1, 2, 3, кодирај ! – Научни осврт – Алгоритми, резоновање и информатичка мисао

Знамо да се алгоритми могу писати с неколико елементарних инструкција које се обједињују помоћу контролне структуре.
Најчеће коришћена елементарна инструкција доделе некој варијабли, на пример ***a = 3*** чије извршење се састоји у изједначавању варијабле с вредношћу 3. Појам доделе некој варијабли захтева разумевање концепта варијабле, тј., меморијског порстора који садржи неку вредност која се може користити и модификовати током извршења неког програма.
На пример, варијабла је број који нам говори колико пута је виђен неки видео на некој платформи. Сваки пут када се видео погледа варијабла « ***број виђења*** » је модификована (на пример, инструкција ***број виђења = број виђења + 1*** омогућује да се варијабли ***број виђења*** додели нова вредност која је већа за јединицу).
Друга елементарна инструкција је приказивање неке поруке на екрану, померање лика, итд.

Ове елементарне инструкције се затим обједињују контролном структуром која омогућује формирање комплексних инструкција обједињавањем једноставнијих инструкција попут:

- секвенца : извршење једне а затим друге инструкције;
- тест који омогућује извршење једне или друге инструкције зависно од услова. На пример, код машине за прање рубља, ако је тежина рубља мања од 2 kg онда се покреће економични програм, ако не онда се покреће класични програм.
- петља, која омогућује да се нека инструкција понови више пута.

Постоје и друге контролне структуре попут програмирања вођеног догађајима (посебно у *Скрачу*), код кога се нека инструкција активира пошто се деси неки догађај.

Први компјутери су били способни да изврше алгоритме састављене из више делова... дакле то су **универзалне машине** које су у стању да реализују сваки алгоритам. Пионири информатике су сигурно били узбуђени када су успели да с универзалним машинама реализују неки алгоритам. Програмери почетници су, попут њих, постепено откривали способности свог компјутера. Почели су прво с једноставним алгоритмима математичких операција (сабирање, множење, итд.), а затим су са све комплекснијим алгоритмима репродуковали математичке низове попут Фибонаћиевог или Бернулијевог. Еjда Лавлејс (Ada Lovelace) је прва успела да напише информатички програм пишући програм за прорачун Бернулијевог низа за Бебиџову(Babbage) аналитичку машину

Постоји низ других програма. Универзална машина нам омогућује да применимо познате алгоритме, али и да налазимо алгоритме за решавање, још увек, нерешених проблема. Међутим, да ли налажење неког алгоритма истовремено и значи налажење неког решења? Да бисмо ово разумели погледајмо шта се подразумева под појмом « информатичка мисао ».

Информатичко мишљење (у вези с овим појмом види чланак на француском језику на сајту Interstices) је скуп знања и ставова који нам помажу да усмеримо наше активности у настојању да разумемо свет око нас. Оно обухвата изузетно широк скуп концепата па ћемо неке покушати и да прикажемо.

Свако од нас, сваког дана, примењује алгоритме а да понекад тога и није свестан. Ми смо установили одређену стратегију, навике, резоновања које примењујемо у нашим свакодневним активностима. Када обављамо куповину, у неком супермаркету, можемо да одлучим да проверимо да ли се у њему налази оно што желимо да купим тако што ћемо погледати све његове рафове. Међутим, можемо претходно да проверимо листу приказа производа која се нуди у том супермаркету и да тиме себи олакшамо посао. При одоласку на посао можемо да избегнемо делове преоптерећених улица у одређене сате. Када тражимо неку реч у речнику покушавамо да отворимо стрницу која почиње првим словом речи, а затим насумичним покушајима настојимо да дођемо до саме речи.

Компјутери могу да нам помогну при реализацији свакодневних задатака! Потребно је само поставити проблем на формалан начин, затим написати алгоритам који се заснива на, напред поменута, четири саставна дела (инструкције, петље, тестови и варијабле) потом омогућити његово решавање. За аутоматско решавање ових алгоритама потребно их је превести у програм и препустити некој машини да изврши дате инструкције. Део информатичког мишљења се састоји у рационализацији овог поступка тако да машина може успешно да га реализује. Погледајмо сад које врсте проблема можемо да решимо захваљујући информатичком мишљењу и алгоритмима. Постоје проблеми које можемо решити алгоритмима инспирисаним човековим резоновањем. Велики број алгортима је сличан оном код начина формирања листе за куповину у неком супермаркету. Ово је слично начину на који објашњавамо неки метод, неком другом, уз настојање да при томе не чинимо никакву грешку. Затим, имамо проблеме које ћемо решити с алгоритмом који је инспирисан људским резоновањем и праксом али уз коришћење изучавања које омогућује примену ригорознијег и општијег приступа. Поменимо пример тражења неке речи у речнику. Постоји вежба која се састоји у тражењу нашег презимена, или имена, на некој листи (на пример, да ли смо успели на неком конкурсу). Покушајмо да формализујемо начин решавања овог задатка.

Замислимо неку неуређену листу од десетак презимена. Сваком је потребно свега неколико секунди да би пронашао своје име (или пак да га сигурно на њој нема). Ако је пак листа већ уређена према азбучном реду онда нам је потребно знатно мање времена да нађем, или не, наше име!
Ова разлика је последица чињенице да користимо знатно ефикаснији метод када је листа уређена него када није. Јер, када је уређена тражимо њен део са почетним словом нашег презимена, а затим слова које га следи, итд. Код неуређене листе тражимо наусмично наше презиме или пак идемо систематски редом преко целе листе.

