Lab3 Solution hint

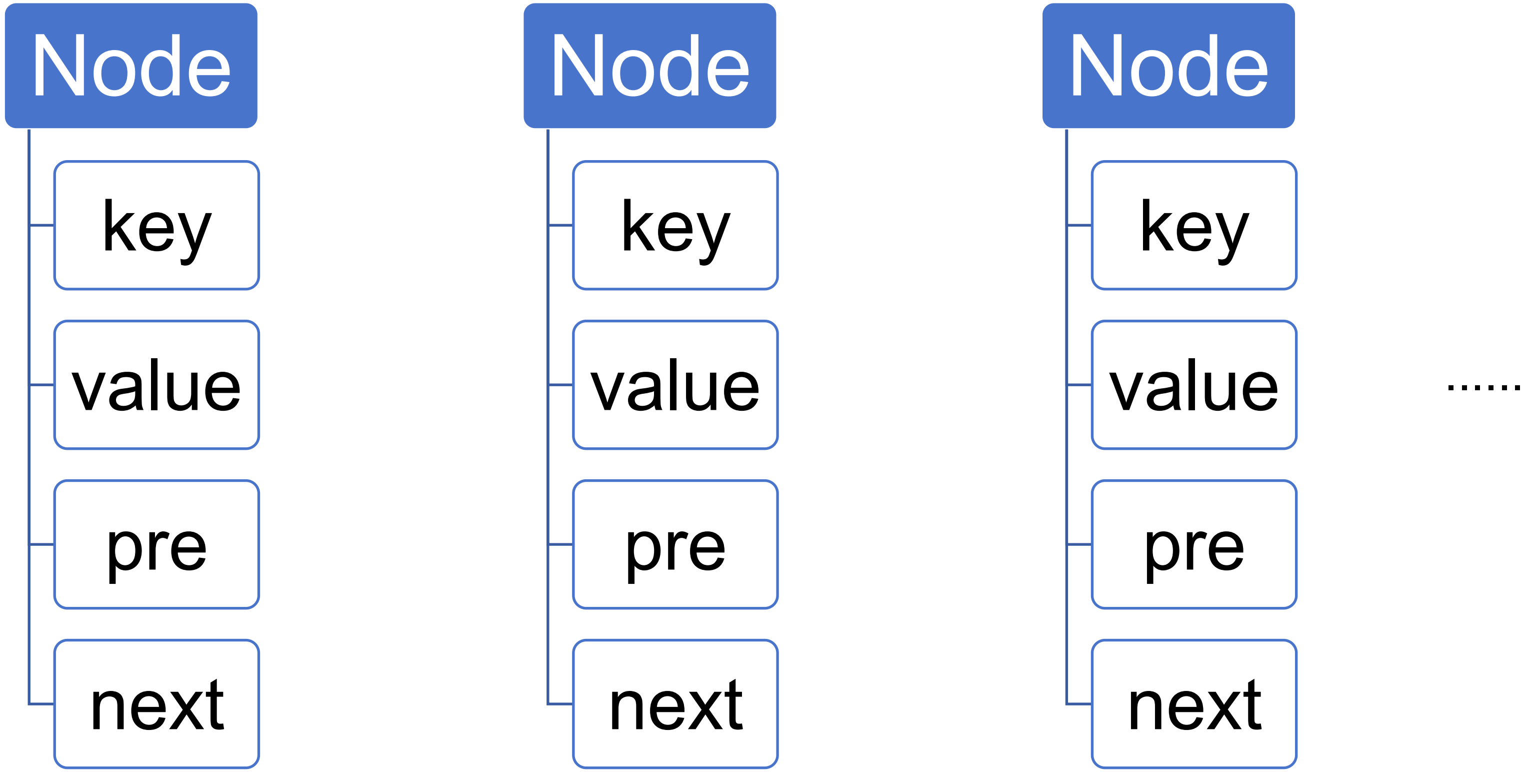
by wwy

Lab3A

- # LRU Cache
- You need to implement a **Least Recently Used (LRU) Cache** that supports the following operations efficiently:
- 1. **get key** - Print the value of the key if it exists; otherwise, return **-1**.
- 2. **put key value -** Update the value of the key if the key exists. Otherwise, add the key-value pair to the cache. If the cache exceeds its capacity after this operation, evict the **least recently used** key.
- Both operations must run in **O(1) average time complexity**.
- N and M (1 ≤ N ≤ 1,000,000, 1 ≤ M ≤ 2,000,000) — the cache capacity and the number of operations.
- The next M lines contain one of the following operations: put key value (1 ≤ key ≤ 1e9, value ≤ 1e9), get key (1 ≤ key ≤ 1e9)

