СТУДЕНТСКА УНИЈА МАТЕМАТИЧАРА

Финално такмичење
Б категорија
HS Algorithm 2024

шифра

Број бодова по задацима

1.		6.	
2.		7.	
3.		8.	
4.		9.	
5.		10.	

Укупан број поена:

СТУДЕНТСКА УНИЈА МАТЕМАТИЧАРА

Финално такмичење
HS Algorithm 2024

Упутство за израду задатака

Драги такмичари,

Испред вас се налазе задаци за финални део такмичења HS Algorithm. Тест се састоји од 10 задатака при чему је 5 задатака из математике, док је 5 задатака из информатике. У сваком задатку имаћете прилике да се упознате са темама које се изучавају на нашем факултету у оквиру основних академских студија на програмима Математика и Информатика.

Последњи задатак из сваке области је класичног типа - задатак решавате на папиру који касније предајете и **потребно је приказати детаљан поступак** да бисте добили све поене. У неким задацима ће заокруживање нетачног одговора на питање типа вишеструког избора донети негативне поене и то четвртину поена који су предвиђени за тај подзадатак (погледати табелу). Уколико питање оставите незаокружено, то се вреднује са 0 поена. Целокупан рад који на крају предајете мора бити исписан **хемијском оловком**, док слике можете цртати графитном оловком.

р.бр	**област**	**број поена**	**негативни поени**
1	Логика	9	нема
2	Линеарна алгебра	9	нема
3	Аналитичка геометрија	9	нема
4	Вероватноћа	9	има
5	Математичка анализа	14	нема
6	Дигитални запис података	9	има
7	Машинско учење	9	има
8	Алгоритми и структуре података	9	има
9	Лексичка анализа	9	нема
10	Вештачка интелигенција	14	нема

Време за израду задатака је 180 минута. Такмичарска комисија ће два пута, након 60 и након 120 минута, проћи кроз све учионице и одговарати на ваша питања у вези поставки задатака.

СРЕЋНО!

СТУДЕНТСКА УНИЈА МАТЕМАТИЧАРА

Финално такмичење
Б категорија
HS Algorithm 2024

шифра

Задатак 1. Математичка логика

Математичка логика је једна од кључних грана математике, која се бави формалним системима закључивања и основама математике. Док се традиционалне области математике фокусирају на проучавање бројева, облика или функција, математичка логика истражује принципе на којима почивају математички докази и формалне структуре.

Математичка логика има широку примену у информатици и вештачкој интелигенцији. Она је основа развоја програмских језика, алгоритама и теорије аутомата. У области вештачке интелигенције, математичка логика омогућава формализацију знања, креирање система закључивања и развој алгоритама за доказивање теорема. Њена примена је нарочито значајна у стварању експертских система и верификацији софтвера.

Овај задатак се састоји од два независна дела. Поступак се не оцењује. Негативне поене на овом задатку не можете да добијете.

Квантификатори су један од главних алата математичке логике, јер омогућавају формално изражавање универзалних ($\forall$) и егзистенцијалних ($\exists$) тврдњи, као и специфичних тврдњи попут постојања тачно једног елемента ($\exists!$) који задовољава одређени услов.

Функција f из скупа A у скуп B ($f : A \to B$) је свака релација између елемената скупа A и скупа B ($f \subseteq A \times B$) таква да $(\forall a \in A)\ (\exists! b \in B)$ за које је $(a, b) \in f$ (ово пишемо $b = f(a)$).

За $f : A \to B$ кажемо да је **инјективна** („1-1") ако $(\forall a_1, a_2 \in A)\ (a_1 \neq a_2 \implies f(a_1) \neq f(a_2))$.

За $f : A \to B$ кажемо да је **сурјективна** („на") ако $(\forall b \in B)\ (\exists a \in A)$ тако да је $f(a) = b$.

За $f : A \to B$ кажемо да је **бијекција** ако је „1-1" и „на".

