Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Datové struktury v jazyce R

Bakalářská práce

Autor: Jan, Knejp
Studijní obor: Aplikovaná informatika

Vedoucí práce: prof. RNDr. Hana Skalská, CSc.

Hradec Králové duben 2015

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 29.4.2015 Jan Knejp

Poděkování:

Děkuji vedoucí bakalářské práce prof. RNDr. Haně Skalské, CSc. za metodické vedení práce.

Anotace

Cílem této bakalářské práce je popsat základní charakteristiky datových struktur, jejich vytváření, výběr dat a některé funkce, manipulující s datovými strukturami. Dále popisuje způsoby exportu a importu dat z textových a binárních souborů a z relačních databází. Nakonec práce obsahuje aplikování některých zmíněných funkcí na konkrétní příklad.

Annotation

Title: Data structures in R Language

The goal of this bachelor thesis is to describe basic characteristics of data structures, their creation, selection of their data and some of the functions that manipulate with them. It also describes the options of export and import of data from text and binary files and from relational databases. Finally, the thesis contains an application of some functions mentioned on specific example.

Obsah

1 Úvod

Cílem této práce je popsat fungování základních datových struktur v jazyce R, konkrétně vektoru, matice, pole, seznamu, datové tabulky a faktoru. Cílem je popsat obecné charakteristiky dané datové struktury, způsoby vytváření datových struktur, přidávání nových dat a také seznámit s některými funkcemi pracujícími s danými datovými strukturami. Dalším cílem této práce je prezentovat způsoby importu a exportu dat z R. Účelem je popsat funkce pro export a import do textových souborů různých formátů a také binárních souborů typu RData, a souborů typu xlsx. Dalším cílem této práce je představit dva základní způsoby práce s relačními databázemi v R. Tato práce obsahuje ukázky kódu. Tyto ukázky jsou odlišeny jiným písmem a výstupy jsou navíc označeny tučně. V závěru práce se nachází abecední seznam použitých funkcí. U každé funkce je uvedena strana, na které je poprvé zmíněna.

Při vypracování této práce byl využit program R ve verzi 3.1.1, který je možné stáhnout z webových stránek http://www.r-project.org/. V práci se jako příklady vyskytují výstupy z tohoto programu.

2 Principy jazyka R

R je programovací jazyk a prostředí vhodné zejména pro statistické výpočty. Obsahuje spoustu nástrojů pro statistickou analýzu a pro další funkcionalitu je možné doinstalovat rozšiřující balíčky. Umožňuje efektivně ukládat data, dále s nimi manipulovat, provádět na nich výpočty a také nabízí možnosti grafické reprezentace dat. [1]

Přestože je R využíván především ke statistickým výpočtům, není tím omezen. R je plně funkční programovací jazyk, implementující jazyk S. Další implementací jazyka S je S-PLUS. Oba jazyky jsou si velice podobné, ale přesto existují drobné rozdíly. [1]

Při prvním otevření programu se zobrazí konzole. Konzole čeká na vstup od uživatele, které začínají symbolem >. Příkazy, které mají nějakou návratovou hodnotu, tuto hodnotu vypíší do konzole. Hodnoty lze ukládat do proměnné pomocí symbolu = nebo <-.

```
> x <- 5
> x
[1] 5
```

Lze také využívat funkce, které se vyvolávají jménem funkce a parametry v kulatých závorkách, oddělenými čárkou. Velmi užitečnou je funkce help(), která otevře dokumentaci k funkci, jejíž jméno bylo použito jako parametr. Dokumentaci k samotné funkci help je tedy možné otevřít tímto příkazem:

```
> help(help)
```

Parametrem nemusí být nutně pouze funkce, například dokumentaci k ukládání hodnot do proměnných lze otevřít tímto příkazem:

```
> help(assignOps)
```

R má spoustu funkcí k dispozici ihned po startu. Další lze získat načtením balíčku, což je seskupení funkcí a dat. Základní balíček se jmenuje base a je k dispozici vždy. Další balíčky lze načíst pomocí funkce library(), jejímž parametrem je jméno balíčku. Pokud je funkce zavolána bez parametrů, otevře se okno s balíčky, které je možné načíst. Instalovat balíčky z internetu lze pomocí funkce install.packages(),

parametrem této funkce je text s názvem balíčku. Po zadání příkazu se otevře okno s výběrem zemí, ze kterých lze balíček stáhnout. [1]

Na internetové adrese http://cran.r-project.org/web/packages/ lze najít tisíce balíčků, které lze nainstalovat do R.

3 Popis základních datových struktur

3.1 Vektor

Nejjednodušší datovou strukturou je vektor. Obsahuje libovolně dlouhou řadu hodnot, které musejí být všechny stejného typu. Veškeré výpočty se provádí na vektorech, i jednotlivé hodnoty jsou reprezentovány vektorem s délkou jedna. [1]
Každý objekt má automaticky 2 základní atributy: **mode** a **length**. Tyto atributy lze zobrazit funkcemi mode() a length(). [1]
Atribut **mode** určuje mód daného objektu. Pro vektory je to typ hodnot, které obsahuje. Vektor může nabývat těchto módů: [1]

- numeric
- complex
- logical
- character
- raw

Numeric označuje číselné hodnoty. Vnitřně je ještě rozdělen na integer, tedy celá čísla, a double, tedy desetinná čísla. Mód complex je pro komplexní čísla, tedy čísla s reálnou a imaginární částí a character pro písmena a znaky. Logical může nabývat dvou stavů, logická pravda nebo nepravda, reprezentované hodnotami TRUE a FALSE. Poslední mód raw reprezentuje data v základní bytové formě. [1]
Textové hodnoty (také se jim říká řetězce) lze v R vytvářet uzavřením textu do uvozovek. Tyto uvozovky mohou být dvojité, nebo jednoduché. Protože uvozovky označují začátek i konec řetězce, nelze je použít uvnitř řetězce, lze však použít jednoduché uvozovky uvnitř dvojitých nebo naopak. Alternativou je napsat před uvozovku zpětné lomítko, které slouží k reprezentaci některých speciálních znaků. Pro napsání zpětného lomítka je nutné napsat dvě lomítka za sebou.[1]
Atribut **length** určuje velikost objektu, u vektoru je to počet hodnot. Přestože tento atribut mají automaticky všechny objekty, u některých nemá žádný význam. Tento atribut lze využít pro zkrácení nebo prodloužení vektoru. Při prodlužování vektoru se nová místa zaplní hodnotou NA. [1]

```
> x
[1] 19 28 37 64 55
```

```
> length(x) <- 3
> x
[1] 19 28 37
> length(x) <- 6
> x
[1] 19 28 37 NA NA NA
```

Vektor může mít i nepovinný atribut **name**. Ten reprezentuje jména jednotlivých hodnot. Vektor se však v tomto ohledu chová specificky, neboť jména jsou ve skutečnosti vnitřně uložena jako první hodnota atributu **dimnames**. Atribut se z objektu získává funkcí names() a pro uložení jmen je třeba do atributu vložit vektor řetězců. Při vypisování do konzole slouží jako popisky hodnot. [2]

```
> x <- c(1,2,3,4)
> names(x) <- c("jednička","dvojka","trojka","čtyřka")
> x
jednička   dvojka   trojka   čtyřka
       1        2        3        4
```

V R existuje speciální hodnota NA. Značí, že hodnota není známa, a pokud je použita v nějaké operaci nebo funkci, výsledek je většinou také NA. Zkontrolovat, zda data obsahují hodnotu NA lze pomocí funkce is.na(). [1]

Nad vektory lze provádět základní elementární operace jako je +, -, *, /. Další operace jsou například znak ^ pro umocňování, nebo například trigonometrické funkce sin(), cos() a tan(). Funkce se provede nad každou hodnotou vektoru a výsledkem je zase vektor. V případě operace nad 2 vektory se provede mezi hodnotami obou vektorů na stejném pořadí. [1]

```
> x
[1] 2 5 3
> y
[1]  2 -1  1
> x+y
[1] 4 4 4
```

Vektory však pro aritmetické operace a funkce nemusí být stejně dlouhé. Pokud se ve funkci objeví vektory rozdílné délky, nastává takzvaná *recyklace*. Kratší vektory se zvětší na stejnou délku jako nejdelší vektor a chybějící hodnoty se postupně nahrazují hodnotami ze začátku vektoru. Pokud jeden cyklus nestačí, vektory se recyklují, dokud nedosáhnou dostatečné délky. [1]

```
> x
[1]  1 10
> y
[1] 1 2 3 4 5 6 7 8
> x*y
```

```
[1]  1 20  3 40  5 60  7 80
```

Pro vytvoření libovolného vektoru se používá funkce c(). Parametry jsou hodnoty nebo vektory a výsledkem je spojený vektor.

```
> x <- c(4,5,7)
> x
[1] 4 5 7
> c(3,x,9)
[1] 3 4 5 7 9
```

Vektor lze vytvořit i s názvy hodnot napsáním parametrů ve formátu jméno=hodnota oddělené čárkou.

```
> c(cervena = 255,modra = 23, zelena = 101)
cervena   modra  zelena
    255      23     101
```

Vektor lze také vytvořit sekvencí. Pro jednoduché sekvence má R speciální syntaxi, dvojtečku mezi 2 čísly. První číslo označuje začátek sekvence, a druhé její konec. Krok sekvence je jedna. Sekvence může být i klesající, pokud je druhé číslo menší, a také může být v desetinných číslech. Pokud by konec sekvence nebyl přesně dosažitelný, konečné číslo bude poslední, které ještě nepřekročilo cílové číslo.[1]

```
> 5:8
[1] 5 6 7 8
> 10.2:5.3
[1] 10.2  9.2  8.2  7.2  6.2
```

Tímto způsobem lze vytvořit pouze sekvence s krokem jedna. Pro složitější sekvence lze využít funkci seq(). První parametry jsou stejné jako při využití předchozí metody, třetí parametr je krok. Lze také využít pojmenované parametry, **from** značí začátek, **to** konec, **by** krok a parametr **length** délku sekvence. Pro generování sekvence jsou potřeba tři parametry, pokud ale chybí parametr **by**, je použita výchozí hodnota jedna. Lze také použít parametr **along**, který určuje délku sekvence pomocí délky zadaného vektoru.

```
> seq(0,10,2)
[1]  0  2  4  6  8 10
> seq(from=11,by=-1,length=5)
[1] 11 10  9  8  7
```

