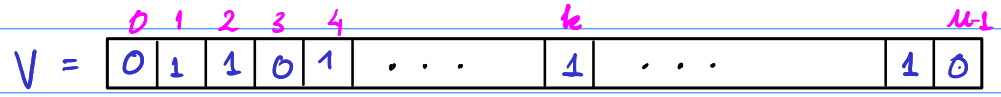
Van Emde Boas

Objetivo TAD p/ coleção dinâmica de inteiros no universo $\mathcal{U} = [u] = \{0, \ldots, u-1\}$ c/ operações:

- Inserção
- Remoção
- Sucessor / min := sucessor(-1)
- predecessor / max := predecessor(u)

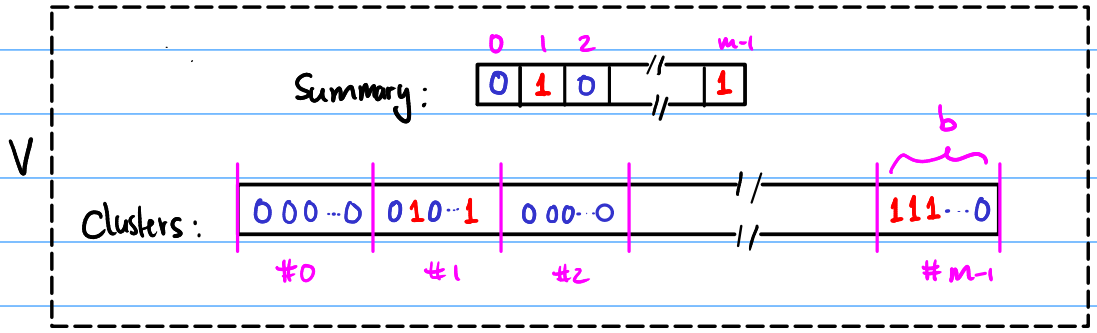
Bitvector



Inserção / Remoção $O(1)$ ✓

Sucessor $O(u)$ ⚠ → Busca linear pode varrer trechos com muitos 0's

Dividir em blocos e marcar blocos onde há 1's:



- Insenr/remover em V requer insenr/remover
  - no cluster
  - no sumário
- Sucessor (V, x):

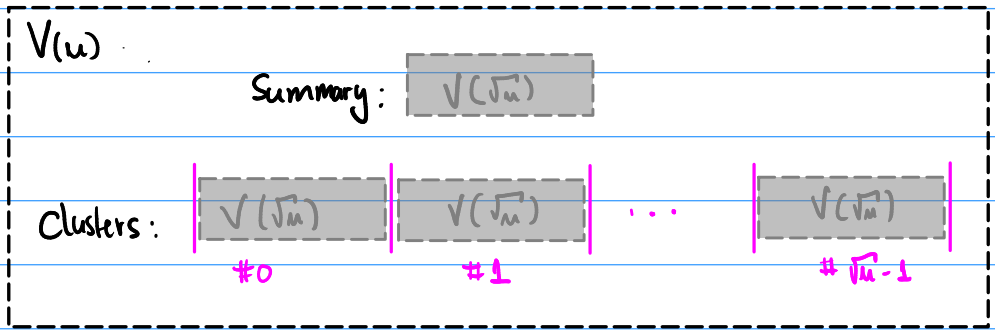
  procura sucessor dentro do bloco i onde está x

  Se não encontrar, procura no bloco j = próximo bloco com 1's

  ⚠ j = sucessor(V.summary, i)

➡ Cada bloco / sumário é um conjunto de inteiros num range menor com as mesmas operações

💡 Dividir em $m = \sqrt{u}$ blocos de tamanho $b = \sqrt{u}$ recursivamente
(por simplicidade $u = 2^{2^r}$, i.e. cada int em $\mathcal{U}$ tem $w = 2^r$ bits)



⚠ Caso base: $u = 2 \Rightarrow V(u)$ = bitvector de tamanho 2

```
insert(V, x):
    if V.u = 2:
        V.clusters[x] ← 1
        return
    h, ℓ ← high(V,x), low(V,x)
    if empty(V.clusters[h]):
        insert(V.summary, h)      // (1)
    insert(V.clusters[h], ℓ)      // (2)
```

```
high(V, x): // # cluster
    return ⌊x / √V.u⌋

low(V, x): // posição no cluster
    return x mod √V.u

index(V, h, ℓ):
    return h * √V.u + ℓ
```