Ако је дата функција $f : X \to Y$ и $A \subseteq X$, тада под **директном сликом** скупа A, у ознаци $f[A]$, подразумевамо

$$f[A] = \{f(a) \mid a \in A\}.$$

Ако је дата функција $f : X \to Y$ и $B \subseteq Y$, тада под **инверзном сликом** скупа B, у ознаци $f^{-1}[B]$, подразумевамо

$$f^{-1}[B] = \{x \in X \mid f(x) \in B\}.$$

Слободнијим речима, директна слика скупа A је скуп свих вредности које функција f додељује елементима скупа A, док је инверзна слика скупа B скуп свих елемената из домена функције који се пресликавају у елементе скупа B.

Напоменимо да је инверзна слика скупа дефинисана за сваку функцију, а не само за бијекције.

Ваш задатак је да разматрајући функцију $f : \mathbb{R} \to \mathbb{R}$ дату са $f(x) = x^2$ одредите:

(а) [1 поен] $f[\ [-2, 1)\ ] =$

(б) [1 поен] $f^{-1}[\ (-3, 2025]\ ] =$

У наредна три подзадатка треба у сваком правоугаонику да заокружите тачно један симбол. У сваком правоугаонику треба да заокружите онај симбол који ће представљати оно што је „најбоље" што можете да закључите о односу два скупа (нпр. „бољи" и „јачи" је закључак да важи једнакост два скупа од $\subseteq$ или $\supseteq$).

Нека је $f : X \to Y$ произвољна функција.

(в) [1.5 поен] Који односи важе без додатних претпоставки о функцији f?

$f[A_1 \cup A_2] \boxed{\subseteq = \supseteq} f[A_1] \cup f[A_2]$; $f[A_1 \cap A_2] \boxed{\subseteq = \supseteq} f[A_1] \cap f[A_2]$; $A \boxed{\subseteq = \supseteq} f^{-1}[f[A]]$; $B \boxed{\subseteq = \supseteq} f[f^{-1}[B]]$.

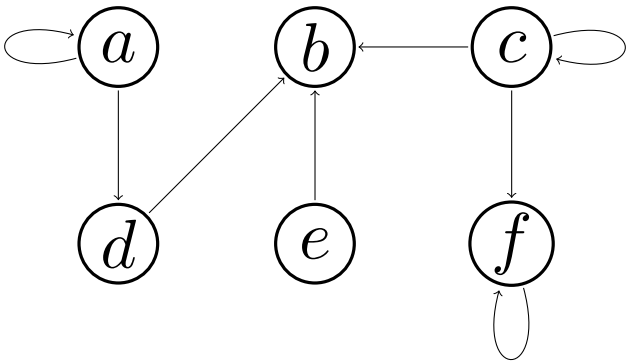
(г) [1.5 поен] Који односи важе уз претпоставку да је f инјективна функција („1-1")?

$f[A_1 \cup A_2] \boxed{\subseteq = \supseteq} f[A_1] \cup f[A_2]$; $f[A_1 \cap A_2] \boxed{\subseteq = \supseteq} f[A_1] \cap f[A_2]$; $A \boxed{\subseteq = \supseteq} f^{-1}[f[A]]$; $B \boxed{\subseteq = \supseteq} f[f^{-1}[B]]$.

(д) [1.5 поен] Који односи важе уз претпоставку да је f сурјективна функција („на")?

$f[A_1 \cup A_2] \boxed{\subseteq = \supseteq} f[A_1] \cup f[A_2]$; $f[A_1 \cap A_2] \boxed{\subseteq = \supseteq} f[A_1] \cap f[A_2]$; $A \boxed{\subseteq = \supseteq} f^{-1}[f[A]]$; $B \boxed{\subseteq = \supseteq} f[f^{-1}[B]]$.

Нека $q(x, y)$ означава $x \to y$. На слици је дат универзум дискурса (то је скуп свих објеката који се разматрају у оквиру датог проблема).



