O predstavitvi podatkov v računalniku: besedilo

↓↓↓
Jure Slak

→ **Velik del podatkov, ki jih shranjujemo na računalniku ali pošiljamo prek interneta, je sestavljen iz običajnega besedila: elektronska pošta, klepet, dokumenti, spletne strani. Vsaka prikazana vrstica besedila, ki jo vidite na zaslonu, pa je morala biti tudi nekje shranjena kot zaporedje ničel in enic. Kot se pogosto zgodi, lahko besedilo shranimo na več kot en način; ti različni načini pa med sabo niso kompatibilni. Če ste kdaj npr. pri podnapisih pri filmu videli élovek namesto človek, ali pa morda kar potegav**   **ina namesto potegavščina, potem ste naleteli na takšno nekompatibilnost med načini kodiranja.**

ASCII

Ko so v angleško govorečih državah v 60-ih letih prejšnjega stoletja začeli z bolj resnim računalništvom, so potrebovali način zapisovanja besedila. Naravno so se odločili, da bodo besedilo zapisovali znak po znak, vsak znak pa bo predstavljen s fiksnim številom bitov. Da bi ugotovili, koliko bitov potrebujemo, moramo prešteti, koliko različnih znakov potrebujemo. Za začetek potrebujemo velike in male črke (dvakrat po 26), številke (10), nekaj ločil za pisanje, finance in matematiko (npr. +−*/.?!@#$%^'" &*()\~), znake za prazen prostor (*white space*), kot so presledek, znak za novo vrstico, znaki za zamike. Na koncu so pristali na 128 znakih, kar lahko spravimo ravno v sedem bitov ($2^7 = 128$); ta standard so poimenovali ASCII (American Standard Code for Information Interchange), kar preberemo áski. Natančno je to kodiranje napisano v tabeli 1. Opazimo lahko, da imamo do znaka 32 kontrolne znake in znake za prazen prostor (ali t. i. bele znake). Znak 32 je npr. presledek (*space*), znak 10 je znak za novo vrstico (*line feed*), znak 7 je zvonec (*bell*), ki je ob izpisu povzročil, da se je oglasil zveneč ding v računalniku. Tudi dandanašnji lahko z nekaj truda računalnike še vedno prisilimo v to. Za razlago vseh znakov se je najbolje posvetovati s kakšno knjigo [6] ali Wikipedijo [1], marsikateri od njih niso več v redni rabi. Po kontrolnih in belih znakih pa sledi rahlo čudna mešanica: nekaj ločil, nato številke, nato spet nekaj ločil, pa velike črke in spet ločila, nato male črke in še malo ločil. Zakaj ne bi dali vsa ločila na kup? Odgovor se najde v dvojiškem zapisu: če pogledamo kode za 0 do 9, vidimo, da imajo po vrsti kode 011**0000** do 011**1001**. Če ignoriramo predpono 011, so to ravno dvojiški zapisi za števila od 0 do 9. Podobno velja pri velikih črkah: A ima kodo 10**00001**, B ima kodo 10**00010** itd. Če ignoriramo predpono 10, je kodiranje zelo naravno: A kodiramo kot 1, B kot 2; tako nadaljujemo do Z kot 10**11010** oz. 26. Tako je mogoče tudi iz dvojiškega zapisa relativno enostavno razbrati besedilo. Podobno velja za male črke, le da imajo predpono 11 namesto 10. Za ločila, za katera ni tako pomembno, kako so zakodirana, so porabili preostale proste kode. Dodatni lepi lastnosti ASCII kodiranja sta, da lahko male črke med seboj primerjamo po abecedi tako, da le primerjamo njihove številske vrednosti, in da lahko malo črko spremenimo v veliko s spremembo enega samega bita.

Divji zahod stoterih standardov

Sočasno s širjenjem računalnikov po svetu se je širila tudi potreba po podpori za druge znake, kot npr. za germanske ä, ö, ü, slovanske ć, č, ś, š, ź, ž, romansko ç in podobno. Poleg tega so računalniki standardno delali z velikostjo bajta, osem bitov, kar pomeni, da je bilo na voljo dodatnih 128 znakov, če bi velikost kodiranja povečali iz sedem na osem bitov.

