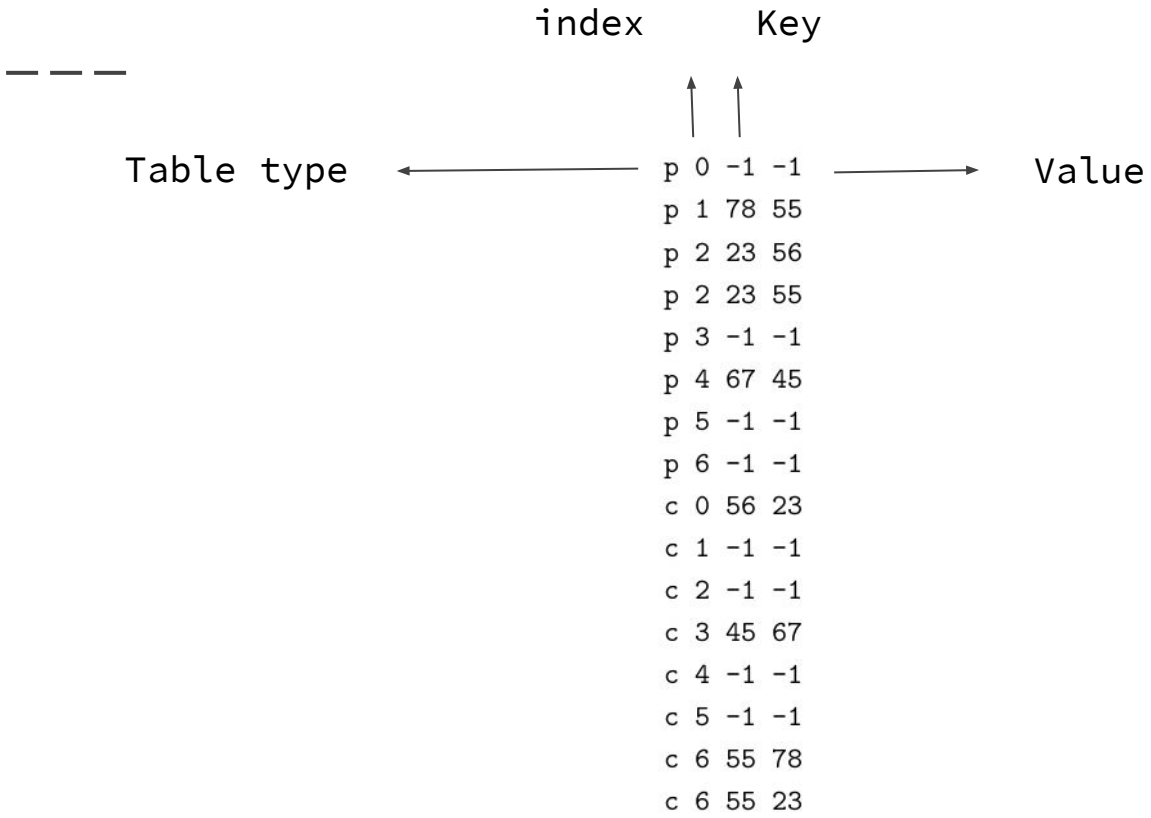



(a) counterparties-by-portfolio map

(b) portfolios-by-counterparty map

Figure 1: Two mirroring hash maps. Collision handling is done using seperate chaining.

# Output Format



# Algorithm (Incomplete)

---

- The log file is traversed top-down.

- For Each line with a “+” sign, insert (key,value) into the two hashmaps.

- For Each line with a “-” sign, delete (key, value) from two hashmap

# Problem

---

- In by-portfolio hashmap, we need to traverse the complete hashmap in order to determine the counterparty (that cancelled the transaction).

# Algorithm (Complete)

- The log file is traversed top-down.

- For Each line with a “+” sign, insert (key,value) into the two hashmaps.

- For Each line with a “-” sign,
  - For by-counterparty hashmap, delete the counterparty ids from it and save the deleted portfolio ids (so gotten) in a list L.
  - For by-portfolio hashmap, delete the portfolio ids which are in L.

