Pripremni materijali za Nedelju informatike

Organizatori Nedelje informatike

5. decembar 2016

Sadržaj

Glava 1

O ovom dokumentu

§1.1. Priprema za predavanja

Svesni smo da predznanje polaznika Nedelje informatike dosta varira u nekim oblastima. Koliko smo mogli, potrudili smo se da predavanja učinimo pristupačnim svima koji su uspeli da prođu naš "ulazni test".

Ipak, pojedina predavanja zahtevaju poznavanje nekih matematičkih pojmova koje možda nisu svi videli do sada, a njihovo uvođenje i objašnjavanje bi nam oduzelo previše vremena koje bismo mogli da iskoristimo za daleko zanimljivije stvari. Radionice bi bilo teško realizovati ukoliko biste po prvi put čuli ili videli za neke stvari koje ćemo na njima koristiti. Ovi materijali postoje da bi vas spremili i ne bi trebalo da vam njihovo čitanje oduzme previše vremena.

Trudili smo se da pojasnimo ove pojmove što je moguće jednostavnije. Ukoliko je nešto nejasno, postoji dosta materijala o stvarima koje su ovde napisane koje možete da nađete i *online* jako lako. Ukoliko još uvek imate problema, obratite nam se!

§1.2. Raspored Nedelje

Raspored predavanja i radionica je u narednoj tabeli:

Ponedeljak	**Utorak**	**Sreda**	**Četvrtak**	**Petak**
Napredne pretrage u grafovima (Nikola Nedeljković)	Veeeeeeliki brojevi (Aleksandar Ivanović)	Neuralne mreže (Petar Veličković)	Napredne tehnike kriptografije (Stefan Crnojević)	Distribuirani sistemi (Nikola Jovanović)
Izračunljivost i složenost (Andrej Ivašković)	NP-kompletnost (Andrej Ivašković)	Keras radionica (Nenad Bauk, Petar Veličković)	Kvantno računarstvo (Marina Ivanović)	Arduino radionica (Andrej Ivašković, Lazar Mitrović)
`git` (Nikola Jovanović)	Obrnuto inženjerstvo (Lazar Mitrović)	Gostujuće predavanje (MDCS)	Autostoperski vodič kroz softverske ranjivosti (David Davidović)	Arduino radionica (Andrej Ivašković, Lazar Mitrović)

Predavanja traju po **sat vremena**. Između predavanja će biti pauze od oko 15 minuta.

U ponedeljak, utorak, četvrtak i petak će predavanja biti održana u **Matematičkoj gimnaziji** (Kraljice Natalije 37). Tada ćemo sve vreme biti u jednom od kabineta na trećem spratu i predavanja će početi u **14:00** ovim danima.

U sredu će predavanja i radionice biti održani u prostorijama **Microsoft Development Center Serbia** (Španskih boraca 3). Tada ćemo se naći kao grupa ispred zgrade u kojoj se nalazi MDCS (Blue Center) u **11:30**. Dođite na vreme, u suprotnom verovatno nećete moći da uđete!

Imajte u vidu ovaj raspored pri čitanju ovih materijala: do ponedeljka pročitajte sve što je relevantno za "Izračunljivost i složenost", do srede sve što je relevantno za "Neuralne mreže" itd.

§1.3. Autori materijala

Ove materijale su sastavili organizatori Nedelje informatike, i to konkretno:

1. **Izračunljivost i složenost** – Andrej Ivašković
2. **Neuralne mreže** – Petar Veličković
3. **Napredne tehnike kriptografije** – Stefan Crnojević
4. **Kvantno računarstvo** – Marina Ivanović
5. **Softverske ranjivosti** – David Davidović
6. **Arduino radionica** – Andrej Ivašković

Glava 2

Pripremni materijali

§2.1. Izračunljivost i složenost

Na predavanju "Izračunljivost i složenost" će nam biti neophodne neke matematičke ideje koje su uglavnom pokrivene srednjoškolskim gradivom, ali na malo drugačiji način. Ovde ćemo pokušati ukratko da pojasnimo važne pojmove iz analize i teorije skupova.

§2.1.1. Parcijalne funkcije i bijekcije

Za potrebe razmatranja teorijskog računarstva nam je neophodno razumevanje nekih matematičkih pojmova – konkretno, koncepata iz teorije skupova. Veći deo ovog odeljka je pokriven gradivom prvog razreda matematike (odnosno analize sa algebrom) u srednjim školama, ali radi potpunosti ponavljamo neke definicije.

Ovo će biti izloženo u pomalo dosadnom formatu, gde će se samo naizmenično navoditi definicije i primeri, ali to je zbog toga što se ovde ne predstavlja nikakva komplikovana ideja: ključno je pročitati tvrđenja kako treba. Takođe, dosta važnih stvari ćemo preskočiti jer nećemo direktno koristiti na predavanju. Ovde pretpostavljamo da ste upoznati sa značenjem simbola koje ćemo koristiti (među kojima su $\in$, $\wedge$, $\Longrightarrow$, $\times$, $\forall$) i nekim osnovnim pojmovima u vezi sa skupovima.

Definicija 2.1.1

Binarna relacija R nad skupovima A i B je podskup Dekartovog proizvoda A i B: $R \subseteq A \times B = \{(a,b) | a \in A \wedge b \in B\}$. Koristimo zapis $x\,R\,y$ kao mogući način da se napiše $(x,y) \in R$.

Primer 2.1.1

Razmotrimo naredne relacije:

1. Za $A = \{1,2,3,4\}$, možemo da definišemo relaciju:

$$R = \{(1,1),(2,2),(3,3),(4,4),(1,3),(2,4),(3,1),(4,2)\} \subseteq A \times A$$

Ovde $x\,R\,y$ može da znači "x i y su brojevi iste parnosti".

2. Na bilo kom skupu X, jednakost je relacija $R = \{(x,x)|x \in X\}$.

3. Ako je $\mathbb{N}_0$ skup nenegativnih celih brojeva, možemo da definišemo relaciju $\leqslant \subseteq \mathbb{N}_0 \times \mathbb{N}_0$ ("manje ili jednako"). Primetimo da smo naviknuti na zapis $4 \leqslant 7$, dok je $(4,7) \in \leqslant$ malo neuobičajen.

Nije teško pojam relacije proširiti sa binarnih na k-narne, jedina izmena definicije je u tome što će se posmatrati Dekartov proizvod k skupova.

Koncept relacije je važan u računarstvu, ali u okviru ovog predavanja nas interesuje jedna karakteristična vrsta:

Definicija 2.1.2

Relacija $R \subseteq A \times B$ je **parcijalna funkcija** iz skupa A u skup B ukoliko za svaki element $a \in A$ postoji najviše jedan $b \in B$ takav da $a\,R\,b$. Ukoliko takvo b postoji, pišemo $R(a) = b$, $R : a \mapsto b$ ili $R(a)\downarrow$ i kažemo da je **definisana** u a, a u suprotnom $R(a)\uparrow$, $R : a \mapsto \uparrow$ ili $R(a) = \uparrow$.

Skup parcijalnih funkcija iz A u B se oznacava sa $A \rightharpoonup B$, a iskaz "f je parcijalna funkcija iz A u B" se zapisuje $f \in A \rightharpoonup B$ ili $f : A \rightharpoonup B$.

Parcijalne funkcije, prema tome, možemo koncizno da definišemo ne u vidu skupa uređenih parova, već navođenjem da li je definisana u konkretnoj tački i, ako jeste, koja je vrednost u pitanju.

Parcijalne funkcije imaju "operaciono" značenje: za neki ulaz x ili dobijemo izlaz $f(x)$ ili on nije definisan tj. $f(x)\uparrow$. Zato su one od velikog značaja u formalnom zasnivanju racunarstva, kao što se da videti na primeru koji sledi: za neke ulaze nijedan izlaz nema smisla. Dakle, ono što se zove *funkcija* ili *potprogram* u programskim jezicima odgovara parcijalnim funkcijama. Teorija izracunljivosti će nam koncept "parcijalna funkcija nije definisana u ovoj tački domena" prevesti u "program nikada neće završiti svoje izvršavanje ako mu je ovo ulaz".

Primer 2.1.2

Razmotrimo naredne relacije:

1. Parcijalna funkcija $\texttt{succ} : \mathbb{N}_0 \rightharpoonup \mathbb{N}_0$ predstavlja sledbenika nekog nenegativnog celog broja:

$$\forall x \in \mathbb{N}_0 \quad \texttt{succ}\, x = x + 1$$

2. Parcijalna funkcija $\texttt{pred} : \mathbb{N}_0 \rightharpoonup \mathbb{N}_0$ nam daje prethodnika nekog nenegativnog celog broja, pri čemu ovo nije definisano za $0 \in \mathbb{N}_0$:

$$\forall x \in \mathbb{N}_0 \quad \texttt{pred}\, x = \begin{cases} \uparrow & x = 0 \\ x - 1 & x \neq 0 \end{cases}$$

3. Parcijalna funkcija $k : (\mathbb{N}_0 \times \mathbb{N}_0) \rightharpoonup \mathbb{N}_0$ ima značenje "količnika" dva

nenegativna cela broja, što je dobro definisan koncept kada delilac (druga koordinata uređenog para) nije 0:

$$\forall (x,y) \in \mathbb{N}_0 \times \mathbb{N}_0 \quad k(x,y) = \begin{cases} \uparrow & y = 0 \\ \lfloor x/y \rfloor & y \neq 0 \end{cases}$$

4. Za $A = \{1,2\}$, $B = \{a,b\}$, relacija $R = \{(1,b),(2,a),(2,b)\}$ nije parcijalna funkcija iz A u B jer postoje dva različita elementa B koji su u relaciji sa istim elementom B.

Koristan je pojam **domena** parcijalne funkcije:

Definicija 2.1.3

Domen parcijalne funkcije $f : A \rightharpoonup B$ je skup $\operatorname{dom} f \subseteq A$ takav da:

$$\forall x \in A \quad f(x)\downarrow \iff x \in \operatorname{dom} f$$

Drugim rečima, domen parcijalne funkcije je skup na kom je ona definisana.

