Programiranje 1

Poglavje 9: Vmesniki

Luka Fürst

Vmesnik

- abstrakten razred z dodatnimi omejitvami
- lahko vsebuje zgolj
  - abstraktne metode
  - statične metode
  - privzete metode (`default`)
  - statične konstante (`static final`)
- vsi elementi vmesnika so javno dostopni
- relevantna dostopna določila se samodejno dodajo
  - namesto

    ```
    public abstract int pretvori(String niz);
    ```

    lahko pišemo kar

    ```
    int pretvori(String niz);
    ```

Implementacija vmesnika

- razred implementira vmesnik, če so v njem definirane vse abstraktne metode vmesnika
- razred je lahko podrazred največ enega razreda, lahko pa (poleg tega) implementira poljubno mnogo vmesnikov

Implementacija vmesnika

```
public interface Narisljiv {
    public abstract void narisi(Platno platno);
}

public interface Izracunljiv {
    public abstract int ploscina();
    public abstract int obseg();
}

public class Pravokotnik implements Narisljiv, Izracunljiv {
    ...
    @Override
    public void narisi(Platno platno) { ... }

    @Override
    public int ploscina() { ... }

    @Override
    public int obseg() { ... }
}
```

Nekaj javinih vmesnikov

- `Comparable`
- `Comparator`
- `Iterable`
- `Iterator`

Vmesnik Comparable

```
public interface Comparable<T> {
    public abstract int compareTo(T drugi)
}
```

- če v razredu *R* implementiramo vmesnik `Comparable<R>`, določimo, da je njegove objekte mogoče med sabo primerjati
- z implementacijo vmesnika `Comparable<R>` določimo naravno urejenost objektov razreda *R*
  - objekti tipa `Integer` so urejeni glede na relacijo $<$
  - objekti tipa `String` so urejeni leksikografsko
  - objekti tipa `Cas` so urejeni kronološko
  - objekti tipa `Oseba` so urejeni primarno po priimkih, sekundarno pa po imenih

Vmesnik `Comparable`

- metodo

  ```
  public abstract int compareTo(T drugi)
  ```

  implementiramo tako, da vrne
  - negativno število, če objekt `this` po naravni urejenosti sodi pred objekt `drugi`
  - 0, če se objekta `this` in `drugi` po naravni urejenosti ne razlikujeta
  - pozitivno število, če objekt `this` po naravni urejenosti sodi za objekt `drugi`

Vmesnik Comparable in razred Cas

```
public class Cas implements Comparable<Cas> {
    private int ura;
    private int minuta;
    ...
    @Override
    public int compareTo(Cas drugi) {
        return 60 * (this.ura - drugi.ura) +
                    (this.minuta - drugi.minuta);
    }
    ...
}
```

Vmesnik Comparable in razred Oseba

```
public class Oseba implements Comparable<Oseba> {
    private String ime;
    private String priimek;
    private char spol;
    private int letoRojstva;
    ...
    @Override
    public int compareTo(Oseba druga) {
        if (this.priimek.equals(druga.priimek)) {
            return this.ime.compareTo(druga.ime);
        }
        return this.priimek.compareTo(druga.priimek);
    }
    ...
}
```

Urejanje vektorja po naravni urejenosti elementov

- urejamo objekt tipa `Vektor<T>`
- elementi (objekti tipa `T`) morajo biti med sabo primerljivi (`T extends Comparable<T>`)
- urejamo z navadnim vstavljanjem, ki ga že poznamo

```
public static <T extends Comparable<T>> void uredi(Vektor<T> vektor) {
    int stElementov = vektor.steviloElementov();
    for (int i = 1; i < stElementov; i++) {
        T element = vektor.vrni(i);
        int j = i - 1;
        while (j >= 0 && vektor.vrni(j).compareTo(element) > 0) {
            vektor.nastavi(j + 1, vektor.vrni(j));
            j--;
        }
        vektor.nastavi(j + 1, element);
    }
}
```

Vmesnika Comparable in Comparator

```
public interface Comparable<T> {
    p a int compareTo(T drugi);
}
```

- implementiramo z razredom, čigar objekte želimo primerjati
- definiramo (eno in edino) naravno urejenost
- `compareTo` mora vrniti < 0 / 0 / > 0, če `this` sodi pred / je enakovreden / sodi za `drugi`

```
public interface Comparator<T> {
    p a int compare(T prvi, T drugi);
}
```

- implementiramo z ločenim razredom
- definiramo eno od več možnih alternativnih urejenosti
- `compare` mora vrniti < 0 / 0 / > 0, če `prvi` sodi pred / je enakovreden / sodi za `drugi`

Vmesnik Comparator in razred Oseba

- alternativna urejenost št. 1: samo po priimku

```
public class Oseba {
    ...
    public static Comparator<Oseba> primerjalnikPoPriimku() {
        return new PrimerjalnikPoPriimku();
    }

    private static class PrimerjalnikPoPriimku
                    implements Comparator<Oseba> {
          @Override
          public int compare(Oseba prva, Oseba druga) {
              return prva.priimek.compareTo(druga.priimek);
          }
      }
}
```

Vmesnik Comparator in razred Oseba

- alternativna urejenost št. 2: primarno po spolu (najprej Z, potem M), sekundarno po padajoči starosti

```
public class Oseba {
    ...
    public static Comparator<Oseba> primerjalnikPoSpoluInStarosti() {
        return new PrimerjalnikPoSpoluInStarosti();
    }

    private static class PrimerjalnikPoSpoluInStarosti
                    implements Comparator<Oseba> {
        @Override
        public int compare(Oseba prva, Oseba druga) {
            if (prva.spol != druga.spol) {
                return druga.spol - prva.spol;
            }
            return prva.letoRojstva - druga.letoRojstva;
        }
    }
}
```