Další užitečnou funkcí je rep(). Jako parametr je třeba vektor a buď parametr **times**, který určí, kolikrát se vektor zkopíruje, nebo **each**, který určí, kolikrát se zkopíruje každá hodnota. [1]

```
> rep(c(2,5),times=3)
[1] 2 5 2 5 2 5
```

```
> rep(c(2,5),each=3)
[1] 2 2 2 5 5 5
```

Z vektoru lze získat hodnotu uvedením jejího pořadí do hranatých závorek. První hodnota má ve vektoru index jedna. [1]

```
> x
[1] 11 22 33 44 55
> x[3]
[1] 33
```

Vybrat lze i více hodnot najednou, pokud je do hranatých závorek vložen vektor čísel. Pokud jsou všechna čísla kladná, výsledný vektor bude mít hodnoty na daných indexech původního vektoru, pokud budou záporná, hodnoty na těchto indexech budou z výsledku vymazána. [1]

```
> x[c(1,4)]
[1] 11 44
> x[c(-2,-4)]
[1] 11 33 55
```

Indexovat lze také logickým vektorem. Výsledkem je vektor s hodnotami, na jejichž místě je v logickém vektoru hodnota TRUE. Pokud není logický vektor stejně dlouhý jako vektor původní, recykluje se.[1]

```
> x[c(TRUE,FALSE,FALSE,TRUE,TRUE)]
[1] 11 44 55
```

Pokud má vektor nastaven atribut **names**, lze indexovat i jménem hodnoty. Pravidla jsou podobná jako při indexaci kladnými čísly. [1]

```
> names(x) <- c("po","ut","st","ct","pa")
> x
po ut st ct pa
11 22 33 44 55
> x[c("po","st","pa")]
po st pa
11 33 55
```

3.2 Pole, matice

Pole je datová struktura podobná vektoru, navíc však dokáže ukládat data ve více rozměrech. Daná hodnota tedy není jednoznačně určená jedním číslem, ale několika čísly, v každém rozměru určující pořadí. Stejně jako vektor musí být všechny hodnoty stejného typu určené módem pole. Zvláštní verzí pole je matice, dvourozměrné pole, které má díky svému častému použití funkce navíc. [1]

Rozměry pole jsou definovány atributem **dim**. Tento atribut musí být vektor kladných čísel a každá hodnota určuje délku pole v jednom rozměru. Délka

atributu odpovídá počtu rozměrů pole. Vektor lze přeměnit na pole přiřazením atributu **dim**, počet hodnot vektoru však musí odpovídat počtu hodnot ve výsledném poli. [1]

```
> x
[1] 1 2 3 4 5 6
> dim(x) <- c(2,3)
> x
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

Pole má také atribut **dimnames**. Reprezentuje popis řádků nebo sloupců ve všech rozměrech. Jedná se o list obsahující vektory řetězců. Pozice v listu se shoduje s pozicí rozměru v atributu **dim**. [3]

```
> dimnames(x) <-
list(c("horni","dolni"),c("leva","stredni","prava"))
> x
      leva stredni prava
horni    1       3     5
dolni    2       4     6
```

Kromě přidání atributu **dim** vektoru popsaném výše, lze pole také vytvořit funkcí array(). První parametr je vektor s daty a druhý je vektor s rozměry pole. V tomto případě však nemusí být vektor s daty dostatečně velký, funkce využívá recyklaci k rozšíření vektoru. [1]

Pole lze indexovat podobně jako vektor, ovšem je potřeba jeden index za každý rozměr navíc. [1]

```
> x
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> x[2,2]
[1] 4
```

Pokud nějaký index chybí, R vrátí všechny hodnoty s libovolným indexem v daném rozměru. [1]

```
> x[2,]
[1] 2 4 6
```

Pole lze také indexovat vektory. Pro každý rozměr je třeba vektor se všemi indexy, které mají být ve výsledném poli.

```
> x
     [,1] [,2] [,3]
```

```
[1,]   11   12   13
[2,]   21   22   23
[3,]   31   32   33
> x[c(1,2),c(1,3)]
     [,1] [,2]
[1,]   11   13
[2,]   21   23
```

Z pole lze také získat více jednotlivých hodnot indexováním maticí. Matice musí mít počet sloupců odpovídající počtu rozměrů. Jednotlivé řádky matice reprezentují hodnoty z původního pole, čísla v řádku jsou indexy požadované hodnoty. Výsledkem je vektor se všemi hodnotami. Délka výsledného vektoru je stejná jako počet řádků matice použité jako index. [1]

```
> x
     [,1] [,2] [,3]
[1,]   11   12   13
[2,]   21   22   23
[3,]   31   32   33
> index
     [,1] [,2]
[1,]    1    2
[2,]    1    3
[3,]    2    2
[4,]    3    2
> x[index]
[1] 12 13 22 32
```

Pro matice existují v R zvláštní funkce. Matice lze vytvořit funkcí matrix(). Této funkci stačí pouze jeden rozměr, druhý dopočítá. Pro sloupce je to parametr **ncol** a pro řádky parametr **nrow**. Ve výchozím stavu funkce zaplňuje matici po sloupcích, toto chování je možné změnit přidáním hodnoty TRUE pro parametr **byrow**. [3]

```
> matrix(1:6,ncol =3)
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> matrix(1:6,ncol =3,byrow=TRUE)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

Další způsobem vytvoření matice je spojením vektorů po řádcích, pomocí metody rbind(), nebo sloupcích, pomocí cbind(). Místo vektorů lze také použít jiné matice, ale počet řádků nebo sloupců musí být stejný.

```
> rbind(1:4,11:14,21:24)
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]   11   12   13   14
[3,]   21   22   23   24
```

Pomocí funkce t() lze matici transponovat, tedy prohodit sloupce a řádky. Pokud vstupem funkce bude pouze vektor, funkce ho bude brát jako matici s jedním sloupcem.

```
> x
     [,1] [,2]
[1,]    1   11
[2,]    2   22
[3,]    3   33
> t(x)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]   11   22   33
```

Další užitečnou funkcí je diag(). V případě že vstupem je matice, vrátí vektor, který bude obsahovat čísla na její diagonále. V případě, že vstupem je vektor, vrátí matici, která bude mít daný vektor jako diagonálu a ostatní prvky s hodnotou nula.

```
> x
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> diag(x)
[1] 1 5 9
> diag(1:3)
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    2    0
[3,]    0    0    3
```

Pokud je jediným parametrem číslo, funkce vrátí jednotkovou matici s délkou daného čísla. Počet řádků a sloupců lze omezit parametry **nrow** respektive **ncol**. V tom případě však funkce nevrátí jednotkovou matici, ale matici, která bude mít všechna čísla na diagonále stejná jako parametr funkce.

```
> diag(4,nrow=3)
     [,1] [,2] [,3]
[1,]    4    0    0
[2,]    0    4    0
[3,]    0    0    4
```

Řádkové součty lze získat pomocí funkce rowSums() a průměry jednotlivých řádků pomocí funkce rowMeans(). Pro sloupce lze využít funkce colSums() respektive colMeans().

```
> x
     [,1] [,2] [,3]
[1,]    1   10  100
[2,]    2   20  200
[3,]    3   30  300
```

```
> rowSums(x)
[1] 111 222 333
> rowMeans(x)
[1]  37  74 111
```

Pomocí funkce max.col() lze získat pozici nejvyšší hodnoty v každém řádku. Výsledkem je vektor stejné délky, jako je počet řádků.

```
> x
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    6    2   55
[2,]    4   95    8    3    4
[3,]    4    1   59    7    8
> max.col(x)
[1] 5 2 3
```

Pokud však nejsou potřeba pouze pozice, ale přímo hodnoty, je možné využít výsledné pozice jako indexy matice. Prvním indexem musí být číslo řádku, a protože funkce pracuje postupně, lze využít sekvenci od jedné do počtu řádků. Jako druhým indexem je sloupec, jehož pozici vrací daná funkce. Výsledek bude vektor nejvyšších hodnot v každém řádku.

```
> x[cbind(1:3,max.col(x))]
[1] 55 95 59
```

Pro matice také existují speciální operátory. První z nich, %*%, se používá pro maticové násobení. Pro úspěšné násobení musí být počet sloupců levé matice stejný jako počet řádků pravé matice.

```
> x
     [,1] [,2]
[1,]    1    2
[2,]    2    5
> y
     [,1] [,2]
[1,]    3    2
[2,]    5    3
> x %*% y
     [,1] [,2]
[1,]   13    8
[2,]   31   19
```

Opačnou operaci lze provést funkcí solve(). Vrací matici, pro kterou platí, že vynásobením prvního parametru funkce s touto maticí, vyjde jako výsledek druhý parametr funkce.

```
> solve(x,x %*% y)
     [,1] [,2]
[1,]    3    2
```

```
[2,]    5    3
```

Druhým operátorem je %o%. Tento operátor vynásobí každý prvek jednoho pole s prvkem toho druhého, a uloží do nového pole, které má počet dimenzí stejný jako součet počtů rozměrů původních polí. Této funkce lze využít pro matici všech kombinací dvou vektorů.

```
> c(1,2,3) %o% c(1,10,100)
     [,1] [,2] [,3]
[1,]    1   10  100
[2,]    2   20  200
[3,]    3   30  300
```

3.3 Seznam

Seznam je kolekce jednotlivých komponent. Komponentou listu může být jakýkoliv objekt a na rozdíl od vektoru nemusí být komponenty stejného typu. [1]

Stejně jako vektor má seznam atributy **mode** a **length**, nepovinně také **names**. Atribut mode je však u seznamu vždy stejný – list. [1]

Length určuje počet komponent seznam. Vnitřní členění komponentů pak nemá na atribut žádný vliv, pokud seznam bude obsahovat 2 vektory o délce 3, do délky se započítávají pouze 2 vektory, ne 6 vnitřních hodnot vektorů. [1]

Names je, stejně jako u vektoru, i u seznamu vektor řetězců pojmenovávající jednotlivé komponenty. [1]

Pro vytváření seznamů se používá funkce list(). Parametry funkce jsou jednotlivé komponenty listu. [1]

```
> list(c(4,6,8),"retezec",5+6i)
[[1]]
[1] 4 6 8

[[2]]
[1] "retezec"

[[3]]
[1] 5+6i
```

Nový seznam lze také vtvořit spojením již existujících seznamů pomocí funkce c(). [1]

```
> c(list(4,5),list("ctyi","pet"))
```

Pro získání určité komponenty ze seznamu se využívá dvojitých hranatých závorek. Indexovat lze číslem pořadí komponenty, nebo jménem komponenty, pokud má seznam atribut **names**. [3]

```
> x <- list(jmeno=c("pert","pavel"),vek=c(22,25))
> x[[1]]
[1] "pert"  "pavel"
> x[["vek"]]
[1] 22 25
```

