Przetwarzanie napisów w R przy użyciu pakietu `stringi` i bibliotek `ICU`

Marek Gągolewski (IBS PAN)
gagolewski.rexamine.com

Spotkanie Entuzjastów R
20 marca 2014 r.

Hadley Wickham (`ggplot2`, `plyr`, `reshape`, `testthat`, `stringr`):

Strings are ***not*** *glamorous, high-profile components of R, but they do play a big role in many data cleaning and preparations tasks. R provides a solid set of string operations, but because they have grown organically over time, they can be inconsistent and a little hard to learn.*

Additionally, they lag behind the string operations in other programming languages, so that some things that are easy to do in languages like Ruby or Python [and Perl] are rather hard to do in R.

[`stringr/README`]

Gągolewski M.,
Programowanie w języku R.
Analiza danych, obliczenia, symulacje,
WN PWN, 2014.

http://rksiazka.rexamine.com

Rozdz. 9: *Przetwarzanie napisów*

Problemy z R

```
> nchar(c("ala", "ość", "", NA))
```

```
## [1] 3 3 0 2
```

(niepoprawna obsługa braków danych)

```
> paste(letters[1:3], NA)
```

```
## [1] "a NA" "b NA" "c NA"
```

(wg ?paste to jest działanie celowe)

```
> grepl(c("[a-z]+", "[0-9]+"), c("abc", "123"))
```

```
## Warning: argument `pattern` has length > 1
## and only the first element will be used
## [1]  TRUE FALSE
```

(brak wektoryzacji)

```
> regexec("([a-z]+)=([a-z]+)", "abc obiad=lody def")[[1]]
```

```
## [1]  5  5 11
## attr(,"match.length")
## [1] 10  5  4
```

(nieczytelny / niewygodny w obsłudze wynik)

Problemy z R

stringr: rozwiązanie?

```
> library("stringr")
> str_length(c("ala", "ość", "", NA))
```

```
## [1]  3  3  0 NA
```

(OK)

```
> str_join(letters[1:3], NA)
```

```
## [1] "aNA" "bNA" "cNA"
```

(patrz uwaga z ?paste)

```
> str_detect(c("abc", "123"), c("[a-z]+", "[0-9]+"))
```

```
## [1] TRUE TRUE
```

(poprawnie zwektoryzowane)

```
> str_match_all("abc obiad=lody def", "([a-z]+)=([a-z]+)")
```

```
## [[1]]
##      [,1]          [,2]    [,3]
## [1,] "obiad=lody" "obiad" "lody"
```

(łatwe w obsłudze)

Problemy z R

stringr: rozwiązanie?

funkcja stringr	**funkcja bazowa**
str_length()	nchar()
str_join()	paste()
str_count()	regexpr()
str_detect()	grepl()
str_extract()	regexpr()
str_extract_all()	gregexpr()
str_locate()	regexpr()
str_locate_all()	gregexpr()
str_replace()	sub()
str_replace_all()	gsub()
str_split()	gregexpr(), strsplit()
...	...

Problemy z R

. . . leżą gdzieś głębiej

```
> toupper("fußball")
```

```
## [1] "FUßBALL"
```

(FUSSBALL)

```
> "hladný" < "chladný"# jęz. słowacki
```

```
## [1] FALSE
```

(powinno być: TRUE)

```
> oldlocale <- Sys.getlocale("LC_COLLATE")
> Sys.setlocale("LC_COLLATE", "sk_SK")  # Linux, ale nie Windows!
> "hladný" < "chladný"
> Sys.setlocale("LC_COLLATE", oldlocale)
```

```
## [1] TRUE
```

(skomplikowane + nieprzenośne rozwiązanie)

POSIX: BCP 47 (np. "pl_PL.UTF-8" – ISO 639 & ISO 3166-1 alpha-2)
Windows: LCID (np. "Polish_Poland.1250")
oraz RFC4646 (od Win Vista, np. "en-US")

Problemy z R

. . . leżą gdzieś głębiej

Norweski i duński: a $<$ z $<$ aa (å)

```
> Sys.setlocale("LC_COLLATE", "no_NO") # Ubuntu
[1] ""
Komunikat ostrzegawczy:
In Sys.setlocale("LC_COLLATE", "no_NO") :
  Żądania raportów OS aby ustawić lokalizację na "no_NO"
  nie mogą zostać wykonane
> sort(c("a", "z", "aa"))
[1] "a"  "aa" "z"
@
```

(powinno być: "a" "z" "aa")

Normalizacja: "ą"."ą" "ą" "a̧" "ą" "ą"

```
> "\u0105" == "\u0061\u0328" # a with ogonek, a and ogonek
```

```
## [1] FALSE
```

(raczej powinno być TRUE (jeśli przetwarzamy tekst))

Problemy z R

. . . leżą gdzieś głębiej

BTW: jak wczytać "March 20, 2014" jako obiekt klasy „data (i czas)”?

```
> strptime("March 20, 2014", "%B %d, %Y")

## [1] NA
```

(March – na pewno nie w pl_PL)

```
> oldlocale <- Sys.getlocale("LC_TIME")
> invisible(Sys.setlocale("LC_TIME", "C"))  # ew. en_US (Linux)
> strptime("March 20, 2014", "%B %d, %Y")

## [1] "2014-03-20"
```

(piece of cake)

```
> invisible(Sys.setlocale("LC_TIME", oldlocale))
```

Problemy z R

. . . jeszcze głębiej: polskie literki a wyrażenia regularne w R

Pattern	Locale		
	pl_PL.UTF-8 (Linux)	pl_PL.iso-8859-2 (Linux)	Polish_Poland.1250 (Windows)
	ERE-Native		
`[[:alpha:]]`	ĄĆĘŁŃÓŚŹŻ ąćęłńóśźż ☺	ĆĘŃÓćęńó ☹	
`[[:digit:]]`	☺		ął ☹
`[[:punct:]]`	☺	ĄŁŚŹŻąłśźż ☹	ĄŁŻąłż ☹

Pattern	Locale		
	pl_PL.UTF-8 (Linux)	pl_PL.iso-8859-2 (Linux)	Polish_Poland.1250 (Windows)
	PCRE-Native		
`[[:alpha:]]`	☹	ĄĆĘŁŃÓŚŹŻ ąćęłńóśźż ☺	
`[[:digit:]]`	☺		
`[[:punct:]]`	☺		

Problemy z R

. . . jeszcze głębiej: polskie literki a wyrażenia regularne w R

	Locale		
Pattern	pl_PL.UTF-8 (Linux)	pl_PL.iso-8859-2 (Linux)	Polish_Poland.1250 (Windows)
	ERE-UTF-8 NORMALIZED		
`[[:alpha:]]`	☺	Óó ☹	
`[[:digit:]]`	☺		
`[[:punct:]]`	☺		

	Locale		
Pattern	pl_PL.UTF-8 (Linux)	pl_PL.iso-8859-2 (Linux)	Polish_Poland.1250 (Windows)
	PCRE-UTF-8 NORMALIZED		
`\p{L}`	ĄĆĘŁŃÓŚŹŻ ąćęłńóśźż ☺		
`\p{N}`	☺		
`\p{S}|\p{P}`	☺		

(np. `str_detect(enc2utf8(text), perl(enc2utf8(regex)))`)

Problemy z R

. . . dno

1. **Orientacja zachodnioeuropejska**
   Z początku R obsługiwał tylko kodowanie ISO-8859-1 (zob. Dodatek 1)
2. **„Przyrostowy" rozwój R**
   Nieprzemyślane w przeszłości rozwiązania pokutują do dziś.
3. **Zlepek różnych bibliotek**
   (zob. *R Installation and Administration*)

   Czym chata bogata:
   - **Kodowania:** GNU libiconv, iconv, win_iconv
   - **Regex:** grep (do R 2.11), TRE (od R 2.12), PCRE
   - **Collation:** glibc, ICU (!), POSIX

Reprezentacja napisów

- Napis w języku C to:
  ciąg „małych liczb całkowitych" zakończony 0 (`char*`)
  `char` == 1 bajt $\in \{0, 1, \ldots, 255\}$.
- Standardy kodowania znaków
  słownik – jaka liczba (ciąg liczb) odpowiada danemu znakowi.
- Przetwarzanie napisów == przetwarzanie ciągów liczbowych
  (zgodnie z pewnymi regułami – zob. dalej)
  - W kodowaniach 8-bitowych każdy znak to *jeden bajt*
    (tylko 255 możliwości!)
  - Standard ASCII określa kody 0–127
    (*American Standard Code for Information Interchange*)

Reprezentacja napisów
Kodowanie ASCII

Kod	Znaczenie	Kod	Znaczenie	Kod	Znak	Kod	Znak
0	Null (\0)	16		32	Spacja	48	0
1		17		33	!	49	1
2		18		34	”	50	2
3		19		35	#	51	3
4		20		36	$	52	4
5		21		37	%	53	5
6		22		38	&	54	6
7	*BEL (\a)*	23		39	’	55	7
8	BSP (\b)	24		40	(	56	8
9	TAB (\t)	25		41	)	57	9
10	LF (\n)	26	SUBST	42	*	58	:
11		27		43	+	59	;
12		28		44	,	60	<
13	CR (\r)	29		45	-	61	=
14		30		46	.	62	>
15		31		47	/	63	?

Reprezentacja napisów
Kodowanie ASCII

Kod	Znak	Kod	Znak	Kod	Znak	Kod	Znak
64	@	80	P	96	‘	112	p
65	A	81	Q	97	a	113	q
66	B	82	R	98	b	114	r
67	C	83	S	99	c	115	s
68	D	84	T	100	d	116	t
69	E	85	U	101	e	117	u
70	F	86	V	102	f	118	v
71	G	87	W	103	g	119	w
72	H	88	X	104	h	120	x
73	I	89	Y	105	i	121	y
74	J	90	Z	106	j	122	z
75	K	91	[	107	k	123	{
76	L	92	\	108	l	124	|
77	M	93	]	109	m	125	}
78	N	94	^	110	n	126	∼
79	O	95	_	111	o	127	

Reprezentacja napisów
Kodowania polskich znaków diakrytycznych

Wśród najpopularniejszych 8-bitowych standardów określających *polskie znaki diakrytyczne* (ś, ą, ę – ogonki) znajdziemy:

- ISO-8859-2 (in. ISO Latin-2, Polska Norma PN-93 T-42118),
- Windows-1250 (CP-1250).

Poza przedziałem kodów 0–127, który pokrywa się z ASCII, standardy te nie są niestety ze sobą całkowicie zgodne.

	Ą	Ć	Ę	Ł	Ń	Ó	Ś	Ź	Ż
ISO-8859-2	**161**	198	202	163	209	211	**166**	**172**	175
Windows-1250	**165**	198	202	163	209	211	**140**	**143**	175

	ą	ć	ę	ł	ń	ó	ś	ź	ż
ISO-8859-2	**177**	230	234	179	241	243	**182**	**188**	191
Windows-1250	**185**	230	234	179	241	243	**156**	**159**	191

Rank	Language	Primary Country	Total Countries	Speakers (millions)
1	Chinese	China	33	1 197
2	Spanish	Spain	31	406
3	English	UK	101	335
4	Hindi	India	4	260
5	Arabic	Saudi Arabia	59	223
6	Portuguese	Portugal	11	202
7	Bengali	Bangladesh	4	193
8	Russian	Russian Fed.	16	162
9	Japanese	Japan	3	122
10	Javanese	Indonesia	3	84
11	German	Germany	18	84
12	Lahnda	Pakistan	7	83
13	Telugu	India	2	74
14	Marathi	India	1	72
15	Tamil	India	6	69
16	French	France	51	69
17	Vietnamese	Viet Nam	3	68
18	Korean	South Korea	6	66
19	Urdu	Pakistan	6	63
20	Italian	Italy	10	61

http://www.ethnologue.com/statistics/size

Reprezentacja napisów
Kodowanie UTF-8

For historical reasons, international text is often encoded using a language or country dependent character encoding. With the advent of the internet and the frequent exchange of text across countries – even the viewing of a web page from a foreign country is a "text exchange" in this context – conversions between these encodings have become important. They have also become a problem, because many characters which are present in one encoding are absent in many other encodings.
To solve this mess, the Unicode encoding has been created. It is a super-encoding of all others and is therefore the default encoding for new text formats like XML.

https://www.gnu.org/software/libiconv/

Reprezentacja napisów
Kodowanie UTF-8

1. Społeczność internetu tworzą użytkownicy prawie wszystkich języków świata, por. FR (é, ç), DE (ö, ß), ES (ñ), SK (š, ǒ), ale też „galimatias” w greckim, cyrylicy, arabskim, hebrajskim bądź tzw. CJK (chińskim-japońskim-koreańskim)...
2. Standard **Unicode**, czyli rodzina kodowań uniwersalnych,
3. **UTF-8** (ang. *Universal character set Transformation Format – 8-bit*) – sposób reprezentacji znaków za pomocą liczb całkowitych zapisywanych za pomocą *zmiennej liczby bajtów*. (MBCS == multibyte character sets)
4. Pierwszych 127 kodów pokrywa się z ASCII.
5. Najbardziej popularny w Internecie.
6. Dostęp do i-tego znaku (code point) $O(n)$; napisy mogą być nieznormalizowane itd.

