Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva

Kristijan Burnik

Druga domaća zadaća iz predmeta "Uvod u teoriju računarstva"

Zadatak broj 2029

Zagreb, lipanj 2010.

Druga domaća zadaća iz predmeta “Uvod u teoriju računarstva”

Student: Kristijan Burnik

Matični broj studenta: 0036444128

Zadatak broj 2029:
Konstruirati sustav (turingov stroj) za praćenje ulazno-izlaznog tijeka vozila kroz tunel Sveti Rok.

Uvod

Zadatak je konstruirati sustav za praćenje ulazno-izlaznog tijeka vozila kroz tunel Sveti Rok, naravno uz uvjet da se isti može primjeniti i na drugim lokacijama (tunelima) i cestovnim građevinama. Osnovni problem koji se riješava jest utvrđivanje da li ulazni niz vozila odgovara izlaznom nizu, primjerice ako u tunel redom uđu automobili A,B i C, tada istim redom moraju i izaći: A, B, C.

U slučaju da nije tako, možemo zaključiti dvije stvari: jedan ili više automobila je preticao u tunelu iako je to nedozovljeno, ili pak je jedan ili više automobila stalo zbog kvara ili moguće nesreće. Koji god slučaj se pojavio, važno je reagirati stoga je potrebno konstruirati pouzdani sustav koji će dojaviti osobi koja nadgleda promet da se poremetio redoslijed vozila dok su prolazila kroz tunel.

Ostvarenje – ideja i koncepti

Sustav za praćenje redoslijeda vozila ima jednostavnu funkciju: vratiti „zeleno svjetlo“ (true) ako je poredak valjan ili pokrenuti „crvenu uzbunu“ (false) ukoliko nije. Koncept ulaska i izlaska vozila iz tunela te općenito koncept *reda* kao niza elemenata u pokretu algoritamski promatramo kao **queue (red)** odnosno kao niz elemenata u memoriji digitalnog sustava koje u jednom trenutku **stavljamo na kraj reda** (metodom PUSH), a u drugom trenutku **obrađujemo onog koji je na početku reda** (metodom POP).

Struktura podataka **QUEUE** se često spominje s pripadnim konceptom **FIFO** (First In First Out) što je skraćenica koja veoma jednostavno dočarava o čemu se radi (koji element prvi uđe, taj prvi izađe). FIFO koncept najčešće rabimo onda kad znamo da neki proces traje veoma dugo ili onda kada ne znamo koliko će isti trajati, također se koristi kada je važno poštivati redoslijed radnji.

Evo primjera kako možemo primjeniti queue u (veoma kratkom) tunelu:

Vrijeme	Analogna pojava	Digitalna pojava	Stanje queue-a
09:00:00	Nema pojave, prazan tunel	NOOP	[S][E]
09:59:59	Ulazak automobila A	Push A	A [S] [E]
10:00:01	Ulazak automobila B	Push B	A B [S] [E]
10:00:20	Ulazak automobila C	Push C	A B C [S] [E]
10:00:21	Izlazak automobila A	? Pop == A	A B C [S] [E]
10:00:33	Izlazak automobila B	? Pop == B	A B C [S] [E]
10:00:35	Ulazak automobila D	Push D	A B C D [S] [E]
10:00:45	Izlazak automobila C	? Pop == C	A B C D [S] [E]
10:00:58	Izlazak automobila D	? Pop == D	A B C D [S][E]

Pojašnjenja tablice:

1. Oznakom [S] označen je početak queue-a (element koji čeka na obradu), dok je [E] kraj (mjesto rezerevirano za novi element ili signal kraja). [S] i [E] su pokazivači
2. Push metoda stavlja na kraj reda i pomiče pokazivač [E] za jedno mjesto udesno
3. Pop metoda vraća element s početka te pomiče [S] za jedno mjesto udesno
4. (? Pop == X) znači da uspoređujemo stanje reda sa stanjem na izlasku iz tunela (npr IF naredba), ukoliko ovaj uvjet nije zadovoljen to znači da se poremetio redoslijed.
5. Red je prazan ukoliko pokazivač početka i kraja gledaju na istu lokaciju ([S] == [E])

Budući je tablica prikazala kako je **FIFO koncept** valjan za ostvaranje našeg sustava, isti ćemo primjeniti za konstruiranje pripadnog automata koji će obavljati tu zadaću.