Alternativně lze využít operátor $, který funguje pro pojmenované komponenty stejně. [3]

```
> x$vek
[1] 22 25
```

Lze použít i jednoduché hranaté závorky, ty ovšem nevrací přímo komponenty, ale seznam s vybranými komponentami. Protože ale vrací seznam, je možné vybrat více komponent pomocí indexování vektorem čísel. [3]

```
> x[c(1,2)]
$jmeno
[1] "pert"  "pavel"


$vek
[1] 22 25
```

3.4 Datová tabulka

Datová tabulka je dvourozměrná datová struktura, která vyžaduje, aby data uvnitř sloupce byla stejná, každý sloupec však může mít vlastní typ. Technicky se jedná o speciální případ seznamu, jehož komponenty mohou být pouze vektory a všechny musí mít stejnou délku. [3]

Datová tabulka se vytváří funkcí data.frame(), která přijímá jednotlivé vektory. Parametry lze pojmenovat, jména pak slouží jako názvy sloupců. [3]

```
> data.frame(jmeno=c("Petr","Pavel","Patrik"),semestr=c(3,1,5
),obor=c("ai","ai","im"))
   jmeno semestr obor
1   Petr       3   ai
2  Pavel       1   ai
3 Patrik       5   im
```

Protože je datová tabulka seznamem, lze ji i stejně indexovat. Nabízí ovšem také indexování podobné matici, tedy pomocí dvou čísel nebo řetězců. [3]

```
> x[2,2]
[1] 1
```

Stejně tak lze indexovat celé sloupce nebo řádky vynecháním indexu. [3]

```
> x[,2]
[1] 3 1 5
> x[3,]
```

```
   jmeno semestr obor
3 Patrik       5   im
```

Do datové tabulky lze přidávat řádky pomocí funkce rbind(), stejně jako tomu bylo v případě matic. Počet sloupců ve spojovaných tabulkách musí být stejný a stejné také musí být názvy sloupců. [3]

```
> 
rbind(x,data.frame(jmeno=c("David","Milan"),semestr=c(3,1),ob
or=c("fm","fm")))
   jmeno semestr obor
1   Petr       3   ai
2  Pavel       1   ai
3 Patrik       5   im
4  David       3   fm
5  Milan       1   fm
```

Přidat sloupce lze metodou cbind(). Lze však také přiřadit vektor k novému indexu datové tabulky. Indexovat můžeme názvem nebo číslem sloupce. [3]

```
> x$vek <- c(19,18,20,19,18)
> x
   jmeno semestr obor vek
1   Petr       3   ai  19
2  Pavel       1   ai  18
3 Patrik       5   im  20
4  David       3   fm  19
5  Milan       1   fm  18
```

Sloupce lze odstranit vložením hodnoty NULL do daného sloupce. [3]

```
> x$vek <- NULL
> x
   jmeno semestr obor
1   Petr       3   ai
2  Pavel       1   ai
3 Patrik       5   im
4  David       3   fm
5  Milan       1   fm
```

Pro spojování více tabulek slouží funkce merge(). Tato funkce spojí dvě tabulky do jedné. Pokud nemají tabulky ani jeden sloupec stejný, výsledná tabulka obsahuje kartézský součin těch původních. To znamená, že výsledná tabulka obsahuje všechny kombinace řádků z první tabulky a z druhé tabulky.

```
> zaci
  jmeno predmet znamka
1 Jakub   TNPW2      2
2 Jakub     ZT2      3
3 Jakub   PGRF2      1
4   Jan   PSIT2      2
5   Jan   TNPW2      1
6 Marie     ZT2      1
7 Marie    DIMA      2
```

```
> predmety
  zkratka                    nazev
2     ZT2 Znalostní technologie 2
3    DIMA    Diskrétní matematika
4   PGRF2    Počítačová grafika 2
5    DBS2    Databázové systémy 2
> merge(zaci,predmety)
   jmeno predmet znamka zkratka                    nazev
1  Jakub   TNPW2      2     ZT2 Znalostní technologie 2
2  Jakub     ZT2      3     ZT2 Znalostní technologie 2
3  Jakub   PGRF2      1     ZT2 Znalostní technologie 2
4    Jan   PSIT2      2     ZT2 Znalostní technologie 2
5    Jan   TNPW2      1     ZT2 Znalostní technologie 2
...
(zbytek řádků vynechán)
```

Jedna z možností, jak spojit tabulky podle jednoho sloupce je pojmenovat daný sloupce v obou tabulkách stejně. Řádky, které nebudou mít v druhé tabulce protějšek, budou vynechány.

```
> predmety2 <- predmety
> names(predmety2)[1] = "predmet"
> merge(zaci,predmety2)
  predmet jmeno znamka                    nazev
1    DIMA Marie      2    Diskrétní matematika
2   PGRF2 Jakub      1    Počítačová grafika 2
3     ZT2 Jakub      3 Znalostní technologie 2
4     ZT2 Marie      1 Znalostní technologie 2
```

Lepší způsob je však využít parametry **by.x** a **by.y**. Parametrem **by.x** je název spojovacího sloupce v první tabulce a parametrem **by.y** název sloupce v druhé tabulce.

```
> merge(zaci,predmety,by.x = "predmet", by.y = "zkratka")
  predmet jmeno znamka                    nazev
1    DIMA Marie      2    Diskrétní matematika
2   PGRF2 Jakub      1    Počítačová grafika 2
3     ZT2 Jakub      3 Znalostní technologie 2
4     ZT2 Marie      1 Znalostní technologie 2
```

Pokud je třeba ve výsledné tabulce ponechat i řádky, které nemají protějšek v druhé tabulce, je třeba předat parametru **all.x** respektive **all.y** hodnotu TRUE. Popřípadě lze nastavit pro obě tabulky najednou parametrem **all**.

```
> merge(zaci,predmety,by.x = "predmet", by.y = "zkratka",
all= TRUE)
  predmet jmeno znamka                    nazev
1    DIMA Marie      2    Diskrétní matematika
2   PGRF2 Jakub      1    Počítačová grafika 2
3   PSIT2   Jan      2                    <NA>
4   TNPW2 Jakub      2                    <NA>
5   TNPW2   Jan      1                    <NA>
6     ZT2 Jakub      3 Znalostní technologie 2
```

```
7     ZT2 Marie      1 Znalostní technologie 2
8    DBS2  <NA>     NA    Databázové systémy 2
```

Funkce names() vrací vektor názvů sloupců. Název sloupce lze tedy změnit pomocí přiřazení nového názvu na místo toho starého.

```
> names(x)[3] <- "studijni obor"
> x
   jmeno semestr studijni obor
1   Petr       3            ai
2  Pavel       1            ai
3 Patrik       5            im
4  David       3            fm
5  Milan       1            fm
```

Protože je datová struktura specializovaný seznam, lze jí také indexovat vektorem. Výsledkem je datová tabulka, která obsahuje indexované sloupce v pořadí, ve kterém byly předány. Toho lze využít k prohození sloupců.

```
> x[c(3,1,2)]
  studijni obor  jmeno semestr
1            ai   Petr       3
2            ai  Pavel       1
3            im Patrik       5
4            fm  David       3
5            fm  Milan       1
```

Pomocí funkce subset() lze z datové tabulky vybrat pouze data, která splňují určité podmínky. Prvním parametrem je samotná datová tabulka. Druhým parametrem jsou podmínky. Podmínky se skládají ze jmen sloupců a logických operátorů. Znak „&“ je operátorem pro logické a. Pro logické nebo se používá znak „|“. Pro rovnost se používá dvojité rovnítko.

```
> x
  id    jmeno prijmeni ročnik
1  1     Petr    Novák      1
2  2    Lukáš  Kopecky      2
3  3      Jan  Zátopek      1
4  4 Vladimír   Hlinka      3
5  5    Jakub  Novotný      1
> subset(x, ročnik == 1 & id < 4)
  id jmeno prijmeni ročnik
1  1  Petr    Novák      1
3  3   Jan  Zátopek      1
```

Funkce subset() má také nepovinný třetí parametr pro určení, které sloupce má funkce vrátit. Výsledná datová tabulka respektuje pořadí sloupců tak, jak byli předány funkci.

```
> subset(x, ročník == 1 & id < 4, c(prijmeni, jmeno))
  prijmeni jmeno
```

```
1    Novák  Petr
3  Zátopek   Jan
```

Místo jmen sloupců lze také použít indexy sloupců.

Datové tabulky mohou obsahovat neúplné řádky, tedy řádky, které nemají hodnotu pro všechny sloupce. Pomocí funkce complete.cases() lze zjistit, které řádky jsou úplné. Tato funkce vrací vektor logických hodnot.

```
> x
                kniha                autor
1     Romeo a julie William Shakespeare
2            Beowulf                 <NA>
3       Bílá velryba      Herman Melville
4            Bídníci          Victor Hugo
5 Tisíc a jedna noc                 <NA>
> complete.cases(x)
[1]  TRUE FALSE  TRUE  TRUE FALSE
```

Pro zobrazení všech úplných řádků je potřeba výsledek funkce použít jako parametr pro indexaci. Index pro sloupce by měl zůstat prázdný, to značí, že by měli být vráceny všechny sloupce.

```
> x[complete.cases(x),]
          kniha              autor
1 Romeo a julie William Shakespear
3  Bílá velryba    Herman Melville
4       Bídníci         Victor Hugo
```

Další možností je využít funkci na.omit(). Tato funkce vrátí data, ale vynechá veškeré řádky, na kterých je hodnota NA.

```
> na.omit(x)
          kniha                autor
1 Romeo a julie William Shakespeare
3  Bílá velryba     Herman Melville
4       Bídníci          Victor Hugo
```

Další funkcí, pracující s chybějícími hodnotami je na.fail(). Tato funkce vrátí chybu, pokud data obsahují hodnoty NA.

```
> na.fail(x)
Error in na.fail.default(x) : missing values in object
```

Pokud objekt neobsahuje hodnoty NA, funkce vrátí původní data. Této vlastnosti lze využít pro řetězení funkcí, pokud je třeba, aby vrátili výsledek pouze v případě, že v objektu nejsou žádné hodnoty NA.

```
> subset(na.fail(x),autor=="William Shakespeare")
Error in na.fail.default(x) : missing values in object
> subset(na.fail(na.omit(x)),autor=="William Shakespeare")
          kniha                autor
1 Romeo a julie William Shakespeare
```

Všechny tyto funkce pracují s chybějícími daty, liší se zejména v návratové hodnotě. Funkce complete.cases() vrací vektor logických hodnot, značících, které řádky neobsahují hodnotu NA, zatímco na.omit() vrací přímo data bez chybějících údajů. Funkce na.fail() vrací chybu, pokud objekt obsahuje hodnoty NA, a vrací původní objekt, pokud je neobsahuje. Díky tomu ho lze využít pro úkoly, které mají selhat, pokud data obsahují chybějící údaje.

