# Van Emde Boasova stabla

**Koturić, Ana**

Sveučilište J. J. Strossmayera u Osijeku
Odjel za matematiku
Sveučilišni preddiplomski studij matematike i računarstva

Ana Koturić

# Van Emde Boasova stabla

Završni rad

Osijek, 2022.

Sveučilište J. J. Strossmayera u Osijeku
Odjel za matematiku
Sveučilišni preddiplomski studij matematike i računarstva

Ana Koturić

# Van Emde Boasova stabla

Završni rad

Mentor: doc. dr. sc. Slobodan Jelić

Osijek, 2022.

## Sažetak

U ovom radu bavimo se strukturom podataka koja se zove van Emde Boasovo stablo. Najprije opisujemo neke jednostavnije strukture podataka, a potom proto van Emde Boasovo stablo te van Emde Boasovo stablo. Definirat ćemo naredbe za proto van Emde Boasovo stablo i van Emde Boasovo stablo te dati njihove pseudokodove.

## Ključne riječi

van Emde Boasovo stablo, proto van Emde Boasovo stablo, strukture podataka, vremenska složenost

# Van Emde Boas Trees

## Abstract

This paper deals with the van Emde Boas Tree data structure. Firstly, it describes simpler data structures, then the proto van Emde Boas Tree, and lastly the van Emde Boas Tree data structures. The paper defines operations on the proto van Emde Boas Tree and for the van Emde Boas Tree and also gives pseudocodes for each of the operations.

## Keywords

van Emde Boas Tree, proto van Emde Boas Tree, data structures, time complexity

# Sadržaj

# Uvod

Van Emde Boasovo stablo je struktura podataka koja se prvi put spominje 1975. godine u radu nizozemskog znanstvenika i profesora Petera van Emde Boasa, prema kojem je i dobila ime.

To je struktura podataka kojoj je gornja asimptotska međa za izvršenje svih naredbi (search, insert, delete, minimum, maximum, succesor, predecessor) jednaka $O(\lg \lg n)$, što je brže nego kod ostalih sličnih struktura podataka. Bitno je napomenuti da kod ove strukture podataka ključevi moraju biti cijeli brojevi i da ponavljanje istih ključeva nije moguće.

Kako bismo što bolje mogli razumijeti van Emde Boasovo stablo, u ovom radu ćemo prvo preko jednostavnijih struktura upoznati strukturu podataka koju zovemo proto van Emde Boasovo stablo. Proto van Emde Boasovo stabla je struktura koja je vrlo blizu onome što trebamo postići da imamo strukturu podataka koja sve operacije izvodi u vremenu, $O(\lg \lg n)$, a onda njenim daljnim modifikacijama dolazimo upravo do van Emde Boasovog stabla.

# 1 Pozadina

## 1.1 Pretpostavke i zahtjevi

**Definicija 1.1.** *Neka je $S$ dinamički skup, $n$ je broj elemenata koji se trenutno nalaze u dinamičkom skupu $S$, a $u$ je raspon mogućih vrijednosti za elemente skupa $S$. Skup $\{0, 1, 2, ..., u-1\}$ nazivamo svemir(eng. univers), a $u$ veličina svemira(eng. univers size). Također ćemo pretpostaviti da je $u$ oblika $u = 2^k$, gdje je $k \geq 1$ cijeli broj.*

Želimo konstruirati stukturu podataka koja će nad skupom $S$ iz našeg svemira sljedeće naredbe izvršavati u vremenu $O(\lg \lg u)$ u najgorem slučaju:

$MEMBER(S, x)$: provjerava je li $x$ element skupa $S$

$MINIMUM(S)$: vraća najmanji element skupa $S$

$MAXIMUM(S)$: vraća najveći element skupa $S$

$SUCCESSOR(S, x)$: vraća najmanji element skupa $S$ koji je veći od $x$

$PREDECESSOR(S, x)$: vraća najveći element skupa $S$ koji je manji od $x$

$INSERT(S, x)$: dodaje element $x$ u skup $S$

$DELETE(S, x)$: briše element $x$ iz skupa $S$.

## 1.2 Početna rješenja

### Polje bitova

Dinamički skup $S$ iz svemira $\{0, 1, ..., u-1\}$ možemo spremiti u neko polje $A[0..u-1]$ s $u$ bitova u kojem bilježimo prisutnost elemenata s 1 i 0. Bit $A[x]$ imat će vrijednost 1 kada je x element dinamičkog skupa $S$, a vrijednost 0 kada $x$ nije element skupa $S$.

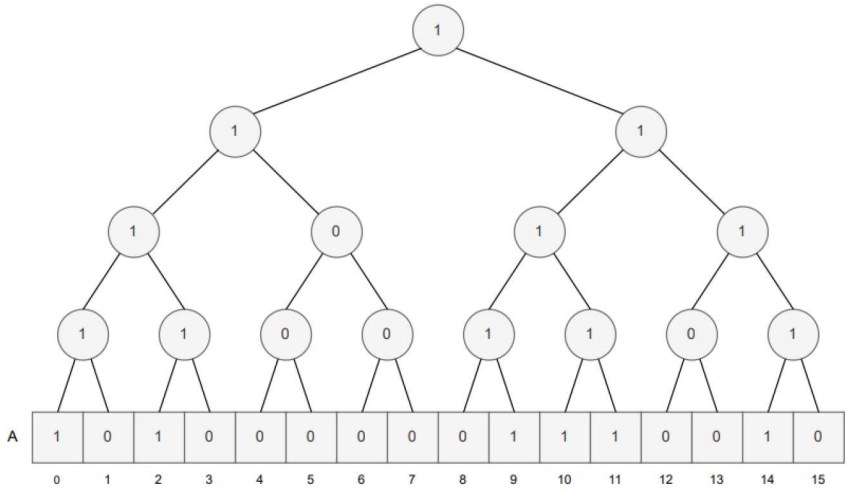
| A | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |

Slika 1: *Polje bitova koje prikazuje dinamički skup $\{0, 2, 9, 10, 11, 14\}$ kada je veličina svemira $u = 16$. Bit na poziciji $x$ je jednak 1 ako i samo ako je $x$ element dinamičkog skupa, a u suprotnom je taj bit jednak 0.*

Kod ovog rješenja operacije MEMBER, INSERT i DELETE izvršit će se u $O(1)$ vremenu, ali zato će operacije MINIMUM, MAXIMUM, SUCCESSOR i PREDECESSOR u najgorem slučaju trebati $\Theta(u)$ vremena jer ćemo u nekim situacijama morati proći kroz cijelo polje.

## Binarno stablo

Kako bismo izbjegli prolaske kroz cijelo polje možemo kreirati binarno stablo u kojem će listove stabla činiti upravo polje bitova $A$. Svaki bit u polju predstavlja jedan list, roditeljski čvor bit će jednak 1 ako i samo ako je barem jedan od čvorova djece jednak 1. Kod ovakve strukture svaki roditeljski čvor je zapravo logičko ili svoje djece.




Slika 2: *Binarno stablo konstruirano iznad polja bitova koje prikazuje dinamički skup* $\{0, 2, 9, 10, 11, 14\}$ *kada je veličina svemira* $u = 16$.

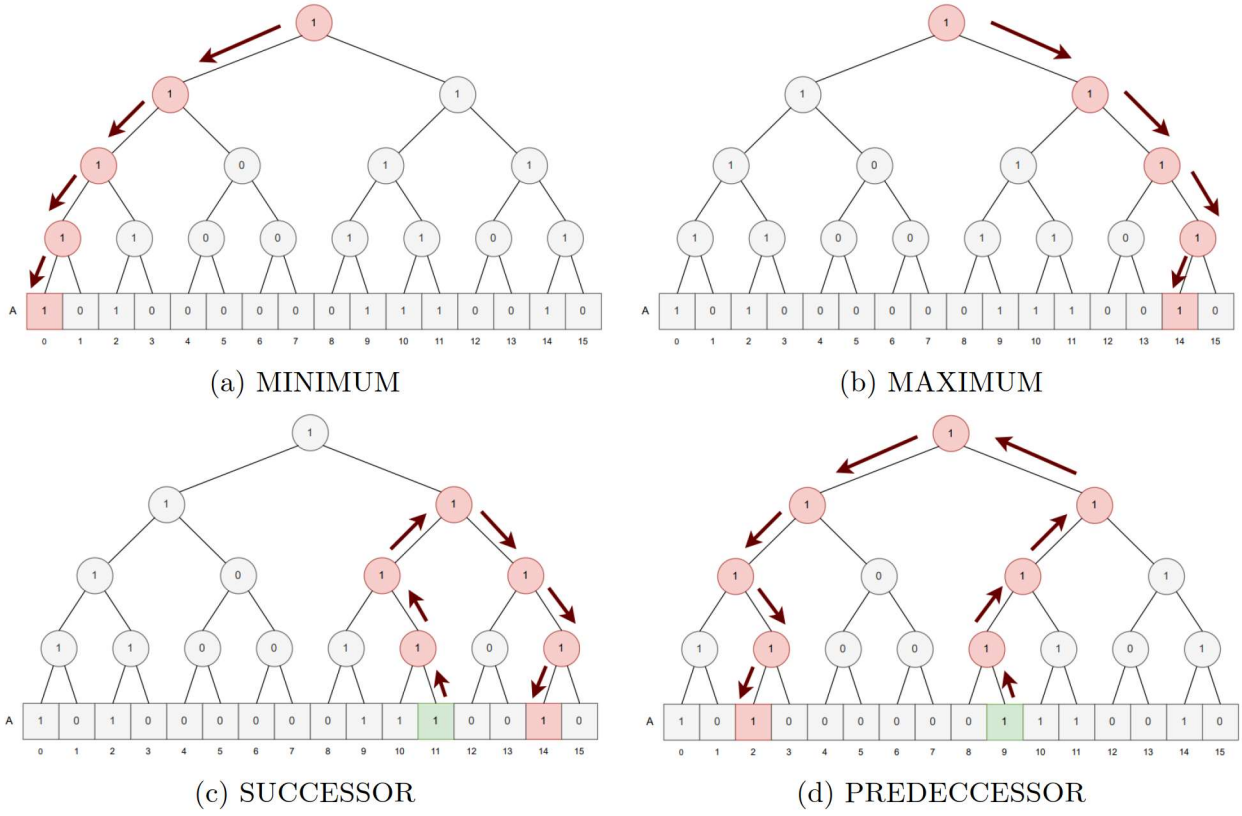
Ovakvom strukturom optimizirali smo vrijeme izvršenja operacija MINIMUM, MAXIMUM, SUCCESSOR i PREDECESSOR koje se izvode na sljedeći način:

MINIMUM: započinjemo iz korijena te prolazimo kroz stablo dok ne dođemo do lista na način da uvijek biramo lijevo dijete ako je ono jednako 1, a u suprotnom biramo desno dijete. List do kojeg dođemo je minimum.