Definicija 2.1.4

Za $f : A \rightharpoonup B$ i $g : B \rightharpoonup C$, **kompozicija** g i f, u oznaci $g \circ f$, jeste parcijalna funkcija $h : A \rightharpoonup C$ koja zadovoljava:

$$\forall x \in A \quad h(x) = (g \circ f)(x) = \begin{cases} g(f(x)) & f(x) \in \operatorname{dom} g \\ \uparrow & f(x) \notin \operatorname{dom} g \end{cases}$$

Primer 2.1.3

Neka je parcijalna funkcija $f : \mathbb{N}_0 \rightharpoonup \mathbb{N}_0$ takva da za sve $x \in \mathbb{N}_0$ važi $f : x \mapsto 2x$. Posmatrajmo tada funkcije $\texttt{pred} \circ f$ i $f \circ \texttt{pred}$:

$$(\texttt{pred} \circ f)(x) = \texttt{pred}(f(x)) = \texttt{pred}(2x) = \begin{cases} \uparrow & x = 0 \\ 2x-1 & x \neq 0 \end{cases}$$

$$(f \circ \texttt{pred})(x) = f(\texttt{pred}\, x) = \begin{cases} \uparrow & x = 0 \\ f(x-1) & x \neq 0 \end{cases} = \begin{cases} \uparrow & x = 0 \\ 2x-2 & x \neq 0 \end{cases}$$

Vidimo da ove dve funkcije nisu iste. Dakle, redosled primene $\circ$ je bitan.

Ono sa čim su verovatno svi upoznati su **funkcije**, koje su samo specijalan slučaj parcijalnih funkcija:

Definicija 2.1.5
Parcijalna funkcija $f : A \rightharpoonup B$ je **funkcija** (ili **totalna funkcija**) ukoliko je dom $f = A$. U tom slučaju pišemo $f : A \to B$ ili $f \in A \to B$.

Primer 2.1.4

1. succ je primer funkcije.
2. Za $A = \{1,2\}$, $B = \{a,b\}$, relacija $R = \{(1,a)\}$ nije funkcija jer nije definisana u $2 \in A$. Međutim, R jeste parcijalna funkcija.
3. Na svakom nepraznom skupu X može da se definiše funkcija $\mathrm{id}_X : X \to X$ (koju zovemo **identitetom** skupa X), gde je za sve $x \in X$ zadovoljeno $\mathrm{id}_X\, x = x$.

Dakle, kod funkcija postoji tačno jedan izlaz za svaki ulaz. Posebno su zanimljive i funkcije kod kojih za svaki izlaz postoji i tačno jedan ulaz:

Definicija 2.1.6
Funkcija $f : A \to B$ je **bijekcija** ukoliko postoji funkcija $g : B \to A$ takva da je $g \circ f = \mathrm{id}_A$ i $f \circ g = \mathrm{id}_B$, pri čemu tada pišemo $g = f^{-1}$.

Primer 2.1.5

1. $f : \mathbb{Z} \to \mathbb{Z}$ takva da je $f(x) = x + 1$ za sve x, jeste bijekcija sa inverznom funkcijom f^{-1} koja za sve $x \in \mathbb{Z}$ zadovoljava $f^{-1}(x) = x - 1$, jer $f^{-1}(f(x)) = (x+1) - 1 = x$ i $f\left(f^{-1}(x)\right) = (x-1) + 1 = x$.

   Sa druge strane, succ nije bijekcija: ako bi postojala $g = \mathtt{succ}^{-1}$, tada bi važilo $\mathtt{succ}\,(g\,(0)) = 0$, što nije moguće jer ne postoji nijedno $x \in \mathbb{N}_0$ takvo da je $\mathtt{succ}\, x = 0$.
2. Postoji bijekcija f između skupa nenegativnih celih brojeva $\mathbb{N}_0$ i skupa parnih nenegativnih celih brojeva $2\mathbb{N}_0$: $f(x) = 2x$.

Bijekcije su korisne jer mogu da objasne neke pojmove iz teorije skupova na koncizan način.

Definicija 2.1.7
Skup A je **konačan** ukoliko postoji neko $n \in \mathbb{N}_0$ za koje postoji bijekcija $f : [n] \to A$, gde je $[n] = \{0, 1, 2, \ldots, n-1\}$.

Intuitivno, A je konačan skup ukoliko možemo da mu " obeležimo" elemente, odnosno da ih " poređamo"koristeći konačno mnogo nenegativnih celih brojeva. Na predavanju ćemo videti da su konačni skupovi nezanimljivi u teoriji izračunljivosti i umesto toga ćemo posmatrati beskonačne *prebrojive* skupove, odnosno beskonačne podskupove $\mathbb{N}_0$.

Postavlja se pitanje kako se ovo tačno uklapa u računarstvo i programiranje – za odgovor sačekajte predavanje![1]

§2.1.2. Asimptotska oznaka O

U dosadašnjem programerskom radu ste se možda susreli sa idejom *vremenske složenosti algoritma* i videli tvrđenja poput "ovaj algoritam ima vremensku složenost $O\left(n^2\right)$, gde je n dužina ulaza". Često se ova "procena vremena izvršavanja" vrši na osnovu nekih intuitivnih zapažanja, uglavnom razmatranjem toga koliko puta se izvršava telo nekog ciklus. Nešto su složenije analize rekurzivnih algoritama: na primer, *Merge sort* ima vremensku složenost $O(n \log n)$, ali zbog čega? Na ovom predavanju ćemo dati preciznu definiciju složenosti algoritma koristeći *Tjuringovu mašinu*, ali je najpre neophodno da shvatimo da oznaka O zapravo potiče iz *matematičke analize* i trebalo bi da znamo njeno opštije značenje da bismo opravdali neke operacije.

Napominjemo da ovde terminologija i oznake variraju od izvora do izvora.

U daljem razmatranju posmatramo **realne funkcije**, odnosno funkcije oblika $f : A \to \mathbb{R}$, gde $A \subseteq \mathbb{R}$. Postavićemo i dodatni uslov da postoji neko $a > 0$ tako da $[a, +\infty) \subseteq A$, odnosno da su definisane za sve vrednosti argumenta počev od nekog. Ovo radimo da bismo "olakšali sebi život" i lakše objasnili šta se dešava za "beskonačno velike" vrednosti argumenta. Ovakve funkcije ćemo nazvati "finim".

Posmatrajmo funkcije $f(x) = x^3$ i $g(x) = 3x^2 + 11$. Pogledajmo koje su im vrednosti za neke realne vrednosti:

x	$f(x)$	$g(x)$
0	0	11
1	1	14
2	8	23
3	27	38
4	64	59
5	125	86
6	216	119
...	...	...

Vidimo da je za male pozitivne vrednosti x vrednost $f(x)$ manja od $g(x)$, ali da se situacija ubrzo menja i $f(x) > g(x)$ za veće vrednosti x. Štaviše, dramatična je razlika između $f(100) = 1000000$ i $g(100) = 30011$. Deluje kao da razlika između $f(x)$ i $g(x)$ može biti proizvoljno velika, ali i količnik. I ovo je tačno: koji god broj $a > 0$ da odaberemo, postojaće neka vrednost $m \in \mathbb{R}$ takva da je $f(m) > ag(m)$ – i ne samo to, već će za sve brojeve $x \geqslant m$ ova nejednakost takođe važiti. Zapisano matematičkim formalizmom:

$$\forall a > 0 \quad \exists m \in \mathbb{R} \quad \forall x \geqslant m \quad f(x) > ag(x)$$

Ovo takođe znači da ne postoji način da se $f(x)$ odozgo ograniči sa $a \cdot g(x)$, gde bi a bila neka konstanta.

[1] Ili pročitajte nešto o ovome pre toga, ali bih voleo da budem baš ja taj koji će vas upoznati sa ovom temom...

Sa druge strane, posmatrajući $h(x) = 2x^2 + 7$, možemo da uočimo sledeće:

x	$g(x)$	$h(x)$
0	11	7
1	14	9
2	23	15
3	38	25
4	59	39
5	86	57
6	119	79
...	...	...

Ovde je jasno da je za sve $x \geqslant 0$ zadovoljeno $g(x) \geqslant h(x)$, ali i da je $g(x) \leqslant 2 \cdot h(x)$. Ono što se "naslućuje" jeste da se odnos $g(x)/h(x)$ "približava" broju $3/2$.

Imajući ovo u vidu, definišemo jednu važnu relaciju između realnih funkcija:

Definicija 2.1.8

"Finu" realnu funkciju g nazivamo **asimptotski manjom** od "fine" realne funkcije f ukoliko:

$$\forall a > 0 \quad \exists m \in \mathbb{R} \quad \forall x \geqslant m \quad x \in (\operatorname{dom} f \cup \operatorname{dom} g) \wedge |f(x)| > a|g(x)|$$

Naravno, tada kažemo i da je f **asimptotski veća** od g.

Skup svih funkcija koje su asimptotski manje ("<") od h se označava sa $o(h)$.

Skup svih funkcija koje *nisu asimptotski veće* (dakle, u značenju "$\leqslant$") od h se označava sa $O(h)$.

Dakle, ako g nije asimptotski veća od f, na osnovu ove definicije pišemo $g \in O(f)$. Međutim, konvencionalno (i neprecizno) se piše i $g = O(f)$. Slično važi i za o.

U definiciji smo stavili i apsolutnu vrednost, iako ćemo u okviru ovog predavanja raditi skoro isključivo sa funkcijama koje su pozitivne za pozitivnu vrednost argumenta. Ovo nam omogućava da, recimo, napišemo $e^{-x} \cos x \in o(1)$.

Primetimo da se $g \in O(f)$ ekvivalentno može zapisati i kao $O(g) \subseteq O(f)$. Slično važi i za o, samo što se tada koristi $\subset$ umesto $\subseteq$.