Pro filtrování duplicitních řádků se používá funkce unique(). Tato funkce vynechá všechny řádky, které již byly v tabulce dříve obsaženy.

```
> x
  jmeno prijmeni
1  Petr    Novák
2  Petr  Kopecký
3 Jakub  Novotný
4  Petr    Novák
5 Jakub  Novotný
> unique(x)
  jmeno prijmeni
1  Petr    Novák
2  Petr  Kopecký
3 Jakub  Novotný
```

Data v datové tabulce lze roztřídit podle jistého sloupce pomocí funkce aggregate(). Tato funkce sloučí řádky, které mají v daném sloupci stejnou hodnotu, a hodnoty ostatních sloupců sloučí pomocí zadané funkce. Prvním parametrem funkce je datová tabulka, ale bez sloupce, podle kterého se mají data sloučit. V následujícím příkladě se tedy jedná pouze o sloupec „cislo“. Druhým je seznam sloupců, podle kterých se mají data sloučit a třetím je funkce, která provede slučování hodnot. Například pro součet je možné využít funkci sum(), do funkce se však předává bez závorek.

```
> x
  jmeno cislo
1  petr     1
2 jirka    20
3 jirka     2
4  petr    10
5 jirka   200
> aggregate(x$cislo,list(x$jmeno),sum)
  Group.1   x
1   jirka 222
2    petr  11
```

Obdobně lze využít funkci mean() pro průměr nebo funkci sd() pro výpočet směrodatné odchylky.

Agregovat data lze také pomocí funkce table(). Tato funkce přijme vektor a spočítá, kolikrát se daná hodnota ve vektoru vyskytuje.

```
> x
  id_studenta obor rocnik
1           1   ai  prvni
2           2   ai  prvni
3           3   fm  druhy
4           4   im  treti
5           5   ai  druhy
6           6   im  treti
> table(x$obor)

ai fm im
 3  1  2
```

Pro agregaci více vektorů lze funkci předat více parametrů. Výsledkem je tabulka, která má stejný počet rozměrů jako počet parametrů funkce. Pokud je parametrů více než 2, je vhodné místo funkce table() použít funkci ftable(), která výsledek lépe zformátuje.

```
> table(x$obor,x$rocnik)

   druhy prvni treti
ai     1     2     0
fm     1     0     0
im     0     0     2
```

Pro celkový souhrn lze také využít funkci summary(), která u číselných sloupců spočítá průměr, medián, nejnižší a nejvyšší hodnotu a další statistické údaje, a u ostatních hodnot spočítá, kolikrát se v tabulce vyskytují.

```
> summary(x)
  id_studenta   obor     rocnik
 Min.   :1.00   ai:3   druhy:2
 1st Qu.:2.25   fm:1   prvni:2
 Median :3.50   im:2   treti:2
 Mean   :3.50
 3rd Qu.:4.75
 Max.   :6.00
```

3.5 Faktor

Faktor je datová struktura podobná vektoru. Liší se však v tom, že každá hodnota je ve faktoru uložena pouze jednou jako úroveň, hodnoty jsou pak ve faktoru reprezentovány číslem jejich úrovně. [4]

Při získávání hodnoty R zjistí, která hodnota patří danému číslu a vrátí přímo uloženou hodnotu. Proto jsou faktory velmi efektivní pro ukládání textových hodnot. [4]

Faktor lze vytvořit z vektoru pomocí funkce factor(). Do konzole se faktor vypíše jako vektor s hodnotami, navíc však také vypíše všechny úrovně.

```
> x
[1] "jedna" "dva"   "jedna" "tri"   "dva"   "tri"   "tri"
"jedna"
> f <- factor(x)
> f
[1] jedna dva   jedna tri   dva   tri   tri   jedna
Levels: dva jedna tri
```

Úrovně lze také získat pomocí funkce levels().

```
> levels(f)
[1] "dva"   "jedna" "tri"
```

Dalším způsobem jak vytvořit faktor, je předat funkci factor() vektor čísel a jako parametr **labels** předat vektor úrovní.

```
> f <- factor(c(1,3,2,4,1,4,3,1,2),
labels=c("A","B","C","D"))
> f
[1] A C B D A D C A B
Levels: A B C D
```

Jednotlivé hodnoty lze indexovat stejně jako vektory, tedy číslem jejich pořadí v hranatých závorkách.

```
> f[2]
[1] C
Levels: A B C D
```

Faktory nejsou považovány za uspořádané, není tedy možnost je porovnávat. R v takovém případě vrátí chybu.

```
> f[2] > f[3]
[1] NA
Warning message:
In Ops.factor(f[2], f[3]) : > not meaningful for factors
```

Pokud je třeba považovat faktor za uspořádaný, je nutné předat parametru **ordered** hodnotu TRUE. Hodnoty budou pokládány za uspořádané vzestupně podle pořadí, v jakém jsou v parametru **labels**.

```
> f <- factor(c(1,3,2,4,1,4,3,1,2),
labels=c("A","B","C","D"),ordered=TRUE)
> f
[1] A C B D A D C A B
Levels: A < B < C < D
```

Pro uspořádaný faktor lze jednotlivé hodnoty porovnávat.

```
> f[2] > f[3]
[1] TRUE
```

Faktor lze změnit zpět na vektor pomocí funkce as.numeric(). Jejím jediným parametrem je daný faktor.

```
> as.numeric(f)
[1] 1 3 2 4 1 4 3 1 2
```

4 Import a export dat

V této části se práce zabývá způsoby importu a exportu dat z jazyka R.

4.1 Textové soubory

Načíst data z textového souboru lze pomocí funkce scan(). Při použití této funkce bez parametru získává data přímo zadaná do konzole, a to tak dlouho, dokud není na vstupu dvakrát enter.

```
> scan()
1: 5 6 8 9
5: 4 2
7:
Read 6 items
[1] 5 6 8 9 4 2
```

Pro import ze souboru je třeba zadat jako parametr cestu k souboru. Pro cestu v systému Windows je třeba zpětná lomítka zdvojit, neboť znak za zpětným lomítkem je interpretováno speciálně, například \n reprezentuje znak pro konec řádku. [4]

Standardně funkce scan() interpretuje data jako čísla. To lze změnit parametrem **what**. Parametrem může cokoliv, ale musí být stejného typu, jako importovaná data. Například pro import textu jsou všechny tyto možnosti správné:

```
> scan("g:\\r.txt",what="")
> scan("g:\\r.txt",what="cokoliv")
> scan("g:\\r.txt",what=character())
```

Parametrem **what** může být také seznam. Funkce v tom případě zaplňuje jednotlivé parametry seznamu. Typ dat je potřeba deklarovat pro každý prvek seznamu. Funkce ukládá data střídavě do každého prvku seznamu. Soubor s textem „1 jedna 2 dve 3 tri“ lz načíst tímto způsobem:

```
> scan("g:\\r.txt",what=list(numeric(),character()))
Read 3 records
[[1]]
[1] 1 2 3

[[2]]
[1] "jedna" "dve"   "tri"
```

Funkce předpokládá, že jsou data oddělena mezerou, odsazením nebo novým řádkem. Pokud jsou data oddělena jiným znakem, je třeba je předat funkci parametrem **sep**. Pro oddělení čárkou by funkce měla vypadat takto:

```
scan("g:\\r.txt",sep=",")
```

Reálná čísla jsou interpretována jako čísla s desetinou tečkou, pokud je použit jiný znak, funkce vrací chybu.

```
Error in scan(file, what, nmax, sep, dec, quote, skip,
nlines, na.strings,  :
  scan() expected 'a real', got '3,14'
```

Funkce ovšem nabízí možnost nastavit místo desetinné tečky jakýkoliv znak, a to parametrem **dec**. Hodnotou by měl být jediný znak. Pro desetinnou čárku by funkce měla vypadat takto:

```
> scan("g:\\r.txt",dec=",")
Read 1 item
[1] 3.14
```

Pro import celých tabulek lze využít funkci read.table(). Tato funkce vrací datovou tabulku, takže je ideální pro načítání dat různých typů. [4] Funkce očekává jako vstup cestu k souboru a stejně jako funkce scan() očekává, že data jsou oddělené mezerou, odsazení nebo novým řádkem. To lze obdobně změnit pomocí parametru **sep**.

Pokud první řádek souboru obsahuje názvy sloupců, je třeba nastavit parametr **header** na hodnotu TRUE, jinak budou načteny jako první řádek dat. Alternativně lze také zadat názvy sloupců jako vektor do parametru **col.names.** Pokud nejsou názvy nijak specifikovány, budou pojmenovány ve formátu velké písmeno V a za ním číslo sloupce. [4]

```
> read.table("g:\\r.txt")
                          V1      V2
1                  Predmet zkratka
2    Diskrétní matematika    DIMA
3 Znalostní technologie 2     ZT2
4    Počítačová grafika 2   PGRF2
5       Počítačové sítě 4   PSIT4
6    Základy matematiky 2   ZMAT2
> read.table("g:\\r.txt",header=TRUE)
                  Predmet zkratka
1    Diskrétní matematika    DIMA
2 Znalostní technologie 2     ZT2
3    Počítačová grafika 2   PGRF2
4       Počítačové sítě 4   PSIT4
5    Základy matematiky 2   ZMAT2
```

Pro pojmenování jednotlivých řádku slouží parametr **row.names**. Hodnotou by měl být vektor všech názvů.

```
> read.table("g:\\r.txt",header=TRUE,row.names=c("První","Druhý
","Třetí","Čtvrtý","Pátý"))
                          Predmet zkratka
První      Diskrétní matematika    DIMA
Druhý  Znalostní technologie 2     ZT2
Třetí      Počítačová grafika 2   PGRF2
Čtvrtý        Počítačové sítě 4   PSIT4
Pátý       Základy matematiky 2   ZMAT2
```

Počet přečtených řádků lze nastavit parametrem **nrows**. V případě, že tento parametr chybí, nebo je neplatný (např. je záporný nebo nula), funkce načte celý soubor.

```
> read.table("g:\\r.txt",header=TRUE,nrows=4)
                   Predmet zkratka
1    Diskrétní matematika    DIMA
2 Znalostní technologie 2     ZT2
3    Počítačová grafika 2   PGRF2
4       Počítačové sítě 4   PSIT4
```

Lze také při čtení přeskočit určitý počet řádků. Slouží k tomu parametr **skip**. Pokud je nastaven parametr **header**, první řádek se jmény sloupců není přeskočen, počítají se pouze řádky s daty.

```
> read.table("g:\\r.txt",header=TRUE,skip=1)
     Diskrétní.matematika  DIMA
1 Znalostní technologie 2   ZT2
2    Počítačová grafika 2 PGRF2
3       Počítačové sítě 4 PSIT4
4    Základy matematiky 2 ZMAT2
```

Pro běžně používané formáty obsahuje R pomocné funkce, které volají metodu read.table() s různými parametry:

- Funkce read.csv() používá jako oddělovač dat čárku, a jako oddělovač desetinných míst používá tečku.
- Funkce read.csv2() používá jako oddělovač dat středník a jako oddělovač desetinných míst používá čárku.
- Funkce delim() používá jako oddělovač dat odsazení tabulátorem a jako oddělovač desetinných míst používá tečku.
- Funkce delim2() používá jako oddělovač dat odsazení tabulátorem a jako oddělovač desetinných míst používá čárku.