BIN	DEC	znak	BIN	DEC	znak	BIN	DEC	znak
0000000	000	NUL	0101011	043	+	1010110	086	V
0000001	001	SOH	0101100	044	,	1010111	087	W
0000010	002	STX	0101101	045	-	1011000	088	X
0000011	003	ETX	0101110	046	.	1011001	089	Y
0000100	004	EOT	0101111	047	/	1011010	090	Z
0000101	005	ENQ	0110000	048	0	1011011	091	[
0000110	006	ACK	0110001	049	1	1011100	092	\
0000111	007	BEL	0110010	050	2	1011101	093	]
0001000	008	BS	0110011	051	3	1011110	094	^
0001001	009	HT	0110100	052	4	1011111	095	_
0001010	010	LF	0110101	053	5	1100000	096	‘
0001011	011	VT	0110110	054	6	1100001	097	a
0001100	012	FF	0110111	055	7	1100010	098	b
0001101	013	CR	0111000	056	8	1100011	099	c
0001110	014	SO	0111001	057	9	1100100	100	d
0001111	015	SI	0111010	058	:	1100101	101	e
0010000	016	DLE	0111011	059	;	1100110	102	f
0010001	017	DC1	0111100	060	<	1100111	103	g
0010010	018	DC2	0111101	061	=	1101000	104	h
0010011	019	DC3	0111110	062	>	1101001	105	i
0010100	020	DC4	0111111	063	?	1101010	106	j
0010101	021	NAK	1000000	064	@	1101011	107	k
0010110	022	SYN	1000001	065	A	1101100	108	l
0010111	023	ETB	1000010	066	B	1101101	109	m
0011000	024	CAN	1000011	067	C	1101110	110	n
0011001	025	EM	1000100	068	D	1101111	111	o
0011010	026	SUB	1000101	069	E	1110000	112	p
0011011	027	ESC	1000110	070	F	1110001	113	q
0011100	028	FS	1000111	071	G	1110010	114	r
0011101	029	GS	1001000	072	H	1110011	115	s
0011110	030	RS	1001001	073	I	1110100	116	t
0011111	031	US	1001010	074	J	1110101	117	u
0100000	032	SP	1001011	075	K	1110110	118	v
0100001	033	!	1001100	076	L	1110111	119	w
0100010	034	"	1001101	077	M	1111000	120	x
0100011	035	#	1001110	078	N	1111001	121	y
0100100	036	$	1001111	079	O	1111010	122	z
0100101	037	%	1010000	080	P	1111011	123	{
0100110	038	&	1010001	081	Q	1111100	124	|
0100111	039	’	1010010	082	R	1111101	125	}
0101000	040	(	1010011	083	S	1111110	126	~
0101001	041	)	1010100	084	T	1111111	127	DEL
0101010	042	*	1010101	085	U			

TABELA 1.
Kodirna tabela ASCII.

Najbolj priljubljena izmed razširitev je bila Windows-1252, ki je dodala večino znakov, potrebnih za evropske jezike, nekaj dodatnih ločil in matematičnih simbolov. Dodani so bili naslednji znaki:

€‚ƒ„…†‡ˆ‰Š‹ŒŽ‘’“”•–—˜™š›œžŸ¡¢£¤¥¦§¨©ª«¬®¯
°±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×
ØÙÚÛÜÝÞßàáâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþÿ

Opazite lahko, da so od slovenskih črk dodane š, Š, ž, Ž, ne pa č in Č. Kljub temu se je Windows-1252 prijel in je bil privzet način kodiranja na sistemih Windows vse do leta 2018. Še zdaleč pa ni bil Windows-1252 edini standard kodiranja, ki je razširil ASCII. Nam bližje je bil standard Windows-1250, ki je vseboval vse slovenske črke. Ta je ASCII razširil z naslednjimi znaki:

€‚„…†‡‰Š‹ŚŤŽŹ‘’“”•–—™š›śťžź ˇ˘Ł¤Ą¦§¨©Ş«¬®Ż
°±˛ł´µ¶·¸ąş»Ľ˝ľżŔÁÂĂÄĹĆÇČÉĘËĚÍÎĎĐŃŇÓÔŐÖ×Ř
ŮÚŰÜÝŢßŕáâăäĺćçčéęëěíîďđńňóôőö÷řůúűüýţ˙

