Bucket-Sort

Let S be a list of n key-element items with keys in $[0, N-1]$.

Bucket-sort uses the keys as indices into auxiliary array B:

- the elements of B are lists, so-called buckets
- Phase 1:
  - empty S by moving each item (k, e) into its bucket $B[k]$
- Phase 2:
  - for $i = 0, \dots, N-1$ move the items of $B[k]$ to the end of S

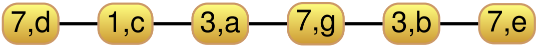
Performance:

- phase 1 takes $O(n)$ time
- phase 2 takes $O(n+N)$ time

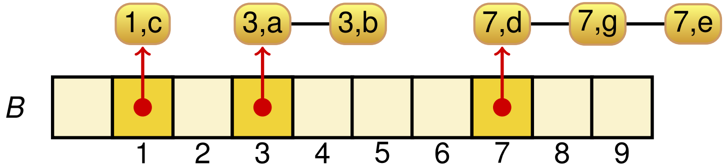
Thus bucket-sort is $O(n+N)$.

Bucket-Sort: Example

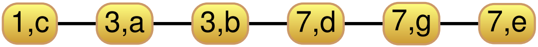
- key range $[0, 9]$



- Phase 1: filling the buckets



- Phase 2: emptying the buckets into the list



Bucket-Sort: Properties and Extensions

The keys are used as indices for an array, thus:

- keys should be numbers from $[0, N-1]$
- no external comparator

Bucket-sort is a stable sorting algorithm.

Extensions:

- can be extended to an arbitrary (fixed) finite set of keys D (e.g. the names of the 50 U.S. states)
- sort D and compute the rank rankOf(k) of each element
- put item (k, e) into bucket $B[\text{rankOf}(k)]$

Bucket-sort runs in $O(n + N)$ time:

- very efficient if keys come from a small intervall $[0, N-1]$ (or in the extended version from a small set D)

Lexicographic Order

A d-tuple is a sequence of d keys $(k_1, k_2, \ldots, k_d)$:

- k_i is called the i-th dimension of the tuple

Example: $(2, 5, 1)$ as point in 3-dimensional space

The **lexicographic order** of d tuples is recursively defined:

$$(x_1, x_2, \ldots, x_d) < (y_1, y_2, \ldots, y_d)$$

$$\Longleftrightarrow$$

$$x_1 < y_1 \vee (x_1 = y_1 \wedge (x_2, \ldots, x_d) < (y_2, \ldots, y_d))$$

That is, the tuples are first compared by dimension 1, then 2,...

Lexicographic-Sort

Lexicographic-sort sorts a list of d-tuples in lexicographic order:

- Let C_i be comparator comparing tuples by i-th dimension.
- Let stableSort be a stable sorting algorithm.

Lexicographic-sort executes d-times stableSort, thus:

- let $T(n)$ be the running time of stableSort
- then lexicographic-sort runs in $O(d \cdot T(n))$

Algorithm lexicographicSort(S):
Input: a list S of d-tuples
Output: list S sorted in lexicographic order

for $i = d$ **downto** 1 **do**
  stableSort(S, C_i)
done

Lexicographic-Sort: Example

$(7,4,6)$ — $(5,1,5)$ — $(2,0,6)$ — $(5,1,4)$ — $(2,1,4)$

$(5,1,4)$ — $(2,1,4)$ — $(5,1,5)$ — $(7,4,6)$ — $(2,0,6)$ dimension 3

$(2,0,6)$ — $(5,1,4)$ — $(2,1,4)$ — $(5,1,5)$ — $(7,4,6)$ dimension 2

$(2,0,6)$ — $(2,1,4)$ — $(5,1,4)$ — $(5,1,5)$ — $(7,4,6)$ dimension 1

Number representations

We can write numbers in different numeral systems, e.g.:

- 43_{10}, that is, 43 in decimal system (base 10)
- 101011_2, that is, 43 in binary system (base 2)
- 1121_3, that is, 43 represented base 3

For every base $b \geq 2$ and every number m there exist unique digits $0 \leq d_0, \ldots, d_l < b$ such that:

$$m = d_l \cdot b^l + d_{l-1} \cdot b^{l-1} + \ldots + d_1 \cdot b^1 + d_0 \cdot b^0$$

and if $l > 0$ then $d_l \neq 0$.

Example

$$\begin{aligned} 43 = 43_{10} &= 4 \cdot 10^1 + 3 \cdot 10^0 \\ = 101011_2 &= 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 \\ = 1121_3 &= 1 \cdot 3^3 + 1 \cdot 3^2 + 2 \cdot 3^1 + 1 \cdot 3^0 \end{aligned}$$

Radix-Sort

Radix-sort is specialization of lexicographic-sort:

- uses bucket-sort as stable soring algorithm
- is applicable if tuples consists of integers from $[0, N-1]$
- runs in $O(d \cdot (n+N))$ time

Sorting integers of fixed bit-length d in linear time:

- consider a list of n d-bit integers $x_{d-1}x_{d-2}\dots x_0$ (base 2)
- thus each integer is a d-tuple $(x_{d-1}, x_{d-2}, \dots, x_0)$
- apply radix sort with $N = 2$
- the runtime is $O(d \cdot n)$

For example, we can sort 32-bit integers in linear time.

Example

We sort the following list of 4-bit integers:

1001 — 0010 — 1101 — 0001 — 1110

0010 — 1110 — 1001 — 1101 — 0001

1001 — 1101 — 0001 — 0010 — 1110

1001 — 0001 — 0010 — 1101 — 1110

0001 — 0010 — 1001 — 1101 — 1110

Exercise C-4.14

Suppose we are given a sequence S of n elements each of which is an integer from $[0, n^2 - 1]$. Describe a simple method for sorting S in $O(n)$ time.

- Each number from $[0, n^2 - 1]$ can be represented by a two digit number in the number system with base n.

$$(n - 1) \cdot n + (n - 1) = n^2 - 1$$

- Conversion of each element into base-n is $O(1)$. ($O(n)$ for the whole list).
- Then use radix-sort to sort in $O(2 \cdot n)$, that is, $O(n)$ time.