[2.5 поена] Који елементи x задовољавају формулу

$$(\forall y)\big(q(x, y) \Rightarrow \neg q(y, y)\big) \ ?$$

Помоћ. $(\forall y)$ подразумева и $y = x$.

Задатак 2. Линеарна алгебра

Линеарна алгебра бави се проучавањем вектора, векторских простора, линеарних трансформација и система линеарних једначина. Налази широку примену у различитим областима науке и технологије. Оне су основа за решавање система линеарних једначина, што је кључно у моделирању и анализи у инжењерским, физичким и економским проблемима. Такође, матрице се користе у компјутерској графици за трансформације слика, у машинском учењу за обраду података, као и у квантној механици за опис стања система.

Матрица је правоугаона табела бројева са m редова и n колона. На пример:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

је матрица димензија 3×3.

Линеарне операције над матрицама

Нека су дате матрице A и B димензија $m \times n$, где је

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}, B = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{pmatrix}$$

Тада је

$$\alpha A + \beta B = \begin{pmatrix} \alpha a_{11} + \beta b_{11} & \alpha a_{12} + \beta b_{12} & \cdots & \alpha a_{1n} + \beta b_{1n} \\ \alpha a_{21} + \beta b_{21} & \alpha a_{22} + \beta b_{22} & \cdots & \alpha a_{2n} + \beta b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha a_{m1} + \beta b_{m1} & \alpha a_{m2} + \beta b_{m2} & \cdots & \alpha a_{mn} + \beta b_{mn} \end{pmatrix}$$

Множење матрица

Нека је A матрица димензија $m \times n$, а B матрица димензија $n \times p$.

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}, B = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{pmatrix}$$

тада је производ матрица A и B матрица $C = AB$ димензија $m \times p$:

$$C = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mp} \end{pmatrix}$$

таква да важи: $c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj} = \sum_{k=1}^{n} a_{ik}b_{kj}$, за све $i = 1, ..., m$ и $j = 1, ..., p$. Другим речима, вредност у i-тој врсти и j-тој колони матрице C добијамо скаларним множењем i-те врсте матрице A и j-те

врсте матрице B. Наведимо пример множења матрице димензије 2×4 матрицом димензије 4×3.

$$\begin{pmatrix} \mathbf{2} & \mathbf{-3} & \mathbf{0} & \mathbf{5} \\ 1 & -1 & 2 & 4 \end{pmatrix} \begin{pmatrix} \mathbf{3} & 3 & -1 \\ \mathbf{-2} & -4 & 0 \\ \mathbf{-1} & -3 & -1 \\ \mathbf{2} & -5 & 10 \end{pmatrix} = \begin{pmatrix} \mathbf{22} & -7 & 48 \\ 11 & -19 & 37 \end{pmatrix}$$

У првој врсти и првој колони добијене матрице се налази број 22 зато што је $22 = 2 \cdot 3 + (-3) \cdot (-2) + 0 \cdot (-1) + 5 \cdot 2$.

Јединична матрица

Јединична матрица је квадратна матрица која има јединице на главној дијагонали (од горњег левог до доњег десног угла), док су сви остали елементи нула. Означава се са I_n, где n означава димензију матрице.
За матрицу I_n димензија $n \times n$, важи:

$$I_n = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

Множење са јединичном матрицом: За сваку матрицу A димензија $n \times n$, важи $A \cdot I_n = I_n \cdot A = A$.

Инверз матрице

Инверз матрице је матрица која, када се помножи са оригиналном матрицом, даје јединичну матрицу. У овој секцији ћемо дефинисати инверз матрице, навести услове под којима постоји, и приказати начин његовог израчунавања.