MAXIMUM: započinjemo iz korijena te se spuštamo kroz stablo dok ne dođemo do lista na način da uvijek biramo desno dijete ako je ono jednako 1, a u suprotnom biramo lijevo dijete. List do kojeg dođemo je maksimum.

SUCCESSOR: započinjemo iz lista koji se nalazi na poziciji x te se penjemo prema korijenu dok s lijeve strane ne dođemo do roditelja čije desno dijete ima vrijednost 1, spustimo se do tog djeteta i nastavimo se spuštati birajući uvijek lijevo dijete ako je ono jednako 1, u suprotnom biramo desno dijete, kada dođemo do lista pronašli smo sljedbenika od x.

PREDECESSOR: započinjemo iz lista koji se nalazi na poziciji x te se penjemo prema korijenu dok s desne strane ne dođemo do roditelja čije je lijevo dijete jednako 1, spustimo se do tog djeteta i nastavimo se spuštati birajući uvijek desno dijete ako je ono jednako 1,u suprotnom biramo lijevo dijete, kada smo došli do lista pronašli smo prethodnika od x.



(a) MINIMUM (b) MAXIMUM

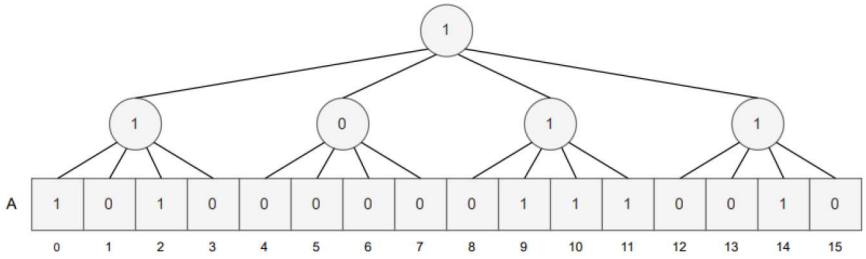
(c) SUCCESSOR (d) PREDECCESSOR

Slika 3: *Strjelice prikazuju putanja za pronalazak:* ***(a)*** *minimuma skupa* $\{0, 2, 9, 10, 11, 14\}$. ***(b)*** *maksimuma skupa* $\{0, 2, 9, 10, 11, 14\}$. ***(c)*** *sljedbenika broja 11 iz skupa* $\{0, 2, 9, 10, 11, 14\}$. ***(d)*** *prethodnika broja 9 iz skupa* $\{0, 2, 9, 10, 11, 14\}$.

Za svemir veličine $u$ imat ćemo binarno stablo visine $\lg u$ i tada će za izvršenje svake od gornjih naredbi trebati najviše $O(\lg u)$ vremena, jer ćemo za izvršenje svake od naredbi imati najviše jedan prolaz prema gore i jedan prolaz prema dolje kroz stablo.

## Stablo stupnja $\sqrt{u}$

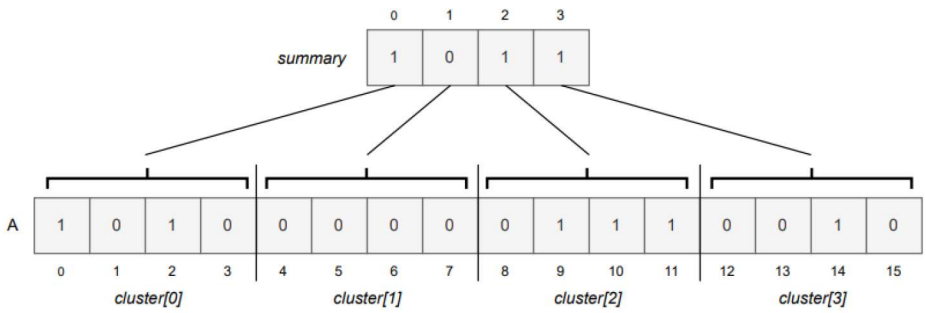
Pretpostavit ćemo da je veličina svemira $u = 2^{2k}$ za neki $k$ iz skupa cijelih brojeva, time smo osigurali da je $\sqrt{u}$ uvijek cijeli broj, što nam je bitno kako bismo mogli kreirati stablo stupnja $\sqrt{u}$ čija će konačna visina uvijek biti 2.



Slika 4: *Stablo stupnja* $\sqrt{16}$ *za veličinu svemira* $u = 16$*, za dinamički skup* $\{0, 2, 9, 10, 11, 14\}$.

Kao i kod binarnog stabla, listove stabla čine bitovi iz polja bitova $A$. Na dubini 1 imamo $\sqrt{u}$ čvorova i svaki od tih čvorova ima $\sqrt{u}$ djece te je on jednak logičkom ILI svoje djece. Korijen stabla je logičko ILI čvorova koji se nalaze na dubini 1. Vidi Slika 4.

Polje bitova možemo podijeliti na $\sqrt{u}$ potpolja, odnosno klastera (*eng. cluster*), s $\sqrt{u}$ elemenata. Onda čvorove na dubini 1 možemo prikazati kao polje bitova $summary[0 \dots \sqrt{u} - 1]$, gdje će $summary[i]$ biti 1, ako i samo ako $cluster[\lfloor i/\sqrt{u} \rfloor]$ na nekoj od pozicija ima 1.




Slika 5: *Stablo stupnja* $\sqrt{u}$, $u = 16$, *za dinamički skup* $\{0, 2, 9, 10, 11, 14\}$, *gdje smo čvorove na dubini 1 prikazali kao polje bitova* $summary[0 \dots \sqrt{u} - 1]$, *a* $summary[i]$ *je logičko ili elemenata koji se nalaze u* $cluster[i]$.

Jesmo li ovakvom strukturom poboljšali brzinu izvršenja naredbi MINIMUM, MAXIMUM, SUCCESSOR i PREDECCESOR u odnosu na njihove brzine kada koristimo binarno stablo? Prema [1] za izvršenje svake od gore navedenih naredbi bit će potrebno $O(\sqrt{u})$ vremena u najgorem slučaju, što je znatno lošije nego kod binarnog stabla, gdje je potrebno $O(\lg u)$ vremena u najgorem slučaju. Za implementaciju navedenih naredbi pogledaj [1, str. 535]. Iako ovom strukturom podataka nismo uspjeli poboljšati vrijeme izvršenja naredbi, ona će nam biti ideja za daljnje definiranje van Emde Boasovog stabla.

# 2 Proto van Emde Boasovo stablo

## 2.1 Rekurzivna definicija proto van Emde Boasovog stabla

U prethodnom poglavlju definirali smo stablo stupnja $\sqrt{u}$, a sada ćemo modificirati tu ideju tako što ćemo rekurzivno definirati struktura podataka kod koje na svakom levelu rekurzije smanjujemo veličinu svemira $u$ za $\sqrt{u}$ puta. Ako započnemo sa svemirom veličine $u$, korijenski čvor će biti struktura koja će sadržavati cijeli svemir, svaki čvor na dubini 1 će biti struktura koja će sadržavati $\sqrt{u} = u^{1/2}$ elemenata svemira, svaki čvor na dubini 2 će biti struktura koja će sadržavati $\sqrt{\sqrt{u}} = u^{1/4}$ elemenata svemira i tako dalje dok ne dođemo do dubine na kojoj će svaki čvor biti struktura koja sadrži samo 2 elementa svemira.

Radi jednostavnosti pretpostavit ćemo da je $u$ oblika $u = 2^{2^k}$, gdje je $k$ cijeli broj, što će nam omogućiti da su $u, u^{1/2}, u^{1/4}, u^{1/8}, ...$ uvijek cijeli brojevi.

**Teorem 2.1.** *Vremenska složenost opisana jednadžbom*

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

*jednaka je* $O(\lg \lg u)$.

*Dokaz.* Primjenom metode zamjene varijabli dobivamo da je $T(u) = O(\lg \lg u)$ (za postupak vidi [1, str 536-537]). □

Vremenska složenost opisana u prethodnom teoremu je upravo vremenska složenost koju želimo postići van Emde Boasovim stablom. Ako promotorimo veličine svemira na svakoj razini naše rekurzivne strukture, dobijemo niz $u, u^{1/2}, u^{1/4}, u^{1/8}, ...$, na nultoj razini rekurzije potrebno nam je $\lg u$ bitova da spremimo cijeli svemir, a na svakoj idućoj razini rekurzije potrebno je upola manje bitova nego na prijašnjoj razini. Što znači ako smo počeli s $b$ bitova, onda ćemo nakon $\lg b$ razina rekurzije doći do 1 bita, a kako je $b = \lg u$, to znači da nakon $\lg \lg u$ razina imamo svemir veličine 2, odnosno visina našeg stabla je $\lg \lg u$, pa i vremenska složenost $O(\lg \lg u)$ ima smisla.

Ako pogledamo Slika 5 iz prethodnog poglavlja možemo uočiti da se element $x$ nalazi u klasteru broj $\lfloor x/\sqrt{u} \rfloor$, odakle imamo sljedeće definicije:

**Definicija 2.1.** *Neka je $x$ element dinamičkog skupa $S$ i $u$ veličina svemira. Funkcija* ***high(x)*** *dana s*

$$high(x) = \lfloor x/\sqrt{u} \rfloor$$

*govori nam u kojem se klasteru nalazi element $x$.*

**Definicija 2.2.** *Neka je $x$ element dinamičkog skupa $S$ i $u$ veličina svemira. Funkcija* ***low(x)*** *dana s*

$$low(x) = x \mod \sqrt{u}$$

*govori nam na kojoj poziciji u klasteru se nalazi element $x$.*

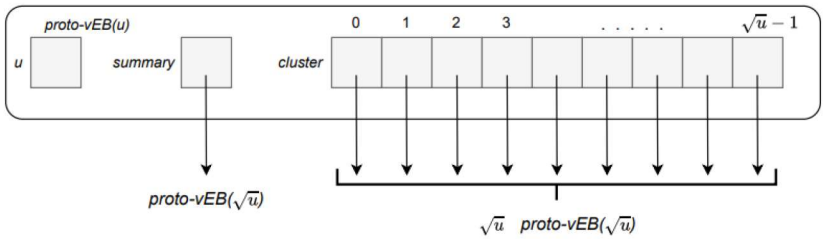
**Definicija 2.3.** *Neka $i$ predstavlja broj klastera, $j$ poziciju u tom klasteru i neka je $u$ veličina svemira. Funkcija* ***index(i,j)*** *dana s*