No LinkedHashMap

Lab3A



Node
key
value
pre
next

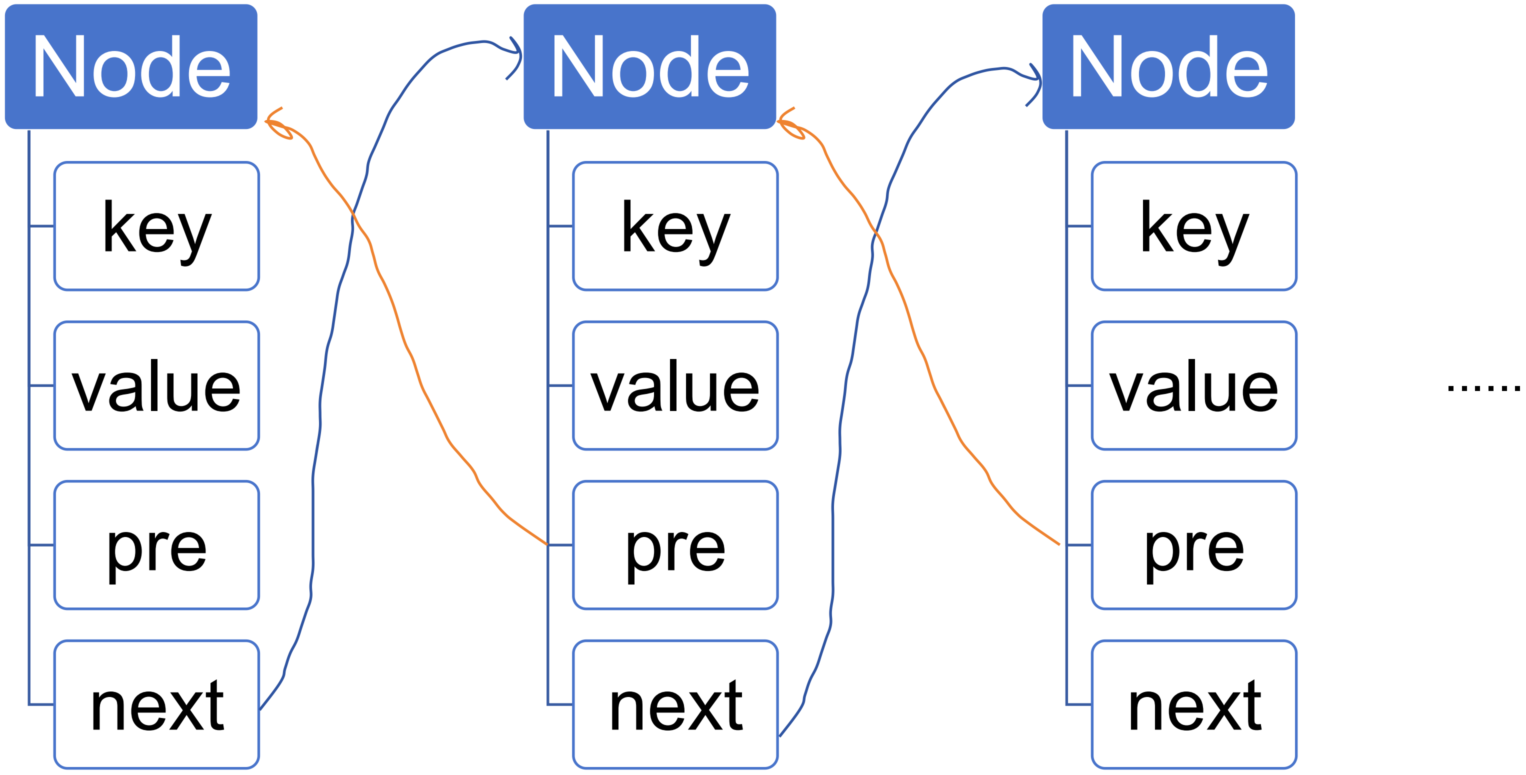
Node
key
value
pre
next

Node
key
value
pre
next

……

```
2 9
put 1 1
put 2 2
get 1
put 3 3
get 2
put 4 4
get 1
get 3
get 4
```

Lab3A



Node

key