Овај приступ је функционалан али се може и усавршити. Јер, ако је број објеката врло велики (попут речника или телефонског именика с милионима почетака) можемо да почнемо обухватом више делова на самом почетку или на крају и тиме уштедимо у времену. Постоји врло ефикасан алгоритам претраге уређене листе познат као « **дихотомија** » (чији назив потиче од грчких речи « дељење на два једнака дела ». Да би оно што тражимо нашли на уређеној листи делимо је на два дела, и ако је оно што тражимо у тој половини претрага је завршена, ако није онда другу половину листе опет делим на два дела, и тако све док не завршимо претрагу. У свакој наредној етапи претраге је листа два пута мања.

Листу од 20 имена је могуће претражити, у најгорем случају, са 4 дељења. Број операција у најгорем случају се добија множењем броја онолико пута с колико смо могли да делимо 20 са 2 пре него што стигнемо до 1 (прво дељење заокружујемо ако је број поља непаран). Без дихотомије би (у најгорем случају) требало 20 сукцесивних итерација да би се прочтало свако име, од првог до последњег, да би се уверили да ту нема нашег имена. Моћ дихотомије се јасно види код знатно дуже листе. На пример, код листе са 300 000 имена, применом дихотомије претражимо целу листу са само 19 операција. Замислимо да свака операција читања табеле захтева милисекунд на компјутеру. Следећа табела даје процену најдужег времена за примену два алгоритма:

Број имена на листи	300	3 000	30 000	300 000	3 000 000
Алгоритам 1: читање од првог до последњег имена на листи	0,3 s	3 s	30 s	300 s	3 000 s
Алгоритам 2: дихотомија	0,008 s	0,012 s	0,015 s	0,018 s	0,022 s

Поређење најдужег времена потребног да се нађе име на некој уређеној листи коришћењем два различита алгоритма. Најједноставнији алгоритам (читање листе од почетка до краја) захтева 10 000 пута више времена ако је величина листе увећана за 10 000 пута (од 3 десетине скеунде до 1 часа), док код примене дихотомије за исти случај потребно нека стотинка секунде више.

Бројни су приступи за прављење неког алгоритма. Овде нисмо у могућности да поменемо све, али ћемо се зато позабавити случајем када можемо да тачно напишемо неки алгоритам за који никад нисмо у могућности да знамо резултат, односно нећемо добити решење у неком прихватљивом времену. Овде је у питању један од основних аспеката алгортимике. Размотримо зато « проблем трговачког путника » који смо описали у циклусу 3 (час III-2.5). Показали смо да између 2 града он има једну најкраћу путању.Између 3 града има 6 могућих путања, а између 4 града 24 могуће путање. За 20 градова има већ милијарде путева а компјутеру би требало да их све тестира више милијарди година да би нашао најбоље решење! Бројни су проблеми овог типа (путања поштара, камиона за смеће, чистача улица од снега...) код којих је потребно наћи који редом ће се обилазити улице? Проблем истог типа је и код робота који треба да поставе шрафове на различитим деловима плоче, односно којим редом то да раде да би остварили економичну потрошњу енергије и да имају дужи животни век? Са растом броја « објеката » енормно се повећава број могућности. Реч је о експоненционалном расту или комбинаторној експлозији па самим тим и о непознатом решењу оваквог проблема!

Можемо помислити да је решење проблема могуће наћи коришћењем савршенијих компјутера или ефикаснијег програмског језика. Коришћењем програма у неком новом савршенијем језику проблем би се могао решити десет до сто пута брже. Међутим, ако је време прорачуна процењено на 10 милијарди година онда је то побољшање занемарљиво. Замислимо да можемо да нађемо такву информатичку архитектуру, и језик програмирања, да смо у могућности да нађемо егзактно решење путање између 20 градова за један минут. Међутим, када додамо само један град требаће нам 21 минут за решење проблема, а када се дода још један град онда би суперкомпјутер морао да ради 7 часова, а трећи град би повећао време прорачуна на више од 7 дана. Дакле, машине нам овде врло мало могу помоћи.

Ситуација ипак није безнадежна јер постоји више метода којим можемо на задовољавајући начин решити проблеме овог типа. Предлаже се налажење решења које нам изгледа задовољавајуће, а које затим побољшавамо у више наврата, и кад закључимо да његовим даљим побољшавањем не уочавамо неки бољитак, онда то нађено решење прихватамо као не тако лош резултат. Овај проступ у информатици, који даје задовољавајући али не и оптимални резултат, је познат као **хеуристични.** Препознавање проблема за које нисмо у могућности да прорачун реализујемо у прихватљивом времену је изузетно важан аспект алгортимике. Ако неки програм траје неочекивано дуго онда то не мора бити последица неког бага или лошег познавања програмског језика. Значи, алгоритам је тачан, програм такође, али време потребно за његово извршење је сувише дуго.

У сваком случају, програмерима би требало да експериментишу и побољшавају функционалност својих омиљених програмских језика, а важан је и технолошки напредак. Међутим, напредак у области алгоритмике је неопходан као услов за боље разумевање природе проблема и начина на који желимо да га решимо.

Као илустрацију можемо узети тражење најкраћег пута између два града (на пример, Крагујевца и Београда) коришћењем Гугла. Предложени пут није и најбоље могући, а да би се он добио потребно је милијарду година прорачуна. Зато се применом хеуристике налази компромис између квалитета пута и времена потребног за његово налажење прорачуном.