Nesrečno slaven je primer znaka č, ki mu je v Windows-1250 tabeli dodeljena enaka številka kot znaku é v Windows-1252, od koder prihajajo težave s prikazovanjem č pri podnapisih. Če predvajalnik dobi zaporedje bajtov

- 01110000 01101111 11101000 01100101 01110000.

lahko to interpretira na dva načina. Če ga interpretira v privzetem kodiranju Windows-1252, bo znak s kodo 11101000 (232) predstavljal é in bo besedo poéep, če pa interpretira v kodiranju Windows-1250, pa bo znak s kodo 232 enak č in bo besedo prikazano kot počep. Ena možna rešitev, da se znaki é pri podnapisih prikazujejo kot č, je, da se v predvajalniku nastavi privzeto kodiranje na Windows-1250.

Ko program dobi podatke, ki so zapisani kot zaporedje ničel in enic, ter mora to pretvoriti v besedilo, ne ve, katero kodiranje uporabiti. Privzeto kodiranje ni vedno pravilno, če je bila datoteka npr. prinešena s sistema z drugačnim privzetim kodiranjem ali pa če je prišla od uporabnika na drugem koncu sveta. To povzroči težave, saj je datoteka interpretirana v najboljšem primeru z nekaj napačnimi znaki, v najslabšem pa je povsem neberljiva. Mnogo medsebojno nekompatibilnih standardov, ki so bili posledica tega, da 256 mest še vedno ni bilo dovolj niti za vse evropske jezike (kaj šele azijske), je en razlog za uvedbo enega samega standarda, ki bi zadovoljil vse.

Unicode

V letu 1991 je bil ustanovljen Konzorcij Unicode[4], ki je neprofitna organizacija, v kateri danes sodeluje mnogo dovolj vplivnih podjetij, kot so Adobe, Apple, Facebook, Google, IBM, Microsoft, Netflix in SAP. Njihova naloga je, da določajo, kateri znaki bodo na voljo v Unicode naboru znakov in kako naj se znaki obnašajo pri urejanju, pri pretvarjanju v velike črke in podobno.

Unicode podpira 159 pisav[3] (latinico, cirilico, grške, kitajske, arabske znake) in mnogo simbolov, kot so ločila, številke, matematični in tehnični simboli, puščice, glasbeni simboli. Vsak znak ima standardno ime, številko in opis, znake pa lahko raziskujemo s pomočjo spletne strani `www.compart.com/en/unicode/`.

Celotna zbirka znakov je razdeljena na bloke, ki predstavljajo zaključene celote znakov, ki sodijo skupaj. Prvi blok, s številkami od 0 do 127, je Osnovna latinica (*ang.* Basic Latin), ki vsebuje natanko enake znake kot ASCII. Nato sledi blok Dodatek latin-1 (latin-1 supplement), ki vsebuje naslednjih 128 mest z nekaj dodatnimi znaki, kot so ± in ü. Sledi blok Razširitev latinice-A (*ang.* Latin Extended-A), kjer je veliko črk z različnimi okraski, med drugim tudi slovenske čšž. Če želimo, da naša izbrana pisava podpira znake čšž, moramo izbirati med pisavami, ki podpirajo blok Razširitev latinice-A. Vseh blokov v trenutnem standardu (14.0) je 308, vseh znakov pa 144.697. Vseh mest, ki so na voljo, je 1 112 064.