Reprezentacja napisów
Wieża Babel

```
Tekst w CP-1250 interpretowany przy użyciu ISO-8859-2.

-- We tš zółtš kredkš œlad mi tu narysuj, Muńku!
-- Cóż, robię co mogę, żółć mnie już zalewa!
```

```
Tekst w ISO-8859-2 interpretowany przy użyciu CP-1250.

-- WeĽ t± zółt± kredk± ¶lad mi tu narysuj, Muńku!
-- Cóż, robię co mogę, żółć mnie już zalewa!
```

Powyższe teksty w CP-1250 i ISO-8859-2 – niepoprawne w UTF-8.

```
Tekst w UTF-8 interpretowany przy uĽżyciu ISO-8859-2.

-- WeĹş tÄ… ĹźĂłĹ‚tÄ… kredkÄ… Ĺ›lad mi tu narysuj, MuĹ„ku!
-- CĂłĹźÂ, robiÄ™ co mogÄ™, ĹźĂłĹ‚Ä‡ mnie juĹź zalewa!
```

```
Tekst w UTF-8 interpretowany przy uĽĽyciu CP-1250.

-- WeĹş tÄ… ĹĽĂłĹ‚tÄ… kredkÄ… Ĺ›lad mi tu narysuj, MuĹ„ku!
-- CĂłĹĽÂ, robiÄ™ co mogÄ™, ĹĽĂłĹ‚Ä‡ mnie juĹĽ zalewa!
```

Operacje na napisach
Wyzwania

Raz jeszcze:

Przetwarzanie napisów == przetwarzanie ciągów liczbowych

(zgodnie z pewnymi regułami)

Owe reguły są właściwe różnym językom/dialektom itd.

Potrzebna jest wyczerpująca, jednolita, przenośna i wiarogodna *baza danych* reguł (R do takowej dostępu nie daje).

Np. porównywanie napisów (sortowanie, wyszukiwanie – *collation*):

- Kody ASCII: a $<$ b $<$ ... $<$ z, A $<$ B $<$ ... $<$ Z, Z $<$ a
- Reguły dla różnych języków: hladný $<$ chladný
- "2" $<$ "123"
- Waluty: \$1.23 a 1,23zł

Biblioteki ICU

Informacje ogólne

International Components for Unicode (ICU)

ICU was originally developed by the **Taligent** company.
The Taligent team later became the Unicode group at the **IBM®** Globalization Center of Competency in Cupertino.
`http://site.icu-project.org/`:

- ICU is a mature, widely used set of C/C++ and Java libraries providing Unicode and Globalization support for software applications.
- ICU is widely portable and gives applications the same results on all platforms and between C/C++ and Java software.
- ICU is released under a nonrestrictive open source license that is suitable for use with both commercial software and with other open source or free software.

Biblioteki ICU
Usługi

- **Code Page Conversion**
  Convert text data to or from Unicode and nearly any other character set or encoding.
  ICU's conversion tables are based on charset data collected by IBM over the course of many decades, and is the most complete available anywhere.
- **Collation**
  Compare strings according to the conventions and standards of a particular language, region or country. ICU's collation is based on the Unicode Collation Algorithm plus locale-specific comparison rules from the Common Locale Data Repository, a comprehensive source for this type of data.

Biblioteki ICU
Usługi

- **Formatting**
  Format numbers, dates, times and currency amounts according the conventions of a chosen locale.
  This includes translating month and day names into the selected language, choosing appropriate abbreviations, ordering fields correctly, etc. This data also comes from the Common Locale Data Repository.
- **Time Calculations**
  Multiple types of calendars are provided beyond the traditional Gregorian calendar.

Biblioteki ICU
Usługi

- **Unicode Support**
  ICU closely tracks the Unicode standard, providing easy access to all of the many Unicode character properties, Unicode Normalization, Case Folding and other fundamental operations as specified by the Unicode Standard.
- **Regular Expression**
  ICU's regular expressions fully support Unicode while providing very competitive performance.

Biblioteki ICU
Usługi

- **Bidi**
  Support for handling text containing a mixture of left to right (English) and right to left (Arabic or Hebrew) data.
- **Text Boundaries**
  Locate the positions of words, sentences, paragraphs within a range of text, or identify locations that would be suitable for line wrapping when displaying the text.

Pakiet stringi

stringi: Character string processing facilities

> *stringi allows for very fast, correct, consistent, and convenient character string/text processing in each locale and any native encoding. Thanks to the use of the ICU library, the package provides R users with a platform-independent functionality known to Java, Perl, Python, PHP and Ruby programmers.*

URL: http://cran.r-project.org/web/packages/stringi/,
http://stringi.rexamine.com,
https://github.com/Rexamine/stringi/

License: BSD-3-clause + ICU license

Version: 0.1-25

(Historia: zob. Dodatek #4)

Pakiet stringi

stringi: Character string processing facilities

Depends: R ($\geq$ 2.15.0)

SystemRequirements: ICU4C ($\geq$ 50)

Author: Marek Gagolewski and Bartek Tartanus (stringi source code);
IBM and other contributors (ICU4C 52.1 source code);
Unicode, Inc. (Unicode Character Database)

Pakiet stringi

Założenia projektowe i funkcjonalne

- API zgodne z `stringr`
- Input: dowolne kodowanie
  (R od jakiegoś czasu pozwala współegzystować „w środowisku" napisom w ASCII, kodowaniu natywnym i UTF-8, zob. Dodatek #2)
  (`cat()`, `print()` i inne przekodowują napisy automatycznie)
- Output: zawsze UTF-8
  (z oczywistymi wyjątkami jak `stri_encode()`)
- Wnętrzności: UTF-16 i UTF-8 – ICU API
- Nastawienie na wydajność (w szczeg. dla UTF-8)
  (każda funkcja napisana od zera w C++, minimalizacja kodu R)
- Spójność: wektoryzacja, poprawna obsługa braków danych, przenośność (wyczerpujące `icudt`)

Pakiet stringi

Przegląd funkcji w stringi_0.1-25

```
> stri_join("a", 1:2)

## [1] "a1" "a2"
```
(zgodne z str_join(), czyli sep='' domyślnie)

```
> stri_join("aku", c("ku", NA), sep='_')

## [1] "aku_ku" NA
```
(to jest R-owy styl radzenia sobie z NA)

```
> stri_dup(1:2, c(3, 5))

## [1] "111"   "22222"
```
(zgodne z str_dup(), ale 100× szybciej)

```
> stri_flatten(c("a", "b", "c"))

## [1] "abc"
```
(jak paste(wektor, collapse=''))

Pakiet stringi

Przegląd funkcji w stringi_0.1-25

```
> x <- "stringr"; stri_sub(x, -1) <- "i"; x

## [1] "stringi"
```

(zastąp 1. znak od końca)

```
> stri_sub(c("stringi", "STRINGI"), 3, len=c(2,4))

## [1] "ri"   "RING"
```

(wszystko zwektoryzowane)

```
> stri_trim("    stringi\n\t  ")

## [1] "stringi"
```

(por. np. trim() w PHP)

Pakiet stringi

Przegląd funkcji w stringi_0.1-25

```
> stri_cmp("hladný", "chladný", list(locale="sk_SK"))

## [1] -1
```
(to je úžasné!; stri_cmp_lt() w stringi_0.2-X)

```
> stri_cmp("\u0105", "\u0061\u0328")

## [1] 0
```
(a&ogonek == a+ogonek – tego oczekujemy)

```
> stri_enc_nfc("\u0061\u0328")

## [1] "ą"
```
(ręczna normalizacja, tu NFC)

```
> stri_cmp("a2", "a123", list(numeric=TRUE))

## [1] -1
```
$(2 < 123)$

Pakiet stringi

Przegląd funkcji w stringi_0.1-25

```
> stri_sort(c("a", "b", "w", "z"), opts = list(locale = "et_EE"))

## [1] "a" "b" "z" "w"
```
(Eesti)

```
> stri_order(c("a", "ą", "b"), decreasing=TRUE)

## [1] 3 2 1
```
(por. benchmarki)

```
> stri_locale_get()

## [1] "pl_PL"
```
(aktualne *locale*)

```
> length(stri_locale_list())

## [1] 593
```
(liczba obsługiwanych ustawień)

Pakiet stringi

Przegląd funkcji w stringi_0.1-25

```
> stri_trans_toupper("straße")

## [1] "STRASSE"
```
(Wunderbar!)

```
> stri_split_lines("aa\nbb")

## [[1]]
## [1] "aa" "bb"
```
(taka ciekawostka: wektor $\rightarrow$ lista wektorów)

```
> length(stri_enc_list(TRUE))

## [1] 1198
```
(liczba obsługiwanych kodowań, bez aliasów)

Pakiet stringi

Przegląd funkcji w stringi_0.1-25

Wyszukiwanie: `stri_co_jak()`

`co`:

- `detect`
- `count`
- `extract_all`, `extract_first`, `extract_last`
- `locate_all`, `locate_first`, `locate_last`
- `replace_all`, `replace_first`, `replace_last`
- `split`

`jak`:

- `charclass`
- `regex`
- `fixed` (2 silniki)

Pakiet stringi

Przegląd funkcji w stringi_0.1-25

```
> stri_split_charclass("a   b\nc", "WHITESPACE", omit_empty=TRUE)

## [[1]]
## [1] "a" "b" "c"
```

(podziel napisy wzg. białych znaków; b. szybkie)

```
> stri_count_regex("bab baab baaab",c("b.*?b","b.b"))

## [1] 3 2
```

(wyrażenia regularne; tu: wektoryzacja)

```
> stri_match_all_regex("zupa=rosol, deser=lody", "(\\w+)=(\\w+)")

## [[1]]
##      [,1]          [,2]    [,3]
## [1,] "zupa=rosol" "zupa"  "rosol"
## [2,] "deser=lody" "deser" "lody"
```

(ekstrakcja podwyrażeń w grupach)

Pakiet stringi

Przegląd funkcji w stringi_0.1-25

Ciekawostka:

```
> apply(
+    sapply(
+       list.files(path="../../../ProgramowanieR/rozdzialy/",
+          pattern=glob2rx("*.tex"),
+          recursive=TRUE, full.names=TRUE),
+       function(x)
+       stri_stats_latex(readLines(x))
+    ), 1, sum)

##      CharsWord CharsCmdEnvir    CharsWhite         Words
##         718755        458403        281989        120202
##           Cmds       Envirs
##          37055          6119 (str_stats_latex() – dla autorów)
```

Milestone 0.2

Brakujące funkcje, które są dostępne w `base` R / `stringr`:

- %==%, %<%, , %>=%, ...
- `stri_wrap`
- `stri_pad`
- `stri_trans_char`
- `stri_enc_tonative`
- ...

