Drishti Maharjan 26 Apr, 2019

Assignment 9 Algorithms & Data Structures

Problem 9.1

a) Here, $m = 5$

$h_1(k) = k \bmod 5$

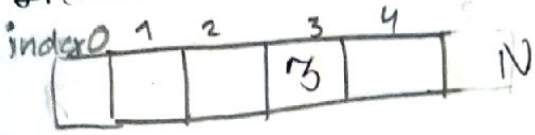
$h_2(k) = 7k \bmod 8$

=> Hashing $k = 3$

Applying $h_1$ on $k = 3$

From $h_1$, we have index $= 3 \bmod 5 = 3$

Index 3 is available, so current index = 3.



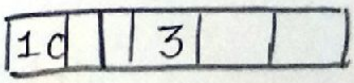

No collision

Hashing $k = 10$

Applying $h_1$ on $k = 10$

From $h_1$, we have index $= 0$

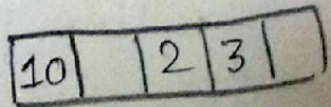
Index 0 is available, so, we don't need to look for $h_2$.



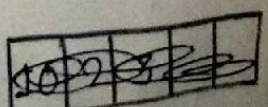

No collisio

Hashing $k = 2$

Applying $h_1$ on $k = 2$




from $h_1$, we have index $= 2$.



Index 2 is available, so index = 2.

No collision.

Hashing k = 4,

Applying $h_1$ on k = 4,

index = 4

Index 4 is available,

no collision occured.

Final Array sequence:

| 10 | | 2 | 3 | 4 |
|---|---|---|---|---|

b) Hashtable.cpp

## Problem 9.2

a) This can be proved with the help of a counter example.

Let's say our start-finish intervals in this manner: [(2,4), (3,5), (4,11)]. If we take the activity with the shortest duration, we pick (3,5). Then, we can't choose any other activity as they overlap with it. So, our resulting activity is only the 2nd activity (3,5).

This is not a globally optimal solution as for this problem, we could have done activity 1 and 3 [(2,4), (4,11)], which is a globally optimal solution. Since, we know that this greedy choice doesn't always give the globally optimal solution, the statement is proved.

b) //Note: using[] contains selected activities in //reverse order (at last)

Demo: (A)

| Activity no | start time | End Time |
|---|---|---|
| 1 | 2 | 3 |
| 1 | .. | |
| 2 | | |
| 3 | | |
| 4 | | |

Array Structure

=)
```
1. for i ← 1 to n
2. { A[i][1] ← i
3.     A[i][2] ← starting time
4.     A[i][3] ← ending time
5. }
6. Sort A according to finish time using Quick Sort // sort
                                    // using algorithm
                                    // lesser than O(n lg n)
```

Array using

| Index / Activity no | 1 | | | .. |
|---|---|---|---|---|

```
7. latest_start ← A[i][2]
8. last_activity ← A[i][1]
9. for i ← 1 to n
10. { if (A[i][2] > latest_start)
11.     { latest_start ← A[i][2]
12.       last_activity ← A[i][1]
13.       last_finish ← A[i][3]
14.     } } // get activity with latest starting time
15. j ← n; k ← 2
16. using[1] ← last_activity // first selected activity
17.   sj ← latest_start
18. for i ← n to 1 // start looking from back
19.   { if (A[i][1] != last_activity) // avoid repeating
20.     { fi ← A[i][3]
21.       if (fi ≤ sj) // if previous final time is less
22.       { using[k] ← A[i][1] // activity selected
23.         j ← i, k ← k+1 // change index
24.         sj ← A[j][2] // update start check point
25.       } }
26. }
```