Poglejmo nekaj primerov znakov, ki bodo zanimivi kasneje. Kode znakov se pogosto navaja v šestnajstiškem zapisu oblike U+1234, kjer (če zapis vidimo izven konteksta) predpona U+ pove, da gre za zapis Unicode kode znaka (*Unicode code point*).

znak A

Formalni opis: latinična velika črka A.
Verzija: dodano v verziji 1.1 (Junij 1993).
Blok: osnovna latinica.
Pisava: latinica.
Kategorija: velike črke.
Dvosmerno pisanje: iz leve proti desni.
Številka: 65 (U+0041).

znak λ
Formalni opis: mala grška črka lambda.
Verzija: dodano v verziji 1.1 (junij 1993).
Blok: grški in koptski znaki.
Pisava: grška.
Kategorija: male črke.
Dvosmerno pisanje: od leve proti desni.
Številka: 955 (U+03BB).

znak 𐡀
Formalni opis: samarijanska črka it.
Verzija: dodano v verziji 5.2 (oktober 2009).
Blok: samarijanski znaki.
Pisava: samarijanska.
Kategorija: druge črke.
Dvosmerno pisanje: od desne proti levi.
Številka: 2055 (U+0807).

znak ♪
Formalni opis: glasbeni simbol za noto osminko.
Verzija: dodano v verziji 3.1 (marec 2001).
Blok: glasbeni simboli.
Pisava: nedoločena.
Kategorija: drugi simboli.
Dvosmerno pisanje: od leve proti desni.
Številka: 119136 (U+1D160).

znak 😱
Formalni opis: obraz, kričoč v strahu
Verzija: dodano v verziji 6.0 (marec 2010).
Blok: emotikoni.
Pisava: nedoločena.
Kategorija: drugi simboli.
Dvosmerno pisanje: nedoločeno.
Številka: 128561 (U+1F631).

Trenutni standard 14.0 je izšel septembra 2021 in je dodal 838 znakov, med drugim emotikon obraz s poševnimi usti.

UTF-8

Do sedaj smo opisali, kako je strukturiran nabor znakov Unicode, nismo pa povedali, kako se te znake dejansko predstavi oz. kodira z biti. Obstaja več načinov kodiranja znakov Unicode, od katerih je najbolj priljubljen UTF-8, ki ga trenutno uporablja 97 % vseh spletnih strani[1].

Najenostavnejši način, glede na to, da je vseh možnih znakov 1 112 064, bi bil, da uporabimo štiri bajte (32 bitov), oštevilčimo znake po vrsti in vsakega predstavimo s svojo številko. Slaba stran tega je, da bi obstoječe besedilo z ASCII znaki sedaj potrebovalo štirikrat več prostora, saj bi za A namesto 01000001 morali napisati 00000000000000000000 00001000001.

Kodiranje UTF-8 to težavo reši tako, da različne znake zakodiramo z različnim številom bajtov. Prvih 128 znakov zakodiramo z enim samim bajtom, popolnoma enako kot v kodiranju ASCII. To pomeni, da so obstoječe ASCII datoteke že veljavne UTF-8 datoteke in ni potrebnega nobenega spreminjanja vsebine ali velikosti – to je lastnost, ki je bila ključna za uspešno širitev in splošno uporabo standarda. Pojavi pa se vprašanje, kako vemo, ali moramo za dekodiranje znaka prebrati en bajt, dva, tri ali še več.

Odgovor na to je skrit v prvem bajtu samem: pogledati moramo število vodilnih enic. Če bajt nima vodilnih enic (in se torej začne z 0), ni treba prebrati nič dodatnih bajtov in je znak predstavljen s samo enim bajtom. Primer tega je znak A: **0**1000001. Če se bajt začne na dve enici, moramo prebrati še en naslednji bajt, če se začne na tri, moramo prebrati še naslednja dva bajta, če se značne na štiri enice, preberemo še naslednje tri. Drugače povedano, veljavna zaporedja bajtov so oblike

```
0xxxxxxx
110xxxxx 10xxxxxx
1110xxxx 10xxxxxx 10xxxxxx
11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
```

Bajti, ki sledijo npr. 1110xxxx, se začnejo z 10, zato da vemo, da spadajo v kontekst več bajtov, tudi če jih pogledamo same po sebi, saj se noben drug bajt ne začne z 10.

Vzemimo npr. zaporedje bajtov

- 01110000 01101111 11000100 10001101 01100101 01110000.

