КУРСОВИ ПРОЕКТИ ПО СТРУКТУРИ ОТ ДАННИ

Факултет по математика и информатика на СУ „Св. Климент Охридски", Специалности „Информатика“ и „Софтуерно инженерство“, 2 курс

Зимен семестър 2016/2017

Обща информация за проектите

Проектите се оценяват по редица от критерии, всеки от които носи точки. Общата сума от точките надвишава необходимия брой за оценка отличен. Тъй като курсът се фокусира върху алгоритмите и структурите данните и тяхната реализация на езика C++, **най-важното изискване за проектите е те да включват подходящи реализации на различни алгоритми и структури от данни**. Проект без такива реализации се оценява с нула точки. Други важни критерии за оценка на проектите са:

- **Дали решението работи коректно**, съгласно спецификацията. Решение, което не работи и/или не се компилира носи минимален брой (или нула) точки.
- **Дали решението отговаря на заданието на проекта**.
- **Дали решението използва подходящи алгоритми и структури от данни**. Това е най-важното изискване към проектите. Тъй като курсът се фокусира върху алгоритмите и структурите от данни и тяхната реализация, проекти в които няма реализация на подходящи алгоритми и структури от данни се оценяват с нула точки. Включването им в проектите трябва да бъде обосновано. По време на защитата се очаква да можете да отговорите на различни въпроси, като например: (1) кои алгоритми сте реализирали, (2) защо сте избрали именно тях, (3) как сте ги реализирали, (4) каква е тяхната сложност и др. Ако за някои части на проекта не можете да реализирате подходящ алгоритъм, можете да използвате и brute force решения но те носят минимален брой (или нула) точки.
- **Дали решението е било добре тествано**. Проверява се какви тестове са били проведени за приложението (напр. unit test-ове). Очаква се по време на защитата да можете да посочите как сте тествали приложението, за да проверите дали то работи коректно и как се държи в различни ситуации. Не е задължително да използвате формален подход, като например TDD или да включите unit test-ове, но трябва да помислите за това как да проверите дали приложението ви прави точно това, което искате.
- **Оформление на решението**. Проверява се дали кодът е добре оформен, дали е спазена конвенция за именуване на променливите, дали е добре коментиран и т.н.

- **Дали проектът е документиран**. Проверява се дали кодът е добре коментиран и дали въз основа на коментарите автоматично се генерира техническа документация (напр. с Doxygen). Този критерий не е задължителен и носи бонус точки.

Оценката на всеки от проектите се формира от онази негова част, която е била самостоятелно разработена от вас. Допустимо е да използвате код написан от някой друг (напр. готова библиотека или помощ от ваш приятел/колега), но (1) той не носи точки към проекта и (2) това трябва да бъде ясно обявено при предаването и защитата на проекта, като посочите коя част от проекта сте разработили самостоятелно. Това означава, че:

1. Използваните наготово код трябва да се маркира ясно, като поставите коментари на подходящи места в кода.
2. По време на защитата трябва да посочите кои части сте разработили самостоятелно и кои са взети от други източници.

Както е написано по-горе, когато в проекта си използвате чужд код, сам по себе си той не ви носи точки. Допълнителни точки могат да се дадат или отнемат, според (1) способността ви за внедряване на кода във вашето решение (напр. в случаите, когато се използва външна библиотека) и за това (2) дали добре разбирате какво прави той.

1: Търсене на текст с регулярен израз

В рамките на този проект трябва да разработите приложение, което позволява на потребителя да търси в текстови файлове с помощта на регулярни изрази.

Приложението ви трябва да получава от командния ред следните аргументи:

1. Име на файл или директория, който/която трябва да се обработи
2. Регулярен израз.

Ако на приложението ви бъде подадено име на файл, то трябва да изведе на екрана всички редове на файла, които отговарят на регулярния израз. Извеждането трябва да бъде в следния формат:

```
<име на файла>:<номер на ред>:<текст>
```

Например:

```
> RegExSearch.exe MyFile.txt "(a*)"
MyFile.txt:5:aaaaaa
MyFile.txt:15:aaa
MyFile.txt:16:aaaaaaaaaa
```

При обработката на редовете приложението ви НЕ ТРЯБВА да прави разлика между големи и малки букви (case-insensitive).