Primer 2.1.6

Ovo su primeri nekih asimptotskih poredaka:

1. Na osnovu prethodnog razmatranja vidimo da je $3x^2 + 11$ asimptotski manja funkcija od x^3, te je $3x^2 + 11 \in O\left(x^3\right)$. Međutim, važi i $3x^2 + 11 \in O\left(x^2\right)$. Sa druge strane $3x^2 + 11 \in o\left(x^3\right)$, ali $3x^2 + 11 \notin o\left(x^2\right)$
2. Uopšte, za polinom p stepena m i neko $n > m$ će važiti $p \in o(x^n)$ (a samim tim i $p \in O(x^n)$).

3. $\log_2 x \in o(x)$ i $\log_2 x \in o\left(x^2\right)$, ali i za svaki realan stepen $n > 0$ važi $\log_2 x \in o(x^n)$. Primera radi, $\log_2 x \in o\left(x^{1/2}\right) = o\left(\sqrt{x}\right)$.

4. Kako je $\log_3 x = \log_3 2 \cdot \log_2 x \approx 0.63 \cdot \log_2 x$, vidimo da je $\log_2 x \in O(\log_3 x)$ i $\log_3 x \in O(\log_2 x)$. Zapravo, $O(\log_2 x) = O(\log_3 x)$ i zato se često piše samo $O(\log x)$.

Na predavanju će nas uglavnom interesovati oznaka O umesto o.

Ovo pravda neka uobičajena zapažanja pri proceni složenosti algoritma: u $O\left(x^3 + 100x^2 + \log_2^2 x\right)$ je dovoljno posmatrati samo sabirak x^3 jer je on ovde "najznačajniji" kada je x jako veliki broj (odnosno asimptotski najveći) i dovoljno je napisati $O\left(x^3\right)$. Sično, konstante kojima se množi slobodno mogu da se uklone, $O(30x \log x) = O(x \log x)$.

Ključno zapažanje koje treba imati na umu je naredni niz konkretnih primera asimptotskih poredaka, koji nam daje predstavu o hijerarhiji funkcija:

$$O(1) \subset O(\log n) \subset O\left(\sqrt{n}\right) \subset O(n) \subset O(n \log n) \subset \\ \subset O\left(n^2\right) \subset O\left(n^3\right) \subset O(2^n) \subset O(3^n)$$

§2.2. Neuralne mreže

Predavanje o neuralnim mrežama na ovogodišnjoj Nedelji informatike će biti ambiciozno, u smislu količine materijala koju planiramo pokriti unutar jednog sata—da bi mogli da ostavimo što više vremena za Keras radionicu. Stoga, ovde prilažemo objašnjenja nekih osnovnih pojmova na kojima se nećemo mnogo zadržavati tokom samog predavanja.

Preporučujemo da ove materijale pregledate pre samog predavanja, ali mogu biti i koristan *cheat sheet* tokom istog.

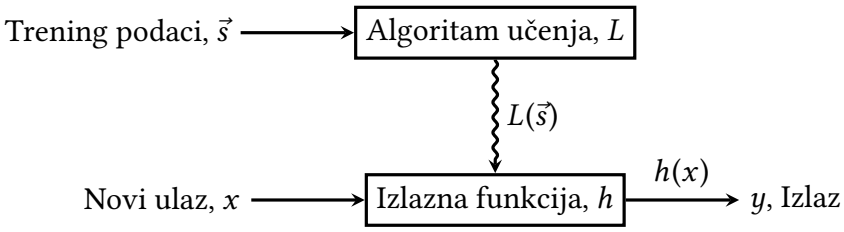
§2.2.1. Supervizirano mašinsko učenje

- Jedan od najčešćih vidova mašinskog učenja—kojim ćemo se jedino i baviti tokom ovog predavanja—je *supervizirano učenje*.

- U problemu superviziranog učenja, cilj je naučiti *funkciju* $f : X \to Y$, iz skupa *trening podataka* $\vec{s} = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ koji predstavljaju poznate parove ulaza i izlaza[2] za funkciju f.

- Cilj je izgraditi *algoritam učenja*, $L : (X \times Y)^m \to (X \to Y)$, koji će iz trening skupa da *nauči* aproksimaciju funkcije f—pišemo $L(\vec{s}) = h$. Funkcija $h : X \to Y$ se onda može koristiti za procesiranje do tada *neviđenih* ulaza. Uspešnost algoritma učenja se ogleda u tome koliko precizno funkcija h radi na ovakvim podacima!

[2]U realnim problemima, postoji mogućnost da i u ovim parovima postoje netačnosti—o ovom problemu više na samom predavanju.



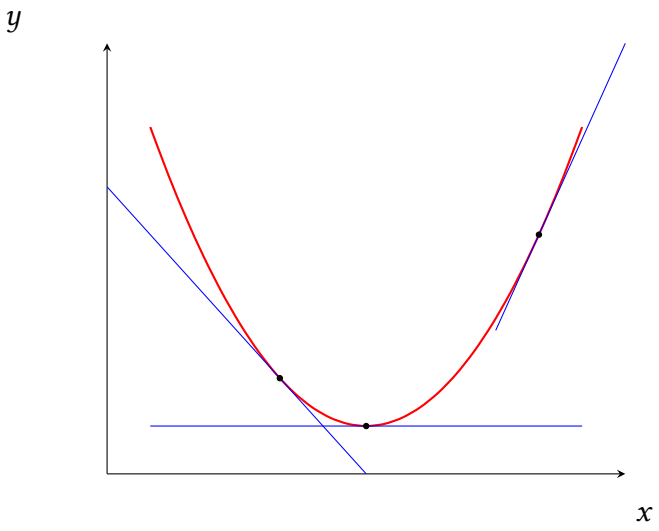
- Neuralne mreže su specifično dizajnirane za supervizirano učenje: njihove primene u drugim kontekstima (npr. igranju igrica) podrazumevaju prvobitnu *reformulaciju* problema u problem superviziranog učenja!

§2.2.2. Parcijalni izvodi i gradijenti

- Pretpostavimo da imamo funkciju $f : \mathbb{R} \to \mathbb{R}$. Njen *izvod* po promenljivoj[3] x, $\frac{df}{dx}$, u tački $x' \in \mathbb{R}$, se definiše kao koeficijent pravca tangente na grafik funkcije u toj tački:

$$\frac{df}{dx}(x') = \lim_{\Delta x \to 0} \frac{f(x' + \Delta x) - f(x')}{\Delta x}$$

- Stoga, izvod u tački x' nam daje direktnu informaciju o *brzini promene funkcije* u toj tački. Na slici ispod možete videti tangente na kvadratnu funkciju u različitim tačkama—primetite da je u minimumu funkcije izvod (koeficijent pravca/brzina promene) jednak nuli!



- Generalno (pogotovo u mašinskom učenju) će nas zanimati *funkcije više promenljivih*: $f : \mathbb{R}^n \to \mathbb{R}$. Ovde možemo da posmatramo izvode po svakoj promenljivoj x_i ponaosob, dok se sve ostale promenljive x_j drže fiksiranim—ovaj termin se zove *parcijalnim izvodom* po promenljivoj x_i:

$$\frac{\partial f}{\partial x_i}(x_1, \ldots, x_n) = \lim_{\Delta x \to 0} \frac{f(x_1, \ldots, x_i + \Delta x, \ldots, x_n) - f(x_1, \ldots, x_n)}{\Delta x}$$

[3] Ponekad označavan i sa $f'(x)$.

- Ukoliko uračunamo doprinose parcijalnih izvoda po svakoj promenljivoj x_i u zasebnoj dimenziji, dobijamo n-dimenzionalni vektor koji se zove *gradijent* funkcije f (označava se sa ∇f) , i on označava pravac najbržeg rasta funkcije u ovom prostoru, za neku datu tačku $(x_1', \ldots, x_n')$.

$$\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x_1} & \cdots & \frac{\partial f}{\partial x_n} \end{pmatrix}$$

- Za parcijalne (kao i obične) izvode, važi *pravilo kompozicije*. Ukoliko imamo funkciju $f(u_1(x_1, \ldots, x_m), \ldots, u_n(x_1, \ldots, x_m))$ (tako da svaki od n ulaza za funkciju f direktno zavisi od m promenljivih $(x_1, \ldots, x_m)$), onda parcijalni izvod ove funkcije po promenljivoj x_i određujemo kao:

$$\frac{\partial f}{\partial x_i} = \sum_{j=1}^{n} \frac{\partial f}{\partial u_j} \frac{\partial u_j}{\partial x_i}$$

- *Podsetnik*: Tablica izvoda nekih standardnih funkcija:

$f(x)$	$\frac{df(x)}{dx}$	$f(x)$	$\frac{df(x)}{dx}$
C	0	a^x	$a^x \ln a$
x	1	e^x	e^x
x^n	nx^{n-1}	$\log_a(x)$	$\frac{1}{x \ln a}$
$\frac{1}{x^n}$	$-\frac{n}{x^{n+1}}$	$\ln x$	$\frac{1}{x}$

§2.2.3. Osnove verovatnoće i statistike

- Posmatrajmo slučajnu promenljivu X sa mogućim ishodima $x_1, x_2, x_3, \ldots$ (gde broj ishoda može biti i beskonačan) i verovatnoćom $\mathbb{P}(X = x_i)$ za svaki ishod tako da važi:
  - $\mathbb{P}(X = x_i) \in [0, 1]$;
  - $\sum_{x_i} \mathbb{P}(X = x_i) = 1$.
- Tada možemo definisati *matematičko očekivanje* ove promenjive, $\mathbb{E}(X)$ kao prosečni ishod nakon mnogo viđenih ishoda:

$$\mathbb{E}(X) = \sum_{x_i} x_i \mathbb{P}(X = x_i)$$

- Takođe možemo definisati *varijansu* ove promenljive, $\mathrm{Var}(X)$, kao očekivano (kvadratno) odstojanje od matematičkog očekivanja:

$$\mathrm{Var}(X) = \mathbb{E}((X - \mathbb{E}(X))^2)$$

  Varijansa se ponekad označava i sa σ^2, zato što je $\sqrt{\mathrm{Var}(X)}$ često korišćena metrika—*standardna devijacija*, σ, ove promenljive.

