МАТЕМАТИЧКА ГИМНАЗИЈА

МАТУРСКИ РАД

из предмета рачунарство, информатика

Колмогорова комплексност и проблем халтовања

Ученик

Андреј Милисављевић, IV_C

Ментор

Нина Алимпић

Београд, мај 2020.

САДРЖАЈ

0. Основни појмови

Објекат – Било какав уређен скуп података, резултат рада програма, најчешће низ карактера.

Низ – Уређена листа карактера, низ карактера.

Машина – Рачунар.

Тјурингова машина – Теоретски модел рачунара, са меморијом и временом рада које се може бесконачно продужавати.

Машина са коначним бројем стања (finite state machine) – Рачунар чија се меморија не може бесконачно продужавати (за разлику од Тјурингове машине), најчешће се односи на рачунаре у реалном свету.

Халтовање – Коначни прекид извршавања програма.

Језик – Најчешће се односи на програмски језик.

Минимални опис [објекта] – Најкраћи програм, у одабраном програмском језику, који производи жељени објекат.

Колмогорова комплексност/Колмогорова сложеност [објекта] – Целобројна вредност, која представља број карактера минималног описа објекта.

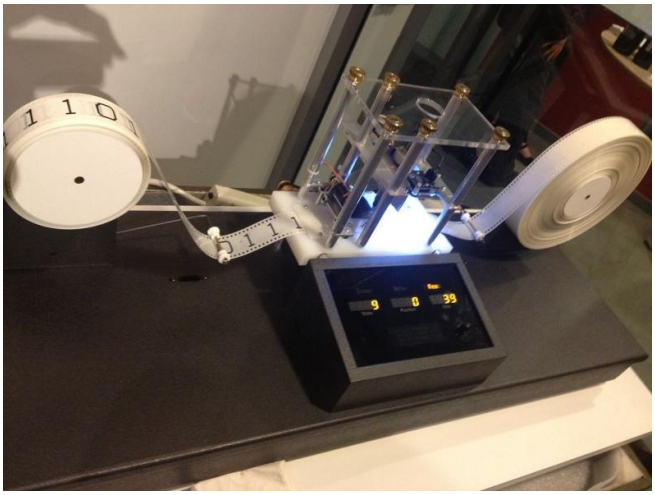
***K* –** Скраћено од Колмогорова комплексност.

Епистемологија – Теорија сазнања, област филозофије која истражује крајње домете људског сазнања.

1. Проблем халтовања

1.0 Тјурингове машине

1936. године Алан Тјуринг је дефинисао појам “аутоматске машине“, којим је испитивао теоретске границе способности рачунара. Овај једноставан модел је касније био назван Тјурингова машина. Механичком „главом“ она пише и чита симболе са бесконачно дугачке траке, на основу унапред дефинисане таблице са правилима. На траку се пре покретања могу уписати симболи, који би представљали неки програм. Она увек има и посебан услов који ће је халтовати, тј. зауставити њен рад. Битно је напоменути и да оваква машина не прима никакве улазне податке, осим оних који су пре покретања већ били записани на траци, тако да заиста ради потпуно аутоматски. Овакав модел се показао јако корисним, и често је заступљен у теорији везаној за рачунарство и информатику. Тјуринг није засновао свој модел на рачунарима, већ је као инспирацију користио начин на који људи „рачунају“. Можемо рећи да његов модел јесте сличан рачунару, али и особи која поседује папир, оловку, гумицу, и задатак. [I] Овај модел би се простије могао описати и као „рачунар чија се радна меморија може бесконачно продужавати“.



Слика 1. [II] Макета са неким елементима Тјурингове машине. Током рада машина може симболе са „исписаног“ дела траке брисати и мењати другима.

1.1 Поставка проблема халтовања

Дата нам је Тјурингова машина, са познатим скупом правила, и неки програм чији изворни код и улазни подаци су нам исто познати.

- Да ли ће програм, након покретања на машини, у неком тренутку изазвати да она халтује, или ће пак машина са овим програмом бесконачно дуго радити?

Испоставило се да се одговор на ово питање не може наћи.

1.2 Решивост проблема халтовања на Тјуринговим машинама

Разматрајући Тјурингове машине, проблем одређивања да ли ће се неки програм завршити или не, не може се решити на било какав начин, чак и ако су нам позната сва својства машине и изворни код тог програма. Могуће је изазвати контрадикцију у било каквој функцији којом бисмо проверавали да ли ће доћи до халтовања, што лако можемо показати.

