Програмирање за први разред специјализованих ИТ одељења

Душа Вуковић
Мијодраг Ђуришић

Petlja

САДРЖАЈ

Садржај 2
Предговор 3
1. ПОЈАМ И ПРИМЕРИ АЛГОРИТАМА 5
Појам и карактеристике алгоритма 5
Начини приказивања алгоритма 9
Дијаграми тока програма 9
Дијаграми MIT Scratch/Blockly 10
Псеудо код 12
Корак 12
Наредба псеудо кода 12
2. ОСНОВНИ КОНЦЕПТИ ПРОГРАМСКИХ ЈЕЗИКА И ОКРУЖЕЊА ЗА РАЗВОЈ
У изради
3. ОСНОВНИ АЛГОРИТМИ ЛИНИЈСКЕ И РАЗГРАНАТЕ СТРУКТУРЕ
У изради
4. ОСНОВНИ АЛГОРИТНИ ЦИКЛИЧНЕ СТРУКТУРЕ
У изради
5. ДЕТАЉАН ПРЕГЛЕД ОСНОВНИХ ТИПОВА ПОДАТАКА
У изради
6. НИЗОВИ
У изради

Напомена за кориснике

У овом тренутку доступно је само прво поглавље приручника „Програмирање за први разред специјализованих ИТ одељења“. У садржају се могу пронаћи наслови наредних поглавља која ће бити сукцесивно објављивана на порталу petlja.org. Такође, постоји могућност да ће садржај и формат овог приручника у будућности бити промењени, тако да препоручујемо да редовно посећујете наш портал како бисте користили актуелну верзију овог материјала.

Тим Фондације Петља
3. септембар 2018.

ПРЕДГОВОР

Наставни предмет Програмирање заузима значајно место у оквиру плана и програма за гимназију за ученике са посебним способностима за рачунарство и информатику. Приручник који је пред вама је припремљен за наставу Програмирања у првом разреду.

Прва година изучавања наставног предмета Програмирање је јако важна зато што представља основу за даље напредовање у овој области, како кроз гимназију, тако и кроз даље школовање и рад. Важно је да се градиво пролази постепено и онолико полако колико је потребно да се на прави начин усвоје основе.

Материјал изложен у Приручнику обезбеђује постизање циља наставног предмета Програмирање који укључује развој алгоритамског приступа решавању проблема, овладавање техникама програмирања и стицање знања о савременим програмским језицима. Концепт Приручника се заснива на званично објављеном начину остваривања програма који уводи иновације у наставу програмирања. Основна идеја је да се знање стиче кроз практични рад на проблемима и да се што пре крене са креирањем апликација. Постепено се, паралелно са практичним радом, усвајају сви потребни појмови и полако се гради дубље разумевање.

Одабрани језик за реализацију програма је савремени програмски језик C#, а предложено развојно окружење за креирање апликација је Visual Studio Community. Програмски језик C# и његово развојно окружење омогућавају креирање конзолних апликација и апликација са графичким корисничким интерфејсом. Приручник покрива основе креирања обе врсте апликација и оставља довољно слободе наставнику да се одлучи да ли ће користи само конзолне апликације, или само апликације са графичким корисничким интерфејсом, или комбиновати обе врсте апликација током реализације наставног програма. Избор врсте апликација које ће се креирати није од суштинског значаја када посматрамо циљ наставног предмета. Није потребно да се превише улази у специфичности једне или друге врсте апликација, већ је суштина да се савлада алгоритамски приступ решавању проблема, а та срж је у обе врсте апликација готово идентична.

Значајан део Приручника чине задаци преузети из наставних материјала под називом „Збирка алгоритамских задатака – основни ниво“ који су објављени на следећој адреси:

https://petlja.org/BubbleBee/r/kursevi/Zbirka

У Приручнику су изложени репрезентативни и важни примери задатака из Збирке, док настава може да се обогати комплетном Збирком која садржи велики број задатака који прате теме наставног предмета Програмирање. Конзолне апликације које су решења задатака из Приручника и Збирке је могуће тестирати у онлајн окружењу портала

Peljta.org. Након што се напише програм који решава одређени задатак, он може да се пошаље на портал на аутоматско оцењивање. На тај начин се у процес учења уносе динамика и забава, што може да допринесе квалитету усвајања знања.

Аутори Приручника се надају да ћете га са уживањем користити.

ПОЈАМ И ПРИМЕРИ АЛГОРИТАМА

Појам и карактеристике алгоритма

Термин алгоритам потиче од имена Арапског математичара Ал-Хорезмија (око 825 године нове ере). Он је извршавање аритметичких операција приказивао у облику упутстава која се састоје од основних корака. Такав начин описивања решења од тада се у математици назива алгоритам. Даљим развојем науке алгоритми су постали један од основних појмова у математици и рачунарству.

Ко жели да зна више?

У 9. веку арапски математичар Абу Абдулах Мухамед ибн Муса ал Хорезми (обично се памти само Ал Хорезми) написао је неколико књига о математици. Једна од њих је говорила о поступцима за решавање једначина и из њеног латинизованог дела наслова „ал џабр" потекла је данашња реч „алгебра". Друга књига је у Европи постала позната под латинским називом Algoritmi de numero indorum, што је требало да значи „Ал Хорезми о индијским бројевима". Међутим, заборавило се да је Алгоритми име аутора и усталио се превод „Поступци рачунања индијским бројевима". Од тада реч алгоритам означава произвољан, обично математички поступак, а одомаћила се и у области рачунарства.

Алгоритам је коначан низ једноставних корака којим се прецизно описује поступак решавања одређеног реалног проблема. Ово није формална дефиниција појма алгоритма јер нисмо дефинисали једноставне кораке нити ниво прецизности описа поступка, али се интуитивно може прихватити.

Многе реалне ситуације се одвијају по тачно одређеним корацима тако да можемо рећи да су алгоритми саставни део свакодневног живота. Постоји релативно прецизан поступак по коме покрећемо аутомобил, телефонирамо, прелазимо улицу, кувамо кафу… Чак можемо рећи да се и сам живот одвија по неким правилима. Добар пример алгоритма је сваки кулинарски рецепт, у њему јасно разликујемо шта је улаз (потребни састојци), шта је обрада (прецизно описан поступак за кување) и шта је излаз (готово јело).

Пример 1:

Посматрајмо поступак телефонирања:

1) Подигни слушалицу

2) Бирај број

3) Разговарај

4) Прекини везу

У описаном поступку можемо уочити низ недостатака. Шта ако немамо одговарајући сигнал приликом подизања слушалице, или ако је

линија заузета после бирања броја? Шта у случају када добијемо везу а позвани корисник се не јавља? То су ситуације које претходним описом поступка нису предвиђене.

