\d is less efficient than [0-9]

Asked 6 years, 5 months ago Active 1 year, 6 months ago Viewed 80k times

1209

235

I made a comment yesterday on an answer where someone had used `[0123456789]` in a regular expression rather than `[0-9]` or `\d` . I said it was probably more efficient to use a range or digit specifier than a character set.

I decided to test that out today and found out to my surprise that (in the C# regex engine at least) `\d` appears to be less efficient than either of the other two which don't seem to differ much. Here is my test output over 10000 random strings of 1000 random characters with 5077 actually containing a digit:

```
Regular expression \d           took 00:00:00.2141226 result: 5077/10000
Regular expression [0-9]        took 00:00:00.1357972 result: 5077/10000  63.42 % of
first
Regular expression [0123456789] took 00:00:00.1388997 result: 5077/10000  64.87 % of
first
```

It's a surprise to me for two reasons:

1. I would have thought the range would be implemented much more efficiently than the set.
2. I can't understand why `\d` is worse than `[0-9]` . Is there more to `\d` than simply shorthand for `[0-9]` ?

Here is the test code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Diagnostics;
using System.Text.RegularExpressions;

namespace SO_RegexPerformance
{
    class Program
    {
        static void Main(string[] args)
        {
            var rand = new Random(1234);
            var strings = new List<string>();
            //10K random strings
            for (var i = 0; i < 10000; i++)
            {
                //Generate random string
                var sb = new StringBuilder();
                for (var c = 0; c < 1000; c++)
                {
                    //Add a-z randomly
                    sb.Append((char)('a' + rand.Next(26)));
                }
                //In roughly 50% of them, put a digit
                if (rand.Next(2) == 0)
                {
                    //Replace one character with a digit, 0-9
                    sb[rand.Next(sb.Length)] = (char)('0' + rand.Next(10));
                }
                strings.Add(sb.ToString());
            }

            var baseTime = testPerfomance(strings, @"\d");
            Console.WriteLine();
            var testTime = testPerfomance(strings, "[0-9]");
            Console.WriteLine("  {0:P2} of first", testTime.TotalMilliseconds /
baseTime.TotalMilliseconds);
            testTime = testPerfomance(strings, "[0123456789]");
            Console.WriteLine("  {0:P2} of first", testTime.TotalMilliseconds /
baseTime.TotalMilliseconds);
        }

        private static TimeSpan testPerfomance(List<string> strings, string regex)
        {
            var sw = new Stopwatch();

            int successes = 0;

            var rex = new Regex(regex);

            sw.Start();
            foreach (var str in strings)
            {
                if (rex.Match(str).Success)
                {
                    successes++;
                }
            }
            sw.Stop();

            Console.Write("Regex {0,-12} took {1} result: {2}/{3}", regex, sw.Elapsed,
successes, strings.Count);
```

```
            return sw.Elapsed;
        }
    }
}
```

c# regex performance

edited May 22 '13 at 21:06
Peter Mortensen
14.6k 19 89 118

asked May 18 '13 at 7:18
weston
40.9k 17 104 179

172 Maybe `\d` deals with locales. E.g. Hebrew uses letters for digits. – Barmar May 18 '13 at 7:20

6 related: stackoverflow.com/a/6479605/674039 – wim May 18 '13 at 15:04

35 This is an interesting question precisely because `\d` does not mean the same thing in different languages. In Java, for example `\d` does indeed match 0-9 only – Ray Toal May 18 '13 at 17:59

17 @Barmar Hebrew does not use letters for digits normally, rather the same latin numeral digits [0-9]. Letters can be substituted for digits, but this is a rare use and reserved for special terms. I would not expect a regex parser to match כ"ג יורדי סירה (with כ"ג being a substitue for 23). Also, as can be seen in Sina Iravanian's answer, Hebrew letters do not appear as valid matches for \d. – Yuval Adam May 20 '13 at 9:20

7 Porting weston's code to Java yields: -- Regex \d took 00:00:00.043922 result: 4912/10000 -- Regex [0-9] took 00:00:00.073658 result: 4912/10000 167% of first -- Regex [0123456789] took 00:00:00.085799 result: 4912/10000 195% of first – Lunchbox May 22 '13 at 16:35

5 Answers

1536 ✓

`\d` checks all Unicode digits, while `[0-9]` is limited to these 10 characters. For example, Persian digits, ۱۲۳۴۵۶۷۸۹ , are an example of Unicode digits which are matched with `\d` , but not `[0-9]` .

You can generate a list of all such characters using the following code:

```
var sb = new StringBuilder();
for(UInt16 i = 0; i < UInt16.MaxValue; i++)
{
    string str = Convert.ToChar(i).ToString();
    if (Regex.IsMatch(str, @"\d"))
```