Pokud data nejsou v textu ničím oddělena, ale jednotlivé sloupce zabírají v textu pevný počet znaků, lze je načíst pomocí funkce read.fwf(). Prvním parametrem je

samotný soubor, druhým pak vektor čísel, reprezentující počet znaků pro jednotlivé sloupce.

```
> read.fwf("data.txt",c(21,5,1))
                        V1    V2 V3
1 Principy počítačů     PRIPO  1
2 Architektura počítačů ARCH   2
3 Programování 1        PRO1   2
4 Databázové systémy 1  DBS    3
5 Počítačová grafika I  PGRF1  2
```

Pokud bude ve vektoru délek sloupců záporné číslo, funkce daný počet znaků přeskočí. Tímto lze přeskočit sloupce, které nemají být načteny.

```
> read.fwf("data.txt",c(21,-5,1))
                        V1 V2
1 Principy počítačů      1
2 Architektura počítačů  2
3 Programování 1         2
4 Databázové systémy 1   3
5 Počítačová grafika I   2
```

V případě že soubor obsahuje názvy sloupců, je třeba nastavit parametr **header** na hodnotu TRUE, obdobně jako u funkce read.table(). Názvy sloupců ovšem musí být odděleny, nelze je definovat počtem znaků jako data. Pokud nejsou odděleny odsazením, je třeba oddělující znak předat funkci jako parametr **sep**.

```
> read.fwf("data.txt",c(21,5,1),header=TRUE)
                   nazev zkratka znamka
1 Principy počítačů       PRIPO       1
2 Architektura počítačů   ARCH        2
3 Programování 1          PRO1        2
4 Databázové systémy 1    DBS         3
5 Počítačová grafika I    PGRF1       2
```

Funkce read.fwf() v názvech nechává prázdná místa, kterými jsou vyplněny prvky kratší než sloupec. Pokud je třeba tyto mezery odstranit, je možné předat funkci parametr **strip.white** s hodnotou TRUE.

```
> read.fwf("data.txt",c(21,5,1),header=TRUE,strip.white=TRUE)
                   nazev zkratka znamka
1     Principy počítačů   PRIPO       1
2 Architektura počítačů    ARCH       2
3        Programování 1     PRO1       2
4  Databázové systémy 1      DBS       3
5  Počítačová grafika I    PGRF1       2
```

Pokud jsou data, která reprezentují v tabulce jeden řádek, v souboru na více řádcích, lze místo vektorů šířek sloupců předat funkci seznam vektorů. Každý prvek v seznamu reprezentuje řádek v souboru. Jednotlivé prvky seznamu jsou

vektory šířek jednotlivých sloupců stejně jako v případě jednořádkových dat. Například pokud by do souboru z předchozích příkladů byl ke každému předmětu přidán nový řádek s datem splnění předmětu a s počtem pokusů na předmět, funkce by vypadala takto:

```
>
read.fwf("data.txt",list(c(21,5,1),c(10,1)),header=TRUE,strip
.white=TRUE)
                  nazev zkratka znamka      datum pokus
1     Principy počítačů   PRIPO      1  4.1.2012     1
2 Architektura počítačů    ARCH      2  9.1.2012     3
3        Programování 1    PRO1      2  5.1.2012     2
4  Databázové systémy 1     DBS      3 17.1.2012     1
5  Počítačová grafika I   PGRF1      2 19.1.2012     1
```

Pro zápis textových soborů se využívá funkcí write() pro vektory či matice a write.table() pro zápis datových tabulek.

Funkce write() požaduje pouze data, která má uložit. Data uloží jako soubor pojmenovaný „data“ do pracovní složky. Tu lze vypsat pomocí funkce getwd(), a změnit ji lze pomocí funkce setwd() s jediným parametrem, a to řetězcem s cestou ke složce. Pro uložení na jiné místo lze zadat funkci write() cestu jako druhý parametr.

Funkce write() píše data sekvenčně za sebou. Pokud je třeba data uložit do sloupců, je třeba nastavit parametr **ncolumns** na příslušný počet sloupců. Funkce v tom případě ukládá data po řádcích, vždy když dosáhne posledního sloupce, ukládá na další řádek.

```
> write(c(1,10,100,2,20,200,3,30,300),"data.txt",ncolumns =
3)
> read.table("data.txt")
  V1 V2  V3
1  1 10 100
2  2 20 200
3  3 30 300
```

Stejně jako funkce read(), i funkce write() obsahuje parametr **sep**. Tento parametr označuje znak, kterým budou jednotlivá data oddělena od sebe. Dalším užitečným parametrem je **append**. Pokud je jeho hodnota TRUE, data jsou zapsána na konec souboru. V opačném případě jsou existující data přepsána.

```
> write(c(4,40,400,5,50,500),"data.txt",ncolumns = 3, append
= TRUE)
> read.table("data.txt")
  V1 V2  V3
1  1 10 100
```

```
2  2 20 200
3  3 30 300
4  4 40 400
5  5 50 500
```

Funkce write.table() má navíc některé další parametry. Parametr **na** určuje, jaký text se uloží místo hodnoty NA. Parametr dec, stejně jako u funkce read.table(), určuje, jaký oddělovač desetinných míst se při ukládání použije. Parametr **col.names** může být logická hodnota, nebo vektor řetězců. Pokud je TRUE, do souboru budou vypsána jména sloupců. Pokud je FALSE, jména zapsaná nebudou. Pokud je hodnotou atributu **col.names** vektor řetězců, budou použity jako jména sloupců a vypsána. Obdobně lze využít parametru **row.names** pro řádky.

Vstupem pro tyto funkce nemusí být pouze soubor na lokálním počítači, lze také načítat data z internetu. Místo cesty k souboru je třeba předat funkci internetovou adresu souboru.

```
> scan("http://www.r-project.org/",what="",sep="\n",n=5)
Read 5 items
[1] "<!DOCTYPE html>"
[2] "<html lang=\"en\">"
[3] "  <head>"
[4] "    <meta charset=\"utf-8\">"
[5] "    <meta http-equiv=\"X-UA-Compatible\"
content=\"IE=edge\">"
```

Pokud jsou načítány například soubory csv, lze s daty okamžitě pracovat. Html soubory však neobsahují data ve snadno čitelné formě, pokud je třeba získat z nich data, je potřeba tyto data v textu vyhledat. K tomuto účelu lze využít funkci grep(). Tato funkce přijímá jako parametry regulérní výraz a text, ve kterém ho má najít. Tato funkce vrací indexy vektoru, ve kterých najde shodu, pro navrácení nalezeného textu je třeba předat parametru **value** hodnotu TRUE. Takto lze například získat nadpis stránky.

```
> x = scan("http://www.r-project.org/",what="",sep="\n")
> grep("<title>",x,value=TRUE)
[1] "    <title>R: The R Project for Statistical
Computing</title>"
```

4.2 Binární soubory

R umožňuje pomocí funkce save() uložit libovolný objekt do binárního souboru a pomocí funkce load() ho znovu načíst. Pro uložení je třeba zadat název proměnné, ve které je objekt uložen, a také parametr **file** pro soubor, do kterého se data uloží.

Soubor může mít libovolnou koncovku, popřípadě být bez ní, ovšem pokud má koncovku *.RData, systém soubor rozezná a bude ho automaticky spouštět v programu R.

```
> barvy
    nazev     rgb
1 červená #ff0000
2   modrá #0000ff
3  zelená #00ff00
> save(barvy,file="rgb_barvy.RData")
```

Pro opětovné načtení proměnné je třeba zavolat funkci load(). Parametrem funkce je uložený soubor. Funkce ovšem nevrací uložený objekt, nýbrž název uloženého objektu. Nahrávaný objekt se načte do původní proměnné, ze které byl uložen. To může znamenat ztrátu informací, pokud byla do původní proměnné uložena nová data, neboť budou přepsána načítanými daty.

```
> barvy <-
data.frame(nazev=c("fialova","žlutá","tyrkysová"),rgb=c("#ff0
0ff","#ffff00","#00ffff"))
> barvy
      nazev     rgb
1   fialova #ff00ff
2     žlutá #ffff00
3 tyrkysová #00ffff
> stare_barvy <- load("rgb_barvy.RData")
> stare_barvy
[1] "barvy"
> barvy
    nazev     rgb
1 červená #ff0000
2   modrá #0000ff
3  zelená #00ff00
```

Lze také ukládat více objektů do stejného souboru. Jednotlivé objekty je třeba zapsat za sebou jako parametry funkce. Při načtení souboru se načtou všechny objekty do svých proměnných.

```
> save(objekt1,objekt2,objekt3,file="objekty.RData")
```

Ukládaný soubor lze také komprimovat. Metodu komprimace určuje parametr **compress**, a může nabývat 3 hodnot: „gzip", „bzip2“ nebo „xz“. Pokud je hodnota tohoto parametru TRUE, funkce využije metodu gzip. Navíc lze volitelně specifikovat i úroveň komprese pomocí parametru **compress_level**.

Pro ukládání všech objektů existuje funkce save.image(). Tato funkce uloží do souboru .RData momentální stav všech objektů. Tento stav se načte při příštím spuštění R.