Реалне ситуације су по правилу знатно сложеније и да би смо их описали неопходан је прецизнији приступ свим детаљима. У процесу решавања одређених проблема, често се сусрећемо са условима који диктирају даљи ток решавања. Зато је неопходно при изради алгоритма омогућити такозване условне кораке. Неке недостатке уочене у претходном опису поступка телефонирања можемо превазићи на следећи начин:

1) Подигни слушалицу

2) Ако нема одговарајућег сигнала, крај процеса (неуспешно телефонирање)

3) Бирај број

4) Ако се позвани корисник јави, разговарај

5) Прекини везу, крај процеса.

У претходном опису, кораци 2) и 4) су условни кораци. Ако је у кораку 2) услов испуњен, алгоритам се завршава са резултатом „неуспешно телефонирање", а иначе се поступак наставља кораком 3).

Описани поступак отклања неке од наведених недостатака, али и даље остају неразјашњене многе ситуације. У поступку телефонирања уобичајено је да се у случају заузећа линије понавља поступак док се веза не успостави или док не одустанемо од позива. Потреба за понављањем делова процеса врло често постоји у реалним ситуацијама. Зато је неопходно при опису алгоритма користити такозване цикличне кораке. Један од начина како циклични корак можемо уградити у претходни пример је следећи:

1) Подигни слушалицу

2) Ако нема одговарајућег сигнала, крај процеса (неуспешно телефонирање)

3) Бирај број

4) Ако добијеш сигнал заузећа прекини везу и понови корак 3)

5) Ако се позвани корисник јави, разговарај

6) Прекини везу, крај процеса.

У претходном опису поступка телефонирања кораци 3) и 4) чине циклус, и то: корак 3) је тело циклуса (корак који се извршава при сваком проласку кроз циклус), а корак 4) је условни корак којим је одређено трајање циклуса. И овако описаним поступком

телефонирања нису до краја обухваћене све ситуације које се могу јавити при том процесу, али због великог броја тих ситуација, у овом тренутку нећемо се бавити даљом разрадом овог алгоритма.

Пример 2:

У дубљу посуду сипајте 200 г брашна, 2 јаја и 2 дл млека или воде. Тесто мутите док не постане без грудвица. У умућено тесто додајте 1 дл уља и промешајте. Палачинке пеците у тигању, нафилујте их по жељи и уживајте!

Наведени рецепт за палачинке представља пример алгоритма (додуше, није математички и не може да се спроведе на рачунару). У њему се крију кораци који нас воде до решења нашег проблема (а проблем је у овом случају то што смо се ужелели укусних палачинки). Следећи кораци нам говоре како се прави тесто за палачинке:

1) *Узми дубљу посуду.*
2) *Сипај у њу 200 г брашна и 2 јаја.*
3) *Сипај 2 дл млека, а ако немаш млека, сипај 2 дл воде.*
4) *Мути све док смеса не буде без грудвица.*
5) *Додај 1 дл уља.*
6) *Промешај.*

Приметите да је у сваком кораку коришћен императив (узми, сипај, мути, додај, промешај). То није случајно јер су кораци алгоритма обично задати у форми наредби, које се могу препознати по императивној форми. Алгоритам је сачињен од низа наредби које се извршавају истим редоследом којим су наведене. Кажемо да се наредбе нижу једна иза друге и да се извршавају секвенцијално (у низу, једна за другом). Наредбе обично представљају елементарне кораке (у наведеном примеру кораци 1), 2), 5) и 6) заиста су елементарни кораци), али постоје и комплексније наредбе, које описују сложеније кораке (у примеру су такви 3) и 4)).

У кораку 3) понашање зависи од тога да ли је наведени услов испуњен. Услов је питање да ли има млека, и у зависности од тога да ли је одговор на то питање да или не, у смесу ћемо сипати млеко

или воду (подразумева се да воду свакако имамо). Дакле, наредба је облика: ако је услов испуњен, онда уради то и то, у супротном уради то и то, и ток извршавања алгоритма на том месту се грана (једну грану чини случај када је услов гранања испуњен, а другу грану чини случај када услов гранања није испуњен). Зато се каже да је у том кораку присутна наредба гранања.

У кораку 4) радња се понавља све док неки услов не постане испуњен. Овде се понавља мућење смесе и проверава све време да ли и даље постоје грудвице. Тек када грудвице нестану, престаје се са мућењем. Иако се мућење може посматрати као трајна радња, са становишта алгоритмике боље је да се посматра као низ елементарних корака (на пример, окрет варјачом или секунда мућења миксером) који се понављају. Тиме се постиже да се неки део алгоритма циклично понавља све док је неки услов испуњен или све док неки услов не буде испуњен. У таквим ситуацијама се каже да је присутна петља (каже се и циклус, наредба понављања, репетитивна наредба или наредба итерације). Постоје различити облици петљи (обично су то наредбе облика „толико пута уради то и то", „ради то и то све док се не деси то и то" или „док важи то и то, ради то и то").

Приметимо још и да смо корак 4 могли да формулишемо на следећи начин:

4) Мути. Ако у смеси има грудвица, врати се опет на корак 4).

Дакле, уместо петље је могла да се употреби наредба гранања у комбинацији са наредбом облика „иди на корак тај и тај". Пошто на том месту извршавање алгоритма скаче на неко друго место, такву наредбу називамо и наредбом скока. Наредба скока понекад се назива и GOTO наредба. Она није пожељна у описима алгоритама јер доводи до алгоритама и програма које је веома тешко пратити и анализирати. Стога ћемо је избегавати кад год је то могуће.

Претходна два примера описују реалне ситуације из свакодневног живота, које, најчешће, нису строго формално дефинисане.

Развојем рачунарства и програмирања у другој половини 20. века, термин алгоритам се све чешће користи за поступке који се могу аутоматски извршавати, на рачунарима. Да би се алгоритам могао аутоматски извршавати, неопходно је да сваки корак буде прецизно, једнозначно дефинисан.

Такође, пожељно је да алгоритам обезбеди решавање целе групе проблема који се разликују само по улазним величинама. Рецимо, ако је потребно да израчунамо површину правоугаоне ливаде ширине 200 и дужине 300 метара, при креирању алгоритма треба имати у виду да можемо бити у прилици да исти алгоритам користимо и за израчунавање површине произвољне правоугаоне области. У складу са тим, треба обезбедити могућност задавања димензија правоугаоне области како би алгоритам био масовно употребљив.