$$index(i, j) = i\sqrt{u} + j$$

*govori nam koji se element nalazi u klasteru broj $i$ na poziciji $j$.*

## 2.2 Struktura proto van Emde Boasovog stabla

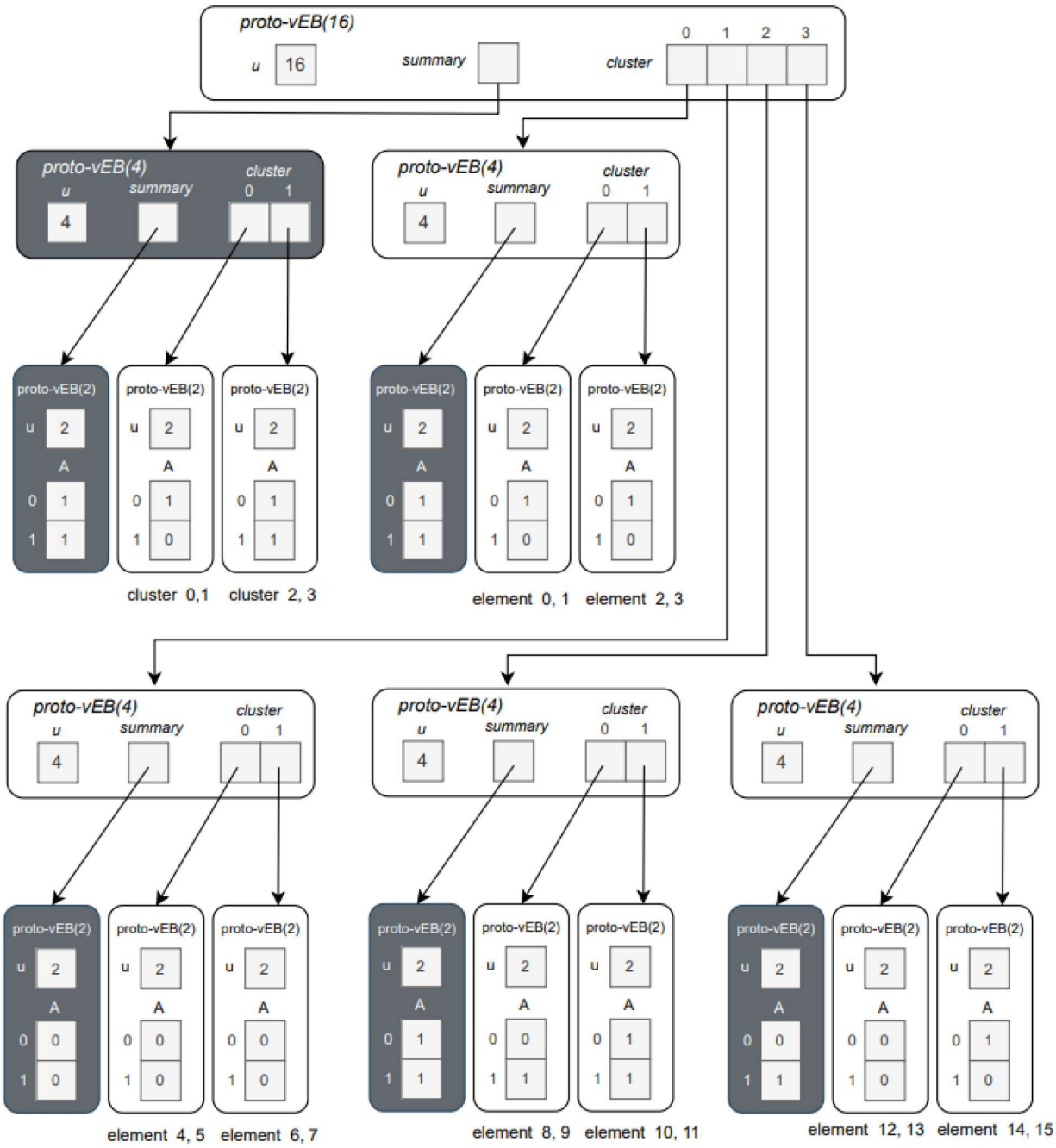
**Definicija 2.4.** *Za svemir* $\{0, 1, \ldots, \sqrt{u}-1\}$ *rekurzivno definiramo* ***proto van Emde Boasovo stablo****, koje označavamo s* ***proto-vEB(u)*** *gdje atribut u označava veličinu svemira. Proto van Emde Boasovo stablo sadrži:*
*a) Pokazivač* ***summary*** *na proto-vEB*($\sqrt{u}$) *i polje pokazivača* ***cluster[0 . . $\sqrt{u}$-1]*** *s* $\sqrt{u}$ *pokazivača na* $proto - vEB(\sqrt{u})$*, kada je* $u = 2^{2^k} \neq 2$.
*b) Polje bitova* ***A[0 . . 1]*** *koje se sastoji od dva bita, kada je* $u = 2^{2^k} = 2$.




Slika 6: *Grafički prikaz sadržaja* $proto - vEB(u)$ *strukture za* $u = 2^{2^k} \neq 2$.

Neki element $x$ za koji vrijedi $0 \leq x \leq u$, rekurzivno će biti spremljen u klasteru broj $high(x)$ na poziciji $low(x)$, pri ćemu su funkcije $high(x)$ i $low(x)$ definirane kao u prethodnom potpoglavlju.

Pogledajmo Slika 7 koja prikazuje potpuno raspisanu $proto-vEB(16)$ strukturu za dinamički skup $\{0, 2, 9, 10, 11, 14\}$. $proto-vEB(16)$ struktura ima četiri pokazivača na $proto-vEB(4)$ sturkture, koja su spremljena u polje $cluster[0\,..\,3]$ te jedan pokazivač na *summary* koji je također $proto - vEB(4)$ sturktura. Svaka $proto - vEB(4)$ struktura ima dva pokazivača na $proto - vEB(2)$ strukture koji su spremljeni u polje $cluster[0\,..\,1]$ i pokazivač na *summary* koji je također $proto - vEB(2)$ struktura. A svaka $proto - vEB(2)$ struktura sadrži polje $A[0\,..\,1]$ koje se sastoji od dva bita. Tamnije obojane $proto-vEB(u)$ strukture su *summary* strukture na koje pokazuje pokazivač iz polja *summary*, dok su svijetlije obojane $proto - vEB(u)$ strukture, strukture na koje pokazuju pokazivači iz polja *cluster*. Sve $proto - vEB(2)$ strukture označene s "element i, j" spremaju bitove $i$ i $j$ iz polja bitova $A[0\,..\,15]$, dok $proto-vEB(2)$ strukture označene s "cluster i, j" spremaju *summary* bitove za klastere $i$ i $j$ iz $proto - vEB(16)$ strukture.




Slika 7: *Primjer potpuno raspisane proto−vEB*(16) *strukture, koja prikazuje dinamički skup* $\{0, 2, 9, 10, 11, 14\}$.

## 2.3 Naredbe za proto van Emde Boasovo stablo

### 2.3.1 Član proto van Emde Boasovog stabla

Za $proto - vEB$ strukturu $V$ i vrijednost $x$, definiramo naredbu proto-vEB-MEMBER$(V, x)$ koja provjerava je li vrijednost $x$ prisutna u dinamičkom skupu $S$ koji je pohranjen u $proto - vEB$ $V$ strukturi.

**Algoritam 1** proto-vEB-MEMBER$(V, x)$

```
1 if V.u == 2 then
2     return V.A[x]
3 else
4     return proto-vEB-MEMBER(V.cluster[high(x)], low(x))
```

Kako bismo provjerili je li vrijednost $x$ element dinamičkog skupa $S$ moramo pronaći bit u odgovarajućoj $proto - vEB(2)$ strukturi koji predstavlja vrijednost $x$. Upravo prva linija naredbe provjerava je li $V$ $proto - vEB(2)$ struktura,a ukoliko je, onda je $V.A[x]$ bit koji tražimo i druga linija ga vraća. Ako $V$ nije $proto - vEB(2)$ struktura u trećoj liniji ulazimo u rekurziju, kroz koju prolazimo dok ne dođemo do $proto - vEB(2)$ strukture. Vrijednost $high(x)$ nam govori u koju $proto - vEB(\sqrt{u})$ strukturu idemo, a $low(x)$ koji element unutar te strukture provjeravamo.

**Propozicija 2.1.** *Vremenska složenost naredbe proto-vEB-MEMBER$(V, u)$ je* $O(\lg \lg u)$.

*Dokaz.* Za svaki poziv rekurzije potrebno je konstantno vrijeme, a naredba proto-vEB-MEMBER$(V, u)$ poziva rekurziju na $proto - vEB(\sqrt{u})$ strukturi, pa vremensku složenost ove naredbe možemo opisati jednadžbom $T(u) = T(\sqrt{u}) + O(1)$ iz Teorema 2.1 i znamo da je njeno rješenje $T(u) = O(\lg \lg u)$. □

**Primjer 2.1.** Pozivamo naredbu proto-vEB-MEMBER$(V, 5)$ gdje je V proto-vEB(16) struktura iz Slike 7. Kako je high(5) = 1 ulazimo u rekurziju nad proto-vEB(4) strukturom na koju pokazuje klaster 1 i zanima nas element low(x)=1, znači da je novi x=1. Kako je u=4 nastavljamo dalje u rekurziju nad proto-vEB(2) strukturom na koju pokazuje klaster high(1)=0 iz proto-vEB(4) i zanima nas element low(1)=1, kako je u=2 sada smo u početnom slučaju i A[1]=0 što znači da 5 nije član od V.

### 2.3.2 Minimum i maksimum

Kako bismo mogli saznati koji je najmanji element dinamičkog skupa $S$ spremljenog u $proto-vEB$ $V$ strukturi, definiramo naredbu proto-vEB-MINIMUM($V$) koja će nam vratiti najmanji element skupa $S$ ili NIL u slučaju kada je $S$ prazan skup.

**Algoritam 2** proto-vEB-MINIMUM($V$)

```
1  if V.u == 2 then
2      if V.A[0] == 1 then
3          return 0
4      else if V.A[1] == 1 then
5          return 1
6      else  return NIL
7  else minCluster = proto-vEB-MINIMUM(V.summary)
8      if minCluster == NIL then
9          return NIL
10     else offset = proto-vEB-MINIMUM(V.cluster[minCluster])
11         return index(minCluster, offset-)
```

Prvih 6 linija se odnosi na početni slučaj, kada je $V$ $proto-vEB(2)$ stuktura i u tom slučaju će nam linije 2-6 pronaći najmanji element ili vratit NIL ukoliko se radi o praznom skupu. Ako smo došli do linije 7, znamo da je veličina svemira od $V$ veća od 2. Linija 7 traži klaster u kojem se nalazi najmanji element i tu informaciju sprema u varijablu $minCluster$. Kada je $V$ prazno stablo, minimum ne postoji i tada će $minCluster$ biti jednak NIL i linija 9 će vratiti NIL. Kad smo došli u liniju 10, znamo da $V$ nije prazno stablo, pa linija 10 pronalazi na kojoj se poziciji unutar klaster $minCluster$ nalazi minimum i sprema tu informaciju u varijablu *offset*, a onda linija 11 računa i vraća minimum.

Analogno definiramo i naredbu proto-vEB-MAXIMUM($V$) koja nam vraća najveći element pohranjen u $proto-vEB$ $V$ strukturi ili NIL u slučaju kada je u $proto-vEB$ $V$ pohranjen prazan skup $S$.