- Na primer, ukoliko bacamo kockicu sa ishodima $1, 2, \ldots, 6$, i verovatnoćom $\frac{1}{6}$ za svaki ishod, onda je očekivana vrednost ovog eksperimenta 3.5:

$$\mathbb{E}(X) = 1 \times \frac{1}{6} + 2 \times \frac{1}{6} + \cdots + 6 \times \frac{1}{6} = 3.5$$

  Varijansa, tj. očekivano kvadratno odstupanje od ove vrednosti, se onda može izračunati kao:

$$\begin{aligned} \mathrm{Var}(X) &= (1 - 3.5)^2 \times \frac{1}{6} + (2 - 3.5)^2 \times \frac{1}{6} + \cdots + (6 - 3.5)^2 \times \frac{1}{6} \\ &= 0.25 \times \frac{1}{3} + 2.25 \times \frac{1}{3} + 6.25 \times \frac{1}{3} \approx 2.92 \end{aligned}$$

- Ukoliko ne znamo sve moguće ishode i/ili njihove verovatnoće, očekivanje i varijansu možemo proceniti iz datog niza ishoda, $\{x_1, \ldots, x_n\}$, koje smo direktno izmerili:
  - Procena očekivane vrednosti je *aritmetička sredina* ovih ishoda:

$$\overline{\mathbb{E}}(X) = \frac{1}{n} \sum_{i=1}^{n} x_i$$

  - Procena varijanse je *aritmetička sredina kvadratnih odstupanja* svakog ishoda od procene očekivane vrednosti[4]:

$$\overline{\mathrm{Var}}(X) = \frac{1}{n} \sum_{i=1}^{n} (x_i - \overline{\mathbb{E}}(X))^2$$

§2.2.4. Algoritam propagacije unazad (posle predavanja)

Kao što je diskutovano u sklopu predavanja, primena algoritma gradijentnog spusta (i njegovih varijanti) na treniranje neuralnih mreža je iterativno ažuriranje težina koristeći formulu

$$\vec{w}_{t+1} \leftarrow \vec{w}_t - \eta \nabla \mathcal{L}(\vec{w})$$

gde je $\nabla \mathcal{L}(\vec{w})$ gradijent *funkcije gubitka*, koju želimo našim odabirom težina $\vec{w}$ da *minimiziramo*. Radi pojednostavljenja analize, smatraćemo da u mreži nema *bias* vrednosti. Da bismo ovo umeli da uradimo, neophodno je odrediti sve vrednosti parcijalnih izvoda

$$\frac{\partial \mathcal{L}(\vec{w})}{\partial w_{i \to j}}$$

(za težinu koja spaja izlaz neurona i sa jednom od ulaza neurona j). Ukoliko ovo umemo da uradimo za *jedan trening primer* $(\vec{x}', y')$, možemo ovo uraditi i za ceo trening skup, kao sumu parcijalnih izvoda po svim primerima odvojeno. Prvi korak (propagacija unapred) podrazumeva ubacivanje ovog primera kao ulaza

[4] Striktno govoreći, ukoliko ne znamo pravu vrednost $\mathbb{E}(X)$, pravilnije je koristiti $\overline{\mathrm{Var}}(X) = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \overline{\mathbb{E}}(X))^2$—međutim, za potrebe ovog predavanja ovo nije neophodno.

za mrežu i izračunavanje $y = h(\vec{w}; \vec{x}')$, uz to da za svaki neuron j beležimo *aktivaciju* a_j i *konačan izlaz* z_j (rezultat primene aktivacione funkcije ovog neurona, σ_j, na aktivaciju):

$$a_j = \sum_k w_{k\to j} z_k$$

$$z_j = \sigma_j(a_j)$$

Zatim, koristimo pravilo izvoda kompozicije:

$$\frac{\partial \mathcal{L}(\vec{w})}{\partial w_{i\to j}} = \frac{\partial \mathcal{L}(\vec{w})}{\partial a_j} \frac{\partial a_j}{\partial w_{i\to j}}$$

Međutim, drugi deo ovog proizvoda lako možemo odrediti:

$$\frac{\partial a_j}{\partial w_{i\to j}} = \frac{\partial}{\partial w_{i\to j}} \left(\sum_k w_{k\to j} z_k \right) = z_i$$

Dakle, ako možemo odrediti veličine

$$\delta_j = \frac{\partial \mathcal{L}(\vec{w})}{\partial a_j}$$

onda smo rešili problem, jer

$$\frac{\partial \mathcal{L}(\vec{w})}{\partial w_{i\to j}} = z_i \delta_j$$

Ove vrednosti možemo izračunati redom od izlaznih ka ulaznim neuronima, unazad (po čemu je ovaj algoritam i dobio ime—propagacija unazad ili *backpropagation*).

Pođimo od slučaja δ_j za izlazne neurone j. Za ove neurone važi da je z_j jedan od izlaza mreže. Možemo primeniti pravilo kompozicije sa ovom promenljivom:

$$\delta_j = \frac{\partial \mathcal{L}(\vec{w})}{\partial a_j} = \frac{\partial \mathcal{L}(\vec{w})}{\partial z_j} \frac{\partial z_j}{\partial a_j}$$

Ovo se može lako izračunati, zato što:

- $\frac{\partial \mathcal{L}(\vec{w})}{\partial z_j}$ je direktan izvod funkcije gubitka za neuron j (a funkcija gubitka uglavnom tretira gubitke kao zbir gubitaka po svakom izlaznom neuronu ponaosob, tako da možemo ignorisati sve ostale u ovoj analizi).
- Pošto je $z_j = \sigma_j(a_j)$, $\frac{\partial z_j}{\partial a_j} = \frac{dz_j}{da_j}$, tj. obični izvod aktivacione funkcije ovog neurona u a_j. Dakle: $\frac{\partial z_j}{\partial a_j} = \frac{d\sigma_j}{dx}(a_j)$.

Napomenimo da su ove veličine lake za izračunavanje (kako za izlazne tako i za sve ostale neurone) *samo ako* biramo funkcije aktivacije i gubitka kojima je izvod po nekoj promenljivoj lako izračunljiv!

Za sve ostale neurone j, promena aktivacije a_j direktno menja neke druge neurone (sve neurone k za koje postoji težina $w_{j\to k}$). Kako računamo δ vrednosti

unazad od izlaznih neurona, i u mreži nema ciklusa, možemo izraziti vrednost δ_j preko (već izračunatih!) vrednosti δ_k za sve ovakve neurone, koristeći, ponovo, pravilo kompozicije:

$$\delta_j = \frac{\partial \mathcal{L}(\vec{w})}{\partial a_j} = \sum_k \frac{\partial \mathcal{L}(\vec{w})}{\partial a_k} \frac{\partial a_k}{\partial a_j} = \sum_k \delta_k \frac{\partial a_k}{\partial a_j}$$

Međutim

$$\frac{\partial a_k}{\partial a_j} = \frac{\partial}{\partial a_j} \left(\sum_i w_{i \to k} \sigma_i(a_i) \right) = w_{j \to k} \frac{d\sigma_j}{dx}(a_j)$$

Kombinovanjem ova dva rezultata dobijamo:

$$\delta_j = \frac{d\sigma_j}{dx}(a_j) \sum_k w_{j \to k} \delta_k$$

čime smo izrazili δ_j preko već poznatih ili izračunatih vrednosti, i time uspešno završili opis ovog algoritma. Da rezimiramo, algoritam propagacije unazad radi sledeće korake, u svrhu računanja izvoda $\frac{\partial \mathcal{L}(\vec{w})}{\partial w_{i \to j}}$ za jedan test primer $(\vec{x}', y')$:

1. *Propagacija unapred*: ubacite $\vec{x}'$ kao ulaz za mrežu i izračunajte sve aktivacije $a_j = \sum_i w_{i \to j} z_i$ i izlaze $z_j = \sigma_j(a_j)$ za sve neurone j u mreži;

2. *Propagacija unazad—bazni slučaj*: za izlazne neurone j:

   $$\delta_j = \frac{d\sigma_j}{dx}(a_j) \frac{\partial \mathcal{L}(\vec{w})}{\partial z_j}$$

   gde se parcijalni izvod funkcije gubitka može direktno izračunati, poredjenjem z_j sa očekivanim izlazom y' (ili delom izlaza, ako postoji više izlaznih neurona).

3. *Propagacija unazad—ostali slučajevi*: za sve ostale neurone j:

   $$\delta_j = \frac{d\sigma_j}{dx}(a_j) \sum_k w_{j \to k} \delta_k$$

   gde se redosled neurona bira tako da su već prethodno izračunate vrednosti δ_k za sve neurone k koji direktno uzimaju izlaz neurona j. Pošto neuralna mreža nema cikluse, ovakav redosled je uvek moguće naći.

4. *Ažuriranje težina*: koristimo algoritam gradijentnog spusta da ažuriramo sve težine $w_{i \to j}$ u mreži:

   $$w_{i \to j} \leftarrow w_{i \to j} - \eta z_i \delta_j$$

§2.2.5. Unakrsna entropija (posle predavanja)

Ovde ćemo demonstrirati kako, koristeći osnovna pravila verovatnoće, izvesti funkciju unakrsne entropije koja se, u kombinaciji sa softmaks izlaznim neuronima, koristi kao funkcija gubitka u gotovo svim primenama neuralnih mreža

na (probabilističku) klasifikaciju—uključujući i primere sa predavanja i radionice.

Kao što smo diskutovali, softmaks neuroni nam, za problem klasifikacije u k klasa, daju izlaze $h(\vec{w};\vec{x})_i = \mathbb{P}(C = i|\vec{w},\vec{x})$.