При изради алгоритма имамо у виду и време потребно за његово извршавање. Рецимо, алгоритам који би бирао потез играча шаха тако да испита све могуће последице потеза, захтевао би милијарде година на најбржем рачунару. Зато се при решавању таквих проблема, прибегава поступцима који не испитују све могућности, али имају велики проценат успешности и дају резултат у „разумном" времену.

Поступци израде алгоритама нису једнозначни иначе би већ постојали генератори алгоритама. Самим тим алгоритмизација проблема захтева и одређену дозу креативности.

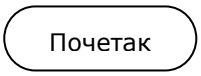
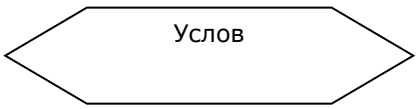
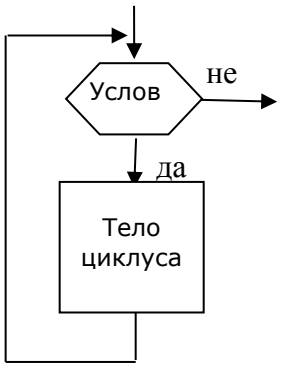
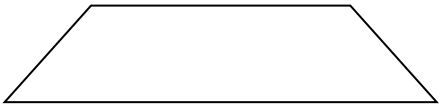
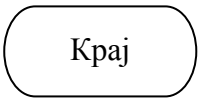
Начини приказивања алгоритма

У пракси алгоритми се на рачунару представљају изворним кодом (енг. *source* - извор) неког вишег програмског језика. Сви програмски језици подлежу строгим правилима писања инструкција и дефинисања података који су за сваки програмски језик различити. Са обзиром да је у почетној фази учења програмирања најважније развити само логику и креативност при алгоритамском решавању проблема, често се за приказ алгоритама користе начини који нису у тој мери оптерећени ограничењима које са собом носи сваки програмски језик.

Дијаграми тока програма

У другој половини 20. века веома популаран начин за опис алгоритама били су такозвани дијаграми тока програма (енг. *control-flow graphs, flowcharts*) тј. дијаграми кода (енг. *code-charts*). Иако се данас не користе као раније, ови дијаграми су важан део историје рачунарства и инспирисали су многе дијаграмске технике које се и данас примењују.

Дијаграми тока програма спадају у блок-дијаграме јер се алгоритам описује дијаграмом који се састоји од појединачних блокова (представљених правоугаоницима, трапезима, елипсама итд.) повезаних стрелицама. Иако дијаграми тока нису јединствено стандардизовани, постојала су нека општа правила како се пишу. Тако се, на пример, трапез оријентисан наниже користио за опис улазних података, трапез оријентисан навише за опис резултата израчунавања, правоугаоници су се користили за опис израчунавања (обично су обухватали математичке формуле које описују како се нове вредности израчунавају на основу постојећих), ромбови су се користили да опишу гранања у програмима (питања на основу чијег се одговора одређује путања којом извршавање алгоритма треба да се настави) итд. Линије означене стрелицама означавају наредни блок који треба да буде извршен. Пример блокова дијаграма и њиховог значења у представљању алгоритма дат је следећом табелом:

Блокови дијаграма	Значење
	**Почетак алгоритма**
	**Учитавање информација**
	**Обрада информација (најчешће израчунавање)**
	**Условни блок**
	**Блок за понављање**
	**Испис информација**
	**Крај алгоритма**

Мана дијаграма тока програма је у томе што омогућавају пренос тока у произвољну тачку програма (то инспирише употребу наредбе скока тј. *GOTO*), чиме настају „шпагети" програми, које је тешко анализирати и који често крију неке грешке.

Дијаграми MIT Scratch/Blockly

У данашње време постоји више окружења за учење програмирања заснованих на блоковском, тј. визуелном, програмском језику. Сва ова окружења имају заједничко то што се алгоритми описују слагањем

блокова (попут лего коцкица) и обично су такви да описују кретање неке фигуре (цртаног јунака) по екрану. Рад у овим окружењима је мотивишући за почетнике у програмирању зато што јако брзо и једноставно могу да се креирају игрице или анимације, тј. да се дође до коначног резултата програмирања. Како је процес програмирања у окружењима овог типа примерен почетницима и упрошћен тако да су неке грешке немогуће, сва концентрација може да се окрене ка алгоритамском начину размишљања у решавању проблема.

Блоковско програмирање се појавило 2006. у виду система за учење *Scratch* развијеног на универзитету MIT у САД. Такође популаран пројекат *An hour of code (http://code.org/)*, који се бави популаризацијом програмирања, користи веома сличан дијаграмски језик. Дијаграмске описе омогућава библиотека *Blockly* компаније Google. Алгоритми се описују слагањем блокова (попут лего коцкица) и обично су такви да описују кретање неке фигуре (цртаног јунака) по екрану. Већи блокови, који обухватају мање, обично се користе за опис понављања корака у мањим блоковима.

Окружење ***Scratch*** је бесплатно и доступно на следећој адреси:

- https://scratch.mit.edu/

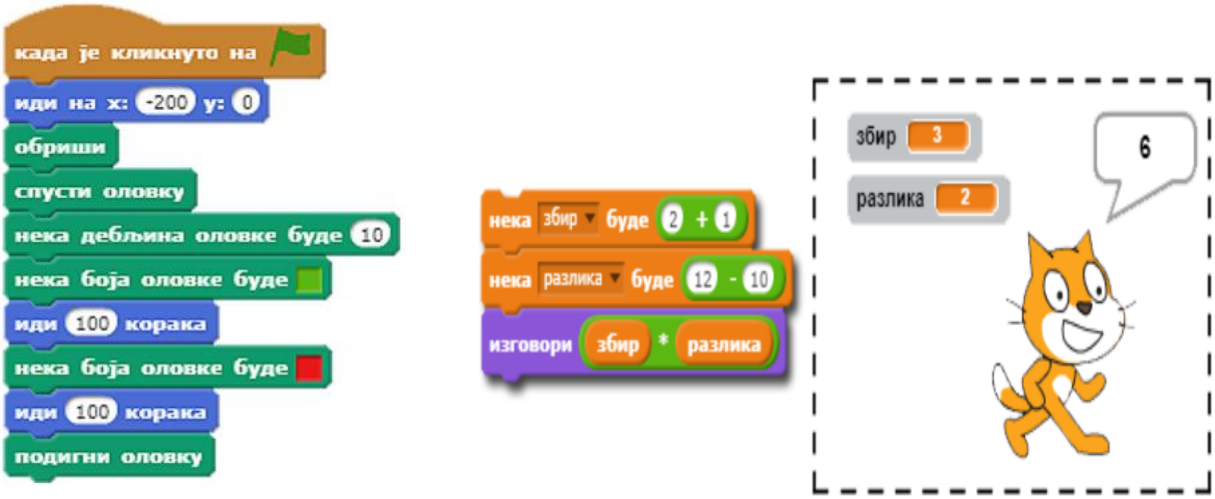
На порталу *Petlja.org* се налази приручник посвећен првим корацима у програмирању употребом окружења *Scratch*. Овај алат се сматра добрим избором за прве кораке у програмирању без обзира на узраст. Како је настава програмирања нова у образовном систему, неко ће пре, а неко касније кренути са учењем програмирања.