Дефиниција инверза матрице

Нека је A квадратна матрица димензија $n \times n$. Матрица A^{-1} се назива инверз матрица матрице A ако важи:

$$A \cdot A^{-1} = A^{-1} \cdot A = I_n,$$

где је I_n јединична матрица димензија $n \times n$, тј. матрица у којој су сви елементи ван главне дијагонале нула, а сви елементи на главној дијагонали су једнаки 1.

Услови за постојање инверза

Ако матрица A има инверз, кажемо да је *инвертибилна* и тада морају бити испуњен следећи услови:
- Матрица A мора бити квадратна (тј. број редова мора бити једнак броју колона).
- Детерминанта матрице A мора бити различита од нуле, тј. $\det(A) \neq 0$. Ако је $\det(A) = 0$, матрица нема инверз. У каснијем тексту ћемо дефинисати детерминанту.

Рачунање инверза матрице Гаусовим операцијама

Рачунање инверза матрице A помоћу Гаусових операција захтева извођење трансформација на редовима матрице A како би је претворили у јединичну матрицу, уз истовремено извођење истих операција на јединичној матрици како бисмо добили инверзну матрицу A^{-1}.

Претпоставимо да је дата матрица A и да желимо да израчунамо њен инверз. Кораци су следећи:
1. Почињемо са проширеном матрицом $[A|I]$, где је A матрица коју желимо да инвертујемо, а I је јединична матрица исте димензије као A.
2. Примењујемо операције на редовима на проширену матрицу тако да лева страна постане јединична матрица. Десна страна на крају овог поступка ће бити A^{-1}.

Пример. Размотримо матрицу $A = \begin{pmatrix} 4 & 7 \\ 2 & 6 \end{pmatrix}$.
Прво проширимо матрицу A са јединичном матрицом I:

$$[A \mid I] = \left(\begin{array}{cc|cc} 4 & 7 & 1 & 0 \\ 2 & 6 & 0 & 1 \end{array}\right)$$

Корак 1: Правимо први елемент у првом реду да буде једнак 1. Поделимо први ред са 4:

$$\left(\begin{array}{cc|cc} 1 & \frac{7}{4} & \frac{1}{4} & 0 \\ 2 & 6 & 0 & 1 \end{array}\right)$$

Корак 2: Правимо елемент испод прве 1 у првом реду да буде једнак 0. Одузимамо 2 пута први ред од другог реда:

$$\left(\begin{array}{cc|cc} 1 & \frac{7}{4} & \frac{1}{4} & 0 \\ 0 & \frac{5}{2} & -\frac{1}{2} & 1 \end{array}\right)$$

Корак 3: Правимо други елемент на дијагонали да буде једнак 1. Поделићемо други ред са $\frac{5}{2}$:

$$\left(\begin{array}{cc|cc} 1 & \frac{7}{4} & \frac{1}{4} & 0 \\ 0 & 1 & -\frac{1}{5} & \frac{2}{5} \end{array}\right)$$

Корак 4: Правимо елемент изнад друге 1 да буде једнак 0. Одузимамо $\frac{7}{4}$ пута други ред од првог реда:

$$\left(\begin{array}{cc|cc} 1 & 0 & \frac{3}{5} & -\frac{7}{10} \\ 0 & 1 & -\frac{1}{5} & \frac{2}{5} \end{array}\right)$$

Лева страна је сада јединична матрица, а десна страна је инверзна матрица A^{-1}.
Дакле, инверз матрице A је:

$$A^{-1} = \begin{pmatrix} \frac{3}{5} & -\frac{7}{10} \\ -\frac{1}{5} & \frac{2}{5} \end{pmatrix}.$$

Сопствене вредности и сопствени вектори

За дату матрицу A димензија $n \times n$, сопствена вредност λ и сопствени вектор v задовољавају следећу једначину: $A \cdot v = \lambda \cdot v$, где је $v \neq 0$. У овом контексту, сопствени вектор v представља се као колона матрица:

$$v = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix}.$$

Како се налазе?