Pokud není žádoucí, aby se objekty ukládaly a načítaly i se jménem proměnné, lze místo funkcí save() a load() využít funkce saveRDS() a readRDS(). Tyto funkce ukládají a načítají pouze jeden objekt do jednoho souboru, ovšem tento objekt lze později načíst do libovolné proměnné. Pokud tedy byla původní proměnná změněna, není třeba se bát, že bude přepsána.

```
> barvy
    nazev     rgb
1 červená #ff0000
2   modrá #0000ff
3  zelená #00ff00
> saveRDS(barvy,file="rgb_barvy.RData")
> barvy <-
data.frame(nazev=c("fialova","Žlutá","tyrkysová"),rgb=c("#ff0
0ff","#ffff00","#00ffff"))
> barvy
      nazev     rgb
1   fialova #ff00ff
2     Žlutá #ffff00
3 tyrkysová #00ffff
> stare_barvy <- readRDS("rgb_barvy.RData")
> stare_barvy
    nazev     rgb
1 červená #ff0000
2   modrá #0000ff
3  zelená #00ff00
> barvy
      nazev     rgb
1   fialova #ff00ff
2     Žlutá #ffff00
3 tyrkysová #00ffff
```

Pro statistické údaje se také často používají soubory programu Microsoft Excel. R neumí v základu s těmito soubory pracovat, ovšem existuje pro ně balíček xlsx. Tento balíček je nejdříve třeba nainstalovat a také načíst.

```
> install.packages("xlsx")
> library(xlsx)
```

Pro import xlsx souborů se používá funkce read.xlsx(). Soubory xlsx mohou obsahovat více listů, proto je kromě samotného souboru potřeba specifikovat i číslo listu.

```
> read.xlsx("predmety.xlsx",1)
  zkratka                    nazev
1     ZT2 ZnalostnĂ- technologie 2
2    DIMA   DiskrĂŞtnĂ- matematika
3   PGRF2 PoÄŤĂ-taÄŤovĂ˘ grafika 2
4    DBS2  DatabĂ˘zovĂŞ systĂŞmu 2
```

Některé znaky se ovšem nezobrazují správně. Toto je způsobeno špatným použitím kódovaní. Pro určení kódování souboru slouží parametr **encoding**. Pro soubory Excel je třeba nastavit kódování na „UTF-8.

```
> read.xlsx("predmety.xlsx",1,encoding="UTF-8")
  zkratka                    nazev
1     ZT2 Znalostní technologie 2
2    DIMA    Diskrétní matematika
3   PGRF2    Počítačová grafika 2
4    DBS2    Databázové systémy 2
```

Místo čísla listu lze také předat název listu jako parametr **sheetName**.

```
>
read.xlsx("predmety.xlsx",sheetName="predmety",encoding="UTF-
8")
  zkratka                    nazev
1     ZT2 Znalostní technologie 2
2    DIMA    Diskrétní matematika
3   PGRF2    Počítačová grafika 2
4    DBS2    Databázové systémy 2
```

Pokud je třeba zobrazit pouze určité řádky, je možné využít parametru **rowIndex**. Hodnotou by měl být vektor čísel řádků, které se mají načíst.

```
> read.xlsx("predmety.xlsx",1,encoding="UTF-8",
rowIndex=c(2,4))
    ZT2 Znalostní.technologie.2
1 PGRF2    Počítačová grafika 2
```

V tomto případě slouží chybně druhý řádek jako jména sloupců. To protože byl první řádek z vybraných. Funkce read.xlsx má stejně jako funkce read.table() parametr **header**, ovšem zde má výchozí hodnotu TRUE. Pro správné načtení vybraných řádků je tedy třeba buď přidat první řádek, nebo předat parametru **header** hodnotu FALSE.

```
> read.xlsx("predmety.xlsx",1,encoding="UTF-8",
rowIndex=c(2,4),header=FALSE)
     X1                         X2
1   ZT2 Znalostní technologie 2
2 PGRF2    Počítačová grafika 2
```

Pokud je třeba načíst jistý rozsah řádků, je možné využít parametr **startRow** pro první řádek a **endRow** pro poslední řádek, který se má načíst. Parametr **rowIndex** však nesmí být zadán, neboť má před těmito parametry přednost.

```
> read.xlsx("predmety.xlsx",1,encoding="UTF-
8",startRow=3,endRow=5,header=FALSE)
     X1                   X2
1  DIMA Diskrétní matematika
2 PGRF2 Počítačová grafika 2
3  DBS2 Databázové systémy 2
```

Podobně lze vybrat pouze určité sloupce pro import. Parametrem je **colIndex** a stejně jako rowIndex přijímá vektor čísel.

```
> read.xlsx("predmety.xlsx",1,encoding="UTF-8",colIndex=2)
                    nazev
1 Znalostní technologie 2
2    Diskrétní matematika
3    Počítačová grafika 2
4    Databázové systémy 2
```

Soubory xlsx také mohou obsahovat vzorce pro různé výpočty. Při importu do R jsou načteny již vypočítané hodnoty. Pokud je třeba načíst vzorce jako text, lze předat parametru **keepFormulas** hodnotu TRUE. Pokud tedy budou do souboru přidány kredity a také výpočet součtu kreditů daných předmětů, bude výsledná datová tabulka vypadat takto:

```
> read.xlsx("predmety.xlsx",1,encoding="UTF-8")
  zkratka                    nazev kredity
1     ZT2 Znalostní technologie 2       5
2    DIMA    Diskrétní matematika       5
3   PGRF2    Počítačová grafika 2       6
4    DBS2    Databázové systémy 2       5
5    <NA>                 celkově      21
```

S využitím parametru **keepFormulas** je možné vidět původní vzorec pro výpočet celkových kreditů.

```
> read.xlsx("predmety.xlsx",1,encoding="UTF-
8",keepFormulas=TRUE)
  zkratka                    nazev    kredity
1     ZT2 Znalostní technologie 2          5
2    DIMA    Diskrétní matematika          5
3   PGRF2    Počítačová grafika 2          6
4    DBS2    Databázové systémy 2          5
5    <NA>                 celkově SUM(C2:C5)
```

Data se do souboru xlsx zapisují pomocí funkce write.xlsx(). Prvním parametrem je datová tabulka, druhým je soubor, do kterého se mají data zapsat.

```
> obory
  zkratka                   nazev
1      ai Aplikovaná informatika
2      im  Informační management
3      fm    Finanční management
> write.xlsx(obory,"obory.xlsx")
```

Pokud není zadaný název listu, jsou data uložena do listu „Sheet1“. Pokud soubor obsahuje jiné listy, jsou ve výchozím nastavení vymazány.

```
> read.xlsx("obory.xlsx",sheetName="Sheet1",encoding="UTF-8")
  NA. zkratka                   nazev
1   1      ai Aplikovaná informatika
```

```
2   2      im  Informační management
3   3      fm    Finanční management
```

V tomto případě je však v souboru navíc první sloupec. Stalo se tak, protože původní datová tabulka byla uložena i se jmény řádků, které jsou ve výchozím nastavení čísla řádků. Toto chování lze vypnou předáním hodnoty FALSE parametru **row.names**. Obdobně lze zamezit ukládání jmen sloupců pomocí parametru **col.names**.

```
> write.xlsx(obory,"obory.xlsx",row.names=FALSE)
> read.xlsx("obory.xlsx",sheetName="Sheet1",encoding="UTF-8")
  zkratka                  nazev
1      ai Aplikovaná informatika
2      im  Informační management
3      fm    Finanční management
```

Jméno listu, do kterého se zapisují data, lze určit parametrem **sheetName**. Pro zapsání dat do nového listu v již existujícím souboru je však nutné předat hodnotu TRUE parametru **append**. To způsobí, že data jsou do souboru přidána, a soubor není celý přepsán.

```
> write.xlsx(obory,"predmety.xlsx",row.names=FALSE,
sheetName="obory",append=TRUE)
>
read.xlsx("predmety.xlsx",sheetName="predmety",encoding="UTF-
8")
  zkratka                    nazev
1     ZT2 Znalostní technologie 2
2    DIMA    Diskrétní matematika
3   PGRF2    Počítačová grafika 2
4    DBS2    Databázové systémy 2
> read.xlsx("predmety.xlsx",sheetName="obory",encoding="UTF-
8")
  zkratka                  nazev
1      ai Aplikovaná informatika
2      im  Informační management
3      fm    Finanční management
```

Jak je vidět v ukázce, oba listy jsou stále v souboru. Balíček xlsx také umožňuje manipulaci s listy uvnitř souboru. Tyto funkce však neoperují přímo na souboru, je třeba nahrát soubor jako objekt do paměti funkcí loadWorkbook(). Parametrem je pouze daný soubor.

```
> soubor_predmety <- loadWorkbook("predmety.xlsx")
```

Z tohoto objektu lze zjistit, které listy jsou v souboru pomocí funkce getSheets(). Tato funkce vrací seznam. Jednotlivé prvky seznamu jsou však reference na Java objekty, neboť tato knihovna využívá pro funkci Javu.

```
> getSheets(soubor_predmety)
$predmety
[1] "Java-Object{Name: /xl/worksheets/sheet1.xml - Content
Type: application/vnd.openxmlformats-
officedocument.spreadsheetml.worksheet+xml}"

$obory
[1] "Java-Object{Name: /xl/worksheets/sheet2.xml - Content
Type: application/vnd.openxmlformats-
officedocument.spreadsheetml.worksheet+xml}"
```

Názvy listů jsou použity jako jména prků v seznamu. Pro získání pouze názvů listů v souboru lze výsledek vložit do funkce names().

```
> names(getSheets(soubor_predmety))
[1] "predmety" "obory"
```

Nový list lze přidat pomocí funkce createSheet(). Je třeba funkci pouze předat objekt xlsx souboru a název listu. Funkce pro přidání nového listu vrací jeho Java objekt.

```
> createSheet(soubor_predmety,"studenti")
[1] "Java-Object{Name: /xl/worksheets/sheet3.xml - Content
Type: application/vnd.openxmlformats-
officedocument.spreadsheetml.worksheet+xml}"
> names(getSheets(soubor_predmety))
[1] "predmety" "obory"    "studenti"
```

Obdobně lze list smazat funkcí removeSheet().

```
> removeSheet(soubor_predmety,"studenti")
> names(getSheets(soubor_predmety))
[1] "predmety" "obory"
```

Nakonec je třeba objekt znovu uložit do souboru. K tomu slouží funkce saveWorkbook(). Parametry funkce jsou samotný objekt a soubor, do kterého se má uložit.

```
> saveWorkbook(soubor_predmety,"predmety.xlsx")
```

4.3 Relační databáze

Pro práci s relačními databázemi v R existují 2 možnosti. Tou první je ODBC (zkratka pro Open DataBase Connecivity). Je to univerzálně používaný standart pro práci s databází, lze jej tedy použít k jakékoliv databázi. Durhou metodou je balíček DBI, který slouží ke spojení s konkrétní databází. Pokud existuje pro danou databázi balíček DBI, je lepší jej z výkonnostních důvodů použít, pokud však takový balíček není, je třeba využít ODBC. [4]

