acs
applied computer science

UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA
KATEDRA ZA PRIMENJENE RAČUNARSKE NAUKE

Osnovi programiranja i programskih jezika

Računarske vežbe – vežba 10

Zimski semestar 2025/2026.

Studijski program: Informacioni inženjering

Rekurzivne funkcije
Binarna stabla
Heš tabele

Rekurzivne funkcije

- Rekurzija nastaje kada se pojam definiše pomoću samog sebe
- Rekurzivna funkcija u svom telu ima poziv same sebe
- Rekurzivna funkcija mora sadržati uslov za poziv same sebe

Primer rekurzivne funkcije

- Rekurzivno računanje faktorijela

$$n! = \begin{cases} n \cdot (n-1)!, & n > 0 \\ 1, & n = 0 \end{cases}$$

```
int faktorijel(int n) {
    if (n <= 1)
        return 1;
    else
        return n*faktorijel(n-1);
}
```

Binarno traženje

```
int binarySearch(int arr[], int l, int r, int x) {
    if (r >= l) {
         int mid = l + (r - l)/2;

         if (arr[mid] == x)  return mid;

         // Ako je manji od središnje, tražimo u levoj polovini niza
         if (arr[mid] > x)
              return binarySearch(arr, l, mid-1, x);
         // U suprotnom, tražimo u desnoj polovini niza
         else
              return binarySearch(arr, mid+1, r, x);
    }

    // Element nije nađen
    return -1;
}
```

Zadatak 1

Napisati C program kojim će se pomoću rekurzivne funkcije vršiti konverzija nenegativnih celih brojeva iz dekadnog zapisa u binarni.

Primeri poziva funkcije:

- toBinary(17) vraća broj 10001
- toBinary(53) vraća broj 110101

Stabla

- Nelinearna struktura podataka
- Modeluju hijerarhijski odnos među elementima
- Svaki element ima:
  - tačno jednog roditelja (osim korenskog)
  - nula ili više dece
- Element koji nema dece zove se list

Binarna stabla

- Uređena stabla kod kojih svaki čvor ima najviše dvoje dece
- Podstablo čiji je koren levo (desno) dete zove se levo (desno) podstablo
- Realizacija
  - sekvencijalna
  - spregnuta

Binarna stabla

Operacije na binarnim stablima:

- generičke
  - size
  - isEmpty
- pristupne
  - root
  - parent
  - left
  - right
  - sibling

Binarna stabla

Operacije nad binarnim stablima:

- upiti
  - isLeaf – da li je list
  - isRoot – da li je koren
- ažuriranje
  - insert – dodavanje elementa
  - delete – brisanje elementa

Obilazak binarnog stabla

Vrste obilazaka

- preorder – čvor, levi, desni
- postorder – levi, desni, čvor
- inorder – levi, čvor, desni

Binarno stablo – spregnuta realizacija

Čvor stabla sadrži:

- podatak (ili pokazivač na podatak)
- pokazivač na roditeljski čvor
- pokazivač na levo dete
- pokazivač na desno dete

```
typedef struct node {
    int key;
    struct node *parent, *left, *right;
} TNODE;
```

Dodavanje elementa u stablo

```
void insert(TNODE **proot, int key) {
    TNODE *new = makeNode(key);

    if (*proot == NULL) {
        *proot = new;
        return;
    }

    insertNode(*proot, new);
}
```

```
TNODE* makeNode(int key) {
    TNODE *new;
    new = (TNODE*)malloc(sizeof(TNODE));

    new->key = key;
    new->left = new->right = NULL;
    new->parent = NULL;

    return new;
}
```

```
void insertNode(TNODE *root, TNODE *new) {
    if (root->key == new->key) {
        free(new);
        return;
    }

    if (root->key < new->key) {
        if (root->right == NULL) {
            new->parent = root;
            root->right = new;
        } else
            insertNode(root->right, new);
    } else {
        if (root->left == NULL) {
            new->parent = root;
            root->left = new;
        } else
            insertNode(root->left, new);
    }
}
```

Traženje elementa u stablu

```
TNODE* find(TNODE *node, int key) {
    if (!node)
        return NULL;

    if (node->key == key) return node;

    if (node->key > key)
        return find(node->left, key);
    else
        return find(node->right, key);
}
```

Brisanje elementa iz stabla (1/2)

```
void delete(TNODE **proot, int key) {
    TNODE *del = find(*proot, key);

    if (!del) return;

    if (*proot == del) {
        if (del->left == NULL) {
            *proot = del->right;
            if (*proot != NULL)
                (*proot)->parent = NULL;
            free(del);
            return;
        }
        if (del->right == NULL) {
            *proot = del->left;
            (*proot)->parent = NULL;
            free(del);
            return;
        }
    }
    deleteNode(del);
}
```