**Algoritam 3** proto-vEB-MAXIMUM($V$)

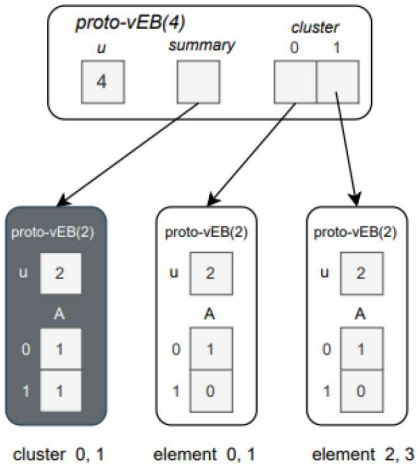
```
1  if V.u == 2 then
2      if V.A[1] == 1 then
3          return 1
4      else if V.A[0] == 1 then
5          return 0
6      else  return NIL
7  else maxCluster = proto-vEB-MAXIMUM(V.summary)
8      if maxCluster == NIL then
9          return NIL
10     else offset = proto-vEB-MAXIMUM(V.cluster[maxCluster])
11         return index(maxCluster, offset)
```

**Propozicija 2.2.** *Vremenska složenost naredbi proto-vEB-MINIMUM(V) i proto-vEB-MAXIMUM(V) je jednaka* $O(\lg u)$ *u najgorem slučaju.*

*Dokaz.* S obzirom na to da naredba proto-vEB-MINIMUM($V$) kao i proto-vEB-MAXIMUM($V$) ima dva rekurzivna poziva na $proto-vEB(\sqrt{u})$ strukturi, njihova vremenska složenost može se opisati sljedećom jednadžbom:

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

Kod ove jednadžbe također možemo koristiti metodu zamjene varijabli kao i kod Teorema 2.1 i tada dobijemo da je $T(u) = \Theta(\lg u)$ (za postupak vidi [1, str 542]). Odakle nam slijedi da je vremenska složenost naredbi proto-vEB-MINIMUM($V$) i proto-vEB-MAXIMUM($V$) $O(\lg u)$ u najgorem slučaju, umjesto očekivanih $O(\lg \lg u)$, upravo zato što navedene naredbe imaju po dva rekurzivna poziva. □




Slika 8: *proto-vEB(4) struktura koja prikazuje dinamički skup {0,2}.*

**Primjer 2.2. a)** Pokrećemo naredbu proto-vEB-MINIMUM($V$) nad proto-vEB(4) strukturom sa Slike 8. Kako je u = 4, pokrećemo rekurziju nad *V.summary* koji je oblika proto-vEB(2), pa ulazimo u bazni slučaj gdje je A[0] = 1, što znači da se najmanji element nalazi u klasteru 0. Sada kada znamo da se najmanji element nalazi u klasteru 0, preostaje nam pokrenuti rekurziju nad *V.cluster[0]* kako bismo saznali na kojoj se poziciji unutar tog klastera nalazi najmanji element. *V.cluster[0]* je oblika proto-vEB(2), što znači da smo u baznom slučaju i imamo da je A[0]=1, što znači da se najmanji element nalazi na poziciji nula unutar klastera na poziciji 0. Koristeći funkciju index(0,0) dobivamo da je najmanji element jednak 0.
**b)** Koristeći naredbu proto-vEB-MAXIMUM($V$) pronaći ćemo najveći element proto-vEB(4) stabla prikazanog na Slici 8. Kako se radi o proto-vEB(4) strukturi, prvo ćemo pokrenuti rekurziju nad *V.summary* kako bismo pronašli u kojem se klasteru nalazi najveći element od V. *V.Summary* je oblika proto-vEB(2), što znači da smo u baznom slučaju i imamo da je A[1] = 1 što nam govori da se najveći element nalazi u klasteru 1. Preostaje nam saznati na kojoj se poziciji unutar tog klastera nalazi najveći element pa pokrećemo rekurziju nad *V.cluster[1]*. *V.cluster[1]* ima veličinu svemira u jednaku 2, što znači da smo u baznom slučaju. A[1] = 0, ali je A[0] = 1, pa to znači da se najveći element nalazi na poziciji 0 u klasteru 1, a uz pomoć funkcije index(1,0) dobijemo da je najveći element 2.

### 2.3.3 Sljedbenik i prethodnik

Za broj $x$, takav da je $0 \leq x < V.u$, gdje je $V$ $proto - vEB$ struktura, definiramo naredbu proto-vEB-SUCCESSOR($V$, $x$), koja pronalazi sljedbenik broja $x$ u $proto - vEB$ $V$ strukturi, pri čemu $x$ ne mora biti element skupa $S$ koji je pohranjen u $V$. U slučaju kada u $V$ ne postoji element veći od $x$ naredba vraća NIL.

**Algoritam 4** proto-vEB-SUCCESSOR($V, x$)

```
1  if V.u == 2 then
2      if x == 0 and V.A[1] == 1 then
3          return 1
4      else return NIL
5  else offset = proto-vEB-SUCCESSOR(V.cluster[high(x)], low(x))
6      if offset ≠ NIL then
7          return index(high(x), offset)
8      else succCluster = proto-vEB-SUCCESSOR(V.summary, high(x))
9          if succCluster == NIL then
10             return NIL
11         else offset = proto-vEB-MINIMUM(V.cluster[succCluster])
12             return index(succCluster, offset)
```

Prve četiri linije linije naredbe odnose se na početni slučaj kada je $V$ $proto - vEB$ struktura čija je veličina svemira $u$ jednaka 2 i tada postoji samo jedan način da $x$ ima sljedbenika, a to je da je $x$ jednak 0, pa je onda njegov sljedbenik 1.
Linije 5-12 odnose se na slučaj kada je veličina svemira $u$ od $V$ veća od 2. Linija 5 traži sljedbenika od $x$ unutar klastera u kojem se nalazi $x$ i tu informaciju sprema u varijablu *offset*. Ako se sljedbenik od $x$ nalazi u istom klasteru kao i $x$ (linija 6), onda će linija 7 izračunati i vratiti taj sljedbenik.
Kada smo došli do linije 8 to znači da se sljedbenik od $x$ ne nalazi unutar istog klastera od $x$, pa linija 8 traži prvi idući neprazan klaster i informaciju o njemu sprema u varijablu $succCluster$. Ako su svi klasteri koji slijede nakon klastera od $x$ prazni, varijabla $succCluster$ će biti jednaka NIL (linija 9) i linija 10 će vratiti NIL, a to znači da sljedbenik od $x$ ne postoji unutar stabla $V$.
Ako je linija 8 pronašla neprazan klaster koji slijedi nakon klastera od x, onda je $succCluster$ različita od NIL, pa linija 11 pronalazi na kojoj se poziciji unutar klastera broj $succCluster$ nalazi sljedbenik od $x$, te linija 12 računa i vraća sljedbenik.

**Propozicija 2.3.** *Za izvršenje naredbe proto-vEB-SUCCESSOR(V,x) potrebno je* $O(\lg u \lg\lg u)$ *vremena u najgorem slučaju.*

*Dokaz.* Naredba proto-vEB-SUCCESSOR($V$, $x$) u najgorem slučaju ima dva rekurzivna poziva nad $proto - vEB(\sqrt{u})$ strukturi te jedan poziv na naredbu proto-vEB-MINIMUM($V$, $x$) također nad $proto - vEB(\sqrt{u})$ strukturom, pa se njena vremenska složenost može opisati sljedećom jednadžbom:

$$T(u) = 2T(\sqrt{u}) + \theta(lg\sqrt{u})$$

Ovu jednadžbu možemo srediti koristeći tehniku zamjene varijabli, a prema [1] nakon sređivanja dobijemo $T(u) = \theta(\lg u \lg\lg u)$. □

Analogno definiramo i naredbu proto-vEB-PREDECESSOR($V$, $x$) koja pronalazi prethodnik broja $x$, $0 \leq x < V.u$, unutar $proto - vEB\ V$ strukture.
Naredba proto-vEB-PREDECESSOR($V$, $x$) ima istu vremensku složenost kao i naredba proto-vEB-SUCCESSOR($V$,$x$).

**Algoritam 5** proto-vEB-PREDECESSOR($V, x$)

```
1  if V.u == 2 then
2      if x == 1 and V.A[0] == 1 then
3          return 0
4      else return NIL
5  else offset = proto-vEB-PREDECESSOR(V.cluster[high(x)], low(x))
6      if offset ≠ NIL then
7          return index(high(x), offset)
8      else predCluster = proto-vEB-PREDECESSOR(V.summary, high(x))
9          if predCluster == NIL then
10             return NIL
11         else offset = proto-vEB-MAXIMUM(V.cluster[predCluster])
12             return index(predCluster, offset)
```

**Primjer 2.3.** Tražimo sljedbenik broja 0 u proto-vEB(4) strukturi prikazanoj na Slici 8. Pokrećemo naredbu proto-vEB-SUCCESSOR($V$, 0), u=4 i ne nalazimo se u baznom slučaju, pa prvo provjeravamo je li sljedbenik od 0 u istom klasteru kao i 0 - nije. Kako bismo pronašli u kojem se klasteru nalazi sljedbenik, pokrećemo rekurziju nad $V.summary$ i tražimo sljedbenik elementa high(0) = 0. Kako je $V.summary$ oblika proto-vEB(2), ulazimo u početni slučaj i imamo da je x=0 i vrijedi da je A[1]=1, odakle znamo da se sljedbenik od 0 nalazi u klasteru 1. Preostaje nam pronaći na kojoj poziciji unutar klastera 1 se nalazi sljedbenik od 0, a to ćemo saznati tako da pronađemo minimum tog klastera - dobijemo da je minimum jednak 0. Dakle znamo da se sljedbenik od 0 nalazi u klasteru 1 na poziciji 0, pa uz pomoć funkcije index(1,0) dobijemo da je sljedbenik 2.

### 2.3.4 Spremanje elementa u proto van Emde Boasovo stablo

Definiramo naredbu proto-vEB-INSERT($V$, $x$) koja nam omogućuje da spremimo element $x$, $0 \leq x < V.u$, u $proto - vEB\ V$ strukturu čija je veličina svemira jednaka $u$. Da bismo spremili element $x$ u $proto - vEB\ V$ strukturu, moramo postaviti vrijednost odgovarajućeg bita u klasteru i $summary$ bita za taj klaster na 1.

**Algoritam 6** proto-vEB-INSERT($V, x$)

```
1  if V.u == 2 then
2      V.A[x]=1
3  else
4      proto-vEB-INSERT(V.cluster[high(x)], low(x))
5      proto-vEB-INSERT(V.summary, high(x))
```

Prva linija provjera radi li se o $proto - vEB(2)$ sturkturi, ako da onda u drugoj liniji bitu koji se nalazi na poziciji $x$ mijenjamo vrijednost u 1. Ako se ne radi o $proto - vEB(2)$ strukturi, onda u liniji 4 pozivamo rekurziju koja će postaviti element $x$ u odgovarajući klaster, te u

liniji 5 rekurziju koja će postaviti *summary* bit za taj klaster na 1.