Идеја иза Тјуринговог доказа се може применити само на Тјурингове машине. Већина програмских језика су по могућностима[1] теоретски еквивалентни овим машинама, то јест „Тјуринг-потпуни“. Простије речено, у многим језицима могуће је написати модел Тјурингове машине. Због овога следећи доказ можемо да разматрамо и у *Python* програмском језику[2]:

Замислимо да смо написали следећи програм **p**, и желимо да видимо да ли ће халтовати. Дата нам је функција haltuje(), која наводно проверава да ли ће програм халтовати, са 100% тачности. Речено нам је да она увек враћа „тачно“ ако неки програм халтује, и „нетачно“ ако се тај програм наставља заувек.

Могуће је написати следећи програм **p**:

```
def p():                    # Дефинишемо функцију:
    if haltuje(p):          # ако цео овај програм халтује...
        while true:         # ...уђи у бесконачну петљу
            pass            # празан корак, петља се понавља заувек
```

p ће проверити да ли **p** халтује и, ако халтује, ући ће у бесконачну петљу. У супротном, ако не халтује, програм ће се завршити.
Програм ће „халтовати само ако не халтује“, и обрнуто, што значи да наша функција није прецизна, јер може да изазове овакву врсту грешке.

Ово је главна идеја иза Тјуринговог доказа нерешивости проблема халтовања, који је знатно сложенији. Иако овај део доказа можда делује „намештено“, постоји непознат број програма који би се понашали слично овом наведеном програму, и изазвали исту контрадикцију, тако да проблем халтовања ипак остаје нерешив.

[1] Занемаривши меморијска и временска ограничења специфичних машина, што многи програмски језици и раде.
[2] *Python* је одабран због тога што је програмски језик високог нивоа, који избегава експлицитно декларисање променљивих, као и писање кода специфичног за машину на којој се ради, те је јако читљив и флексибилан.

1.3 Решивост проблема халтовања на реалним машинама

Битно је напоменути да је проблем халтовања за неке програме у теорији могуће решити, ако разматрамо машине са коначним бројем стања, јер ће се програми написани за такву машину сигурно завршити након неког времена, или ће се машина која их покреће наћи у ранијем стању, што би значило да програм чини бесконачну петљу[III].

Рецимо, ако направимо машину са коначним бројем стања **A**, која ће да покреће наш програм, могли бисмо да „већом“ (али исто коначном) машином **B** бележимо сва њена стања, и поредимо их, док не откријемо да се машина већ нашла у два идентична стања, што би значило да је програм бесконачна петља, или док стања машине потпуно не престану да се мењају, што би значило да је програм халтовао. У најгорем случају, машина **A** ће проћи кроз сва могућа стања пре халтовања или понављања целог процеса. Колико ће овај процес трајати зависи само од брзине ових машина, и броја стања машине **A**.

Свакако, због огромног броја стања у којем се рачунари могу наћи (2^n, где је n број битова који су доступни рачунару), решавање овог проблема трајало би **предуго за било какву примену**[IV]. На пример, рачунар са само десет бајтова укупне меморије имаће 2^{80} могућих стања[1], док модерни рачунари могу имати трилионе бајтова радне меморије[2]. Додатно, такав „мали“ рачунар, ако га је могуће конструисати, спречиће нас да унесемо програме дуже од 10 бајтова, те нам он вероватно неће бити нарочито користан. Због овог строгог временског и просторног ограничења, проблем халтовања на машинама са коначним бројем стања остаје решив само у теорији, или за тривијалне програме.

[1] 1208925819614629174706176 могућих стања.

[2] У многе сервере који се тренутно користе може се додати преко терабајт радне меморије, што значи да такви сервери имају преко $2^{1000000000000}$ могућих стања

1.4 Закључак

Једно интуитивно објашњење тога што можемо„решити“ проблем халтовања на реалним машинама може се наћи у чињеници да не разматрамо све могуће програме. Радом на машини са коначним бројем стања ми занемарујемо све оне програме који би својим радом заузели више меморије него што је доступно рачунару.

Ограничен број стања неке машине **A** нам у теорији омогућава да направимо машину **B** којом ћемо анализирати сва могућа стања машине **A**, и, уз довољно времена и меморије, наћи одговор на наше питање „да ли програм који смо покренули халтује или не“. На Тјуринговим машинама ово није могуће, зато што њихова меморија није коначна, те се не може на овај начин анализирати.