I najważniejsze:

- Wydajność: funkcje mają działać co najmniej tak szybko jak funkcje z R.
  (choć czasem nie będzie to możliwe dla wejścia ≠ ASCII/UTF-8)
  (Benchmarki: zob. Dodatek #3)

Milestone >0.2

Niedaleka przyszłość:

- Date and time formatting
- Number formatting
- Rule-based number formatting (Number spellout)
- Read/Write lines

Dalsza przyszłość:

- `match()`, `pmatch()`
- Random string generation
- Text boundary analysis (por. jednak `stri_wrap`)
- Bidi (? – wyświetlanie zależne od OS)
- Spoofing detection
- HTML entities, Base 64
- Read/Write CSV
- ...

Zachęcamy
do użytkowania

```
install.packages("stringi")
library("stringi")
stri_install_check()
```

i do współpracy

https://github.com/Rexamine/stringi/

Dodatek
Trochę historii

NEWS:

- R 0.63.0 (1998-11-14): The PostScript device driver now uses the ISO Latin1 font encoding. This should allow Western Europeans to render their languages correctly. It is likely that additional encodings will be added (e.g. Latin2) when we figure out how to set the correct font encoding in printers.
- R 0.65.1 (1999-10-06): Comparison of strings uses the current locale on systems where this is available, and so is always consistent with the ordering used by `sort()`.
- R 1.2.0 (2000-12-15): Date-time support functions with classes `"POSIXct"` and `"POSIXlt"` to represent dates/times (resolution 1 second) in the POSIX formats (...) these (...) functions know about time zones (if the OS does).

Dodatek
Trochę historii

NEWS:

- R 1.3.0 (2001-06-22): Text input from `file`, `pipe`, `fifo`, `gzfile` and `url` connections can be read with a user-specified encoding.

  `postscript()` allows user-specified encoding, with encoding files supplied for Windows, Mac, Unicode and various others, and with an appropriate platform-specific default.

- R 1.5.0 (2002-04-29): Missing character strings are treated as missing much more consistently, e.g. in logical comparisons and in sorts. `identical()` now differentiates `"NA"` from the missing string.

- R 1.6.0 (2002-10-01): `grep()`, `(g)sub()` and `regexpr()` have a new argument `perl` which if `TRUE` uses Perl-style regexps from PCRE (if installed). New capabilities option `"PCRE"` to say if PCRE is available.

Dodatek
Trochę historii

NEWS:

- R 1.7.0 (2003-04-16): `PCRE` and `bzip2` are built from versions in the R sources if the appropriate library is not found.
- R 1.8.0 (2003-10-08): `sub`, `gsub`, `grep`, `regexpr`, `chartr`, `tolower`, `toupper`, `substr`, `substring`, `abbreviate` and `strsplit` now handle missing values differently from `"NA"`.

  New argument `fixed = TRUE` for `grep()` and `regexpr()` to avoid the need to escape strings to match.
- R 1.9.0 (2004-04-12): A warning is issued at startup in a UTF-8 locale, as currently R only supports single-byte encodings.

  Functions `grep()`, `regexpr()`, `sub()` and `gsub()` (...): The `perl=TRUE` argument now uses character tables prepared for the locale currently in use each time it is used, rather than those of the C locale.

Dodatek
Trochę historii

NEWS:

- R 2.1.0 (2005-04-19): Unix-alike versions of R can now be used in UTF-8 locales on suitably equipped OSes (...).

  `source()` now has an `'encoding'` argument which can be used to make it try out various possible encodings. This is made use of by example() which will convert (non-UTF-8) Latin-1 example files in a UTF-8 locale.

  New function `iconv()` to convert character vectors between encodings, on those OSes which support this. See the new `capabilities("iconv")`

- R 2.2.0 (2005-10-06): Functions `utf8ToInt()` and `intToUtf8()` to work with UTF-8 encoded character strings (irrespective of locale or OS-level UTF-8 support).

Dodatek
Trochę historii

NEWS:

- R 2.3.1 (2006-06-01): The `\uxxxx` notation for Unicode characters in input strings can now be used on any platform which supports MBCS, even if the current locale is not MBCS (provided that the Unicode character is valid in the current character set).
- R 2.7.0 (2008-04-22) There is a new public interface to the encoding info stored on `CHARSXPs`, `getCharCE` and `mkCharCE` using the enumeration type `cetype_t`.

  `grep()`, `strsplit()` and friends with `fixed=TRUE` or `perl=TRUE` work in UTF-8 and preserve the UTF-8 encoding for UTF-8 inputs where supported.

Dodatek
Trochę historii

NEWS:

- R 2.10.0 (2009-10-26): A different regular expression engine is used for basic and extended regexps and is also for approximate matching. This is based on the TRE library of Ville Laurikari, a modified copy of which is included in the R sources.

  `"\uxxxx"` and `"\Uxxxxxxxx"` escapes can now be parsed to a UTF-8 encoded string even in non-UTF-8 locales (this has been implemented on Windows since R 2.7.0). The semantics have been changed slightly: a string containing such escapes is always stored in UTF-8 (and hence is suitable for portably including Unicode text in packages).

- R 2.11.0 (2010-04-22): New functions `enc2native()` and `enc2utf8()` convert character vectors with possibly marked encodings to the current locale and UTF-8 respectively.

Dodatek
Trochę historii

NEWS:

- R 2.12.0 (2010-10-15): The ability of `readLines()` and `scan()` to re-encode inputs to marked UTF-8 strings on Windows since R 2.7.0 is extended to non-UTF-8 locales on other OSes.
- R 2.13.0 (2011-04-13): There is an additional marked encoding `"bytes"` for character strings. This is intended to be used for non-ASCII strings which should be treated as a set of bytes, and never re-encoded as if they were in the encoding of the current locale: `useBytes = TRUE` is automatically selected in functions such as `writeBin()`, `writeLines()`, `grep()` and `strsplit()`.

Dodatek
Trochę historii

NEWS:

- R 3.0.3 (2014-03-06): `strptime()` and the `format()` methods for classes `"POSIXct"`, `"POSIXlt"` and `"Date"` recognize strings with marked encodings: this allows, for example, UTF-8 French month names to be read on (French) Windows.

  `iconv(to = "utf8")` is now accepted on all platforms (some implementations did already, but GNU libiconv did not: however converted strings were not marked as being in UTF-8). The official name, `"UTF-8"` is still preferred.

Dodatek

Reprezentacja napisów: We wnętrzu R

Wektory atomowe w R:

`logical`, `raw`, `integer`, `numeric`, `complex`, `character`.

`character` to dziwoląg: wektor napisów = ciąg ciągów liczb

```
// Rinternals.h
typedef enum {
    CE_NATIVE = 0,
    CE_UTF8 = 1,
    CE_LATIN1 = 2,
    CE_BYTES = 3,
    CE_SYMBOL = 5,
    CE_ANY =99
} cetype_t;
```

Dodatek

Reprezentacja napisów: We wnętrzu R

```
// envir.c
SEXP mkCharLenCE(const char* name, int len, cetype_t enc)
{
   // ...
   for (int slen = 0; slen < len; slen++) {
      if ((unsigned int) name[slen] > 127) is_ascii=FALSE;
   // ...
   /* Search for a cached value */
   // ...
   PROTECT(cval = allocCharsxp(len));
   memcpy(CHAR_RW(cval), name, len);
   // ...
   SET_CACHED(cval); /* Mark it */
   // ...
   return cval;
}
```