- https://petlja.org/BubbleBee/r/kursevi/scratch-support

Приручник прати и збирка задатака са много урађених и објашњених примера.

- https://petlja.org/BubbleBee/r/kursevi/scratch-zbirka

Следећа слика приказује пример једноставних програма креираних у окружењу ***Scratch.***



Псеудо код

У уџбеницима програмирања алгоритми се често представљају у облику псеудо-кода. Такав опис је компромис између флексибилности коју нуди природни језик и прецизности алгоритама које рачунар може да извршава. Псеудо код подлеже одређеним правилима, не тако строгим као у програмским језицима, па омогућава једноставан прелаз на опис алгоритама у произвољним вишим програмским језицима.

На основу претходно изложених примера алгоритама можемо закључити да је за успешан опис алгоритма неопходно обезбедити опис следећих корака:

- почетак
- унос података
- обрада података (корак доделе)
- условни корак
- циклични корак
- испис података
- крај.

У даљем току описа алгоритама користићемо псеудо код који на следећи начин приказује ове кораке:

Корак	*Наредба псеудо кода*
почетак	***Почетак***
унос података	***Читај*** низ улазних података
обрада података (корак доделе)	←
условни корак	*Ако* (Услов)... *иначе*
циклични корак	*Док* је Услов *ради* Тело циклуса
испис података	***Пиши*** низ излазних података
Крај алгоритма	***Крај***

Примери алгоритама линијске структуре

Постоје проблеми које без обзира на вредности улазних података увек решавамо на исти начин, низом корака од којих се сваки извршава тачно једном. То су углавном врло једноставни проблеми у којима се користе тачно утврђена правила обраде података без обзира на спољашње услове. Алгоритми за решавање таквих проблема називају

се алгоритми линијске структуре. У овим алгоритмима користимо само следеће кораке:

- почетак
- унос података
- обрада података
- испис података
- крај.

Први опис поступка телефонирања у Примеру 1 је линијске структуре.

Податке, које обрађујемо у алгоритму, означаваћемо словима енглеске абецеде или речима природног језика које асоцирају на значење податка који обрађујемо. У току обраде, вредност података се може мењати, па ћемо њихове ознаке у даљем тексту звати промењивама, због аналогије са променљивама у програмском језику, којима ћемо се касније бавити.

У псеудо коду, у кораку унос података, наводимо променљиве које уносимо у загради, међусобно одвојене зарезом. На пример, ако уносимо вредности за променљиве x, y и z пишемо наредбу псеудо кода ***Читај***(x,y,z).

У псеудо коду, у кораку испис података, наводимо променљиве које представљају излазне величине или поруке кориснику, међусобно одвојене зарезом. На пример, ако желимо да испишемо вредност променљиве x, можемо употребити наредбу псеудо кода ***Пиши***("Вредност променљиве је ",x).

Корак доделе представља додељивање вредности израза некој променљивој. Писаћемо га на следећи начин: *променљива ← израз.*

Како ћемо у овој фази најчешће радити са бројевним подацима, у изразима ћемо користити следеће аритметичке операције.

Операција	Ознака операције
Сабирање	+
Одузимање	-
Множење	*
Дељење	/
Целобројно дељење	//
Остатак целобројног дељења	%

Кажимо нешто више о операцијама целобројног дељења и остатка целобројног дељења путем примера.

7 // 2 = 3
15 // 3 = 5
6 // 8 = 0

7 % 2 = 1
15 % 3 = 0
6 % 8 = 6

Изразе ћемо градити на уобичајен начин коришћењем променљивих и константи, горе наведених оператора и заграда. Приоритет операција је уобичајен. Пример исправно записаног израза је (12+ x*(a+3))/(b-3)+1.

Корак доделе се извршава тако што се прво израчуна вредност израза а затим се та вредност додељује податку који је означен променљивом. Треба водити рачуна да променљиве које учествују у изразу морају претходно бити постављене на неку вредност, кораком уноса или неком претходном обрадом, да би вредност израза била коректно израчуната. Примери исправно записаног корака доделе су:

- x ← 5
- x ← x+2
- a ← b*3+78

Приметимо да се у другом примеру променљива x појављује како у изразу са десне стране тако и на месту променљиве чијем податку се додељује вредност. Како смо раније нагласили, при извршавању овог корака прво се одређује вредност израза на десној страни, при чему се користи текућа вредност променљиве x, а по извршењу корака доделе податак, представљен променљивом x, добија претходно одређену вредност израза.

Задаци

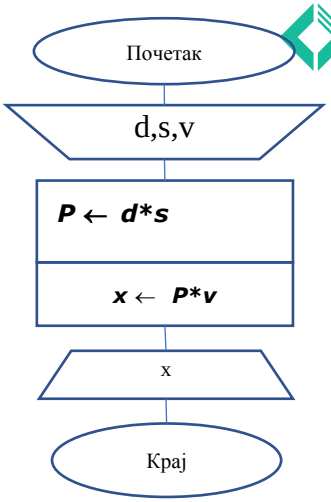
1. *Ливаду правоугаоног облика треба покосити. Описати алгоритам којим се одређује време које је потребно Марку да покоси ливаду ако су дате дужина и ширина ливаде изражене у метрима, и познато је време у минутима за које Марко покоси $1m^2$ ливаде.*

Анализа проблема

Уочимо да су улазне величине дужина, ширина ливаде и време потребно да се покоси $1m^2$ ливаде. Потребно је израчунати време за које Марко може покосити ливаду. Познато је да површину правоугаоника израчунавамо као производ његове дужине и ширине. Да бисмо добили време потребно за кошење целе ливаде довољно је да помножимо површину ливаде са временом које је потребно Марку да покоси $1m^2$.

Разрада алгоритма

За сваки од података које користимо у алгоритму увешћемо по једну променљиву. Нека је дужина ливаде представљена променљивом d, ширина променљивом s, време потребно за $1m^2$ променљивом v, површина променљивом P, а тражено време променљивом x. Променљивој P треба доделити производ променљивих d и s, а затим променљивој x производ променљивих P и v.