**Propozicija 2.4.** *Vremenska složenost naredbe proto-vEB-INSERT(V, x) jednaka je* $O(\lg u)$ *u najgorem slučaju.*

*Dokaz.* Za izvršenje naredbe proto-vEB-INSERT($V, x$) u najgorem slučaju imamo dva rekurzivna poziva, pa njenu vremensku složenost možemo opisati jednadžbom (1) iz Propozicije 2.2 , odakle slijedi da je vremenska složenost ove operacije u najgorem slučaju $O(\lg u)$. □

**Primjer 2.4.** Želimo umetnuti element 3 u proto-vEB(4) stablo prikazano na Slici 8. Pokrećemo naredbu proto-vEB-INSERT($V, 3$). V ima veličinu svemira u jednaku 4, što znači da nismo u početnom slučaju. Prvo ćemo pokrenuti rekurziju nad proto-vEB(2) stablom koja će umetnuti element 3 u klaster high(3)=1 na poziciju low(3)=1. Kako smo pokrenuli rekurziju nad proto-vEB(2) stablom ušli smo u početni slučaj i potrebno je samo postaviti da vrijedi V.cluster[1].A[1]=1. Sada još trebamo ažurirati *summary*, pa pokrećemo rekurziju nad *V.summary* za element high(x)=1. *V.summary* je oblika proto-vEB(2), pa se nalazimo u početnom slučaju i trebamo samo postaviti da vrijedi V.summary.A[1]=1.

### 2.3.5 Brisanje elementa iz proto van Emde Boasovog stabla

Da bismo mogli obrisati neki element iz $proto - vEB$ strukture, prvo trebamo obrisati taj element iz klastera, a nakon toga provjeriti ima li u tom klasteru još neki element te ukoliko nema trebamo ažurirati *summary* bit tog klastera na 0. Za element $x$, $0 \leq x < V.u$, iz $proto - vEB$ $V$ strukture definiramo naredbu proto-vEB-DELETE($V$, $x$) na sljedeći način:

**Algoritam 7** proto-vEB-DELETE($V, x$)

```
1  if V.u == 2 then
2      V.A[x]=0
3  else
4      proto-vEB-DELETE(V.cluster[high(x)], low(x))
5      emptyCluster = True
6      for i = high(x)√u  to  (high(x)+1)√u-1  do
7          if proto-vEB-MEMBER(V.cluster[high(x)], i) == 1 then
8              emptyCluster = False
9              break
10     if emptyCluster == True then
11         proto-vEB-DELETE(V.summary, high(x))
```

Prve 3 linije odnose se na početni slučaj, kada je $V$ $proto - vEB(2)$ struktura i tada je potrebno samo promijeniti vrijednost bita na poziciji $x$ u 0.
Linije 3-11 odnose se na slučaj kad je veličina svemira $V$ veća od 2. U liniji 4 brišemo element $x$ iz pripadajućeg klaster. U liniji 5 definiramo varijablu *emptyCluster* u koju ćemo spremiti informaciju o tome je li klaster u kojem se nalazio element $x$ prazan, prilikom definiranja joj pridružujemo vrijednost True i pretpostavljamo da je klaster prazan. Uz pomoć petlje u linijama 6-9 provjeravamo je li klaster u kojem se nalazio $x$ prazan i tu inforomaciju spremamo u *emptyCluster*. Ako se ispostavi da je klaster u kojem se nalazio $x$ prazan, onda linija 11 uklanja taj klaster iz *summarya*.

**Propozicija 2.5.** *Vremenske složenost naredbe proto-vEB-DELETE(V, x) jednaka je* $O(\lg u \cdot \lg\lg u)$ *u najgorem slučaju.*

*Dokaz.* Kod naredbe proto-vEB-DELETE u najgorem slučaju imamo dva rekurzivna poziva nad $proto-vEB(\sqrt{u})$ strukturama te $\sqrt{u}$ poziva proto-vEB-MEMBER naredbe također nad $proto-vEB(\sqrt{u})$ strukturama, a za izvršenje proto-vEB-MEMBER naredbe nad $proto-vEB(u)$ strukturom potrebno je $O(\lg\lg u)$ vremena. Vremensku složenost naredbe proto-vEB-DELETE možemo opisati sljedećom jednadžbom:

$$\begin{aligned} T(u) &= 2T(\sqrt{u}) + O(\lg\lg\sqrt{u}) \\ &= 2T(\sqrt{u}) + O(\lg\lg u) \end{aligned} \tag{2}$$

Kako bismo riješili ovu jednadžbu koristit ćemo tehniku zamjene varijabli (pogledaj [1, str 86-87]). Neka $m = \lg u$, pa vrijedi da je $u = 2^m$, a kada to uvrstimo u (2) dobijemo

$$T(2^m) = 2T(2^{m/2}) + O(\lg m)$$

. Sada ćemo uvesti da je $S(m) = T(2^m)$, pa dobijemo jednadžbu

$$S(m) = 2S(\frac{m}{2}) + O(\lg m). \tag{3}$$

Ako primjenimo 2. slučaj master metode (pogledaj [1, str 93-97]), znamo da jednadžba (3) ima rješenje $S(m) = \Theta(m \lg m)$, iz čega onda slijedi da je $T(u) = T(2^m) = S(m) = \Theta(m \lg m) = \Theta(\lg u \cdot \lg\lg u)$, pa je za izvršenje naredbe proto-vEB-DELETE potrebno $O(\lg u \cdot \lg\lg u)$ vremena u najgorem slučaju. □

**Primjer 2.5.** U ovom primjeru želimo obrisati element 0 iz proto-vEB(4) strukture prikazane na Slici 8. Pokrećemo naredbu proto-vEB-DELETE$(V, 0)$, u=4 pa ne ulazimo u početni slučaj. Prvo pokrećemo rekurziju na proto-vEB(2) strukturi koja će obrisati element low(0)=0 u klasteru high(0)=0, a s obzirom na to da je sada u=2, ulazimo u bazni slučaj pa je potrebno samo postaviti da vrijedi V.cluster[0].A[0]=0. Sad kada smo obrisali element 0 iz klastera, trebamo provjeriti ima li u tom klasteru još elemenata - nema. Klaster je ostao prazan nakon brisanja elementa 0, pa moramo još pokrenuti rekurziju nad proto-vEB(2) stablom koja će obrisati element high(0)=0 iz *V.summary*, a kako smo sada u početnom slučaju dovoljno je postaviti da vrijedi *V.Summary.A*[0]=0.

# 3 Van Emde Boasovo stablo

U prethodnom poglavlju definirali smo $proto-vEB$ strukturu, ali za izvršenje svake od njenih naredbi potrebno je više od $O(\lg\lg u)$ vremena, uglavnom zbog toga što imaju previše rekurzivnih poziva, a jedino se naredba proto-vEB-MEMBER izvršava u $O(\lg\lg u)$ vremenu. U ovom poglavlju cilj nam je konstruirati sličnu strukturu, ali ćemo pokušati smanjiti broj rekurzija unutar naredbi, kako bi se svaka od naredbi izvršavala u $O(\lg\lg u)$ vremenu.

U prethodnom poglavlju smo radi jednostavnosti pretpostavili da je $u$ oblika $u = 2^{2^k}$, kako bi $u^{1/2}, u^{1/4}, u^{1/8}, ...$ bili cijeli brojevi, što je poprilično ograničilo veličinu svemira $u$. U ovom poglavlju ćemo relaksirati tu pretpostavku i dozvoliti da $u$ bude bilo koja potencija broja 2, odnosno sada pretpostavljamo da je $u$ oblika $u = 2^k$, gdje je $k$ cijeli broj. U slučaju kada je $u$ neparna potencija broja 2, $\sqrt{u}$ neće biti cijeli broj, pa ćemo tada $\lg u$ bitova podijeliti na $\lceil(\lg u)/2\rceil$ najznačajnijih bitova i $\lfloor(\lg u)/2\rfloor$ najmanje značajnih bitova.

**Definicija 3.1.** *Neka je $u$ veličina svemira.* ***Gornji korijen*** *definiran je s*

$$\sqrt[\uparrow]{u} = 2^{\lceil(\lg u)/2\rceil},$$

*a* ***donji korijen*** *je dan s*

$$\sqrt[\downarrow]{u} = 2^{\lfloor(\lg u)/2\rfloor}$$

*pri čemu vrijedi*

$$u = \sqrt[\uparrow]{u} \cdot \sqrt[\downarrow]{u}.$$

Nadalje, ponovno definiramo funkcije

**Definicija 3.2.** *Neka je $x$ element dinamičkog skupa $S$ i $u$ veličina svemira. Funkcija* ***high(x)*** *dana s*

$$high(x) = \lfloor x / \sqrt[\downarrow]{u} \rfloor$$

*govori nam u kojem se klasteru nalazi element $x$.*

**Definicija 3.3.** *Neka je $x$ element dinamičkog skupa $S$ i $u$ veličina svemira. Funkcija* ***low(x)*** *dana s*

$$low(x) = x \mod \sqrt[\downarrow]{u}$$

*govori nam na kojoj poziciji u klasteru se nalazi element $x$.*

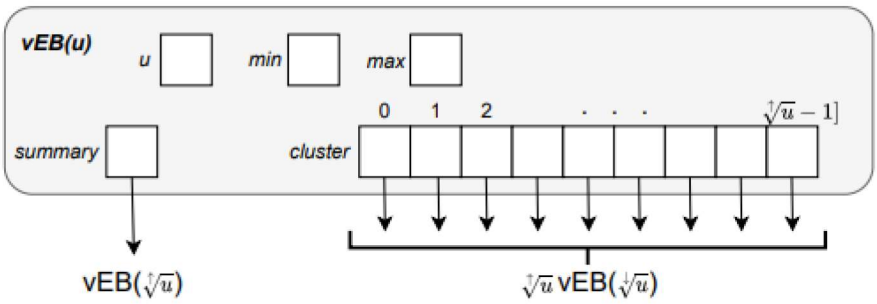
**Definicija 3.4.** *Neka $i$ predstavlja broj klastera, $j$ poziciju u tom klasteru i neka je $u$ veličina svemira. Funkcija* ***index(i,j)*** *dana s*

$$index(i, j) = i\sqrt[\downarrow]{u} + j$$

*govori nam koji se element nalazi u klasteru broj $i$ na poziciji $j$.*

## 3.1 Struktura van Emde Boasovog stabla

**Definicija 3.5.** *Van Emde Boasovo stablo je rekurzivno definirana struktura, označavamo ju s $vEB(u)$, gdje je $u$ veličina svemira.*
*a) Kada je $u = 2^k \neq 2$, $vEB(u)$ sadrži pokazivač* ***summary*** *na $vEB(\sqrt[\uparrow]{u})$, polje pokazivača* ***cluster***$[0\,..\,\sqrt[\uparrow]{u} - 1]$ *koje se sastoji od $\sqrt[\uparrow]{u}$ pokazivača na $vEB(\sqrt[\downarrow]{u})$ i atribute* ***min*** *i* ***max*** *koji pohranjuju najmanji i najveći element stabla.*
*b) Kada je $u = 2$, $vEB(u)$ sadrži atribut* ***min*** *koji pohranjuje najmanju vrijednost i* ***max*** *koji pohranjuje najveću vrijednost.*




Slika 9: *Struktura van Emde Boasovog stabla kada je veličina svemira $u > 2$. Struktura sadrži informaciju o veličini svemira u, najmanji(min) i najveći(max) element, pokazivač summary te polje pokazivača cluster*$[0\,..\,\sqrt[\uparrow]{u}-1]$.