Pretpostavimo sada da imamo samo jedan trening uzorak, $(\vec{x}', c)$. Idealno bismo voleli da odaberemo težine, $\vec{w}$, u našoj neuralnoj mreži tako da maksimiziramo verovatnoću ovakvog trening uzorka:

$$\mathbb{P}((\vec{x}', c)|\vec{w}) = \mathbb{P}(C = c|\vec{w},\vec{x}')\mathbb{P}(\vec{x}') = h(\vec{w};\vec{x}')_c\mathbb{P}(\vec{x}')$$

Generalizujmo odavde na ceo trening skup $\vec{s} = \{(\vec{x}_1, c_1), \ldots, (\vec{x}_m, c_m)\}$. Uz pretpostavku da su trening uzorci *nezavisni*[5], možemo izraziti verovatnoću ovog trening skupa kao *proizvod* verovatnoća svih trening uzoraka u njemu:

$$\mathbb{P}(\vec{s}|\vec{w}) = \prod_{i=1}^{m} \mathbb{P}((\vec{x}_i, c_i)|\vec{w}) = \prod_{i=1}^{m} h(\vec{w};\vec{x}_i)_{c_i}\mathbb{P}(\vec{x}_i)$$

Konačno, želimo odabrati težine $\vec{w}$ koje maksimizuju ovaj izraz, međutim, optimizacija velikog proizvoda je teška sa stanovišta numeričke stabilnosti. Srećom, postoji jednostavan trik kojim proizvode možemo pretvoriti u sume—optimizujemo *logaritam* ovog izraza umesto samog izraza:

$$\begin{aligned}
\vec{w}' &= \operatorname*{argmax}_{\vec{w}} \mathbb{P}(\vec{s}|\vec{w}) \\
&= \operatorname*{argmax}_{\vec{w}} \prod_{i=1}^{m} h(\vec{w};\vec{x}_i)_{c_i}\mathbb{P}(\vec{x}_i) \\
&= \operatorname*{argmax}_{\vec{w}} \log \prod_{i=1}^{m} h(\vec{w};\vec{x}_i)_{c_i}\mathbb{P}(\vec{x}_i) \\
&= \operatorname*{argmax}_{\vec{w}} \sum_{i=1}^{m} \log h(\vec{w};\vec{x}_i)_{c_i} + \sum_{i=1}^{m} \log \mathbb{P}(\vec{x}_i) \\
&= \operatorname*{argmin}_{\vec{w}} - \sum_{i=1}^{m} \log h(\vec{w};\vec{x}_i)_{c_i}
\end{aligned}$$

U poslednjom redu smo iskoristili nezavisnost $\mathbb{P}(\vec{x}_i)$ od izbora težina, kao i činjenicu da možemo pretvoriti maksimizaciju izraza u minimizaciju suprotnog izraza. Ovo lako možemo pretvoriti u funkciju unakrsne entropije koja je prezentovana na predavanju: za jedan trening primer $(\vec{x}, c)$, uzmimo $\vec{y} = h(\vec{x})$ kao izlaz softmaks sloja u neuralnoj mreži, a $\vec{\hat{y}}$ kao *"one-hot"* kodiranje klase c. Npr. ukoliko je broj klasa $k = 5$, a $c = 1$ (indeksiranje od nule), važi $\vec{\hat{y}} = [0\ 1\ 0\ 0\ 0]$. Tada je dobijeni izraz za unakrsnu entropiju (po jednom trening primeru) jednak

$$\mathcal{L}(\vec{y}, \vec{\hat{y}}) = -\sum_{i=1}^{k} \hat{y}_i \log y_i$$

[5]Ovo je često razumna pretpostavka, ali moramo biti jako pažljivi ukoliko rešavamo probleme u kojima može postojati jaka *veza* između uzoraka, npr. ukoliko su dobijeni kao susedna stanja ekrana u video igrici.

Primetite zašto je ovo ekvivalentno: zato što je $\hat{y}_i = 1$ samo za $i = c$, a nula za sve ostale, ova suma efektivno postaje $-\log y_c = -\log h(\vec{w}; \vec{x})_c$.

§2.2.6. Keras dokumentacija

`https://keras.io`

§2.3. Napredne tehnike kriptografije

§2.3.1. Uvod

Ovaj tekst služi da bi Vas upoznao sa osnovnim pojmovima potrebnim za razumevanje prezentacije "Napredne tehnike kriptografije". Prezentacija će se baviti temama koje se nadograđuju na pojmove koje su ovde opisani. Reč je o osnovnim kriptografskim modelima (eng.: *cryptographic primitives*). Sve napredne, moderne kriptografske šeme se sastoje od njih, a ni jedna kriptografska osnova se, u svom sirovom obliku, *ne sme* koristiti, jer sama od sebe ona nikada ne može dati sve rezultate koji su nam neophodni u praksi. Zbog toga u stvarnom svetom ne govorimo samo o kriptografskim primitivima, već o i šemama koje izgrađuju.

Ovi pojmovi, i njihov istorijski razvoj, su bili detaljno obrađeni na prethodnim Nedeljama informatike. Za razumevanje predavanja o naprednim tehnikama kriptografije nije potrebno detaljno razumevanje načina na koji funkcionišu ove osnove u pravom svetu, već samo šta one to rade, odnosno koju funkciju vrše u kriptosistemima. Ovo se na engleskom zove pristup crne kutije (*black box approach* – jer se pojmu pristupa kao apstraktnoj celini čiji se sadržaj "ne vidi"). Na to ćemo se i fokusirati. Za razumevanje materijala ovde nije potrebno nikakvo posebno predznanje.

§2.3.2. Simetrična kriptografija

Ovde je potrebno napomenuti tri različita pojma koji ćete čuti da se koriste: **kriptologija**, **kriptografija** i **kriptoanaliza**. Bilo koji tehnički rečnik će definisati kriptografiju kao nauku pravljenja kriptografskih sistema, kriptoanalizu kao nauku provaljivanja istih, a kriptologiju kao uniju obe oblasti. Upotreba ovih imena je više zbog tradicije nego korisnosti, pošto se, razvojem kriptologije, kriptografija i kriptoanaliza sve više prepliću, i pošto ne možete raditi prvu bez druge. Zbog toga se ime kriptografija češće koristi nego ime kriptologija da bi prenela isto značenje, a i u ovom dokumentu ćemo pratiti taj trend. Kriptografija je takođe u savremenom govoru sinonim za bilo koji vid **šifrovanja**.

Krenimo od najstarijeg i najjednostavnijeg kriptografskog primitiva. To je šifrovanje simetričnim, odnosno deljenim ključem, dakle simetrična kriptografija. Osnovna ideja je, da dve osobe (tradicionalno **Alisa** i **Bob**) žele da iskoriste već deljenu tajnu kako bi dalje razmenjivali tajne. Nazovimo tu tajnu K, kao ključ (eng.: *key*). Ključ je niz nula i jedinica, pošto bilo koji skup potencijalnih tajni koje dve osobe mogu da dele (neki zajednički datum, skup reči, itd.) se može preslikati u skup prirodnih brojeva, a samim tim i u binarni string.

Dakle Alisa ima vrednost P (*plaintext*) koju želi da prosledi Bobu, ali ne želi da neko u sredini ko presretne njenu poruku, može da nauči ništa korisno o P. To će postići tako što bi izračunala vrednost C (*ciphertext*) na sledeći način:

$$C = E_S(K, P)$$

Gde E_S obeležava algoritam simetričnog **šifrovanja**, odnosno funkciju koja daje šifrat C za dat otvoren tekst P i tajni deljeni ključ K. Kakve bismo osobine želeli da ima funkcija E_S? Prvo, osnovni uslov je da se za dato C ne mogu odrediti ni P ni K, odnosno da niko ko presretne šifrat ne može zaključiti ništa o otvorenom tekstu ili ključu. Sa druge strane, želimo da se za dat ključ, P može lako i šifrovati (kao što je to Alisa gore uradila), ali i C **dešifrovati**. Poruku dešifruje onaj ko ima ključ (Bob) na sledeći način:

$$P = D_S(K, C)$$

D_S je ovde algoritam dešifrovanja. On može biti proceduralno isto što i algoritam šifrovanja E_S (a u praksi često to i jeste slučaj), ali to ovde nije bitno, jer je to pitanje unutrašnjeg funkcionisanja algoritama, a mi ih ovde gledamo kao apstrakcije, to jest samo se fokusiramo na njihovu funkciju.

Dakle ovo je način da Bob jednoznačno dešifruje poruku C ključem i dobije P, a Alisa, koja je videla C i nema ključ to ne može da uradi. Tako da je par algoritama, odnosno funkcija, (E_S, D_S) naš simetrični primitiv, što ćemo predpostaviti da svi znaju, a da se suština sigurnosti gore navedenog postupka baš ta deljena tajna K.

BITNA NAPOMENA: Proces šifrovanja i dešifrovanja ovde je opisan u edukacione svrhe i ne sme se koristiti u praksi. Kada biste razmenjivali poruke na ovaj način, to bi otvorilo vaš sistem mnogim ranjivostima, čak i u slučaju da koristite moderne i isprobane funkcije E_S i D_S, koje su otporne na savremene kriptoanalitičke napade, *kao i* da ste ih softverski i hardverski ispravno implementirali, *i* da komunicirate sa fizički obezbeđene lokacije oslobođene od faktora ljudske greške. Ovako opisan kriptosistem bi za iste parove K i P davao iste poruke (sistem je deterministički jer nismo uveli slučajni ili pseudoslučajni element). Pri razmeni velike količine podataka, pametan napadač bi iz same te činjenice mogao svašta da nauči. Još postoje problemi loše generisanih ključeva, premale količine informacije sadržane u otvorenim teksovima (u oba slučaja bi se grubom silom možda moglo doći do nekih informacija, ili do cele poruke ili ključa), čisto da navedemo neke. Napadi koji ne zavise do tipa šifre koja se koristi zovu se **generički napadi**, i jedini način da se oni izbegnu je da se kriptografski primitiv koristi **kao deo nekog većeg sistema**, kao što su oni koje ćemo diskutovati u prezentaciji. Ova pouka važi i za ostale primitive.