Сами решете какво да бъде поведението на приложението при възникване на грешка. Например възможно е да се укаже път към файл, който не съществува или за който нямате права за четене, може да има синтактична грешка в регулярния израз и т.н.

За решаването на задачата, програмата ви трябва да реализира подходящ алгоритъм, който по даден регулярен израз E, конструира недетерминиран краен автомат A с език $\mathcal{L}(E) = \mathcal{L}(A)$. След това автоматът трябва да се използва за анализиране на всеки от редовете и тези от тях, чиято обработка го доведе до крайно състояние, да бъдат изведени на екрана.

Сами можете да изберете какво представяне да изберете за автомата и как да го построите по първоначално зададения регулярен израз. Проектът ви ще се оценява според това каква времева и пространствена сложност има решението ви.

Регулярният израз, който ще бъде подаден на решението ви, може да съдържа нула, един или повече символи. За определеност, ако бъде подаден празен израз (т.е. потребителят подава празен низ в качеството на регулярен израз), считаме, че той съвпада с всеки ред на файла. Валидните символи са тези с кодове от 33 до 126 от ASCII таблицата (т.нар. символи, които имат графично представяне). Ако на входа бъде подаден израз, който съдържа други символи, освен валидните, програмата трябва да изведе съобщение за грешка.

Дефиницията за непразен регулярен израз е както следва:

```
<базов символ> е всеки валиден символ, с изключение на следните:
кръглите скоби, вертикална черта (|), точка (.), звезда (*) и обратна
наклонена черта (\).

<специален символ> ::= \s | \d | \a | \e | \\

<символ> ::= <базов символ> | <специален символ>

<редица символи> ::= <символ> | <символ><редица символи>

<израз> ::= <редица символи>  |
            (<израз>.<израз>) |
            (<израз>|<израз>) |
            (<израз>*)
```

Семантиката на специалните символи и изразите е както следва:

- \s съвпада с произволен празен символ (whitespace). Такива са интервал, табулация, нов ред и т.н.;
- \d съвпада с произволна цифра;
- \a съвпада с произволна буква от латинската азбука – малка или голяма;
- \e се интерпретира като празната дума;
- \\ се интерпретира като една наклонена черта;
- Символът точка (.) е конкатенация. В такъв израз участват два други – E_1 и E_2. Той разпознава текст, чиято първа част отговаря на E_1, а втората му – на E_2.
- Символът вертикална черта (|) е обединение. В такъв израз участват два други – E_1 и E_2. Той разпознава текст, който отговаря на E_1 или на E_2.
- Символът звезда (*) обозначава итерация. В такъв израз участва един друг – E. Той разпознава или празния низ (нула срещания на E) или едно или повече срещания на E, т.е. ($E.E$), (($E.E$).E) ((($E.E$).E).E) и т.н.;

Примери

Израз	**Съвпада със**	**Не съвпада със**
abc	"abc" "Abc"	"abcd" "aaabc"
\s	" " //един интервал	"" // празния низ " " // няколко интервала
\d	"1"	"a" "123"
\e	"" //празния низ	" " //един интервал
\\	"\"	"\\"
Ab\d	"ab1" "aB5"	"ab12" "ab 1"
a\sb	"a b"	"ab" "a b"
(ab|cd)	"ab" "cd"	"abcd"
((ab|cd).(wx|yz))	"abwx" "abyz" "cdwx" "cdyz"	"abcdewxyz" "wxyz" "abcd"
(a*)	"" //празния низ "A" "AaAAa" "aaaaaaaaa"	" a "
((a*).b)	"b" "ab" "aaaaaaaab"	"" //празния низ
(((a*).b).(\\\|/))	"b\" "b/" "ab\" "ab/" "aaaaaaaab\" "aaaaaaaab/"	"ab\\" "ab/\" "ab\/"
(((\s*).(\d*)).(\s*))	"123" " 123 "	

Ако на приложението ви бъде подадено име на директория, вместо име на файл, то трябва да извърши търсене във всички файлове, които се съдържат в нея. Търсенето е същото, което беше описано по-горе, но се прилага върху множество файлове, вместо само върху един отделен.

