UVOD V RAČUNALNIŠTVO

1. Pametne Specializacije

Je program, ki temelji na podjetniškem odkrivanju. Z namenom, da ugotovijo katere stvari posamezniki in podjetja potrebujejo in na kakšen način lahko zadovoljimo porabo sodobnih programov in prog. opremo.

Pametne aplikacije = Običajno temeljijona internetu stvari (IoT), umetni inteligenci, računalniku v oblaku, na robu (edge) ter na tehnologijah veriženja blokov (Blockchain).

Aplikacije, ki imajo še posebej izražene uporabniške, aplikacijske in sistemske zahteve (potrebujejo več pomnilnika za delovanje, precej računajo, so velike in na njihovo delovanje vpliva veliko faktorjev).

Računalništvo v megli = skupno ime za vse te tehnologije. Prednost je v hitrosti. Gre za aplikacije, ki imajo še posebej izražene uporabniške, aplikacijske in sistemske zahteve. Temeljijo na IOT, umetni inteligenci in na tehnologijah veriženja blokov. Rač. v megli je skupno ime za vse te tehnologije.

User (signal) → Cloud → (info) User

Orkestracija = vse aplikacije lahko s pomočjo programa prenašamo iz enega na drugo mesto (mikrostoritve).

Področja pametne specializacije: *zdravstvo, gospodarstvo, mobilnost, javne storitve, industrija, turizem in energija*

Zdravsto: Smart healthcare s pomočjo IoT obdeluje podatke v oblaku in lahko zdravnik zdravi na mestu. Problem lahko nastane, ko ta tehnoloija zataji.

Gospodarstvo: uporabljajo robo-svetovalce, da dajejo strankam finančne nasvete tako, da pridobi cilje in premoženje strank. Če gospodarstvo ne napreduje se družba ustavi kar privede do krize.

Mobilnost: Samovožeči avtomobili. Strojno učenje v oblaku.

Javne storitve: Mestni avtobusi brez voznikov

Energija in trajnost: Pametni domovi, ki uporabljajo AI in strojno učenje s tem več prihraniš na elektriko

Industrija 4.0: Samovožeči vožički v podjetjih. Reševanje problem preden pride do njih

Turizem: Booking, AltexSoft in Winding Tree uporabljajo tehnologijo *Blockchain* in *strojno učenje* za rezervacijo prenočišč in z napovedovanjem cen letalskih kart.

Izobraževanje: S pomočjo *pametnih vsebin* se študentje lažje učijo.

Primeri pametnih aplkacij in okolij

analiziraj apk na trgu
complexity science
napredne rešitve

Opis aplikacij

Uporabniška zgodba (tako kot če bi bila aplikacija že izdelana). Opis problemov, ki jih apk rešuje s postopki.

Analiza video tokov (veliko vhodnih tokov; podatki se zbirajo v pb - baza znanja, ki nam nudi neke info) - konceptualni načrt

Rač. megla: sopomenka za negotovost (zaupranje tehnologiji) alt za rač. v oblaku, ki postavlja vrste transakcij in virov na rob omrežja.

Sematično kompleksne in dinamične aplikacije:

- ravnanje s podatki, tako da ne porabijo veliko energije in nudijo zasebnost
- apk morajo biti heterogene (delajo na več napravah hkrati)
- apk so lahko na večiih mestih (oblak, edge - rob omrežja)
- z blockchainom orkestracija od oblaka do roba
- garantirati mora povezljovst (povezljivost med plastmi)
- **Kibernetika** zaupanja vredni dogodki in zaupanja vredne reakcije na te dogodke
- porazdeljena umetna inteligenca

primeri: v gradbeništvu (varnost pri delu, upravljanje pri gradbišču, nadzor nad gradnjo, zgodnja opozorila in upravljanje z viri)
planiranje → procesi → natančnost → stroški → nadzor → sodelovanje

---

Analiza zahtev

keywords
wireframe - 2D ilustracija spletne strani, ki se osredotoča na prostor in prioriteto vsebine ter na funkcionalnost spletne strani.

Definiicja problema
Namen = določiti cilje, obseg in obliko dela, zagooviti podporo uporabnikom in redno vzdrževati oz. pogodba s podjetjem za vzdrževanje.

Aktivnosti = ugotavljanje in dokumentiranje uporabnikovih zahtev, določiti okolje v katerem bo apk delovala, najširše soglasje glede zahtev.

Uporabnikove zahteve:

- **funkcionalnost** aplikacije (operacije, ki jih uporabnik želi implementirati)
- **omejitve** (protokoli, standradi, strojna oprema)
- **podpora** (uvajanje, nivo in tip podpore)

Zahteve uporabnikov

> stabilnost, vir, preverljivost, prioriteta, potrebnost zahteve (relativna), identifikacijska koda

Aktivnosti = izdelava implementacijsko neodvisnega logčinega modela iz katerega lahko izpeljemo specifične zahteve programske opreme)

Naslednje faze:
načrtovanje, implementacija, testiranje (benchmark), pilot, vzdrževanje

Energetska učinkovitost
Trajnost naprav dosežemo s koncepti, ki imajo poleg zahtev za energetsko učinkovitost upoštevane tudi druge ne funkcionalne zahteve

Korektnost
vključenost uporabnika prinese do razvoja najrazličnejših porogramov
večja kvaliteta izdelkov, zadovoljni uporabniki

Zasebnost
Največja mera zasebnosti mora biti zagotovljena.
Bolj kompleksna apk težje to zagotavljamo (večja šibkost → več možnosti za napad)
DOS, phishing

Skladnost
sistemi vplivajo na vsakdanjo življenje
problem → vgrajevanje nadzornih sistemov znotraj EU
GDPR varovanje osebnih podatkov
problemi iskanja ravnovesja med šibkostimi sistema in državnimi normami in pravili
velika razhajanja med pravili posameznih držav - ko želimo naš produkt uporabljati na globalni ravni

Varnost podatkov, jamstvo
jamstvo varnosti našega produkta
Visokonivojske zahteve: metapodatki, varnost, zasebnost, baza znanja, modeli AI

Sematičnoa kompleksnost - povezanost?

Zaupranje v meglo (slika)

Povzetek

- Poznati prednosti novih tehnlogoij
- analizirati poslovne rpriložnosti
- želja uporabnikov
- izdelati načrt
- analiziramo vse zahteve

2. Računalnik kot stroj

Zgodovina računalništva

zgodnje obdobje do leta 1940:

- avtomatske statve,
- programbilne z luknjastimi kartiami
- prvi programabilni stroj
- Mehanska računala(neprograbilni, osnovna matematika)

Mehanski računalniki → Charles Babbage

- diferenčni stroj(polinomske funckije),
- analitični stroj (poljubne matematične funkcije)
  prva programerka (Ada Byron → program za izračun **Bernoulijevih števil**)
- Programabilni stroj za procesiranje podatkov shranjenih na luknjastih karticah

Rojstvo rač (40 -50)
Elektromehanske naprave (programabilni računalnik)
Elektronski računalnik - Atanasoff
Von Neumannova arhitektura (skupni pomnilnik za program in podatke)

Moderno obdboje

1. generacija (elektronke, Univac)
2. generacija (tranzistorji, prvi diski, prvi OS)
3. generacija (integrirana vezja, prvi mini računalnik,)
4. generacija (strojni jezik, IBMP PC, prva omrežja, grafični uporabniški vmesniki)
5. generacoja (superračunalniki, pametni telefoni, multimedija, rač. v oblaku)

Razvoj stikal - pomembna komponenta računalnika (rele, elektronka, tranzistor)

Moorov zakon: št. elementov, ki jih lahko vgradimo na določen čip (tiskano vezje), se podvoji vsakih 18-24 mesecov, pri tem cena ostane nespremenjena.

Model računalnika

- Funkcionalne enote organizacije računalniškega sistema
- Hierharhija abstrakcij:
    - tranzistorji (omogočajo izračune raznh operacij)
    - vrata (funkcija, ki opravi logične operacije)
    - vezja
- Višji nivo abstrakcij

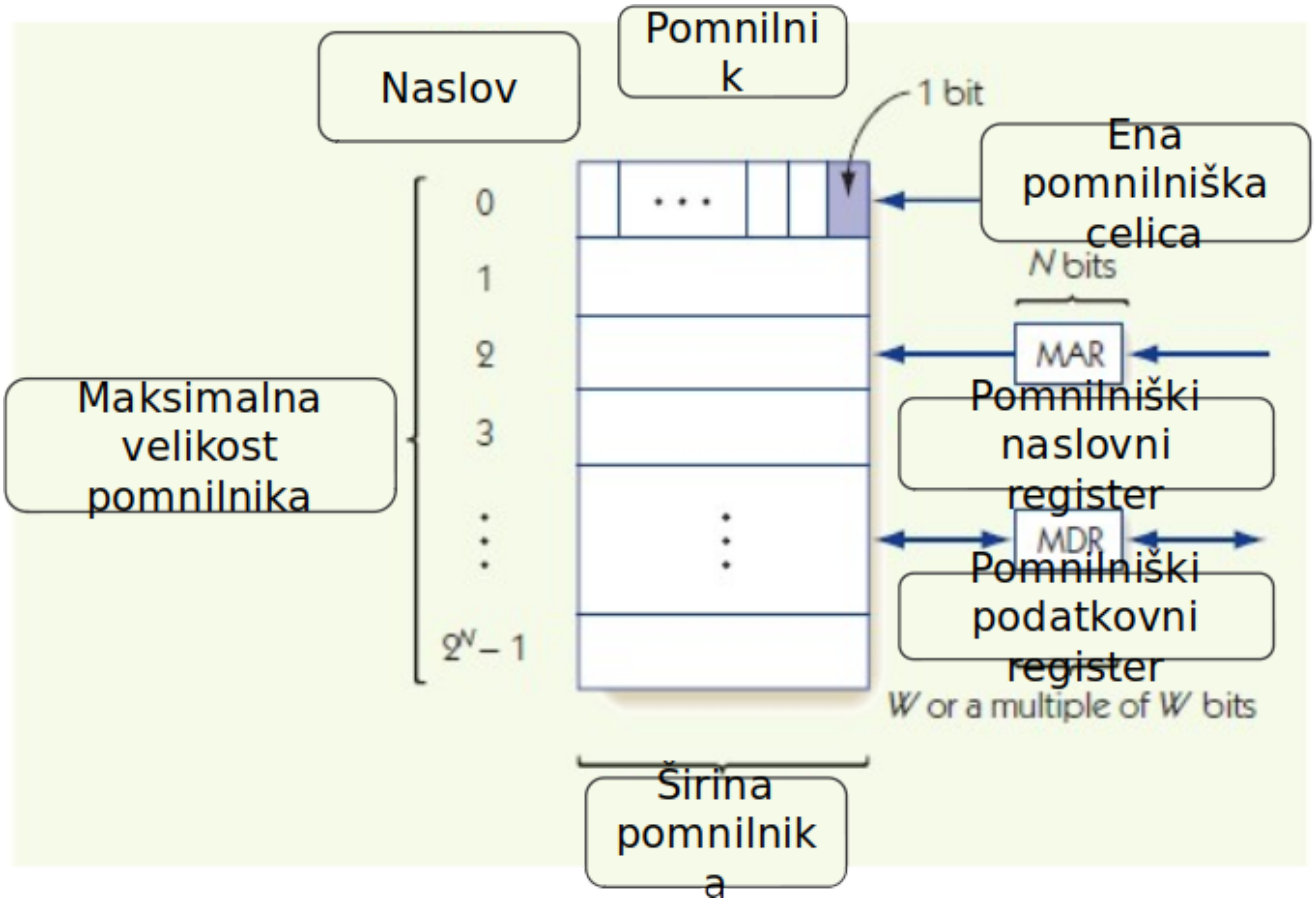
Komponente računalniškega sistema

Von Neumannova arhitektura

- Osnova za skoraj vse moderne računalnike (za probleme lahko zapišemo neki algoritem)

Značilnosti:

- štirje glavni podsistemi:
    - pomnilnik
    - vrata/izhod sistem
    - aritmetino-logična enota
    - krmilna enota (CPE) prenaša podatke po vodilo, komu so podatki namenjeni
- koncept shranjevanja program (v pomnilnik)
- zaporedno izvajanje ukazov



Pomnilnik

Funkcijska enota za shranjevanje in branje podatkov
Pomnilnik z naključnim dostopom
RAM - Random Access Memory
- celice z naslovi (0 in 1)
- enak časa dostopa do vseh celic
- branje in spreminanje vsebin celic
biti se porabijo za shranjevanje naslovov

Naslov in vsebina pom. lokacije
Dve osnovni pom. lokaciji:

- branje: value = fetch(address)
- pisanje: Store(address, value)

MAR - Memory Address Register: naslovni register vsebuje naslov pomnilnikške lokacije
MDR - Memory Data Register: podatkovni register prejme podatke, oz. vsebuje podatke za pisanje

Branje in pisanje

Branje s pom. lokacijo:

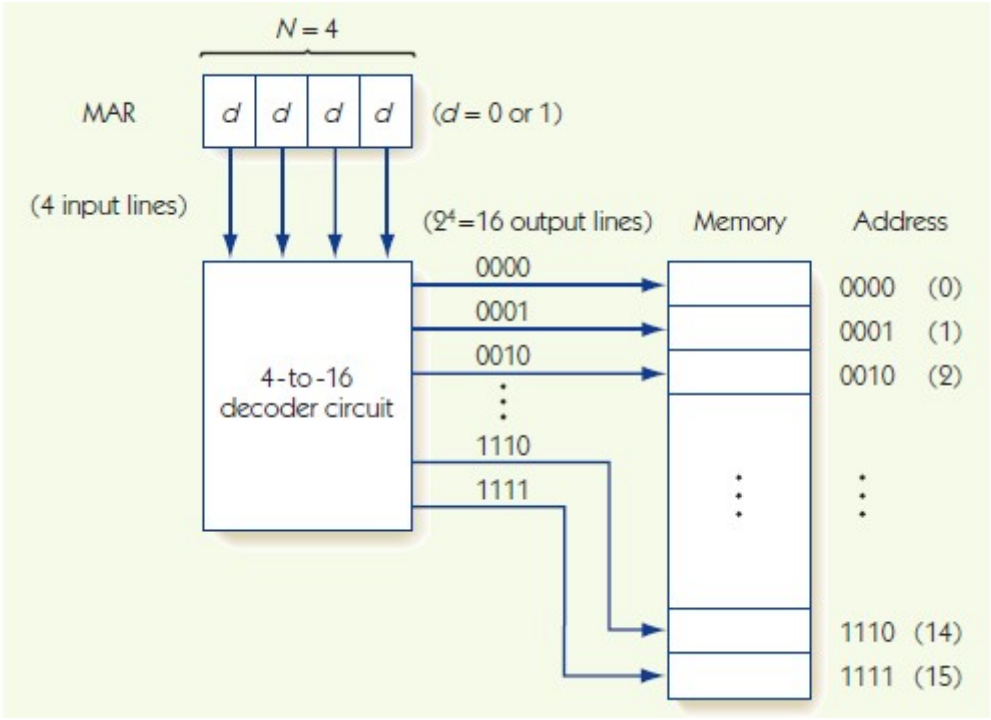
- 1. Naloži naslov v MAR (CPE naloži naslov)
- 2. Dekodira naslov v MAR
- 3. Kopiraj vsebino v pomnilniško lokacijo v MDR

Pisanje na pomnilniko lokacijo:

- 1. Naloži naslov v MAR
- 2. Naloži vrednost v MAR
- 3. Dekodiraj naslov v MAR
- 4. Shrani vsebino MDR na to pomnilniško lokacijo

Naslavljanje

Dekodirnik pretvori MAR v signal za specifično pomnilniško lokacijo (omogoči eno od pomnilniških lokacij za uporabo, podatek se prepiše iz MDR npr.)



Razdelitev naslovnega prostora na dva dela. zgornji in spodnji

Dvodimenzionalna pomnilniška organizacija, eno celico lahko izberemo bolj učinkovito, s tem naredimo bolj učinkovit pomnilnik

Krmilnik za branje/pisanje
Celotno vezje
Preko teh dveh registrov lahko dostopamo do podatke, in jih lahko tudi prenašamo po vodilo do CPE. Dostop do pomnilnika je ponavad ipočasnejši. Pretakanje podatkov vzame veliko časa. Širina vodila pomembna. Večja širina pomeni hitrejši prenos podatkov.