Сопствене вредности се налазе решавањем карактеристичне једначине:

$$\det(A - \lambda I) = 0,$$

где је I јединична матрица исте димензије као A.

Сопствени вектори се добијају решавањем једначине $A \cdot v = \lambda \cdot v$ (или њој еквивалентне $(A - \lambda \cdot I_n)v = 0$) за сваку сопствену вредност λ.

Конкретно, ако је $n = 3$ означимо да је за дату сопствену вредност λ одговарајући сопствени вектор $v = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$ и посматрамо једначину $A \cdot v = \lambda \cdot v$ као систем једначина по x, y и z. За различите λ добијамо различите системе једначина. Добићемо решења тог система једначина у параметарском облику, тј. да треба да важи нпр. $x = 0$, $y = 2z$. Свако решење овог система је сопствени вектор за дату сопствену вредност λ, али за рачун нама потребан у овом задатку тада за сопствени вектор v можемо узети било који представник решења тог система, нпр. $v = (0, 2, 1)$. Разлог зашто можемо узети било који представник је зато што се сва решења овог система могу приказати у облику $z(0, 2, 1)$.

Линеарна независност вектора

Скуп вектора $\{v_1, v_2, \ldots, v_k\}$ у векторском простору је линеарно независан ако ниједан вектор није могуће представити као линеарну комбинацију осталих вектора. Ово се може формално записати као:

$$c_1 v_1 + c_2 v_2 + \cdots + c_k v_k = 0 \Leftrightarrow c_1 = c_2 = \cdots = c_k = 0.$$

На пример:

- Линеарно независни вектори у $\mathbb{R}^2$:
$$v_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad v_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$
- Линеарно зависни вектори у $\mathbb{R}^2$:
$$v_1 = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \quad v_2 = \begin{pmatrix} 2 \\ 4 \end{pmatrix}.$$

Детерминанта матрице

Детерминанта квадратне матрице A димензија $n \times n$ је број који одређује својства матрице. За $n = 3$, детерминанта је:

$$\det(A) = a_{11}(a_{22}a_{33} - a_{23}a_{32}) - a_{12}(a_{21}a_{33} - a_{23}a_{31}) + a_{13}(a_{21}a_{32} - a_{22}a_{31}).$$

За општи случај:

$$\det(A) = \sum_{j=1}^{n} (-1)^{1+j} a_{1j} \det(M_{1,j}),$$

где је $M_{1,j}$ минор матрице A, односно матрица добијена уклањањем првог реда и j-те колоне.
Такође, још једно својство детерминанте неке матрице, јесте да се детерминанта матрице A не мења уколико узмемо неку колону, помножимо је неким реалним бројем, и додамо другој колони. Исто важи и за трансформацију која се састоји из узимања неког реда, множења било којим реалним бројем и додавањем другом реду.
Приметимо да је детерминанта јединичне матрице једнака 1.

Визуелни пример минора

Ако је:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix},$$

тада је $M_{1,2}$:

$$M_{1,2} = \begin{pmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{pmatrix}.$$

Коши-Бинеова теорема

Нека су A и B матрице димензије $n \times n$. Тада је

$$\det(AB) = \det(A) \cdot \det(B).$$

Последица: $1 = \det(I_n) = \det(P \cdot P^{-1})$.

Дијагонална матрица

Матрица се назива *дијагоналном матрицом* ако је квадратна и сви њени елементи ван главне дијагонале су једнаки нули. То значи да важи:

$$D = \begin{pmatrix} d_{1,1} & 0 & 0 & \cdots & 0 \\ 0 & d_{2,2} & 0 & \cdots & 0 \\ 0 & 0 & d_{3,3} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & d_{n,n} \end{pmatrix}.$$

Тада је $\det(D) = d_{1,1} \cdot d_{2,2} \cdots d_{n,n}$.