Algoritam – prvi način

Početak
Čitaj(***d*, *s*, *v***)
P ← ***d**s**
x ← ***P**v**
Piši("Potrebno vreme je ", ***x*)**
Kraj

Možemo uočiti da smo vrednost promenljive x mogli izračunati direktno, bez pretходног računanja površine livade, odnosno bez uvođenja promenljive P. Na taj način se obezbeđuje uštеda memorijskog prostora, ali se gubi na preglednosti programa.

Algoritam – drugi način

Početak
***Čitaj*(*d*, *s*, *v*)**
x ← ***d**s *v**
Piši("Potrebno vreme je ", ***x*)**
Kraj

2. Описати алгоритам којим се размењују вредности две дате променљиве x и y.

Размена вредности подразумева да прва променљива добије вредност друге, а друга почетну вредност прве. Ако ***x*** има вредност 16, а ***y*** 12, по завршетку размене ***x*** мора имати вредност 12, а ***y*** 16. Приметимо да доделама ***x*** ← ***y*** и ***y*** ← x, губимо почетну вредност променљиве ***x***. После наведених додела вредност променљивих ***x*** и ***y*** је 12. Јасно је да ово није решење нашег проблема. Направимо аналогију са проблемом мењања садржаја две исте шоље ако се у једној налази чај а у другој кафа. У овом случају је јасно да морамо имати празну шољу која би служила да се у њу привремено смести садржај једне од пуних шоља. Ако празну шољу означимо променљивом ***p***, размену можемо извршити низом додела ***p*** ← ***x***, ***x*** ← ***y***, ***y*** ← ***p***. По завршетку ових додела јасно је да су размењени садржаји променљивих ***x*** и ***y***.

Početak
***Čitaj*(*x*, *y*)**
p ← ***x***
x ← ***y***
y ← ***p***
***Piši*(*x*, *y*)**
Kraj

Уколико променљиве ***x*** и ***y*** садрже бројевне вредности размену њихових садржаја, користећи својства математичких операција и бројева, можемо реализовати на следећи начин.

```
Почетак
    Читај(x, y)
    x ← x + y
    y ← x - y
    x ← x - y
    Пиши(x, y)
Крај
```

Морамо имати у виду да се претходни алгоритам може користити искључиво у раду са бројевима, што је значајно ограничење у програмирању па се за размену вредности променљивих препоручује први алгоритам који је универзалан.

3. *Дато је време поласка аутобуса **X1** сати, **Y1** мин и време доласка аутобуса на одредиште **X2** сати и **Y2** мин. Описати алгоритам којим се одређује дужина путовања (број сати и број минута), претпостављајући да се путовање одвија у једном дану. Подразумевамо да је време доласка после времена поласка.*

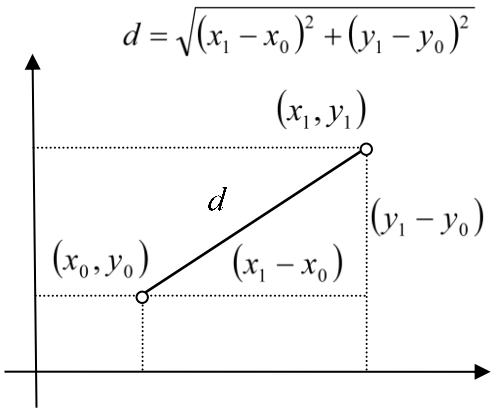
Јасно је да морамо одузети време поласка од времена доласка, али обзиром да је време дато у мешовитим јединицама неопходно је претворити оба времена у нижу јединицу (минуте). По добијању разлике у минутима њу треба довести у тражени облик (сат, минут).

```
Почетак
Читај(X1, Y1,X2, Y2)
Y1 ← X1*60 + Y1
Y2 ← X2*60 + Y2
Y ← Y2 - Y1
X ← Y //60
Y ← Y % 60
Пиши(X, Y)
Крај
```

4. *Планира се изградња трга квадратног облика, који ће имати два супротна темена на локацијама у граду које су представљене координатама (**x0**,**y0**) и (**x1**, **y1**), и који ће бити оивичен улицама које се секу под правим углом. Описати алгоритам којим се одређује површина трга и укупна дужина улица око трга, ако занемаримо ширину улица.*

Пажљивим читањем поставке задатка можемо закључити да се проблем своди на израчунавање површине и обима квадрата коме су познате координате два супротна темена. Дужину дијагонале квадрата

можемо одредити као растојање између датих супротних темена коришћењем Питагорине теореме на следећи начин.



Страницу квадрата рачунамо из познатог односа $d = a\sqrt{2}$.

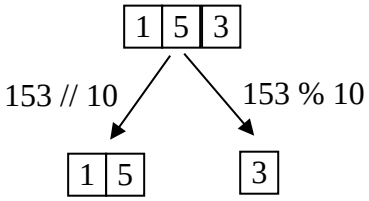
За израчунавање квадратног корена у скоро свим програмским језицима користи се функција ***sqrt***.

```
Почетак
    Читај(x0, y0, x1, y1)
    d ← sqrt((x1-x0)*(x1-x0) + (y1-y0)*(y1-y0))
    a ← d/sqrt(2)
    P ← a*a
    O ← 4*a
    Пиши(P, O)
Крај
```

5. *Описати алгоритам којим се у датом троцифреном броју* ***x*** *одређује сума цифара.*

Коришћењем операција целобројног дељења и остатака целобројног дељења можемо одредити сваку од цифара природног троцифреног броја. Цифру јединица одређујемо као остатак целобројног дељења броја ***x*** са 10. Цифру десетица и стотина можемо одредити следећим изразима:

(***x // 10)%10***	153 // 10 =15	15 %10=5
(***x // 100) %10***	153 // 100 =1	1 % 10= 1




```
Почетак
    Читај(x)
    j ← x % 10
    d ← (x // 10)%10
    s ←x // 100
    suma ←j + d +s
    Пиши("Suma cifara je ", suma)
Крај
```

6. *Описати алгоритам којим у датом природном броју* ***N*** *(N>=100) цифре јединица и стотина размењују вредности.*

```
Почетак
    Читај(N)
    j ← N % 10
    s ← (N // 100)%10
    N ← ( N - j - s *100) + s + j*100
    Пиши("Broj N posle transformacije je ", N)
Крај
```

Израз којим се додељује вредност променљивој N можемо написати и као: *N ← N + (j - s)*99*.

Овај начин омогућава једноставније израчунавање вредности израза, што је понекад у програмирању значајно, али је мање разумљив читаоцу.