Predpomnilnik

Von Neumannov ozko grlo

- pomnilnik RAM je počasen v primerjavi s proersojem, pretakanje po korakih - naprej ukazi nato podatki

Hiter pomnilnik je zelo drag
Rešitev: dvo-nivojski pomnilnik: RAM + predpomnilnik (cache)
Princip lokalnosti
- velika verjetnost je, da bo isti podatek kmalu ponovno uporabljen
- velika verjetnost je, da bodo kmalu uporabljeni podatki, ko si blizu (po naslovu) trenutno uporabljenemu podatku

Uporaba predpomnilnika:
- Poglej v predppomnilk in uporabi podatek, če je tam
- če ga ni, dostopaj do pomnilnika
- Kopraj še k naslednjih podatkov iz pomnilnika RAM v predpomnilnik in zavrzi stare podatke

Tipičen predpomnilnik je 5 - 10 krat hitreši od RAMa
Zelo pomemben je odstotek zadetkov podatkov že v predpomnilniku (cache bit rate)

Vhodno - izhodne naprave

Dve skupini:

- Naprave namenjene prenosu informacij med CPE in glavnim pomnilnikom ter zunanjim svetom
  - Človek: tipkovnica, zaslon, miškka, zvočniki
  - Računalnik: omrežna kartica, TK linija
- Pomožni pomnilnik za shranjevanje podatkov
  - USB, CD, DVD, prenosni diski,...

Notranji pomnilnik:

- RAM: notranji pomnilnik, podatki se po izklopu rač. izgubijo, pišemo in beremo
- ROM: read-only-memory → lahko ga beremo, BIOS, tovarniško trajno zapisani podatki in ukazi

Zunanji pomnilnik - trajni:

- naprave za shranjevanje z neposrednim dostopom
  - DASD (Direct Access Storage Devices)
  - vsaka lokacija ima naslov, ki ga lahko neposredno dosežemo (zunanji pomnilnik)
  - trdi disk, CD, DVD (čas dostopa ni uniformen)
  - SSD disk (čas dostopa je uniformen)
  - naprave za shranjevanje z zaporednim dostopom

- SASD (Sequential access storage devices)
- ne moremo dostopati do lokacij neposredno ampak samo zaporedno
- magnetni trak

Naprave z neposrednim dostopom

Sestavni deli:

- površina (surface) vsebuje več sledi
- sledi (tracks) koncretični krogi
- sektorji (sectors) naslovljivi segmenti na sledi z vnaprej določeno dolžino
- bralna pisalna glava

Čas dostopa:

- čas iskanja (seek time)
  - premik glave na sled
- zakasnitev (latency time)
  - rotacija začetka sektorja pod glavo
- čas preosa (transfer time)
  - rotacija celega sektorja pod glavo

V/I Krmilnik

V/I naprave sp več razredov počasnejše od RAMa, krmili V/I naprave, osvobodi procesor

Aritmetična logična enota

ALE je del procesorja (skupaj s krmilno enoto)
Vsebuje vezja: za aritmetiko, primerjanje in logiko
Vsebuje registre (izredno hitre namenske pom. enote povezane z vezjem ALE)
Podatkovna pot (Data Path) kako potuje informacija v ALE
- iz registrov na vezja
- iz vezij nazaj na registre0

Vezja ALE

opcija 2 bolj uporabljena: poženejo se vsa vezja na koncu pa se izvede samo ta pravi rezultat

Uporaba **multiplekserja** za izbor pravega rezultata ALE (Podatki pridejo na registre ALE, signal pride na multiplekser in signalizira pravo operacijo, rezultat se zapiše nazaj v register od koder gre ven iz ALE)

Krmilna enota

Program je shranjen v pomnilniku
- Zaporedje ukazov v strojnem jeziku

Krmilna enota

- prebere iz pomnilnika

Strojni ukazi

Format ukaza v strojnem jeziku:
- koda operacije
- naslovi pomnilniških lokacij

primer:

- op code: 9
- naslov X: 99
- naslov Y: 100

Nabor strojnih ukazov

RICS; CISC

Strojni ukazi so implementirani za posamezen čip.
RISC: malo ukazov, ki se zelo hitro izvedejo, enostaven dizajn strojne opreme
CISC: velika množica ukazov, tudi bolj kompleksnih, kompleksn dizjan
Moderni rač. uporabljajo oboje.
Skupine strojnih ukazov: prenos podatkov, aritmetika, primerjanje, vejitve

Prenos podatkov - iz pom. v CPE in obratno, brez te funkcije je stroj okrnjen
vejitve - pomemben element vseh programov, posamezne vejitve se prenašajo na posamezne programe

Registri in vezja krmilne enote

Programski števec (program counter)
- vsebuje naslov naslednjega ukaza

Ukazni register (instruction register)
- vsebuje kodo trenutnega ukaza

Vezja za dekodiranje ukaza
- dekodira kdoo in pošilja signale vezjem za posamezne ukaze

Von Neumannov cikel

Cikel naloži-dekodiraj-izvedi:

1. Faza nalaganja
    - naloži naslednji ukaz iz pomnilnika
2. Faza dekodiranja ukaza
3. Faza izvajanja
    - drugačna za vsak ukaz

Ne-Von Neumannov arhitekture

Rast hitrosti procesorjev ni več eksponentna
Fizikalne omejitve:

- hitrost svetlobe
- minimalna bližina vrat
- segrevanje vezja
  Problemi, ki jih rešujemo so vse večji
  Von Neumanovo ozko grlo (ločitev CPE od pomnilnika, prepustnost podatkov je majhna glede na velikost pomnilnika)

Paralelno procesiranje

Pohitritev in nadgradnja. Nadgradnja pomeni da se velikost problema vsakič poveča.
MPJI - protokol
Rešitev je paralelno procesiranje
Večje število procesorjev:
- v preteklosti super računalniki
- več jedrni procesorji(dual-core, quad-core)
- na tisočine procesorjev

Dva pristopa:
- SIMD (Single Instruction Multiple Data Stream) (Izvajanje procesor, ki opravljajo isto operacijo hkrati na večih točkah)
- MIMD (Multiple Instruction Data Stream) - vsak dela nekaj drugega

Model SIMD

En ukaz, več podatkovnih tokov
Računalnik izvrši hkrati natančno en ukaz na več podatkih
Ena krmilna enota, več ALE (vsaka ALE dela na svojih podatkih)
Vektorske operacije
Starejši superračunalniki

Model MIMD

Več ukaznih tokov, več podatkovnih tokov
Računalnik izvrši hkrati več ukazov nad različnimi podatki
Razmnoženi procesorji, vsak opravlja svoje delo
Komunikacija lahko upočasni delovanje

Navadi procesorji primerni
Skalabilnost: vedno lahko dodajamo nove procesorje

Velik izziv: paralelni algoritmi

- da se zagotovi izkoriščenost velikega števila procesorjev

---

3. Zapis podatkov in logični konstrukti

Zapis podatkov

Podatki se zapisujejo v bitih.

Dve pojavni obliki informacije:

- **Analogna** ali zvezna obsega neskočno število vrednosti
- **Digitalna** ali diskretna obsega končno št. vrednosti

Informatika - kako ravnamo z našimi informacijami. Kako iz množic informacij dobimo tiste informacije, ki jih potrebujemo.

Kako lahko elektronski stroj predstavi informacijo?
Ključne zahteve:
- jasnost (moramo biti prepričani kaj je to)
- nedvoumnost
- zanesljivost

Zunanje predstavitev - prirejena človeku:
- desetiška števila
- znaki na tipkovnici

Notranja predstavitev - prirejena računalniku:
- dvojiška števila
- dvojiška koda za znak

Dvojiški številski sistem

osnova: 2

Desetiški sistem:
10 števk: 0 - 9
Vsako mesto ustreza potenci št. 10
2018 = 2 10^3 + 0 10^2 + 1 10^1 + 8 10^0

Dvojiški sistem
2 števki: 0,1
Vsako mesto ustreza potenci št. 2
1101 = 1 2^3 + 1 2^2 + 0 2^1 + 1 2^0 = 13

Pretvarjanje iz **dvojiškega v desetiški sistem**:

- seštej potence števila 2, kjer je števka 1

Pretvarjanje iz **desetiškega v dvojiški**:

- število deli z dva dokler se da in si zapomni ostanke

Fiksna dolžina dvojiških števil - maksimalno število, ki je lahko še predstavljeno

- **aritmetični preliv:** je rezultat izračuna, ki presega pomnilniški prostor, namenjen za njegovo shranjevanje. Na primer, deljenje z ničlo daje veliko večji rezultat

Negativna binarna števila

Notacija predznak in vrednost prvi bit za predznak, ostali za vrednost

+5 = 0101, -5 = 1101
0 = 0000 and 1000 - dve ničli!

- precej nepraktično
- dvoumna predstavitev
- težje računanje

Dvojiški komplement

- za negativna števila obrni vsak bit in dodaj ena
- +5 = 0101, -5 = 1010 + 1 = 1011
- 0 = 0000, -0 = 1111 + 1 = 0000
  - precej bolj praktično
  - nedvoumna predstavitev
  - lažje računanje
  - seštevanje in odštevanje: se vrtimo po številskem krogu

Števila s plavajočo vejico

Znanstvena notacija z osnovo 10: 1.35×10^{-5}
osnova 2: $3.25_{25} = 11.02_{2} = 1.101 \times 2^{1}$

Mantisa in eksponent: ±M × 2±E

- pretvorimo število v znanstveno notacijo
- normaliziramo
- pomaknemo mantiso za binarno vejico in ustrezno povečamo eksponent

0.1 = 0.5
0.01 = 0.25
1.0 = 1

Primer:

- predznak in vrednost
- 16 bitov: mantisa: 1+9 bitov, eksponent 1+5 bitov
- 5,7510 = 101,112 = 101,11 × 20 = ,10111 × 23
- 0101110000000011

Znake lahko pišemo po ASCII tabeli.

Predstavitev zvoka
Zvok je analogni pojav
-amplituda
-perioda
-frekvenca

Pretvorba zvoka
Analogno-digitalna pretvorba:
- vzorčenje
- št. nivojev
- frekvenca vzorčenja

CD in Avdio formati(MP3, WAV, AAC,...)

Predstavitev slike
Vzorčenje slike
- shranjevanje barve
- slikovni elementi(piksli)

Ločljivost slike
- št. slikovnih elementov

Barvna globina
- št. barvnih nivojev

Rastrske slike
- 2D matrika števil

Barvne slike (RGB)

- red, green, blue
- barvne palete

Video: 25slik/sek

Ogromne količine podatkov ⇒ kompresija

Stiskanje podatkov

Nujna stiskanje (kompresija) podatkov

- brez izgubno
    - run-length encoding
    - kode s spremenljivo dolžino
- izgubno
    - izpusti se tisti del podatkov, ki ga človek slabše zazna
    - različne stopnje stiskanja
    - Kompromis med kvaliteto in velikostjo
    - JPG, MP3, MPEG,...

Shranjevanje binarnih podatkov

Desetiški sistem zahteva deset stabilnih stanj
Dvojiški sistem zahteva samo dve stabilni stanji

Zahteve za binarni računalnik:
- dve stabilni energijski stanji
- ločeni z veliko energijsko mejo
- možno preklapljati med stanji
- možno zaznati stanje naprave

Tranzistorji

Elektronsko preklapljanje med stanji
Narejen iz polprevodnikov
- majhen
- GB podatkov
- poceni izdelava

Fotografsko tiskanje
- integrirana vezja

- čipi

Logika in logični konstrukti

Boolova logika

Pravila za delo z dvema vrednostnima:
- True/False
- 1/0

Logično načrtovanje strojne opreme
- Boolovi izrazi se lahko pretvorijo v vezja

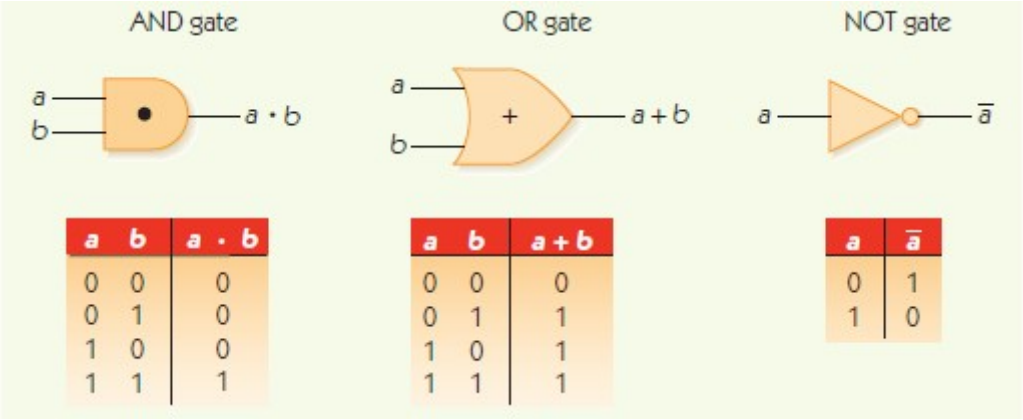
IN (produkt/konjukcija)
ALI (vsota/disjunkcija)
NE (negacija)

Tabeia pravilnosti
Primer: (a *b) + (a* ~b)

a	b	~b	(a • b)	(a • ~b)	(a • b) + (a • ~b)
true	true	false	true	false	true
true	false	true	false	true	true
false	true	false	false	false	false
false	false	true	false	false	false

Boolova vrata

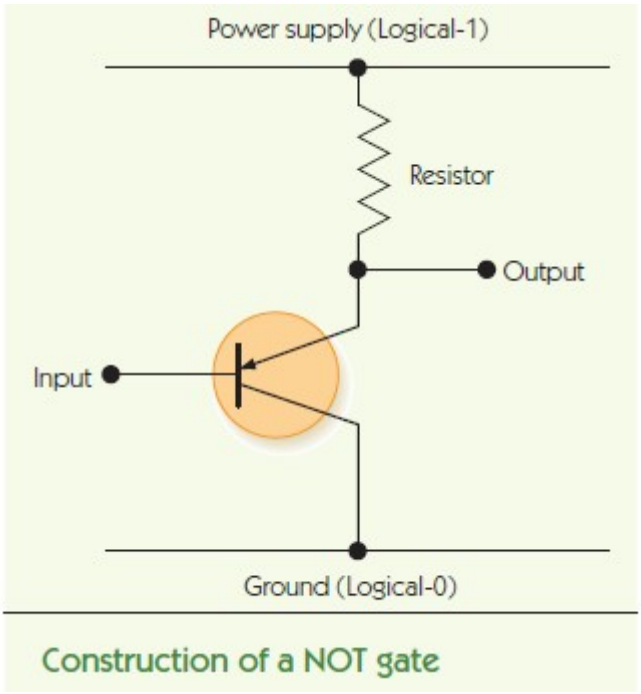
Vrata: elektronska naprava, ki na osnovi vhodnih vrednosti proizvede izhodne vrednosti
Vrata lahko ustrezajo Boolovim operatorjem:



Vrata NOT

0 → 1

1 → 0 (če gre not 1 pride do povezave med spodnjim in zgornjim delom in ven pride 0)