Primer simetričnog algoritma koji se danas koristi je Advanced Encryption Standard (AES).

§2.3.3. Asimetrična kriptografija – RSA

Mlađa ideja koja se pojavila sa razvojem računara se zove šifrovanje asimetričnim ključem, ili asimetrična kriptografija. Ovde ćemo direktno obraditi primer konkretnog sistema koji se danas koristi, a to je Rivest–Šamir–Adelmanov

kriptosistem (RSA). Osnovna ideja je sledeća: postoje dva ključa, jedan javni e i jedan tajni d, koje je Alisa nekako generisala. Ona objavi svoj javni ključ Bobu. Bob iskoristi taj ključ da šifruje poruku m na sledeći način:

$$c \equiv m^e \pmod{n}$$

On je sada šifrovao poruku m i njen šifrat je c. Sad Bob sme da pošalje c preko nesigurnog kanala, jer samo Alisa može da dešifruje poruku svojim tajnim ključem d. Dakle Bob, ili bilo ko sa kim je Alisa podelila javni ključ, može da šifruje poruku, ali samo ona može da je dešifruje na sledeći način:

$$m \equiv c^d \pmod{n}$$
$$m \equiv (m^e)^d \pmod{n}$$

I bez detaljnog razumevanja formalnog odnosa između e, d i n, jasno nam je iz ove jednačine da ed mora da se poništi na neki način (da im je proizvod pod određenim modulom jednak jedinici). Obećali smo da ćemo tretirati kriptografske primitive kao crne kutije, ali smo ovde malo prešli granicu zbog stvaranja osnovnog utiska o tome kako sistemi sa javnim ključem rade, jer u ovom slučaju može biti nezahvalno baratati apstraktnim notacijama bez iskustva sa konkretnim.

§2.3.4. Kriptografske jednosmerne heš funkcije

Postoji više vrsta heševa, ali ćemo se ovde, odnosno u prezentaciji, fokusirati na samo one koje se koriste u kriptografske svrhe. Heš je u suštini algoritam za kodiranje ulaza, često i veoma velike dužine, na uglavnom relativno mali izlaz. Kriptografski heš mora da zadovoljava još neke osobine. Obeležimo A i B kao ulaze proizvoljne dužine, a $h(x)$ kao heš vrednosti x.

1. Jednosmernost: za dato $h(A)$, nepodesivo je odrediti A.
2. Otpornost na koliziju: nepodesivo je naći $A \neq B$ takvo da $h(A) = h(B)$.

Treba primetiti da izlaz heša, iako fiksan, treba biti dovoljno velik. Uzmimo banalan primer gde heš algoritam prosto uzima sve bitove ulaza i sabira ih pri modulu 2. Izlaz je fiksne dužine (1 bit). Primetićemo da je ovo veoma jednosmerna funkcija – ako je izlaz 1, ne možemo odrediti da li je hešovano 1 ili 111 ili 1000110... Sa druge strane, ovaj heš je veoma neotporan na koliziju, jer je veoma lako naći parove koji imaju isti heš a koji su različiti (1111 i 0000). Dakle kriptografski heš teži da bude kao otisak prsta: jedinstven, ali i anoniman bez više informacija.

Za određene nekriptografske svrhe, kao što je izgradnja heš tabela, jednosmernost nije bitna koliko otpornost na koliziju. Kriptografska heš funkcija, osnim navedene dve ključne osobine, mora proći još mnoge pažljivo napravljene statističke testove da bi je naučna zajednica prihvatila kao sigurnu. Zbog toga funkcije koje nama trebaju zovemo punim nazivom kriptografske jednosmerne heš funkcije.

Primer kriptografske jednosmerne heš funkcije koja se danas koristi je Secure Hash Algorithm 3 (SHA3).

§2.3.5. Operacije nad bitovima

Za binarne (logičke) operacije ste sigurno čuli. Dve operacije koje ćemo koristiti u prezentaciji su: $\oplus$ (operacija eksluzivno ili) i $\wedge$ (operacija i). Treba napomenuti da je $\{\oplus, \wedge\}$ *potpun* skup logičkih veznika, što znači da se svaka funkcija može izraziti samo koristeći samo njih. Još jedan razlog zašto je ovaj par tako popularan u literaturi je zato što te operacije predstavljaju sabiranje i množenje bitova pri modulu 2.

$A \oplus B = C$	$A \wedge B = C$
$1 \oplus 1 = 0$	$1 \wedge 1 = 1$
$1 \oplus 0 = 1$	$1 \wedge 0 = 0$
$0 \oplus 1 = 1$	$0 \wedge 1 = 0$
$0 \oplus 0 = 0$	$0 \wedge 0 = 0$

§2.4. Kvantno računarstvo

§2.4.1. Verovatnoća

- Teorija verovatnoće je matematička disciplina koja se bavi slučajnim događajima čiji ishodi nisu strogo definisani.
- Neka je Ω skup svih mogućih ishoda nekog eksperimenta. Ako je ω jedan od mogućih ishoda, onda je $p(\omega)$ verovatnoća da se on dogodi, s tim da važi $0 \leqslant p(\omega) \leqslant 1$. Zbir verovatnoća svih ishoda iz skupa Ω jednak je 1.
- Posmatrajmo konkretan primer, bacanje kockice. Tada je Ω skup svih mogućih dobijenih brojeva, odnosno $\Omega = \{1, 2, 3, 4, 5, 6\}$. Ako pretpostavimo da je verovatoća dobijanja svakog od njih jednaka, onda važi $p(\omega) = \frac{1}{6}$, za svako $\omega \in \Omega$.

§2.4.2. Kompleksni brojevi

- Posmatrajmo jednačinu $x^2 = -1$. Očigledno, ona nema rešenje u skupu realnih brojeva. Imaginarna jedinica i se definiše kao jedno rešenje ove jednačine, tj. $i^2 = -1$ (drugo rešenje je $-i$).
- Skup kompleksnih brojeva je skup čiji su svi elementi oblika $a + bi, a, b \in \mathbb{R}$, gde je i imaginarna jedinica. Kompleksne brojeve možemo posmatrati kao skup uređenih parova $(a, b), a, b \in \mathbb{R}$.
- Konjugat kompleksnog broja $z = a + bi$ je kompleksan broj $\bar{z} = a - bi$.
- Slično kao što kompleksne brojeve možemo posmatrati na *realnoj pravoj*, tako i kompleksne brojeve možemo posmatrati u *kompleksnoj ravni*. U tom slučaju, uređen par (a, b) predstavlja koordinate na x-osi i y-osi kompleksne ravni.

- Moduo kompleksnog broja možemo posmatrati kao *njegovu razdaljunu od koordinatnog početka*, a definisan je kao

$$|z| = |a + bi| = \sqrt{a^2 + b^2} \in \mathbb{R}.$$

Takođe, može se primetiti da važi

$$|z| = \sqrt{z\bar{z}}.$$

§2.4.3. Matrice

- Matricu možemo smatrati *tablicom brojeva* sa m vrsta i n kolona. Transponovana matrica se dobija tako što se početnoj matrici zamene kolone i vrste.

$$A_{m\times n} = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}$$

$$A^T_{n\times m} = \begin{bmatrix} a_{11} & \cdots & a_{m1} \\ \vdots & & \vdots \\ a_{1n} & \cdots & a_{mn} \end{bmatrix}$$

- Dve matrice možemo sabrati samo ako su istih dimenzija. U tom slučaju, njihov zbir je matrica čiji je svaki element zbir njihovih odgovarajućih elemenata.

$$A + B = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11} + b_{11} & a_{12} + b_{12} \\ a_{21} + b_{21} & a_{22} + b_{22} \end{bmatrix}$$

- Ako matricu pomnožimo skalarom dobijamo matricu čiji je svaki element pomnožen njime.

- Ako je matrica A dimenzija $n \times m$, a matrica B dimenzija $m \times k$, onda se one mogu pomnožiti. Njihov proizvod je matrica C dimenzija $n \times k$ čiji je element c_{ij} jednak *proizvodu i-te vrste matrice A i j-te kolone matrice B*, odnosno

$$c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj}.$$

Lako se primećuje da množenje matrica nije komutativno.

$$A \cdot B = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

§2.4.4. Vektori

- Vektor u Dekartovom koordinatnom sistemu možemo smatrati duži koja je sem intenzitetom i pravcem određena i smerom.

- Svaki vektor možemo predstaviti matricom dimenzija $n \times 1$ čiji elementi predstavljaju njegove koordinate u n-dimenzionom prostoru (u Dekartovom koordinatnom sistemu $n = 3$). Elementi te matrice ne moraju biti realni brojevi! U nastavku je dat primer vektora u 4-dimenzionom prostoru čije su koordinate kompleksni brojevi.

$$\begin{bmatrix} i \\ 2+3i \\ -18 \\ 4-2i \end{bmatrix} \in \mathbb{C}^4$$

- Skalarni proizvod dva vektora u n-dimenzionom prostoru definisan je kao

$$\vec{a} \cdot \vec{b} = a^T \cdot b = \sum_{i=1}^{n} a_i b_i.$$

§2.5. Softverske ranjivosti

§2.5.1. O predavanju

Predavanje *Autostoperski vodič kroz softverske ranjivosti* je namenjeno svima koje interesuje kako dolazi do "hakerskih" napada na računarske sisteme i mreže, neretko i poznatih, uglednih kompanija. Osvrnućemo se na neke poznate ranjivosti u softveru koje su ovakve napade omogućile, uz analizu uzroka i mera predostrožnosti koje bi ih (idealno) sprečile. Predznanje koje je potrebno je minimalno, a cilj je iz pristupačnog ugla prikazati softversku bezbednost i pomoći vam da u budućnosti sagledate svoj softver iz perspektive napadača i tako predupredite greške koje mogu koštati milione (ili, čak, ljudske živote).