```
        sb.Append(str);
}
Console.WriteLine(sb.ToString());
```

Which generates:

> 0123456789߉߈߇߆߅߄߃߂߁߀٠١٢٣٤٥٦٧٨٩۰۱۲۳۴۵۶۷۸۹०१२३४५६७८९০১২৩৪৫৬৭৮৯੦੧੨੩੪੫੬੭੮੯૦૧૨૩૪૫૬૭૮૯୦୧୨୩୪୫୬୭୮୯௦௧௨௩௪௫௬௭௮௯౦౧౨౩౪౫౬౭౮౯೦೧೨೩೪೫೬೭೮೯൦൧൨൩൪൫൬൭൮൯๐๑๒๓๔๕๖๗๘๙໐໑໒໓໔໕໖໗໘໙༠༡༢༣༤༥༦༧༨༩၀၁၂၃၄၅၆၇၈၉႐႑႒႓႔႕႖႗႘႙០១២៣៤៥៦៧៨៩᠐᠑᠒᠓᠔᠕᠖᠗᠘᠙□□□□□□□□□□᧐᧑᧒᧓᧔᧕᧖᧗᧘᧙□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□꤀꤁꤂꤃꤄꤅꤆꤇꤈꤉꩐꩑꩒꩓꩔꩕꩖꩗꩘꩙□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□０１２３４５６７８９

edited Oct 13 '16 at 17:35
dakab
3,664 8 30 49

answered May 18 '13 at 7:24
Sina Iravanian
13.6k 4 25 43

116 Here is a more complete list of digits that aren't 0-9: fileformat.info/info/unicode/category/Nd/list.htm – Robert McKee May 18 '13 at 7:29

8 @weston Unicode has 17 planes with 16 bits each. Most important characters are in the basic plane, but some special characters, mostly Chinese, are in the supplemental planes. Dealing with those in C# is a bit annoying. – CodesInChaos May 18 '13 at 7:55

9 @RobertMcKee: Nitpick: The full unicode character set is actually 21 bit (17 planes of 16 bit each). But of course a 21-bit-datatype is impractical, so if you use a power-of-2 datatype, it's true that you need 32 bit. – sleske May 18 '13 at 21:32

3 According to this Wikipedia article, the Unicode Consortium has stated that the limit of 1,114,112 code points (0 to 0x010FFFF) will never be changed. It links to unicode.org, but I didn't find the statement there (I probably just missed it). – Keith Thompson May 20 '13 at 2:50

13 It'll never be changed -- until they need to change it. – Robert McKee Jul 8 '13 at 21:00

265

Credit to ByteBlast for noticing this in the docs. Just changing the regex constructor:

```
var rex = new Regex(regex, RegexOptions.ECMAScript);
```

Gives new timings:

```
Regex \d           took 00:00:00.1355787 result: 5077/10000
Regex [0-9]        took 00:00:00.1360403 result: 5077/10000  100.34 % of first
Regex [0123456789] took 00:00:00.1362112 result: 5077/10000  100.47 % of first
```

answered May 18 '13 at 9:37
weston
40.9k 17 104 179

11 What does the `RegexOptions.ECMAScript` do? – laurent May 20 '13 at 1:36

6 From Regular Expression Options: "Enable ECMAScript-compliant behavior for the expression." – chrisaycock May 20 '13 at 1:58

84 Effectively, I think it removes support for Unicode. – 0xFE May 20 '13 at 3:33

28 @0xFE: Not quite. Unicode escapes are still valid in `ECMAScript` (`\u1234`). It's "just" the shorthand character classes that change meaning (like `\d`) and the Unicode property/script shorthands that go away (like `\p{N}`). – Tim Pietzcker May 20 '13 at 9:51

9 This is not an answer to the "why" part. It is a "fix the symptoms" answer. Still valuable information. – usr May 29 '13 at 16:52

116

From Does “\d” in regex mean a digit?:

> `[0-9]` isn't equivalent to `\d` . `[0-9]` matches only `0123456789` characters, while `\d` matches `[0-9]` and other digit characters, for example Eastern Arabic numerals `٠١٢٣٤٥٦٧٨٩`

edited May 23 '17 at 11:47
Community ♦
1 1

answered May 18 '13 at 7:27
İsmet Alkan
4,535 3 32 64

48 According to: msdn.microsoft.com/en-us/library/20bw873z.aspx `If ECMAScript-compliant behavior is specified, \d is equivalent to [0-9].` – User 12345678 May 18 '13 at 7:30

2 huh, am i wrong or this sentence from the link is telling the opposite. "\d matches any decimal digit. It is equivalent to the \p{Nd} regular expression pattern, which includes the standard decimal digits 0-9 as well as the decimal digits of a number of other character sets." – İsmet Alkan May 18 '13 at 7:51

3 @ByteBlast thanks, using the constructor: `var rex = new Regex(regex, RegexOptions.ECMAScript);` makes them all pretty much indistinguishable in performance terms. – weston May 18 '13 at 7:53

2 oh anyway, thanks everyone. this question turned out to be a great learning for me. – İsmet Alkan May 18 '13 at 7:54

3 Please don't "just copy" answers from other questions. If the question is a duplicate, flag it as such. – BoltClock ♦ May 18 '13 at 12:00

17

An addition to top answer from Sina Iravianian, here is a .NET 4.5 version (since only that version supports UTF16 output, c.f. the first three lines) of his code, using the full range of Unicode code points. Due to the lack of proper support for higher Unicode planes, many people are not aware of always checking for and including the upper Unicode planes. Nevertheless they sometimes do contain some important characters.

Update

Since `\d` does not support non-BMP characters in regex (thanks xanatos), here a version that uses the Unicode character database

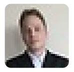
```
public static void Main()
{
    var unicodeEncoding = new UnicodeEncoding(!BitConverter.IsLittleEndian, false);
    Console.InputEncoding = unicodeEncoding;
    Console.OutputEncoding = unicodeEncoding;