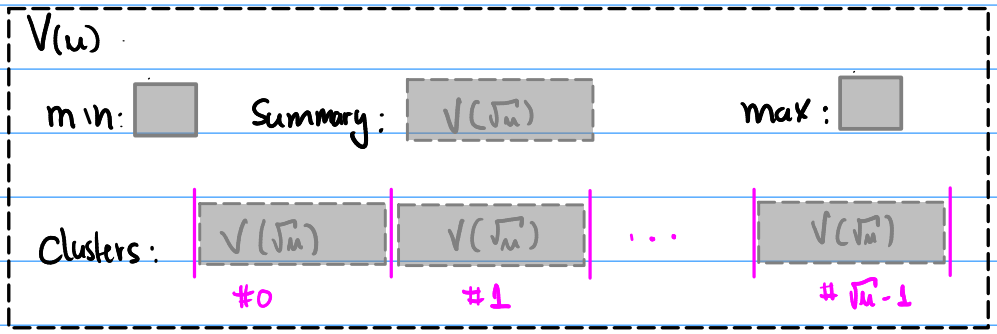
2 chamadas recursivas

$$T(u) = 2\,T(\sqrt{u}) + O(1)$$
$$= O(\lg u)$$

```
Successor (V, x):
    if V.u = 2:
        if x = -1 and V[0] return 0
        else if x ≤ 0 and V[1] return 1
        else return V.u    // = 2 sem sucessor
    h, l ← high(V,x), low(V,x)
    s ← successor(V.clusters[h], l)    // (1)
    if s ≠ V.clusters[h].u
        return index(V, h, s)    // sucessor no cluster de x
    else
        h ← Successor(V.summary, h)    // (2)
        if h = V.summary.u
            return V.u
        else
            s ← min(V.clusters[h])    // (3) min(V) = sucessor(V,-1)
            return index(V, h, s)
```

⚠ 3 chamadas recursivas! $T(u) = 3T(\sqrt{u}) + O(1)$
$= O((\lg u)^{\lg 3})$

- Guardar min em cada VEB p/ (3)
- Guardar max p/ decidir entre (1) ou (2)



- Caso base $u=2$ só precisa min/max
  - $[0,0] \rightarrow$ min = 2 max = -1
  - $[0,1] \rightarrow$ min = max = 0
  - $[1,0] \rightarrow$ min = max = 1
  - $[1,1] \rightarrow$ min = 0 max = 1

```
Successor (V, x):
    if x < V.min :
        return V.min
    if x > V.max :
        return V.u
    if V.u = 2 :
        if x = 0 and V.max = 1 :
            return 1
        else
            return 2
    h, l ← high(V,x), low (V,x)
    if l < V.clusters[h].max :          // sucessor no cluster de x
        s ← successor (V.clusters[h], l) // (1)
        return index (V, h, s)
    else                                // procura no prox cluster ocupado
        h ← Successor (V.summary, h)    // (2)
        if h = V.summary.u
            return V.u
        else
            s ← V.clusters[h].min       // (3) agora O(1)
            return index (V, h, s)
```

⚠ (1) XOR (2) chamado recursivamente

⇒ $T(u) = T(\sqrt{u}) + O(1) = O(\lg \lg u) \approx O(1)$ na prática ✓

⚠ Na inserção: Atualizar min/max em tempo $O(1)$

⇒ $T_{ins}(u) = O(\lg u)$

¿ Será que conseguimos $O(\lg\lg u)$?

Objetivo: ≤ 1 chamada recursiva

💡 Não armazenar min recursivamente

⇒ - 1ª inserção não faz chamadas recursivas e será $O(1)$

- Se insere #cluster no sumário antes da inserção recursiva é pq cluster estava vazio ⇒ inserção no cluster em $O(1)$

```
insert(V, x):
    if x = V.min or x = V.max
        return
    if V.u = 2:
        V.min ← min(V.min, x)
        V.max ← max(V.max, x)
        return
    if V.min = V.u:        // V vazio, armazena min=max=x não-rec.
        V.min, V.max ← x, x
        return
    if x < V.min:          // x é novo min
        swap V.min ↔ x     antigo min tem de ser armazenado recurs.
    V.max ← max(V.max, x)
    h, l ← high(V, x), low(V, x)
    if V.clusters[h].min = V.clusters[h].u:   // cluster vazio
        insert(V.summary, h)                 // (1)
    insert(V.clusters[h], l)                 // (2)  O(1) se (1) chamado
```

