ОДСЕК ЗА СОФТВЕРСКО ИНЖЕЊЕРСТВО

АЛГОРИТМИ И СТРУКТУРЕ ПОДАТАКА 2

2022-2023

- трећи домаћи задатак -

Опште напомене:

1. Домаћи задатак 3 састоји се од једног програмског проблема. Студенти проблеме 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. Формула за редни број **i** комбинације проблема коју треба користити приликом решавања задатка је следећа: (**R** – редни број индекса, **G** – последње две цифре године уписа):

$$\mathbf{i = (R + G) \bmod 2}$$

10. Одбрана домаћег задатка ће се обавити према распореду који ће накнадно бити објављен на сајту предмета.
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.

19.12.2022. године *Са предмета*

Задатак – Хипови [100 поена]

Од ефикасности имплементације приоритетног реда умногоме може зависити ефикасност појединих алгоритама који их користе, као што су на пример Хафманов, Дијкстрин или Крускалов алгоритам. Један од начина за ефикасну имплементацију приоритеног реда је коришћење хип стабала (енг. *heaps*).

У оквиру овог задатка, потребно је имплементирати структуру генерализованог бинарног хипа, односно *m*-арни хип, имплементираног у вектору, која ће се користити за имплементацију приоритетног реда. Генерализовани *m*-арни хип је комплетно или скоро комплетно *m*-арно стабло за које важи правило уређености хипа. Тако је бинарни хип заправо 2-хип, тернарни је 3-хип и слично. За ову примену потребно је имплементирати *min-m-heap*. Написати на језику C++ потребне класе за реализацију хипа који садржи целобројне кључеве. Операције које треба реализовати су:

- **[5 поена]** `void init(int m)` - стварање новог (празног) хипа реда *m*
- **[10 поена]** `void add(int elem, int &steps)` - додавање елемента у хип
- **[5 поена]** `int get()` - дохватање минималног елемента хипа
- **[10 поена]** `int delete(int &steps)` - брисање минималног елемента
- **[15 поена]** `convertTo(int newM)` - претварање хипа у хип задатог новог реда
- **[15 поена]** `union(Heap h, int &steps)` - додавање једног хипа другом (спајање хипова) – резултат је измењен хип за који се операција позива
- **[5 поена]** `operator<<` - испис хипа у облику *m*-арног стабла
- **[5 поена]** `void destroy()` - брисање (уништавање) хипа

У зависности од редног броја проблема **i** који се решава, реализовати и следећу операцију:

0. **[10 поена]** Промена вредности произвољног кључа
1. **[10 поена]** Брисање произвољног кључа

Свака операција спајања, уметања и брисања треба да врати број корака који је био потребан да би хип добио свој коначни изглед. У кораке се броје поређења и замене места елемената (чворова) у хипу.

Исправна реализација хипа 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. Програм такође треба да води рачуна о исправном коришћењу меморије.

Реализовати програм са једноставним интерактивним менијем који кориснику омогућава рад са јавним методама хипа. Главни програм треба да омогући учитавање елемената из командне линије или датотеке, као и стварање случајних вредности за унос.

Главни програм и тестирање [20 поена]

Функција за тестирање користи реализован хип за имплементацију приоритетног реда. Приоритетни ред подржава функције:

- Стварања реда
- Испитивања да ли је ред празан
- Дохватање првог елемента реда
- Брисање (са или без дохватања) првог елемента реда
- Уметање елемента у ред

Реализовати главни програм са једноставним интерактивним менијем који кориснику омогућава рад са јавним методама. Главни програм треба да омогући учитавање елемената из командне линије или датотеке, као и стварање случајних вредности за унос.

Главни програм треба да омогући и мерење перформанси. Перформансе треба дати у следећем табеларном облику:

Величина скупа података и ред хипа	Време уметања	Број корака приликом уметања	Време избацивања	Број корака приликом избацивања
100, 2				
100, 4				
100, 8				
1000, 2				
1000, 4				
1000, 8				
10 000, 2				
10 000, 4				
10 000, 8				
100 000, 2				
100 000, 4				
100 000, 8				

Величина скупа података је број елемената који треба генерисати или учитати из датотеке. Време уметања представља време потребно за убацивање једног по једног елемента из датог скупа у иницијално празан хип. Број корака приликом уметања је сума броја корака за сваки убачени елемент. Време избацивања представља време потребно да се из тог формираног хипа избацује (минимални) елемент по (минимални) елемент. Број корака приликом избацивања је сума броја корака за сваки избачени елемент.

Напомене

По потреби реализовати и додатне методе, где је то примерено.

За тестирање програма се могу користити датотеке са табелама различитих величина, које се налази у оквиру посебне архиве.

Прилог 1 - Рад са датотекама у језику C++

Рад са датотекама у језику C++ захтева увожење заглавља **`fstream`** (именски простор **`std`**). За читање података користи се класа **`ifstream`**. Након отварања датотеке, читање се врши на исти начин као и са стандардног улаза. Кратак преглед најбитнијих метода и пријатељских функција ове класе је дат у наставку.

`void open(`
`  const char *_Filename,`
`  ios_base::openmode _Mode = ios_base::in,`
`  int _Prot = (int)ios_base::_Openprot`
`);`
	Отвара датотеку задатог имена за читање.
	`ifstream dat;`
	`dat.open("datoteka.txt");`

`void close();`	Затвара датотеку.
`bool is_open();`	Утврђује да ли је датотека отворена.
`operator>>`	Преклопљен оператор за просте типове података.

`ifstream dat;`
`dat.open("datoteka.dat");`
`if( ! dat.is_open() )  greska();`
`char niz[20];`
`dat >> niz;`
`dat.close();`
	Пример отварања датотеке, провере да ли је отварање успешно, читање једног знаковног низа из датотеке и затварања датотеке.

Прилог 2 – Мерење перформанси

Библиотека **`std::chrono`** која постоји од верзије C++ 11 садржи неколико функција разних степена прецизности које се могу користити за дохватање системског времена и мерење перформанси, тј. времена извршавања програма. Најпрецизније време се мери помоћу класе **`high_resolution_clock`**, чија метода **`now()`** дохвата тренутно време. Позивом те методе непосредно пре и непосредно након дела кода коме се жели измерити време извршавања, рачунањем разлике та два времена као **`duration`** са одговарајућом прецизношћу (нпр. милисекунде или наносекунде; подразумевано су секунде) добија се време извршавања у траженој јединици времена. Један пример позива дат је у наставку, а више информација и примера позива можете наћи и на линку https://en.cppreference.com/w/cpp/chrono.

Пример примене:

```
#include <chrono>
#include <thread>

using namespace std::chrono;

void main(){
	auto start = high_resolution_clock::now();
	std::this_thread::sleep_for(seconds(1));
	auto end = high_resolution_clock::now();
	duration<double, std::milli> time_ellapsed_ms = end - start;
}
```