value

pre

next

Node

key

value

pre

next

Node

key

value

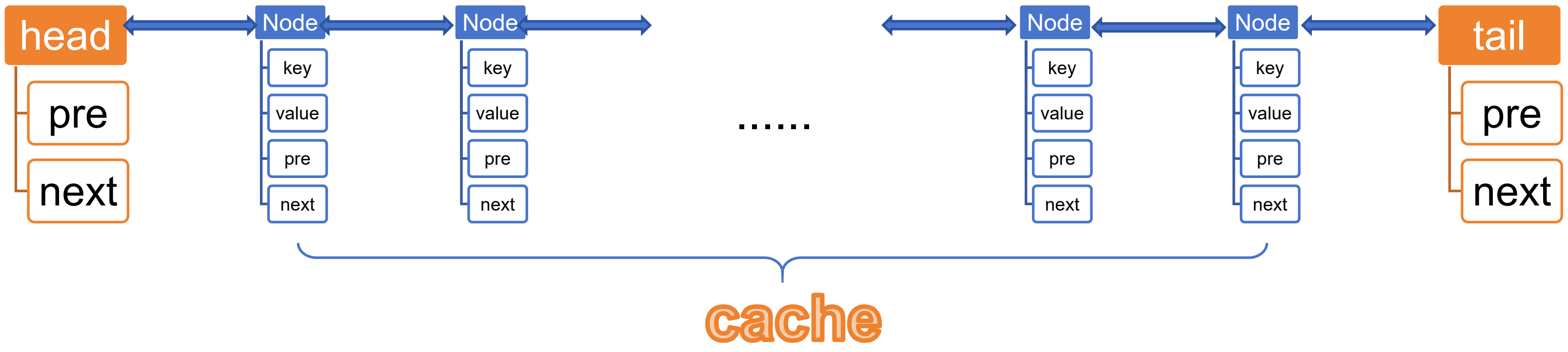
pre

next

......

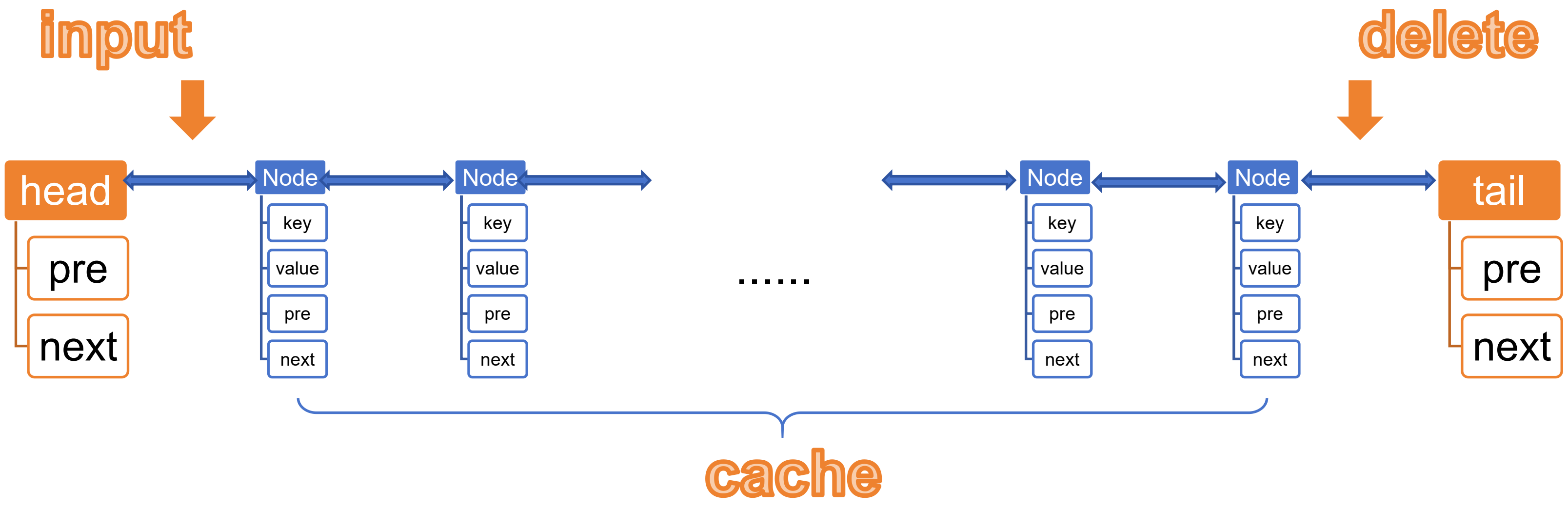
```
2 9
put 1 1
put 2 2
get 1
put 3 3
get 2
put 4 4
get 1
get 3
get 4
```