Urejanje vektorja glede na podani primerjalnik

```
public static <T> void uredi(Vektor<T> vektor,
                             Comparator<T> primerjalnik) {
    int stElementov = vektor.steviloElementov();
    for (int i = 1;  i < stElementov;  i++) {
        T element = vektor.vrni(i);
        int j = i - 1;
        while (j >= 0 && primerjalnik.compare(
                             vektor.vrni(j), element) > 0) {
            vektor.nastavi(j + 1, vektor.vrni(j));
            j--;
        }
        vektor.nastavi(j + 1, element);
    }
}
```

Vmesnik `Iterator`

- iterator je objekt, ki omogoča zaporedni sprehod po elementih vsebovalnika (npr. tabele, vektorja ali slovarja)
- vmesnik `Iterator`

```
public interface Iterator<T> {
    public abstract boolean hasNext();
    public abstract T next();
}
```

- metoda `hasNext` vrne `true` natanko tedaj, ko se v vsebovalniku nahaja še kak element
- metoda `next` vrne naslednji element vsebovalnika
  - če takega elementa ni, sproži izjemo tipa `NoSuchElementException`

Vmesnik `Iterator`

- razred, ki implementira vmesnik `Iterator`, vzdržuje položaj trenutnega elementa v vsebovalniku
- tipična uporaba

```
Iterator<Tip> iterator = ...
while (iterator.hasNext()) {
    Tip element = iterator.next();
    // obdelaj element
}
```

Vmesnik Iterable

```
public interface Iterable<T> {
    public abstract Iterator<T> iterator();
}
```

- vrne objekt tipa `Iterator`, ki omogoča zaporedni sprehod po vsebovalniku `this`
- če razred za definicijo vsebovalnika implementira vmesnik `Iterable`, potem se lahko po elementih vsebovalnika sprehodimo z zanko *for-each*

```
for (T element: vsebovalnik) {
    // obdelaj element
}
```

Vmesnik Iterable v razredu Vektor

```
public class Vektor<T> implements Iterable<T> {
    ...
    @Override
    public Iterator<T> iterator() {
        return new IteratorCezVektor<T>(this);
    }
}
```

Razred `IteratorCezVektor`

- objekt razreda `IteratorCezVektor` potrebuje dostop do samega vektorja, da se bo lahko sprehodil po njegovih elementih
- zato mu ob izdelavi posredujemo kazalec `this`
- razred `IteratorCezVektor` definiramo kot notranji razred v razredu `Vektor`
- zakaj notranji razred?
  - skrivanje implementacijskih podrobnosti (zunanji svet potrebuje samo objekt tipa `Iterator`, ne pa tudi konkretni implementacijski razred)
  - znotraj njegovih metod lahko uporabljamo privatne komponente objektov tipa `Vektor` (ker smo sintaktično še vedno znotraj razreda `Vektor`)

Razred `IteratorCezVektor`

- atribut `indeks` hrani indeks trenutnega elementa vektorja
- v konstruktorju ga nastavimo na 0
- metoda `hasNext` vrne `true` natanko tedaj, ko je indeks manjši od števila elementov vektorja
- metoda `next` vrne trenutni element in poveča indeks

Razred IteratorCezVektor

```
public class Vektor<T> implements Iterable<T> {
    ...
    private static class IteratorCezVektor<T> implements Iterator<T> {
        private Vektor<T> vektor;
        private int indeks;

        public IteratorCezVektor(Vektor<T> vektor) {
            this.vektor = vektor;
            this.indeks = 0;
        }

        @Override
        public boolean hasNext() {
            return (this.indeks < this.vektor.steviloElementov());
        }
        ...
    }
}
```

Razred IteratorCezVektor

```
public class Vektor<T> implements Iterable<T> {
    ...
    private static class IteratorCezVektor<T> implements Iterator<T> {
        ...
        @Override
        public T next() {
            if (this.indeks >= this.vektor.steviloElementov()) {
                throw new NoSuchElementException();
            }
            return this.vektor.vrni(this.indeks++);
        }
    }
}
```

Uporaba vmesnika Iterable

- primer

```
Vektor<Integer> vektor = new Vektor<>();
...
for (Integer element: vektor) {
    System.out.println(element);
}
```

- gornja koda se dejansko izvede takole:

```
Vektor<Integer> vektor = new Vektor<>();
...
Iterator<Integer> iterator = vektor.iterator();
while (iterator.hasNext()) {
    Integer element = iterator.next();
    System.out.println(element);
}
```

Iterator po slovarju

- implementacija vmesnika `Iterable<K>`

```
public class Slovar<K, V> implements Iterable<K> {
    ...
    @Override
    public Iterator<K> iterator() {
        return new IteratorPoKljucih<K, V>(this);
    }

    private static class IteratorPoKljucih<K, V>
                implements Iterator<K> {
        ...
    }
}
```