Задаци за вежбу:

7. *Описати алгоритам којим се одређује најмања број новчаница којима се може платити артикл вредности* ***x*** *динара, коришћењем апоена од 200, 100, 10 или 1 динара.*
8. *Са* ***V*** *литара бензина пређен је пут од* ***S*** *километара. Описати алгоритам којим се одређује колика је потрошња на 100 km.*
9. *Дата су два угла троугла изражена у степенима и минутима. Описати алгоритам којим се одређује трећи угао тог троугла.*
10. *Авион је полетео у* ***X*** *сати и* ***Y*** *минута, у лету је провео* ***A*** *минута. Описати алгоритам којим се одређује време слетања авиона. Претпоставимо да лет не траје више од 24 часа.*
11. *Описати алгоритам којим се одређује тачка реалне праве која дати интервал (a, b) дели у размери p : q.*
12. *Марко и Милена живе у истој улици и истим путем иду у школу. Они истовремено полазе, а Маркова кућа је удаљенија од школе. Марко иде брзином* ***X*** *m/min, а Милена* ***Y*** *m/min (X>Y), док Марко не сустигне Милену после* ***Z*** *минута. Описати алгоритам којим се утврђује колико је растојање између њихових кућа.*

13. *Описати алгоритам којим се дати петоцифрени природан број **N** трансформише тако што му се цифре кружно помере за **Z** места улево, као у примеру.*

    Пример: 56731 -->31567.

14. *На кружној стази трче Јелена и Биљана. Јелена трчи брзином **X** m/min, а Биљана **Y** m/min (X≠Y). Дужина кружне стазе је **Z** m. Јелена и Биљана крећу са исте позиције у исто време. Описати алгоритам којим се одређује колико је метара прешла Јелена до сустизања са Биљаном.*

15. *Њиву облика правоугаоника димензија A x B власник жели да огради тако што полазећи од једног темена поставља стубове на сваких K метара по ивици њиве или паралелно њој. Описати алгоритам којим се одређује колико стубова власник треба да постави тако да на описан начин огради максималну површину њиве. Који проценат површине њиве није ограђен?*

Примери алгоритама разгранате структуре

Већ смо нагласили да се алгоритмима линијске структуре могу решавати само врло једноставни проблеми у којима се користе тачно утврђена правила обраде података без обзира на спољашње услове. Међутим, реални проблеми најчешће нису тако једноставни, и на начин њиховог решавања утичу различити спољашњи услови. У зависности од тога да ли су одређени услови испуњени или не, бирају се различити начини решавања проблема (алгоритам се грана).

Алгоритме за решавање таквих проблема називамо алгоритми разгранате структуре. За представљање таквих алгоритама неопходно је увести и такозвани условни корак који проверава услов и, у зависности од његове испуњености, бира грану којом се наставља решавање проблема.

Други опис поступка телефонирања у Примеру 1 је разгранате структуре.

Условни корак ћемо писати на следећи начин:

Ако (*Услов*) *B1*
иначе B2

Услов у овом кораку представља логички израз. У почетку ћемо користити само једноставне логичке изразе који заправо представљају релације између два аритметичка израза. За представљање релација користићемо следеће симболе.

Релација	Ознака релације
Једнако	=
Мање	<
Веће	>
Мање или једнако	≤
Веће или једнако	≥
Различито	≠

Примери исправно записаних логичких израза су :

$$a+3>b*12$$

$$b=c*(a+2).$$

Вредност логичког израза ћемо одређивати на следећи начин:

- прво израчунавамо вредности аритметичких израза са леве и десне стране ознаке релације
- затим израчунате вредности поредимо у складу са релацијом; ако је релација испуњена вредност логичког израза је **тачно**, а ако није вредност је **нетачно**.

Услов који наводимо у запису условног алгоритамског корака сматрамо испуњеним уколико је вредност одговарајућег логичког израза **тачно**.

Ознаке B1 и B2 у приказу условног корака представљају низ произвољних алгоритамских корака. Уколико у низу постоји више од једног алгоритамског корака, у псеудо коду тај низ корака ћемо одвојити ознакама {, за почетак низа и }, за крај низа.

Условни корак извршавамо тако што прво одређујемо да ли је Услов испуњен, а затим ако јесте извршавамо низ корака B1, а ако није низ корака B2.

Условни корак којим се одређује већи од бројева представљених променљивим ***a*** и ***b*** можемо написати на следећи начин:

Ако **(*a*>*b*) *већи*** ← ***a***
иначе **већи**← ***b***

Посматрајмо следећи условни корак записан у псеудо коду:

Ако **(x**≥0**)** **y** ← **x**
иначе **y** ← **-*x***

На овај начин променљивој y додељујемо апсолутну вредност променљиве ***x***.

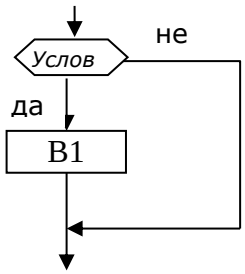
Можемо уочити да променљива ***y*** има исту вредност као променљива ***x*** уколико је ***x≥0***, па се апсолутна вредност променљиве ***x*** може одредити и на следећи начин:

y ← ***x***
Ако ***(x <0) y*** ← ***-x***
Иначе

Уколико услов није испуњен, ако није потребно извршавање ниједног алгоритамског корака, као у претходном примеру, нема потребе за приказивањем гране *иначе* у условном кораку, па се претходни пример може записати:

y ← ***x***
Ако ***(x <0) y*** ← ***-x***

У пракси је честа ситуација да се неки низ алгоритамских корака извршава само ако је одређени услов испуњен а у супротном се прескаче. У таквим ситуацијама користимо условни корак без гране *иначе* на следећи начин:



Ако (*Услов*) *B1*

У низу алгоритамских корака B1 и B2 могу се наћи било који кораци па и условни. Такво гранање називамо угњежденим гранањем. У том случају, грана *иначе* увек припада последњем условном кораку за који није већ дефинисана.

Пример:

Ако **(A)** *Ако* **(B) C1**
иначе **C2**
иначе **C3**

Уколико угњеждени условни корак нема грану *иначе* пишемо:

Ако **(A) {** *Ако* **(B) C1}**
иначе **C3**

Препоручујемо да при представљању алгоритма у псеудо коду грану иначе пишемо увек испод одговарајућег Ако.

Задаци

16. *Описати алгоритам којим се за дате реалне бројеве **a** и **b**, решава једначина ax+b=0. Уколико решење није једнозначно одређено или једначина нема решење исписати одговарајућу поруку.*

Анализа проблема

- Уколико је решење једнозначно одређено његова вредност је ***-b/a***.
- Обзиром да дељење нулом није дозвољено морамо посебно анализирати случај када је ***a=0***:
  - Ако је ***b*** једнако 0 једначина се своди на облик 0****x***+0=0 што је тачно за сваку вредност ***x***.
  - Уколико је ***b*** различито од 0 једначина се своди на облик 0*x+b=0 што није могуће ни за једно ***x*** из скупа реалних бројева.