Lab3A



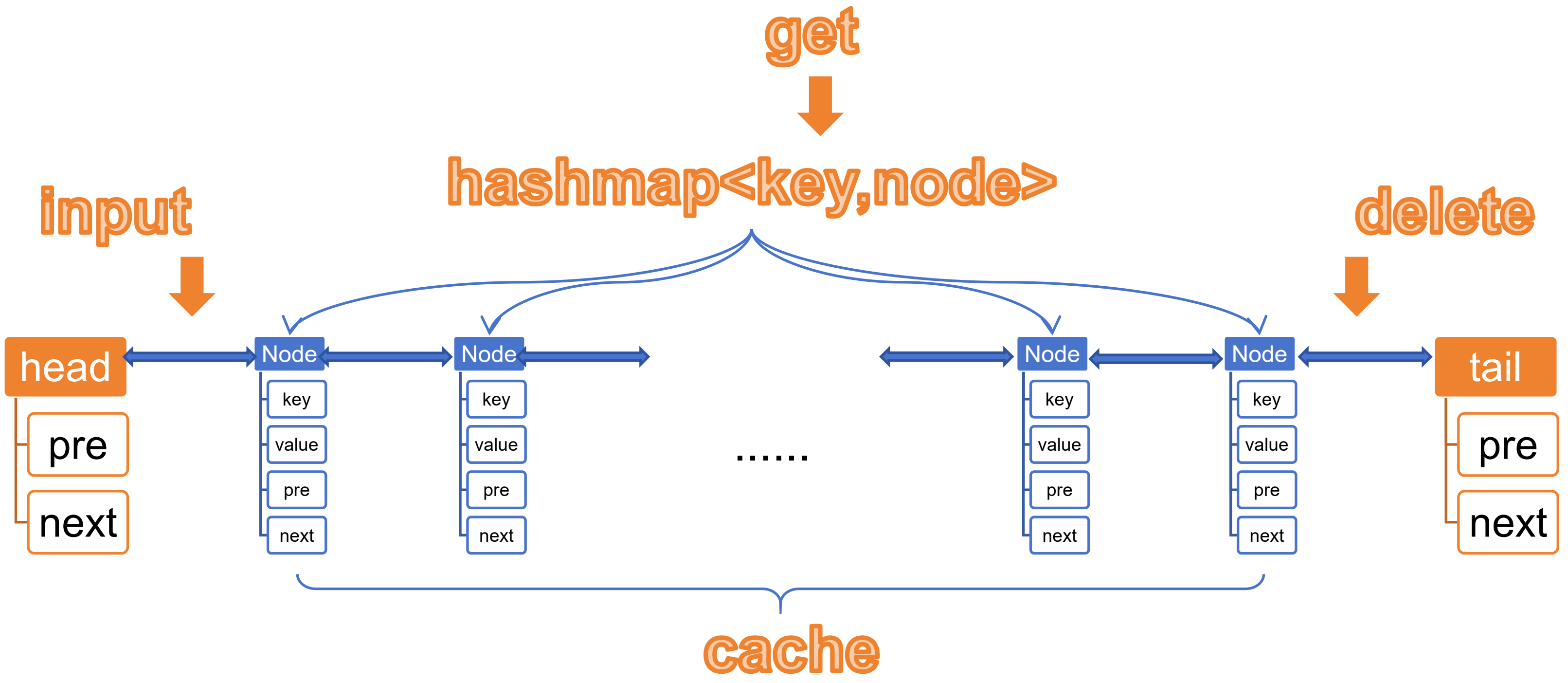
```
2 9
put 1 1
put 2 2
get 1
put 3 3
get 2
put 4 4
get 1
get 3
get 4
```

Lab3A



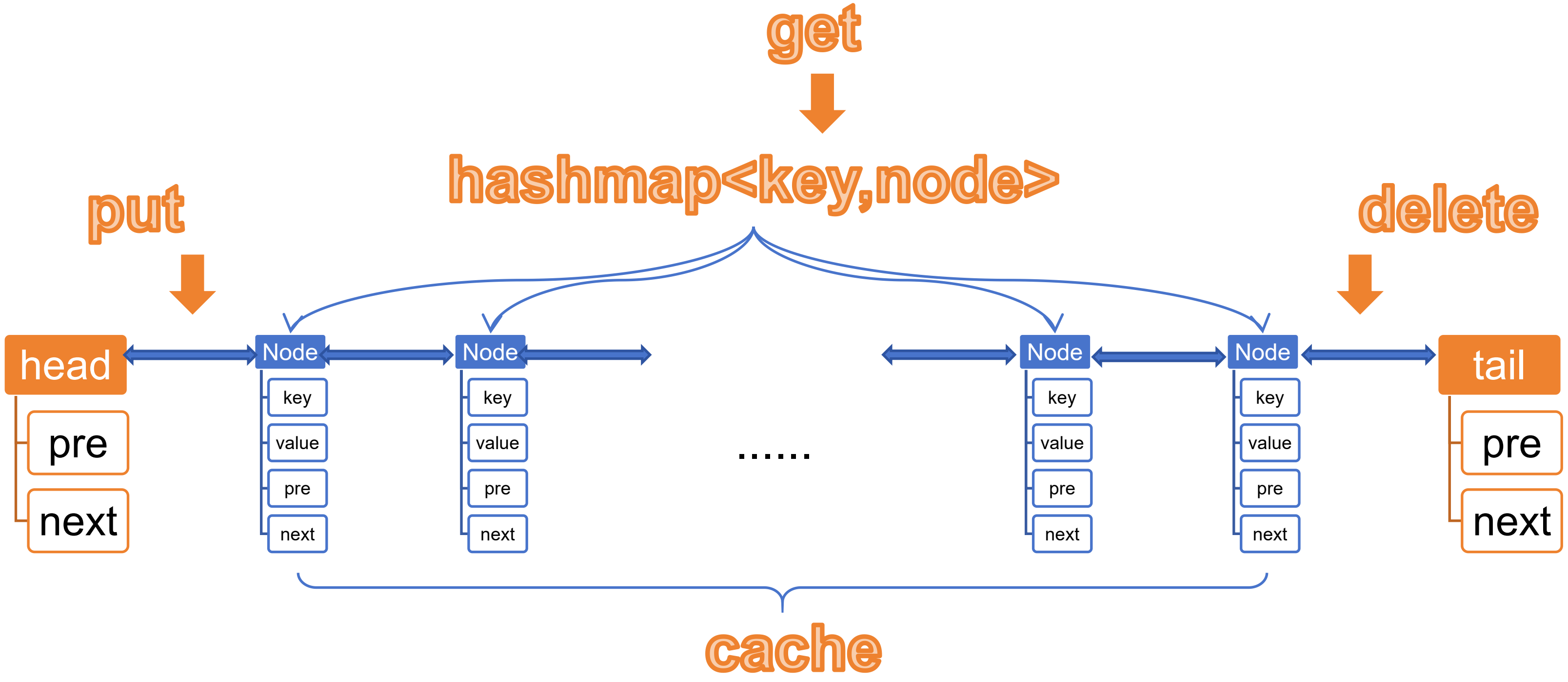
```
2 9
put 1 1
put 2 2
get 1
put 3 3
get 2
put 4 4
get 1
get 3
get 4
```