Дијагонализација матрице

Дијагонализација матрице A подразумева налажење дијагоналне матрице D и инвертибилне матрице P такве да важи:

$$A = PDP^{-1}.$$

Поступак дијагонализације
Следећи кораци воде до дијагонализације матрице A:
1. **Израчунавање сопствених вредности матрице** A: Пронађите сопствене вредности матрице A решавањем карактеристичне једначине:

$$\det(A - \lambda I) = 0.$$

Овде је I јединична матрица исте димензије као A.
2. **Провера дијагонализабилности**:
НАПОМЕНА (За решавање овог задатка, можете прескочити овај корак) Проверите да ли је свака сопствена вредност недегенерисана, тј. да има довољно сопствених вектора да попуни димензију простора. Ако нека сопствена вредност нема довољно сопствених вектора, матрица није дијагонализабилна.
3. **Налазак сопствених вектора:** За сваку сопствену вредност λ, нађите што више линеарно независних сопствених вектора решавањем система:

$$(A - \lambda I)v = 0.$$

4. **Креирање матрице** P: Формирајте матрицу P чије су колоне линеарно независни сопствени вектори матрице A.
Пример: Претпоставимо да су сопствени вектори матрице A следећи:

$$v_1 = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, \quad v_2 = \begin{pmatrix} -1 \\ -1 \\ 1 \end{pmatrix}, \quad v_3 = \begin{pmatrix} 5 \\ -4 \\ 1 \end{pmatrix}.$$

Матрицу P формирамо као матрицу чије су колоне сопствени вектори матрице A:

$$P = \begin{pmatrix} 1 & -1 & 5 \\ 2 & -1 & -4 \\ 3 & 1 & 1 \end{pmatrix}.$$

5. **Формирање дијагоналне матрице** D: Направите матрицу D тако што ћете на главну дијагоналу уписати сопствене вредности матрице A. Битно је којим редоследом стављате сопствене вредности. Ако гледамо пример горе, и на пример нек нам је за v_1 сопствена вредност једнака 1, за v_2 једнака 3 и за v_3 једнака -2. Онда ако смо матрицу P добили тако сто смо поређали векторе v_1, v_2 и v_3 тим редом, онда ћемо да на дијагоналу редом ставимо одговарајуће сопствене вредности, па ће D да нам изгледа овако

$$D = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & -2 \end{pmatrix}.$$

6. **Финални резултат**: Након претходних корака, дијагонализација је завршена, и важи:

$$A = PDP^{-1}.$$

Напомена: Да би матрица A била дијагонализабилна, мора имати n линеарно независних сопствених вектора, где је n димензија матрице. Ова напомена је ту само ради комплетности.

Потребно је да коначна решења упишете у за то предвиђене правоугаонике. Поступак се не оцењује. Негативне поене на овом задатку не можете да добијете.

Дата је матрица B димензија 3×3:

$$B = \begin{pmatrix} 13 & 1 & -11 \\ 15 & 5 & -17 \\ 6 & 1 & -4 \end{pmatrix}$$

(а) [**2 + 2 поена**] Пронаћи сопствене вредности и сопствене векторе матрице B.

(б) [**2 поена**] Израчунати детерминанту матрице B.

(в) [**3 поена**] Израчунати B^{2024}.

СТУДЕНТСКА УНИЈА МАТЕМАТИЧАРА

Финално такмичење
Б категорија
HS Algorithm 2024

шифра

Задатак 3. Аналитичка геометрија

Једна од важних грана аналитичке геометрије јесте **сферна геометрија**, која проучава облике на сфери. Док је класична геометрија, попут Еуклидове, заснована на равним линијама и равним површинама, сферна геометрија се бави фигурама на површини сфере.
Најзначајнија разлика је у томе што у сферној геометрији најкраћи пут између две тачке није права, већ најкраћа крива на сфери која повезује те две тачке. Многа тврђења из класичне геометрије не важе у сферној геометрији - на пример, косинусна теорема за троугао.
Сферна геометрија има значајну примену у астрономији и географији, јер се планета Земља може моделовати као сфера. Ова област геометрије омогућава да апроксимирамо дужине рута авионских летова. У астрономији, израчунавање растојања између звезда и њихових кретања по небу заснива се на примени сферне геометрије.