Potrebno je konstruirati turingov stroj koji će primati parove ulaznih i izlaznih grupa automobila, npr.: U I U I U I. Gdje je jedan par grupa (U, I) u analognom svijetu vremenski razdvojen T sekundi (T je prosječno vrijeme potrebno za prolazak kroz tunel), a u digitalnom sustavu je „spojen - uparen“ i prepisan na traku turingovog stroja. Grupa se sastoji od bitova koji opisuju vrstu vozila: 0 predstavlja osobni automobil, a 1 teretno vozilo.

Veličina jedne grupe odnosno broj bitova koji sačinjavaju grupu su proizvoljni što u analognom svijetu znači da u jednom periodu može ući/izaći npr. 3 automobila, dok u drugom periodu 30 automobila...

U slučaju kada neki automobil uđe u tunel, a ne izađe odmah u idućoj izlaznoj grupi to najčešće znači da mu je bilo potrebno više od T sekundi za prolazak kroz tunel ili da je ušao u tunel pri samom kraju perioda. U tom slučaju taj automobil izlazi van s idućom grupom (promatrano digitalno). Kada ovakav slučaj ne bi bio moguć tada ne bi bilo potrebno konstrutirati QUEUE, bilo bi dovoljno uspoređivati samo susjedne grupe bitova i time bi se utvrdila ispravnost poretka.

Definicija turingovog stroja za utvrđivanje ispravnosti poretka ulazno-izlaznog tijeka

Turingov stroj (s jednom trakom) koji bi ispitivao ispravnost poretka formalno se definira:
(*stanja su definirana opisnim imenima radi jednostavnosti pri referenciranju istih*)

$TS = (Q, \Sigma, \Gamma, \delta, q0, B, F);$

Skup stanja
Q = {
init,
set_queue_start0, set_queue_start1,
startpush, push0, push1, dopush0, dopush1, returnpush,
startpop, pop0, pop1, dopop0, dopop1, returnpop, acceptonempty,
accept, reject
}