Како се (повећањем меморије рачунара) будемо приближавали бесконачности, то ћемо више различитих програма моћи успешно да покренемо, и то ће теже бити одговорити на питање да ли ће они халтовати или не. На машини на којој можемо покренути било какав програм, без обзира на његову величину или структуру, проблем халтовања нећемо моћи да решимо. [1]

2. Колмогорова комплексност

2.0 Преглед

Колмогорова комплексност је кључан концепт у алгоритамској теорији информација, подобласти рачунарства и информатике која се бави везом између *информације* и *прорачуна*, као и тражењем различитих начина да се опише комплексност, тј. сложеност, произвољних скупова података. Велике напретке у овом пољу направио је Совјетски математичар Андреј Колмогоров (1942-1987).
У свом раду „О табелама насумичних бројева“ [V] Колмогоров дефинише сложеност скупа података као дужину најкраћег компјутерског програма, написаног у унапред одређеном програмском језику, који ће при покретању генерисати тај тражени скуп.

Најкраћи програм који производи тражене податке је њихов *минимални опис*
Дужина, тачније број карактера минималног описа неких података, је њихова *Колмогорова комплексност*. Она зависи само од објекта који описујемо, и од програмског језика у којем се опис налази.

[1] Овакав уређај би била Тјурингова машина, за коју је проблем доказано нерешив.

2.1 Илустрација 1: Запис

Ова илустрација служи да истакне битан концепт везан за Колмогорову комплексност: предвиђање сложености. Узмимо следећи „произвољан" низ:

```
BCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwx
```

55 карактера

Описаћемо овај низ програмски.
У *Python* језику се позивамо на print() функцију када желимо да произведемо текст, те би очигледно решење био овакав програм, који ће исписати жељени низ:

```
print('BCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwx')
```

64 карактера

Због привидне сложености низа, можда бисмо претпоставили да је овај наведени програм његов минимални опис, и закључили да је Колмогорова комплексност датог низа, у *Python* језику, наведена дужина од 64 карактера.
Међутим, ово није тачно. Програмским језицима најчешће можемо произвести резултат на више различитих начина, а у овом случају га можемо добити значајно краћим програмом. Ако смо упознати са распоредом карактера у *ASCII* табели, можда ћемо приметити да се дати низ поклапа са елементима табеле од 66. до 120. [VI], и написати краћи програм, који се на ову табелу позива *chr()* функцијом *Python* програмског језика:

```
for i in range(55):
     print(str(chr(66+i)),end='')
```

54 карактера

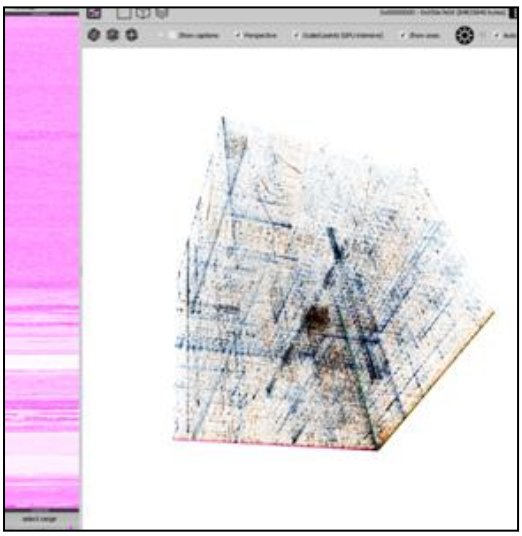
Ово решење речима можемо описати као:
„Испиши низове од по једног карактера, са индексима од 66+0 до 66+55, без размака“