# Basic Structures

```
typedef struct HashNode
{
    int key;
    int value;
    struct HashNode *next;
}HashNode;
```

```
typedef struct HashTable
{
    int size;
    struct HashNode **table;
}HashTable;
```

# Initialization

```
HashTable *initializeHashTable(int size)
{
   HashTable *hashTable = (HashTable *)malloc(sizeof(HashTable));
   hashTable->size = size;
   hashTable->table = (HashNode **)malloc(sizeof(HashNode *) * size);
   int i;
   for(i = 0; i < size; i++)
   {
       hashTable->table[i] = NULL;
   }
   return hashTable;
}
```

# Insertion

```
void insertHashNode(HashTable *hashTable, int key, int value)
{
   int hash = key % hashTable->size;
   HashNode *newNode = (HashNode *)malloc(sizeof(HashNode));
   newNode->key = key;
   newNode->value = value;
   newNode->next = hashTable->table[hash];
   hashTable->table[hash] = newNode;
}
```

# Deletion

```
void deleteHashNode(HashTable *hashTable, int key)
{
   int hash = key % hashTable->size;
   HashNode *temp = hashTable->table[hash];
   while(temp != NULL)
   {
        HashNode *next = temp->next;
        free(temp);
        temp = next;
   }
   hashTable->table[hash] = NULL;
}
```

# Reading input file

```
HashTable* readInputFile(char *fileName)
{
   FILE *file = fopen(fileName, "r");
   char sign;
   int x1,x2, testcases;
   fscanf(file, "%d", &hashtableSize);
   HashTable *hashTable = initializeHashTable(hashtableSize);
   fscanf(file, "%d", &testcases);
   while(fscanf(file, "%c", &sign) != EOF)
   {
       if(sign == '+')
       {
           fscanf(file, "%d %d", &x1, &x2);
           insertHashNode(hashTable, x1, x2);
       }
       else if(sign == '-')
       {
           fscanf(file, "%d", &x1);
           deleteHashNode(hashTable, x1);
       }
   }
   fclose(file);
   return hashTable;
}
```

# Writing into the output file

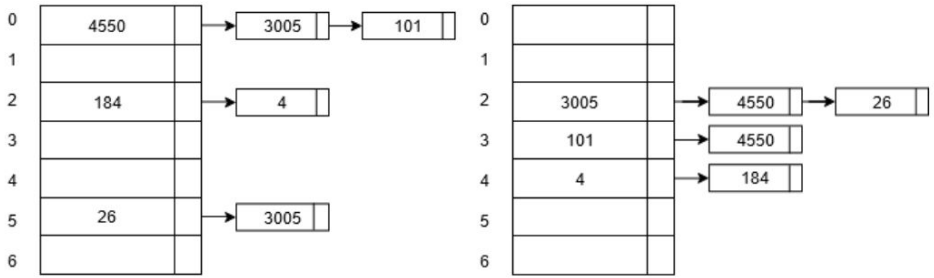
```
void writeOutputFile(char *fileName, HashTable *hashTable, char symb)
{
    FILE *file = fopen(fileName, "w");
    int i;
    for(i = 0; i < hashTable->size; i++)
    {
        HashNode *temp = hashTable->table[i];
        if(temp==NULL)
        {
            fprintf(file, "%c %d %d %d\n",symb, i, -1, -1);
        }
        else
        {
            while(temp != NULL)
            {
                fprintf(file, "%c %d %d %d\n",symb, i, temp->key, temp->value);
                temp = temp->next;
            }
        }
    }
    fclose(file);
}
```

## What is the problem with this solution ?

# Problem with this solution

---

- Chaining : Since we are using chaining as collision resolution technique, hence the search time may not be O(1) always
- What is the solution ?




(a) counterparties-by-portfolio map (b) portfolios-by-counterparty map

Figure 1: Two mirroring hash maps. Collision handling is done using seperate chaining.