Lab3A



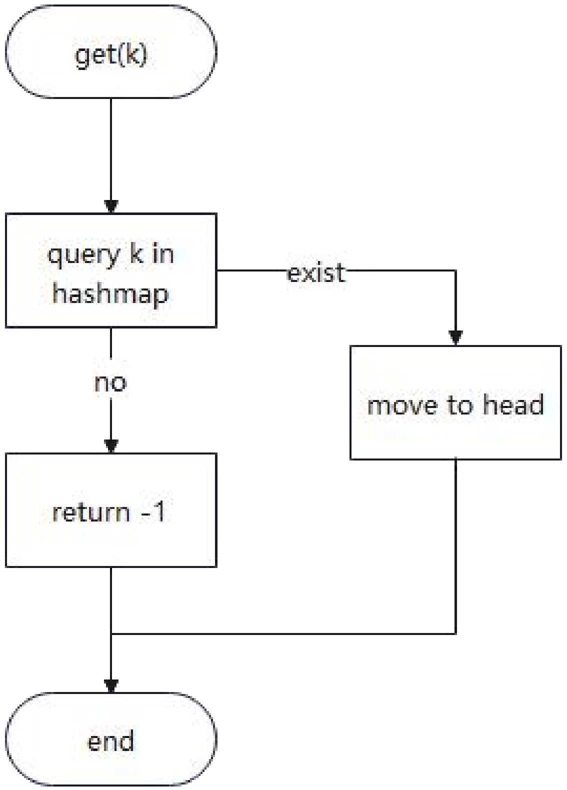
```
2 9
put 1 1
put 2 2
get 1
put 3 3
get 2
put 4 4
get 1
get 3
get 4
```

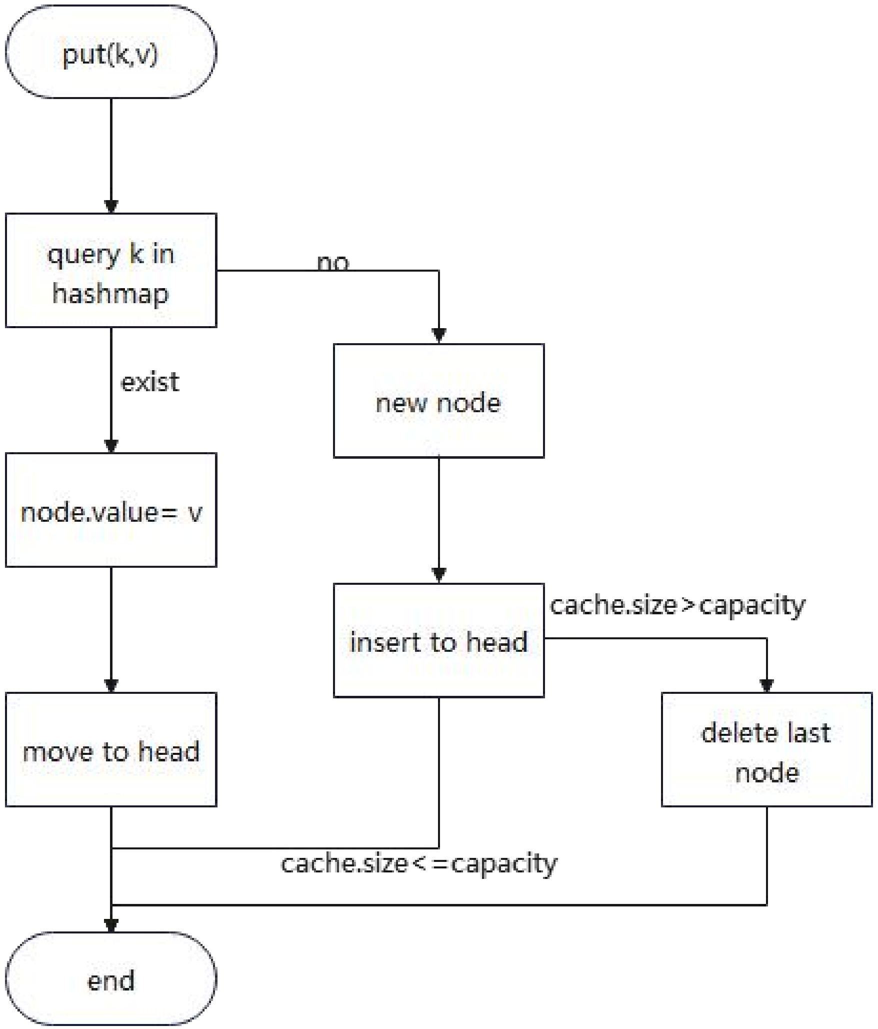
Lab3A



```
2 9
put 1 1
put 2 2
get 1
put 3 3
get 2
put 4 4
get 1
get 3
get 4
```