Kako bi ovaj predgovor bio dovoljno kratak, pojmovi su objašnjeni sa minimalno detalja koji su potrebni da bi se samo predavanje razumelo. Ohrabrujemo vas da, ukoliko želite da saznate više o nekom pojmu, ili vas zbunjuje neki od narednih opisa, potražite pojmove (preporučljivo, na engleskom jeziku) na Internetu, gde ćete naći iscrpna objašnjenja sa slikovitim primerima i dijagramima koji vam mogu pomoći.

§2.5.2. Neophodni pojmovi

U nastavku se nalaze neki osnovni pojmovi sa kojima je potrebno da budete upoznati kako biste pratili predavanje.

- **Softverska ranjivost (engl. software vulnerability)** je neželjeno ponašanje određenog komada softvera (što uključuje, na primer, i biblioteke koje se koriste u drugom softveru, serverski softver, kao i operativne sisteme) koje može dovesti do kompromitacije bezbednosti podataka ili sistema. Ukoliko zlonamerni napadač može, koristeći softversku ranjivost, da natera softver da se ponaša na način koji nije predviđen, on ili ona može s pažnjom (u zavisnosti od lične sposobnosti, prirode same ranjivosti i potencijalnih dodatnih uslova) naterati softver da "oda" poverljive informacije kojima napadač pod normalnim okolnostima ne bi imao pristup, natera softver da izvrši proizvoljan kod ili onemogući softveru da izvršava svoju primarnu funkciju.

- **Virtuelna memorija (engl. virtual memory)**. Dozvoljavati procesima na sistemu direktan pristup fizičkoj (radnoj) memoriji (iliti RAM-u) donosi mnogo problema. Procesi tada mogu menjati bilo koje regione memorije, čak i delove u kojima je smešten kod operativnog sistema, i time ugroziti bezbednost sistema i narušiti izolaciju procesa. Takođe, operativni sistem želi da sam organizuje radnu memoriju na najbolji način. Na primer, zamislimo da u radnoj memoriji postoji prazan prostor od 200 MiB, zatim zauzet prostor od 1 MiB, i konačno još praznog prostora od 300 MiB. Ako program želi da alocira niz od 250 MiB, korisno je da operativni sistem može da mu dodeli 200 MiB prvog praznog prostora i 50 MiB drugog praznog prostora, ali tako da program ima iluziju kao da je dobio kontinuiran blok — tako se izbegava rasipanje radnom memorijom. Zbog ovoga se uvodi *virtuelna memorija* — svakom procesu se dodeljuje poseban *adresni prostor*, odnosno, matematički, preslikavanje iz adresa u procesovoj virtuelnoj memoriji u adrese u stvarnoj, radnoj memoriji. Operativni sistem tako može procesu dodeliti kontinuiran blok u procesovom adresnom prostoru, ali tako da različiti delovi tog bloka odgovaraju potencijalno neuzastopnim blokovima radne memorije. Na ovaj način, operativni sistem može ograničiti procese u smislu memorije kojoj smeju da pristupaju, pa čak im dozvoliti read-only pristup nekim delovima kojima smeju da pristupaju ali ne smeju da ih menjaju. Takođe, ovo omogućava operativnom sistemu da blokove memorije koji se trenutno ne koriste privremeno pomeri na disk kako bi se radna memorija oslobodila za nešto važnije, i zatim pročita te blokove tek kada proces proba da im pristupi.

- **Stek (engl. stack)** je oblast memorije na kojoj se čuvaju određeni podaci prilikom izvršavanja programa. Na steku se pamte mesta sa kojih su funkcije pozvane, kako bi, nakon komande *return*, funkcija "znala" na koje mesto da vrati izvršavanje programa. Takođe, stek se koristi i za pamćenje privremenih promenljivih (uključujući i nizove) koje deklarišete u telu funkcije. Na procesorima iz x86 familije i drugim procesorima koji koriste x86 familiju arhitekturi (koju verovatno koristi vaš računar), stek se tipično postavlja na visoku memorijsku lokaciju (unutar virtuelne memorije — nevezano za to gde se on zaista nalazi u fizičkoj, radnoj memoriji) i "raste" nadole — dakle, podaci skorije ubačeni na stek se nalaze na *nižim* virtuelnim memorijskim lokacijama.

- **Hip (engl. heap)** je oblast memorije u kojoj se čuvaju promenljive i nizovi napravljeni dinamičkom alokacijom memorije (na primer, pomoću `new` u C++-u ili `malloc` u C-u). Hip se na x86 i srodnim arhitekturama tipično postavlja na nižu (virtuelnu) memorijsku lokaciju i "raste" nagore — novije alocirani blokovi memorije se nalaze na višim virtuelnim memorijskim lokacijama. Kada se hip i stek "susretnu" — proces nema više virtuelne memorije!

- **Kernelski i korisnički prostor (engl. kernel space and user space)**. Svi moderni procesori dozvoljavaju da se instrukcije izvršavaju u nekoliko režima — u cilju jednostavnosti, pretpostavićemo da ih ima dva, i to *kernelski* i *korisnički*. U kernelskom režimu izvršava se privilegovan kod, koji je uglavnom kernel (srž) operativnog sistema, i u ovom režimu dozvoljeno je sve — komunikacija sa hardverom, konfiguracija samog procesora, menjanje tabele virtuelne memorije, i slično. U korisničkom režimu moguće je samo upisivati i čitati sa lokacija u virtuelnoj memoriji i izvršavati instrukcije na bezbedan način, izolovano od ostatka sistema. Operativni sistem, tipično, prebaci procesor u korisnički režim pre nego što se započne izvršavanje koda nekog procesa, a proces (specijalnim procesorskim instrukcijama) traži od operativnog sistema da izvrši privilegovane operacije umesto njega (komunikacija sa hardverom, upisivanje u fajlove, i sl.). Operativni sistem tada, iz kernelskog režima, ima priliku da proveri da li proces ima dozvolu da izvede neku akciju, i odbije zahtev ukoliko to nije slučaj, čime se osigurava bezbednost sistema. *Kernelski prostor* se koristi ili kao sinonim za kernelski režim, ili kao skup svog koda koji se izvršava u kernelskom režimu, na nekom sistemu. (Značenje je uglavnom jasno iz konteksta). Analogno, *korisnički prostor* se koristi ili kao sinonim za korisnički režim, ili kao skup svog aplikativnog softvera (svega što nije deo operativnog sistema) na sistemu.

- **Race condition**. *Race condition* je, jednostavno rečeno, situacija u kojoj više procesa mogu doći do različitih rezultata u zavisnosti od tajminga. Zamislimo da u čitaonici postoje dva portira koji dodeljuju studentima slobodna mesta, ili im govore da sačekaju ukoliko slobodna mesta ne postoje. Zamislimo da oba portira imaju program koji pristupa jednoj bazi sa popisom svih mesta i stanjem (zauzeto, slobodno). Ako portiri rade nezavisno i jedino interaguju kroz bazu, može doći do sledećih događaja:

  1. Student A dolazi kod portira 1 i traži mesto.
  2. Student B dolazi kod portira 2 i traži mesto.
  3. Portir 1 proveri stanje u bazi i zaključi da je jedno jedino mesto, X, slobodno.
  4. Portir 2 proveri stanje u bazi i zaključi da je to isto mesto X jedino slobodno.
  5. Portir 1 dodeljuje studentu 1 mesto X, i u bazi ga označava kao zauzeto.
  6. Portir 2 dodeljuje studentu 2 isto mesto, i u bazi ga označava kao zauzeto, iako je u međuvremenu ono dodeljeno drugom studentu.

U ovom slučaju, konačni efekat ovog “algoritma” zavisi od redosleda operacija: da je portir 1 dovoljno brzo uneo u bazu da je mesto sada zauzeto, portir 2 bi na vreme video informaciju i obavestio studenta 2 da nema slobodnih mesta.

Kako operativni sistemi redom izvršavaju instrukcije različitih procesa kako bi stotine procesa moglo da radi prividno istovremeno na jednom procesoru (na primer, izvršava se 10 instrukcija procesa 1, 10 instrukcija procesa 2, i tako do kraja liste procesa, kada se izvršava sledećih 10 instrukcija procesa 1, itd.), prost tajming može uticati na finalni ishod algoritma.

§2.6. Arduino radionica

Veći deo ovog materijala smo pokrili na prethodnoj Nedelji informatike, u okviru predavanja "Uvod u programiranje mikrokontrolera: Arduino" Lazara Mitrovića. Na našoj veb stranici se nalaze relevantni slajdovi:

`http://www.csnedelja.mg.edu.rs/materijali`

§2.6.1. Malo elektronike

Najjednostavniji elementi strujnih kola jednosmerne struje su **provodnici** i **otpornici**. Provodnici se na šemama predstavljaju "punim" linijama, dok se otpornici u elektrotehnici označavaju ili "cik-cak" linijama ili kao pravougaonici. Osnovna karakteristika jednog otpornika je njegov **otpor**, u oznaci R, a jedinica otpora je **om** (Ω). U digitalnoj elektronici se uglavnom sreću otpori reda veličine $1\,\mathrm{k}\Omega$. Provodnici se često smatraju idealnim i nemaju sopstveni otpor.

Otpornici koje ćemo koristiti izgledaju kao na slici. Postoji konvencija za njihovo obeležavanje koja govori o redu veličine otpora, ali mi ćemo vam dati "normalne" oznake na radionici:

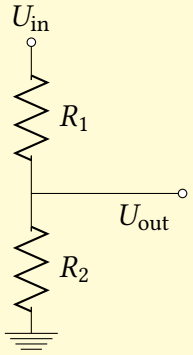
Slika 2.1: Otpornici

U kolu nastaje **kratak spoj** ukoliko stvorimo razliku potencijala uz premali (maltene nulti) otpor. Tada je struja u tom kolu izuzetno velika i često dolazi do kvara uređaja. Imajte ovo u vidu!

