316 11-10-09

Note Title 11/10/2009

<fun defs> → {<fun def>}$^+$

<fun def> → <header><body>

<header> → <fun name> "(" <parameters> ")"

<parameter list> → ∈ | <id> {, <id>}

- In each function, determine the leading token(s)
- Also in while-loops, termination conditions are determined by leading tokens of the first category repeated.

```
void funDefs()
{
  funDef();
  while(t is <id>)
    funDef();
}
```

```
void parameterList()
{
  if(t is <id>)
  {
    process {, <id>}
  }
}
```

N-dimensional General Case $[a_1 \dots b_1, \dots, a_n \dots b_n]$

$$\begin{aligned} rank([i_1, i_2, i_3, i_4]) = {} & (i_1 - a_1)(b_2 - a_2 + 1)(b_3 - a_3 + 1)(b_4 - a_4 + 1) + \\ & (i_2 - a_2)(b_3 - a_3 + 1)(b_4 - a_4 + 1) + \\ & (i_3 - a_3)(b_4 - a_4 + 1) + \\ & (i_4 - a_4) \end{aligned}$$

The Closed Form

$$rank([i_1, \dots, i_n]) = \sum_{1 \le k \le n} (i_k - a_k)(b_{k+1} - a_{k+1} + 1) \times \dots \times (b_n - a_n + 1)$$

$$= \sum_{1 \le k \le n} \left((i_k - a_k) \prod_{k+1 \le j \le n} (b_j - a_j + 1) \right)$$

$[0 \dots 9, 0 \dots 9, 0 \dots 9]$

$rank([4,2,7]) = 427$

$$rank([i_1, i_2, i_3]) = \sum_{1 \le k \le 3} \left((i_k - 0) \prod_{k+1 \le j \le 3} (9 - 0 + 1) \right)$$

$$= \sum_{1 \le k \le 3} \left(i_k \times \prod_{k+1 \le j \le 3} \right)$$

missed the rest

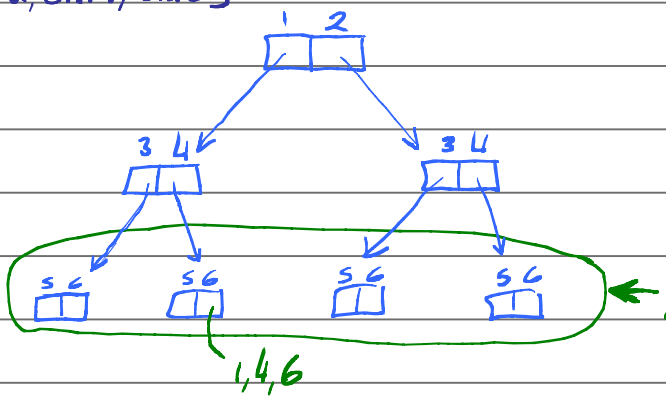
$$address([i_1,\dots,i_n]) = BA + \sum_{1\le k\le n}\left((i_k - a_k)\prod_{k+1\le j\le n}(b_j - a_j + 1)\right)\times ES$$

$$= \underbrace{BA - \sum_{1\le k\le n} a_k \prod_{k+1\le j\le n}(b_j - a_j + 1)\times ES}_{\text{virtual B.A.}} + \sum_{1\le k\le n} i_k \prod_{k+1\le j\le n}(b_j - a_j + 1)\times ES$$

Examples of Non-Contiguous Allocation

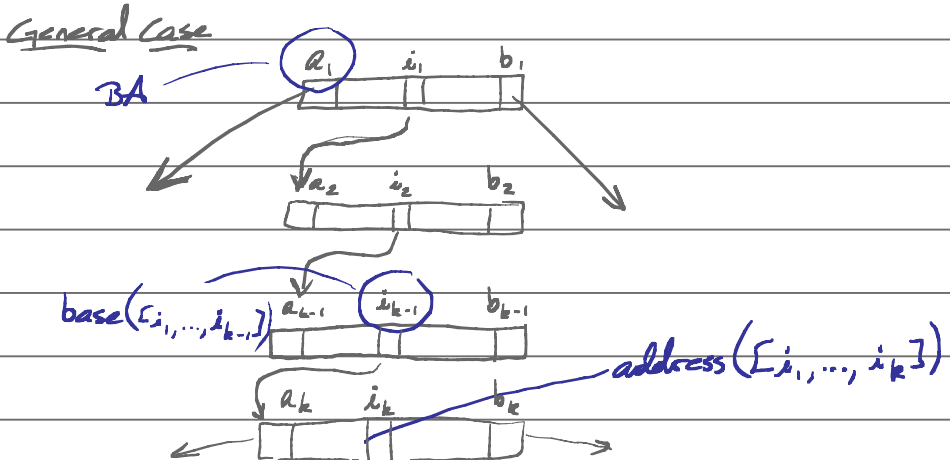
- Why?
  - The "heap" region may not have a long enough contiguous region to accomodate the whole arrays due to fragmentation
    - Defragmentation, garbage collection,
    - Non-contiguous allocation of the arrays
  - Variable-sized subarrays (dynamically created)
- Divide the whole arrays into one-dimensional slices and arrange them in the access tree form

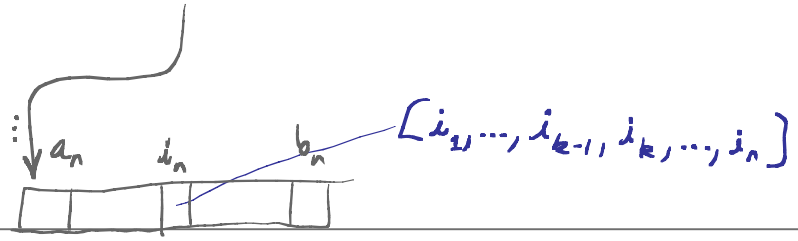
Eg = [1...2, 3...4, 5...6]



No longer random access

General Case





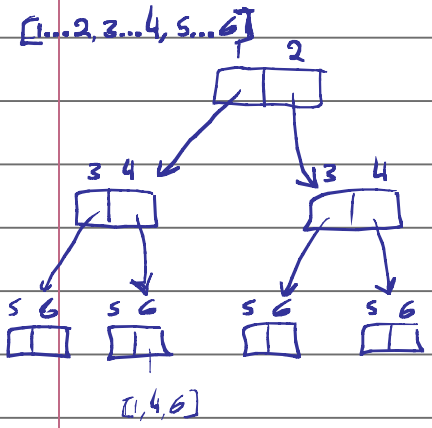
base([]) = BA = the start address of $[a_1 \ldots b_1]$

base($[i_1, \ldots, i_{k-1}]$) = the value of the cell at address ($[i_1, \ldots, i_{k-1}]$)

address($[i_1, \ldots, i_k]$) = base($[i_1, \ldots, i_{k-1}]$) + $(i_k - a_k) \times S$

$S = \begin{cases} ES & \text{if } k = n \\ AS & \text{if } k < n \end{cases}$

(AS = size of 1 reference pointer)



address([1,4,6]) = base([1,4]) + (6-5) × ES

= base([1,4]) + 1 × ES

base([1,4]) = the value of the cell at address ([1,4])

address([1,4]) = base([1]) + (4-3) × AS'

= base([1]) + 1 × AS'

n-1 pointers must be followed to access array elements

$\Theta(n)$

this method works even if b_k has different values