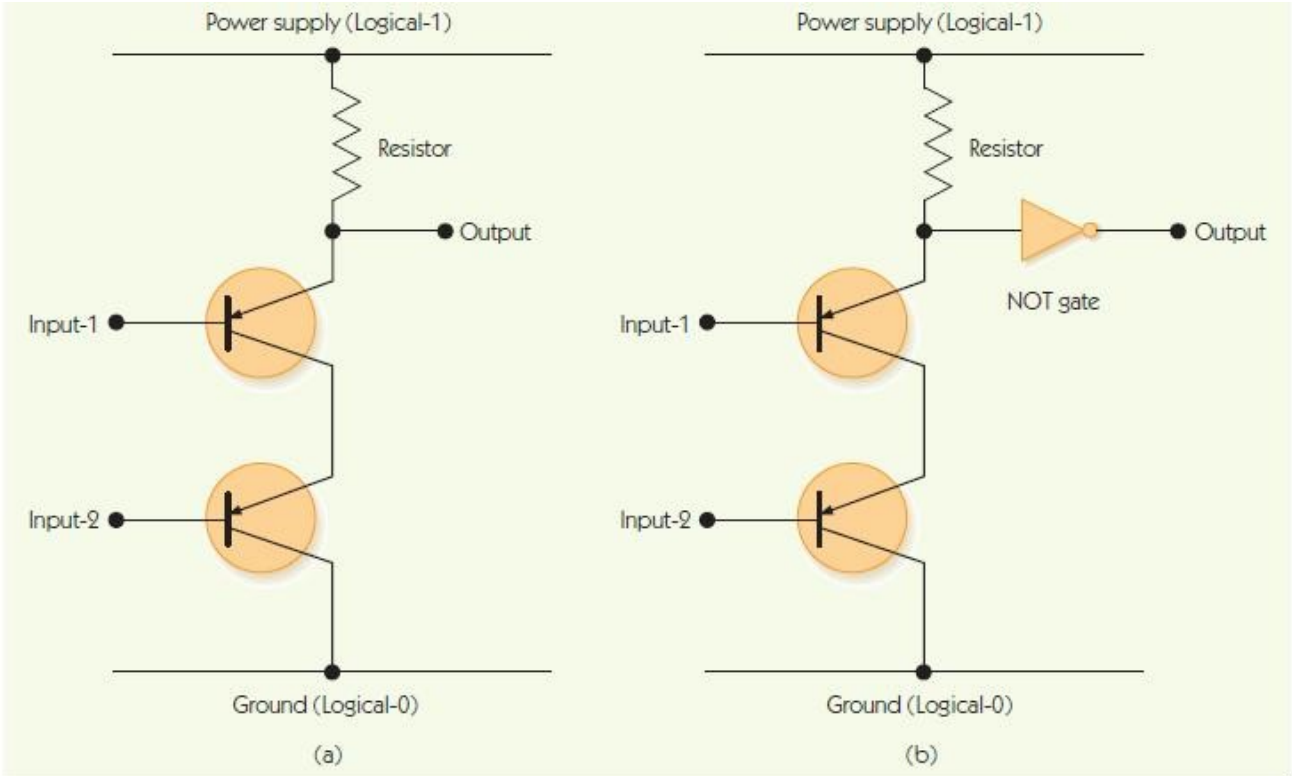


Construction of a NOT gate

Vrata AND

Vrata NAND: dva tranzistorja

Vrata AND: trije tranzistorji

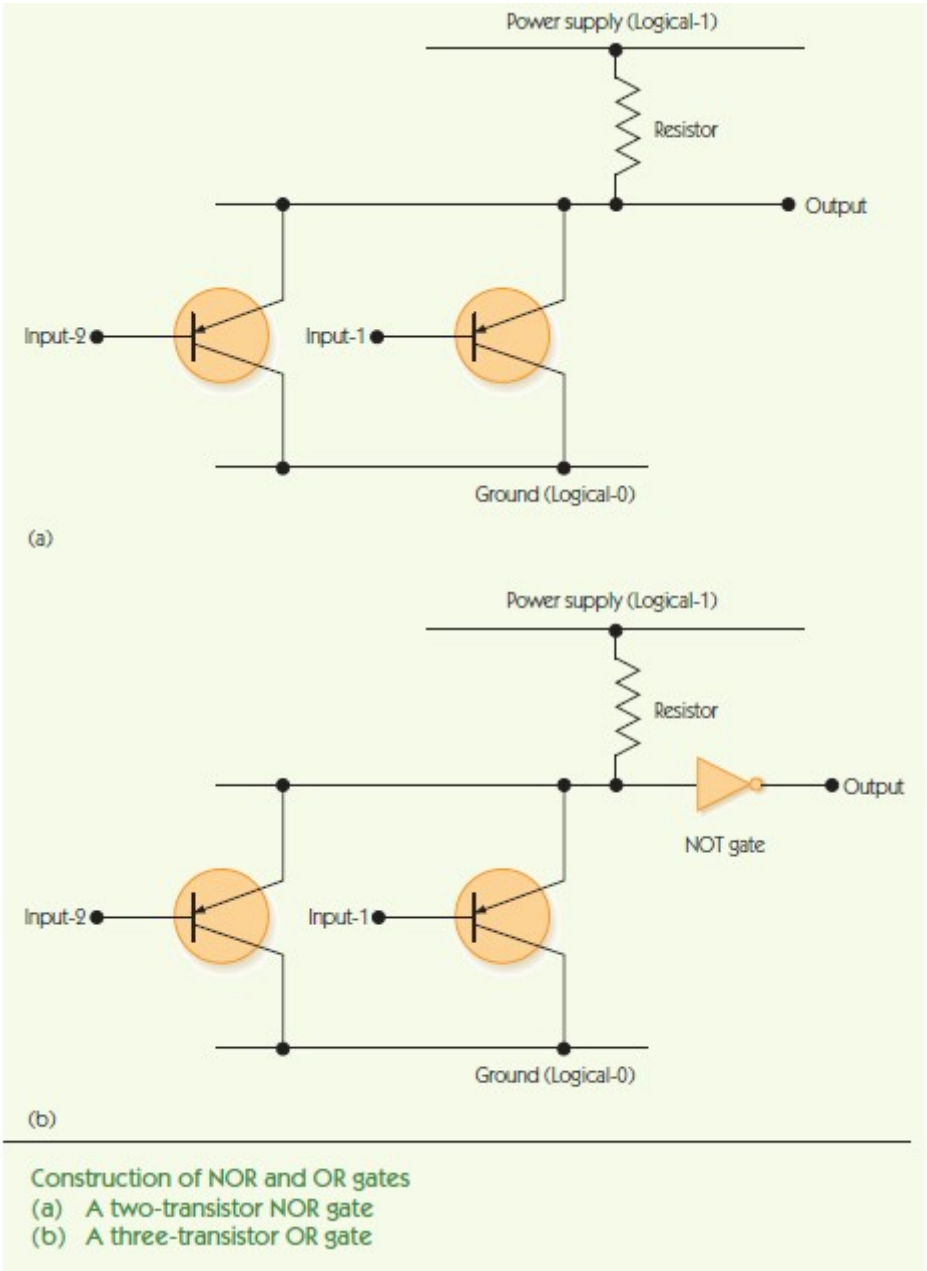


Construction of NAND and AND gates
(a) A two-transistor NAND gate
(b) A three-transistor AND gate

Vrata OR

Vrata NOT

- dva tranzistorja
  Vrata OR
- trije tranzistorji



Construction of NOR and OR gates
(a) A two-transistor NOR gate
(b) A three-transistor OR gate

Abstrakcija

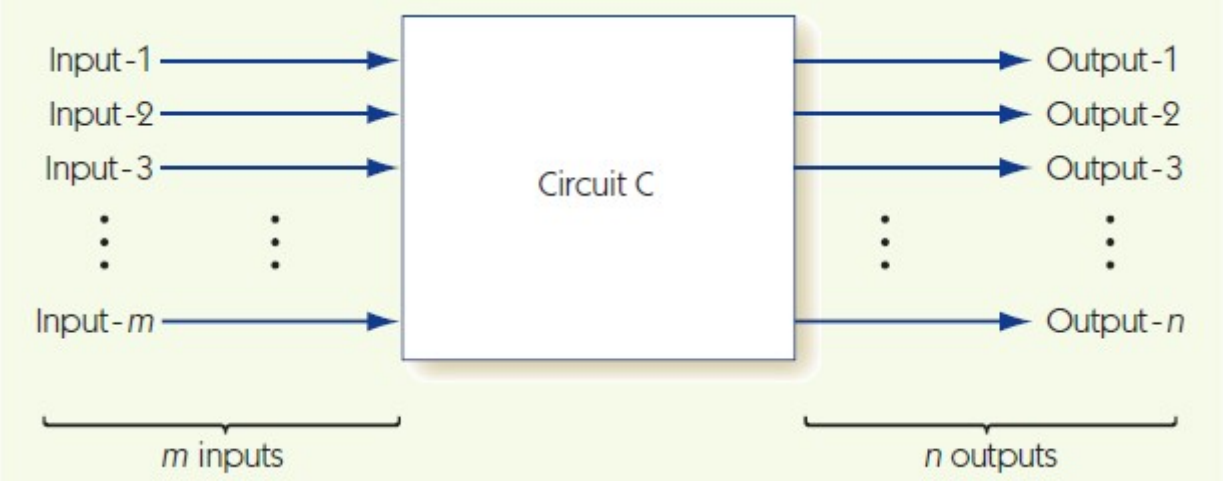
Pri načrtovanju vezij bomo abstrahirali elektrosnke podrobnsti

Gradnja računalniških vezij

Vezje:
- vhodne povezave
- vsebuje vrata povezana s povezavami
- ima izhodne povezave

Izhod je odvisen samo od trenutnega stanja (ni stanj - vezja nimajo stanja; določena vezja lahko vsebujejo povratne zanke).

Sekvenčna vezja vsebujejo tudi povratne zanke
- vrednost odvisna od prejšnjega vhoda



Algoritem za konstrukcijo vezij

Algoritem vsota produktov

1. Izdelaj tabelo pravilnosti

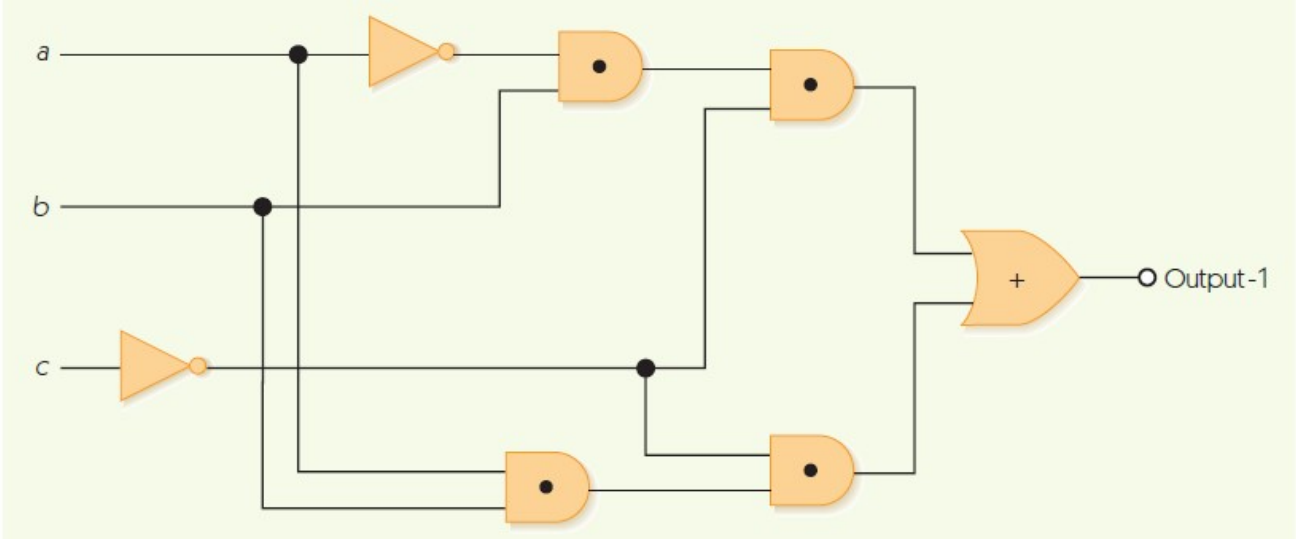
a	b	c	Output1	Output2
0	0	0	0	1
0	0	1	0	0
0	1	0	1	1
0	1	1	0	1
1	0	0	0	0
1	0	1	0	0
1	1	0	1	1
1	1	1	0	0

2. Izdelaj podizraze z vrati AND in NOT
   za vsako tako vrstico izdelaj podizraz:
   a = 0, b = 1, c = 0: (~a *b* ~c)
   a = 1, b = 1, c = 0: (a *b* ~c)

a	b	c	Output1	Output2
0	0	0	0	1
0	0	1	0	0
0	**1**	**0**	**1**	1
0	1	1	0	1
1	0	0	0	0
1	0	1	0	0
1	**1**	**0**	**1**	1
1	1	1	0	0

3. Poveži podizraze z vrati OR

(~a *b* ~c) + (a *b* ~c) ponovimo še za Output 2



4. Izdelaj diagram vezja

1. Construct the truth table describing the behavior of the desired circuit
2. While there is still an output column in the truth table, do Steps 3 through 6
3. Select an output column
4. Subexpression construction using AND and NOT gates
5. Subexpression combination using OR gates
6. Circuit diagram production
7. Done

The sum-of-products circuit construction algorithm

Seštevalnik

Vhod: dve nepredznačeni N-bitni dvojiški števili
Izhod: eno nepredznačeno N-bitno dvojiško število (vsota)

Začni z enobitnim seštevalnikom (1-ADD)
Razširi ga za N bitov

00101 + 01001 = 01110

Vezje za primerjanje:
vhod: dve števili in prenos
izhod: vsota in prenos

Kontrolna vezja

Sprejemajo odločitve, določajo vrstni red operacij, izbirajo vrednosti podatkov.

Izbirnik:
- Multiplekser (to kar piše spodaj)
- izbere enega izmed mnogih vhodov

Dekodirnik:
- Dekoder (to kar piše spodaj)
- pošlje na izhod en signal z vhodom

Izbirnik

Izbere enega izmed mnogo vhodov

- 2N vhodnih linij
- N izbirnih linij
- 1 izhodna linija

Vsaka vhodna linija ustreza vzorcu vrednosti na izbirnih linijah
Vrednost izbranega vhoda prenese na izhod

Uporaba izbirnika

Izbira vrednosti podatka iz množice, kot to določa vzorec na izbirnih linijah
Veliko podatkov pride v izbirnik, samo eden izbrani pride ven
Primer: izbira pravega registra

Dekodirnik

- Pošlje signal na en izhod izbran z vrednostjo na vhodnih linijah
- Vsaka izhodna linija ustreza vzorcu vrednosti na vhodnih linijah
- Izbrani izhod ima vrednost 1, vsi ostali izhodi pa 0

Primer:
dekodirnik 2-v-4

Uporaba dekodirnika

Izbira izhoda glede na vzorec na vhodu

Primer:

- Izberi en aritmetični ukaz, podan s kodo za ta ukaz
- Koda aktivira izhodno linijo, le-ta aktivira ustrezno aritmetično vezje
  št. rač. operacij izboljša optimizacijo kode.

Povzetek:

- Desetiška in dvojiška števila
- Negativna števila in dvojiški komplement
- Števila s plavajočo vejico
- Tranzistorji
- Boolova logika
- Kontrolna vezja

4. Turingov stroj

Formalni jezik

Formalni jezik je množica besed končne dolžine, ki so sestavljene iz simbolov neke izbrane končne množice simbolov (abeceda).

Primer: {a; b} je množica: {aaa, aab, aba, bbb,...}
vsebuje enako št. a in b-jev {ab, ba, abab, abba, baab, baba,...}

Ada Lovelace(Byron) - prva programerka

Gramatika

Je sistem prepisovalnih pravil, s katerimi iz enega niza simbolov izpeljemo drug niz simbolov.
Pravilom pravimo **produkcije**, vsaka produkcija pa ima L in D stran. Če v nekem nizu simbolov najdemo podniz, ki ustreza levi strani neke produkcije, lahko ta podniz nadomestimo z nizom simbolov na desni strani produkcije.

Če lahko uporabimo več produkcij, pač glede na trenuten niz simbolov in leve strani produkcijm potem lahko svobodno izberemo produkcijo in jo uporabimo.

Jezik Gramatike

Jezik gramatike sestavljajo vse besede, ki se jih da z ravnokar opisanim načinom uporabe produkcij izpeljati iz nekega vnaprej določenega simbola gramatike.