Skup ulaznih znakova trake
$\Sigma = \{0, 1, \_ \}$

Skup svih znakova trake
$\Gamma = \{0, 1, \_, N, J, W, Q, |, ., B\}$

Početno stanje stroja
q_0 = init

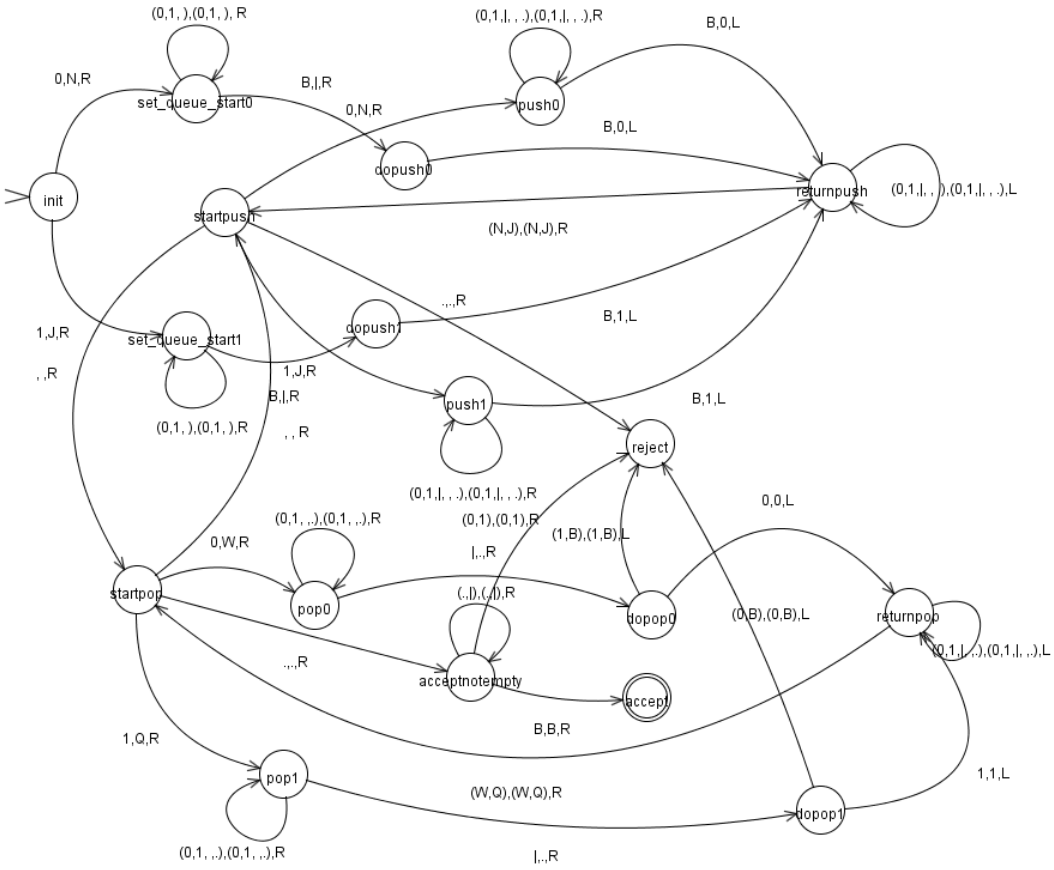
Oznaka prazne ćelije
B = „B“

Prihvatljiva stanja
F = { accept }

Funkcija prijelaza

δ(init,0) = (set_queue_start0,N,R); δ(init,1) = (set_queue_start1,J,R); δ(init,) = (init, ,R);	δ(set_queue_start0,0) = (set_queue_start0,0,R); δ(set_queue_start0,1) = (set_queue_start0,1,R); δ(set_queue_start0,) = (set_queue_start0, ,R); δ(set_queue_start0,B) = (dopush0,|,R);	δ(set_queue_start1,0) = (set_queue_start1,0,R); δ(set_queue_start1,1) = (set_queue_start1,1,R); δ(set_queue_start1,) = (set_queue_start1, ,R); δ(set_queue_start1,B) = (dopush1,|,R);	δ(dopush0,B) = (returnpush,0,L);
δ(dopush1,B) = (returnpush,1,L);	δ(returnpush,|) = (returnpush,|,L); δ(returnpush,0) = (returnpush,0,L); δ(returnpush,1) = (returnpush,1,L); δ(returnpush,) = (returnpush, ,L); δ(returnpush,.) = (returnpush,.,L); δ(returnpush,N) = (startpush,N,R); δ(returnpush,J) = (startpush,J,R);	δ(startpush,0) = (push0,N,R); δ(startpush,1) = (push1,J,R); δ(startpush,) = (startpop, ,R); δ(startpush,.) = (reject,.,R);	δ(push0,0) = (push0,0,R); δ(push0,1) = (push0,1,R); δ(push0,) = (push0, ,R); δ(push0,|) = (push0,|,R); δ(push0,.) = (push0,.,R); δ(push0,B) = (returnpush,0,L);
δ(push1,0) = (push1,0,R); δ(push1,1) = (push1,1,R); δ(push1,) = (push1, ,R); δ(push1,|) = (push1,|,R); δ(push1,.) = (push1,.,R); δ(push1,B) = (returnpush,1,L);	δ(startpop,) = (startpush, ,R); δ(startpop,.) = (acceptonempty,.,R); δ(startpop,0) = (pop0,W,R); δ(startpop,1) = (pop1,Q,R);	δ(acceptonempty,.) = (acceptonempty,.,R); δ(acceptonempty,|) = (acceptonempty,|,R); δ(acceptonempty,1) = (reject,1,R); δ(acceptonempty,0) = (reject,0,R); δ(acceptonempty,B) = (accept,B,R);	δ(pop0,0) = (pop0,0,R); δ(pop0,1) = (pop0,1,R); δ(pop0,) = (pop0, ,R); δ(pop0,.) = (pop0,.,R); δ(pop0,|) = (dopop0,|,R);
δ(pop1,0) = (pop1,0,R); δ(pop1,1) = (pop1,1,R); δ(pop1,) = (pop1, ,R); δ(pop1,.) = (pop1,.,R); δ(pop1,|) = (dopop1,|,R);	δ(dopop0,0) = (returnpop,|,L); δ(dopop0,1) = (reject,1,L); δ(dopop0,B) = (reject,B,L);	δ(dopop1,1) = (returnpop,|,L); δ(dopop1,0) = (reject,0,L); δ(dopop1,B) = (reject,B,L);	δ(returnpop,|) = (returnpop,.,L); δ(returnpop,0) = (returnpop,0,L); δ(returnpop,1) = (returnpop,1,L); δ(returnpop,) = (returnpop, ,L); δ(returnpop,.) = (returnpop,.,L); δ(returnpop,W) = (startpop,W,R); δ(returnpop,Q) = (startpop,Q,R);

Graf stanja i prijelaza



Osvrt na definiciju:

Budući je stroj sastavljen od ukupno 18 stanja te da postoji mnogo prijelaza, radi jednostavnosti ću pojasniti što se nastoji učiniti kada se stroj nalazi u pojedinom stanju. Analogna stanja za bitove (0|1) su grupirana.