Lab3A

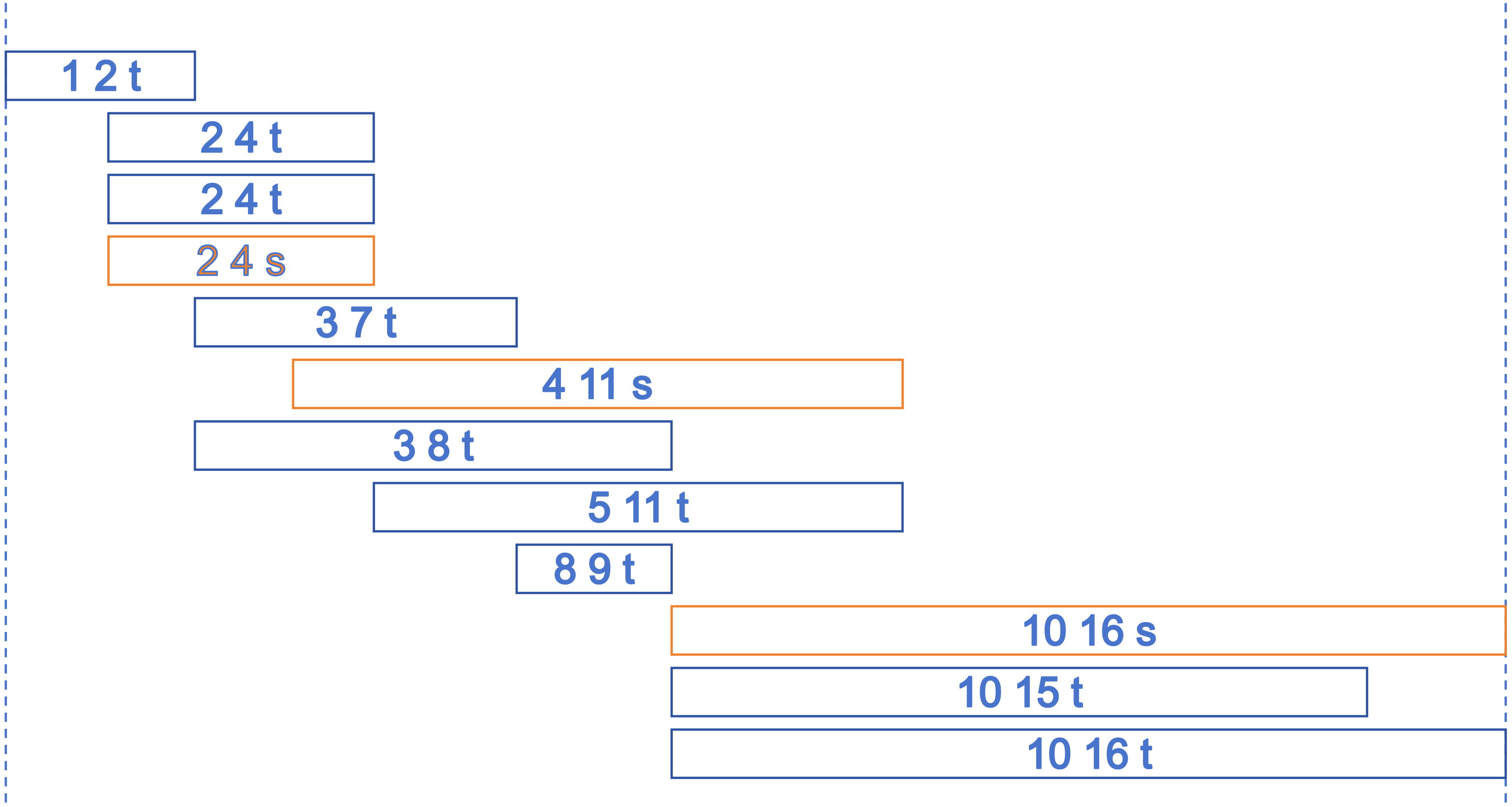

get(k)
query k in hashmap
exist
no
move to head
return -1
end


put(k,v)
query k in hashmap
no
exist
new node
node.value= v
insert to head
cache.size>capacity
move to head
delete last node
cache.size<=capacity
end

