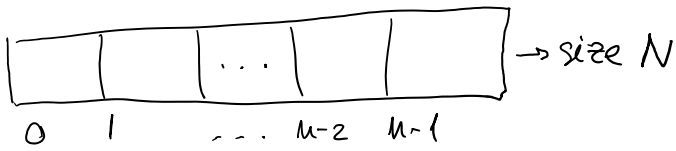
Seminar 03

Thursday, March 31, 2022 2:59 PM

Bucket Sort



$l: (k_1, v_1), (k_2, v_2)$

$k_i \in \mathbb{N}, i = \overline{0, N-1}$ (N keys)

- sorting a sequence of elements;
- a bucket will contain a sequence.

Space Complexity:
$O(N+m)$, where m = length of sequence

BC: $\Theta(N)$
WC: $\Theta(N+m)$
$\Rightarrow T: O(N+m)$

Time Complexity:
$\Theta(N+m)$

Operations of the sequence:

isEmpty(S): bool
first(S): element
removeFirst(S)
addLast(S, l)

Time Complexity: $\Theta(1)$

Implementation:

```
subalgorithm BucketSort(S, N) is:
    @allocate array B with size N with m sequences of elements
    while not isEmpty(S) execute:
        (k, v) ← first(S)
        removeFirst(S)
        addLast(B[k], (k, v))
    end while
```

```
for i←0,N-1 execute:
    while not isEmpty(B[i]) execute:
        (k,v)←first(B[i])
        removeFirst(B[i])
        addLast(S,(k,v))
    end_while
end_for
end_subalgorithm
```

- stability: Bucket Sort preserves the order of the elements with an equal key.

Lexicographic Sort

elements $(x_1, x_2, \dots, x_d)$; d - dimension

$(1,2,3), (4,5,6)$

$S: l_1, l_2, \dots, l_m$

Lexicographic order: $l_1 < l_2 \iff x_1^1 < x_1^2 \vee (x_1^1 = x_2^1 \wedge (x_2^1 < x_2^2 \vee \dots))$

$R: R_1, R_2, \dots, R_d$ - relations

```
subalgorithm LexicographicSort(S,d,R) is:
    for i←d,1,-1 execute:
        sort(S,Ri) @stable!!
    end_for
end_subalgorithm
```

Time Complexity: $\Theta(T_{stableSort} * d)$

Radix Sort

- variant of lexicographical sort;
- $l_i = (x_{i1}, x_{i2}, \ldots, x_{id}), \forall i, j\ x_{ij} \in \mathbb{N},\ x_{ij} = \overline{0, N-1}$

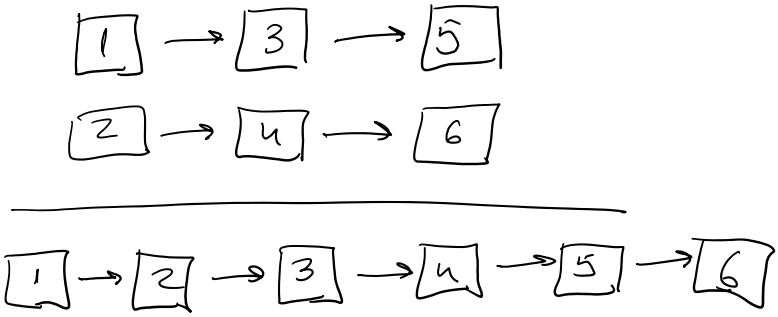
uses Bucket Sort is the stable sort

Time Complexity: $\Theta((m+N) * d)$

Merging 2 Sorted Linked Lists

Representation:

Node:
data TComp
next ↑Node



List:
head ↑Node
// relation

a) preserve the lists (create new list)
b) destroy the initial lists.

Implementation of b)

subalgorithm Merge(L_1, L_2, L_R) is:
    currentL1 ← L1.head
    currentL2 ← L2.head
    LR ← NIL

```
currentL2 ← L2.head
headLR ← NIL
tailLR ← NIL
while currentL1 ≠ NIL ∧ currentL2 ≠ NIL execute:
    if [currentL1].data < [currentL2].data then:
        if headLR = NIL then:
            headLR ← currentL1
            currentL1 ← [currentL1].next
        else:
            tailLR ← currentL1
            currentL1 ← [currentL1].next
        end_if
    else
        if headLR = NIL then:
            headLR ← currentL2
            currentL2 ← [currentL2].next
        else:
            tailLR ← currentL2
            currentL2 ← [currentL2].next
        end_if
    end_if
end_while
LR.head ← headLR
if currentL2 ≠ NIL then:
    [tailLR].head
```

[tailLR].head