tags: 資料結構

HW28

本題需要簡單實現線性插入的 hash table 並且提出方法修復課本中的刪除及搜尋函數。

Insert

因為需要進行線性插入，因此我們在這邊採取的作法是先對 key 進行 hash 做為 index，接著再判斷該位置能不能插入，如果不行的話則持續位移直到找到一個可以插入的位置，這邊採用的方法為持續加一，其實也可以運用位元位移來讓 index 更為分散，不過為了方便這邊就不這麼麻煩。

```
1  void insert(int key, int val)
2  {
3      int idx = hash(key);
4      Node* tmp = init(key, val);
5      while (arr[idx] && arr[idx]->key != key && ~arr[idx]->key)
6          idx++, idx %= maxN;
7      if (!arr[idx] || !~arr[idx]->key)
8          arr[idx] = tmp;
9  }
```

尋找 index 時會有幾種情況不能進行插入需要進行位移，如下。

- 該位置已經有其他元素存在了且該位置的元素的 key 不等於我們要進行插入的 key (如果允許等於就變相是更新)
- 該位置值不為 -1，原因是我將 -1 作為刪除標記

最後只需要位移直到找到允許插入的位置就可以結束操作。

Delete

刪除操作一樣需要先將 key hash 作為起始 index，接著持續位移直到找到目標 key 並將其標記為 -1，刪除操作就結束了。

```
void destroy(int key)
{
    int idx = hash(key);
    while (arr[idx]) {
        if (arr[idx]->key == key) {
            arr[idx]->key = -1;
            return;
        }
        idx++, idx %= maxN;
    }
}
```

Search

搜尋操作也需要先將 key hash 作為起始 index，處此之外還需要多一個 count 參數用來記錄我們位移的次數以防造成無限迴圈，最後也是一樣一直持續位移找到目標 key 返為其值，如果找到位置為空或是 count 超出 hash table 容量就代表不存在該 key，直接返回 -1。

```
int search(int key)
{
    int idx = hash(key), cnt = 0;
    while (arr[idx] && cnt <= maxN) {
        if (arr[idx]->key == key)
            return arr[idx]->val;
        idx++, idx %= maxN;
    }
    return -1;
}
```