Elektrotehnički fakultet Sarajevo
Odsjek za računarstvo i informatiku
Predmet: Algoritmi i strukture podataka

Pripremna zadaća za Vježbu 9

Rok je naveden u informacionom sistemu zamger.
Cilj vježbe je upoznavanje sa konceptom hash-funkcije i strukturom podataka hash-mapa.

Zadatak 1

Hash-funkcija (heš funkcija)

Hash funkcija je funkcija koja prima jedan podatak određenog tipa (cijeli broj, string ili nekog drugog) i transformiše ga u podatak određenog izlaznog tipa koji nije nužno isti kao ulazni tip. Ovisno o primjeni, hash-funkcije imaju sljedeće osobine:

1. Domen ulaznih vrijednosti je tipično neograničen (npr. string proizvoljne dužine), ali domen izlaznih vrijednosti mora biti striktno ograničen (npr. cijeli broj u nekom datom opsegu).
2. Funkcija mora za dati isti ulaz uvijek vratiti isti izlaz.
3. Ako se funkcija pravi za primjenu u kriptografiji, ne smije postojati način da se iz izlaza odredi ulaz (reverzno hashiranje). Čak ne smije postojati način da se nasluti ulaz odnosno da se odredi ikakva korisna informacija o ulazu (tipa veličina ulaza, redoslijed kojim je niz ulaza dat i slično).
4. S druge strane, ako se funkcija pravi za primjenu u hash mapama ili tabelama, prioritet je da hash funkcija ima što bolje performanse odnosno da se što brže izračunava.
5. Poželjno je da statistička distribucija izlaznih vrijednosti na datom domenu bude što uniformnija. Npr. recimo da je ulaz cijeli broj tipa int a izlaz je cijeli broj u opsegu 1-1000. Ako korisnik funkciju poziva redom za brojeve 1, 2, … n, poželjno je da n izračunatih hash vrijednosti budu ravnomjerno raspoređene na opsegu 1-1000.

Pošto izlazni domen može biti manji od ulaznog, zaključujemo da se kod takvih hash funkcija mora dešavati da se za dvije različite ulazne vrijednosti dobije isti izlaz. Ova pojava se naziva *kolizija*. Poželjno je da hash funkcija za neki tipičan skup ulaznih vrijednosti ima što manje kolizija. Ipak, svaki program koji koristi hash-funkciju mora računati na mogućnost kolizije i planirati je.

Hash funkcije se koriste u kriptografiji i sigurnosti (npr. tipično se u bazama korisnika ne drži šifra korisnika nego hash šifre, tako da čak ni administrator ne može znati šifru korisnika ali se može provjeriti da li je neka unesena šifra tačna), za implementaciju hash-mape (mapa koja koristi hash funkciju za bolje performanse), za kontrolu integriteta podataka (checksumi, hash liste i hash stabla) itd. Poznati primjeri hash funkcija su MD5, SHA-1 i SHA-2, CRC (Cyclic Redundancy Check).

Napisati funkciju

```
unsigned int hash(string ulaz, int max)
```

Funkcija treba izračunavati hash primljenog stringa ulaz, a izlaz treba biti cijeli broj u opsegu 0-max tako da su zadovoljena ranije data pravila.

Možemo zamisliti trivijalnu hash funkciju koja samo sumira ASCII vrijednosti karaktera, a zatim operatorom modulo (%) osigurava da je rezultat u traženom opsegu:

```
unsigned int hash(string ulaz, unsigned int max) {
      unsigned int suma=0;
      for (int i(0); i<ulaz.length(); i++)
```

```
            suma += ulaz[i];
        return suma % max;
}
```

Pokazuje se da za neke realne primjene ova funkcija daje veliki broj kolizija. Ne samo da anagrami (preslaganje slova) daju isti hash nego i npr. stringovi “TATA” i “TELE” (84+65+84+65=298, 84+69+76+69=298). Istraživanje je pokazalo da najmanji broj kolizija uz najbolje performanse daje kombinacija množenja i sabiranja pri čemu se kao konstante koriste ko-prosti brojevi[1]. Primjer takve funkcije je djbhash funkcija čiji je autor Dan J. Bernstein:

```
unsigned int djbhash(string ulaz, unsigned int max) {
      unsigned int suma=5381;
      // 5381 je pocetna vrijednost koja poboljsava distribuciju
      for (int i(0); i<ulaz.length(); i++)
            suma = suma*33 + ulaz[i];
      return suma % max;
}
```