Iterator po slovarju

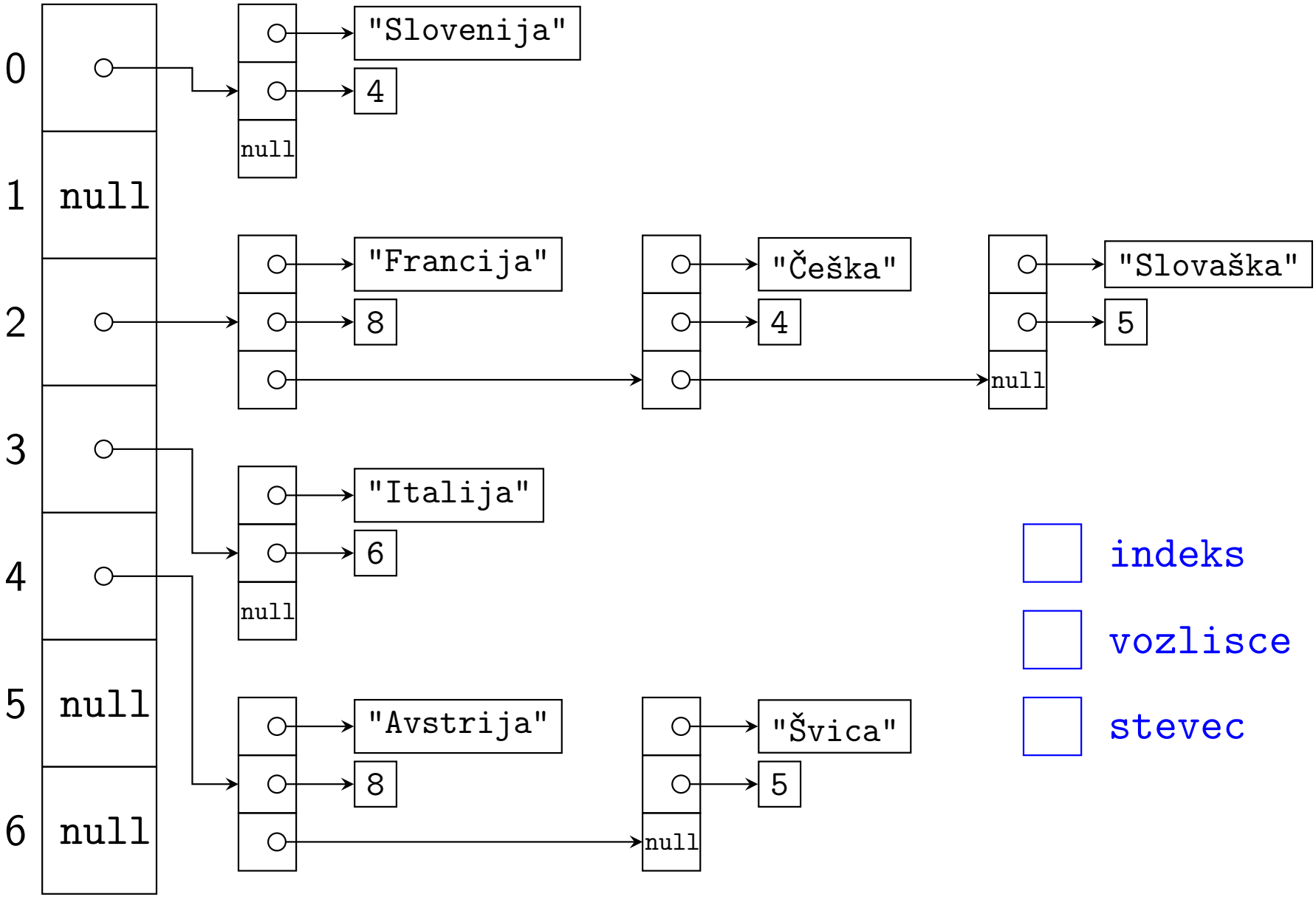
- iterator (objekt razreda `IteratorPoKljucih`) se sprehodi po ključih v slovarju
- uporaba

```
for (K kljuc: slovar) {
    V vrednost = slovar.vrni(kljuc);
    // obdelaj par kljuc-vrednost
}
```