Primer: Vzemimo abecedo {a; b} in gramatiko s petimi produkcijami
P1: S → epsilon (prazno, nič)
P2: S → aAS
P3: aA → Aa
P4: Aa → bAb
P5: A → bAb

P2 → P1 → P3 → P4 →
S → aAS → aAaAS → aAaA → bAbA → bbbA → bbbb
bbbb - ja pripada zgornji gramatiki

Regularni izrazi

regex;
Hierhija formalnih jezikov Chomsky
turingov jezik → kontekstno odvisni jeziki → kontekstno neodvisni jeziki → regularni

Model računskega agenta

Model računskega agenta:

- ohranimo samo najbolj pomembne lastnosti oz. operacije
- "idealni" računski agenti

Model računskega agenta mora:

- brati vhodne podatke
- shranjevati in brati podatke iz pomnilnika
- izvajati ukaze v odvisnosti od trenutnega vhoda in trenutnega stanja
- proizvesti rezultat

Model

- zajame pomembne lastnosti pravega agenta oz. fenomen
- je običajno podan v različnem merilu
- opusti nekatere podronosti
- nima vseh funkcionalnosti

Napovedovanje z modeli

- z opazovanjem obnašanja modela napovemo obnašanje
- varneje, bolj poceni, lažje

- omogoča testiranje ne da bi zgradili pravega agenta (fenomena)

Matematični modeli računalnikov

Primer takih ra.: **kočni avtomat McCulloch in Pitts 1943** (regularni jeziki, ki jih sprejemajo končni avtomati; Dualnost: jezik predstavimo z vrsto avtomatov, ki ga sprejemajo ali z razredom sintaks); Model kontekstno neodvisnih jezikov, Chomsky

Skladovni računalnik, Turingov stroj

Matematično predznanje

Malo po obsegu, nezanemarljivo po globini.
Teorija množic: unija, presek, kartezično produkt, potečna množica, korespondence med množicami (relacije: injekcija, surjekcija, bijekcija)
OSnovni jezik mat. logike: predikatni račun prvega reda, reflesksivnost, simetrija, tranzitivnost
Množica naravnih števil: gotovo najpomembnejši mat. objekt
Množica besed nad abecedo = SIGMA

Končni avtomat

Jezik
Nedeterministični avtomat (pri eni lahko prideš do konca pri drugi pa ne)
Deterministični avtomat (ima konec)

Hierarhija jezikov in strojev:
0. Turingovi jeziki: gramtike brez omejitev, turingovi stroji

1. Kontekstno odvisni jeziki: kontekstno odvisne gramatike, liearno omejeni avtomati
2. Kontekstno neodvisni jeziki: kontekstno neodvisne gramatike, skladovni avtomati
3. Regularni jeziki: linearne gramatike, končni avtomati

Turingov stroj

Lastnosti Turingovega stroja:

- Ima trak, neskončen v obeh smereh (vsako polje na traku hrani 1 simbol; abeceda končna množica simbolov, ki se berejo in zapisujejo na trak; na traku so zapisani vhodni podatki; trak služi kot pomnlinik)
- Vedno je v enem od kočno mnogih stanj (1 do k)
- Ukazi za Turingov stroj(v odvisnosti od trenutnega stanja in vhoda, zapiši nov simbol, spremeni stanje, ter se pomakni za eno polje v levo ali desno)

Izvajanje ukazov:

- if (v stanju i) and (prebere simbol j) then
- zapiši simbol k na trak (v trenutno polje)
- spremeni stanje v stanje s (ki lahko ostane enako)
- premakni se v smer d (za eno polje)
- pravilo: (i, j, k, s, d)

Primer

- pravilo: (1, 0, 1, 2, right)
- if (stanje=1) and (vhod=0) then
- zapiši 1
- pojdi v stanje 2
- premakni se desno

Turingov stroj kot računski model

- Množica pravil: program Turingovega stroja
- Prvo stanje je vedno 1
- Stroj vedno začne brati najbolj levo neprazno polje
- Ne smeta obstajati dve pravili z istim stanjem in vhodnim simbolom
  - Se izogiba dvoumnostim
- Če ne obstaja nobeno pravilo za trenutno stanje in vhodni simbol, se stroj ustavi
- Turingov stroj je torej primeren model računskega agenta:
  - bere vhodne podatke s traku
  - trak uporablja kot pomnilnik s katerega bere in na katerega zapisuje podatke
  - izvaja akcije v odvisnosti od trenutnega stanja in vhoda
  - proizvede rezultat in ga zapiše na trak

Model algoritma mora biti:

- popolnoma urejeno zaporedje,
  začetno stanje in začetna pozicija traku sta definirani
  največ eno pravilo se lahko hkrati sproži
  definirano je kaj se zgodi, če ni nobenega ustreznega pravila
  Turingov stroj ve kje začeti in kaj kdaj storiti
- nedvoumnih in učinkovito izračunljivih operacij,
  ukazi Turingovega stroja popolnoma specificirajo kaj naj stroj naredi
  ukazi ne morejo biti dvoumni
- proizvede rezultat,
  Turingov stroj gre lahko v mrtvo zanko (kot slabi algoritmi)
  lahko zagotovimo, da se ustavi pri pravilnem reševanju pravega problema

- se ustavi v končnem času
  izhod je zapisan na traku, ko se Turingov stroj ustavi

Turingov stroj je torej dober model algoritma!

Naloga:
na koncu niza 0 in 1 postavi paritetni bit
število vseh 1 mora biti liho

Rešitev:
Stanje 1 predstavlja sodo pariteto do danega trenutka
Stanje 2 predstavlja liho pariteto do danega trenutka
Vhodni simbol 0 ne spremeni stanja
Vhodni simbol 1 spremeni stanje 0 v 1 ter 1 v 0
Ko stroj naleti na vhodni simbol b, naj zapiše partitetni bit in gre v stanje 3 za katerega ni nobenega pravila ⇒ se bo zaustavil

```
![[Pasted image 20220114121613.png]]
```

...	b	**1 (1)**	0	0	1	b	b	...
...	b	1	**0 (2)**	0	1	b	b	...
...	b	1	0	**0 (2)**	1	b	b	...
...	b	1	0	0	**1 (2)**	b	b	...
...	b	1	0	0	1	**b (1)**	b	...
...	b	1	0	0	1	1	**b (3)**	...

Nerešljivi problemi

- Turingov stroj definira meje izračunljivosti
- Če problem ne more biti rešen s Turingovim strojem, ne more biti rešen niti z algoritmom
  - je neizračunljiv oz. nerešljiv

Obstajajo neizračunljivi problemi!

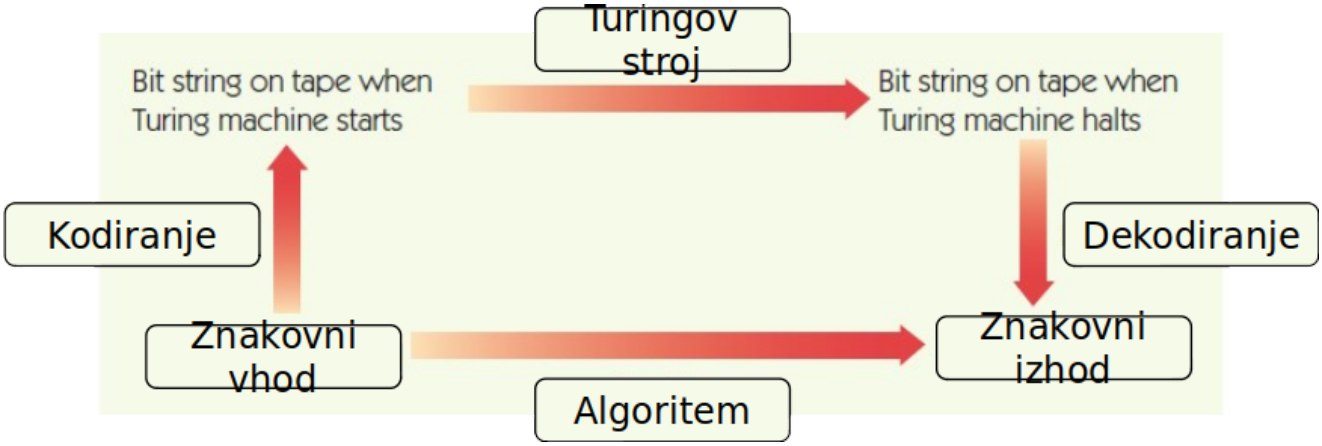
- Dokaz: Problem ustavljanja
- Ali obstaja Turingov stroj, ki na vhodu prejme program Turingovega stroja ter zanj ugotovi ali se bo ustavil ali ne?
- Dokaz s protislovjem: predpostavimo, da takšen Turingov stroj obstaja ⇒ nas privede v protislovje ⇒ takšen Turingov stroj ne obstaja ⇒ ta problem je nerešljiv ⇒ ne obstaja niti algoritem, s katerim bi ga lahko rešili!

Problem ustavljanja

Obstajajo problemi za katere ne obstaja algoritmična rešitev
Lahko dokažemo, da ne more obstajati algoritem, ki bi lahko rešil dan problem
V luči kvantnih računalnikov lahko reduciramo čas računanja
“Halting problem” – problem ustavljanja.

Church-Turingova teza

Teza ni dokazana, najbrž niti ni dokazljiva, skoraj gotovo pa drži. Nihče še ni nikoli na nobenem problemu dokazal nasprotno. Za vse druge računske modele so pokazali, da so ekvivalentni Turingovemu stroju.



5. Jeziki in prevajalniki

Hierhija po Chomskemu

Korespondenca: jeziki - avtomati - izračuni

Primeri gramatik
Velike črke - neterminali, majhne črke - terminali, začetni simbol S

Primer: {a, b}, {S, A, B}, produkcijska pravila

P1: S → AB
P2: S → ε (where ε is the empty string)
P3: A → aS
P4: B → b

Definira jezik: anbn (n kopij a, potem n kopij b)

Enostavnejša gramatika, isti jezik:
Terminali {a, b}, neterminali {S}, začetni simbol S, produkcijska pravila

P1: S → aSb
P2: S → ε

Opisovanje strukture jezika

Ideja opisovanja strukture jezika je delo Pāṇinija (cca. 6-4. stoletje pred novim štetjem)
Notacija za opis Sanskrita
Ekvivalentna Backusovi gramatiki
Backus-Naurova Forma (BNF) je notacija za kontekstno-neodvisne gramatike po Chomskemu
Prva aplikacija: metajezik za opisovanje programskega jezika ALGOL

BNF

```
<expr> ::= <term>|<expr><addop><term> <integer> ::= <digit>|<integer><digit>
```

Prevajalniki

Uvod

- CPE izvajajo samo strojni jezik
  vsak program se mora prevesti v strojni jezik
- Prevajanje iz zbirnega v strojni jezik je trivialno
  prevajanje 1:1
- Prevajanje iz visoko-nivojskih jezikov je veliko bolj kompleksno
  prevajanje 1:M
- Prevajalniki prevajajo programe iz visoko-nivojskih jezikov

Dve glavni zahtevi za prevajalnike:

- pravilnost: strojni ukazi morajo narediti natančno to kar pomenijo visoko-nivojski ukazi
- učinkovitost in jedrnatost: strojna koda mora biti optimizirana in se mora izvajati hitro

Proces prevarjanja

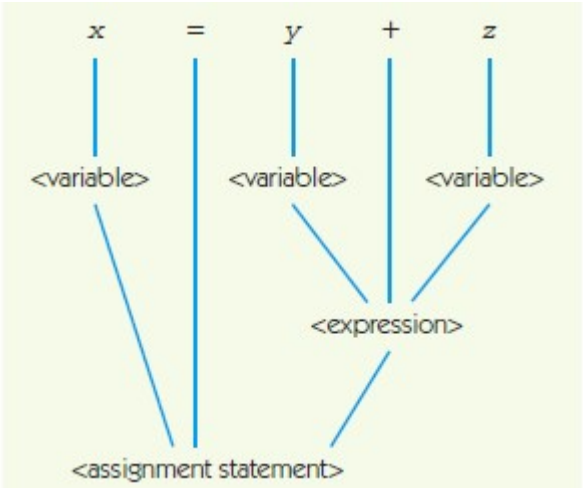
1. Leksikalna analiza

- združevanje znakov v lekseme
- izpusti nepomembne znake (presledke, komentarje, ...)
- določi tip posameznega leksema (simbol, število, oklepaj itn.)
- postopek:
  - izpusti nepomembne znake in poišči začetek leksema
  - združuj znake dokler ne zaznaš konec leksema

- primer klasifikacije
  običajno je št. leksemov precej večje (>50)

2. Sintaksna analiza

- preverjanje sintakse in gradnja notranje predstavitve programa
- ugotovi gramatično strukturo
- zgradi sintaksno drevo
- definirana je z množico pravil (produkcij)
- BNF (Backus-Naur Form) notacija za opis pravil
- levaStran: gramatična kategorija
- desnaStran: vzorec, ki zajema strukturo kategorije (kočni simbol se ne členi naprej)
- Zaporedje leksemov je sintaksno pravilno, če jih lahko sintaksni analizator z zaporedno aplikacijo pravil pretvori v začetni simbol



3. Semantična analiza in generiranje kode

- analiza pomena in generiranje strojnih ukazov
- preverja, da je pomen pravi (npr. ali so vse spremenljivke deklarirane)
- Semantični zapisi
  hrani informacije o vmesnih simbolih (ime, podatkovni tip,...)

V prvem prehodu čez kodo

- se pregleduje, če so vse veje semantično veljavne
- gleda se semantične zapise
- primerja se podatkovne tipe,...

Primer:
x=y+z

4. Optimizacija kode

- izboljševanje časovne in prostorske učinkovitosti kode

- Lokalna optimizacija
  preverja majhne dele kode v zbirniku (<5 ukazov)

1. Evaluacija konstant
   če se da, v naprej izračuna rezultate aritmetičnih operacij
2. Redukcija po moči
   počasne aritmetične operacije so zamenjane s hitrejšimi
3. Odstranjevanje nepotrebnih operacij
   odstranjevanje sicer pravilnih a nepotrebnih operacij

6. Programski jeziki in algoritmi

Jeziki

- Zbirni jezik (asembler)
  Ukazi se preslikajo neposredno v strojne ukaze eden v drugega
  Simbolične kode ukazov (ne binarne)
  Simbolični naslovi za ukaze in podatke
  Pseudo ukazi z uporabniku prijaznejšimi servisi (npr. generiranje podatkov)
- Nizko-nivojski programski jeziki: strojni jezik, zbirni jezik
- Visoko-nivojski programski jeziki: Java, C, Python
- Jeziki za opisovanje pomena (W3C), npr. XML (eXtensible Markup Language)

Proceduralni jezik

Proceduralno programiranje
Zaporedje ukazov

FORTRAN

- FORmula TRANslation
- prvi implementirani višje-programski jezik
- osredotočen na numerično izračunavanje, zahtevne izračune
- GO TO stavek (kot JUMP v zbirnem jeziku)
- zunanje knjižnice: preverjena namenska koda
- zelo uporabljan jezik

COBOL

COmmon Business-Oriented Language

- Poudarek je na poslovnih aplikacijah
- obdelovanje podatkov na datotekah

- Zelo dolgo zelo uporabljan programski jezik
- Ogromno programske kode je še delujoče

C, C++, Java, C, Python#

C: neposredno delo s pomnilnikomnaslovi pomnilniških lokacij, kazalci, zelo pogosto uporabljan za sistemsko programiranje
C++: objektno-orientirano programiranje, standardizacija, zbirke knjižnic programske kode
Java: (začetno) močna integracija s spletnimi brskalniki, objektno-orientiran jezik, Poudarek na prenosljivosti
C#: Podobna oblika in cilji kot pri Javi, Integriran v ogrodje .NET, Podoben C++
Python: Razvit za naloge sistemske administracije in spletne vmesnike, Interpretiranje:, izvorna koda se interpretira med izvajanjem

Namenski programski jezik

Nekateri jeziki so razviti za točno določene naloge
te naloge rešujejo zelo učinkovito
ne morejo rešiti drugih nalog
Primeri: SQL, HTML, XML, OWL, JavaScript

SQL: poizvedovalni jezik, delo po pb
HTML in JS: oblikovanje spletne strani, značke, js se vgradi v html, skriptni jezik

Funkcijsko programiranje

LISP kot primer
Osnovni gradniki programa so (matematične) funkcije

- Brez eksplicitnih detajlov procedure izvajanja
- Poudarek na transformaciji podatkov

Logično programiranje

Program opisuje dejstva in pravila; KAJ želimo in ne detajlno proceduro KAKO to dobiti. Poizvedbe uporabljajo logično dedukcijo. Deklarativni jeziki. Pogosti v AI.

Prolog: Dejstva opisujejo relacije med specifičnimi objekti, Pravila opisujejo splošne relacije med spremenljivkam, rekurzija, Programer programira tako, da zgradi bazo znanja.

Pararelno programiranje

Programski jeziki, ki omogočajo paralelno izvajanje programov na več procesorjih (SIMD, MIMD).

7. Jeziki, Algoritmi, računalništvo

Algoritmi

"Algoritem je popolnoma urejeno zaporedje nedvoumnih in učinkovito izračunljivih operacij, ki, ko se le-te izvedejo, proizvede rezultat in se ustavi v končnem času."

Algoritem je procedura za reševanje (matematičnih) problemov v končnem številu korakov.

Urejeno zaporedje ukazov, ki zagotovljeno reši dani problem.