Бонус точки

Към решението ви ще бъдат дадени бонус точки, ако реализирате операциите обединение (|) и конкатенация (.) като инфиксни, а итерация – като постфиксна. В този случай при използването им няма нужда да се поставят скоби. Вместо това скобите могат да се използват за промяна на

приоритета на операциите. По подразбиране приоритетът е както следва: с най-висок е итерацията (*), след нея следва конкатенацията (.) и накрая е обединението. По подразбиране се счита, че те се прилагат върху символите, които стоят непосредствено до тях, освен ако не са сложени скоби. Например:

```
Abc* разпознава "ab", "abc", "abcc", "abccc" и т.н.
AB*.cd* разпознава "ac", "abc", "acd", "abcd", "abbcddddd" и т.н.
(ab)*.(cd)* разпознава "", "ab", "cd", "abcd", "ababcdcdcdcd" и т.н.
ab*.cd*|ef* разпознава "ac", "abc", "acd", "abcd", "e", "ef", "efff" и
т.н. но НЕ разпознава например "abcdef"
```

Втора възможност за получаване на бонус точки е да реализирате приложението си така, че то да може да работи с голяма входна азбука. В този случай програмата ви трябва да бъде написана така, че вътрешно да работи с Unicode, вместо ANSI символни низове. Регулярният израз, който се подава като вход ще може да съдържа произволни символи. В този случай кръглите скоби, вертикална черта (|), точка (.), звезда (*) и обратна наклонена черта (\) отново имат специално значение и се използват така, както е описано по-горе. Ако приемем, че азбуката е с размер N, то от всеки възел в автомата, потенциално биха могли да излязат N дъги. Това означава, че с увеличаването на N и при голям брой възли, трябва да се намери подходящ начин за описване на функцията на преходите, така че да се събира в наличната памет. Един възможен подход е да се имплементира алгоритъма на Таржан (http://i.stanford.edu/pub/cstr/reports/cs/tr/78/683/CS-TR-78-683.pdf). Можете да реализирате и друг подобен алгоритъм за работа с разредени матрици/таблици (sparse matrix/array).

2: Архивиране на файлове

В рамките на този проект трябва да се разработи приложение, което сканира дадена директория и архивира нейното съдържание. Програмата трябва да създаде един архивен файл, който пази в себе си оригиналното съдържание на директорията. Съдържанието на файла трябва да се компресира, като за кодирането трябва да се използва алгоритъмът на Хъфман, а моделирането можете да изберете сами как да реализирате.

Относно разликата между моделиране и кодиране: Нека имаме следния стринг:

```
"abcd abab cdcd cdab"
```

Най-грубо казано, моделирането се опитва да анализира входа и да определи кои са отделните единици, които след това трябва да се представят по оптимален начин (кодиране). Например една възможност е моделирането да заключи, че в текста има четири думи, които подлежат на кодиране:

```
"abcd ", "abab ", "cdcd ", "cdab"
```

Ако моделирането им съпостави следните числа:

```
1:"abcd ", 2:"abab ", 3:"cdcd ", 4:"cdab"
```

Тогава кодираният текст ще изглежда така:

```
1234.
```

Възможно е обаче моделирането да реши, че имаме следните четири срички:

```
"ab", "ab ", "cd", "cd "
```

Нека отново предположим, че кодирането ще ги номерира от едно до четири:

```
1:"ab", 2:"ab ", 3:"cd", 4:"cd "
```

Тогава кодираният текст ще изглежда така:

```
14123431
```

Когато компресирате стринга, не можете да предадете просто кодираното представяне, защото зад него може да стоят безброй неща. Трябва да предадете и използвания речник. Така за последния пример, ако искаме да предадем на някого кодирания текст, можем да предадем например:

```
1:"ab", 2:"ab ", 3:"cd", 4:"cd ", 14123431
```

Както се вижда от примерите, при една и съща схема на кодиране, може да се получат различни резултати, в зависимост от резултатите от моделирането. Както вече написахме, в проектите трябва да се използва Хъфманово кодиране, а сами можете да изберете как да моделирате входа. Най-лесно, но не особено ефективно ще бъде да четете входа байт по байт и да кодирате отделните байтове.