$T_{ins}(u) = T_{ins}(\sqrt{u}) + O(1) = O(\lg\lg u)$ ✓

Remoção

- Desfaz ops. da inserção em ordem reversa
- Caso particular ao remover min/max

```
Delete(V, x):
    if V.u = 2:
        if x = V.min:
            V.min ← V.max if V.max ≠ V.min else V.u(=2)
        if x = V.max:
            V.max ← V.min if V.min ≠ V.u else -1
        return
    if x = V.min:    // caso especial remove min que ñ está rec. armaz.
        i ← V.summary.min  // 1º cluster ñ-vazio (não conta com x=V.min)
        if i = V.summary.u  // não há outros elts em V além de V.min
            V.min, V.max ← V.u, -1  // apaga esse último valor e retorna
            return                  em tempo O(1) (*)
        else    // "puxa" sucessor do min para novo min
            V.min ← V.index(i, V.clusters[i].min) // novo min ok em V
            x ← V.min  // clroca em x p/ apagá-lo recursivamente
    h, l ← high(V,x), low(V,x)
    Delete(V.clusters[h], l)    // (1)
    if V.clusters[h].min = V.clusters[h].u:  // cluster agora vazio
        Delete(V.summary, h)    // (2)
    if x = V.max:  // se removeu max, procura predecessor p/ atualizar
        i ← V.summary.max // Note: x já rec. removido do cluster e summary
        if i = -1          // não há outros elts exceto V.min
            V.max ← V.min
        else
            V.max ← V.index(i, V.clusters[i].max) // novo max ok!
```

⚠ Se (2) foi chamada então cluster acabou de ficar vazio
⇒ a chamada (1) acima acabou de apagar o último elto = min em tempo $O(1)$ (*)

↳ $T_{del}(u) = T_{del}(\sqrt{u}) + O(1) = O(\lg\lg u)$ ✓

Espaço

Visualizando como árvore:
- Cada nó tem $\sqrt{u}+1 \geq 3$ filhos de tamanho $\sqrt{u}$ (clusters + summ.)
- Cada nó guarda min, max e ptrs p/ os filhos

$\therefore S(u) = (\sqrt{u}+1)\, S(\sqrt{u}) + \Theta(\sqrt{u})$
$= O(u)$ ⚠

💡 Armazenar apenas os clusters/summaries não vazios
↳ Usar hashtables de clusters ao invés de arrays de clusters
[Cormen et al. Intr. to Algorithms 3rd ed Problem 20.1]

Recorrências

① $T(n) = 2T(\sqrt{n}) + \Theta(1)$

$T(n) = \Theta(1)$ $= \Theta(1)$ $= \dots$

$T(n^{1/2})$ $T(n^{1/2})$ — $\Theta(1)$ $\Theta(1)$ — $T(n^{1/4})$ $T(n^{1/4})$ $T(n^{1/4})$ $T(n^{1/4})$

$= \Theta(1)$

$\Theta(1)$ $\Theta(1)$

$\Theta(n^{1/2^h})$ $\dots$ $\Theta(n^{1/2^h})$ — h

$n^{1/2^h} = 2$ (caso base univ = 2)

$\Rightarrow \frac{1}{2^h} = \log_n 2 = \frac{\lg 2}{\lg n} = \frac{1}{\lg n}$

$\Rightarrow 2^h = \lg n \Rightarrow h = \lg \lg n$

$\therefore T(n) = \Theta(1 + 2 + 4 + \dots + \lg \lg n) = \Theta(2^{\lg \lg n}) = \Theta(\lg n)$

② $T(n) = T(\sqrt{n}) + \Theta(1)$

$= T(n^{1/2}) + \Theta(1)$

$= T(n^{1/4}) + \Theta(1) + \Theta(1)$

$= T(n^{1/8}) + \Theta(1) + \Theta(1) + \Theta(1) = \dots$

$= T(n^{1/2^h} = 2) + h * \Theta(1)$ // $h = \lg \lg n$

$= \Theta(\lg \lg n)$

③ $S(u) = (\sqrt{u}-1)\,S(\sqrt{u}) + \Theta(\sqrt{u})$ $\quad [u = 2^{2^w} \text{ p/ algum } w]$

Seja b a cte do termo $\Theta(\sqrt{u})$ da expressão, i.e.

$S(u) \leq (\sqrt{u}-1)\,S(\sqrt{u}) + b\sqrt{u}$

Af $\exists$ cte $a > b$ t.q $S(u) \leq a \cdot u$

Por indução

Base: $u = 2$. Trivial pois $S(2)$ tem espaço cte

Passo:

$$\begin{aligned} S(u) &= (\sqrt{u}-1)\,S(\sqrt{u}) + \Theta(\sqrt{u}) \\ &\leq \sqrt{u}\,S(\sqrt{u}) + b\sqrt{u} \\ &\leq \sqrt{u} \cdot a \cdot \sqrt{u} + b\sqrt{u} && \text{(h.i.)} \\ &\leq a \cdot u + a\sqrt{u} && (a > b) \\ &\leq 2 \cdot a \cdot u \quad \square \end{aligned}$$

$\therefore S(u) = \mathcal{O}(u)$