# Solution 2 : Multi Level Hashing

# HashSet

---

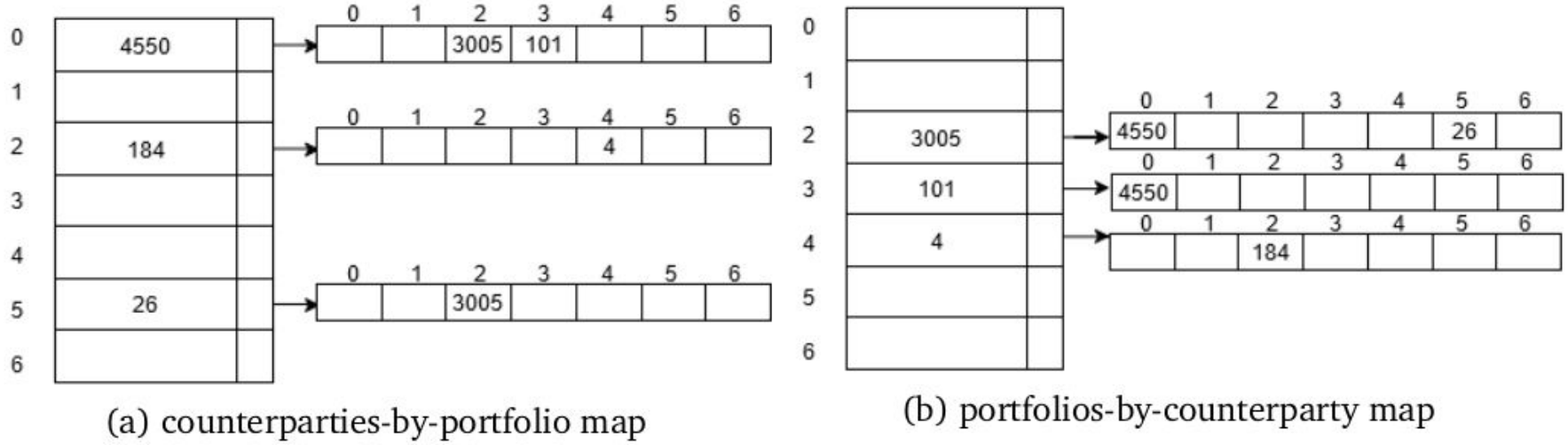
- Each element in the hashset can be searched in O(1) time
- How to implement HashSet ?

# Solution

---

- For each index of the hashmap, we can create another hashmap

# Visual Demonstration




(a) counterparties-by-portfolio map

(b) portfolios-by-counterparty map

Figure 2: Hash map multi level example

# Basic Structures

```
typedef struct HashNode
{
    int key;
    int value;
}HashNode;
```

```
typedef struct HashTable
{
    int size;
    struct HashNode **table;
}HashTable;
```

# Initialization

```
HashTable *initializeHashTable(int size)
{
   HashTable *hashtable = (HashTable *)malloc(sizeof(HashTable));
   hashtable->size = size;
   hashtable->table = (HashNode **)malloc(sizeof(HashNode *) * size);
   for(int i = 0; i < size; i++)
   {
       hashtable->table[i] = (HashNode *)malloc(sizeof(HashNode) * size);
   }
   for(int i = 0; i < size; i++)
   {
       for(int j = 0; j < size; j++)
       {
           hashtable->table[i][j].key = -1;
           hashtable->table[i][j].value = -1;
       }
   }
   return hashtable;
}
```

# Insertion

```
void insertKeyValuePair(HashTable *hashtable, int key, int value)
{
    int hashIndex = hashFunction(key);
    int hashIndex2 = hashFunction(value);
    hashtable->table[hashIndex][hashIndex2].key = key;
    hashtable->table[hashIndex][hashIndex2].value = value;
}
```

# Deletion

```
void deleteKeyValuePair(HashTable *hashtable, int key)
{
    int hashIndex = hashFunction(key);
    for(int i=0;i<hashtableSize;i++)
    {
        hashtable->table[hashIndex][i].key = -1;
        hashtable->table[hashIndex][i].value = -1;
    }
}
```

# Complete Code

---

Complete code is available at :
https://gist.github.com/duttaprasanta/6df26226f0a79f106539c1227c7d20a0

---

# Can it be solved using one hashmap/ hashset ?

- If so, then which hashmap should we use?

# Answer

---

- Yes, using by-counterparty hashmap

- In that case, after getting the output, we need to find out the unique portfolios.

- What will be the time complexity in that case?

# Other solution ?

# Algorithm

---

- S = Empty Set
- L = dict{}
- Traverse the log file from bottom to up.
- If you get ‘-’, L[counterparty] = True
- If you get ‘+’
  - if counterparty in L, continue
  - else S = S U {portfolio}
- return S as set of all existing portfolios.

---

# Thank you

- Do you have any question regarding this?
- My email : prasantadutta@kgpian.iitkgp.ac.in