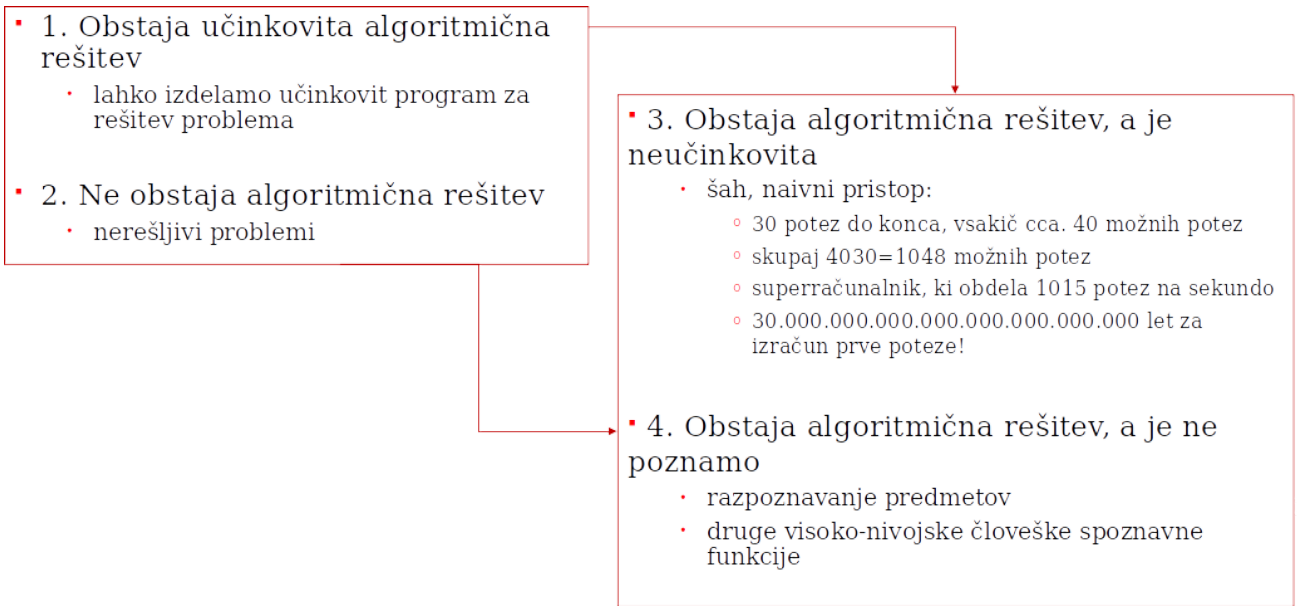
Tri kategorije operacij:

- zaporedne operacije
- pogojne operacije
- iterativne operacije (skrajšaš pisave nekaterih programov, ponoviš več operacij glede na neki pogoj)

Algoritmično reševanje problemov

Če lahko zapišemo rešitev, ki reši dani problem. Lahko avtomatiziramo rešitev problema.

- Kreiranje pravilnih in učinkovitih algoritmov
- Preučevanje njihovih lastnosti
- Načrtovanje programskih jezikov, v katerih so algoritmi zapisani
- Načrtovanje in gradnja računalniških sistemov, ki jih izvajajo



Algoritmični način razmišljanja
Naučiti se analizirati problem, poiskati proceduro za njegovo rešitev
Programiranje računskih agentov
Industrijska revolucija – stroji prevzamejo fizično delo
Računalniška revolucija – stroji prevzamejo (rutinske) ponovljive mentalne naloge
Človek – odličen računski agent za visoko-nivojske probleme (načrtovanje, sklepanje, razumevanje, učenje, ...)

Računalništvo

Kaj je računalništvo?

Vloga računalnikov v računalništvu je podobna kot vloga teleskopov v astronomiji, mikroskopov v biologiji in epruvet v kemiji.

- Zgodnje delo na področju računalništva: leta 1920-1940
- Teoretično računalništvo brez računalnikov
- Formalni modeli

Računalnik je stroj

Programiranje je zelo pomembno, a: računalniški program je tudi zgolj orodje

Ne samo program:

- metode, ki so implementirane
- storitve, ki jih nudi
- rezultati, ki jih proizvaja

Program je sredstvo za dosego cilja

Računalništvo je veda o načrtovanju in
razvoju algoritmov za reševanje
množice pomembnih problemov.

Računálništvo je znanstvena veda o delovanju računalnikov in o njihovi uporabi, kar vključuje strojno in programsko opremo. V praksi je računalništvo povezano z mnogimi drugimi vedami, od abstraktne analize algoritmov do bolj stvarnih tem, kot so programski jeziki, programska in strojna oprema. Kot znanstvena veda se računalništvo loči od matematike, programiranja, programskega inženirstva in računskega inženirstva, čeprav se ta področja pogosto zamenjujejo. Računalniško pismeni so ljudje, ki znajo na računalniku narediti in opraviti dejanja, ki jih morajo. Informacijsko pismeni so ljudje, ki presodijo kdaj je informacija potrebna, kje jo potrebujejo, kje jo bodo dobili, na kakšen način jo bodo predali naprej.

Preučevanje obnašanje algoritmov, ali so pravilni in učinkoviti
formalne in matematične lastnosti algoritmov

Načrtovanje in gradnja računalniških sistemov, ki lahko izvajajo algoritme
strojna realizacija
Načrtovanje programskih jezikov in prevajanje algoritmov v te jezike
lingvistična realizacija
Identifikacija pomembnih problemov in načrtovanje pravilnih in učinkovitih programskih paketov za reševanje teh problemov
aplikacije algoritmov

Pseudokoda in načrtovanje algoritmov

Kako predstaviti algoritem?
Naravni jezik

- zelo ekspresiven, bogat, enostaven za uporabo, pogosto uporabljan
- redundanten, nestrukturiran, dvoumen, kontekstno odvisen
  Programski jezik
- strukturiran, načrtovan za računalnike, neredundanten
- gramatično tečen, preveč precizen, kriptičen, veliko tehničnih podrobnosti
  Pseudokoda je nekje vmes
- „programski jezik brez podrobnosti“
- enostavna, berljiva, strukturirana, a brez strogih pravil
- enostavna za vizualizacijo
- enostavna za prevod v programske jezike

Zaporedne operacije

Tri osnovne zaporedne operacije:

- računske: posamezen numerični izračun
- vhod: dobi podatke v algoritem
- izhod: posreduje podatke izven algoritma

Spremenljivke: poimenovana lokacija za hranjenje vrednosti

Zaporedni algoritem

je sestavljen samo iz zaporednih operacij
„premočrtni“ algoritem

Nazorne operacije

Operacije za nadzor poteka izvajanja algoritma
Pogojni stavki (vejitve)
Iterativni stavki (iteracije)

Pogojni stavek

Če-potem-sicer stavki (if-then-else)
Kaj, če...

1. Oceni pogoj
2. Če pogoj drži, se izvedejo stavki1
3. Če pogoj ne drži, se izvedejo stavki2

Iteracije

While stavek (delaj dokler)
Blok operacij (telo zanke) se iterativno izvaja v zanki dokler je izpolnjen pogoj za nadaljevanje
Testiranje pogoja na začetku zanke (pretest loop):

Testiranje pogoja na koncu zanke (posttest loop):

Vedno moramo poskrbeti za izhodni pogoj
Izogibati se neskončni zanki

Primitivne operacije

Računske
Vhod/izhod
Pogojni stavek
Iteracije
Z njimi lahko predstavimo katerikoli algoritem

8. Programsko inženirstvo

Zbirni jezik

Zbirni jezik je bolj razumljiv in lažji za uporabo kot strojni jezik
Ukazi se preslikajo neposredno v strojne ukaze (eden v enega)
Simbolične kode ukazov
Simbolični naslovi za ukaze in podatke
Pseudo ukazi

Pomanjkljivosti zbirnega jezika
Programer mora skrbeti za premikanje podatkov med pomnilnikom in registri
Mikroskopski pogled na naloge
Strojno odvisni jeziki (neprenosljiva koda)

Visoko-nivojski

Visoko-nivojski programski jeziki so bolj razumljivi in lažji za uporabo kot zbirni jezik.

Pričakovanja:
Programerju ni potrebno skrbeti za pomikanje podatkov med pomnilnikom in registri
Makroskopski pogled na naloge
Programi prenosljivi med različnimi računalniki
Programski konstrukti bližje naravnemu jeziku in matematični notaciji

- Prevajalnik (compiler) prevede izvorno kodo v zbirni jezik (oz. v podobno nizkonivojsko obliko)
- Zbirnik oz. podoben konverter jo pretvori v objektno kodo
- Programske knjižnice vsebujejo uporabno preverjeno kodo
- Povezovalnik (linker) združi več datotek objektne kode v en izvršljivi modul

Proceduralni jeziki

Programi so zaporedja ukazov

Primeri: C++, Java, Phyton

Podobna filozofija/model
Razlike v
sintaksi: kako so programski stavki napisani
semantiki: pomen stavkov
Grob pregled nad razlikami med tremi programskimi jeziki

Analiza značilnosti

Sintaksa

- opisovanje podatkov ali spremenljivk
- povezovanje konstruktov, zanke in pogojne vejitve

Semantika

- pomen funkcijskih klicev
- pomen operatorjev

Globlje strukture

- moduli
- razredi
- bseg (scope)

Programsko inženirstvo

Programsko inženirstvo:

- veliki projekti razvoja programske opreme
- zahtevajo sodelovanje, upravljanje, organizacijo
- formalni procesi razvoja

Razvoj programske opreme

Pred implementacijo: oceni stroške, specifikacija problema, načrt programa, izbor ali razvoj ter analiza algoritmov
Implementacija: Kodiranje, razhroščevanje
Po implementaciji: testiranje, verifikacija, primerjanje, dokumentacija in vzdrževanje

Kaskadni model → Agilni razvoj programske opreme → Modelna razvojna okolja (IDE)

9. Zbirni jezik

Gol stroj

- na nivoju strojne opreme
- pisanje binarnih ukazov
- pisanje podatkov z dvojiškimi števili/kodami
- nalaganje ukazov v pomnilnik

Zelo neprijazno (nemogoče) za človeka. Potrebno je zgraditi vmesnik med strojem in računalnikom

Sistemska programska oprema

- Vmesnik med človekom in strojno opremo
- Navidezni stroj: množica storitev in virov, kot jih človeku predstavi sistemska programska oprema

Naloge sistemske programske opreme:

- skriti kompleksne in nepomembne detajle notranje strukture Von Neumannove arhitekture
- predstaviti uporabniku pomembne informacije na razumljiv način
- dovoliti uporabniku dostop do virov na enostaven in učinkovit način
- agotoviti varno okolje za delovanje računalnika

Navidezni stroj

Gol stroj:

1. Napiši program v dvojiškem sistemu
2. Naloži ukaze enega za drugim v pomnilnik
3. Vstavi začetni ukaz na pomnilniško lokacijo 0 in zaženi delovanje
4. Preberi rezultate enega za drugim v dvojiškem sistemu

Navidezni stroj:

1. Napiši program v visoko-nivojskem programskem jeziku z urejevalnikom besedil
2. Shrani program v mapo
3. Prevedi program v dvojiški strojni jezik
4. Uporabi razporejevalnik za nalaganje in zagon programa
5. Uporabi V/I sistem za izpis rezultatov

Programski jeziki

Zbirni jezik (asembler)

- Ukazi se preslikajo neposredno v strojne ukaze eden v drugega)
- Simbolične kode ukazov (ne binarne)
- Simbolični naslovi za ukaze in podatke
- Pseudo ukazi z uporabniku prijaznejšimi servisi (npr. generiranje podatkov)

Nizko-nivojski programski jeziki: strojni jezik, zbirni jezik
Visoko-nivojski programski jeziki: Java, C, Python

Zbirnik

- Proces uporabe izbirnega jezika
- Izvorni program v zbirnem jeziku.
- Zbirnik ga prevede v objektni program v strojnem jeziku.
- Nalagalnik (loader) ga naloži v pomnilnik.
- Strojna oprema ga izvede.

Format zbirnega jezika
Oznaka: opcijsko simbolično ime za pomnilniško lokacijo
Simbolično ime (okrajšava, mnemonik) za strojni ukaz
Naslov pomnilniške lokacije
-- komentar
Veliko bolj razumljivo kot binarni ukazi!

NEXTSTEP: LOAD X --Put X into reg. R

Prednosti zbirnega jezika

- Simbolična imena za ukaze
- Simbolična imena za naslove pomnilniških lokacij

- Pseudo ukazi:
    - ukazi, ki niso neposredno prevedeni v strojne ukaze:
    - .BEGIN in .END označujeta začetek in konec programa za prevajanje – za generiranje programa
    - .DATA pretvori podatke v desetiškem sistemu v dvojiški sistem in jih shrani na navedeni lokaciji – generiranje podatkov

Prednosti pred strojnim jezikom:

- jasnost
- berljivost
- vzdrževalnost

Tipična struktura programa v zbirnem jeziku:

- Predel za program
- Predel za podatke

Prevajanje in nalaganje

Zbirnik prevede program v zbirnem jeziku v strojni jezik:

- pretvori simbolične kode ukazov v strojne ekvivalente
- pretvori simbolične oznake v naslove pomnilniških lokacij
- izvede pseudo ukaze
- zapiše objektno (izvršljivo) datoteko s strojnimi ukazi

Nalagalnik pripravi program na izvajanje

- naloži ukaze v pomnilnik
- sproži strojno opremo, da začne izvajati program

Tabela simbolov

Pretvorba simbolnih oznak v naslove pomnilniških lokacij
Zbirnik potrebuje dva koraka:

- Zgradi tabelo simbolov
- Simbole nadomesti z binarnimi vrednostmi iz tabele simbolov

1. Prvi korak: Gradnja tabele simbolov

2. Drugi korak: Poišče in zamenja simbolične kode ukazov z binarnimi. Zamenja simbolne oznake z naslovi pomnilniških lokacij iz tabele simbolov. Izvede pseudo ukaze .DATA in na ustrezne naslove zapiše ustrezne binarne vrednosti. Zapiše ukaze v objektno (izvršljivo) datoteko.

Operacijski sistemi

Operacijski sistemi

Tipični ukazi operacijskega sistema:

- prevedi program
- poženi program
- shrani informacijo v datoteko
- poišči predhodno shranjeno datoteko
- izpiši podatke o datotekah
- vzpostavi mrežno povezavo
- sporoči trenuten čas in datum

Glavne naloge operacijskih sistemov

- Uporabniški vmesnik
- Dostop do sistema in datotek
- Varnost in zaščita sistema
- Učinkovito razporejanje virov
- Varna uporaba virov
- premikanje programov v/iz pomnilnika
- Procesiranje V/I,...

Uporabniški vmesnik

Uporabnik preko uporabniškega vmesnika komunicira z operacijskim sistemom (in računalnikom). Operacijski sistem sprejema ukaze in jih razporeja naprej

Varnost in zaščita sistema

Operacijski sistem dovoli samo avtorizirane posege oz. dostope do virov

Dostop običajno dovoljen samo z uporabniškim imenom in geslom

- individualna gesla
- superuporabniki (superusers) imajo več pooblastil
- podatki o geslih so običajno šifrirani

Datoteke in direktoriji imajo sezname dovoljenih posegov:

- beri datoteko (R)
- dodaj novo informacijo v datoteko (A)
- spremeni trenutno informacijo (C)
- izbriši datoteko (D)

Učinkovito razporejanje virov

- Operacijski sistem učinkovito razporeja vire programov, da so čim bolje izkoriščeni
  CPE naj bo čim bolj uporabljena.
- Več programov je lahko hkrati aktivnih

Vsak program je v enem od treh stanj:

- Se izvaja: program trenutno uporablja CPE
- V pripravljenosti: program je pripravljen, da se izvede v CPE
- Čaka: program čaka, da se zaključijo V/I operacije

Operacijski sistem vzdržuje vrsto programov v pripravljenosti in ustrezno porazdeli vire

Varna uporaba virov

- Zagotavljanje, da se računalnik ne ujame v mrtvo zanko
- Hkratno izvajanje več programov in dostopanje do virov
- Mrtva zanka nastopi, ko več programov zaseda nekaj virov, hkrati pa čaka na vire, ki jih zasedajo drugi programi
- Izogibanje mrtvim zankam, če ne moreš dobiti vseh zahtevanih virov, osvobodi vse vire, ki jih zasedaš in poizkusi ponovno pozneje
- Reševanje mrtvih zank, če se ne zgodi pričakovani dogodek, povprašaj ponovno

Zgodovinski pregled

- Prva Generacija (45 -55) - skoraj gol stroj, skromni OS, veliko neporabljenih virov
- Druga Generacija (55 - 65) - paketni operacijski sistem, operator, jezik za nadzor pravil
- Tretja Generacija (65 - 75) - večopravilni OS, več programov, preklaplanje med programi, varnost računalnikov, delitev procesorskega časa
- Četrta Generacija (85 - danes) - omrežni operacijski sistem, podpira vse lokalne storitve, grafični uporabniški vmesnik, podpira storitve v omrežju

Razširjen navidezen stroj: Namen vzpostavitve navideznega stroja je skrivanje lastnosti določenega sistema pred uporabniškimi programi

Računalništvo v oblaku: je slog računalništva, pri katerem so dinamično razširljiva in pogosto virtualizirana računalniška sredstva na voljo kot storitev preko interneta. Zanj je značilna avtomatizacija, samopostrežnost in elastičnost zagotavljana virov. Računalništvo v oblaku je povezano s koncepti storitveno usmerjene arhitekture (SOA), mrežnega računalništva in virtualizacijo. V praksi se računalništvo v oblaku najpogosteje uporablja v obliki spletne pošte] in družbenih omrežjih.