При оценяването на проектите ви ще се взима предвид и как точно моделирате входа си.

Като функционалност приложението ви трябва да може да архивира или разархивира съдържанието на дадена директория. При стартирането му потребителят трябва да може или да укаже директория, която да се архивира или да укаже файл, който трябва да се разархивира.

Входът на програмата трябва да се получава от командния ред. Първият параметър указва каква операция да се изпълни. Възможните операции са както следва:

№	Команда	Описание
1.	`-Pack`	Архивиране. В този случай след командата се подава път до директория, която да се обработи и името на архивния файл, който трябва да се създаде. Например: `> Archive.exe -Pack C:\Test C:\Temp\myfile.arc`
2.	`-Unpack`	Разархивиране. В този случай след командата се подава път до съществуващ архив и път до директория, в която той да се разархивира. Например: `> Archive.exe -Unpack C:\Temp\myfile.arc C:\AnotherDirectory\`
3.	`-List`	Показване на съдържанието на архив. След тази команда се подава път до съществуващ архив. Програмата ви трябва да отвори архива и да изведе на екрана имената на всички файлове и директории, които се съдържат в него. Например: `> Archive.exe -List C:\Temp\myfile.arc`

Докато работи, приложението трябва да извежда някаква информация, за да може потребителят да знае дали то работи или е „увиснало“. Например може да се извежда броят на обработените файлове, колко процента от работата е свършена и т.н. Един сравнително прост вариант и за трите команди, е да извеждате името на всеки обработен файл.

Обърнете внимание, че във входната директория може да има не само файлове, но и други директории. Тъй като се иска да архивирате цялата директория, това означава, че трябва да можете рекурсивно да влезете в тези поддиректории и да запазите и тяхното съдържание. Също така, възможно е някои от поддиректориите да са празни (да не съдържат файлове) или да имате файлове с размер нула байта (празни файлове). Приложението ви не трябва да ги пропуска и след разархивиране, те трябва да се възстановят така, както са били в оригиналната директория.

Бонус точки

За допълнителни точки, приложението може да се разшири, като в него се реализира някакъв вид шумозащитно кодиране. Това означава, че освен съдържанието на файловете, в архива се пази и допълнителна информация (например CRC), която позволява (1) да се провери дали информацията в архива не е повредена, а при някои схеми на кодиране дори и (2) да се коригират възникналите грешки и да се възстанови оригиналното съдържание. Специално CRC не дава втората възможност, но има други схеми на кодиране, като например BCH (Bose-Chaudhuri-Hocquenghem), които го позволяват. Респективно, при разархивиране, тази информация трябва да се използва, за да се провери дали всички файлове са били възстановени успешно. В случай на грешка трябва да се изведе съобщение и ако е възможно, тя да се коригира.

Втори вариант за разширение за бонус точки е да се реализира търсене в архив. За целта трябва да се реализира следната команда:

4.	`-Search`	Търсене на текст във файловете в даден архив. След тази команда се подава път до съществуващ архив и символен низ (стринг) T. Програмата ви трябва да претърси файловете, които се съдържат в архива и да изведе на екрана имената на онези от тях, които съдържат в себе си текста T. Търсенето трябва да се реализира директно върху кодираните данни. Например: `> Archive.exe -Search C:\Temp\myfile.arc "Hello world"`

Търсенето на текст трябва да реализирате сами и то трябва да се изпълнява върху кодираните данни. Например решение, при което декодирате всеки файл във временна директория или в паметта и след това изпълнявате върху всеки негов ред функцията `strstr()`, не носи точки.