    var sb = new StringBuilder();
    for (var codePoint = 0; codePoint <= 0x10ffff; codePoint++)
    {
        var isSurrogateCodePoint = codePoint <= UInt16.MaxValue
               && (  char.IsLowSurrogate((char) codePoint)
                  || char.IsHighSurrogate((char) codePoint)
                  );

        if (isSurrogateCodePoint)
            continue;

        var codePointString = char.ConvertFromUtf32(codePoint);

        foreach (var category in new []{
        UnicodeCategory.DecimalDigitNumber,
            UnicodeCategory.LetterNumber,
            UnicodeCategory.OtherNumber})
        {
        sb.AppendLine($"{category}");
            foreach (var ch in charInfo[category])
        {
                sb.Append(ch);
            }
            sb.AppendLine();
        }
    }
    Console.WriteLine(sb.ToString());

    Console.ReadKey();
}
```

Yielding the following output:

```
DecimalDigitNumber
0123456789٠١٢٣٤٥٦٧٨٩۰۱۲۳۴۵۶۷۸۹߀߁߂߃߄߅߆߇߈߉०१२३४५६७८९০১২৩৪৫৬৭৮৯੦੧੨੩੪੫੬੭੮੯૦૧૨૩૪૫૬૭૮૯୦୧୨୩୪୫୬୭୮୯௦௧௨௩௪௫௬௭௮௯
౦౧౨౩౪౫౬౭౮౯೦೧೨೩೪೫೬೭೮೯൦൧൨൩൪൫൬൭൮൯๐๑๒๓๔๕๖๗๘๙໐໑໒໓໔໕໖໗໘໙༠༡༢༣༤༥༦༧༨༩၀၁၂၃၄၅၆၇၈၉႐႑႒႓႔႕႖႗႘႙
០១២៣៤៥៦៧៨៩᠐᠑᠒᠓᠔᠕᠖᠗᠘᠙᥆᥇᥈᥉᥊᥋᥌᥍᥎᥏᧐᧑᧒᧓᧔᧕᧖᧗᧘᧙᪀᪁᪂᪃᪄᪅᪆᪇᪈᪉□□□□□□□□□□꘠꘡꘢꘣꘤꘥꘦꘧꘨꘩□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□꣐꣑꣒꣓꣔꣕꣖꣗꣘꣙꤀꤁꤂꤃꤄꤅꤆꤇꤈꤉□□□□□□□□□□□□□□□□□□□□꧐꧑꧒꧓꧔꧕꧖꧗꧘꧙꩐꩑꩒꩓꩔꩕꩖꩗꩘꩙꯰꯱꯲꯳꯴꯵꯶꯷꯸꯹□□
□□□□□□□□𐒠𐒡𐒢𐒣𐒤𐒥𐒦𐒧𐒨𐒩０１２３４５６７８９
𑁦𑁧𑁨𑁩𑁪𑁫𑁬𑁭𑁮𑁯𑃰𑃱𑃲𑃳𑃴𑃵𑃶𑃷𑃸𑃹𑄶𑄷𑄸𑄹𑄺𑄻𑄼𑄽𑄾𑄿□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
𝟓𝟔𝟕𝟖𝟗𝟶𝟷𝟸𝟹𝟺𝟻𝟼𝟽𝟾𝟿