OS: Microsof, Linux, Mac X, Mac OS, Android, iOS,...

Virtualizacija

Navidezni stroj, poganja en operacijski sistem, v tem OS emuliramo druge OS. Lahko poganjamo več navideznih strojev.

- Peta generacija: multimedijski uporabniški vmesnik (slike, govor, video, zvok), vzporedni procesorski sitemi (enostavno vzporedno izvajanje programov), porazdeljeno računsko okolje (porazdeljeno izvajanje programov, računalništvo v oblaku).

10. Algoritmi in kompleksnost

Uvod

Za isti problem obstaja več rešitev.

Ocenjevanje algoritma:

- enostavnost razumevanja
- eleganca
- časovna in prostorska učinkovitost

Atributi algoritmov

Najpomembnejši atributi:

- pravilnost delovanja
- razumljivost
- eleganca
- učinkovitost

Atribut = pridružiti, pripisati, odkazati) pomeni dodatek, pripis, pridevek in se uporablja z raznimi bolj specifičnimi pomeni.

Pravilnost algoritmov

Najpomembnejša lastnost: algoritem mora biti pravilen

Razumljivost in eleganca

Razumljivost, jasnost algoritmov
Enostavnost za vzdrževanje in nadgrajevanje
Lažje preverjati pravilnost
Algoritmi se bodo uporabljali za druge namene
Algoritme bodo uporabljali drugi

Eleganca, lepe rešitve (vsota števil od 1 do 100)

Včasih sta si razumljivost in elengaca nasprotujoči včasih pa ne.

Učinkovist algoritmov

Učinkovitost algoritma = učinkovita izraba virov

Časovna učinkovitost/potratnost
prostorska učinkovitost/potratnost

Prostorka učinkovitost

Količina informacije, ki jo mora algoritem dodatno shraniti

Časovna učinkovitost

Merjenje časa izvajanja algoritma:
- s katerim računalnikom?
- na katerih podatkih?
bolj sistematična analiza časovne učinkovitost algoritmov

Primerjanje algoritmov (benchmarking)

merjenje časa izvajanja algoritmov na standardnih množicah podatkov
primerjanje zmogljivosti računalnikov

Red velikosti - razred n

Θ(n): množica funkcij, ki rastejo linearno
linearna časovna kompleksnost
v odvisnosti od velikosti problema (št. vhodnih podatkov)
sprememba v velikosti je konstantna z večanjem n
red velikosti n

Primer sortiranja z izbiranjem

Sortiranje: naloga urejanja neurejenih elementov v urejeno zaporedje (npr. po velikosti, po abecedi)
Zelo pogosta naloga, zelo koristne rešitve
Zelo veliko pristopov (različno učinkovitih)
Sortiranje z izbiranjem:
vedno znova preišči neurejen del seznama
v vsakem prehodu izberi največji element
zamenjaj z njim zadnji element neurejenega seznama

Glavne enote dela: skrite v koraku „najdi največjega“
S časom se ta čas manjša:
(n-1) + (n-2) + … + 2 + 1 = n (n-1) / 2
Kvadratična časovna kompleksnost Θ(n2)

Red velikosti - razred n^2

Red velikosti n2: $\Theta(n^2)$ – algoritem opravi cn2 dela, da sprocesira n elementov. Vsaka funkcija reda velikosti n2 ima od neke točke naprej večje vrednosti od funkcije reda velikosti n
ne glede na konstantni faktor.

Analiza algoritmov

Primerjamo algoritme, ki opravijo enako nalogo in vrnejo isti rezultat
Primerjamo časovne kompleksnosti
Primerjamo prostorske kompleksnosti

Čiščenje algoritmov Primer 1

Čiščenje nezaželenih (označenih) podatkov iz niza:

> *“Given a collection of age data, where erroneous zeros occur, find and remove all the zeros from the data, reporting the number of legitimate age values that remain”*

Tri različne rešitve (best, worst, best average)
Algoritme bomo primerjali z analizo algoritmov

Algoritem 1: Premakni v levo
gremo čez seznam
ko najdemo 0, pomaknemo vse elemente na desni za en element levoAlgoritem 1: Premakni v levo
gremo čez seznam
ko najdemo 0, pomaknemo vse elemente na desni za en element levo

```
1.  Get values for n and the n data items
2.  Set the value of legit to n
3.  Set the value of left to 1
4.  Set the value of right to 2
5.  While left is less than or equal to legit do Steps 6 through 14
6.      If the item at position left is not 0 then do Steps 7 and 8
7.          Increase left by 1
8.          Increase right by 1
9.      Else (the item at position left is 0) do Steps 10 through 14
10.         Reduce legit by 1
11.         While right is less than or equal to n do Steps 12 and 13
12.             Copy the item at position right into position (right – 1)
13.             Increase right by 1
14.         Set the value of right to (left + 1)
15. Stop
```

Časovna učinkovitost:
Štejemo primerjave, ko iščemo 0 in premike elementov
Najboljši primer:

- brez 0
- samo preverjamo vse vrednosti
- red velikosti Θ(n)
  Najslabši primer:
- vse vrednosti so 0
- najprej premaknemo n-1 elementov, nato n-2 elementov, itn.
- red velikosti Θ(n2)
  Prostorska učinkovitost:
  samo pomožne spremenljivke
  nič pomembnega razen vhoda
  Ne potrebujemo dodatnega pomnilnika

Primer 2

Algoritem 2: Kopiraj
gremo čez seznam

kopiraj dobre elemente na drugi (na začetku prazni) seznam

1. Get values for *n* and the *n* data items
2. Set the value of *left* to 1
3. Set the value of *newposition* to 1
4. While *left* is less than or equal to *n* do Steps 5 through 8
5. If the item at position *left* is not 0 then do Steps 6 and 7
6. Copy the item at position *left* into position *newposition* in new list
7. Increase *newposition* by 1
8. Increase *left* by 1
9. Stop

The copy-over algorithm for data cleanup

Časovna učinkovitost:
Štejemo primerjave, ko iščemo 0 in premike elementov
Najboljši primer:

- brez 0
- samo preverjamo vse vrednosti, jih ne kopiramo
- red velikosti Θ(n)
  Najslabši primer:
- vse vrednosti so 0
- preverimo vse vrednosti in vse kopiramo
- red velikosti Θ(n)
  Prostorska učinkovitost:
  Najboljši primer: vse 0, ne potrebujemo nič dodatnega prostora
  Najslabši primer: nobene 0, potrebujemo n dodatnega prostora
  Kompromis med časovno in prostorsko učinkovitostjo

Primer 3

Algoritem 3: Bližanje kazalcev
levi kazalec gre proti desni in se ustavi na 0
desni kazalec gre z desne proti levi
ko je najdena vrednost 0, se zamenjata vrednosti
ustavi, ko se kazalca srečata

1. Get values for *n* and the *n* data items
2. Set the value of *legit* to *n*
3. Set the value of *left* to 1
4. Set the value of *right* to *n*
5. While *left* is less than *right* do Steps 6 through 10
6. If the item at position *left* is not 0 then increase *left* by 1
7. Else (the item at position *left* is 0) do Steps 8 through 10
8. Reduce *legit* by 1
9. Copy the item at position *right* into position *left*
10. Reduce *right* by 1
11. If the item at position *left* is 0, then reduce *legit* by 1
12. Stop

Časovna učinkovitost:
Štejemo primerjave, ko iščemo 0 in premike elementov
Najboljši primer:

- brez 0
- levi kazalec gre gladko do desnega, samo preverjamo vse vrednosti
- red velikosti Θ(n)
  Najslabši primer:
- vse vrednosti so 0
- preverimo vse vrednosti in vse kopiramo z desne
- red velikosti Θ(n)
  Prostorska učinkovitost:
  samo pomožne spremenljivke
  nič pomembnega razen vhoda
  Ta verzija je tako časovno kot prostorsko zelo učinkovita

Binarno iskanje

Iskanje vrednosti po urejenem seznamu
Preglej srednji element in potem preišči ustrezno polovico preostalih podatkov

1. Get values for *NAME*, n, $N_1, \ldots, N_n$ and $T_1, \ldots, T_n$
2. Set the value of *beginning* to 1 and set the value of *Found* to NO
3. Set the value of *end* to n
4. While *Found* = NO and *beginning* is less than or equal to *end* do Steps 5 through 10
5. Set the value of m to the middle value between *beginning* and *end*
6. If *NAME* is equal to N_m, the name found at the midpoint between *beginning* and *end*, then do Steps 7 and 8
7. Print the telephone number of that person, T_m
8. Set the value of *Found* to YES
9. Else if *NAME* precedes N_m alphabetically, then set *end* = $m - 1$
10. Else (*NAME* follows N_m alphabetically) set *beginning* = $m + 1$
11. If (*Found* = NO) then print the message 'I am sorry but that name is not in the directory'
12. Stop

Binary search algorithm (list must be sorted)

Časovna učinkovitost:
Štejemo primerjave s ciljno vrednostjo
Najboljši primer:

- iskana vrednost je že prva sredinska vrednost
- samo 1 primerjava
  Najslabši primer:
- vrednosti ni na seznamu
- ponovi toliko krat kolikor lahko razdeliš seznam na pol
- red velikosti Θ(log n)
  Prostorska učinkovitost:
  samo pomožne spremenljivke
  nič pomembnega razen vhoda

Nismo žrtvovali prostorske za časovno učinkovitost
Smo pa žrtvovali splošnost – potrebujemo urejen seznam!

Red velikosti - razred log n

Red velikosti log n: Θ(log n) – algoritem opravi le c*log(n) dela, da sprocesira n elementov
narašča zelo počasi

Ujemanje vzorcev

Časovna učinkovitost:
Štejemo primerjave

Najboljši primer:
vzorca ni v besedilu
prvega znaka vzorca ni v besedilu
KLMNPQRST ← ABC
samo n-m+1 primerjav
m<<n ⇒ Θ(n)

Najslabši primer:
vzorca ni v besedilu, vsi znaki vzorca razen zadnjega so v besedilu
AAAAAAAAAA ← AAAB
vzorec je v besedilu, na vsaki lokaciji
AAAAAAAAAA ← AAAA
samo m(n-m+1) primerjav
m<<n ⇒ Θ(m*n)

Red velikosti razred k^n

Večina algoritmov je polinomsko omejenih (reda nk)
Nekateri niso
ne moremo jih rešiti v polinomskem času, niti pri velikih k!
Eksponentna časovna kompleksnost: Θ(kn)

Hamiltonov cikel

Ogromno število pregledovanj:
višina drevesa: n+1
število listov: 2n= število poti, ki jih moramo pregledati

Eksponentni algoritem: Θ(kn)

Praktično nedosegljive rešitve (ang. intractable)
Problemi , ki niso polinomsko omejeni
Hamiltonov cikel
Problem trgovskega potnika
Polnjenje nahrbtnikov
Šah

Redi velikosti

V računalništvu poznamo več razredov časovne kompleksnosti:
Konstantna O(1)
Logaritemska O(log n)
„Linearitmična“ O(n * log n)
Linearna O(n)
Kvadratična O(n2)

Kubična O(n3)
Polinomska O(np), p > 0
Eksponentna O(2n)

Pravila pri določanju razredov kompleksnosti

Preštejemo število operacij:
Aritmetične operacije: x+1
Primerjave: x>0?
Prirejanje: x:=1
Klice sistemskih funkcij: Vnesi x

Produkt: f*O(g) ϵ O(f*g)
Ponavljanje dela kode
Primer: če se n-krat ponovi del kode z O(n), je skupna kompleksnost O(n2)
Vsota: O(f1 + f2) = O(max(f1, f2))
Zaporedni segmenti kode
Skupna kompleksnost je enaka večji kompleksnosti
Pri vejitvah upoštevamo pot z večjo kompleksnostjo
Množenje s konstanto: O(k*g) = O(g)
Množenje s konstanto ne vpliva na razred kompleksnosti

Približni algoritmi

Približni algoritmi rešijo problem približno
ne zagotavljajo optimalne rešitve
običajno pa zagotovijo dovolj dobro rešitev (v sprejemljivem času)

Primer: Polnjenje nahrbtnikov
v osnovi je to eksponentni problem
približni algoritem:
preveri vsak trenutni nahrbtnik
če gre nov paket v nahrbtnik, ga postavi vanj
če paket ne gre v noben nahrbtnik, dodaj nov nahrbtnik
dovolj dobra (in praktična) rešitev

ONTOLOGIJA na ustnem!

11. Varnost, komunikacija in zaščita vsebin

IKS

Podatki in informacije imajo veliko vrednost in so večinoma v elektronski obliki Prinašajo veliko prednosti in tudi veliko novih nevarnosti.

Nevarnosti za podatke

- Izguba: okvare naprav, izpad energije, nesreče, zlonamerno brisanje
- nedostopnost: pozabljeno geslo, prekinjene komunikacijske povezave
- kraja: vdor v sistem, nepooblaščen dostop do naprav
- neželeno spreminjanje: napake pri shranjevanju, zlonamerno spreminjanje
- naravni/človeški: neprevidnost, neznanje, težko obvadljivi

buzzword:
blockchain: Blockchain je sistem beleženja informacij na način, ki otežuje ali onemogoča spreminjanje, vdiranje ali goljufanje sistema. Blockchain je v bistvu digitalna knjiga transakcij, ki je podvojena in porazdeljena po celotnem omrežju računalniških sistemov v blockchainu.

Stopnja varnosti IKS

Varnost IKS:

- poskušamo zmanjševati tveganje
- večja varnost, manjša uporabnost

Stroški IKS:

- stroški varovanja ne presegajo vrednost informacij
- IKS ni varen kadar stroški presegajo njegove koristi

Dokumenti in celovitost podatkov

Dokumenti so podatki s posebnim statusom:

- lahko imajo določeno pravno veljavo
- veljava odvisna od vrste dokumenta
- ima določeno lastnost: vidike celovitosti podatkov

Celovitost podatkov:

- eden najpomembnejših vidikov varnosti IKS
- dokument mora izpolnjevati vse ali zgolj nekatere vidike celovitosti

Vidiki:

- Verodostojnost: od potrditve ni bil spremenjen
- Avtentičnost: lahko ugotovimo kdo je avtor
- Neovrgljivost: avtor ne more zanikati avtorstvo

- Časovna opredeljenost: z gotovostjo lahko ugotovimo kdaj je bila zagotovljena verodostojnost
- Tajnost: zakriptiramo dokumente
- Trajnost: da ne pride do izgube v času njihove veljavnosti

Zagotavljanje celovitosti podatkov

Papirni dokumenti
verodostojnost, avtentičnost in neovrglivost se zagotavlja z lastnoročnim podpisom
časovna opredeljenost z notarsko overitvijo

Elektronski dokumenti
verodostojnost, avtentičnost, neovrgljivost, tajnost in časovno opredeljenost se zagotavlja s šifrirnimi postopki

Varnostna politika IKS

Sprejeti ga mora vsako podjetje, standard ISO/IEC. Zaščiti moramo nosilce podatkov.

Elementi varnostne politike IKS: opredelitev ekipe, ocena tveganja, potencialno ogroženi elementi, postopek za spreminjanje, pravila za varno in odgovorno ravnanje posameznikov

BOTNET - Botnet je več naprav, povezanih z internetom, od katerih vsaka poganja enega ali več botov. Botneti se lahko uporabljajo za izvajanje napadov porazdeljene zavrnitve storitve, krajo podatkov, pošiljanje neželene pošte in omogočanje napadalcu dostop do naprave in njene povezave.

NDA - Pogodba o nerazkritju informacij je pogodba, s katero stranki vzpostavita dolžnost varovanja tajnosti določenih informacij, ki so med njima zaupana.

Informacijsko komunikacijski sistem IKS

Skup strojne in programske opreme, ki omogoča učinkovit način za shranjevanje, urejanje, upravljanje, iskanje, prenos in prikaz podatkov (podatki na njem in uporabniki).

Strojna oprema: strežniki, terminali, miške, tipkovnice, zasloni, mikrofoni,...