Generalno bolje performanse se dobijaju korištenjem pokazivačke aritmetike (C stringovi umjesto C++) i ako umjesto množenja koristimo binarne operatore (shift) pa se ova funkcija može efikasnije napisati ovako:

```
unsigned int djbhash(char* ulaz, unsigned int max) {
      unsigned int suma=5381;
      int c;
      while (c = *ulaz++)
            suma = ((suma << 5) + suma) + c;

      return suma % max;
}
```

Zadatak. Hash-mapa

Napravite klasu HashMapa izvedenu iz klase Mapa koju ste razvili u pripremi za vježbu 7. Klasa HashMapa će biti po svemu vrlo slična klasi NizMapa uz jednu vrlo bitnu razliku:
U NizMapi novi element smo dodavali na prvo slobodno mjesto u nizu. Kod HashMape indeks u nizu ključeva i vrijednosti treba biti hash ključa.

- Operator [] najprije treba pronaći ključ u nizu ključeva. To se radi tako što se izračuna hash ključa pomoću hash funkcije i na dobijenom indeksu je ključ.
- Parametar **max** hash funkcije predstavlja trenutnu dimenziju niza (kao i kod NizMape, nizove ćemo na početku alocirati na neki početni kapacitet a kada se on popuni vršiti proširivanje).
- Zatim treba provjeriti da li se na datom indeksu nalazi ključ. Naime, ako je došlo do kolizije, više različitih ključeva će imati isti hash.
- Ako se ključ nalazi u nizu ključeva, vrijednost je na istom indeksu u nizu vrijednosti.
- U suprotnom dodajemo ključ u niz ključeva a vrijednost u niz vrijednosti.
- Koja je onda vremenska složenost pronalaska elementa u hash-mapi i dodavanja novog elementa (zapamtite da je **n** ovdje veličina niza)?

Pošto je klasa Mapa generička mi ustvari ne znamo koji je TipKljuca, a implementacija hash funkcije zavisi od tipa ulaznih vrijednosti! Ovaj problem ćemo riješiti tako što ćemo dodati metodu kojom korisnik može zadati hash funkciju:

---

1 Ko-prosti broj je složeni broj koji ima samo jednu moguću faktorizaciju i to sa dva prosta faktora.

- Metoda **definisiHashFunkciju** treba primati *pokazivač na funkciju* sa dva parametra: prvi parametar je tipa TipKljuca i predstavlja ulaz, a drugi je tipa unsigned int i predstavlja opseg izlaza. Povratna vrijednost hash funkcije je tipa unsigned int i predstavlja izračunati hash. Nakon poziva metode definisiHashFunkciju primljeni pokazivač treba pohraniti u atribut, tako da naredni pozivi operatora [] koriste ovu funkciju. Ako funkcija nije definisana, operator [] treba baciti izuzetak.

Ostaje problem ***kolizija***. Za rješavanje kolizija koristite najjednostavniji pristup obrađen na predavanjima a to je *otvoreno adresiranje sa linearnim ispitivanjem*.

Ovu klasu HashMapa dodajte u program koji ste razvili na vježbi 7 i ovih zadataka. Znači sada biste u vašem programu trebali imati klase:

- Mapa (apstraktna klasa)
- NizMapa
- BinStabloMapa
- **HashMapa**

Pored toga biste trebali imati i glavni program koji poredi sve ove klase na način da poredi performanse dodavanja novog člana, pristupa postojećem članu i brisanja člana, pri čemu naravno morate koristiti dovoljno veliki broj elemenata da bi se ta razlika uopšte vidjela. **Obavezno stavite i komentare kojim objašnjavate razlike između izmjerenih rezultata.**