LetterNumber

ᛮᛯᛰⅠⅡⅢⅣⅤⅥⅦⅧⅨⅩⅪⅫⅬⅭⅮⅯⅰⅱⅲⅳⅴⅵⅶⅷⅸⅹⅺⅻⅼⅽⅾⅿↀↁↂↅↆↇↈ〇〡〢〣〤〥〦〧〨〩〸〹〺
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□𒐀𒐁𒐂𒐃𒐄𒐅𒐆𒐇𒐈𒐉𒐊
𒐋𒐌𒐍𒐎𒐏𒐐𒐑𒐒𒐓𒐔𒐕𒐖𒐗𒐘𒐙𒐚𒐛𒐜𒐝𒐞𒐟𒐠𒐡𒐢𒐣𒐤𒐥𒐦𒐧𒐨𒐩𒐪𒐫𒐬𒐭𒐮𒐯𒐰𒐱𒐲𒐳𒐴𒐵𒐶𒐷𒐸𒐹𒐺𒐻𒐼𒐽𒐾𒐿𒑀𒑁𒑂𒑃𒑄𒑅𒑆𒑇𒑈𒑉𒑊𒑋𒑌𒑍𒑎𒑏𒑐𒑑𒑒𒑓𒑔𒑕𒑖𒑗𒑘𒑙𒑚𒑛𒑜𒑝𒑞𒑟𒑠𒑡𒑢

OtherNumber
²³¹¼½¾৴৵৶৷৸৹୲୳୴୵୶୷௰௱௲౸౹౺౻౼౽౾൰൱൲൳൴൵༪༫༬༭༮༯༰༱༲༳፩፪፫፬፭፮፯፰፱፲፳፴፵፶፷፸፹፺፻፼៰៱៲៳៴៵៶៷៸៹᧚⁰⁴⁵⁶⁷⁸⁹₀₁₂₃₄₅₆₇₈₉⅐⅑⅒⅓⅔⅕⅖⅗
⅘⅙⅚⅛⅜⅝⅞⅟↉①②③④⑤⑥⑦⑧⑨⑩⑪⑫⑬⑭⑮⑯⑰⑱⑲⑳⑴⑵⑶⑷⑸⑹⑺⑻⑼⑽⑾⑿⒀⒁⒂⒃⒄⒅⒆⒇⒈⒉⒊⒋⒌⒍⒎⒏⒐⒑⒒⒓⒔⒕⒖⒗⒘⒙⒚⒛
⓪⓫⓬⓭⓮⓯⓰⓱⓲⓳⓴⓵⓶⓷⓸⓹⓺⓻⓼⓽⓾⓿❶❷❸❹❺❻❼❽❾❿➀➁➂➃➄➅➆➇➈➉➊➋➌➍➎➏➐➑➒➓⳽㆒㆓㆔㆕㈠㈡㈢㈣㈤㈥㈦㈧㈨㈩
㉈㉉㉊㉋㉌㉍㉎㉏㉑㉒㉓㉔㉕㉖㉗㉘㉙㉚㉛㉜㉝㉞㉟㊀㊁㊂㊃㊄㊅㊆㊇㊈㊉㊱㊲㊳㊴㊵㊶㊷㊸㊹㊺㊻㊼㊽㊾㊿
꠰꠱꠲꠳꠴꠵□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□𐅈𐅉𐅊𐅋□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□𐌠𐌡𐌢𐌣□□□□□□□□□□□□□□□□□□□□□□□□□□□□𐡘𐡙𐡚𐡛𐡜𐡝𐡞𐡟
□□□□□□□□□□□□□□□□□□□□□□□□□𐤖𐤗𐤘𐤙𐤚𐤛𐩀𐩁𐩂𐩃𐩄𐩅𐩆𐩇𐩽𐩾𐭘𐭙𐭚𐭛𐭜𐭝𐭞𐭟□□□□□□□□□□𐭸𐭹𐭺𐭻𐭼𐭽𐭾𐭿□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□𐹠𐹡𐹢𐹣𐹤𐹥𐹦𐹧𐹨𐹩𐹪𐹫𐹬𐹭𐹮𐹯𐹰𐹱𐹲𐹳𐹴𐹵𐹶𐹷𐹸𐹹𐹺𐹻𐹼𐹽𐹾𑁒𑁓𑁔𑁕𑁖𑁗𑁘𑁙𑁚𑁛𑁜𑁝𑁞𑁟𑁠𑁡𑁢𑁣𑁤𑁥□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□🄀🄁🄂🄃🄄🄅🄆🄇🄈🄉🄊□□□□□□□□□□□🄋🄌
```

edited Apr 12 '18 at 8:33

answered Sep 13 '13 at 8:24



Sebastian
2,770 4 27 45

The sad thing is that the Win32 Console does not display astral characters – Sebastian May 27 '14 at 21:57

4 If I remember corretly, sadly in .NET `Regex` doesn't support non-BMP characters. So in the end checking for characters > 0xffff with a regex is useless. – xanatos Apr 12 '17 at 10:19

0

\d checks all Unicode, while [0-9] is limited to these 10 characters. If just 10 digits, you should use. Others I recommend using \d，Because writing less.

answered Mar 11 '16 at 10:27

dengkai

1 2