Dodatek

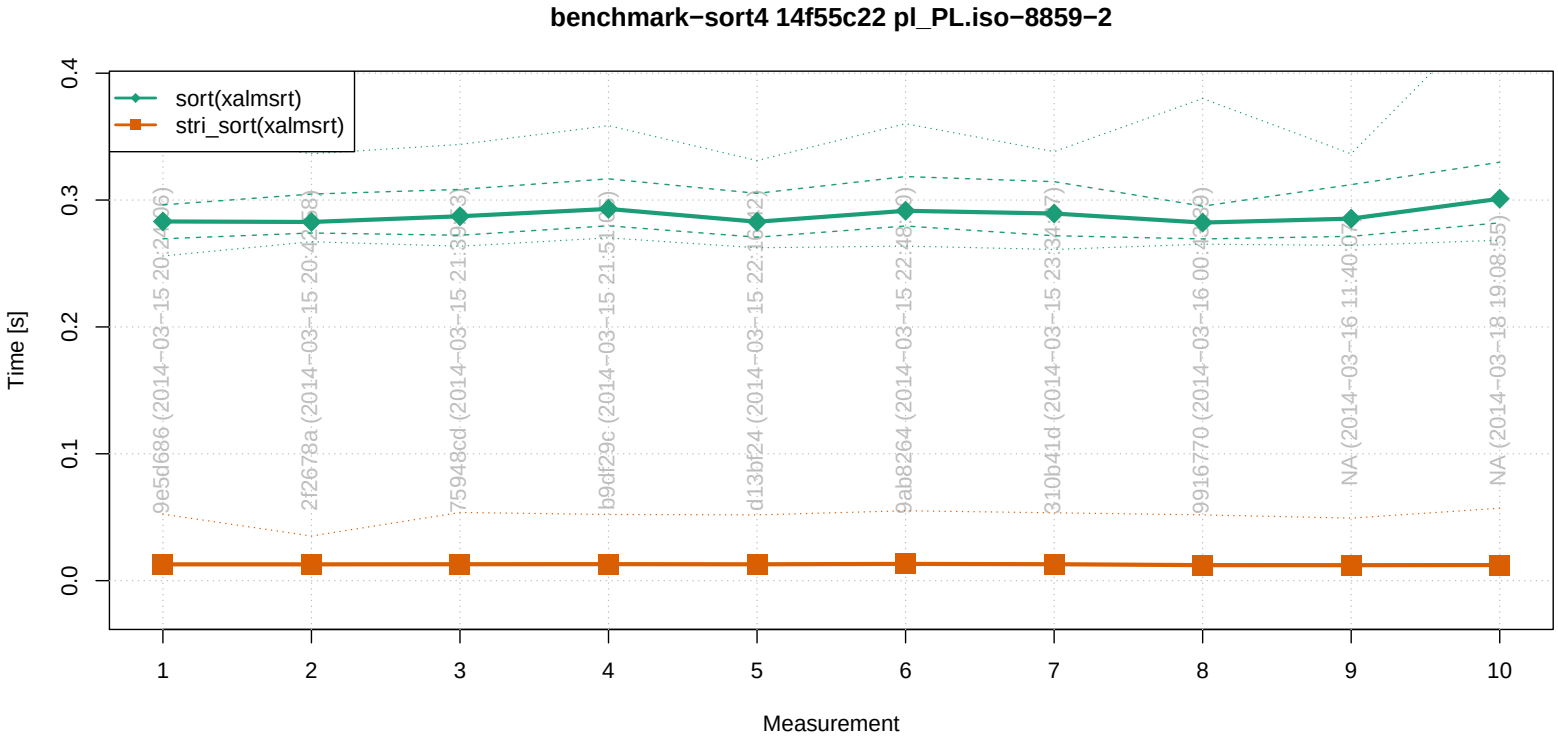
Reprezentacja napisów: We wnętrzu R

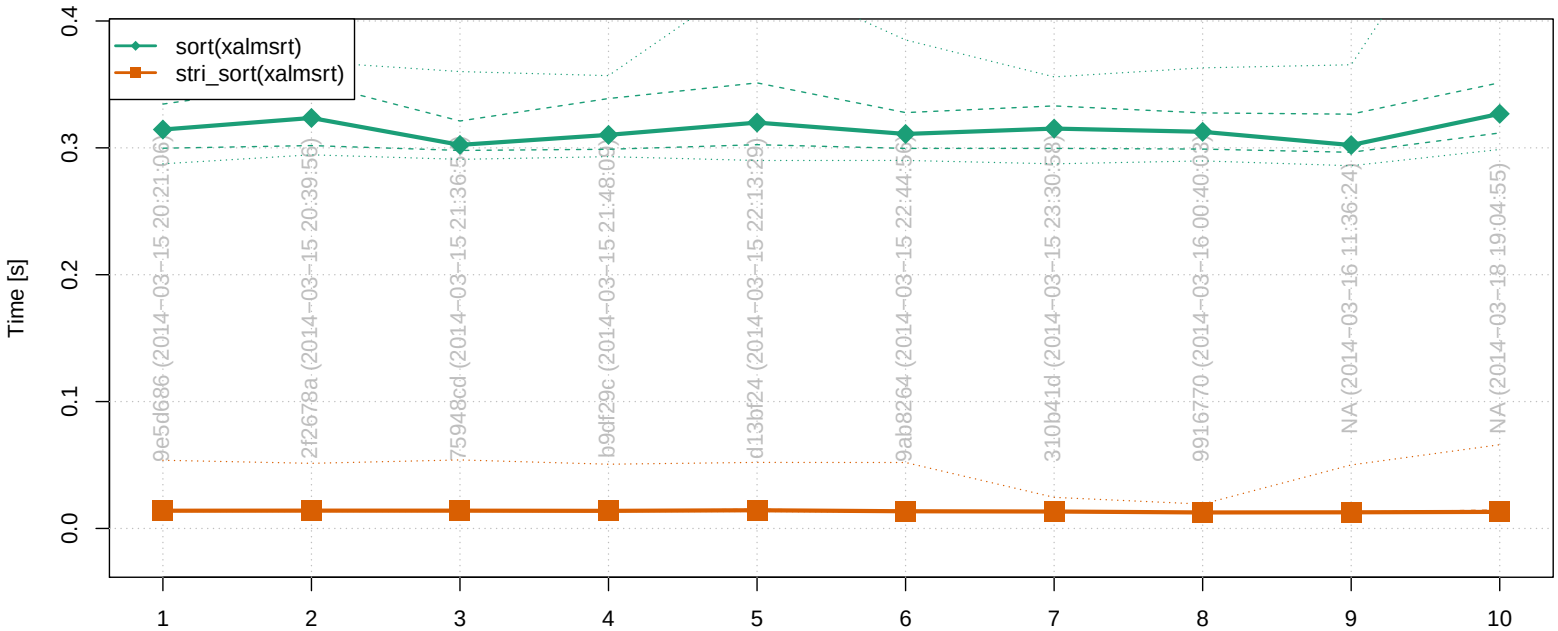
```
// memory.c
SEXP allocCharsxp(R_len_t len) {
    return allocVector(intCHARSXP /* 73 */, len);
}

SEXP allocVector(SEXPTYPE type, R_xlen_t length) {
    // ...
    case intCHARSXP:
        size = BYTE2VEC(length + 1);
    // ...
}
```

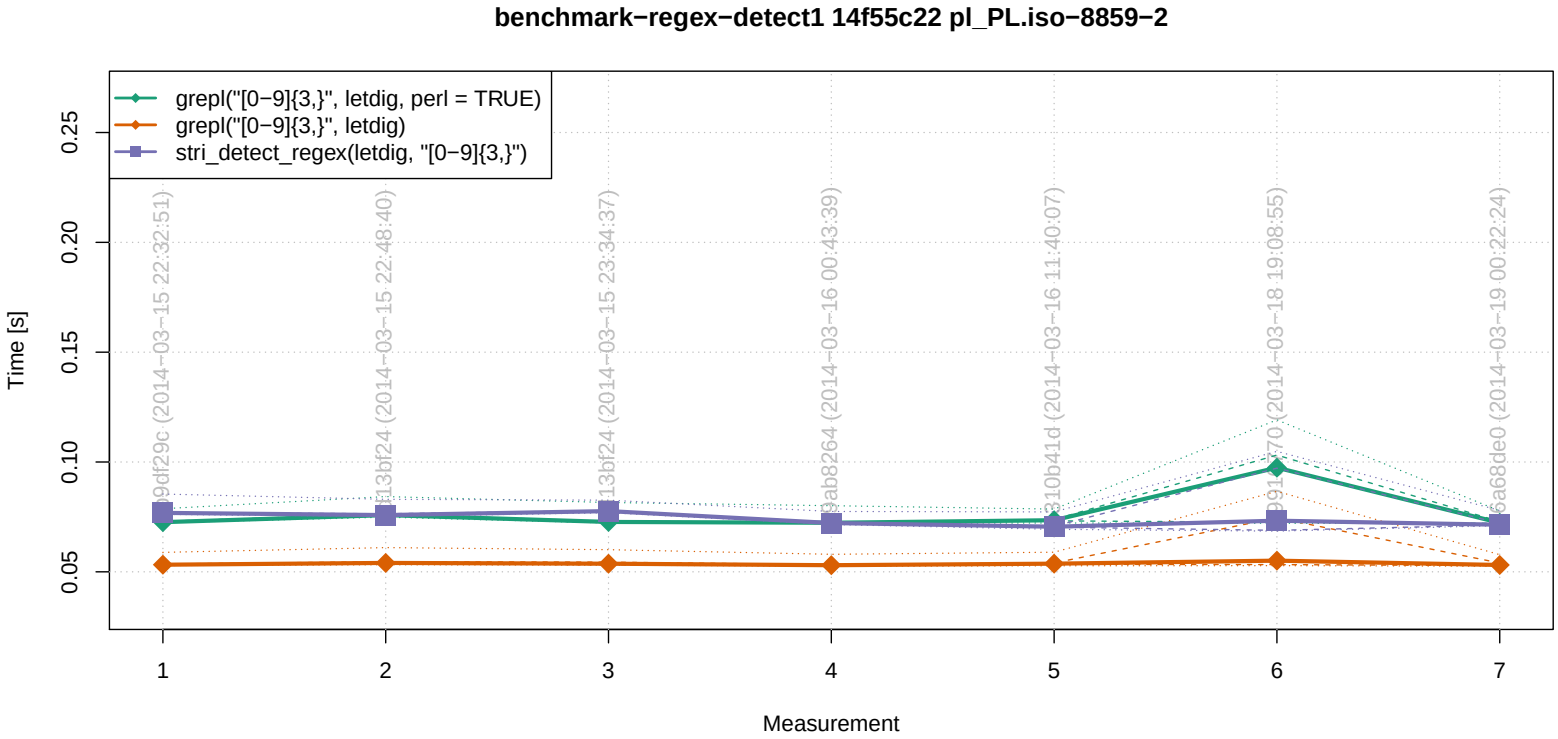
Dodatek

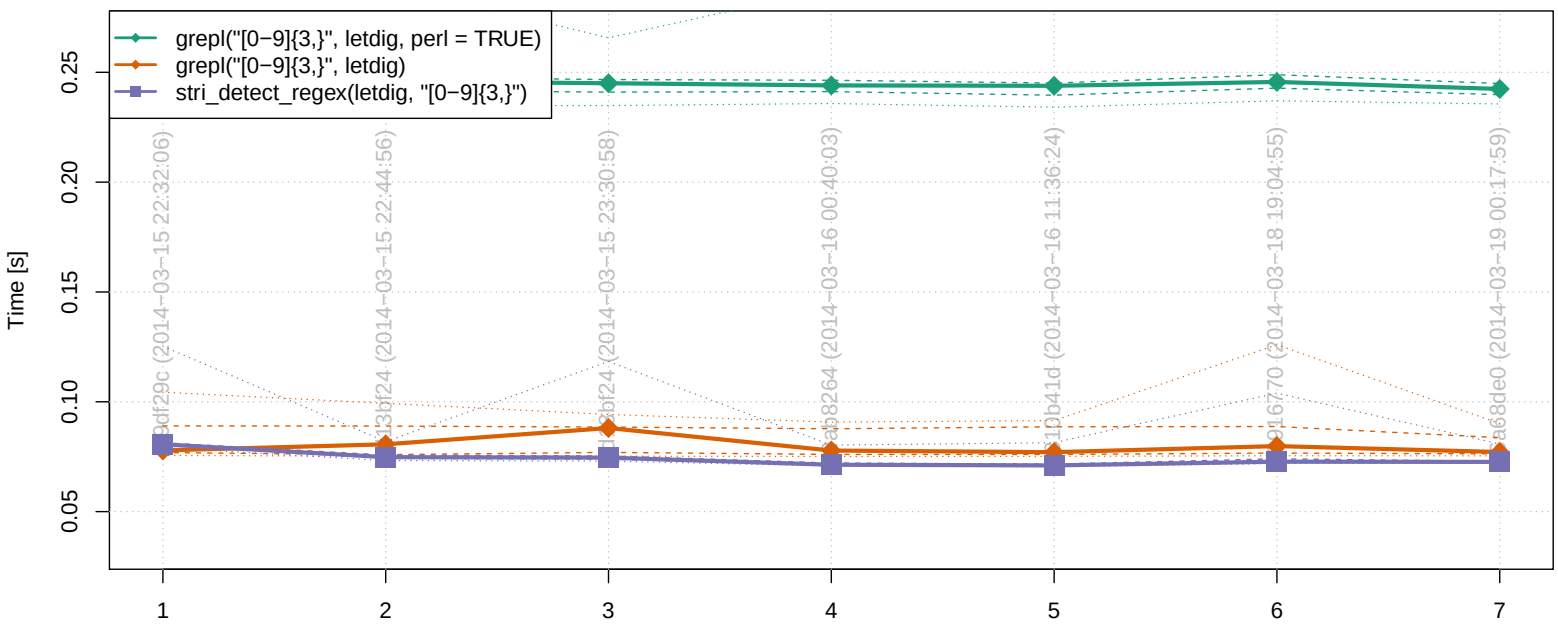
Reprezentacja napisów: We wnętrzu R

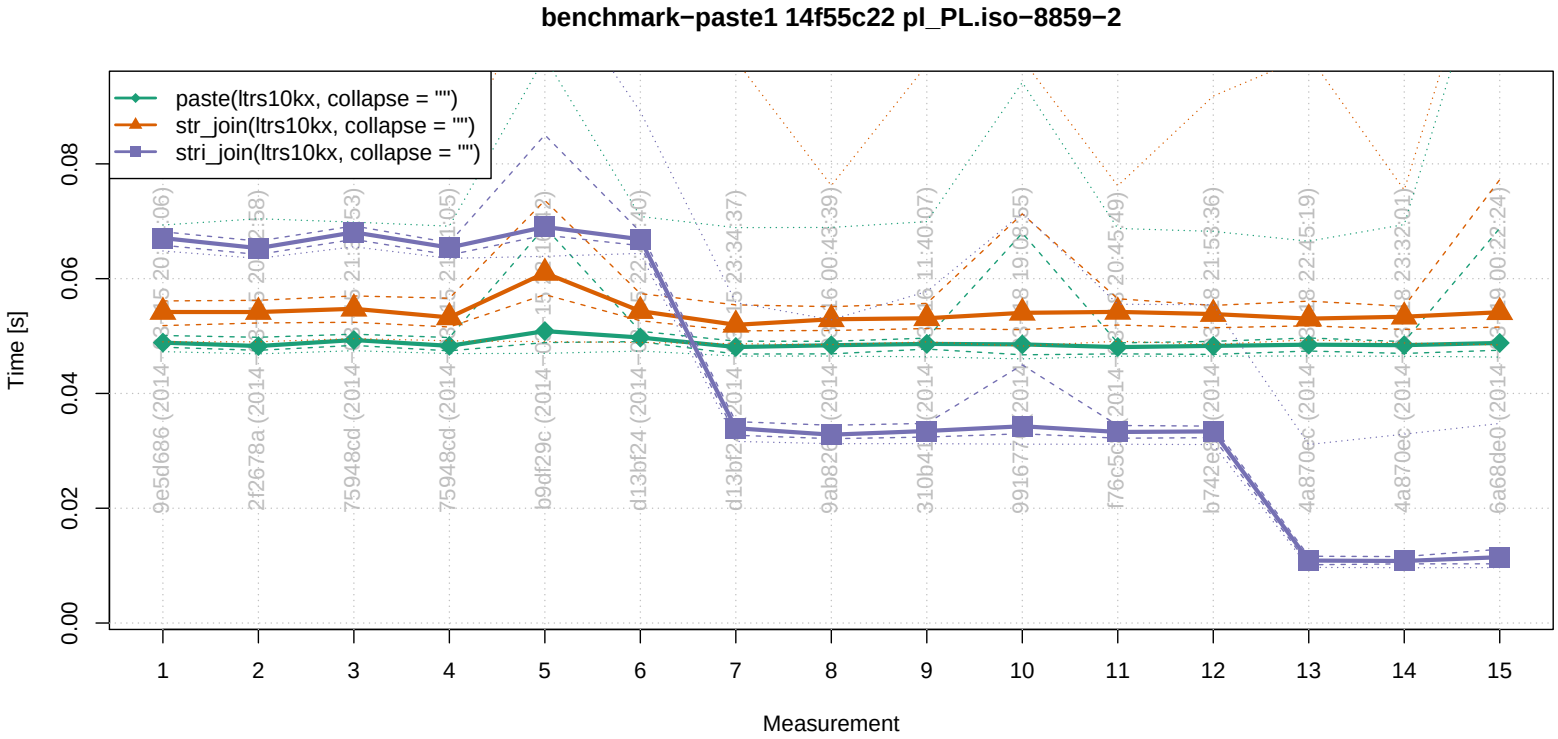
```
// Rinternals.h
typedef struct SEXPREC* SEXP;

// SEXPREC / sxpinfo:
// type
// length
// gp (np. #define IS_UTF8(x) \
//       ((x)->sxpinfo.gp & UTF8_MASK))
```

