Digitaalne maailmapilt, nädal 3 - 2026

Head Eesti Vabariigi 108. sünnipäeva!

Kursuse t**eine** nädal on läbi ja praegu peaks käima kibekiire töö kolmanda nädala materjali õppimise ja testiks valmistumisega.

Praeguseks peaksite olema omandanud esimese sissejuhatuse sellest, et infot mõõdetakse bitiga ja et arvuti “arvutab” matemaatilise loogika reeglitega, mida tänapäeval realiseerivad transistorid jt elemendid integraalskeemide peal. Teise nädala jooksul vaatasime aga juba tänapäevase kõrgtaseme keele Python abil programmeerimise näiteid. Programmeerimises on üsna vähe programmeerimiskeele nüansse, mis on vajalikud alustamiseks. Muutujad, nendele omistamine ja kasutamine, tehted omavahel sobivate andmetüüpide vahel, andmestruktuurid - tabelid, loendid, sõnastikud jne, ja programmi voog - **valikulaused** (if-then-else), **tsüklid**, uute **funktsioonide** tegemise võimalus. Otsused, mida programmiga teha, millises järjekorras, mis teeke kasutada, kuidas oma muutujaid ja funktsioone nimetada - teevad kõik inimesed ise. Head tavad on need, mis teevad programmidest loetavad ka teistele inimestele.

Osad teist pole kahetsusväärselt veel alustanud esimeste testide sooritamisega. Palun kindlasti see nüüd ette võtta, sest muidu tekib arusaamises mahajäämus ja seda pole võimalik liiga kiiresti järele võtta.

Kolmanda nädala jooksul on peamine eesmärk tutvuda **algoritmi** mõistega. Bittidega tehteid tehes täidab arvuti lihtsat sisemist algoritmi (n. kuidas bitid arvudena kokku liita, korrutada vmt). Eelmise nädala loengusalvestuses näitasin ka juba rekursiooni (Fibonacci arvude välja arvutamisel) ja seda, et “naiivsel” rekursiivsel viisil (fib(n)=fib(n-1)+fin(n-2)) muutub see keeruliseks juba üsna väikeste arvude korral. Kuid sama asja teisiti arvutades -> 0,1,1,2,3,5,8… – ei ole see sugugi keeruline. Salvestuses näitasin ka sea, et rekursioonis pole midagi halba, kui funktsiooni väärtusi lihtsalt “meeles peetakse”, selle asemel, et meid uuesti ja uuesti välja arvutada. Vt ka vastavat Google Colaboratory notebook.

Algoritm on (inimkeelne või programmis kirja pandud) eeskiri, mida, mis järjekorras, kuidas teha. Ettekirjutus, kuidas programm realiseerida. Thti tuuakse võrdluseks kokaraamatu retsepte iga konkreetse toidu valmistamiseks - mis koostisosadest, mis järjekorras, kuidas tuleb teha. Osa samme on üldised, nagu näiteks “lisa näpuotsaga soola”, ütlemata kust kapist või poest mis raha eest soola tuleb osta enne. Teised võivad olla üsna täpsed - mis temperatuuril, kui kiiresti-aeglaselt midagi teha, siis jahutada, jne.

Algoritmid võivad olla kõrgel või madalamal tasemel väljendatud, teoreetilisele arvutusmasinale või konkreetses programmeerimiskeeles kirjeldatud; algoritmi pakutud lahendused võivad olla korrektsed või vigased, kiired aga ka aeglased – isegi kui tulemus on tegelikult sama. Arvutiteadus on osalt pühendatud sellele, mida üldse saab piisavalt efektiivselt arvutada.

Selgub, et osa probleeme ongi sellised, millele ei ole tõhusaid algoritme võimalik luua. Teised jällegi peavad massiivsetes süsteemides olema lihvitud võimalikult efektiivseteks.

Ühes loenguvideos seletan neid ideid küpsiste näitel, **kuidas sorteerida küpsised visuaalselt võrreldes šokolaadi nähtava hulga järgi.** Väikese sisendi puhul töötavad algoritmid enamvähem sarnaselt. Kuid ka küpsise sorteerimise näites on algoritmid, mille soorituse aegade erinevus kasvab lõpmatuseni. Väga lihtne on näidata, et suhteliselt väikeste andmetega mis mahub arvutisse erineb sorteerimisalgoritmide kiirus tuhandeid ja kümneid tuhandeid kordi.

Agoritmidest rääkides on kaks peamist küsimust - **kuidas lahendada konkreetne ülesanne** või **probleem** põhimõtteliselt ja teiseks, kui kiiresti ülesanne lahendub. Lihtsalt öeldes mõistetakse algoritmide keerukuse alll küsimust, kui palju elementaarseid tehteid peab arvuti sooritama, ehk kui kaua võtab kogu arvutus aega. Kuigi arvutite kiirus võib varieeruda palju, on tehete loendamine ausam. Näiteks võib 1000 korda suurema andmemahu puhul mõni algoritm olla “1000x aeglasem” - umbes lineaarselt kasvada sisendi kasvuga. Teine algoritm aga vaid 10x aeglasem kahendlogaritm 1000-st), mõni algoritm aga 10,000 korda või hoopis miljon korda aeglasem. Need on üsna tavalised logaritmiline (log n), lineaarne (n), n log n, n^2 keerukused. Kuid münui ülesanne muutub 1000 korda suurema sisendi puhul põhimõtteliselt 2**10000 korda aeglasemaks ehk on eksponentsiaalse keerukusega - sisuliselt ei saa selliseid ülesandeid arvutiga otse naiivselt lahendada. Ja kõik ülesanded ei olegi lahenduvad.

Näiteks esialgu on meie ID-kaardi turvavõtmed veel piisavalt turvalised, kuid kvantarvutitega võivad ka need saada lahti muugitud. Kogu meie ID kaardi turva baseerub sellel, et teadaolevate algoritmidega läheks ründajal tuhandeid aastaid aega, et leida võtmeid muukiv bitikombinatsioon. Ehk selline algoritm on tänaste arvutite jaoks küll teoorias lihtne - katseta läbi kõik variandid - kuid selliselt lihtsalt liiga aeglane. Ohuks on praegu arendatavad kvantarvutid, mis ei täida enam algoritme samm sammult vaid füüsika kvantmehaanika omaduste tõttu “korraga” kõiki bitikombinatsioone läbi vaadates.

Praktikas on vaja, et kõik iga päev töötavad programmid oleks **võimalikult kiired,** see hoiab kokku protsessori aega ja vähendab tarbetut energiakulu. Panoptos on veel üks video kus on sorteerimisalgoritme võrreldud omavahel; eesmärk on veidi demoda seda, millele programmeerijad peavad ka mõtlema ja mis arvutiteaduses on püha graal - milliseid ülesandeid saab kjui kiiresti üldse lahendada. Sorteerimine on kiire ja skaleeruv; kuid enamusele “keerulistele” ülesannetele pole efektiivseid lahendusi üldse teada.

Tervitustega ja edu elamusi soovides!

-Jaak Vilo