1) **init** – početno stanje
   U ovom stanju, nastojimo inicijalizirati queue, potrebno je postaviti razdjelnik (znak „|“) na desni kraj trake (prva slobodno ćelija)
   Kako bi se uštedio jedan povratak na početak samo da bi se napravio prvi PUSH, stoga u ovom stanju odmah „pamtimo“ koji bit treba staviti na kraj QUEUE-a. Pamćenje bita se vrši pomoću dva stanja (set_queue_start0 i set_queue_start1) koje biramo ovisno o prvom pročitanom bitu. Prvi bit (0|1) se preimenuje u (N|J) što su oznake za neparne grupe koje sadrže obrađene bitove. Neparne grupe su ulazne i iniciraju PUSH, dok su parne izlazne i iniciraju POP

2) **set_queue_start(0|1)** - analogna stanja za bitove 0 ili 1 koja će postaviti razdjelnik „|“ te pozvati **dopush(0|1).** Stroj se nalazi u ovom stanju samo na početku (sve dok se ne inicijalizra queue) te se više u njega ne vraća.

3) **dopush(0|1)** – stavlja bit (0|1) na kraj stoga (trenutno stanje glave) i vraća se nazad pozivom stanja **returnpush**.

4) **returnpush –** stanje u kojem se „vraćamo po još“, krećemo se lijevo po traci sve dok ne naiđemo na neki obrađeni znak u neparnoj grupi (N|J), tada iniciramo novi PUSH pozivom stanja **startpush**.

5) **startpush –** stanje u kojem određujemo da li ćemo napraviti PUSH nekog bita (0|1) pozivom stanja **push(0|1)** ili ako smo obradili cijelu neparnu grupu ćemo prijeći u parnu grupu pa time automatski inicirati POP pozivom stanja **startpop**. Prijelaz iz neparne u parnu grupu određuje znak razmaka u trenutku kad odlučujemo hoćemo li nastaviti s PUSH ili inicirati POP. Startpush preimenuje (0|1) u (N|J) unutar neparnih grupa bitova.

6) **push(0|1)** – stanje koje nastoji prenjeti bit (0|1) do desnog kraja trake i upisati ga na kraj queue-a kada naiđe na praznu ćeliju.

7) **startpop** – u parnoj grupi bitova, za bilo koji bit (0|1) pozivamo stanje **pop(0|1).** Inače ukoliko pročitamo „ . “ (znak točke je mjesto nekadašnjeg elementa na queue-u koji je POP-an) tada je pod uvjetom da je QUEUE prazan potrebno prekinuti rad stroja u prihvatljivom stanju, a to vršimo pozivom stanja **acceptonempty (accept on empty queue).** startpop preimenuje (0|1) u (W|Q) unutar parnih grupa bitova.

8) **acceptonempty –** stanje koje će utvrditi da li je QUEUE prazan te u tom slučaju prijeći u stanje **accept**, a inače u stanje **reject**. Ovo stanje koristimo kad smo npr. na samom kraju niza (zadnji POP).

9) **pop(0|1)** – stanje koje nastoji doći do početka QUEUE-a. Kada je to postignuto (pročitan je znak „|“) poziva se stanje **dopop(0|1)**.

10) **dopop(0|1) –** ovo stanje obavlja nekoliko zadaća, ukoliko pročita suprotni bit od zadanog (0|1) → (1|0) to znači da je poredak neispravan, potrebno je prekinuti stroj u neprihvatljivom stanju. Dakle u ovom stanju provjeravamo da li je poredak ulazno izlaznog tijeka ispravan (str. 3 → ? pop == X). Ako je poredak ispravan, potrebno je pomaknuti početni [S] pokazivač QUEUE-a za jedno mjesto udesno (dakle prebrisati pročitani bit sa znakom „|“) te pozvati **returnpop** koji će stari znak „|“ preimenovati u točku te se „vratiti po još“

11) **returnpop –** analogno stanju returnpush, nastojimo se vratiti na zadnje obrađivanu parnu grupu i pozvati **startpop**

