ОДСЕК ЗА РАЧУНАРСКУ ТЕХНИКУ И ИНФОРМАТИКУ

АЛГОРИТМИ И СТРУКТУРЕ ПОДАТАКА

2017-2018

- домаћи задатак -

Опште напомене:

1. Домаћи задатак састоји се од једног програмског проблема. Студенти проблем решавају **самостално**, на програмском језику C++. **Није дозвољено коришћење готових структура података из STL и сличних библиотека.**
2. Право да раде домаћи задатак имају сви студенти који прате предмет. Студенти треба да се пријаве за одбрану домаћег задатка најкасније до **петка, 22.12.2017.** године путем одговарајућег линка на сајту предмета.
3. Домаћи задатак замењује други део испита, односно задатак на испиту који се пише у конкретном језику. Поени са домаћег задатка ће стога бити скалирани у одговарајућем односу. Одбраном домаћег задатка студент се одриче права да ради други део испита. Студент на испиту треба да назначи да ли је радио домаћи задатак како би му били признати поени.
4. Реализовани програм треба да комуницира са корисником путем једноставног менија који приказује реализоване операције и омогућава сукцесивну примену операција у произвољном редоследу.
5. Унос података треба омогућити било путем читања са стандардног улаза, било путем читања из датотеке.
6. Решење треба да буде отпорно на грешке и треба да кориснику пружи јасно обавештење у случају детекције грешке.
7. Приликом оцењивања, биће узето у обзир рационално коришћење ресурса. **Примена рекурзије се неће признати као успешно решење проблема.**
8. 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.
9. Одбрана домаћег задатка ће бити одржана у **четвртак, 28.12.2017.** према распореду који ће благовремено бити објављен на сајту предмета.
10. Формула за редни број операција над скупом **i** и репрезентације уланчане листе **j** коју треба користити приликом решавања задатка је следећа:

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

    $$\mathbf{i = (R + G) \bmod 3}$$

    $$\mathbf{j = (R + G) \bmod 4}$$
11. Име датотеке која се предаје као решење мора бити **dz.cpp**
12. Предметни наставници задржавају право да изврше проверу сличности предатих домаћих задатака и коригују освојени број поена након одбране домаћих задатака.

13.12.2017. године *Са предмета*

Задатак – Рад са скуповима [100 поена]

Скупови уобичајено нису подржани као уграђени типови у већини програмских језика. Стога се они често имплементирају кроз библиотечку подршку. Циљ овог задатка је имплементација и тестирање библиотеке за рад са целобројним скуповима. Задатак решити писањем одговарајућег **система класа**. Приликом решавања задатка, водити рачуна о добрим праксама објектно-оријентисаног програмирања.

Поставка проблема

Потребно је имплементирати основне операције за рад са једним или више целобројних скупова и измерити њихове перформансе за различите репрезентације скупа. Потребно је омогућити следеће операције за рад скуповима:

a) Стварање празног скупа и уништавање скупа
b) Претрагу на задати елемент (утврђивање припадности скупу)
c) Уметање елемента у скуп
d) Брисање елемента из скупа
e) Брисање елемената скупа у задатом опсегу
f) Дохватање укупног броја елемената у скупу

У зависности од редног броја проблема који се решава, имплементирати и следеће операције за рад са скупом:

0. Унију два скупа
1. Пресек два скупа
2. Разлику два скупа

Имплементација скупа коришћењем уланчане листе [30 поена]

Скуп целих бројева се може имплементирати помоћу уланчане листе. У зависности од редног броја проблема који се решава, користити следећу имплементацију уланчане листе:

0. Једноструко уланчану листу
1. Једноструко уланчану кружну листу
2. Двоструко уланчану листу
3. Двоструко уланчану кружну листу

По потреби, уланчаној листи се може додати заглавље и могу се имплементирати додатне функције за рад са листом.

Имплементација скупа коришћењем стабла бинарног претраживања [40 поена]

Скуп целих бројева се може имплементирати помоћу стабла бинарног претраживања. Чвор стабла треба да садржи **само** целобројни кључ и показиваче на лево и десно подстабло. По потреби, стаблу бинарног претраживања се могу додати заглавље и могу се имплементирати додатне функције за рад са стаблом.

Мерење перформанси извршавања [20 поена]

Потребно је измерити перформансе примене реализованих операција за рад са скуповима имплементираних путем уланчане листе и стабла бинарног претраживања. Тестирање спровести над скуповима за различите величине (видети табелу испод) на следећи начин. За дату величину скупа, формирати скуп уметањем случајних бројева, а затим над њим извршити операције. Поступак извршити 3 пута и просечно време извршења операција усвојити као меру перформансе. Посебно треба мерити време формирања скупа од више елемената, уметања појединачног елемента, брисања појединачног елемента и претраге појединачног елемента, као и време потребно да се изврши реализована скуповна операција. За тестирање задате скуповне операције треба формирати два скупа уметањем случајних бројева. Резултате табеларно исписати на стандардном излазу, према формату који по структури одговара Табели 1. За мерење протеклог времена користити класу *PerformanceCalculator* (видети прилог). **Напомена**: у циљу веродостојног поређења различитих имплементација, скупове, за дату величину скупа и дату операцију, формирати употребом истих случајно генерисаних кључева.

Табела 1: просечно време извршења операција над скуповима. Време је изражено у милисекундама

Величина скупа	200	500	1000	10000	100000	1000000
формирање						
уметање						
брисање						
претрага						
скуповна операција						

Главни програм [10 поена]

Демонстрацију рада програма остварити кроз једноставан интерактивни мени који ће омогућити комуникацију са корисником и приказати рад са скуповима имплементираним на претходно описани начин. Програм треба да омогући читање скупа целобројних кључева који се умећу у стабло са стандардног улаза или из датотеке и да има могућности исписивање садржаја скупа.

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 - Изворни код класе PerformanceCalculator

Класа је писана за оперативни систем Windows, под развојним окружењем Visual Studio. Употреба класе се своди на 3 методе: `start()` да би се отпочело мерење, `stop()` да би се мерење прекинуло и `elapsedMillis()` да би се дохватио број милисекунди који је протекао између позива метода `start()` и `stop()`.

Пример примене:

```
#include "PerformanceCalculator.h"
void main(){
PerformanceCalculator pc;

	pc.start();
	Sleep(100);
	pc.stop();
	double protekloVreme = pc.elapsedMillis();
}
```

Изворни код класе *PerformanceCalculator* (датотека: PerformanceCalculator.h)

```
#ifndef PERFORMANCE_CALCULATOR_H__
#define PERFORMANCE_CALCULATOR_H__

#include "windows.h"

class PerformanceCalculator {
	LARGE_INTEGER	startCounter;
	LARGE_INTEGER	stopCounter;
	LARGE_INTEGER	frequency;
public:
	PerformanceCalculator() {
		startCounter.QuadPart = stopCounter.QuadPart = 0;
		QueryPerformanceFrequency(&frequency);
	}

	void start() {
		QueryPerformanceCounter(&startCounter);
	}

	void stop()  {
		QueryPerformanceCounter(&stopCounter);
	}

	double elapsedMillis() const {
		return (stopCounter.QuadPart-startCounter.QuadPart)*1000./frequency.QuadPart;
	}
};

#endif
```