Računalniška oprema

Strežnik: je naprava ali proces, ki omogoča dostop do svojih virov enemu ali več uporabnikom hkrati.

Naprave za shranjevanje

*Kriptografija: ukvarja z zaščito informacij, tako da jih zašifrira v neberljiv format, ki se imenuje šifriran tekst (**ciphertext**). Temu postopku pravimo šifriranje oziroma **enkripcija** podatkov*

Shranjevanje podatkov je ena od ključnih nalog IKS-ja.

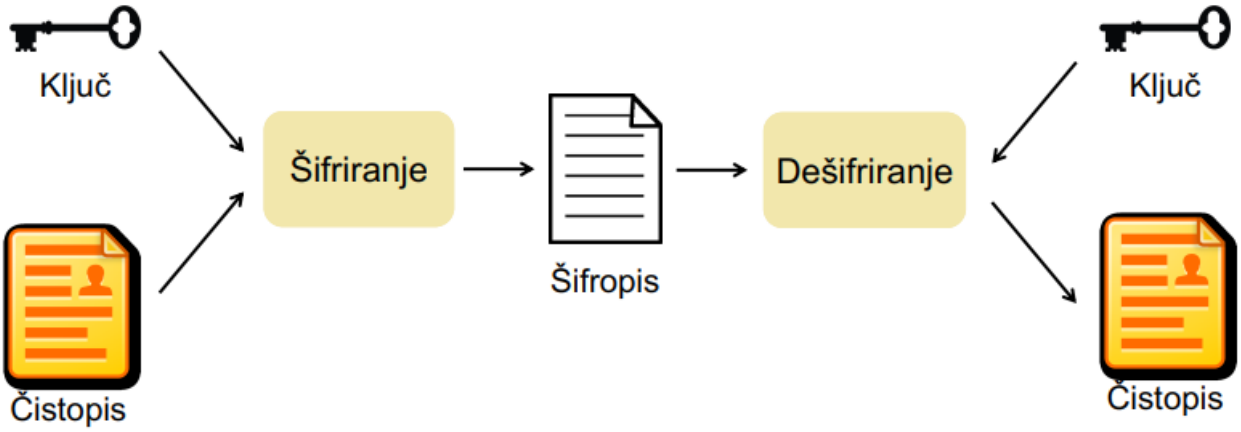
Lastnosti: stalnost zapisa (nestanovitni (napajanje) / stanovitvni), spremenljivost zapisa (večkratno zapisovanje), dostop do podatkov (neposreden/zaporeden)

naprave: obnavljalnik, stikalo, most, modem, prehod, požarni zid,...

Šifriranje

Šifriranje je postopek, ki razumljive podatke pretvori v obliko, ki je v splošnem nerazumljiva. Da bi podatke pretovrili nazaj v razumljivo obliko potrebujemo tajni ključ, da odklenemo zaklenjene podatke.

- Čistopis (**plain text**) – nešifrirani podatki
- Šifropis (**cipher text**) – šifrirani podatki
- Šifriranje (**encryption**) – postopek pretvorbe čistopisa v šifropis
- Dešifriranje (**decryption**) – postopek pretvorbe šifropisa v čistopis
- Šifra (**cypher**) – šifrirni postopek, ki združuje šifriranje in dešifriranje



Tajnost šifriranega sporočila je lahko osnovana na: tajnost šifre, **tajnost ključa**

Osnovne delitve šifer

- Glede na vrsto ključa: Simetrične (isti ključ uporabljen za šifriranje in dešifriranje) /Asimetrične šifre (dva ključa: javni in zasbeni, relacija med njima je kompleksna)
- Glede na potek šifriranje: Pretočne (uporablja se ko informacija ni popolnoma znana, videokonferenca)/Blokovne šifre (imamo že podatke, in podatke zvijemo v kose in jih posamezno zvijemo)
- Glede na osnovni princip: Tajnost ključa, tajnost šifriranega postopka

Šifriranje

Kriptografija - veda, ki se ukvarja s skrivanjem vsebine sporočila z uporabo šifriranja in dešifriranja
Šifriranje- pretvori originalno sporočilo v šifrirano sporočilo
Dešifriranje - pretvori šifrirano sporočilo nazaj v originalno obliko

Enostavni šifrirni algoritmi

Cezarjeva šifra: 25 ključev, pretočno šifriranje (šifrira en znak naenkrat), substitucijsko šifriranje (preslikaj posamezne znake v zanke oddaljene za določeno razdaljo)

Blokovno šifriranje: šifrirajo se bloki in posamični znaki, matrična bločna šifra, vsak znak prispeva k šifriranju več znakov

Varnost šifer

Napad na šifro izkorišča: šibkosti postopka, šibkosti ključev. Osnova za napad je lahko znan en ključ, šifropis, čistopis ali šifro-ćistopis. Cilj napada je dešifriranje enega šifropisa, razkriavanje ključa in razbijanje šifre.

Pri načrtovanju šifer se moramo izgoniti šibkostim in praktično preizkusiti njihovo varnost.

Teoretično varne šifre ni mogoče razbiti niti teoretično, četudi ima napadalec na voljo neomejena sredstva in neomejeno časa (Enkrat uporabimo ključ enkrat besedilo).

Šifra je **praktično varna**, če je uspešen napad nanjo praktično neizvedljiv

Sodobne šifre so izredno močne, odporne proti znanim postopkom kriptoanalize, napadalci iščejo slabosti v varnovanju IKS

Varnost - postopki

Avtentikacija - ugotavljanje identitete: uporabniško ime, geslo, datoteka z gesli je šifrirana, dodatna vprašanja, biometrična prijava, identični gesli se ne bosta enako šifrirali.

Avtorizacija: z avtorizacijo upravljamo kaj lahko uporabnik počne. OS dovoli samo avtorizirane posege/dostop do virov. Admin univerzalen dostop.

Grožnje iz omrežja

Zlonamerna programska oprema (Malware)

- **virus**: program vdelan v drugi program, se razmnožuje in razširja po omrežju, ko je program pognan (npr. v datotekah pripetih sporočilom elektronske pošte)

- **črv**: se lahko replicira in razširja sam, ne da bi bil posredovan v okuženem programu
- **Trojanski konj**: program, ki zgleda koristen, a vsebuje skrito zlonamerno kodo

Napad DOS (Denial of Service) – onemogočanje storitve: veliko računalnikov skuša hkrati dostopati do istega URL; DDOS – porazdeljen DOS, armada zombijev (botnet)

Spletno ribarjenje (phishing) - pridobitev osebnih podatkov

Zaščita pred grožnjami iz omrežja

Protivirusna programska oprema (antivirus software)

- detektira in odstrani črve, viruse, Trojanske konje
- redno osveževanje!

Požarni zid (firewall software)

- blokira komunikacijo, ki ni dovoljena

Protivohunska programska oprema (antispyware)

- detektira zlonamerno vohunsko programsko opremo

Vedno namesti zadnje varnostne programske popravke

Šifriranje z DES: simetrični šifrirni algoritem, načrtovan za digitalne podatke, hiter in učinkovit

Kongurečni razredi:
Mali Fermatov izrek:
DH - simetrični:
RSA - Asimetrični

Varen prenos preko spleta

Za varno poslovanje potrebujemo algoritme:: SSL (Secure Socket Layer), TLS (Transport Layer Security). Uporabljata RSA in DES (delujeta po principu odjemalec/strežnik). Rokovanje na začetku inicializira varen prenos. RSA je računsko zahteven zato se z njim na varen način prenese samo simetrični ključ. DSA je zelo hiter, zato se z njim prenese celotni sporočilo.

Varnost v vgrajenih računalnikov: vgrajeni računalniki, nov trend [povezava vgrajenih računalnikov na mrežo], napadi na vgrajene računalnike.

12. Omrežja

Mrežni protokoli in tehnologije

Osnovni koncepti

Računalniško omrežje:

- vozlišča
- povezave

Načini povezave: **zično**(klicne povezave, šiirokopasovna omrežja) in **brezžično omrežja** (WLAN, WWAN)

Software-defined networking - *omrežna tehnologija je pristop k upravljanju omrežja, ki omogoča dinamično, omrežno konfiguracijo za izboljšanje zmogljivosti in spremljanja omrežja, zaradi česar je bolj podobno računalništvom v oblaku kot tradicionalno upravljanje omrežja*.

Analogne klicne povezave

- Analogne telefonske linije
- Prenos digitalnih podatkov
- Modem modulira podatke
- Prenos do 56kbps
- Povezava se prekine

Širokopasovne povezave

Prenos > 256kbps
Domači uporabniki (DSL, ADSL: neprestana povezava, naročniška linija), kabelski modem (TV), optična povezava (100Mbps)

Poslovni uporabniki:

- **Ethernet** (koaksialni kabel, 10Mbps),
- **Fast Ethernet**(koaksialni kabel, optična vlakna, bakrene parice, 100Mbps),
- **Gigabit Ethernet Standard** (IEEE Standard, 1Gb/s)

Brezžične povezave

WLAN (WIreless Local Area Network)

- brezžično krajevno omrežje
- brezžični usmerjevalnik (wireless router)
- povezan na širokopasovno povezavo

Wi-FI (Wireless Fidelity)

- IEEE 802.11 wireless network standard
- doseg do 100m
- frekvenca 2.4GHz
- prenos 10-50Mbps

Bluetooth

- doseg 10-15m
- izmenjava informacij med bližnjimi napravami

WWAN (Wireless wide area network)

- brezžično prostrano omrežje
- preko infrastrukture mobilne telefonije
- 4G, 5 - 20 Mbps
- mobilno računalništvo

NFC

- visokofrekvenčna komunikacijska tehnologija kratkega dosega,
- omogoča izmenjavo podatkov na razdalji do 10 cm,
- standard ISO/IEC 14443
- pametna kartica in čitalec v eno napravo

Krajevno omrežje

LAN (Local Area Network)

- celotna infrastruktura v lasti uporabnika
- en kabel za krajše razdalje
- več kablov za daljše razdalje
- ponavljalnik ojača signal
- most pošilja sporočila le, ko je potrebno

Prostrano omrežje

WAN (Wide Area Network)

- žična povezave
- velike razdalje
- namenske povezave
- paketni prenos med vozlišči od začetnega do ciljenga vozlišča
- redundatne poti, robustnost, odprava napak
- prilagodljive poti gleden na promet v omrežju

Internet

Struktura interneta

- kombinacija WAN in LAN,
- povezani z usmerjevalniki, ki usmerjajo promet,
- ponudniki intenetnih storitev (ISP)

Komunikacijski protokoli

Protokol: standardizirana množica pravil za komunikacijo
Nevladna neprofitna organizacija Internet Society
Internetna hierhija protokol:
5. Aplikacijska plast (HTTP, SMTP, FTP),
4. Prenosa (TCP, UDP),
3. Omrežna (IP),
2. Povezavna (PPP, Ethernet),

1. Fizična (Modem, DSL, Wifi, 4G)

FIzična plast

- Skrbi za prenos bitov prek prenosnega medija.
- Zagotavlja stanardno aparaturno prikljjučevanje sistemov na prenosni medij.
- Pravila za izmenjavo podatkov prek fizičnega kanala:
  - ali je bit prisoten ali ne
  - koliko časa bo bit prisoten
  - kaj predstavlja 0 in kaj 1
  - oblika priključka med računalnikom in omrežjem

Povezavna plast

- Zagotavlja zanesljiv prenos bitov
- Uokvirjanje - združevanje bitov v sporočila
- Medium Access Control:
  - Nadzor nad dostopod do medija,
  - določanje naprave, ki lahko v določenem času uporablja povezavo,
  - porazdeljen nadzor
- Detekcija in popravljanje napak pri prenosu

Logical Link Control (nadzor nad logično povezavo, pravila za detekcijo in odpravljanje napak, algoritem ARQ)
paket vsebuje: označbe za začetek in konec paketa, zaporedno številko paketa, podatke, bite za preverjanje napak

Mrežna plast

- Skrbi za prenos sporočil po omrežju od začetnega do ciljenga vozlišča

Zahteve: standard za poenoteno naslavljanje vseh vozlišč, metoda za usmerjanje sporočil (**routing**)

Internetni omrežni protokol (IP)

Naslavljanje:

- ime gostitelja
- IP naslov: enolični naslov
- DNS (Domain Name Service) preslika imena v IP naslove - DNS strežnik

Usmerjanje sporočil: izbiranje optimalne (najkrajše/najhitrejše) poti do izvora do ponora dinamično in ogromna omrežja, učinkovitost poti

Prenosna plast

- Povezovanje aplikacije z aplikacijo
- Številka vrat enolično določa aplikacijo
    - spletni strežnik: 80
    - DNS: 42
    - SMTP: 25
- TCP (Transport Control Protocol)
    - zagotavlja pravilni vrstnih red paketov
    - drugi algoritem ARQ (Automatic Repeat reQuest) is an error control and packet recovery method for data transmission in which the receiver sends an alert to the sender
    - zagotavlja navidezno neposredno povezavo med programi

Aplikacijska plast

- Zagotavlja prenos ustrezno formatiranih podatkov med aplikacijami

Nekaj priljubljenih aplikacijskih protokolov:

Acronym	Name	Application	Well-Known Port
HTTP	Hypertext Transfer Protocol	Accessing Web pages	80
SMTP	Simple Mail Transfer Protocol	Sending electronic mail	25
POP3	Post Office Protocol	Receiving electronic mail	110
IMAP	Internet Message Access Protocol	Receiving electronic mail	143
FTP	File Transfer Protocol	Accessing remote files	21
TELNET	Terminal Emulation Protocol	Accessing remote terminals	23
DNS	Domain Name System	Translating symbolic host names to 32-bit IP addresses	42

URL (Uniform Resource Locator) – enolični določitelj vira

- protokol://naslovGostitelja/stran
- protokoli: http, mailto, news, ftp, ...
- Primer: HTTP (Hypertext Transfer Protocol)

Mrežna komunikacija

Medosebna komunikacija

- elektronska pošta
- internetni forumi
- videokonferenčni sistemi (Skype)
- socialna omrežja (Twitter, Facebook)

Skupna raba

- tiskalniški strežniki
- datotečni strežniki
- paradigma odjemalec/strežnik
- porazdeljene podatkovne baze
- sistemi za skupinsko delo (Wiki,...)

Elektronsko poslovanje

- bančni avtomati,...
- spletne trgovine, spletno bančništvo, elektronsko plačevanje, poslovanje med podjetji...

13. Umetna inteligenca

Uvod

Imamo probleme nepolinomske probleme, tukaj nam pomaga AI.
Umetna inteligenca (artificial intelligence, AI): izdelati računalniške sisteme, ki bi se ponašali z nekaterimi aspekti inteligence

Kvántna prepletènost je kvantnomehanski fizični pojav, ki nastane ko se pari ali grupe delcev ustvarijo, interaktirajo ali ko si delijo prostorsko bližino na tak način da se kvantno stanje vsakega delca ne da opisati neodvisno od stanja drugih, tudi če so delci ločeni na veliki razdalji - kvantno računalništvo ... če foton na enmu koncu sveta obrnemo se bo na drugem koncu sveta istočasno obrnil

Kaj je to inteligenca?

Turingov test

- človek se pogovarja z dvema sogovornikoma
    - eden je človek
    - eden je stroj

če ne more razločiti med človekom in strojem, je stroj opravil **Turingov test**

Turingov test je preizkus zmožnosti stroja izkazovati inteligentno obnašanje, ki je ekvivalentno oz. nerazločljivo od človeškega.

Različni tipi opravil

- Računske naloge
    - računanje, avtomatska obdelava podatkov (npr. izračun plač)
    - tipično imajo (poznane) algoritmične rešitve
    - računalnik hitrejši in natančnejši od človeka
- Razpoznavanje
    - zaznavanje, razpoznavanje, akcija, razpoznavanje obrazov, slik, akcij, govora
    - procesiranje velike količine senzorske informacije
    - dostop do shranjenih modelov dobljenih z izkušnjami
    - človek precej boljši od računalnika
- Zdravorazumsko sklepanje
    - razumevanje, zdravorazumsko sklepanje, načrtovanje,...
    - enostavno formalno sklepanje se da avtomatizirati, kompleksno zelo težko
    - zelo težke naloge za stroj

Predstavitev znanja

1. Naravni jezik

- uporaba zahteva razumevanje pomena posameznih besed

2. Formalni jezik

- jezik formalne logike (IN, ALI,...)

3. Grafična predstavitev

- znanje predstavljeno z vozlišči povezanimi s povezavami (lahko narišemo)
- Semantične mreže

4. Idr.

- ustreznost, učinkovitost, razširljivost, primernost

DL izračunljiva

Razpoznavanje

Razpoznavanje

- Objektov
- Lastnosti
- Obrazov
- Prostorov
- Govora,...