Brisanje elementa iz stabla (2/2)

```
void deleteNode(TNODE *node) {
    TNODE *parent = node->parent;
    // Ako je list
    if (isLeaf(node)) {
        if (parent->left == node)
            parent->left = NULL;
        else
            parent->right = NULL;
        free(node);
        return;
    }
    // Ako ima tačno jedno dete
    if (node->left == NULL || node->right == NULL){
        TNODE *child;
        if (node->left) child = node->left;
        if (node->right) child = node->right;
        child->parent = parent;
        if (parent->left == node)
            parent->left = child;
        else
            parent->right = child;
        free(node);
        return;
    }
    // Ako ima oba deteta
    TNODE *maxLeft = node->left;

    // Pratimo pokazivač do krajnjeg desnog
    // elementa (najvećeg) u levom podstablu
    while (maxLeft->right)
        maxLeft = maxLeft->right;

    // Zapamtimo ključ tog elementa
    int maxLeftKey = maxLeft->key;

    // Obrišemo taj element
    deleteNode(maxLeft);

    // Sačuvani ključ dodelimo elementu koji
    // je trebalo obrisati
    node->key = maxLeftKey;
}
```

Inorder obilazak i visina stabla

```c
void inorder(TNODE *root) {
    if (root->left)
        inorder(root->left);

    printf("%d ", root->key);

    if (root->right)
        inorder(root->right);
}
```

```c
int height(TNODE *node) {
    if (!node) return 0;

    int l, r;
    l = height(node->left);
    r = height(node->right);

    return l > r ? l+1 : r+1;
}
```

Brisanje stabla

```
void destroy(TNODE *node) {
    if (node) {
        destroy(node->left);
        destroy(node->right);
        free(node);
    }
}
```

Zadatak 2

Napisati C program koji korisniku nudi meni sa sledećim operacijama nad binarnim stablom:

- unos novog elementa
- brisanje postojećeg elementa
- traženje elementa u stablu
- obilazak stabla (inorder, preorder, postorder)
- prikaz
    - visine stabla
    - broja elementa u stablu

Zadatak 3

Proširiti Zadatak 2 tako da nudi i sledeće akcije:

- izračunavanje sume elemenata niza
- ispis svih listova
- prikaz
  - broja elemenata na N-tom nivou u stablu
  - broja elemenata koji imaju oba deteta

Zadatak za domaći 1

Napisati C program kojim se iz tekstualne datoteke `"studenti.txt"` za svakog studenta učitava broj indeksa, ime, prezime i broj osvojenih poena na testu i upisuje u binarno stablo sortirano po broju indeksa. U izlaznu datoteku `"polozili.txt"` ispisati podatke o studentima koji su osvojili više poena od srednje vrednosti brojeva poena dvoje studenata koji su najbolje i najlošije uradili test.

Zadatak za domaći 2

Napisati C program kojim se iz tekstualne datoteke `"klubovi.txt"` za svaki klub učitava naziv kluba, oznaka konferencije, broj pobeda i broj poraza. Za svaku konferenciju kreirati binarno stablo sa klubovima sortiranim u opadajućem odnosu broja pobeda i poraza. Podatke o najboljih 8 klubova iz svake konferencije ispisati u datoteku čiji se naziv dobija spajanjem oznake konferencije i stringa `"_playoff.txt"`.