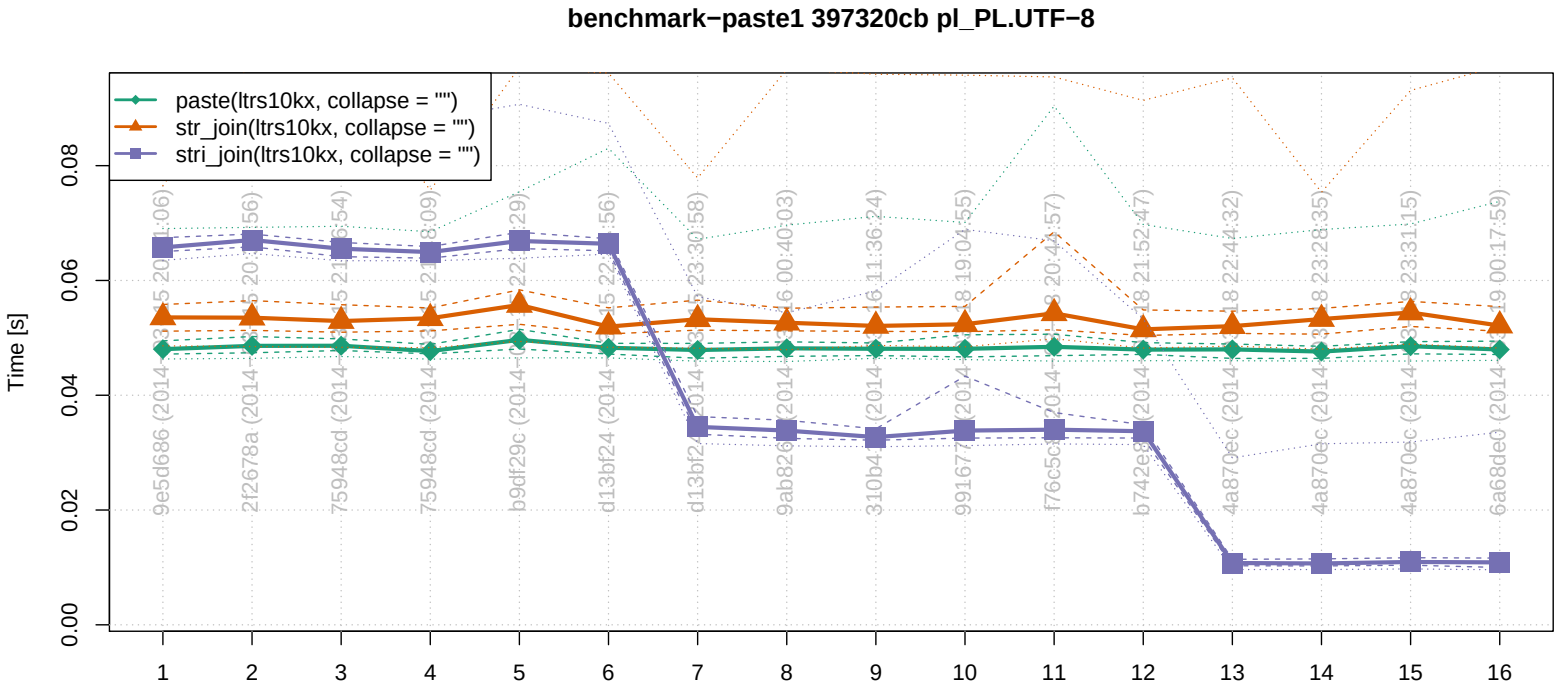

benchmark−sort4 14f55c22 pl_PL.iso−8859−2
sort(xalmsrt)
stri_sort(xalmsrt)
Time [s]
0.0
0.1
0.2
0.3
0.4
1
2
3
4
5
6
7
8
9
10
Measurement
9e5d686 (2014−03−15 20:21:06)
2f2678a (2014−03−15 20:39:58)
75948cd (2014−03−15 21:36:58)
b9df29c (2014−03−15 21:51:[illegible])
d13bf24 (2014−03−15 22:13:29)
9ab8264 (2014−03−15 22:48:[illegible])
310b41d (2014−03−15 23:34:[illegible])
9916770 (2014−03−16 00:40:[illegible])
NA (2014−03−16 11:40:07)
NA (2014−03−18 19:08:55)

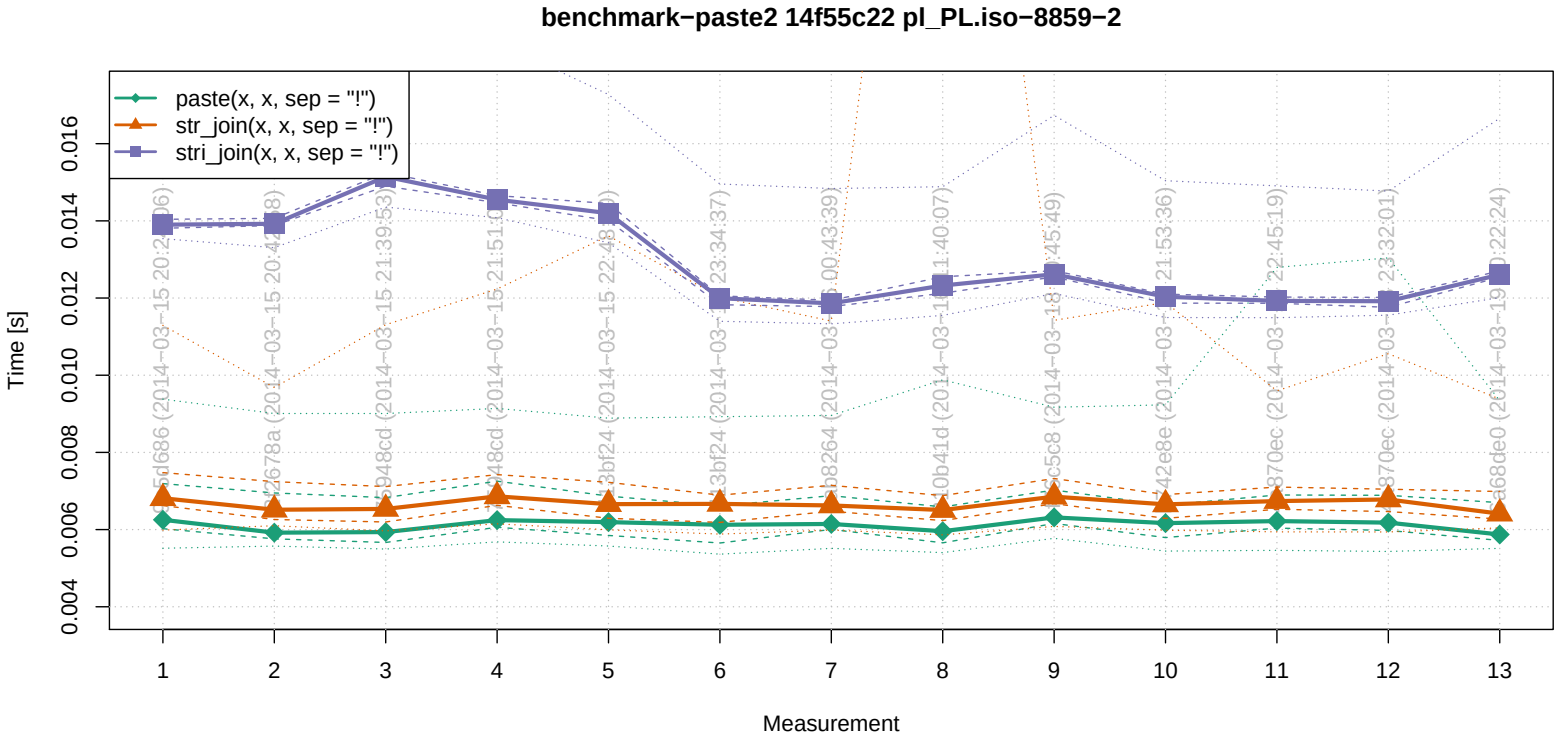

benchmark−sort4 397320cb pl_PL.UTF−8
sort(xalmsrt)
stri_sort(xalmsrt)
Time [s]
0.0
0.1
0.2
0.3
0.4
1
2
3
4
5
6
7
8
9
10
9e5d686 (2014−03−15 20:21:06)
2f2678a (2014−03−15 20:39:58)
75948cd (2014−03−15 21:36:58)
b9df29c (2014−03−15 21:48:09)
d13bf24 (2014−03−15 22:13:29)
9ab8264 (2014−03−15 22:44:56)
310b41d (2014−03−15 23:30:58)
9916770 (2014−03−16 00:40:03)
NA (2014−03−16 11:36:24)
NA (2014−03−18 19:04:55)