Разрада алгоритма:

Улазне величине су реални бројеви ***a*** и ***b***. При изради алгоритма треба издвојити сваки од три случаја наведена у анализи проблема коришћењем одговарајућих условних корака. Алгоритам ће имати за сваки од случајева различите излазне вредности: решење једначине је ..., поруку да је сваки реалан број решење једначине и поруку да једначина нема решења.

```
Почетак
   Читај(a, b)
   Ако (a≠0) { x ← - b /a
   Пиши("Rešenje jednačine je ",x)}
   иначе Ако (b=0) Пиши("Svaki realan broj je rešenje jednačine")
   иначе Пиши (" Jednačina nema rešenja")
Крај
```

17. *Описати алгоритам којим се одређује максимум три дата цела броја **a**, **b** и **c**.*

```
Почетак
      Читај(a, b, c)
      Ако (a>b) Ако (a>c) max ← a
            иначе max ← c
      иначе Ако (b>c) max ← b
                  иначе max ← c
      Пиши("Maksimum unetih brojeva je ", max)
Крај
```

Ово решење је интуитивно јасно. Наиме, пореде се две величине а затим се трећа пореди са већом од њих. Међутим ако би желели да одредимо максимум више величина на овај начин, решење би се

компликовало сразмерно броју величина чији максимум тражимо. Зато је са програмерске тачке гледишта знатно прихватљивије следеће решење:

```
Почетак
   Читај(a, b, c)
   max ← a
   Ако (c>max) max ← c
   Пиши("Maksimum unetih brojeva je ", max)
Крај
```

У овом решењу величине посматрамо по реду којим се уносе. Прва величина је природно максимум свих до сада унетих величина, а сваку наредну поредимо са до сада највећом и уколико је већа од ње она је максимална величина. При оваквом приступу број величина које посматрамо није битан јер у сваком тренутку променљива ***max*** има вредност тренутно највеће величине, па се када завршимо унос у њој налази резултат.

18. *Описати алгоритам који на основу датог просека ученика и броја његових недовољних оцена одређује успех ученика по следећем правилу: ако ученик има недовољних оцена његов успех је недовољан, а уколико нема недовољних оцена успех се одређује на основу просечне оцене по познатим правилима.*

```
Почетак
      Читај(brojJedinica, prosek)
      Ако (brojJedinica>0) Пиши("Učenik je nedovoljan")
      inače Ако(Prosek≥4.5) Пиши("Učenik je odličan")
            inače Ако(Prosek≥3.5) Пиши("Učenik je vrlo dobar")
                  inače Ако(Prosek≥2.5) Пиши("Učenik je dobar")
                        inače Пиши("Učenik je dovoljan")
Крај
```

19. *Дата су три реална броја. Описати алгоритам којим се утврђује да ли постоји троугао чије су дужине страница представљене са та три броја, и ако постоји да ли је тај троугао једнакостранични, једнакокраки или разностран.*

```
Почетак
        Читај(a,b,c)
        Ако (a+b>c)
            Ако (a+c>b)
                Ако(b+c>a)
                    Ако(a=b)
                        Ако(b=c) Пиши("Jednakostraničan trougao")
                        иначе Пиши("Jednakokraki trougao")
                    иначе Ако(b=c) Пиши("Jednakokraki trougao")
                            иначе Ако(a=c) Пиши("Jednakokraki trougao")
                                    иначе Пиши("Raznostrani trougao")
                иначе Пиши("Ne čine trougao")
            иначе Пиши("Ne čine trougao")
        иначе Пиши("Ne čine trougao")
Крај
```

Посматрајући решење овог задатка лако можемо уочити да се неке истоветне команде понављају више пута у различитим ситуацијама. Ова понављања се могу избећи обједињавањем више условних корака у један, креирањем сложених логичких израза, уз употребу логичких операција коњункције (&&) и дисјункције (||). Вредност коњункције два логичка израза је **тачно** само ако су вредности оба логичка израза **тачно**. Вредност дисјункције два логичка израза је **нетачно** само ако су вредности оба логичка израза **нетачно**. Употребом логичких операција претходни алгоритам може се поједноставити на следећи начин:

```
Почетак
        Читај(a, b, c)
        Ако( (a+b>c) && (a+c>b) && (b+c>a))
                Ако(a=b)
                        Ако(b=c) Пиши("Jednakostraničan trougao")
                        иначе Пиши("Jednakokraki trougao")
                иначе Ако ((b=c)||(a=c))Пиши("Jednakokraki trougao")
                        иначе Пиши("Raznostrani trougao")
        иначе Пиши("Ne čine trougao")
Крај
```

20. *Описати алгоритам којим се вредности променљивих **x**, **y** и **z** размењују тако да је **x**<=**y**<=**z**.*

Алгоритам размене вредности две променљиве је познат из претходног поглавља. Потребно је само одредити редослед и услове под којима ће се вршити размена вредности променљивих. Вредности

променљивих ћемо размењивати ако није задовољен тражени поредак. Неопходно је извршити поређење вредности сваке две променљиве и у случају потребе разменити те вредности. Било би добро да при реализацији алгоритма водимо рачуна о редоследу извршавања поређења ради прегледности.

```
Почетак
    Читај(x, y, z)
    Ако (x>y) { p ← x
                x ← y
                y ← p  }
    Ако (x>z) { p ← x
                x ← z
                z ← p  }
    Ако (y>z) { p ← y
                y ← z
                z ← p  }
    Пиши(x, y, z)
Крај
```

21. *Мика, Пера и Лаза треба да се састану. Први који дође чека највише **x** минута другог, а онда заједно још највише **y** минута трећег. Дата су времена доласка изражена у сатима и минутима Мике, Пере и Лазе, редом (времена нису дата по редоследу доласка). Описати алгоритам којим се проверава да ли су се сва три друга састала.*

Потребно је уредити времена доласка у неопадајући редослед, а затим анализирати размак између тих времена у односу на величине ***x*** и ***y***.

```
Почетак
    Читај(S1,M1,S2,M2,S3,M3,X,Y)
    M1 ← S1*60+M1
    M2 ← S2*60+M2
    M3 ← S3*60+M3
    Ако (M1>M2) { p ← M1
                  M1 ← M2
                  M2 ← p  }
    Ако (M1>M3) { p ← M1
                  M1 ← M3
                  M3 ← p  }
    Ако (M2>M3) { p ← M2
                  M2 ← M3
                  M3 ← p  }
    Ако ((M2-M1≤X)&&(M3-M2≤Y)) Пиши("Sastali su se ")
    иначе Пиши("Nisu se sastali ")
Крај
```

22. *Дата су два интервала реалне праве (**a**,**b**) и (**c**,**d**). Описати алгоритам којим се одређује*

    а) најмањи интервал реалне праве који садржи дате интервале

б) највећи интервал реалне праве који је садржан у оба интервала

ц) дужина дела реалне праве који је покривен са та два интервала

д) дужина дела реалне праве који припада и једном и другом интервалу.