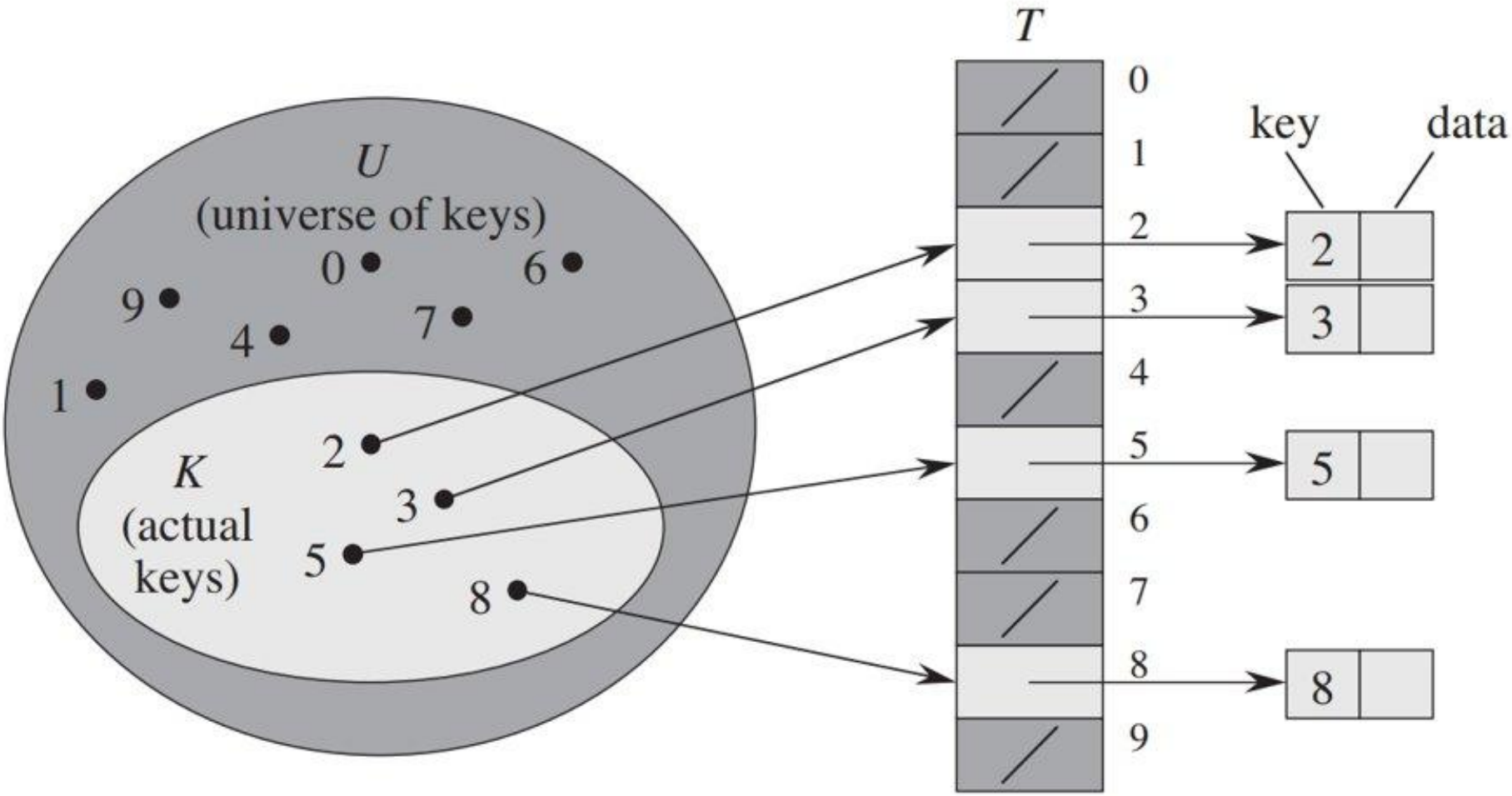
Rečnik (*dictionary*)

- Apstraktni tip podataka
- Sastoji se od skupa elemenata gde svaki element ima ključ
- Operacije
  - Dodavanje
  - Brisanje
  - Traženje po ključu
- Poželjno imati ove operacije u konstantom vremenu

Primitivni pristup – direktno adresiranje

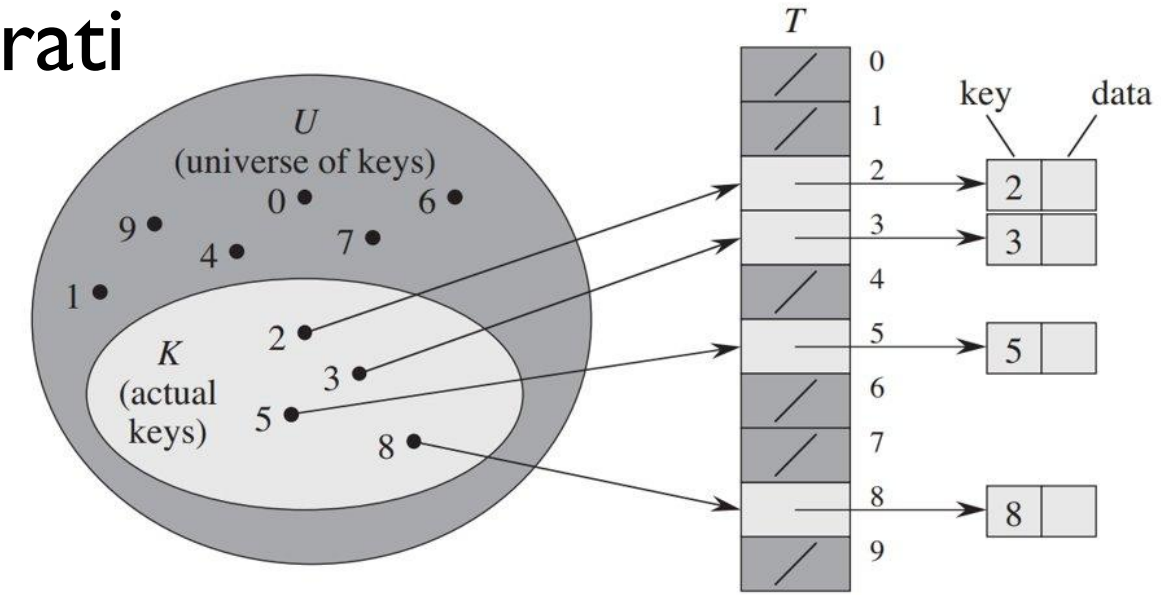
- Niz kao reprezentacija rečnika
- Pristup po ključu
- Celobrojne vrednosti kao ključevi
- Nedostaci
  - Ključevi nekog drugog tipa
  - Velika potrošnja memorije