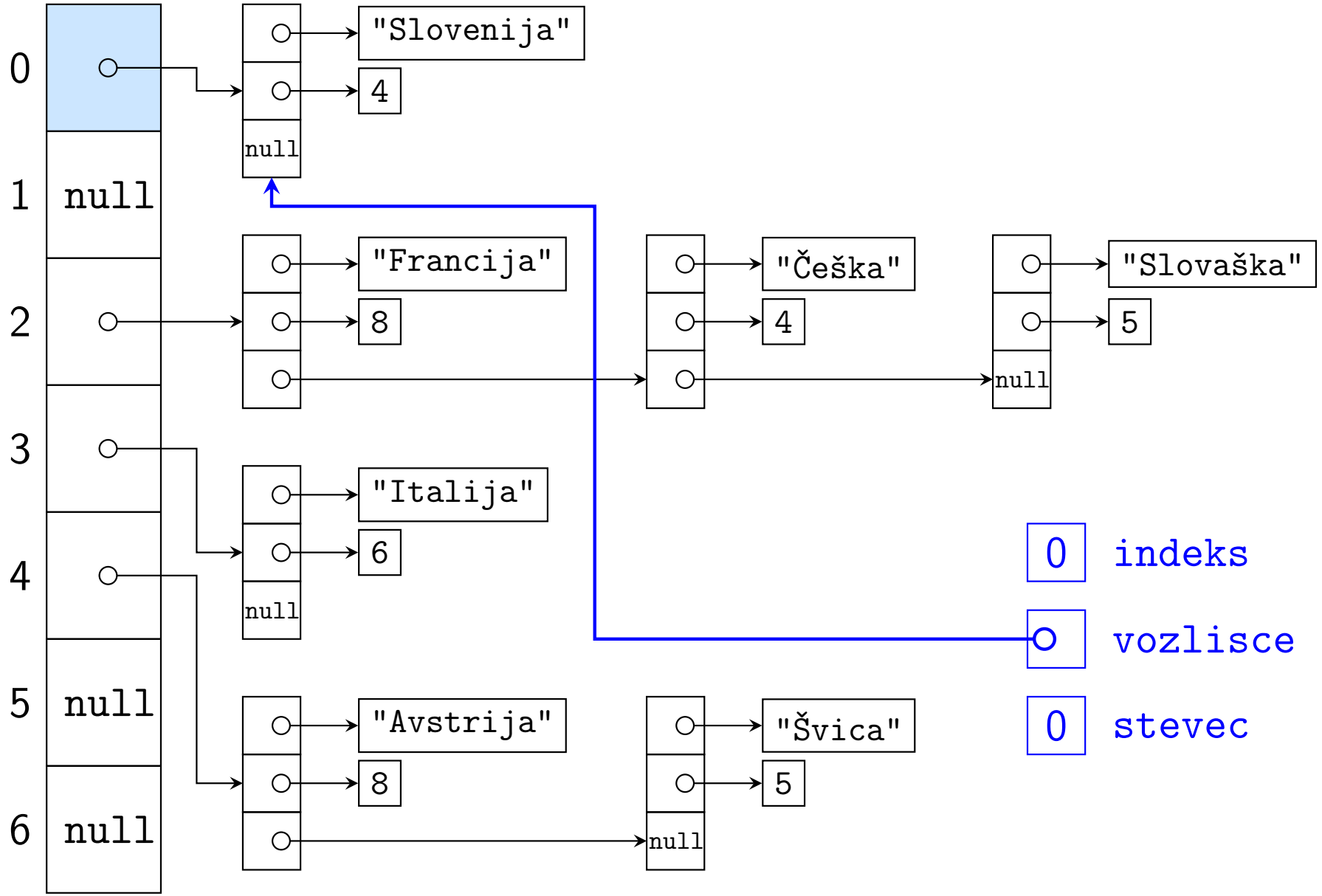
Sprehod po slovarju

- v razredu `IteratorPoKljucih` vzdržujemo sledeče atribute:
  - `slovar`: slovar, po katerem se sprehajamo
  - `indeks`: indeks trenutnega elementa tabele kazalcev
  - `vozlisce`: kazalec na trenutno vozlišče v povezanem seznamu
  - `stevec`: globalni indeks trenutnega vozlišča