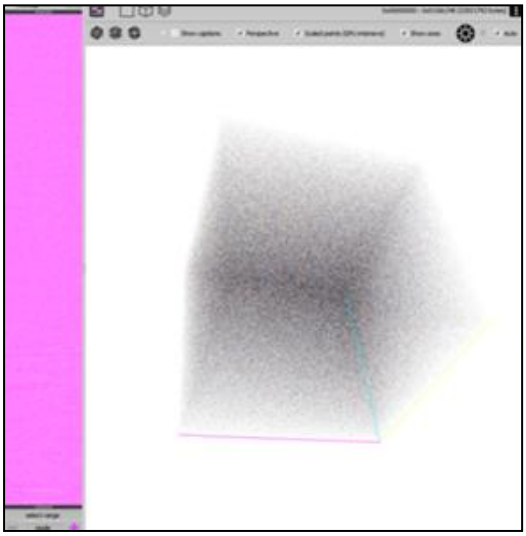
Наведени Python програм је читавих 10 карактера краћи од претходног, и при покретању ће генерисати потпуно исти резултат. Ипак, не можемо сигурно рећи да је ово тражени „минимални опис“, јер је могуће да постоји краћи приступ проблему који није био уочен, или бољи начин да се овакво решење запише.

2.2 Илустрација 2: Компресија података

Налажење начина да се подаци запишу на што ефикаснији начин јесте једна од основних идеја компресије података[VII]. Можемо приметити да ће успешно компресовани[1] фајлови имати значајно теже уочљиве обрасце од оригинала, јер смањењем понављања, по дефиницији, увек добијамо мање предвидиве податке.

У овој мање формалној илустрацији компресоваћемо погодан фајл[2] од 80.9MB, користећи RAR алгоритам за компресију[3], и сами упоредити резултате са почетним фајлом, употребом open-source алата за статистичку анализу и визуелизацију података, VELES[VIII].





Слике 2, 3. VELES визуелизација blender.exe-а (лево), и компресована варијанта тог фајла (десно).

Погледом на леву слику уочавамо одређене обрасце у просторном статистичком приказу података (у средини оба прозора), као и у линијском очитавању фајла (са леве стране оба прозора).
На слици компресованог фајла обрасци нису уочљиви.
Величина компресоване варијанте је само 21.8MB, те је она 73% мања од оригинала, и самим тиме ближа најкраћем могућем запису.
Ова појава слична је илустрацији из претходног поглавља, где смо проблем исписивања низа решили користећи мање очигледан, али и значајно краћи програм.
Битно је напоменути да је сам алгоритам за компресију и читање компресованих фајлова значајно мањи од простора који је сачуван овом применом. Читава Windows инсталација WinRAR-а, који је искоришћен за компресију података у овој илустрацији, мања је од 4 мегабајта, те заузима само ~7% простора који је уштеђен самом компресијом овог једног фајла.

[1] Фајлове чија дужина је редукована компресијом сматрамо успешно компресованим.
[2] blender.exe, фајл програма за дигитално моделовање Blender 2.8, искоришћен јер у себи садржи многе правилности, те се највероватније може значајно компресовати.
[3] Подешен на најефикаснију компресију по питању дужине фајла („Compression method: best”).

2.3 Напомена: Укупан меморијски простор

Сваки програмски језик или алгоритам за компресију који се на неком рачунару користи захтеваће присуство одређених података у меморији, који омогућавају употребу тог језика или алгоритма. Заузимање тог простора може учинити компресију неког малог објекта у циљу уштеде простора узалудном, јер програм који је неопходан да се компресовани фајлови прочитају може бити већи од простора који је уштеђен таквом компресијом. Рецимо, ако бисмо компресовали фајл од 50KB на 40KB, програмом који има 10KB, не би уштедели простор. Овај проблем није био превише важан када смо причали о томе колико простора је сачувано компресијом, јер је простор који језици заузимају константан, и најчешће мали.

2.4 Алгоритамско одређивање *K*

„Можемо ли одредити колико је нешто насумично?“ је филозофско питање, али и основна идеја иза Колмогорове комплексности - када бисмо знали тачно колико је неки објекат предвидив или непредвидив, лакше бисмо га предвидели.

Нажалост, на ово питање се не може лако дати одговор.
Као и са проблемом халтовања, покушаји алгоритамског одређивања минималних описа ће се свести на контрадикцију (ако разматрамо Тјурингове машине), или трајати предуго да би имали било какву значајну употребу, и односити се само на коначан број програма (ако разматрамо реалне рачунаре).

Постоји више формалних доказа[IX] који нам говоре да је ***K*** немогуће одредити на Тјуринговој машини, ради једноставности изложено је објашњење које разматра приступ „грубом силом“, и своди се на проблем халтовања поменут у претходном поглављу:

- Да бисмо одредили минимални опис од **S**, могли бисмо да генеришемо и покрећемо све могуће програме[1], почевши од најмањег, док не стигнемо до првог програма **који производи S као резултат**.
- Дужина овог програма била би Колмогорова комплексност од **S**.[X]

Проблем са овим процесецом јавља се при чекању да се генерисани програми заврше, то јест да халтују, јер је могуће да се неки од тих програма неће завршити. Можемо да видимо да решивост проблема налажења ***K*** поседује идентична ограничења као и проблем халтовања.