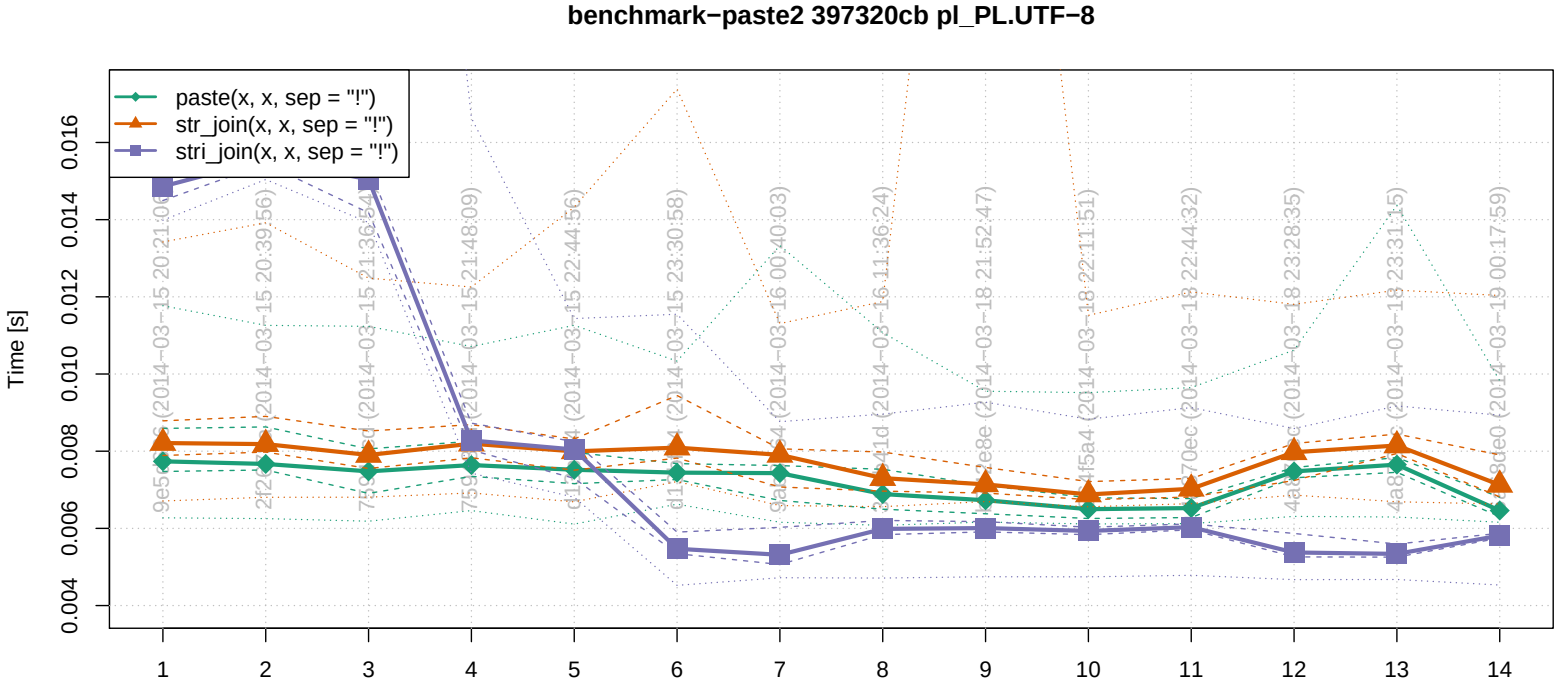

benchmark−regex−detect1 14f55c22 pl_PL.iso−8859−2
grepl("[0−9]{3,}", letdig, perl = TRUE)
grepl("[0−9]{3,}", letdig)
stri_detect_regex(letdig, "[0−9]{3,}")
Time [s]
0.05
0.10
0.15
0.20
0.25
1
2
3
4
5
6
7
Measurement
9df29c (2014−03−15 22:32:51)
13bf24 (2014−03−15 22:48:40)
13bf24 (2014−03−15 23:34:37)
3ab8264 (2014−03−16 00:43:39)
310b41d (2014−03−16 11:40:07)
91
70 (2014−03−18 19:08:55)
6a68de0 (2014−03−19 00:22:24)

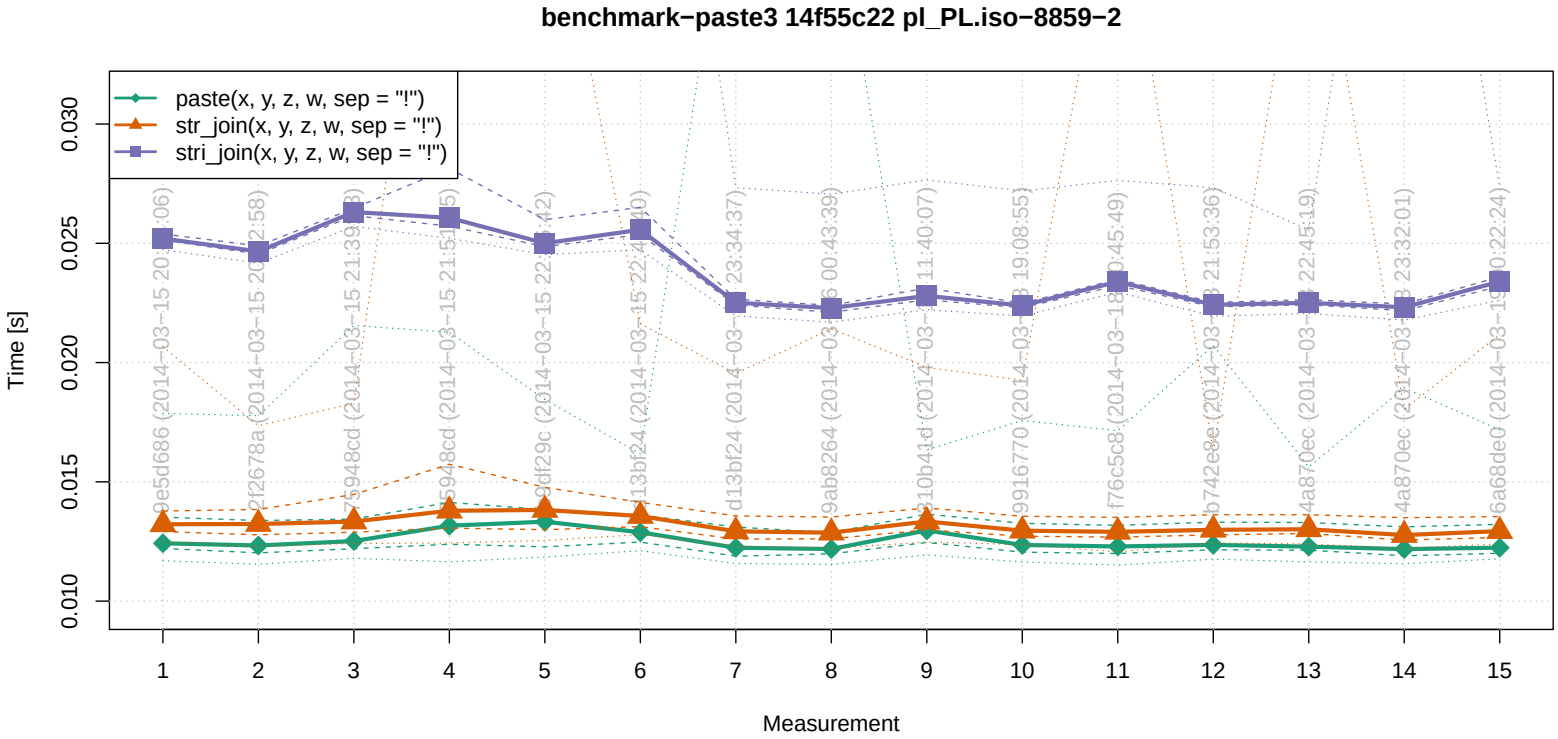

benchmark−regex−detect1 397320cb pl_PL.UTF−8
grepl("[0−9]{3,}", letdig, perl = TRUE)
grepl("[0−9]{3,}", letdig)
stri_detect_regex(letdig, "[0−9]{3,}")
Time [s]
0.05
0.10
0.15
0.20
0.25
1
2
3
4
5
6
7
9df29c (2014−03−15 22:32:06)
13bf24 (2014−03−15 22:44:56)
13bf24 (2014−03−15 23:30:58)
3b8264 (2014−03−16 00:40:03)
10b41d (2014−03−16 11:36:24)
916770 (2014−03−18 19:04:55)
68de0 (2014−03−19 00:17:59)

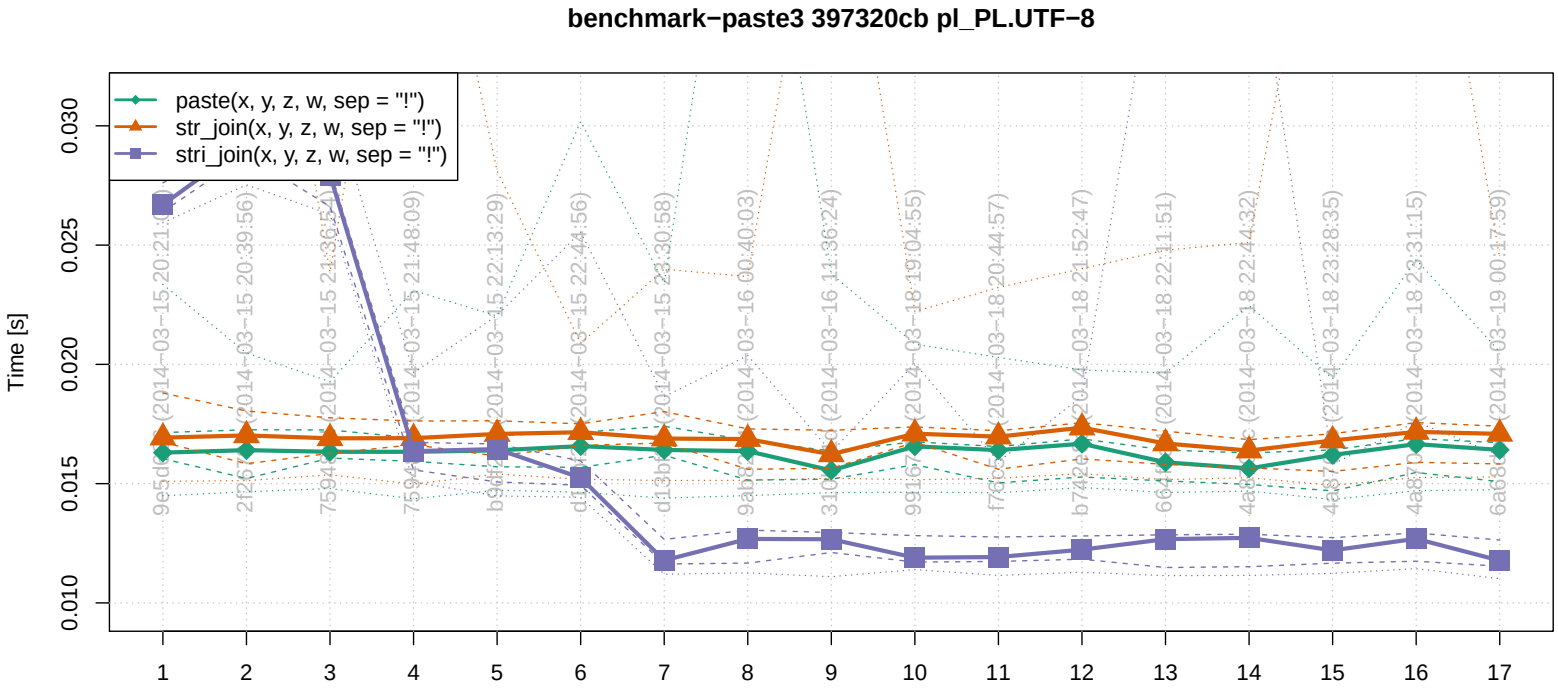

benchmark−paste1 14f55c22 pl_PL.iso−8859−2
paste(ltrs10kx, collapse = "")
str_join(ltrs10kx, collapse = "")
stri_join(ltrs10kx, collapse = "")
Time [s]
0.00
0.02
0.04
0.06
0.08
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
Measurement