Pro použití ODBC je nejdříve třeba vytvořit zdroj dat (DSN). Na platformě Windows lze vytvořit pomocí nástroje „Zdroje dat (ODBC)“. Tento nástroj lze nalézt přes Ovládací nástroje -> Nástroje pro správu -> Zdroje dat (ODBC). Při přidávání nových zdrojů dat je třeba zadat ovladač pro databázi, název zdroje, přes který se R bude připojovat, server, jméno databáze, popřípadě uživatelské jméno a heslo. Pokud ovladač k dané databázi není v seznamu, je třeba jej nainstalovat. [4]
Po nastavení DSN je možné se k databázi připojit. Nejdříve je nutné nainstalovat a načíst balíček „RODBC“.

```
> install.packages("RODBC")
> library(RODBC)
```

Také je třeba vytvořit objekt pro spojení k databázi. Lze tak učinit pomocí funkce odbcConnect(), jejímž parametrem je název zdroje.

```
> odbc <- odbcConnect("predmety_mysql");
```

Pomocí funkce sqlQuery() lze položit databázi jakýkoliv SQL dotaz. Prvním parametrem je objekt spojení s databází, druhým je řetězec s libovolným SQL dotazem. Funkce vrací datovou tabulku.

```
> sqlQuery(odbc,"DESC predmety")
    Field         Type Null Key Default           Extra
1      id     int(11)   NO PRI      NA auto_increment
2 zkratka     char(5)  YES          NA
3   nazev varchar(80)  YES          NA
```

Data lze do databáze ukládat pomocí SQL dotazu. Před uvozovky je třeba napsat zpětné lomítko, jinak je bude R interpretovat jako konec textu.

```
> sqlQuery(odbc,"INSERT INTO predmety (zkratka,nazev) VALUES
("UOMO","Úvod do objektového modelování")")
> sqlQuery(odbc,"SELECT * FROM predmety")
  id zkratka                          nazev
1  1    UOMO Úvod do objektového modelování
```

Balíček však také obsahuje funkci sqlSave(), která ukládání dat zjednodušuje. Tato funkce uloží data do vybrané tabulky, nebo vytvoří novou tabulku a do té data uloží. Prvním parametrem funkce je objekt spojení s databází, druhým je datová tabulka a třetím název tabulky. Pokud je třeba data přidat k již existujícím datům, je třeba předat parametru **append** hodnotu TRUE. Funkce se také pokusí uložit číslo řádku do databáze do sloupce „count“. Tomu lze zabránit předáním parametru **rownames** hodnotu FALSE.

```
> predmety
```

```
  id zkratka                    nazev
1  2     ZT2 Znalostní technologie 2
2  3    DIMA    Diskrétní matematika
3  4   PGRF2    Počítačová grafika 2
4  5    DBS2    Databázové systémy 2
>
sqlSave(odbc,predmety,"predmety",append=TRUE,rownames=FALSE)
> sqlQuery(odbc,"SELECT * FROM predmety")
  id zkratka                          nazev
1  1    UOMO Úvod do objektového modelování
2  2     ZT2        Znalostní technologie 2
3  3    DIMA           Diskrétní matematika
4  4   PGRF2           Počítačová grafika 2
5  5    DBS2           Databázové systémy 2
```

Pokud by chyběl parametr **append**, funkce by vrátila chybu, protože tabulka „predmety“ již existuje a funkce v původním nastavení tabulky nepřepisuje.

```
> sqlSave(odbc,predmety,"predmety",rownames=FALSE)
Error in sqlSave(odbc, predmety, "predmety", rownames =
FALSE) :
  table 'predmety' already exists
```

Pokud je třeba tabulku opravdu přepsat, je třeba předat parametru **safer** hodnotu FALSE.

Pokud se při dotazu na databázi vyskytne chyba, funkce vrátí její text. Je možné toto potlačit předáním parametru **errors** hodnotu FALSE.

```
> sqlQuery(odbc,"SELECT id,zkratka,jmeno FROM predmety")
[1] "42S22 1054 [MySQL][ODBC 5.3(w) Driver][mysqld-
5.6.12]Unknown column 'jmeno' in 'field list'"
[2] "[RODBC] ERROR: Could not SQLExecDirect 'SELECT
id,zkratka,jmeno FROM predmety'"
> sqlQuery(odbc,"SELECT id,zkratka,jmeno FROM
predmety",errors=FALSE)
```

K textu chyby se lze dostat pomocí funkce odbcGetErrMsg(). Jejím parametrem je objekt spojení s databází.

```
> odbcGetErrMsg(odbc)
[1] "42S22 1054 [MySQL][ODBC 5.3(w) Driver][mysqld-
5.6.12]Unknown column 'jmeno' in 'field list'"
[2] "[RODBC] ERROR: Could not SQLExecDirect 'SELECT
id,zkratka,jmeno FROM predmety'"
```

Pokud tabulka obsahuje velké množství řádků, může být vhodnější načítat data po částech. K tomu slouží parametr **max**. Hodnotou je počet řádků, které se načtou najednou. Všechna zbylá data lze získat pomocí funkce sqlGetResults() s parametrem spojení s databází.

```
> sqlQuery(odbc,"SELECT * FROM predmety",max=2)
  id zkratka                          nazev
```

```
1  1    UOMO Úvod do objektového modelování
2  2     ZT2        Znalostní technologie 2
> sqlGetResults(odbc)
  id zkratka                nazev
1  3    DIMA Diskrétní matematika
2  4   PGRF2 Počítačová grafika 2
3  5    DBS2 Databázové systémy 2
```

Parametr **max** lze použít i na funkci sqlGetResults(). Takto lze získávat data postupně.

```
> sqlQuery(odbc,"SELECT * FROM predmety",max=2)
  id zkratka                           nazev
1  1    UOMO Úvod do objektového modelování
2  2     ZT2        Znalostní technologie 2
> sqlGetResults(odbc,max=2)
  id zkratka                nazev
1  3    DIMA Diskrétní matematika
2  4   PGRF2 Počítačová grafika 2
> sqlGetResults(odbc,max=2)
  id zkratka                nazev
1  5    DBS2 Databázové systémy 2
```

Pokud je třeba pouze získat z databáze celou tabulku a není třeba žádného filtrování pomocí SQL, je možné využít funkci sqlFetch(), která tuto činnost zjednodušuje. První parametrem je objekt spojení s databází a druhým název tabulky.

```
> sqlFetch(odbc,"predmety")
  id zkratka                           nazev
1  1    UOMO Úvod do objektového modelování
2  2     ZT2        Znalostní technologie 2
3  3    DIMA           Diskrétní matematika
4  4   PGRF2           Počítačová grafika 2
5  5    DBS2           Databázové systémy 2
```

Tato funkce má, stejně jako sqlQuery(), parametr **max**, kterým lze limitovat počet řádků. Zbytek řádků lze získat pomocí funkce sqlFetchMore().

```
> sqlFetch(odbc,"predmety",max=2)
  id zkratka                           nazev
1  1    UOMO Úvod do objektového modelování
2  2     ZT2        Znalostní technologie 2
> sqlFetchMore(odbc)
  id zkratka                nazev
1  3    DIMA Diskrétní matematika
2  4   PGRF2 Počítačová grafika 2
3  5    DBS2 Databázové systémy 2
```

Balíček „RODBC“ také obsahuje funkci sqlUpdate(). Tato funkce slouží k upravení již existujících řádků v tabulce. Jejími parametry jsou objekt spojení s databází, datová tabulka se změnami a název tabulky. Je třeba, aby funkce dokázala zjistit,

které řádky má změnit. Nejlepší způsob je zachovat sloupec s primárním klíčem, podle kterého lze řádky jednoznačně identifikovat.

```
> sqlFetch(odbc,"predmety")
  id zkratka                    nazev
1  1     hgf          Ajhfjskb lksdhf
2  2     ZT2 Znalostní technologie 2
3  3    DIMA    Diskrétní matematika
4  4     drf             Ksghjk lokjf
5  5    DBS2    Databázové systémy 2
> opraveni_predmetu
  id zkratka                          nazev
1  1    UOMO Úvod do objektového modelování
2  4   PGRF2           Počítačová grafika 2
> sqlUpdate(odbc,opraveni_predmetu,"predmety")
> sqlFetch(odbc,"predmety")
  id zkratka                          nazev
1  1    UOMO Úvod do objektového modelování
2  2     ZT2        Znalostní technologie 2
3  3    DIMA           Diskrétní matematika
4  4   PGRF2           Počítačová grafika 2
5  5    DBS2           Databázové systémy 2
```

Po ukončení práce s databází je vhodné uzavřít spojení pomocí funkce close().

```
> close(odbc)
```

Druhou metodou pro práci s databází je balíček DBI konkrétní databáze. V práci budou ukázány příklady s databází MySQL. Pro práci s MySQL je nejdříve potřeba nainstalovat a načíst balíček „RMySQL“. [4]

```
> install.packages("RMySQL")
> library(RMySQL)
```

Pro práci s databází je třeba získat objekt spojení s databází. Ten lze získat pomocí funkce dbConnect(). Prvním parametrem je ovladač pro databázi. Ten lze získat pomocí funkce dbDriver() a názvu ovladače. Dále je možné zvolit název databáze pomocí parametru **dbname**, server pomocí parametru **host**, uživatelské jméno pomocí parametru **user**, a heslo pomocí parametru **password**. Pokud databáze běží na stejném počítači jako R, je možné použít jako server hodnotu „localhost“.

```
> dbi <-
dbConnect(dbDriver("MySQL"),host="localhost",dbname="r",user=
"root",password="")
```

Databázi lze položit libovolný SQL dotaz pomocí funkce dbGetQuery(). Prvním parametrem je objekt spojení s databází a druhým je dotaz.

```
> dbGetQuery(dbi,"SELECT * FROM predmety")
  id zkratka                          nazev
1  1    UOMO Úvod do objektového modelování
```

```
2  2     ZT2        Znalostní technologie 2
3  3    DIMA           Diskrétní matematika
4  4   PGRF2           Počítačová grafika 2
5  5    DBS2           Databázové systémy 2
```