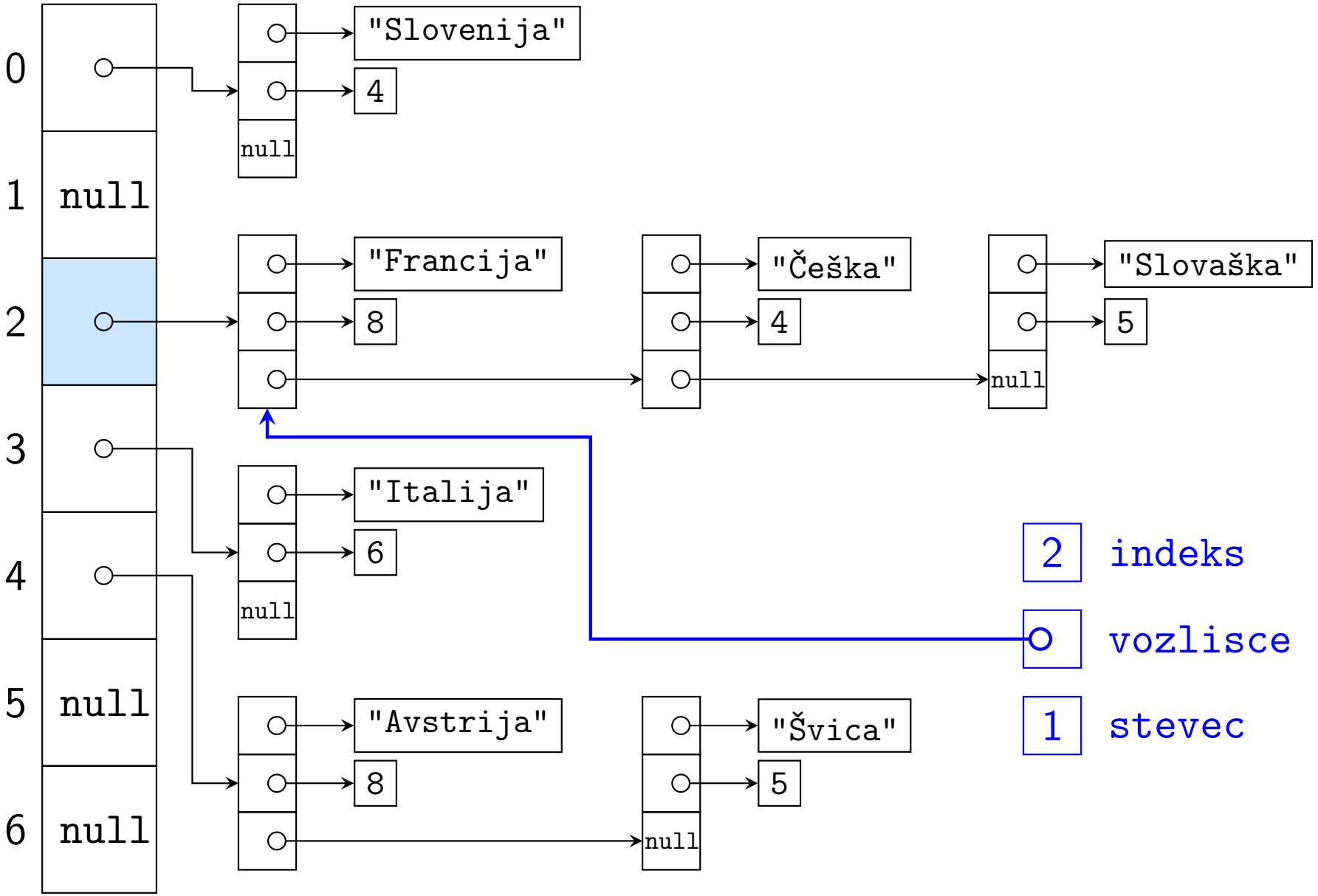
Sprehod po slovarju



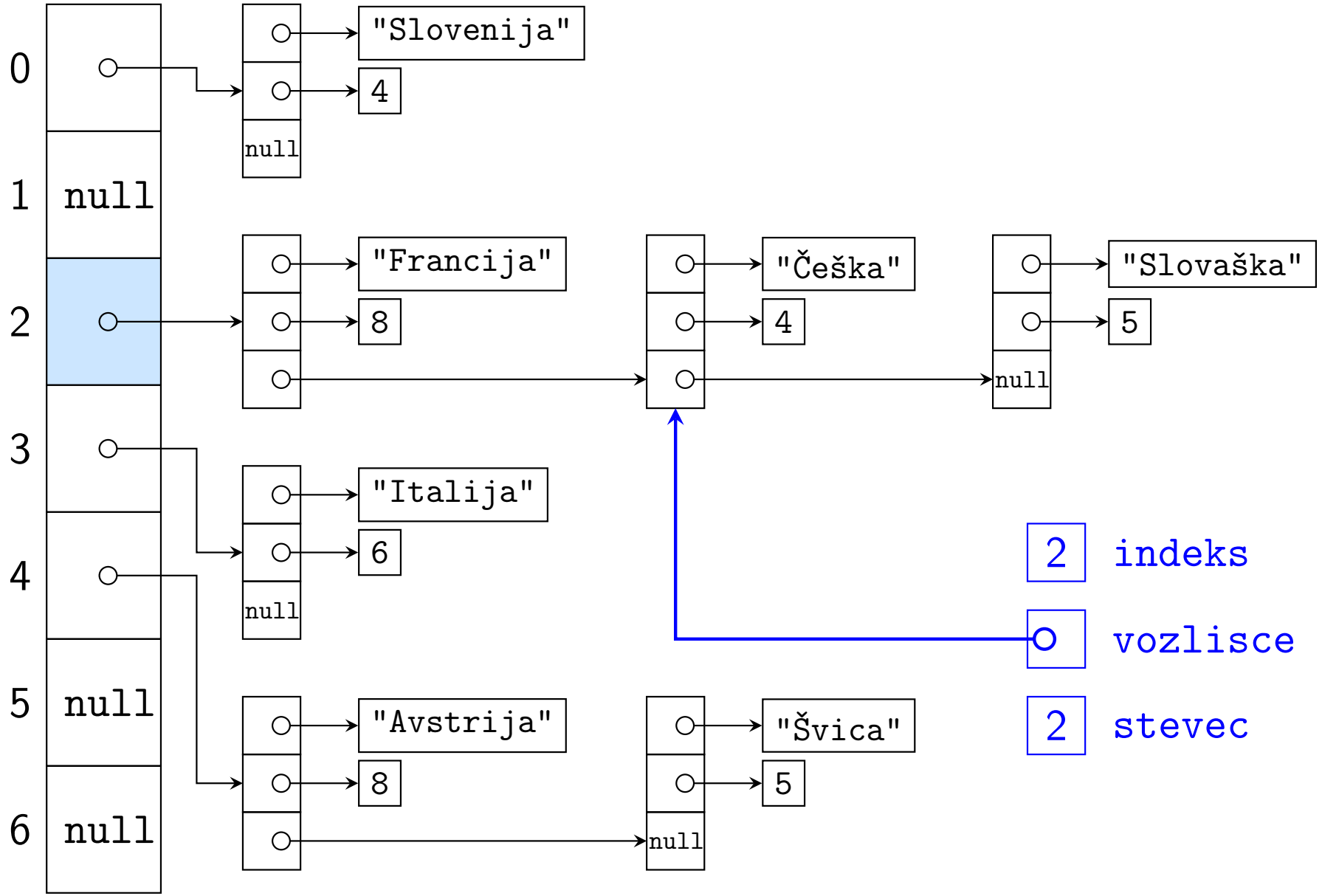
Sprehod po slovarju



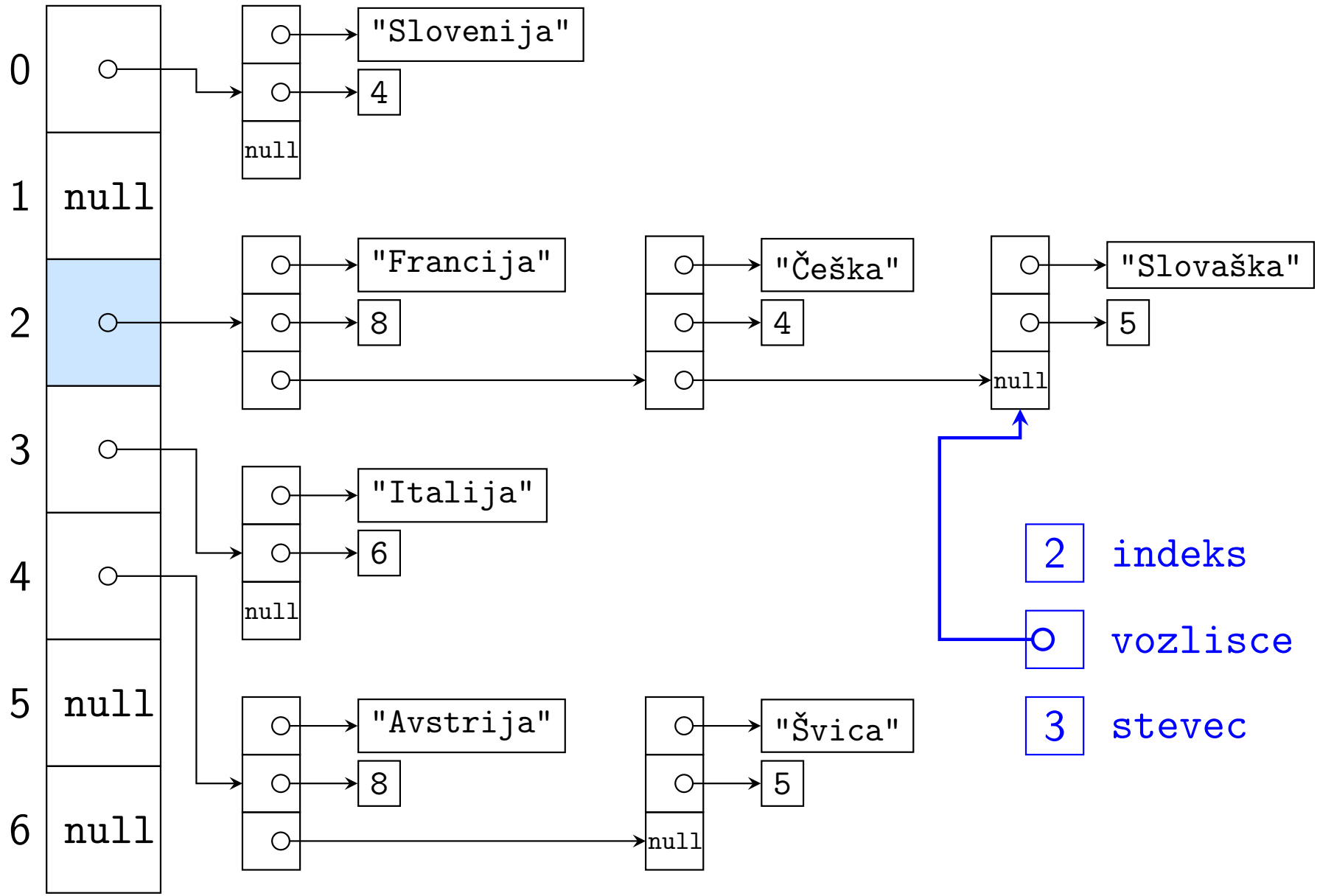
Sprehod po slovarju



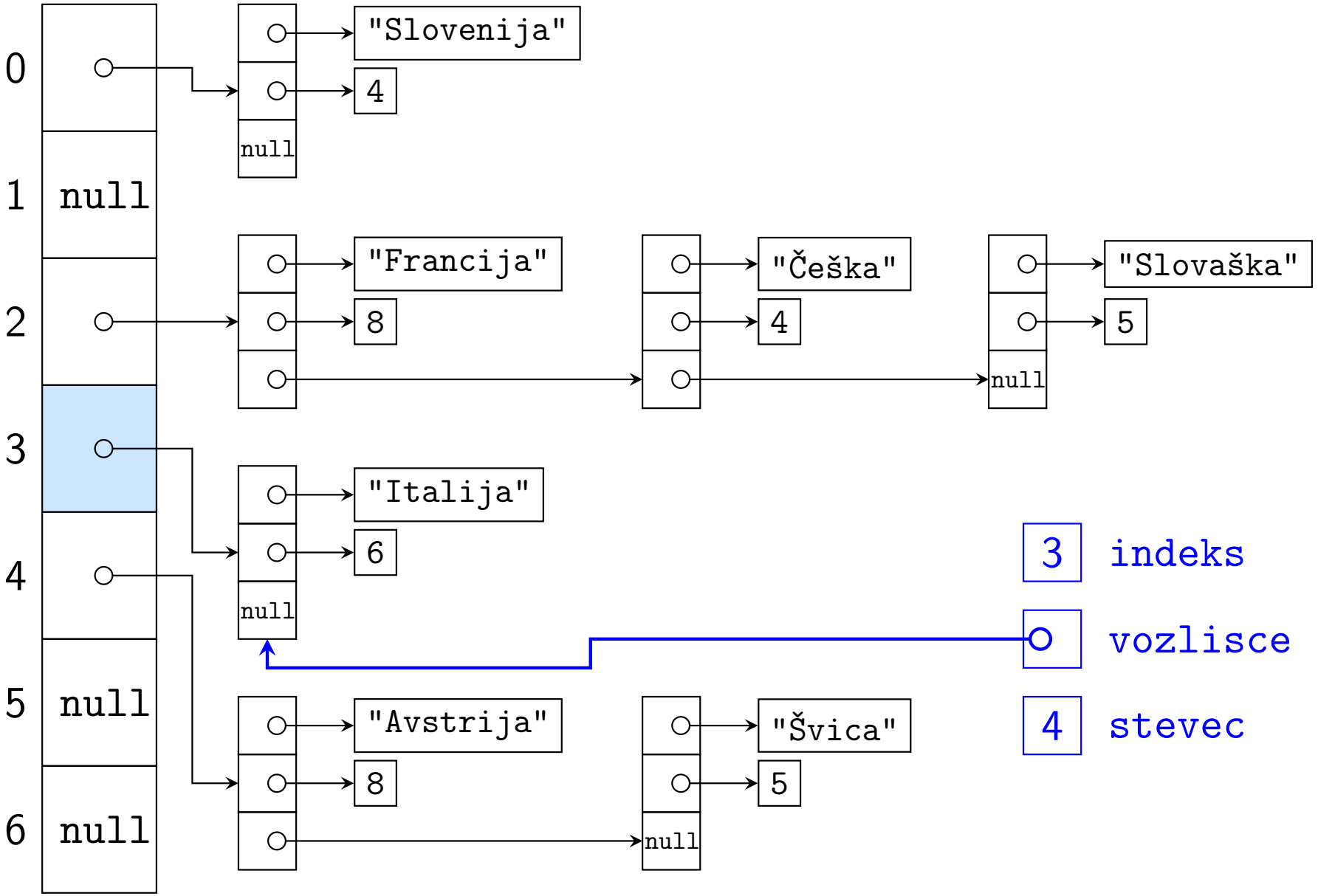
Sprehod po slovarju



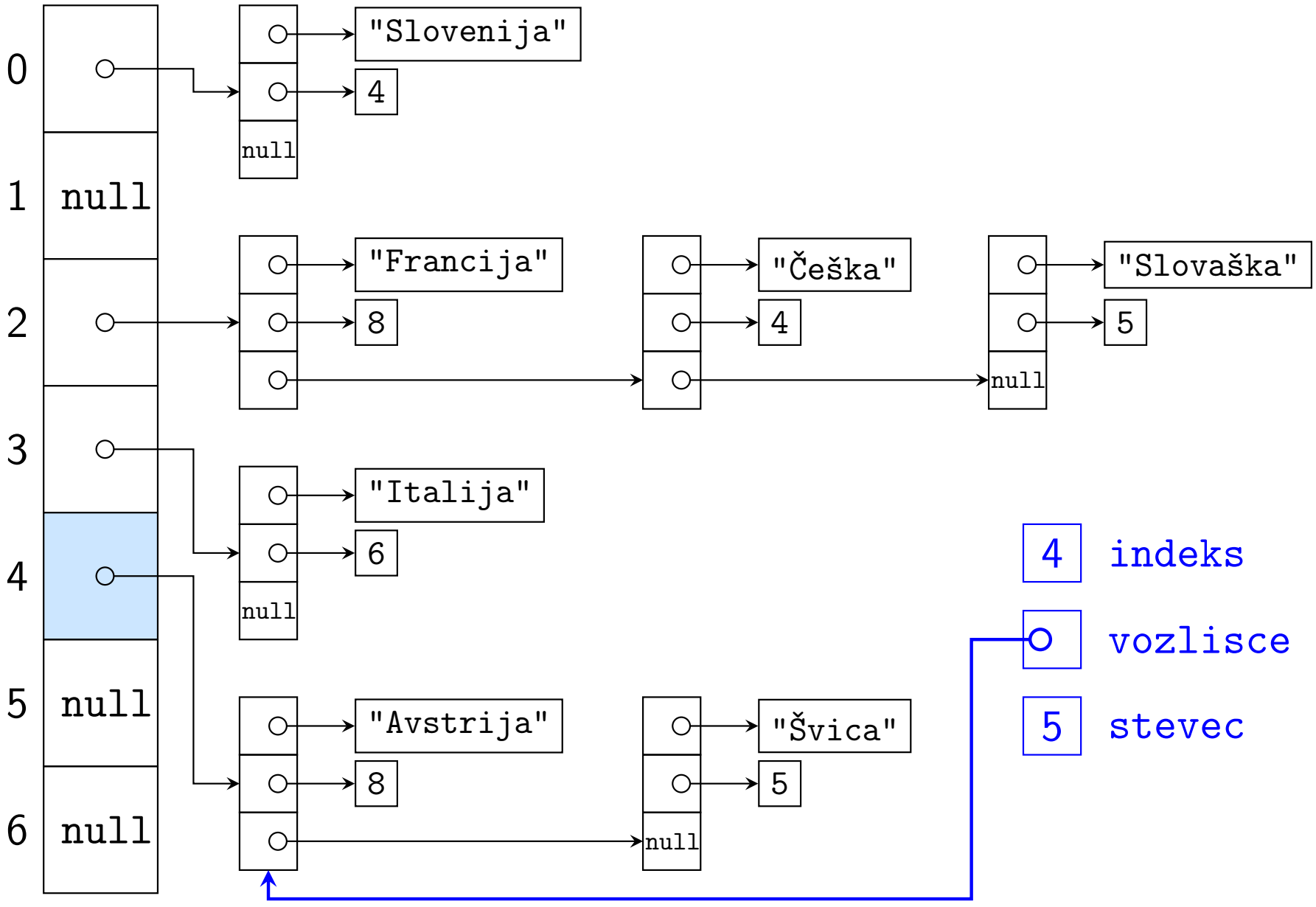
Sprehod po slovarju



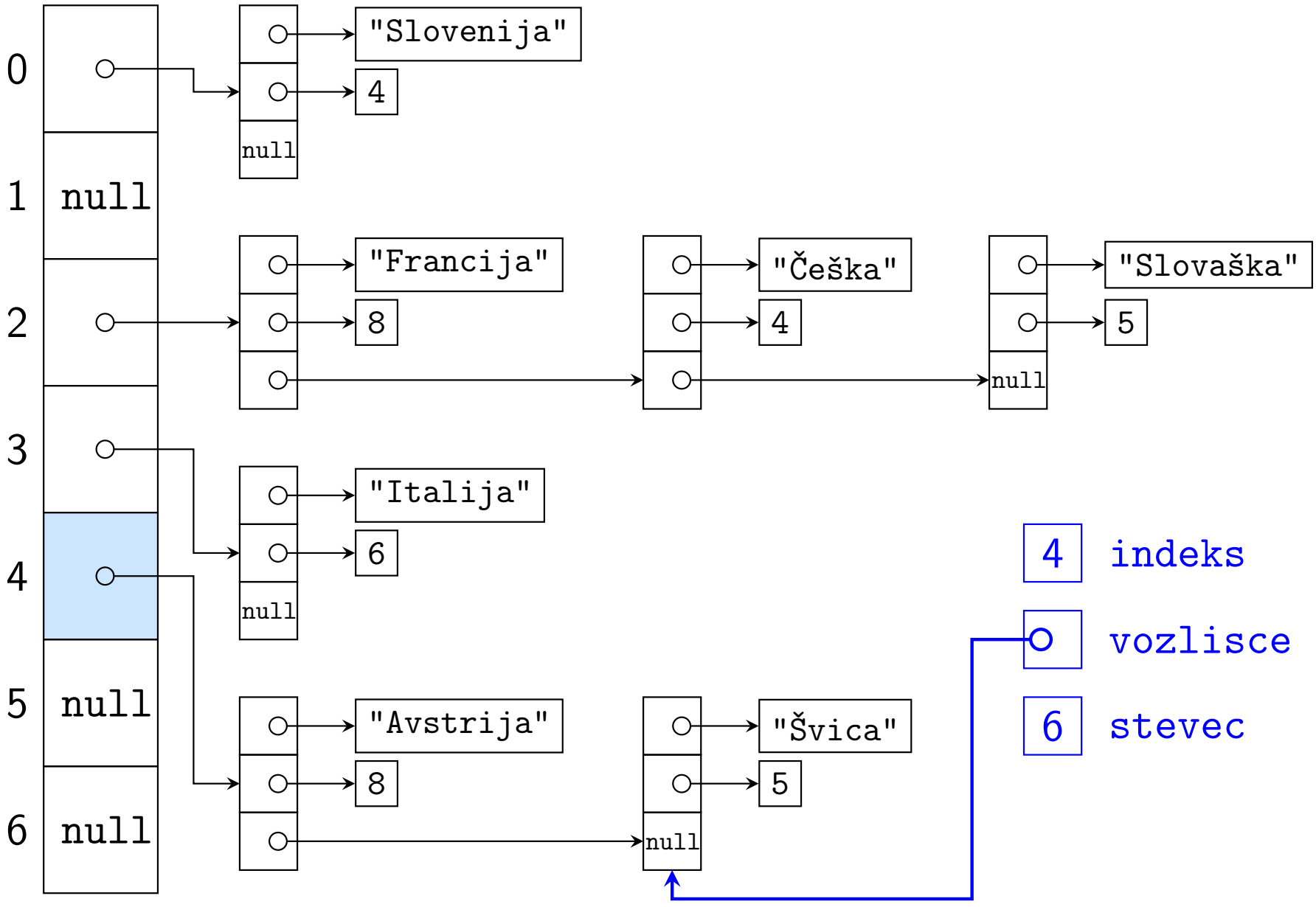
Sprehod po slovarju



Sprehod po slovarju