Prvi bajt se začne na 0 in sam predstavlja črko, ki je enaka svoji ASCII vrednosti p. Enako velja za dru-

[1] w3techs.com/technologies/cross/character_encoding/ranking

gega, ki je za 1 manjši in predstavlja o. Nato sledi bajt, ki se začne na dve enici, in bo torej naslednji znak predstavljen z dvema bajtoma, zato preberemo še enega. Bajta 110**00100** 10**001101** predstavljata znak z vrednostjo 00100001101 (vrednost smo dobili tako, da smo skupaj zložili bite, ki ležijo na mestih, označenih z x), kar je 269, oz., če bi pogledali v Unicode tabelo, je to č. Sledita še dva bajta, ki se vsak začne z 0 in predstavljata ASCII črki e in p. Ponovno smo zakodirali besedo počep, ki pa je v nasprotju z Windows-1250 porabila en bajt več za kodiranje č, kar pa je majhna cena za to, da imamo eno kodiranje za vse. Žal pa tukaj naletimo tudi na prvo nekompatibilnost: UTF-8 in Windows-1250 posebne znake predstavita različno, celo tako različno, da zapis besede počep v Windows-1250 kodiranju ni veljaven UTF-8 in bi pri branju sprožil napako. V prehodnem času, ko se je Unicode še uveljavljal, je to uporabnikom povzročalo kar nekaj težav, saj so operacijski sistemi imel pogosto težave z napačnim interpretiranjem Unicode datotek. Da bi bilo lažje prepoznati Unicode datoteke, so uvedli BOM (*byte order mark*), poseben znak s kodo U+FEFF, ki ga je bilo v nekaterih kodiranjih (ne pa v UTF-8) obvezno dati na začetek datoteke. Dandanes se tega ne počnemo več in ta znak opuščamo v UTF-8 kodiranju, saj povzroča nekompatibilnosti z ASCII, večina programske opreme pa se je tudi posodobila za delo z UTF-8. Ima pa BOM zanimivo vlogo na spletu in v UTF-16 datotekah [2].

Tudi za grško črko lambda s kodo 955 velja, da jo lahko zakodiramo z dvema bajtoma, samarijanska črka it s kodo 2055 pa potrebuje že tri bajte. Znaki, ki so še bolj proti koncu tabele, kot npr. nota osminka, razne pismenke in emotikoni, pa potrebujejo štiri bajte. Kot primer, emotikon obraz, kričoč v strahu s kodo 128561 (binarno 11111011000110001) se zakodira kot

- 11110000 10011111 10011000 10110001.

Še ena lepa lastnost UTF-8 kodiranja je, da majhno število posebnih znakov le malo spremeni velikost datoteke, saj je tisti znak zapisan z več bajti, ostali pa zavzamejo enako prostora kot prej. To pri različnih računalniških sistemih ne velja vedno. Primer tega so SMS sporočila: čim vključimo samo en poseben znak, se celotno sporočilo kodira v drugačnem (UCS-2) kodiranju kot običajno, in vsi znaki zavzamejo 16 bitov, število preostalih znakov, ki jih lahko napišemo v isti SMS, pa se kar naenkrat močno zmanjša. Nekateri narodi, predvsem azijski, katerih znaki, ki jih redno uporabljajo, so bolj proti koncu Unicode tabele, morajo kljub temu skoraj za vsak znak porabiti štiri bajte.

Poleg UTF-8, sta znana načina kodiranja Unicode tabele še UTF-16 in UTF-32. UTF-16 kodira večino znakov s 16 biti, tiste bolj proti koncu pa z dvema paroma 16 bitov. Kot posledica kodiranja s 16 biti pride nekompatibilnost z ASCII, zato na spletu kodiranje ni priljubljeno, ga pa kljub temu uporabljamo za interno predstavitev nizov v programskih jezikih Java in Javascript. UTF-32 je enostavno kodiranje fiksne širine, kjer vsak znak zakodiramo z 32 biti. Je najbolj enostavno izmed vseh, vendar ga zaradi potratnega prostora redko uporabljamo.

Dodatne lastnosti

Poleg nabora znakov standard Unicode opisuje tudi, kako se posamezni znaki kombinirajo, kako so med seboj povezani in kako se jih izriše. Primer tega so razne arabske pisave, ki nimajo koncepta črk, kot jih imamo mi, poleg tega pa se besedilo piše od desne proti levi. Nam najbližja razlaga kombiniranja je, da lahko znak č napišemo kot kombinacijo znaka c in strešice. Seveda lahko č napišemo direktno kot 11000100 10001101, toda Unicode podpira tudi, da ga napišemo kot kombinacijo c (koda 99) in strešice (koda 708), ali z biti:

- 01100011 11001100 10001100.