Nevronske mreže

Inspiracija je dogajanje v človeških možganih
Nevroni procesirajo informacijo
Ogromno zelo enostavnih procesnih enot

- konekcionistični princip

(Umetna) nevronska mreža

- aktivirani ali neaktivirani uteženi vhodi
  če vsota uteženih vhodov presega prag, nevron odda signal na izhodu

YOLO library - real time object detection (procesira slika pri 30FPS)

Problem: Ne vidi na čem sklepa ta mreža na kakšen način odloča. Imamo metode, ki nam omogočajo, da vidimo kako sklepa.

Učenje

Na osnovi učnih primerov se nevronska mreža nauči pravilne parametre; na podlagi podatkov → Strojno učenje

Različni načini učenja

- usmerjano, deloma usmerjano, samostojno
- pasivno, interaktivno
- paketno, inkrementalno

Dve vrsti učenja:

- unsupervised: nimamo kontrolne kako se nekaj uči algoritem (napad na omrežje)
- supervised: ugotavljamo, zapisujemo in mreža se uči na podlagi nečesa kar je (problem v omrežju)

Prirojeno:priučeno

Sklepanje

Odločitvena drevesa - pravila lahko postavimo v hierhični sistem, začnemo z najbolj pomembnimi kriteriji na tak sistem se sistem nauči najbolj pomembne kriterije

Graf prostora stanj (state-space graph)

- vsako vozlišče je stanje v našem problemu
- povezano je z drugimi stanji v katera se lahko neposredno pride iz trenutnega stanja
  Cilj: poiskati pot od začetnega stanja do končnega stanja

Algoritmi za iskanje

- groba sila
- Hevristike

Primeri: dama, tri v vrsto, šah, GO

Ekspertni sistemi

Ekspertni sistem oponaša razmišljanje na neki specifični domeni → Baza znanja (lahko brskamo po njen in ugotavljamo nove stvari - Prolog jezik)

Deduktivno sklepanje:

- Na osnovi dejstev in pravil se ugotovi nova dejstva in pravila (in se dodajo v bazo znanja)

Primer: **Prolog**

Programiranje v tem jeziku poteka v simbolni logiki, jezik pa je izvirno zasnovan za dokazovanje izrekov, sedaj pa je bolj splošno v uporabi v umetni inteligenci.

Razlaga odločitev

- veriga sklepanja

Inteligentni sistemi

Programski inteligentni sistemi

- Personalizirani iskalniki
- Priporočilni sistemi (pametna pogodba, sestavi pogodbo za programerja npr.)
- Inteligentni programski asistenti

Pasivni umeščeni (situated): robotski sistemi

Aktivni utelešeni (embodied): robotski sistemi

Robotika

Robot je stroj, ki ga nadzoruje računalnik in ga lahko programiramo, da samostojno opravlja določeno opravilo.

Vrste robotov:

- Industrijski roboti
- Robotski manipulatorji
- Mobilni roboti
- Brezpilotni letalniki, ...

Spoznavni sistemi

Kognitivni asistent

- Razišče okolico in zgradi zemljevid
- Se nauči prepoznati in identificirati predmete
- Razume namen in funkcije predmetov
- Zna interpretirati verbalno in neverbalno komunikacijo ljudi v okolici
- Zazna nove situacije in ustrezno reagira

Vgrajene osnovne funkcionalne sposobnosti, ki jih razvija in nadgrajuje z učenjem

14. Računalniška grafika

Na tem področju je razvitih veliko algoritmov v zadnjih 30 letih.
1974 - Atari Pong

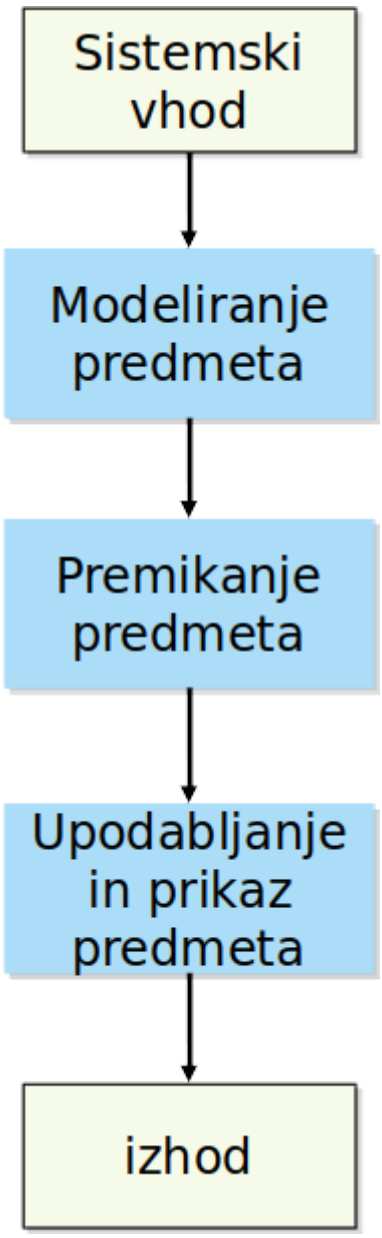
Ray Tracing

Animacija

V preteklosti je bila **risana animacija, animacija s transparentnimi plastmi, animacija z ustavljanjem gibanja** danes pa računalniška grafika

Osnovni principi

Grafični cevovod (vhod): modeliranje predmetov → gibanje predmov → upodabljanje in prikaz predmetov



Modeliranje predmeta

Modeliranje z žičnim modelim (wireframe modelling - predstavitev 3D fizičnega predmeta, ki se uporablja v računalniški grafiki), modelira se površina predmeta z poligonsko mrežo (trikotniki)

Gibanje predmetov

Nekaj transformacij:

- togo gibanje:
  - traslacija (vsaka točka se enako spremeni)
  - rotacija
  - zrcaljenje
  Elementi se preslikajo v osnovne operacije linearne algebre. Kontrolne točke nadzirajo givanje,

Interpolacija parametrov gibanja predmeta (kako se prilega objekt → orientacija, različne krivulje lahko uporabimo da izračunamo pot gibanja, smer, spreminjamo rotacijo predmeta, autocad)

Upodabljanje predmetov

Upodabljanje (rendiranje) predmetov v 3D sliko. Upoštevamo: Osvetlitev, Barvanje, Senčenje, Lepljene strukture (na poligone nalepi teksturo)m Meglenje

Ray Tracing - algoritem, lahko z različnimi viri svetlobe osvetlimo predmet in naredimo projekcijo in izračunamo moč osvetljenosti in smer svetlobe, ki iz vira svetlobe pada na poligon na ta način sestavimo 2D sliko. Slediti moramo žarkom, kar je lahko računsko zelo potratno.

Prikaz predmetov

Določimo pogled opazovalca → na osnovi pogleda prikažemo sliko

Video igre

Realistične igre zahtevajo dobro rač. grafiko, v običajni grafiki se okolje ne spreminja (enkrat se zgenerira animacije, generiranje je lahko potratno, prikažemo jih lahko poljubno krat, enak prikaz vedno).

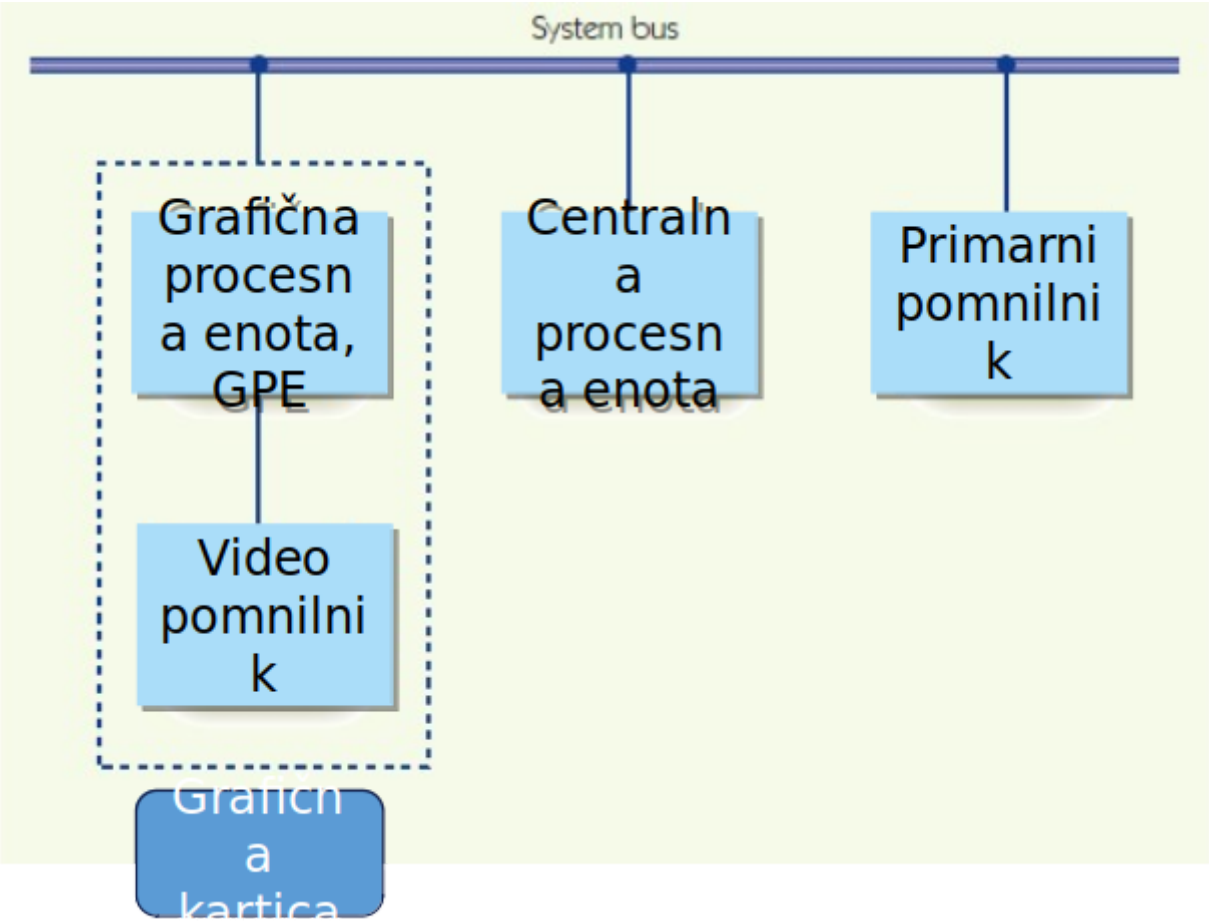
V video igrah je okolje bolj dinamično in interaktivno (reakcije igralca, drugačen potek, izjemno hitro izrisovanje).

Za to so pa potrebni zelo učinkoviti algoritmi in strojna oprema (Grafična procesna enota GPU).

GPU

Grafična procesna enota:

- ločeni procesor optimizran za delo z rač. grafiko
- namenski pomnilnik
- na grafični kartici



Pohitritve

Tehnike, ki pohitrijo izvajanje iger:

- izogibanje računsko zahrevnim algoritmom
- no v-sync, no shadows, low quality texture
- cullling: naprej ugotovi kateri poligoni so vidni, šele nato jih izloči (FOV)
- uporaba vnaprej izrisanih predmetov (cut-ins): shranje v knjižnico

Igre z več igralci

MMOG - igre z tišoč igralcev

Arhitektura odjemalec/strežnik: igralni strežniki, navidezni svet, igralci igrajo preko odjemalnik programov (Steam)

15. Tehnologija in družba

Tehnologija in družbeno nomiranje

pacing problems problem tempa ali problem of regulatory connection (problem regulative povezave): nesinhroniziranost tehnološkega razvoja in regulatornega odziva

Paradoks negotovosti ali Collingridgeva dilema

Metodološka uganka, v kateri imajo prizadevanja za reguliranje tehnoologije dvojno težavo:

- informacijsko težava: dokler tehnologija ni dovolj razvita, ni dovolj informacij o mrebitnih posledicah
- težava moči: ko je tehnologija dovolj razvita/razširjena, jo je težko spreminjati ali nadzirati

Poenostavljeno:

- Pretirana začetna regulacija preprečuje razvoj
- Neučinkovitost prepozne regulacije

Precautionary priciniple (Načelo previdnosti): Pred uporabo novih tehnologij je treba dokazati, da ne bodo povzročile škode za družbo

1. Pametne Specializacije
   Je program, ki temelji na podjetniškem odkrivanju. Z namenom, da ugotovijo katere stvari posamezniki in podjetja potrebujejo in na kakšen način lahko zadovoljimo porabo sodobnih programov in prog. opremo.

Pricinple-based regulation (Regulacija temelječa na načelih): odpoved natančnim pravnim pravilom, soglasje o splošnih vodilnih načelih, ki naj usmerjajo razvoj tehnologije.

Etika

Etika se ukvarja s kriteriji moralnega vrednotenja. Delitev na: meta-etika - preučevanje narave (kaj), Normativna etika (kako; etika dolžnosti, etika vrline, teleološka etika), Uporabna etika (v katerih primerih).

16. Tehnologije veriženja blokov

Aplikacije: Pametne pogodbe (program je skopiran povsod - ko se vsi strinjajo se izvede program), decentralizirane aplikacije, Decentralizirane avtonomne oragnizacije (DAO) (Podatki so porazdeljeni po večih računalnikih ... omogoča izredno hitro sodelovanje). Denarnica. Razviti ekonomske sisteme, ki predstavljajo ekosistem (menjava izposoja avtomobilov).

Izvajalno okolje: Virtualni stroji (hrani bloke in trenutno stanje transakcije), bloki, transakcije.

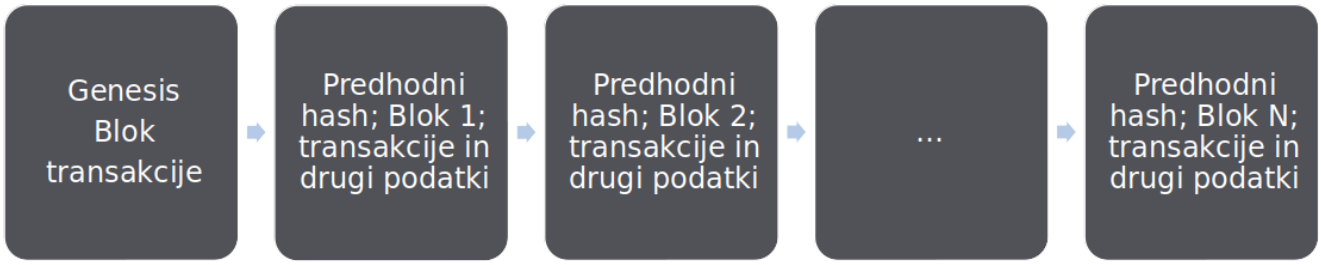
Konsenz (zmenijo kdo bo ustvaril naslednji blok): podvajanje stanjskih strojev, konsenz na podlagi dokuzov, tradicionalni bizantinski protokol tolerance napak ()

Kriptografija: na podlagi javnih ključev, digitalni podpisi, zgostitvene funkcije

Enak z enaki (P2P): tračevski protokol, usmerjevalni protokol, poplavni protokol

Mreža: Internet, TCP/IP

Veriga blokov



Prve transakcije → Naredi se hash, ki določa prejšni blok, naredijo se nove transakcije, vse transakcije ne morejo priti v en blok. Več plačaš hitreje grejo bloki čez.

Veriga dogodkov: veliko dogodkov hkrati (notranje in zunanje transakcije), transakcije se čakajo, postopke pri verigi blokov počasen.

Izrek CAP

porazdeljen čez večje število računalnikov

Konsistentnost - vsi klienti vidijo enake podatke po posodobitvah
Availabliity: podatki vedno dostopni tudi če omrežje pade
Partioning - sistem še naprej obratuje tudi v primeru težav z omrežjem

Metamask - lahko plačamo z njo.

Solidity

- Solidity je visokonivojski programski jezik za izvajanje pametnih pogodb
- Na Solidity močno vplivajo C++, Python in JS
- Uporablja se na virtualnem stroju Ethereum (EVM)
- Solidity je statilčno tipiziran, podpira dedovanje, knjižnice in kompleksne uporabniško definirane tipe.
- Solidity lahko uporabite za ustvarjanje pogodb za namene, kot so glasovanje množično glasovanje funanciranje, slepe dražbe in denarnice z več podpisi

ABI - način komentiranja argumentov funkcij
ERC - standard za proizvajanje kripto žetonov. Dobiš lahko svoj NFT