Bitno je napomenuti da se element koji je spremljen u atributu *min* ne pojavljuje niti u jednom od $vEB(\sqrt[\downarrow]{u})$ stabala na koje pokazuju pokazivači iz polja *cluster*. To znači da se u stablu $vEB(u)$ $V$ nalaze elementi koji su spremljeni u $vEB(\sqrt[\downarrow]{u})$ stabla na koja pokazuje $V.cluster[0\,..\,\sqrt[\uparrow]{u}-1]$ i još element $V.min$. Za razliku od elementa koji je spremljen u atribut *min*, element koji je spremljen u atribut *max* će biti spremljen u jednom od klastera, osim u slučaju kada su *min* i *max* isti elementi. U slučaju kada imamo $vEB$ stablo s nula elementa, *min* i *max* će biti NIL.

Prednosti uvođenje atributa *min* i *max* su sljedeće:

- Naredbe MINIMUM i MAXIMUM nemaju rekurzivni poziv, već samo vraćaju vrijednosti atributa *min* i *max*

- Iz atributa *min* i *max* možemo saznati nalazi li se u $vEB$ stablu jedan element, dva i više elemenata ili je prazno, što nam pomaže kod naredbi INSERT i DELETE. Ako znamo da je $vEB$ stablo prazno, novi element možemo umetnuti tako da samo ažuriramo *min* i *max*, a ako znamo da je u $vEB$ stablu samo jedan element, taj element možemo obrisati također tako da samo ažuriramo atribute *min* i *max*.

- Kod naredbi SUCCESSOR i PREDECESSOR više nećemo trebati rekurzivni poziv da bismo saznali nalazi li se sljedbenik, odnosno prethodniku nekog elementa x u istom klasteru kao taj element. Ako je x manji od *max*, onda se njegov sljedbenik nalazi u istom klasteru kao i on, te ukoliko je x veći od *min* onda je njegov prethodnik u istom klastera kao i on.

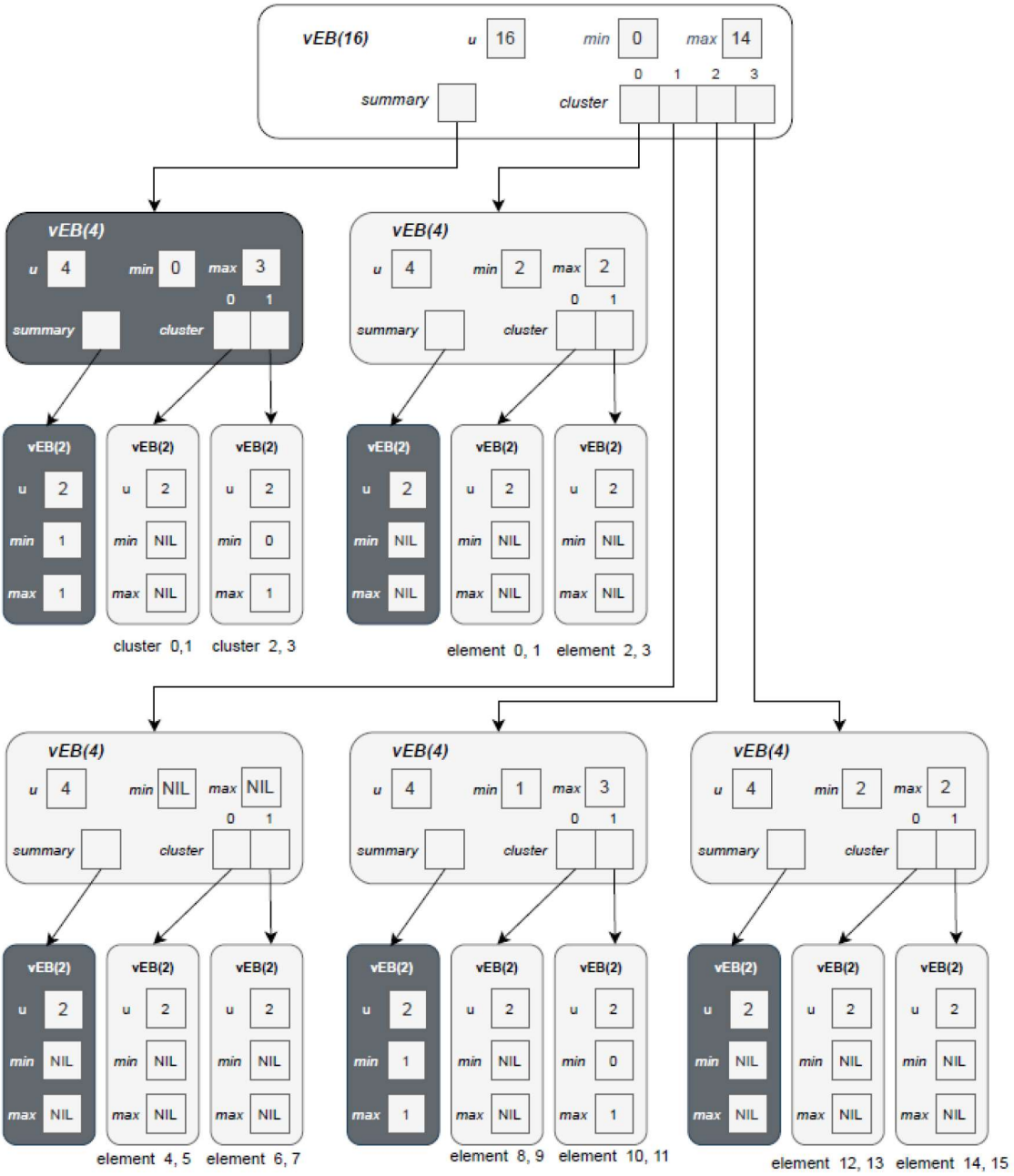
**Teorem 3.1.** *Vremenska složenost opisana nejednadžbom*

$$T(u) \leq T(\sqrt[\uparrow]{u}) + O(1)$$

*jednaka je* $O(\lg \lg u)$ *u najgorem slučaju.*

*Dokaz.* Vidi [1, str 547-549]. □

Gore navedene prednosti atributa *min* i *max* omogućile su da smanjimo broj rekurzija unutar naredbi, pa vremensku složenost svake od naredbi za van Emde Boasovo stablo možemo opisati nejednadžbom iz Teorema 3.1 .




Slika 10: *Potpuno raspisano* $vEB(16)$ *stablo koje prikazuje dinamički skup* $\{0, 2, 9, 10, 11, 14\}$. *Najmanji element u* $vEB(u)$ *stablu spremljen je u atribut min i ne pojavljuje se u klasterima.*

## 3.2 Naredbe za van Emde Boasovo stablo

Definirat ćemo naredbe za van Emde Boasovo stablo, koje primaju van Emde Boasovo stablo $V$ čija je dimenzija svemira jednaka $u$ ($vEB(u)$) i element $x$, takav da vrijedi $0 \leq x < V.u$.

### 3.2.1 Minimum i maksimum

**Propozicija 3.1.** *Vremenska složenosti naredbi vEB-MINIMUM(V) i vEB-MAXIMUM(V) jednaka je* $O(1)$.

Naredbe vEB-MINIMUM i vEB-MAXIMUM za $vEB$ stablo su u potpunosti pojednostavljene u odnosu na one koje smo definirali za $proto - vEB$ stablo, zato što ih jednostavno možemo dohvatiti jednom linijom koda koristeći atribute $min$ i $max$, pa je za njihovo izvršenje potrebno konstantno vrijeme $O(1)$.

**Algoritam 8** vEB-MINIMUM($V$)

```
return V.min
```

**Algoritam 9** vEB-MAXIMUM($V$)

```
return V.max
```

**Primjer 3.1.** U ovom primjeru želimo saznati minimum i maksimum vEB(16) stabla prikazanog na Slici 10. Kako bismo saznali minimum pokrećemo naredbu vEB-MINIMUM(V) nad tim stablom, a ona će nam vratiti vrijednost atributa $V.min$ koja je jednaka 0. Da saznamo maksimum pokrećemo naredbu vEB-MAXIMUM(V) koji nam vraća vrijednost atributa $V.max$=14 odakle saznajemo da je maksimum jednak 14.

### 3.2.2 Član van Emde Boasovog stabla

**Algoritam 10** vEB-MEMBER($V, x$)

```
1 if x == V.min or x == V.max then
2     return True
3 else if V.u == 2 then
4     return False
5 elsereturn vEB-MEMBER(V.cluster[high(x)], low(x))
```

Naredba vEB-MEMBER slična je kao kod $proto - vEB$ stabla, razlikuje se u tome što kod $vEB$ stabla prije svega provjerimo je li $x$ jednak minimalnom ili maksimalnom elementu, ukoliko je, naredba vraća True, a ako nije onda provjeravamo početni slučaj odnosno je li $V$ $vEB(2)$ stablo.
U slučaju da $V$ je $vEB(2)$ stablo, to znači da je $V$ prazno jer $x$ nije jednak ni minimumu ni maksimumu, a $vEB(2)$ stablo može imati samo dva elementa, minimum i maksimum, pa naredba vraća False.
Kada $x$ nije jednak ni minimumu ni maksimumu te $V$ nije $vEB(2)$ stablo onda pozivamo rekurziju nad $vEB(\sqrt[\downarrow]{u})$ stablom.

**Propozicija 3.2.** *Za izvršenje naredbe vEB-MEMBER(V) potrebno je* $O(\lg \lg u)$ *vremena u najgorem slučaju.*

*Dokaz.* Vremenska složenost ove naredbe opisana je nejednadžbom iz Teorema 3.1, pa je za njeno izvršenje potrebno $O(\lg \lg u)$ vremena u najgorem slučaju. □

**Primjer 3.2.** Zanima nas je li broj 5 element vEB(16) stabla prikazanog na Slici 10, pa ćemo pokrenuti naredbu vEB-MEMBER($V, 5$) nad tim stablom. Niti minimum, niti maksimum nisu jednaki 5, a u je jednak 16, pa pokrećemo rekurziju nad vEB(4) stablom i tražimo element low(5)=1 u klasteru high(5)=1. Sada provjeravamo je li minimum ili maksimum jednak low(5)=1 - nije, a u=4 pa nismo još uvijek u početnom slučaju, tako da pokrećemo rekurziju nad vEB(2) stablom i tražimo element low(1)=1 u klasteru high(1)=0. Provjeravamo je li low(1)=1 jednako minimumu ili maksimumu - nije, ali u=2, što znači da 5 nije element vEB(16) stabla prikazanog na Slici 10.