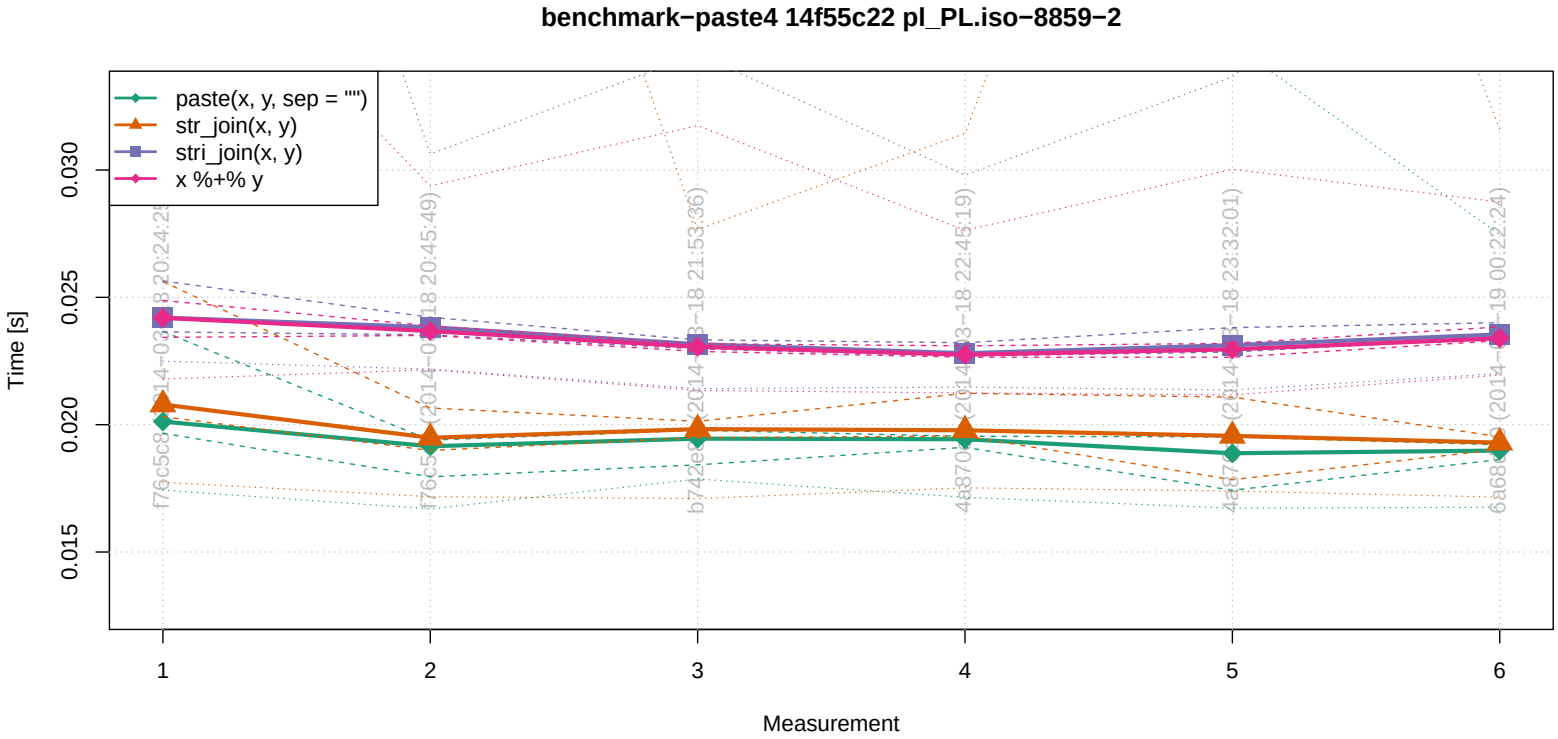

benchmark−paste1 397320cb pl_PL.UTF−8
paste(ltrs10kx, collapse = "")
str_join(ltrs10kx, collapse = "")
stri_join(ltrs10kx, collapse = "")
Time [s]
0.00
0.02
0.04
0.06
0.08
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

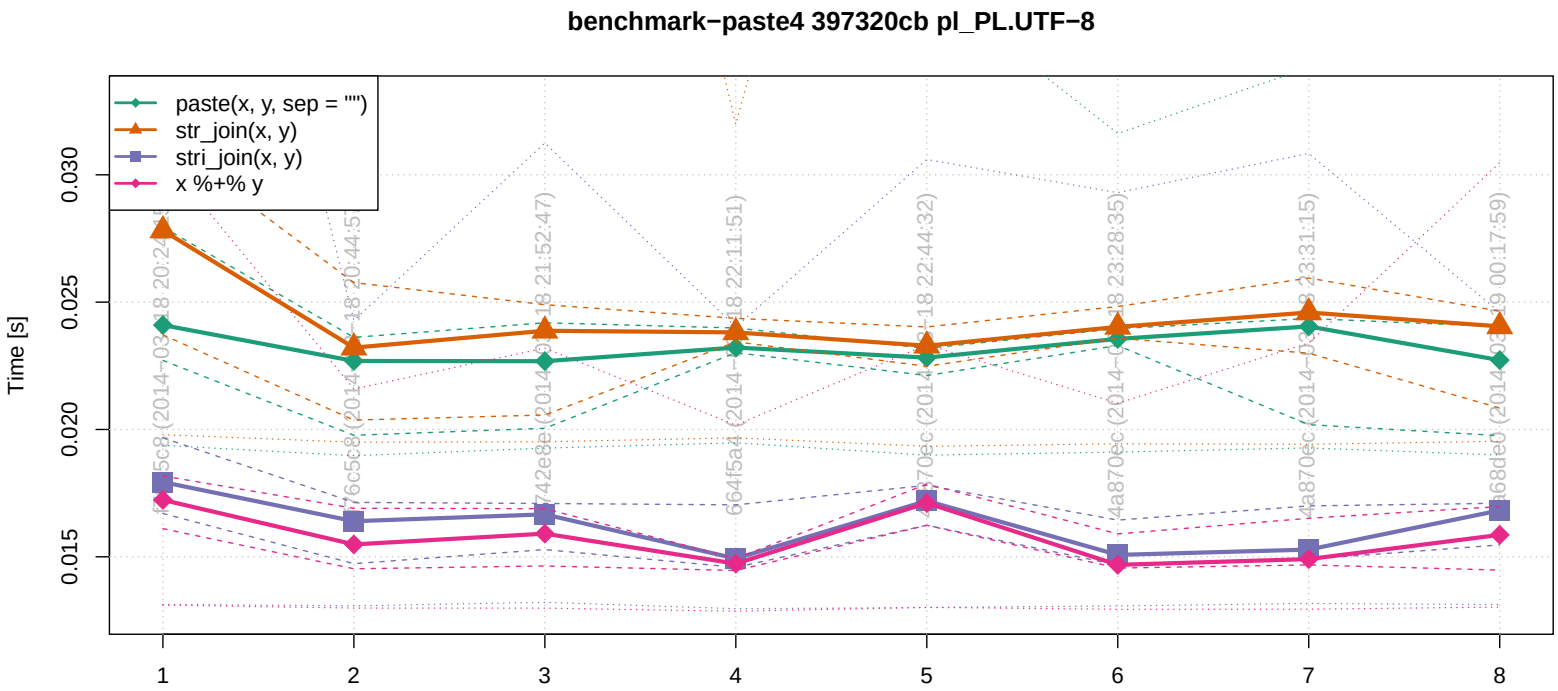

benchmark−paste2 14f55c22 pl_PL.iso−8859−2
paste(x, x, sep = "!")
str_join(x, x, sep = "!")
stri_join(x, x, sep = "!")
Time [s]
0.004
0.006
0.008
0.010
0.012
0.014
0.016
1
2
3
4
5
6
7
8
9
10
11
12
13
Measurement

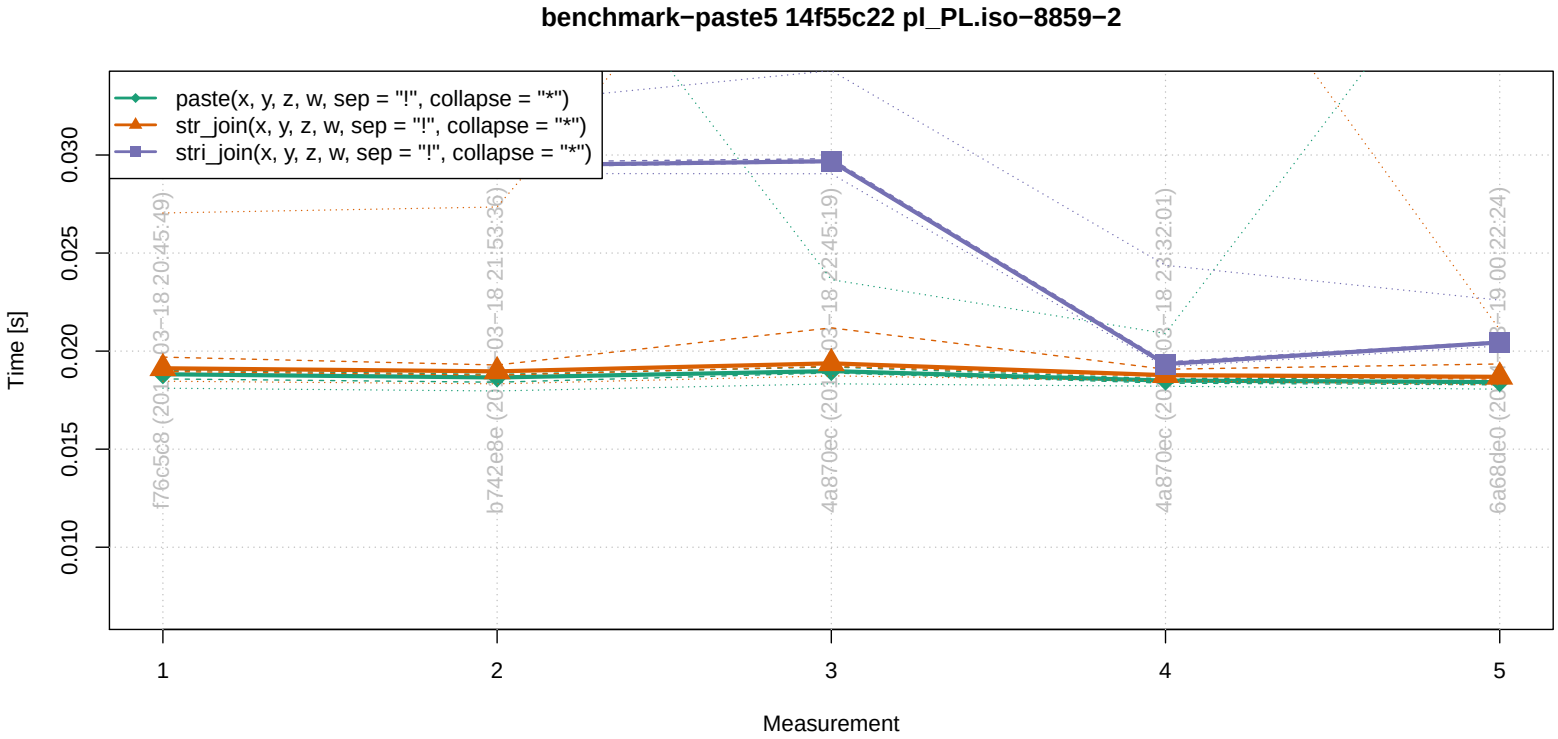

benchmark−paste2 397320cb pl_PL.UTF−8
paste(x, x, sep = "!")
str_join(x, x, sep = "!")
stri_join(x, x, sep = "!")
Time [s]
0.004
0.006
0.008
0.010
0.012
0.014
0.016
1
2
3
4
5
6
7
8
9
10
11
12
13
14