Sprehod po slovarju



Razred IteratorPoKljucih

```
private static class IteratorPoKljucih<K, V> implements Iterator<K> {
    private Slovar<K, V> slovar;
    private int indeks;
    private int stevec;
    private Vozlisce<K, V> vozlisce;

    public IteratorPoKljucih(Slovar<K, V> slovar) {
        this.slovar = slovar;
        this.indeks = -1;
        this.stevec = 0;
        this.vozlisce = null;
    }

    @Override
    public boolean hasNext() {
        return (this.stevec < this.slovar.stParov);
    }
    ...
}
```

Razred IteratorPoKljucih

```
@Override
public K next() {
    if (!this.hasNext()) {
        throw new NoSuchElementException();
    }
    if (this.indeks < 0 || this.vozlisce.naslednje == null) {
        // poišči naslednji neprazni povezani seznam
        do {
            this.indeks++;
        } while (this.indeks < this.slovar.podatki.length &&
                 this.slovar.podatki[this.indeks] == null);
        this.vozlisce = this.slovar.podatki[this.indeks];
    } else {
        // premakni se na naslednje vozlišče v trenutnem povezanem seznamu
        this.vozlisce = this.vozlisce.naslednje;
    }
    this.stevec++;
    return this.vozlisce.kljuc;
}
```