12) **accept –** stroj se zaustavlja u prihvatljivom stanju

13) **reject –** stroj se zaustavlja u neprihvatljivom stanju

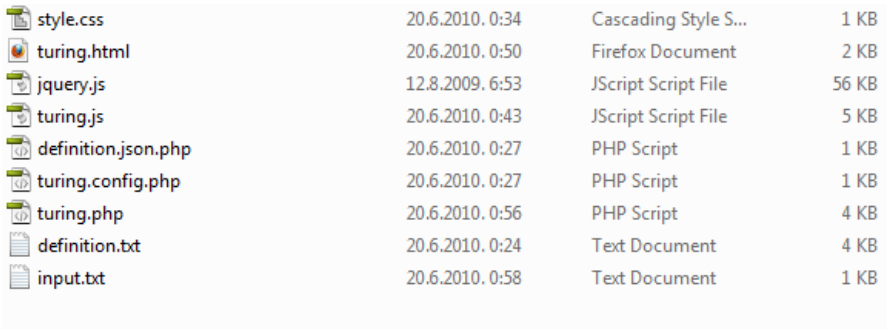
Programsko ostvarenje:
Simulacija turingovog stroja za utvrđivanje ispravnosti poretka ulazno-izlaznog tijeka

Za izradu simulatora turingovog stroja odabrao sam programske (skriptne) jezike PHP i Javascript radi sljedećih prednosti:

1) Jezici su slični C-u po sintaksi
2) Podržavaju asocijativna polja (assoc. array) koja su iznimno korisna za definiranje gotovo bilo kakve strukture podataka i objekata
3) Laka konverzija iz assoc. array u JSON (Javascript Object Notation) i obratno
4) Dostupnost na Webu (kompatibilnost/iskoristivost)
5) Mogućnost animirane simulacije pomoću javascript-a
6) Vlastito višegodišnje iskustvo u radu s web tehnologijama.

Struktura projekta:

Glavni dio projekta je **turing simulator**, to je objekt koji prima definiciju stroja i ulazni niz znakova te pokreće simulaciju, dakle isti simulator se može koristiti za bilo koji turingov stroj s jednom trakom koji prati formalnu definiciju. Definicija stroja kojim je ostvaren projektni zadatak i probni ulazni niz se nalaze u zasebne dvije datoteke. Ulazni niz znakova se može editirati putem stranice turing.php.



turing.php	Turing simulator (PHP)
turing.js	Turing simulator (JS)
turing.html	Turing simulator - animacija
turing.config.php	Sadrži nazive datoteka u kojima se nalazi definicija i ulazni niz
definition.txt	Definicija turingovog stroja u PHP assoc. array obliku
input.txt	Datoteka s ulaznim nizom znakova (početno stanje trake)
definition.json.php	Konvertira PHP assoc. array iz **definition.txt** u JSON oblik radi animacije koju pokreće JS verzija. Poziv skripte se vrši ajax-om u turing.js kad se stranica turing.html učita
style.css	Stilovi (fontovi i boje)
jquery.js	www.jquery.com

Čitav projekt je također dostupan na Webu:

1) Turing simulator (PHP)
http://www.invision-web.net/turing/turing.php

2) Turing animirani simulator (HTML+JS)
http://www.invision-web.net/turing/turing.html

3) Popis datoteka (dirlist):
http://www.invision-web.net/turing/

4) Download cijelog projekta:
http://www.invision-web.net/turing/turing.rar

5) Seminar
http://www.invision-web.net/turing/UTRZ-2010-ZAD2029-Kristijan-Burnik.pdf

6) XAMPP instalacija (za Windowse):
http://www.invision-web.net/turing/xampp-win32-1.7.1.exe

Projekt je testiran na Firefox-u i Google Chrome-u.

Zaključak

Ovim seminarskim radom prikazana je jedna od mogućih primjena turingovog stroja, konkretno je riješen problem utvrđivanja ispravnosti poretka ulazno-izlaznog tijeka automobila po tipu vozila. Za rješavanje problema korišten je koncept FIFO odnosno struktura podataka QUEUE. Moguće poboljšanje za pouzdanost (vjerodostojnost) čitavog sustava je da se postavi nekoliko kontrolnih točaka (dakle ne samo na ulazu i izlazu iz tunela).

Izrada simulatora turingovog stroja te čitavog projekta mi je veoma pomoglo u shvaćanju moći i upotrebljivosti turingovog stroja u nizu konkretnih problema automatizacije, a također me je potaklo na jedan novi način razmišljanja i mogućnost postizanja određenih rješenja na drukčiji način od samog programskog kroz više programske jezike.