[1] Програми се могу генерисати генерисањем низова, и проверавањем да ли поштују синтаксу одабраног програмског језика.

2.5 Закључак

Објаснили смо да је Колмогорова комплексност својство било којег објекта, апстрактна „доња граница“ дужине програма који тај објекат генерише у унапред одабраном језику. По самој дефиницији минималног описа, ниједан његов корак није сувишан, нити може да се редукује без промене резултата.

Колмогорова комплексност се може сагледати и као покушај да се дефинише укупна количина информација коју неки скуп података *мора* да садржи[XI], тако да је интуитивно јасно да ћемо приближавањем дужини минималног описа наилазити на решења са све мање сувишних корака.

У првој илустрацији уочавамо да се краће решење позива на више функција, и има неколико различитих корака, док дуже решење производи исти резултат на мање динамичан начин. Сличне правилности можемо уочити и у другој илустрацији, а често и при било каквом другом смањењу дужине записа података.

3. „Проблем халтовања“ као епистемолошко ограничење

3.0 Раселов парадокс

Када су Бертранд Расел (1872 - 1970) и Алфред Вајтхед (1861 - 1947) покушали да напишу књигу у којој би читаву математику извели из неколико почетних аксиома (*Principia Mathematica*), наишли су на проблем. Сваки пут када би исправили неку недоследност у њиховим аксиомама, она би се појавила негде другде. Када су објавили свој рад, он је био недовршен, са теоремама које нису могле да се докажу почетним аксиомама, а једна од основних недоследности на коју су наишли названа је Раселов парадокс. Расел је дао једноставан опис његовог проблема следећом питалицом:

„ У једном граду постоји берберин који брије искључиво све оне мушкарце који сами себе не бријy. Питање је, ко брије берберина? “

Ако берберин не брије себе, требало би, јер брије све оне који се не бријy. Са друге стране, берберин брије *само* оне који се не бријy, те ипак не може обријати себе.

Раселов парадокс се односи на скупове, где исти овај проблем настаје када затражимо „скуп свих скупова који не садрже сами себе“, и питамо да ли се он садржи у самом себи или не. „Ако се не садржи, садржи се“, и обрнуто. Закључак који можемо да изведемо је да овај „скуп“ не постоји, тачније да није прави скуп. [XII]

Постоје сличности између описивања скупа елемената и писања програма.
У оба случаја желимо да разматрамо неке елементе, и за њих креирамо (што простији) опис. Ако бисмо да произведемо неки низ података на рачунару, написаћемо програм да то уради за нас. Ако желимо да располажемо елементима који су обједињени по томе што поштују једно заједничко правило, описаћемо скуп неком функцијом.

Поставља се питање: Да ли постоје битне разлике између ових појмова, или је сам „проблем халтовања“ последица једног епистемолошког ограничења, описаног на више различитих начина?

3.1 Геделова теорема о непотпуности

Курт Гедел је 1931. године формализовао опис једног битног ограничења, теоремом[1] која говори следеће:

- Ако је могуће доказати две контрадикторне изјаве из скупа аксиома, онда је тај скуп аксиома **недоследан**.
- Ако постоји теорема коју је немогуће доказати или оспорити употребом скупа аксиома, онда је тај скуп аксиома **непотпун**.
- Сваки формални аксиоматски систем, са коначним бројем аксиома, је или **недоследан**, или **непотпун**. [2]

Ово можемо директно повезати са проблемом халтовања, јер нам је познато да је проблем халтовања:

- Доказано нерешив на Тјуринговим машинама - своди се на **контрадикцију** ако разматрамо потпун скуп програма.
- Теоретски решив искључиво на машинама са коначним бројем стања – одговор се може наћи само ако разматрамо **непотпун скуп програма**.

Можемо закључити да ће решење проблема халтовања увек бити или **недоследно** („халтује само ако не халтује“), или **непотпуно** (проблем ће бити решен само за одређен број програма), што иде у прилог Геделовој тврдњи о **свим аксиоматским системима**.