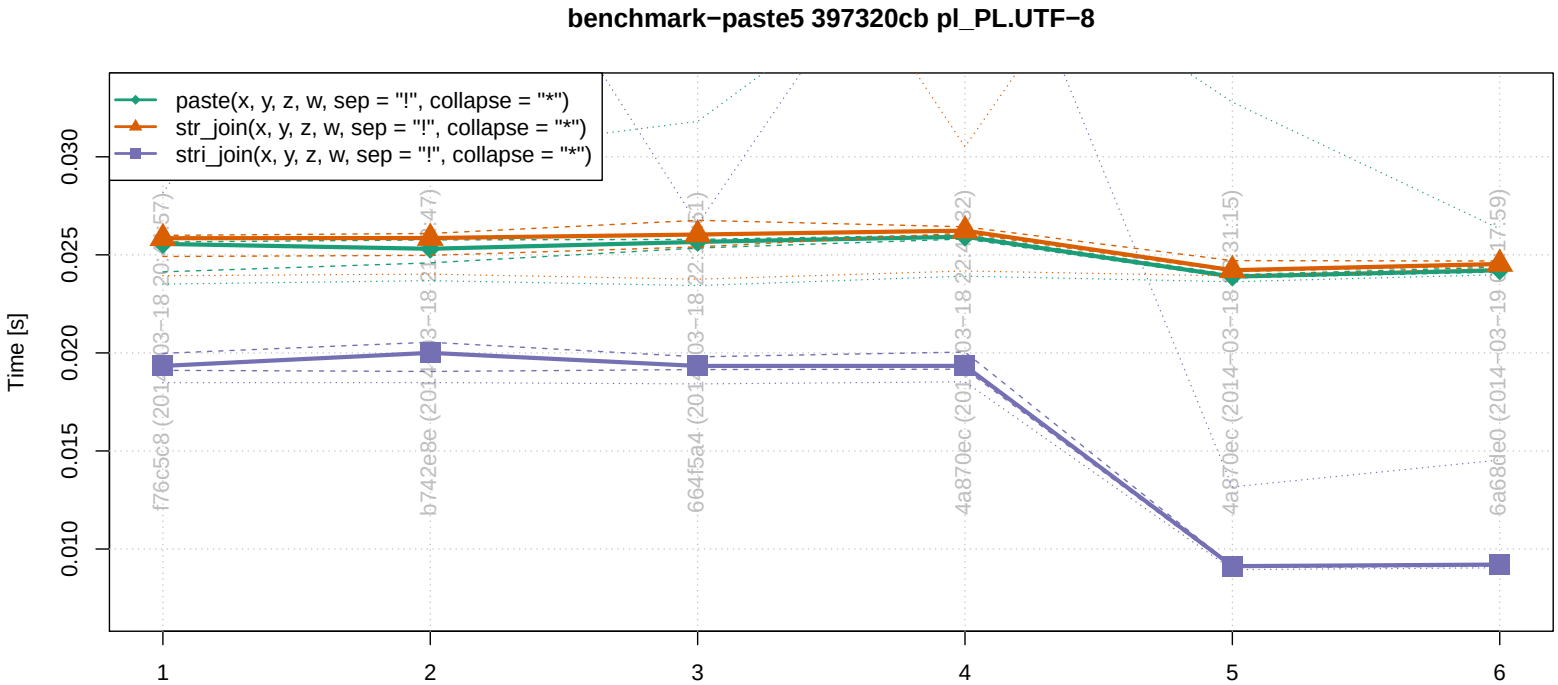

benchmark−paste3 14f55c22 pl_PL.iso−8859−2
paste(x, y, z, w, sep = "!")
str_join(x, y, z, w, sep = "!")
stri_join(x, y, z, w, sep = "!")
Time [s]
0.010
0.015
0.020
0.025
0.030
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
Measurement

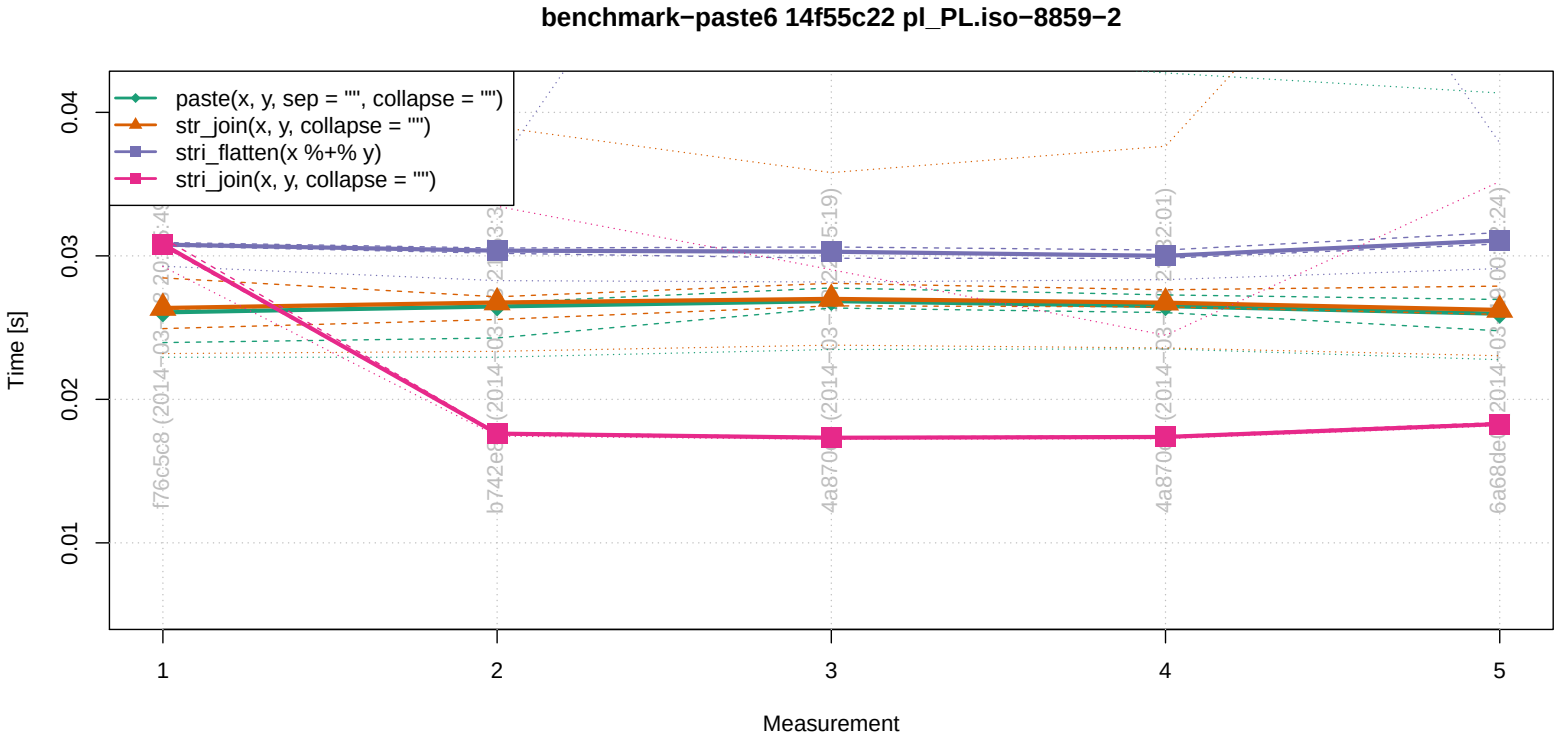

benchmark−paste3 397320cb pl_PL.UTF−8
paste(x, y, z, w, sep = "!")
str_join(x, y, z, w, sep = "!")
stri_join(x, y, z, w, sep = "!")
Time [s]
0.010
0.015
0.020
0.025
0.030
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17

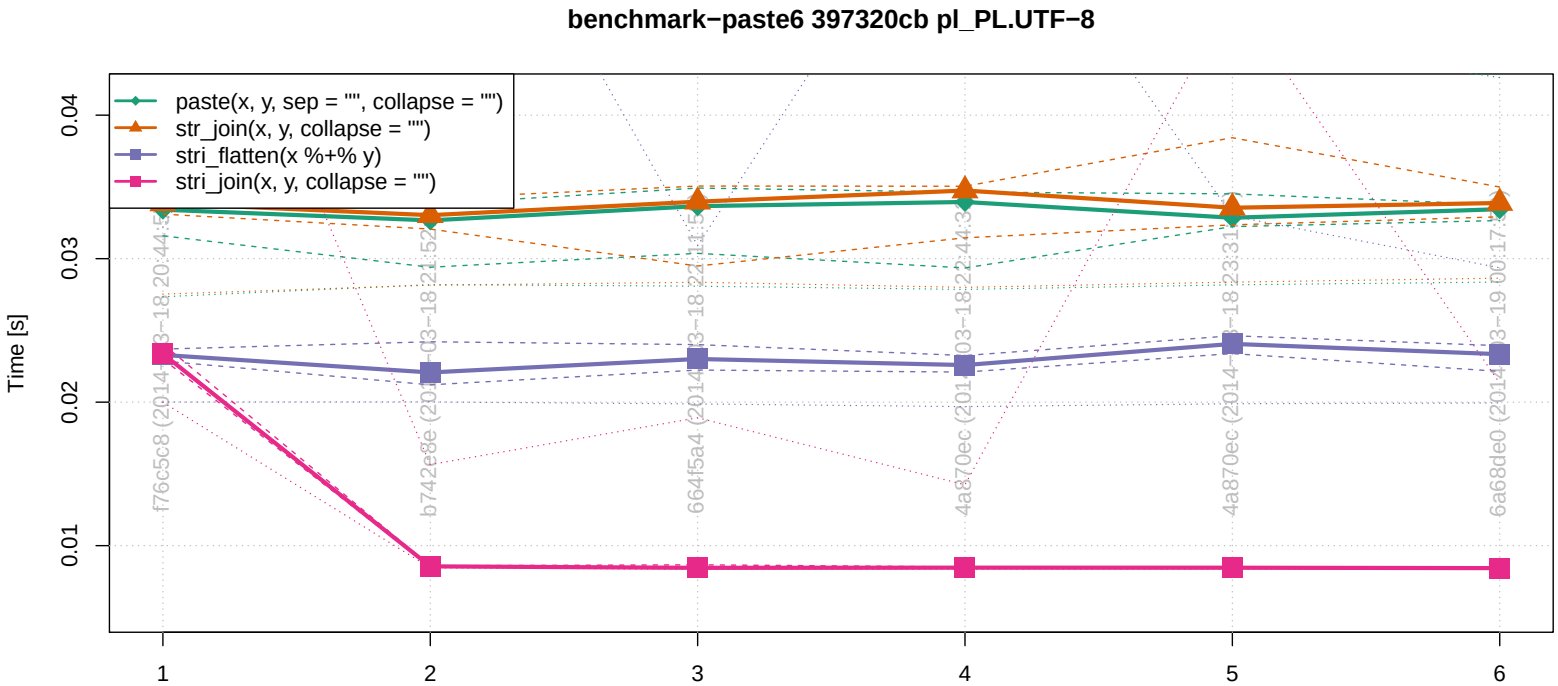

benchmark−paste4 14f55c22 pl_PL.iso−8859−2
paste(x, y, sep = "")
str_join(x, y)
stri_join(x, y)
x %+% y
Time [s]
0.015
0.020
0.025
0.030
1
2
3
4
5
6
Measurement


benchmark−paste4 397320cb pl_PL.UTF−8
paste(x, y, sep = "")
str_join(x, y)
stri_join(x, y)
x %+% y
Time [s]
0.015
0.020
0.025
0.030
1
2
3
4
5
6
7
8


benchmark−paste5 14f55c22 pl_PL.iso−8859−2
paste(x, y, z, w, sep = "!", collapse = "*")
str_join(x, y, z, w, sep = "!", collapse = "*")
stri_join(x, y, z, w, sep = "!", collapse = "*")
Time [s]
0.010
0.015
0.020
0.025
0.030
f76c5c8 (2014−03−18 20:45:49)
b742e8e (2014−03−18 21:53:36)
4a870ec (2014−03−18 22:45:19)
4a870ec (2014−03−18 23:32:01)
6a68de0 (2014−03−19 00:22:24)
1
2
3
4
5
Measurement


benchmark−paste5 397320cb pl_PL.UTF−8
paste(x, y, z, w, sep = "!", collapse = "*")
str_join(x, y, z, w, sep = "!", collapse = "*")
stri_join(x, y, z, w, sep = "!", collapse = "*")
Time [s]
0.010
0.015
0.020
0.025
0.030
f76c5c8 (2014−03−18 20:45:57)
b742e8e (2014−03−18 21:53:47)
664f5a4 (2014−03−18 22:45:51)
4a870ec (2014−03−18 23:32:32)
4a870ec (2014−03−18 23:31:15)
6a68de0 (2014−03−19 00:17:59)
1
2
3
4
5
6


benchmark−paste6 14f55c22 pl_PL.iso−8859−2
paste(x, y, sep = "", collapse = "")
str_join(x, y, collapse = "")
stri_flatten(x %+% y)
stri_join(x, y, collapse = "")
Time [s]
0.01
0.02
0.03
0.04
1
2
3
4
5
Measurement


benchmark−paste6 397320cb pl_PL.UTF−8
paste(x, y, sep = "", collapse = "")
str_join(x, y, collapse = "")
stri_flatten(x %+% y)
stri_join(x, y, collapse = "")
Time [s]
0.01
0.02
0.03
0.04
1
2
3
4
5
6

Dodatek

GitHub: Commits to master