Pokud nemá být výsledek navrácen ihned lze využít funkci dbSendQuery(). Tato funkce pouze pošle databázi SQL dotaz. Výsledek nevrací jako datovou tabulku, ale jako objekt typu DBIResult. Výsledky ve formě datové tabulky lze z toho objektu získat pomocí funkce dbFetch(). Pomocí parametru **n** lze také určit, kolik řádků se má získat.

```
> vysledek <- dbSendQuery(dbi,"SELECT * FROM predmety")
> dbFetch(vysledek,n=2)
  id zkratka                          nazev
1  1    UOMO Úvod do objektového modelování
2  2     ZT2        Znalostní technologie 2
> dbFetch(vysledek,n=2)
  id zkratka                nazev
1  3    DIMA Diskrétní matematika
2  4   PGRF2 Počítačová grafika 2
> dbFetch(vysledek,n=2)
  id zkratka                nazev
1  5    DBS2 Databázové systémy 2
```

Po ukončení práce s výsledkem je nutné tento objekt uzavřít a uvolnit tak zdroje databáze. Výsledek se uzavírá funkcí dbClearResult(). Funkce vrací logickou hodnotu TRUE pokud se podaří výsledek uzavřít, v opačném případě vrací hodnotu FALSE.

```
> dbClearResult(vysledek)
[1] TRUE
```

Balíček „DBI“ také obsahuje funkce pro zjednodušené načítání a ukládání tabulek. Funkce dbWriteTable() slouží k ukládání tabulek. Funkce potřebuje objekt spojení s databází, název tabulky v databázi a datovou tabulku. Pro přidávání do již existující tabulky je třeba předat parametru **append** hodnotu TRUE. Stejně jako v případě ODBC je třeba předat hodnotu FALSE parametru **row.names**, aby se jména řádků neukládala do tabulky.

```
> x
  id zkratka                    nazev
1  6     OA1 Odborný anglický jazyk 1
2  7     OA2 Odborný anglický jazyk 2
> dbWriteTable(dbi,"predmety",x,append=TRUE,row.names=FALSE)
[1] TRUE
```

Funkce dbReadTable() naopak tabulku z databáze načte. Jako parametry slouží objekt spojení s databází a název tabulky.

```
> dbReadTable(dbi,"predmety")
  id zkratka                          nazev
1  1    UOMO Úvod do objektového modelování
2  2     ZT2        Znalostní technologie 2
3  3    DIMA           Diskrétní matematika
4  4   PGRF2           Počítačová grafika 2
5  5    DBS2           Databázové systémy 2
6  6     OA1       Odborný anglický jazyk 1
7  7     OA2       Odborný anglický jazyk 2
```

Po skončení práce s databází je i u této metody vhodné spojení uzavřít. Slouží k tomu funkce dbDisconnect().

```
> dbDisconnect(dbi)
[1] TRUE
```

5 Aplikace

Tato část práce obsahuje příklad využití R k práci s daty. Úkolem, který je v tomto příkladu řešen, je získat data z MySQL databáze o hodnocení nějakého produktu uživateli a také o uživatelích samotných. Dále je úkolem hodnocení roztřídit podle věku a u každého věku určit průměrné hodnocení a tyto data uložit do souboru aplikace Excel. Také je třeba určit celkové průměrné hodnocení, medián a směrodatnou odchylku. Tyto data je třeba uložit do stejného souboru, ovšem do jiného listu.

Tento příklad předpokládá, že databáze obsahuje tabulku „uzivatel“ která obsahuje sloupce „id“ a „vek“ a také tabulku „hodnoceni“, která obsahuje sloupce „hodnota“ a „id_uzivatele“.

První, co je třeba udělat, je nainstalovat a načíst balíček „RMySQL“, pokud ještě načtený není.

```
> install.packages("RMySQL")
> library(RMySQL)
```

Pro získání dat je třeba spojit se s databází a načíst potřebné tabulky.

```
> dbi <-
dbConnect(dbDriver("MySQL"),host="localhost",dbname="r",user=
"root",password="")
> uzivatel <- dbReadTable(dbi,"uzivatel")
> hodnoceni <- dbReadTable(dbi,"hodnoceni")
```

Více již není databáze potřeba, je tedy vhodné ukončit spojení.

```
> dbDisconnect(dbi)
```

Dále je třeba tyto dvě tabulky spojit podle id uživatele. K tomu slouží funkce merge(). Z výsledku jsou však potřeba pouze sloupce „vek“ a „hodnota“, ty lze vybrat pomocí indexace vektorem názvů. První index by měl být prázdný, neboť je třeba vybrat všechny řádky.

```
> spojene <-
merge(uzivatel,hodnoceni,by.x="id",by.y="id_uzivatele")[,c("v
ek","hodnota")]
```

Spojení tabulek lze také docílit pomocí SQL dotazu obsahujícího příkaz JOIN, ovšem protože se tato práce zabývá jazykem R, jsou tabulky spojeny přímo v tomto jazyce.

Dalším krokem je roztřídění dat podle věku pomocí funkce aggregate(). Pro výpočet průměru se používá funkce mean().

```
> vysledek <-
aggregate(spojene$hodnota,list(vek=spojene$vek),mean)
```

Dále je třeba uložit tyto data do souboru aplikace Excel. K tomu je třeba načíst balíček xlsx, pokud již není načten.

```
> library(xlsx)
```

Pro zapsání dat slouží funkce write.xlsx(). Název souboru je „hodnoceni.xlsx" a název listu „podle_veku".

```
>
write.xlsx(vysledek,"hodnoceni.xlsx",sheetName="podle_veku",r
ow.names=FALSE)
```

Následně se má vypočítat celkový průměr. Ten lze zjistit pomocí funkce mean(). Jejím parametrem by měl být vektor hodnot, jejichž průměr je třeba spočítat. Ten lze získat jako indexace sloupce hodnota z proměnné hodnocení.

```
> prumer <- mean(hodnoceni$hodnota)
```

Medián se spočítá obdobně, pouze místo funkce mean() je pro výpočet použita funkce median().

```
> median <- median(hodnoceni$hodnota)
```

Směrodatná odchylka se spočítá pomocí funkce sd(). Tato funkce se chová obdobně k předchozím dvěma funkcím.

```
> smerodatna_odchylka <- sd(hodnoceni$hodnota)
```

Nyní je třeba uložit tyto proměnné do souboru. Nejdříve je třeba vytvořit datovou tabulku s těmito proměnnými. V prvním sloupci budou názvy proměnných, a ve druhém jednotlivé hodnoty.

```
> vysledek2 <-
data.frame(nazev=c("průměr","medián","směrodatná
odchylka"),hodnota=c(prumer,median,smerodatna_odchylka))
```

Nyní je třeba uložit tuto datovou tabulku do stejného souboru, jako předchozí data, ovšem tentokrát do listu „celkove". Je třeba předat parametru **append** hodnotu TRUE, jinak by se původní soubor přepsal. Stejně tak je vhodné předat parametru **col.names** hodnotu FALSE, aby se neukládaly názvy sloupců, neboť v tomto případě jednotlivé řádky spolu přímo nesouvisí a data nejsou opravdovou

tabulkou, tabulka je použita pouze pro snadné zapsání do souboru. Z toho důvodu by názvy sloupců byli zbytečné.

```
>
write.xlsx(vysledek2,"hodnoceni.xlsx",sheetName="celkove",row
.names=FALSE,col.names=FALSE,append=TRUE)
```

Nyní je ukázkový příklad hotový. Tento příklad ukázal práci s databází, manipulaci s daty v datové tabulce a také import do souboru aplikace Excel do více různých listů.

6 Závěr

Cílem této práce bylo popsat datové struktury v jazyce R. V první části se práce věnuje vektoru. Nejprve obsahuje základní popis struktury, dále se věnuje atributům, základním operacím a vytváření dané datové struktury, nejdříve spojováním hodnot a následně sekvencemi. Nakonec se práce věnuje způsobům indexování vektoru. V druhé podkapitole se práce věnuje poli. Stejně jako u vektoru práce popisuj základní charakteristiky, atributy a způsoby indexování, poslední část se věnuje matici, speciálnímu druhu pole. V následujících částech práce obdobně popisuje seznam, datovou tabulku a faktor. Kapitola 4 se věnuje importu a exportu dat. Nejdříve jsou probrány textové soubory v různých formátech. Následující část se věnuje binárním souborům, se kterými nativně pracuje R a také popisuje balíček xlsx, který umožňuje pracovat se soubory programu Microsoft Excel. Práce popisuje načítání a ukládání dat do souboru a jednotlivých listů. V podkapitole relační databáze práce popisuje dva způsoby manipulace s relační databází. Na konci práce je příklad využití některých funkcí.

Při vypracování této bakalářské práce byla v každé kapitole nejdříve nastíněna daná problematika, následně byly popsány funkce, které s danou problematikou souvisí. Po popisu funkcí obsahuje práce také příklad využití dané funkce.

7 Seznam použité literatury

[1] VENABLES, W. N., D. M. SMITH a R CORE TEAM. An Introduction to R. *The R Project for Statistical Computing* [online]. 2014-07-10 [cit. 2014-08-18]. Dostupné z: http://cran.r-project.org/doc/manuals/R-intro.pdf

[2] R CORE TEAM. R Language Definition. *The R Project for Statistical Computing* [online]. 2014-07-10 [cit. 2014-08-19]. Dostupné z: http://cran.r-project.org/doc/manuals/r-release/R-lang.pdf

[3] An Introduction to the Fundamentals & Functionality of the R Programming Language: art II: The Nuts & Bolt. *Department of Biostatistics: Vanderbilt University School of Medicine* [online]. 2008-12-22 [cit. 2014-08-21]. Dostupné z: http://biostat.mc.vanderbilt.edu/wiki/pub/Main/TheresaScott/Scott.IntroToR.II.pdf

[4] SPECTOR, Phil. *Data Manipulation with R*. Berkeley, California: Springer, 2008. ISBN 978-0-387-74730-9

8 Seznam použitých funkcí

9 Zadání práce

FIM UHK
UNIVERZITA HRADEC KRÁLOVÉ
Fakulta informatiky a managementu
Rokitanského 62, 500 03 Hradec Králové, tel: 493 331 111, fax: 493 332 235

Zadání k závěrečné práci

Jméno a příjmení studenta: **Jan Knejp**

Obor studia: Aplikovaná informatika
Jméno a příjmení vedoucího práce: **Hana Skalská**

Název práce:
Datové struktury v jazyce R

Název práce v AJ:
Data structures in R language

Podtitul práce:

Podtitul práce v AJ:

Cíl práce: Cílem práce je popsat fungování datových struktur v jazyce R a způsoby, jakými s nimi lze manipulovat.

Osnova práce:
1.Úvod
2.Principy jazyka R
3.Popis základních datových struktur
4.Import a export dat
5.Aplikace
6.Závěr

Projednáno dne: 7. 10. 2014

Podpis studenta

Podpis vedoucího práce