### 3.2.3 Sljedbenik i prethodnik

Kod naredbe proto-vEB-SUCCESSOR($V$, $x$) imamo dva rekurzivna poziva u najgorem slučaju, prvi rekurzivni poziv, provjerava nalazi li se sljedbenik od $x$ u istom klasteru kao $x$, a ukoliko se ne nalazi, onda drugi rekurzivni poziv pronalazi u kojem se klasteru nalazi sljedbenik. Kod *vEB* stabla imamo brzi pristup maksimumu, pa možemo izbjeći jedan rekurzivni poziv. Kod ove naredbe $x$ ne mora biti element od V, bitno je samo da vrijedi $0 \leq x < V.u$.

**Algoritam 11** vEB-SUCCESSOR($V, x$)

```
1  if V.u == 2 then
2      if x == 0 and V.max == 1 then
3          return 1
4      else return NIL
5  else if V.min ≠ NIL and x < V.min then
6      return V.min
7  else max-low = vEB-MAXIMUM(V.cluster[high(x)])
8      if max-low ≠ NIL and low(x) < max-low then
9          offset = vEB-SUCCESSOR(V.cluster[high(x)], low(x))
10         return index(high(x), offset)
11     else succCluster = vEB-SUCCESSOR(V.summary, high(x))
12         if succCluster == NIL then
13             return NIL
14         else offset = vEB-MINIMUM(V.cluster[succCluster])
15             return index(succCluster, offset)
```

Linije 1-4 odnose se na početni slučaj kada je veličina svemira od $V$ jednaka 2, linija 3 će vratit 1 ako tražimo sljedbenika od $x = 0$ i ako se 1 nalazi u $V$ tj. ako je maksimum jednak 1, u protivnom linija 4 vraća NIL.
Do linije 5 dolazimo ako nismo u početnom slučaju i ona provjerava je li $x$ manji od minumuma od $V$, ukoliko je, linija 6 vraća minimum i on je sljedbenik od $x$.
Ako nismo u početnom slučaju i $x$ nije manji od minimuma, onda znamo da je $x$ veći od minimuma. Linija 7 varijabli *max-low* pridružuje maksimalni element klastera u kojem se nalazi $x$. Linija 8 provjerava je li $x$ manji od *max-low*, ako je manji, to znači da se sljedbenik od $x$ nalazi u istom klasteru kao i $x$, linija 9 pronalazi na kojoj poziciji u tom klasteru se on

nalazi, a linija 10 uz pomoć funkcije index računa koji je to točno broj.
Ako se sljedbenik od $x$ ne nalazi u istom klasteru kao $x$, onda u liniji 11 tražimo klaster u kojem se nalazi sljedbenik i broj klastera spremamo u varijablu *offset*. Linija 12 provjerava je li varijabla *offset* jednaka NIL, ako je to znači da ne postoji sljedbenik od $x$ i linija 13 vraća NIL. Ako je varijabla *offset* različita od NIL, onda postoji sljebenik od $x$ u klasteru broj *offset* i tada linija 14 pronalazi na kojoj se točno poziciji unutar tog klastera nalazi sljedbenik, a linija 15 računa i vraća sljedbenik.

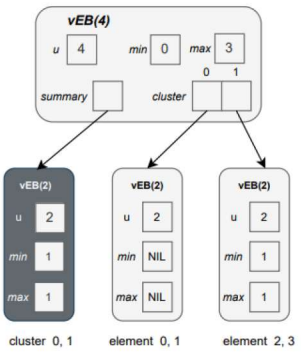
**Propozicija 3.3.** *Vremenska složenost naredbe vEB-SUCCESSOR($V, x$) jednaka je $O(\lg \lg u)$ u najgorem slučaju.*

*Dokaz.* Ako $V$ nije $vEB(2)$ stablo i ako je $x$ veći od minimuma od $V$, imat ćemo jedan rekurzivan poziv u naredbi ovisno o tome je li $x$ manji ili veći od maksimalnog elementa u svom klasteru. Kada je $x$ manji od maksimalnog elementa u svom klasteru, onda se izvršava rekurzivni poziv iz linije 9 nad $vEB(\sqrt[\downarrow]{u})$ stablom, a u suprotnom se izvršava rekurzivni poziv iz linije 11 nad $vEB(\sqrt[\uparrow]{u})$ stablom. Neovisno o kojem se slučaju radi rekurzivni poziv se izvršava nad $vEB$ stablom čija je veličina svemira najviše $\sqrt[\uparrow]{u}$), pa se vremenska složenost ove naredbe može opisati nejednadžbom iz Teorema 3.1 iz čega slijedi da je za izvršenje naredbe vEB-SUCCESSOR($V$,$x$) potrebno $O(\lg \lg u)$ vremena u najgorem slučaju. □

**Algoritam 12** vEB-PREDECESSOR($V, x$)

```
 1 if V.u == 2 then
 2     if x == 1 and V.min == 0 then
 3         return 0
 4     else return NIL
 5 else if V.max ≠ NIL and x > V.max then
 6     return V.max
 7 else min-low = vEB-MINIMUM(V.cluster[high(x)])
 8     if min-low ≠ NIL and low(x) > min-low then
 9         offset = vEB-PREDECESSOR(V.cluster[high(x)], low(x))
10         return index(high(x), offset)
11     else predCluster = vEB-PREDECESSOR(V.summary, high(x))
12         if predCluster == NIL then
13             if V.min ≠ NIL and x > V.min then
14                 return V.min
15             else return NIL
16         else offset = vEB-MAXIMUM(V.cluster[predCluster])
17             return index(predCluster, offset)
```

Naredba vEB-PREDECESSOR($V$, $x$) definira se analogno narebi vEB-SUCCESSOR($V$, $x$) uz dodatak linije 13 i 14 koje se odnose na slučaj kada se prethodnik od $x$ ne nalazi u istom klasteru kao i $x$. Tada, ako imamo slučaj kada su svi klasteri koji prethode klasteru od $x$ prazni, preostaje provjeriti je li minimum od $V$ prethodnik od $x$, jer se minimum od $V$ ne pojavljuje niti u jednom od klastera.
Linije 13 i 14 ne utječu na vremensku složenost naredbe, pa je za izvršenje naredbe vEB-PREDECESSOR($V$, $x$) potrebno $O(\lg \lg u)$ vremena u najgorem slučaju.




Slika 11: *vEB(4) struktura koja prikazuje dinamički skup {0,3}.*

**Primjer 3.3. a)** U ovom primjeru pokrećemo naredbu vEB-SUCCESSOR(V, 0) nad vEB(4) stablom koje je prikazano na Slici 11 i tražimo sljedbenik elementa 0. $V.u$=4 i $0 < V.min$, pa ne ulazimo u prva dva slučaja, nego tek u treći. Provjeravamo ima li klaster high(0)=0 u kojem se treba nalaziti 0 maksimum i je li low(0)=0 manji od tog maksimuma. S obzirom na to da je taj klaster prazan i njegov maksimum je jednak NIL, tražimo sljedeći klaster koji nije prazan tako što pokrećemo rekurziju nad $V.summary$ i unutar njega tražimo sljedbenik od high(0)=0. Sada ulazimo u početni slučaj jer je $V.summary$ oblika vEB(2) i imamo da je x=high(0)=0 i da je V.summary.max=1, pa znamo da se sljedbenik nalazi u $V.cluster$[1]. Preostaje nam još saznati na kojoj se poziciji unutar $V.cluster$[1] nalazi sljedbenik od 0, a to ćemo saznati tako da pronađemo minimum od $V.cluster$[1]. Minimum od V.cluster[1] je jednak 1, pa sad znamo da se sljedbenik od 0 nalazi u $V.cluster$[1] na poziciji 1 te uz pomoć funkcije index(1,1) dobijemo da je 3 sljedbenik od 0.
**b)** U ovom primjeru ćemo tražiti prethodnik broja 2 u vEB(4) stablu koje je prikazano na Slici 11, pa pokrećemo naredbu vEB-PREDECESSOR($V$, 2).
$V.u$=4, a $2 < V.max$=3, pa ne ulazimo u prva dva slučaja nego tek u treći. High(2)=1, pa prvo tražimo minimum od $V.cluster$[1], jer bi se u njemu trebao nalaziti broj 2. Minimum od $V.cluster$[1] je jednak 1, a to je veće od low(2)=0, što znači da u klasteru u kojem bi se trebao nalaziti broj 2, nema elementa koji je manji od 2. Sada moramo pronaći *predCluster* - prvi neprazan klaster koji prethodi klasteru $V.cluster$[1] i to ćemo napraviti tako da pokrenemo rekurziju nad $V.summary$ i tražimo prethodnik od high(2)=1. $V.summary$ je oblika vEB(2), pa ulazimo u prvi slučaj i dobivamo da je *predCluster*=NIL, uz to nam vrijedi da je $x=2 > V.min=0$, pa kao rezultat dobivamo da je prethodnik od 2 jednak $V.min$=0.

### 3.2.4 Spremanje elementa u van Emde Boasovo stablo

Kod $proto - vEB$ stabla naredba proto-vEB-INSERT($V, x$) ima dva rekurzivna poziva, prvi sprema element u klaster, a drugi ažurira *summary* bit za taj klaster. Kod $vEB$ stabla su atributi *min* i *max* omogućili da naredba vEB-INSERT($V, x$) ima samo jedan rekurzivni poziv. Ako se u klasteru već nalaze drugi elementi onda je broj tog klastera već spremljen u *summary* pa nema potrebe za rekurzivni poziv koji ažurira *summary*, a ako je klaster prazan onda element $x$ postaje jedini element tog klastera, pa nema rekurzivnog poziva za umetanje elementa $x$ u prazno $vEB$ stablo.
Prvo definiramo naredbu vEB-EMPTY-INSERT($V, x$) koja sprema element $x$ u $vEB$ stablo $V$ koje je prazno.

**Algoritam 13** vEB-EMPTY-INSERT($V, x$)

```
1 V.min = x
2 V.max = x
```

Naredba vEB-INSERT($V, x$) sprema element $x$ u $vEB$ stablo $V$, pri čemu vrijedi pretpostavka da se element $x$ ne nalazi u $V$.