Korisno bi bilo navesti i koncept **mase** ili **uzemljenja** (*ground*): to je tačka u kojoj je potencijal jednak nuli. U strujnim kolima se redovno koristi da bi se istaklo da su neke tačke na jednakom potencijalu. Oznaka za masu je niz crta čiji je pravac normalan na pravac provodnika koji vodi do te mase.

Primer 2.6.1

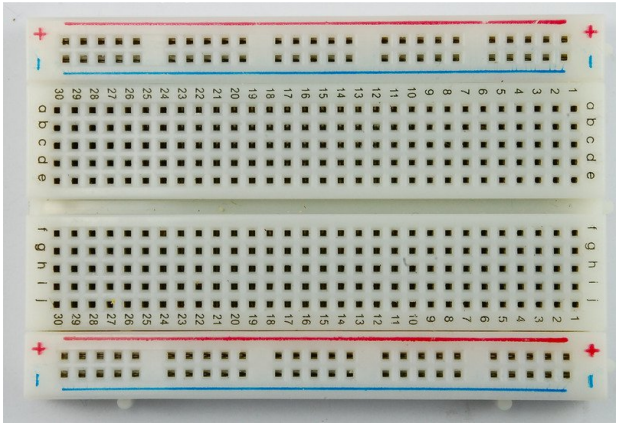
Naredno kolo služi za generisanje nekog izlaznog potencijala/napona (U_{out}) koji je udeo ulaznog (U_{in}):



Na osnovu elementarnih zakona električne struje može da se ustanovi da je:

$$U_{\text{out}} = \frac{R_2}{R_1 + R_2} U_{\text{in}}$$

Za prototipiranje jednog strujnog kola bi bilo previše zahtevno vršiti lemljenje svaki put kada se unese izmena u kolo. Umesto toga koristimo **protobord** ili **bredbord** (*breadboard*). Elementi strujnog kola (otpornici, žice, diode itd) se povezuju ubadanjem u rupice. Uočimo nizove od po pet rupica: ti kontakti su *kratko spojeni*. Korišćenjem ove činjenice ne pravimo gužvu pri sastavljanju kola. Izuzetak od ovoga su **magistrale** koje se nalaze na krajevima protoborda: tu je čitav niz (po polovini dužine protoborda) kratko spojen.



Slika 2.2: Bredbord

Prilikom vezivanja kola trudimo se da koliko-toliko liče na odgovarajuće šeme! Srećom, nijedno kolo koje ćete sklapati neće biti mnogo komplikovano, tako da malo toga može da počne naopako.[6]

Još jedan element strujnih kola koji ćemo koristiti će biti **dioda**. Diode se prave od poluprovodnih materijala (poput silicijuma) i, u najuprošćenijem mo-

[6] *Famous last words…*

gućem modelu, omogućavaju protok struje *u samo jednom smeru*, od **anode** do **katode**. Nama će od interesa biti **LED**-ovi, odnosno diode koje emituju svetlost kada kroz njih teče struja. Diode mogu da pregore ukoliko kroz njih teče velika struja, tako da bi trebalo da budemo oprezni.

Slika 2.3: LE dioda

Primetimo da je jedna "noga" LED-a duža: to je anoda.

Slika 2.4: Oznaka za diodu u strujnom kolu

§2.6.2. Šta je Arduino?

Mikrokontroler je minijaturni računar koji je implementiran na jednom integrisanom kolu, i pri tome sadrži procesor, memoriju i periferije. Ključna razlika u odnosu na jedan "legitmni" računar jeste činjenica da je skup zadataka koje mikrokontroler može da obavi jako mali, često je u pitanju neka relativno jednostavna operacija "niskog nivoa". Ipak, zbog ovog su manji, jeftiniji i izuzetno primenljivi u različitim oblastima (bežična komunikacija, robotika, navigacija, embeded sistemi itd), gde te naredbe "niskog nivoa"treba brzo da se obave.

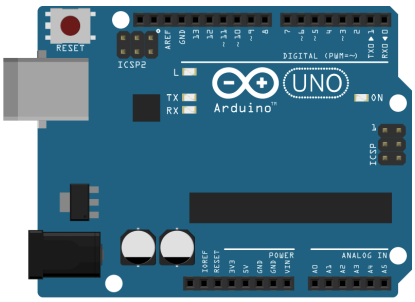
Slika 2.5: Mikrokontroler

Mikrokontroleri se nalaze u velikom broju uređaja: veš mašinama, daljinskim upravljačima, štampačima, automobilima… Dakle, glavna primena je u kontroli perifernih komponenata.

Mikrokontroler u svojoj memoriji sadrži i program koji izvršava, koji se može pomoću specijalnih programatora "staviti" u memoriju. Nama će od interesa biti razvojne ploče, koje pružaju neke dodatne mogućnosti.

Arduino je *open-source* platforma koja služi za prototipiranje mikrokontrolera. Pravljen je sa namerom da bude izuzetno jednostavan i da mogu brzo da se naprave funkcionalni sistemi. Sastoji se od programskog jezika, razvojnih ploča, dodataka (*shield*-ova) i dodatnih alata. Programski jezik koji se koristi za Arduino je sličan jezicima C-ovske familije (ali je daleko jednostavniji!) i poseduje svoj karakterističan API tj. poseduje ugrađene metode pravljene posebno

za naše potrebe. Odlučili smo se za rad sa Arduino jer ima i odličnu dokumentaciju.



Slika 2.6: Arduino Uno

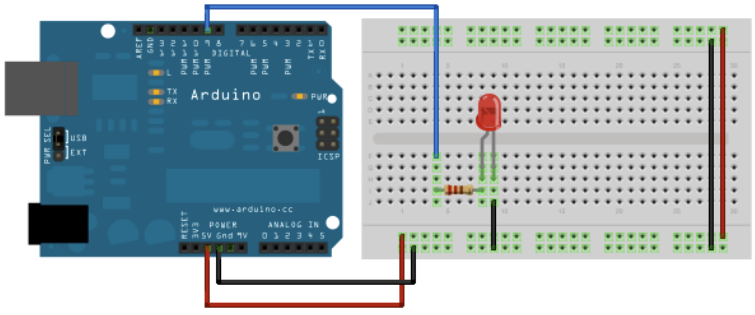
Da bismo mogli da radimo sa razvojnom pločom, neophodni su nam neki alati na računarima: odgovarajući drajveri i razvojno okruženje. Oni su već instalirani na školskim računarima i ove faze podešavanja ćemo vas poštedeti u toku same radionice.

Da biste dobili osećaj o tome šta ćemo raditi, kao i da biste videli mnogo lepih slika, posetite:

- `www.arduino.cc`
- `www.processing.org`

§2.6.3. Programiranje u Arduino platformi

U okviru ove radionice se očekuje da ćete koristiti dokumentaciju i uputstva koja ćemo vam pripremiti za vežbe. Ovde ćemo proći kroz implementaciju prvog zadatka koji ćemo raditi: `https://www.arduino.cc/en/Tutorial/Fade`



Slika 2.7: Kolo u Fade primeru

Ovo kolo se sastoji od otpornika, LE diode i koristi Arduino razvojnu ploču kao izvor napajanja. Sam mikrokontroler, kada je na toj ploči, određuje vrednost ulaznog napona ovog kola (plava žica, pin 9). Primetimo da masa (GND) u okviru ove ploče predstavlja i masu sklopljenog kola, a da se napajanje od 5 V ne koristi ni za šta (efektivno nije deo kola). Što je veća vrednost ulaznog napona, to veća struja teče kroz LED i intenzitet emitovane svetlosti je takođe veća.

Cilj je da intenzitet LED svetla osciluje između stanja u kom je potpuno ugašen i potpuno upaljen. Arduino program koji to postiže je sledeći:

```
int led = 9;
int brightness = 0;
int fadeAmount = 5;

void setup() {
  pinMode(led, OUTPUT);
}

void loop() {
  analogWrite(led, brightness);
  brightness = brightness + fadeAmount;
  if (brightness <= 0 || brightness >= 255) {
    fadeAmount = -fadeAmount;
  }
  delay(30);
}
```

Program za Arduino mikrokontroler sadrži bar dva potprograma: `setup()` i `loop()`.

- Potprogram `setup()` se poziva jednom: kada mikrokontroler dobije izvor napajanja (ili se resetuje). U ovom slučaju, inicijalno treba da se podesi da će pin 9 biti OUTPUT, odnosno da će na pinu 9 mikrokontroler da proizvodi neki izlaz (nasuprot tome, pin je INPUT ukoliko služi kao ulaz program u mikrokontroleru, tipičan primer je očitavanje podataka sa senzora).

- Potprogram `loop()` se poziva *stalno*: nakon `setup()`, počinje beskonačan ciklus koji u svom telu izvršava `loop()`. Ovde je zadatak `loop()` da generiše na izlazu (ovde pin 9) neki promenljivi napon, kao i da ažurira taj napon. Metoda `analogWrite(pin, value)` uzima za argumente broj pina za izlaz, kao i jedan broj koji predstavlja napon koji će se generisati. Ovde `value=0` odgovara izlazu 0 V, a `value=255` izlazu 5 V, između važi linearna zavisnost, a izvan ovog intervala je ili 0 V ili 5 V. Postoji jednostavna kontrola osvetljenosti (promenljive `brightness` i `fadeAmount`), kao neophodan `delay(30)`, što znači da se ubacuje "pauza" od 30 milisekundi do izvrsavanja naredne iteracije `loop()`.

Ovo je prilično jednostavan primer, ali u sebi sadrži i jedan detalj koji bi valjalo pomenuti. Za nas `analogWrite(pin, value)` samo generiše neki napon iz intervala. Međutim, to nije baš tačno: ovo generiše digitalni signal koji predstavlja **impulsno-širinsku modulaciju** signala koji generišemo (eng. *pulse-width modulation*, ili skraćeno PWM): signal koji uzima samo dve vrednosti (0 V i 5 V) koje se smenjuju, i na taj način dovoljno dobro "aproksimiraju" traženi signal. Ovaj detalj je ključan: kada kažemo 3 V, to zapravo znači "malo 5 V, malo manje 0 V".