Direktno adresiranje - primer



Heš tabela

- Poznata i kao heš mapa
- Heširanje - transformacija ključa u poziciju elementa
- Implementacija pomoću niza
- Neophodno izabrati
  - heš funkciju



Problem kolizije ključeva

- Problem kolizije ključeva nastaje kada za dva ključa *a* i *b*, *a* != *b* i funkciju *h* važi *h*(*a*) = *h*(*b*)
- Osnovni problem kod heš tabela
- Ne postoji univerzalno rešenje

Rešavanje kolizije ključeva - lančanje

- Svi elementi čiji su ključevi u koliziji smeštaju se u jednostruko spregnutu listu
- Odnos broja elemenata i veličine mape zove se faktor popunjenosti
- Ako heš funkcija ravnomerno "raspoređuje" ključeve po mapi i ako je faktor popunjenosti a tada za vreme operacija pristupa, dodavanja i brisanja $\Theta(1+a)$

Heš funkcija

- Kako izabrati odgovarajuću heš funkciju?
- Teoretski – dobra heš funkcija treba imati svojstvo da uniformno raspoređuje elemente u mapi
  - pomenuto svojstvo je teško utvrditi u praksi
- Heš funkciju treba birati tako da bude nezavisna od bilo kakvih šablona u ulaznim podacima (ključevima)

Heš funkcija

- Neke metode za heš funkcije
  - Metod ostatka
    - $h(x) = x \bmod a$
  - Metod množenja
    - $h(x) = [m(Ax \bmod 1)]$
    - m – uglavnom stepen 2
    - $0 < A < 1$
    - Knuth predlaže $A = (\sqrt{5}-1)/2$

Heš funkcija za stringove

- Jednostavan pristup – posmatrati ASCII string kao broj u brojnom sistemu sa osnovom 128
  - Potencijalno mnogo neiskorišćenog prostora
- Suma ASCII vrednosti karaktera u stringu
  - Potencijalno mnogo kolizija ("ab" i "ba" su u koliziji)
- djb2
  - `hash(i) = hash(i - 1) * 33 + str[i]`
  - `hash(-1) = 5381`
- sdbm
  - `hash(i) = hash(i - 1) * 65599 + str[i]`

Univerzalno heširanje

- Randomizacija heš funkcije
- $h(k) = [(ak+b) \bmod p] \bmod m$
  - p je prost broj veći od ukupnog broja ključeva
  - a i b su slučajni brojevi iz skupa $[0..p-1]$
  - m broj mesta u heš mapi
- U najgorem slučaju verovatnoća kolizije ključeva $1/m$

Otvoreno adresiranje

- Drugi pristup rešavanju kolizije ključeva
- Svi elementi se nalaze u tabeli – nema lančanja
- Prilikom dodavanja elementa se radi *probing*
  - Pokušava se naći sledeće slobodno mesto u skladu sa nekom funkcijom od rednog broja pokušaja
    - Linearno
    - Kvadratno
    - Sekundarno heširanje

Zadatak 4

Napisati C program koji implementira rad sa heš tabelom (rečnikom) sa celobrojnim ključevima. Kao heš funkciju uzeti ostatak pri deljenju sa nekim prostim brojem (npr. 73). Kolizije ključeva rešavati otvorenim adresiranjem, odnosno traženjem sledećeg slobodnog mesta u nizu.

Zadatak za domaći 3

Napisati C program koji implementira rad sa heš tabelom (rečnik) sa ključevima tipa string. Kao heš funkciju koristiti *djb2*. Kolizije ključeva rešavati otvorenim adresiranjem i ponovnim heširanjem pomoću *sdbm*.