ОДСЕК ЗА СОФТВЕРСКО ИНЖЕЊЕРСТВО

АЛГОРИТМИ И СТРУКТУРЕ ПОДАТАКА 2

2016-2017

- трећи домаћи задатак -

Опште напомене:

1. Домаћи задатак 3 састоји се од два програмска проблема. Студенти проблеме решавају **самостално**, на програмском језику C++.
2. Реализовани програми треба да комуницирају са корисником путем једноставног менија који приказује реализоване операције и омогућава сукцесивну примену операција у произвољном редоследу.
3. Унос података треба омогућити било путем читања са стандардног улаза, било путем читања из датотеке.
4. Решења треба да буду отпорна на грешке и треба да кориснику пруже јасно обавештење у случају детекције грешке.
5. Приликом оцењивања, биће узето у обзир рационално коришћење ресурса. **Примена рекурзије се неће признати као успешно решење проблема.**
6. 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 и слично). Неоправдано увођење ограничавајуће претпоставке повлачи негативне поене.
7. Одбрана трећег домаћег задатка ће се обавити у **среду, 11.01.2017. и четвртак, 12.01.2017.** према распореду који ће накнадно бити објављен на сајту предмета.
8. 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.
9. Формула за редни број **i** комбинације проблема коју треба користити приликом решавања **првог** задатка је следећа:

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

   $$\mathbf{i = (R + G) \bmod 4}$$

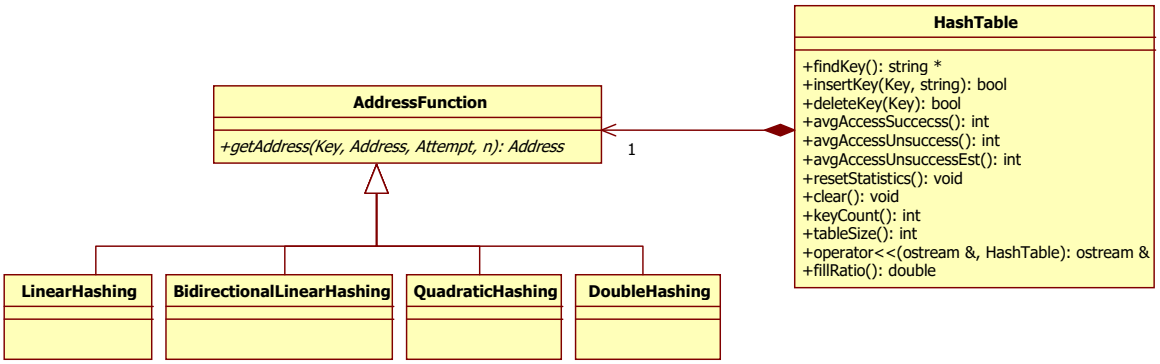
   Други задатак је исти за све студенте.
10. Имена датотека које се предају морају бити **dz3p1.cpp** и **dz3p2.cpp**
11. Предметни наставници задржавају право да изврше проверу сличности предатих домаћих задатака и коригују освојени број поена након одбране домаћих задатака.

27.12.2016. године *Са предмета*

Задатак 1 – Хеш табела [70 поена]

Написати на језику C++ потребне класе за реализацију хеш табеле у коју се умећу подаци типа знаковног низа (*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
- `int avgAccessUnsuccessEst()` – враћа просечан број приступа табели приликом неуспешног тражења кључа, добијен на основу процене базиране на тренутном степену попуњености табеле
- `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, Dim n);
```

Параметри ове методе су:

k – кључ,

a – матична адреса,

i – редни број покушаја приступа,

n – величина хеш табеле.

Метода враћа нову адресу на којој треба потражити кључ (или локацију где га треба сместити), водећи рачуна о величини табеле. 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 адресирање се параметризује кораком **s** тако да као резултат даје матичну адресу увећану за **i** $\cdot$ **s** као резултат према следећој формули:

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

Класа за бидирекционо линеарно адресирање се параметризује кораком **s** тако да за непарно **i** враћа матичну адресу увећану за $\mathbf{s \cdot ceil(i\ div\ 2)}$, а за парно **i** враћа матичну адресу умањену за $\mathbf{s \cdot ceil(i\ div\ 2)}$.

$$\mathbf{return\_address = a + (-1)^{i+1} \cdot s \cdot ceil(i\ div\ 2)}$$

Класа за квадратно адресирање се параметризује коефицијентом **c**, а у **i** –том покушају враћа вредност према следећој формули:

$$\mathbf{return\_address = a + c \cdot i^2}$$

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

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

[20 поена] Главни програм и функција за тестирање

Функција за тестирање умеће у задату хеш табелу задате кључеве, а затим генерише задати број кључева случајних вредности у одговарајућем опсегу и врши претрагу на њих. Након тога исписује резултате (просечан број приступа при успешној претрази, процењен 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 тестирање 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.

Задатак 2 – „Куку“ хеширање [30 поена]

Модификовати решење из претходног задатка тако да се за смештање кључева и разрешавање колизија користи техника отвореног адресирања под називом „куку“ или „кукавичје“ хеширање (https://en.wikipedia.org/wiki/Cuckoo_hashing). Код овог начина хеширања, алоцирају се две хеш табеле једнаких димензија и користе се две хеш функције $h_1(x)$ и $h_2(x)$ које се бирају на случајан начин из класе универзалних хеш функција. Сваки кључ има две матичне адресе на које се може сместити, а сваки кључ се може сместити у само једну од њих. Хеширање се заснива на два принципа:

- **вишеструком избору:** сваки елемент може да се нађе на више од једне локације у хеш табели,
- **релокацији кључева:** кључеви у табели могу да се померају након иницијалног смештања у табелу.

„Куку“ хеширање омогућава претраживање и брисање у константном времену, јер захтева проверу само две локације у комплетној хеш табели.

Приликом уметања, може се десити да су локације $h_1(x)$ и $h_2(x)$ у обе табеле заузете. Тај проблем се решава на исти начин на који птица кукавица полаже своја јаја: „птица кукавуца полаже своје јаје у гнездо неке друге птице, а њено јаје или младунца избацује из гнезда“. Аналогно томе, уколико је локација кључа и у једној и у другој табели заузета, нови кључ се смешта на место старог, а стари кључ избацује из табеле и покушава се његово уметање у другу табелу. Процедура се понавља док год се неки од кључева не смести на слободну локацију или се не упадне у бесконачну петљу. Бесконачна петља се обично апроксимира бројем итерација (покушаја) који је једнак логаритму величине табеле. У том случају, две нове хеш функције се бирају из класе универзалних хеш функција и врши се поновно хеширање свих кључева, док год се не изврши смештање свих кључева или не искористе све хеш функције. За потребе задатка треба осмислити класу универзалних хеш функција за целобројне кључеве које ће бити коришћене у имплементацији решења. Допунити главни програм тако да омогући тестирање извршених модификација.

Уз поставку задатка је доступан додатан документ о „куку“ хеширању од аутора саме идеје. Више информација о универзалним хеш функцијама се може наћи у књизи професора Томашевића.