Oba načina sta veljavna za zapis znaka č in bosta prikazana enako. Kljub temu pa nista enaka. Ena razlika je, da če pri kombiniranem zapisu zbrišemo zadnji znak s tipko vračalko (*Backspace*), se odstrani samo strešica, pri prvem pa celotna črka. Prav tako nista enaka, če ju primerjamo direktno, kot zaporedje bitov. Nalsednja Python koda to demonstrira:

```
>>> c1 = bytes([0b01100011, 0b11001100,
        0b10001100]).decode('utf-8')
>>> c1
'č'
>>> c2 = bytes([0b11000100, 0b10001101]).
        decode('utf-8')
```

```
>>> c2
'č'
>>> c1 == c2
False
```

Za smiselno delo z nizi, ki vsebujejo Unicode znake, moramo uporabljati posebne knjižnice, ki podpirajo več zapisov pomensko enake črke. Prav tako je potrebno te knjižnice uporabljati za pravilno pretvarjanje med malimi in velikimi črkami, za urejanje in podobno, saj naivne rešitve zaradi raznolikosti svetovnih jezikov skoraj nikoli niso pravilne.

Kljub temu, da Unicode vsebuje mnogo znakov, so del kodne tabele namenoma pustili prazen, kjer lahko uporabniki definirajo svoje znake (smiselno pa je zraven dodati tudi pisavo, ki te znake tudi dejansko vsebuje). Primer uporabe te lastnosti Unicode standarda je ZRCola [5], ki definira svoje fonetične simbole.

Literatura

[1] *ASCII – Wikipedia*, dostopno na en.wikipedia.org/wiki/ASCII, ogled 7. 12. 2021.

[2] *The byte-order mark (bom) in html*, dostopno na www.w3.org/International/questions/qa-byte-order-mark, ogled 7. 12. 2021.

[3] *Supported scripts*, dostopno na www.unicode.org/standard/supported.html, ogled 7. 12. 2021.

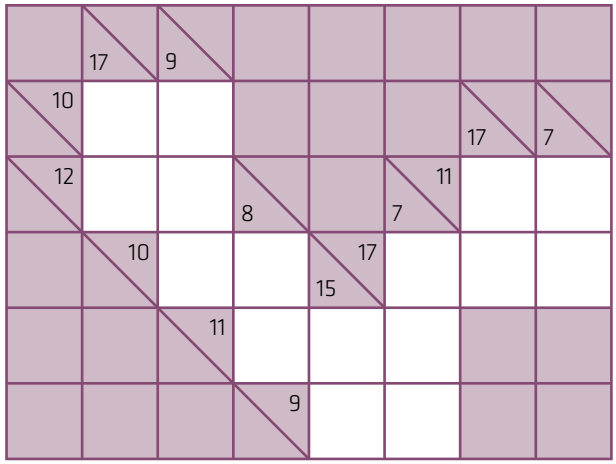
[4] *Unicode – The World Standard for Text and Emoji*, dostopno na home.unicode.org/, ogled 7. 12. 2021.

[5] *ZRCola 2 – Vnašalni sistem za jezikoslovno rabo*, dostopno na zrcola.zrc-sazu.si/, ogled 7. 12. 2021.

[6] Unicode Consortium. The Unicode standard 5.0. Addison-Wesley, version 5.0 edition, 1991–2006. Bibliografija: 1411–1417 Kazalo.

Križne vsote

→ Naloga reševalca je, da izpolni bele kvadratke s številkami od 1 do 9 tako, da bo vsota števk v zaporednih belih kvadratkih po vrsticah in po stolpcih enaka številu, ki je zapisano v sivem kvadratku na začetku vrstice (stolpca) nad (pod) diagonalo. Pri tem morajo biti vse števke v posamezni vrstici (stolpcu) različne.



REŠITEV KRIŽNE VSOTE