ОДСЕК ЗА СОФТВЕРСКО ИНЖЕЊЕРСТВО

АЛГОРИТМИ И СТРУКТУРЕ ПОДАТАКА 2

2023-2024

- трећи домаћи задатак -

Опште напомене:

1. Домаћи задатак 3 састоји се од два програмска проблема. Студенти проблем решавају **самостално**, на програмском језику C++.
2. Пре одбране, сви студенти раде тест знања за рачунаром коришћењем система *Moodle* (http://elearning.rcub.bg.ac.rs/moodle/). **Сви студенти треба да се пријаве на курс пре почетка лабораторијских вежби**. Пријава на курс ће бити прихваћена и важећа само уколико је студент регистрован на систем путем свог налога електронске поште на серверу mail.student.etf.bg.ac.rs.
3. Реализовани програми треба да комуницирају са корисником путем једноставног менија који приказује реализоване операције и омогућава сукцесивну примену операција у произвољном редоследу.
4. Унос података треба омогућити било путем читања са стандардног улаза, било путем читања из датотеке.
5. Решења треба да буду отпорна на грешке и треба да кориснику пруже јасно обавештење у случају детекције грешке.
6. Приликом оцењивања, биће узето у обзир рационално коришћење ресурса. **Примена рекурзије се неће признати као решење проблема које може освојити максималан број поена.**
7. За све недовољно јасне захтеве у задатку, студенти треба да усвоје разумну претпоставку у вези реализације програма. Приликом одбране, демонстраторе треба обавестити која претпоставка је усвојена (или које претпоставке су усвојене) и која су ограничења програма (на пример, максимална димензија низа). Неоправдано увођење ограничавајуће претпоставке повлачи негативне поене.
8. Предаја домаћег задатка ће бити омогућена преко *Moodle* система. Детаљније информације ће бити благовремено објављене.
9. Одбрана домаћег задатка ће се обавити према распореду који ће накнадно бити објављен на сајту предмета.
10. Други проблем је исти за све студенте, а формула за редни број **i** метода који треба користити приликом решавања првог проблема је следећа:

    (**R** – редни број индекса, **G** – последње две цифре године уписа):

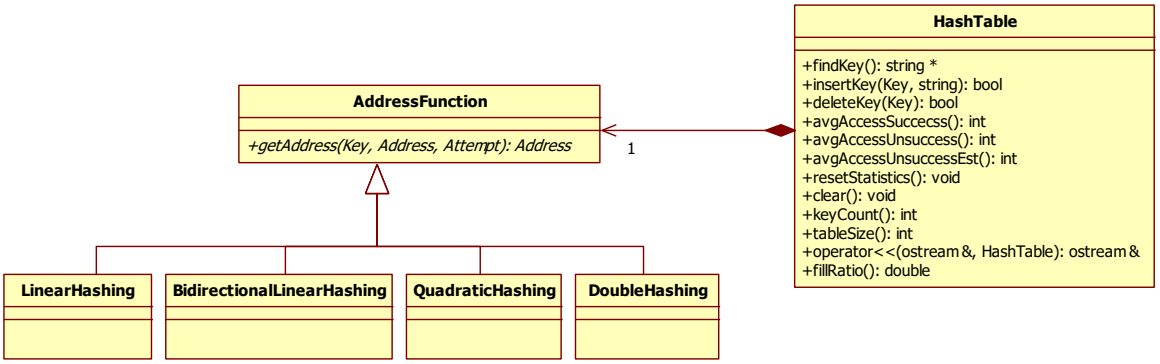
    i = (R + G) mod 4
11. Предметни наставници задржавају право да изврше проверу сличности предатих домаћих задатака и коригују освојени број поена након одбране домаћих задатака, као и да пријаве теже случајеве повреде Правилника о дисциплинској одговорности студената Универзитета у Београду Дисциплинској комисији Факултета.

19.12.2023. године *Са предмета*

Проблем 1 – хеш табела [70 поена]

Написати на језику C++ потребне класе за реализацију хеш табеле у коју се умећу подаци типа знаковне ниске (*string*) индексирани целобројним кључевима. За разрешавање колизије се примењује техника отвореног адресирања.

Концептуални дијаграм класа је приказан на следећој слици:



Класа која представља апстракцију адресне функције коју ће хеш табела користити за разрешавање колизије се задаје објекту хеш табеле приликом њеног стварања.

[30 поена] Имплементација хеш табеле

Величина хеш табеле се задаје приликом креирања табеле и не мења се током извршавања. Приликом уметања кључа, додељена адресна функција се позива само ако је матична адреса заузета. Класа **`HashTable`** треба да реализује следеће јавне методе:

- **`string findKey(Key k)`** – проналази задати кључ и враћа показивач на одговарајућу ниску (string) (0 ако се кључ не налази у табели)
- **`bool insertKey(Key k, string s)`** – умеће кључ и пратећи информациони садржај у табелу и враћа статус (***true*** за успешно уметање, ***false*** за неуспешно). Спречити уметање постојећег кључа.
- **`bool deleteKey(Key k)`** – брише кључ из табеле и враћа статус успеха (***true*** за успешно брисање, ***false*** за неуспешно)
- **`int avgAccessSuccess()`** – враћа просечан број приступа табели приликом успешног тражења кључева
- **`int avgAccessUnsuccess()`** – враћа просечан број приступа табели приликом неуспешног тражења кључа израчунат на основу броја (до тог тренутка) неуспешних приступа табели и броја кључева који нису нађени у табели
- **`void resetStatistics()`** – поставља све податке потребне за бројање приступа ради одређивања просечног броја приступа за неуспешно тражење кључа на почетну вредност
- **`void clear()`** – празни табелу (брише све кључеве)
- **`int keyCount()`** – враћа број уметнутих кључева
- **`int tableSize()`** – враћа величину табеле
- **`operator<<`** – испис садржаја табеле на стандардни излаз, у сваком реду по један улаз табеле. Празне улазе табеле означити са "EMPTY".
- **`double fillRatio()`** – враћа степен попуњености табеле (реалан број између 0 и 1)

Исправна реализација хеш табеле подразумева да, поред наведених метода, постоје друге потребне методе (попут конструктора и деструктора). Студентима се препушта да у класу додају оне методе које сматрају потребним за успешну реализацију.

[20 поена] Имплементација апстрактне класе адресне функције и једне од метода отвореног адресирања

Апстрактна класа декларише јавну методу коју користи класа **`HashTable`** за одређивање наредне адресе приликом разрешавања колизије.

```
Address getAddress(Key k, Address a, Attempt i);
```

Параметри ове методе су:

`k` – кључ,

`a` – матична адреса,

`i` – редни број покушаја приступа.

Метода враћа нову адресу на којој треба потражити кључ (или локацију где га треба сместити). **Враћена адреса може бити ван опсега адреса табеле**. Хеш табела која користи ову класу треба о томе да води рачуна и враћену адресу сведе на исправан опсег. Изведене класе треба да конкретизују начин одређивања следеће адресе.

У зависности од редног броја проблема који се решава, реализовати следећу методу за решавање колизија:

0. Линеарно адресирање
1. Бидирекционо линеарно адресирање
2. Квадратно адресирање
3. Двоструко хеширање

Класа за линеарно хеширање се параметризује кораком **`s`** тако да као резултат даје матичну адресу увећану за **`i`** $\cdot$ **`s`** као резултат.

$$\texttt{return\_address} = a + i \cdot s$$

Класа за бидирекционо линеарно адресирање се параметризује кораком **`s`** тако да за непарно **`i`** враћа матичну адресу увећану за **`i`** $\cdot$ **`s`**, а за парно **`i`** враћа матичну адресу умањену за **`(i-1)`** $\cdot$ **`s`**.

$$\texttt{return\_address} = a + i \cdot s, \text{ за } i \bmod 2 = 1$$

$$\texttt{return\_address} = a - (i-1) \cdot s, \text{ за } i \bmod 2 = 0$$

Класа за квадратно адресирање се параметризује коефицијентима c_1 и c_2, а у **`i`** –том покушају враћа вредност према следећој формули:

$$\texttt{return\_address} = a + c_1 \cdot i + c_2 \cdot i^2$$

Класа за двоструко хеширање се параметризује подацима **`p`** и **`q`** и враћа следећу вредност:

$$\texttt{return\_address} = a + i \cdot (q + (k \bmod p))$$

[20 поена] Евалуација перформанси и главни програм

Евалуација перформанси хеш табеле се врши уметањем у задату хеш табелу задатих кључева (у одређеном опсегу вредности), а затим генерисањем задатог броја кључева псеудослучајних вредности и вршењем претраге на њих. Након тога се исписују резултати (просечан број приступа при успешној претрази и израчунат број приступа при неуспешној претрази). Табела, скуп кључева који се умећу и број кључева на које

се врши претрага се задају као параметри функције. Опсег у коме се генеришу случајни бројеви одредити тако да одговара опсегу вредности кључева уметнутих у табелу.

Реализовати главни програм са једноставним интерактивним менијем који кориснику омогућава рад са јавним методама хеш табеле. Такође, главни програм треба да омогући учитавање знаковних низова и додељених кључева са стандардног улаза или задате датотеке и позивање описане функције за статистику за реализовану варијанту хеширања. Број кључева на које ће се вршити претрага треба да буде 10 пута већи од броја кључева који се умећу у табелу.

Уз поставку задатка је доступна датотека која садржи 10 000 линија са паровима реч-кључ (у свакој линији по један пар) која се може користити за тестирање решења.

Проблем 2 – Адаптивна хеш-табела [30 поена]

Модификовати решење из претходног задатка тако да подржава динамичку промену величине хеш табеле. Табела треба аутоматски да прати своје перформансе и да се прилагођава подацима тако да обезбеди ефикасно извршење операције претраге (што мањи број приступа). То ће се постићи тако што ће се, уколико дође до губитка перформанси, сви кључеви преместити у нову табелу повећаног капацитета. При том треба водити рачуна и о ефикасној употреби меморије.

Конкретно, као индикаторе перформанси треба користити:

- попуњеност табеле (табела се аутоматски проширује када попуњеност пређе задату границу)
- просечан број приступа приликом успешне или неуспешне претраге (табела се проширује у случају да је просечан број приступа већи од задате вредности).

За успешно решење овог дела задатка, потребно је конкретно формулисати и имплементирати наведена два критеријума за аутоматско проширење величине табеле.

Допунити главни програм тако да омогући тестирање извршених модификација.