Приказаћемо решење проблема б) и д).

Обележимо тражени интервал са (***x***,***y***). Да би интервал (***x***,***y***) био садржан у оба дата интервала (***a***,***b***) и (***c***,***d***), ***x*** и ***y*** морају задовољавати следеће услов: ***x***≥***a***, ***x***≥***c***, ***y***≤***b*** и ***y***≤***d***. Како тражимо највећи заједнички интервал датих интервала, ***x*** мора бити једнако већем од бројева ***a*** и ***c***, а ***y*** мањем од бројева ***b*** и ***d***. Уколико је ***x***≤***y,*** (***x***,***y***) ће представљати највећи заједнички интервал, а у супротном он не постоји. Када заједнички интервал постоји, његову дужину је лако одредити одузимањем ***y***-***x*** и она представља решење проблема д), а у супротном решење проблема д) је 0.

```
Почетак
    Читај(a,b,c,d)
        Ако (a>c) x ← a
        иначе x ← c
        Ако (b<d) y ← b
        иначе y ← d
        Ако (x≤y) Пиши("Traženi interval je (",x ,",", y,")")
        иначе Пиши("Traženi interval ne postoji")

d)
        Ако (x≤y) d ← x-y
        иначе d ← 0
        Пиши("Tražena dužina je ",d,".")
Крај
```

23. Правоугаоник чије су странице паралелне координатним осама задат је координатама темена једне његове дијагонале. Описати алгоритам којим се проверава да ли дата тачка припада правоугаонику.

Како у поставци задатка није прецизирано која темена су приказана датим координатама, ако би анализирали све могућности решење би било сложено. Зато ћемо на основу координата темена дијагонале одредити координате доњег левог (***x1***,***y1***) и горњег десног (***x2***,***y2***) темена правоугаоника, а затим проверити да ли се координате дате тачке налазе између одговарајућих координата та два темена.

```
Почетак
    Читај(x1, y1, x2, y2, x, y)
    Ако (x1>x2) { p ← x1
                  x1 ← x2
                  x2 ← p  }
    Ако (y1>y2) { p ← y1
                  y1 ← y2
                  y2 ← p  }
    Ако ((x1≤x)&&(x≤x2)&&(y1≤y)&&(y≤y2))
                  Пиши("Тачка је у правougaoniku")
    иначе Пиши("Тачка није у правougaoniku")
Крај
```

24. Описати алгоритам којим се одређује која цифра се налази на К-тој (1<K<189) позицији записа 123456789101112...9899 где су редом исписани природни бројеви од 1 до 99.

```
Почетак
    Читај(K)
    Ако (K<10) Пиши(K)
    иначе {
            K ← K-9
            M ← K //20+1
            K ← K-(M-1)*20
           Ако (K % 2 = 1) Пиши(M)
              иначе {
                      M ← K //2-1
                      Пиши(M)
                    }
          }
Крај
```

Задаци за вежбу:

25. *Описати алгоритам којим се у датом природном броју **N** (N<1000) замењују прва и последња цифра.*

26. *Описати алгоритам којим се проверава да ли дата тачка **A**(X_A,Y_A) припада кругу чији је центар дата тачка **O**(X_O,Y_O) и полупречник **r**.*

27. *Описати алгоритам којим се проверава да ли дате тачке A(X_A,Y_A) и B(X_B,Y_B) припадају истом квадранту.*

28. *Сваки пакетић треба да садржи 2 чоколаде, 1 играчку и 4 кесице бомбона. Ако имамо на располагању **A** чоколада, **B** играчки и **C** кесица бомбона, описати алгоритам којим се одређује колико највише пакетића можемо направити.*

*Од **A** чоколада можемо направити највише **P**=**A**//2 пакетића, од **B** играчки највише **Q**=**B** пакетића и од **C** кесица бомбона највише **R**=**C**//4*

*пакетића. Највећи број пакетића који се могу направити представља минимум бројева **P**, **Q**, **R**. Проблем одређивања минимума можемо реализовати слично као у задатку 16.*

29. *Описати алгоритам којим се одређује ком квадранту припада дата тачка A(**x**,**y**) при чему је x*y≠0.*

30. *Описати алгоритам којим се на основу броја поена ученика остварених на писменом задатку одређује његова оцена по следећој табели:*

поени	*оцена*
90..100	*5*
75..89	*4*
60..74	*3*
45..59	*2*
0..44	*1*

31. *Описати алгоритам којим се од датог троцифреног природног броја **N** формира највећи могући број **X** који може настати разменом места цифара у броју **N**.*

    Овај задатак се своди на решавање два већ решена проблема:

    - *издвајање цифара троцифреног прородног броја*
    - *уређење издвојених цифара*

32. *Описати алгоритам којим се проверава да ли кутија облика квадра димензија **a1**, **b1** и **c1** може да се смести у кутију облика квадра димензија **a2**, **b2** и **c2** тако да су им ивице паралелне.*

33. *У 2 кутије се налазе само црвене и беле куглице. У првој кутији се налази **A**-црвених и **B**-белих, а у другој **X**-црвених и **Y**-белих. Описати алгоритам којим се одређује најмањи број премештања куглица тако да после премештања у свакој кутији буду куглице исте боје. Једно премештање је пребацивање једне куглице.*

34. *Правоугаоник чије су странице паралелене координатним осама може се задати координатама темена једне његове дијагонале. Описати алгоритам којим се за два дата правоугаоника проверава да ли се секу и ако се секу одређује површина њиховог пресека.*

35. *Описати алгоритам којим се утврђује да ли је троугао датих дужина страница правоугли, оштроугли или тупоугли.*

36. *Три аутомобила крећу са стартне позиције у тренуцима **T1<T2<T3** константним брзинама **V**1, **V**2 и **V**3. Описати алгоритам којим се одређују стартни бројеви аутомобила који су на водећој позицији у тренутку **T** (**T>T3**).*

37. *Написати програм који за дати месец одређује број дана у том месецу за дату годину.*

37. *Написати програм којим се за дати датум задат помоћу дана, месеца и године одређује датум следећег дана (претходног дана).*