**Algoritam 14** vEB-INSERT($V, x$)

```
1  if V.min == NIL then
2      vEB-EMPTY-INSERT(V, x)
3  else
4      if x < V.min then
5          exchange x with V.min
6      if V.u > 2 then
7          if vEB-MINIMUM(V.cluster[high(x)]) == NIL then
8              vEB-INSERT(V.summary, high(x))
9              vEB-EMPTY-INSERT(V.cluster[high(x)], low(x)
10         else vEB-INSERT(V.cluster[high(x)], low(x)
11     if x > V.max then
12         V.max = x
```

Linija 1 provjerava je li $V$ prazno $vEB$ stablo i ako je linija 2 sprema $x$ u prazno $vEB$ $V$ stablo.
Za slučaj kada $V$ nije prazno $vEB$ stablo odgovorne su linije 3-12. Linije 4 i 5 provjeravaju je li $x$ manji od minimuma od $V$ i ako je, $x$ postaje minimum, a minimum postaje element $x$ koji je potrebno spremiti u neki od klastera od $V$.
Linije 6-10 pokrivaju slučaj kada je $V$ $vEB$ stablo čija je veličina svemira $u$ veća od 2, tada imamo dva slučaja. Prvi slučaj je kada je klaster u koji trebamo umetnuti $x$ prazan, onda imamo jedan rekurzivni poziv naredbe vEB-INSERT nad $vEB(\sqrt[\uparrow]{u})$ stablom koja sprema broj klastera od $x$ u *summary* (linija 8) i poziv naredbe vEB-EMPTY-INSERT koja će umetnuti $x$ u prazno stablo (linija 9). Drugi slučaj je kada klaster u koji treba spremiti $x$ nije prazan, tada ne treba spremiti broj klastera u *summary*, jer je on već spremljen, nego samo treba spremiti $x$ u klaster, a za to je odgovorna linija 10 koja rekurzivno poziva naredbu vEB-INSERT nad $vEB(\sqrt[\downarrow]{u})$ stablom.
Linije 11 i 12 pokrivaju slučaj kada je $x$ veći od maksimuma od $V$, pa je potrebno ažurirati atribut $max$.

**Propozicija 3.4.** *Vremenska složenost naredbe vEB-INSERT($V, x$) jednaka je $O(\lg \lg u)$ u najgorem slučaju.*

*Dokaz.* Kod ove naredbe imamo jedan rekurzivni poziv nad $vEB$ stablom čija je veličina svemira u najgorem slučaju jednaka $\sqrt[\uparrow]{u}$, pa se vremenska složenost može opisati nejednadžbom iz Teorema 3.1, odakle nam slijedi da je vrijeme potrebno za izvršenje naredbe vEB-INSERT($V, x$) $O(\lg \lg u)$ u najgorem slučaju. □

**Primjer 3.4.** U ovom primjeru spremit ćemo broj 1 u vEB(4) stablo prikazano na Slici 11 koristeći naredbu vEB-INSERT($V, 1$). $V$ nije prazno stablo, pa ulazimo u drugi slučaj.

$V.u$=4, high(1)=0, pa tražimo minimum od $V.cluster[0]$ i dobijemo da je on jednak NIL, što znači da je $V.cluster[0]$ prazan klaster i zbog toga trebamo ažurirati $V.summary$ i nakon toga umetnuti 0 u prazan klaster $V.cluster[0]$.
Prvo ćemo ažurirati *summary* tako što ćemo pokrenuti rekurziju nad $V.summary$ i u njega umetnuti high(1)=0. Sada imamo slučaj da je $x$=high(1)=0 < $V.summary.min$ = 1, pa ćemo zamijeniti te dvije varijable tako da vrijedi $V.min$ = 0, a x=1. Kako je $V.summary$ oblika vEB(2), preostaje nam još samo provjeriti je li x=1 veće od $V.summary.max$=1 - nije, pa smo sada gotovi s ažuriranjem *summarya*.
Sada nam još samo preostaje da umetnemo broj 0 u prazan klaster $V.cluster[0]$, a to ćemo napraviti tako što pozovemo naredbu vEB-EMPTY-INSERT($V.cluster[0]$, 0) koja u prazno vEB(2) stablo $V.cluster[0]$ sprema element low(0)=0, tako što ažurira $V.cluster[0].min$ i $V.cluster[0].max$ da oba budu jednaki 0.

### 3.2.5 Brisanje elementa iz van Emde Boasovog stabla

Posljednja naredba koju definiramo za van Emde Boasovo stablo je naredba vEB-DELETE($V, x$) koja briše element $x$ iz $vEB$ stabla $V$, pri čemu vrijedi pretpostavka da se $x$ nalazi u $V$.

**Algoritam 15** vEB-DELETE($V, x$)

```
1  if V.min == V.max then
2      V.min = NIL
3      V.max = NIL
4  else if V.u == 2 then
5      if x == 0 then
6          V.min = 1
7      else V.min = 0
8      V.max = V.min
9  else
10     if x == V.min then
11         firstCluster = vEB-MINIMUM(V.summary)
12         x = index(firstCluster, vEB-MINIMUM(V.cluster[firstCluster]))
13         V.min = x
14     vEB-DELETE(V.cluster[high(x)], low(x))
15     if vEB-MINIMUM(V.cluster[high(x)]) == NIL then
16         vEB-DELETE(V.summary, high(x))
17         if x == V.max then
18             summaryMax = vEB-MAXIMUM(V.summary)
19             if summaryMax == NIL then
20                 V.max = V.min
21             else
22                 V.max = index(summaryMax, vEB-MAXIMUM(V.cluster[summaryMax]))
23     else if x == V.max then
24         V.max = index(high(x), vEB-MAXIMUM(V.cluster[high(x)]))
```

Linije 1-3 pokrivaju slučaj kada je u $vEB$ stablu $V$ samo jedan element i to je upravo $x$, pa je potrebno samo postaviti atribute $min$ i $max$ na NIL.
Ako smo došli do linije 4, to znači da $V$ ima barem 2 elementa. Linije 4-8 odnose se na slučaj kada je $V$ $vEB(2)$ stablo, što znači da $V$ ima samo minimum i maksimum, pa je potrebno

odraditi kojem od ta dva elementa je jednak $x$ te prema tome ažurirati atribute *min* i *max*, a upravo to rade linije 5-8.
Kada smo došli do linije 9, znamo da je $V$ stablo s 2 ili više elemenata, te da je veličina svemira $u \geq 4$ i linije 9-24 će se pobrinuti za taj slučaj. Linije 10-13 provjeravaju je li $x$ jednak minimumu od $V$, ako je onda pronalaze klaster u kojem se nalazi idući najmanji element iz $V$ i poziciju na kojoj se on nalazi, te taj element postaje novi minimum i novi $x$. Linija 14 briše $x$ iz klastera, neovisno o tome je li $x$ jednak onom $x$ koji je proslijeđen naredbi ili je $x$ element koji je postao novi minimum.
Linija 15-22 pokrivaju slučaj kada je klaster u kojem se nalazio $x$ ostao prazan nakon brisanja elementa $x$. Linija 16 briše taj klaster iz *summarya*. Linija 17 provjerava je li $x$ bio maksimum od $V$, ako je linija 18 pronalazi klaster u kojem se nalazi novi maksimum. U slučaju da ne postoji novi maksimum, onda će maksimum biti jednak minimumu i za to su odgovorne linije 19 i 20, a ako je pronađen klaster u kojem se nalazi novi maksimum, linija 22 računa taj maksimum i ažurira atribut *max*.
Ako klaster u kojem se nalazio $x$ nije prazan nakon brisanja, onda će linija 23 provjeriti je li $x$ bio maksimum od $V$ i ako je, linija 24 će pronaći novi maksimum i ažurirati atribut *max*.

**Propozicija 3.5.** *Vremenska složenost naredbe vEB-DELETE($V, x$) jednaka je $O(\lg \lg u)$ u najgorem slučaju.*

*Dokaz.* U naredbi vEB-DELETE($V, x$) moguće je da se izvrše dva rekurzivna poziva, jedan u liniji 14, a drugi u liniji 16. Nakon što se izvršio rekurzivni poziv iz linije 14, rekurzivni poziv iz linije 16 izvršit će se samo ako je linija 15 pokazala da je klaster u kojem se nalazio $x$ ostao prazan, što znači da je u tom klasteru bio samo jedan element. A s obzirom na to da je u klasteru bio samo jedan element, to znači da su se prilikom rekurzivnog poziva iz linije 14 izvršile samo linije 1-3, koje su odgovorne za slučaj kada se u $vEB$ stablu nalazi samo jedan element i za njihovo izvršenje potrebno je $O(1)$ vremena.
Iz prethodnog imamo dva slučaja. Prvi slučaj je da se izvršio samo rekurzivni poziv iz linije 14. Drugi slučaj je da su se izvršila oba rekurzivna poziva, ali za izvršenje onog iz linije 14 je trebalo konstantno vrijeme. U oba slučaja vremensku složenost naredbe vEB-DELETE($V, x$) možemo opisati nejednadžbom iz Teorema 3.1, što znači da je za izvršenje naredbe potrebno $O(\lg \lg u)$ vremena u najgorem slučaju. □

**Primjer 3.5.** U ovom primjeru obrisat ćemo element 3 iz vEB(4) stabla koje je prikazano na Slici 11 koristeći naredbu vEB-DELETE($V, 3$). $V.min$ je različit od $V.max$ i $u$ je različit od 2, pa ne ulazimo u prva dva slučaja, nego tek u treći. High(3)=1, pa pokrećemo rekurziju nad vEB(2) stablom $V.cluster[1]$ i iz njega brišemo element low(3)=1 - $V.cluster[1]$.min=$V.cluster[1]$.max, što znači da ulazimo u prvi slučaj i trebamo postaviti vrijednosti od $V.cluster[1]$.min i $V.cluster[1]$.max na NIL da bismo obrisali element 1 iz $V.cluster[1]$. Nakon brisanja $V.cluster[1]$ je ostao prazan i minimum od $V.cluster[1]$ je jednak NIL, pa moramo ažurirati i *summary*. Pokrećemo rekurziju nad vEB(2) stablom $V.summary$ i brišemo element high(3)=1 iz njega - ulazimo u početni slučaj jer je $V.summary.min = V.summary.max$, pa oba atributa postavljamo na NIL. Sada je maksimum od $V.summary$ jednak NIL, pa je novi $V.max$ jednak $V.min$.

# Literatura

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms, The MIT Press, Massachusetts London, England, 2009.

[2] P. van Emde Boas, Preserving order in a forest in less than logarithmic time, 16th Annual Symposium on Foundations of Computer Science (sfcs 1975), 1975, 75-84, doi: 10.1109/SFCS.1975.26.

[3] Lecture 12: van Emde Boas Trees, Lunds Universitet. Dostupno na: `https://fileadmin.cs.lth.se/cs/Personal/Rolf_Karlsson/lect12.pdf` (16.5.2022.)

[4] proto van Emde Boas Trees, GeeksforGeeks. Dostupno na: `https://www.geeksforgeeks.org/proto-van-emde-boas-trees-set-1-background-introduction/` (16.5.2022.)

[5] Van Emde Boas Tree, GeekforGeeks. Dostupno na: `https://www.geeksforgeeks.org/van-emde-boas-tree-set-1-basics-and-construction/?ref=lbp` (16.5.2022.)