Lab3B

- You are given **n** tasks and **m** available time slots.
- Each task has a time interval **[l_i, r_i]**, and each time slot has an interval **[s_j, e_j]**.
- A task can be scheduled **only if it is completely contained** within a single time slot, i.e., **$l_i \geq s_j$** and **$r_i \geq e_j$**. A time slot can be reused for multiple tasks.
- Your goal is to **schedule as many tasks as possible**.
- The first line contains two integers **n** and **m** ($1 \leq n, m \leq 100{,}000$) — the number of tasks and time slots.
- The next **n** lines each contain two integers **l_i** and **r_i** ($1 \leq l_i < r_i \leq 1e9$), representing a task.
- The next **m** lines each contain two integers **s_j** and **e_j**($1 \leq s_j < e_j \leq 1e9$), representing a time slot.

Lab3B



1 2 t

2 4 t

2 4 t

2 4 s

3 7 t

4 11 s

3 8 t

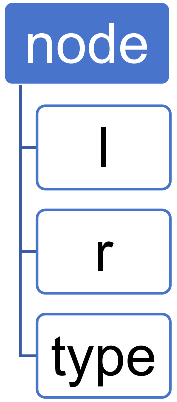
5 11 t

8 9 t

10 16 s

10 15 t

10 16 t



node

l

r

type

task->t; slot->s

Lab3B

1 2 t

2 4 s

2 4 t

2 4 t

3 8 t

3 7 t

4 11 s

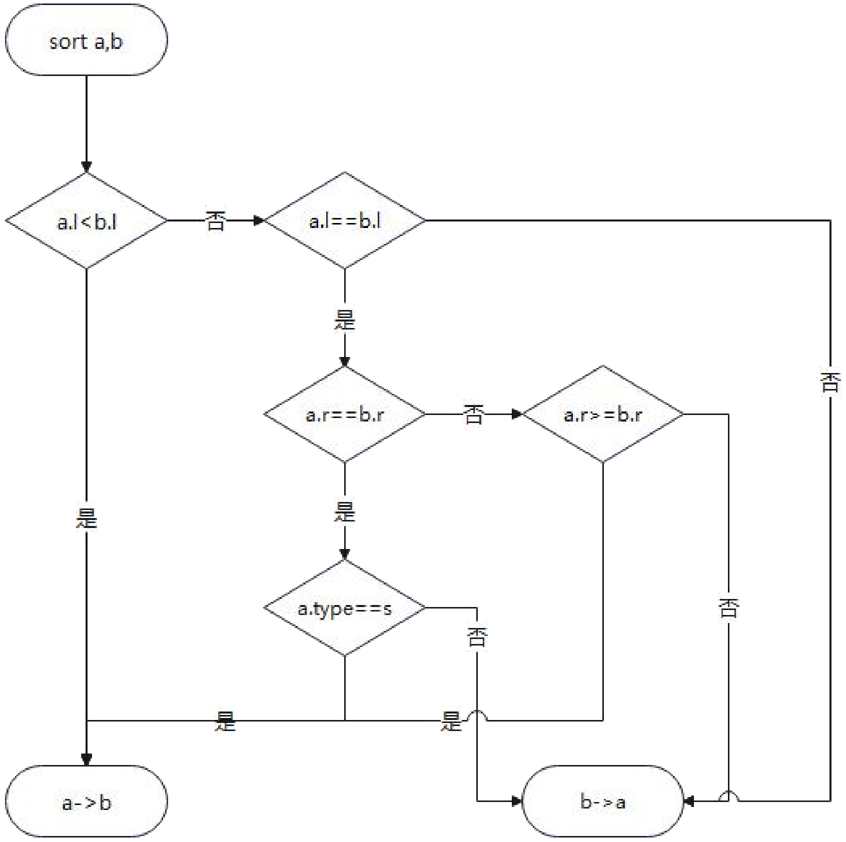
5 11 t

8 9 t

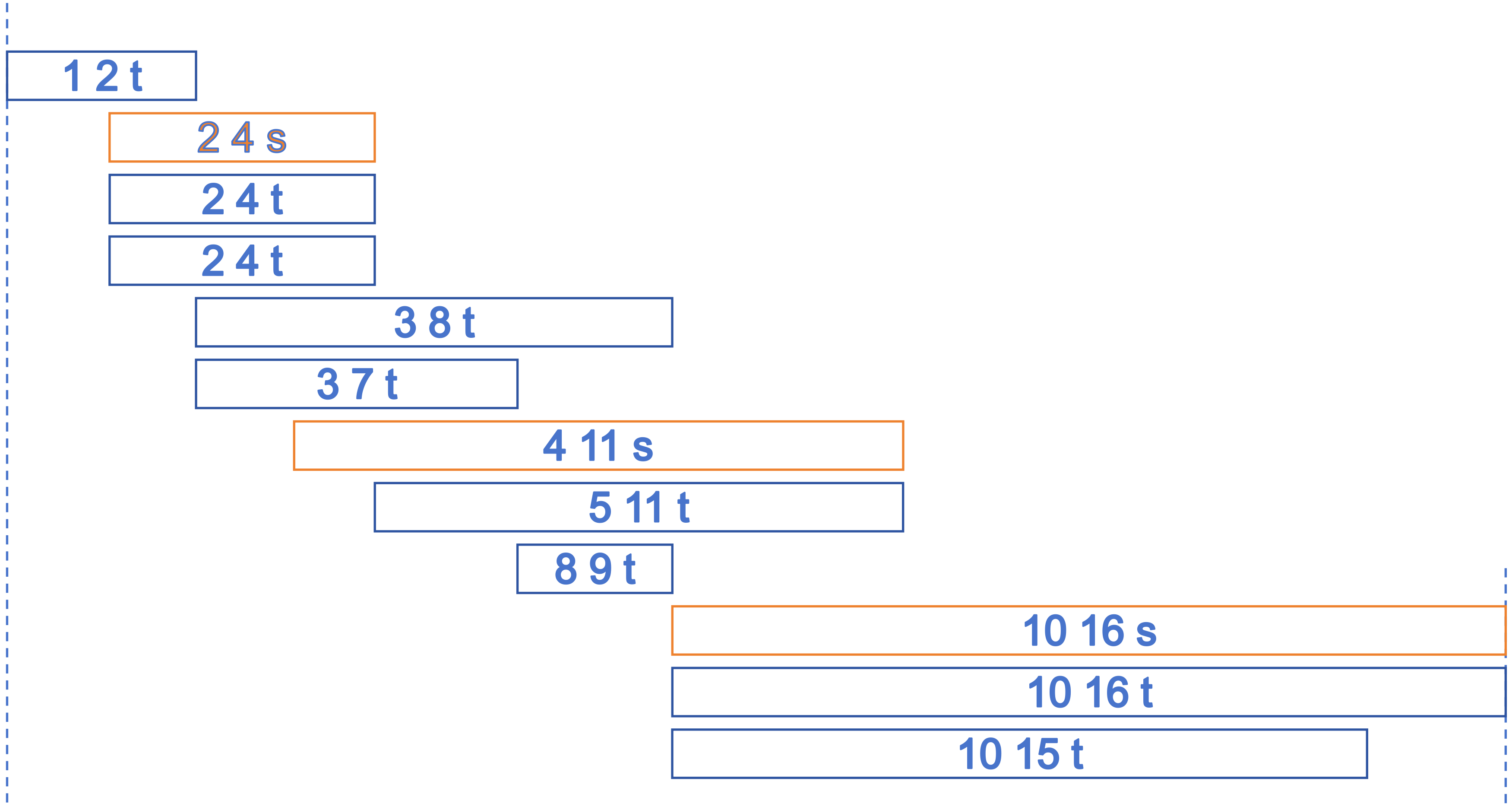
10 16 s

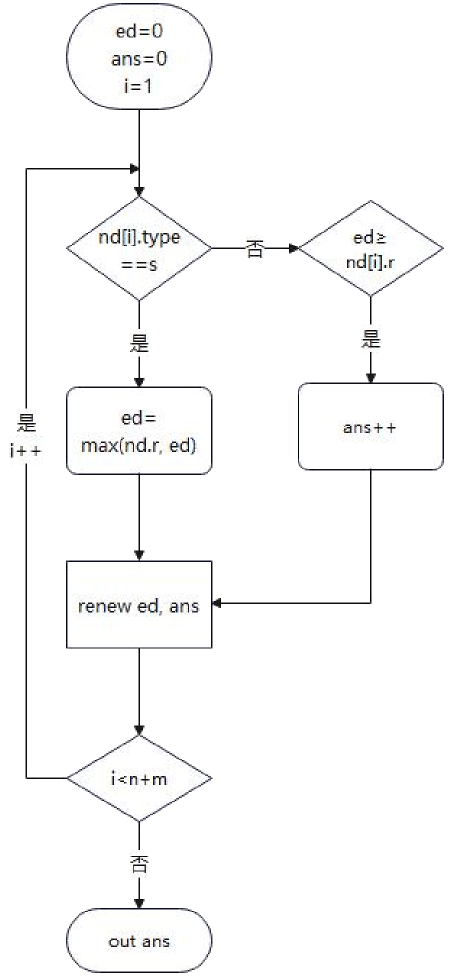
10 16 t

10 15 t



Lab3B


1 2 t
2 4 s
2 4 t
2 4 t
3 8 t
3 7 t
4 11 s
5 11 t
8 9 t
10 16 s
10 16 t
10 15 t


ed=0
ans=0
i=1
nd[i].type
==s
否
ed≥
nd[i].r
是
是
ed=
max(nd.r, ed)
ans++
是
i++
renew ed, ans
i<n+m
否
out ans