[1] Геделова теорема о непотпуности

[2] Делови овог поглавља написани су на основу кратког текста *P. Dazingera* о проблему халтовања, Геделовој теореми, и Раселовом парадоксу, који је наведен у референцама.

4. Закључак

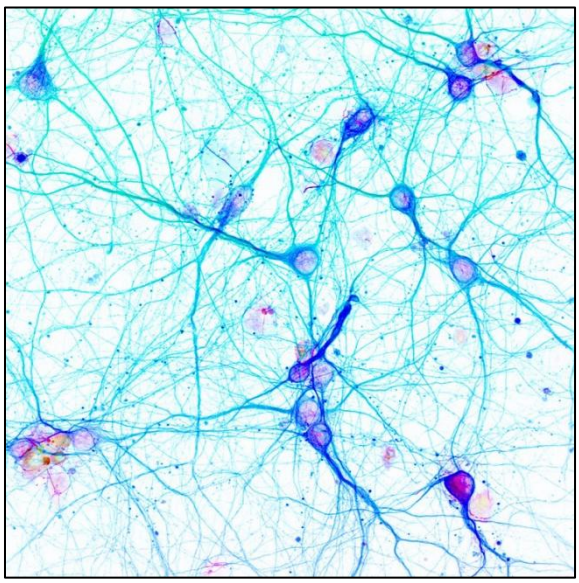
У овом раду био је изложен низ повезаних, апстрактних појмова, са циљем да читаоца упозна са често занемареним концептима у рачунарству, и са ограничењима откривеним у сржи различитих наука, која потенцијално указују на фундаменталне границе самог сазнања.

Објашњење Тјуринговог проблема халтовања служило је да упозна читаоца са битним појмовима у теорији рачунарства, као и да објасни ограничења сваког рачунара. Затим је уведен појам Колмогорове комплексности, са циљем да се пружи бољи поглед на сам концепт информација, као и да се илуструје нераскидива веза сложености са проблемом халтовања.

Видели смо и да су нека необична ограничења била откривена у математици, деценијама пре Тјуринговог истраживања. Затим смо се упознали са вероватно најапстрактнијим појмом наведеним у овом раду, Геделовом теоремом о непотпуности. Тиме је истакнута универзалност ограничења попут проблема халтовања и Раселовог парадокса, иако импликације ове универзалности нису биле посебно разматране.

Свакако, треба имати на уму и то да се мозак скоро сигурно може представити у виду машине са коначним бројем стања, тако да поменута ограничења највероватније представљају и градивни елемент саме људске перцепције.

Неки од изложених појмова већ су нашли битно место у филозофији, али је могуће да ће у будућности утицати и на развој рачунарских наука. Читаоцу се зато оставља питање - да ли ово, релативно недавно откривено и потенцијално свеприсутно ограничење, треба више изучавати, или би труд био узалудан?



Слика 4. Градивни елементи најсложенијег познатог „рачунара“ – неурони људског мозга.[XIII]

5. Литература

[I] A. M Turing “On Computable Numbers, with an Application to the Entscheidungsproblem”

[II] https://commons.wikimedia.org/w/index.php?curid=26270095
Рад Викимедија корисника GabrielF, CC BY-SA 3.0

[III] Minsky “Computation: Finite and Infinite Machines”, strana 24.

[IV] Minsky “Computation: Finite and Infinite Machines”, strana 25.

[V] A.N. Kolmogorov “On tables of random numbers”.

[VI] http://www.asciitable.com/ - Ascii Table - ASCII character codes

[VII] A. Shen, V. A. Uspensky, N. Vereshchagin “Kolmogorov Complexity and Algorithmic Randomness”, strana 1.

[VIII] https://codisec.com/veles/ - Veles - Binary Analysis Tool – CodiSec

[IX] G. J. Chaitin, A. Arslanov, C. Calude “Program-Size Complexity Computes the Halting Problem”

[X] Paul M.B. Vitanyi “How incomputable is Kolmogorov complexity?”, 2. поглавље

[XI] A. Shen, V. A. Uspensky, N. Vereshchagin Kolmogorov Complexity and Algorithmic Randomness, str 45.

[XII] P. Danziger „Russell’s Paradox and the Halting Problem“

[XIII] https://commons.wikimedia.org/wiki/File:Neuronal_web.tif
Рад Викимедија корисника ALol88, CC BY 4.0