Уколико посматрамо раван и сферу, постоји неколико могућности за њихов међусобни положај. У крајње неинтересантном случају раван ће промашити сферу. У супротном, раван и сфера имају заједничких тачака, где је прва могућност да се ради о само једној заједничкој тачки и тада је у питању тангентна раван на сферу. Преостаје могућност када се раван и сфера секу по кругу. Лако је приметити да је пресечни круг највећи када раван пролази кроз центар сфере. Такав круг са сфере, чији се центар поклапа са центром сфере, зовемо **велики круг**. Географски примери су меридијани који су половине великог круга, док су све паралеле мали кругови, осим екватора који јесте велики круг. Велики кругови добијају на значају када схватимо да се најкраће растојање између две тачке на сфери реализује баш по луку великог круга који их спаја.

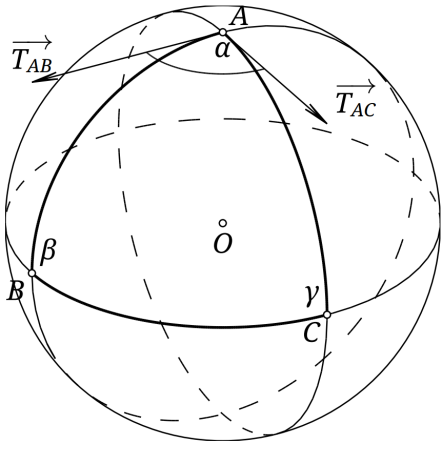
Посматрамо најједноставнију сферу у тродимензионом простору, сферу полупречника 1, са центром у координатном почетку:

$$\mathbb{S}^2 = \{(x_1, x_2, x_3) \ : \ x_1^2 + x_2^2 + x_3^2 = 1\}.$$

Сваки тачку са сфере $X \in \mathbb{S}^2$ можемо поистоветити са њеним вектором положаја у односу на центар сфере и уједно координатни почетак O, што је јединични вектор $\overrightarrow{OX}$. **Сферни троугао** ABC представљају тачке $A, B, C \in \mathbb{S}^2$, при чему захтевамо да су њихови вектори положаја, $\overrightarrow{OA}, \overrightarrow{OB}$ и $\overrightarrow{OC}$ линеарно независни.
Разлог због којег уводимо овај додатни услов линеарне независности је да избегнемо двосмислености. Наиме, уколико су вектори $\overrightarrow{OA}, \overrightarrow{OB}$ и $\overrightarrow{OC}$ линеарно зависни, то су тачке O, A, B, C копланарне (припадају истој равни), одакле следи да A, B, C припадају неком великом кругу. Аналогија са раванском геометријом је та да не можемо говорити о троуглу ABC уколико су тачке колинеарне.

Странице сферног троугла дефинишемо као краће лукове великих кругова који спајају темена. Дужине страница можемо обележити са a, b и c.
Угао између страница сферног троугла је угао између одговарајућих тангенти у теменима:

$$\alpha = \sphericalangle(\overrightarrow{T_{AB}}, \overrightarrow{T_{AC}}), \quad \beta = \sphericalangle(\overrightarrow{T_{BA}}, \overrightarrow{T_{BC}}), \quad \gamma = \sphericalangle(\overrightarrow{T_{CA}}, \overrightarrow{T_{CB}}).$$



Испоставља се да за сферни троугао важи

$$\cos a = \cos b \cos c + \sin b \sin c \cos \alpha,$$

што је пандан косинусној теореми еуклидске геометрије. Пандан синусне теореме је:

$$\frac{\sin a}{\sin \alpha} = \frac{\sin b}{\sin \beta} = \frac{\sin c}{\sin \gamma}.$$

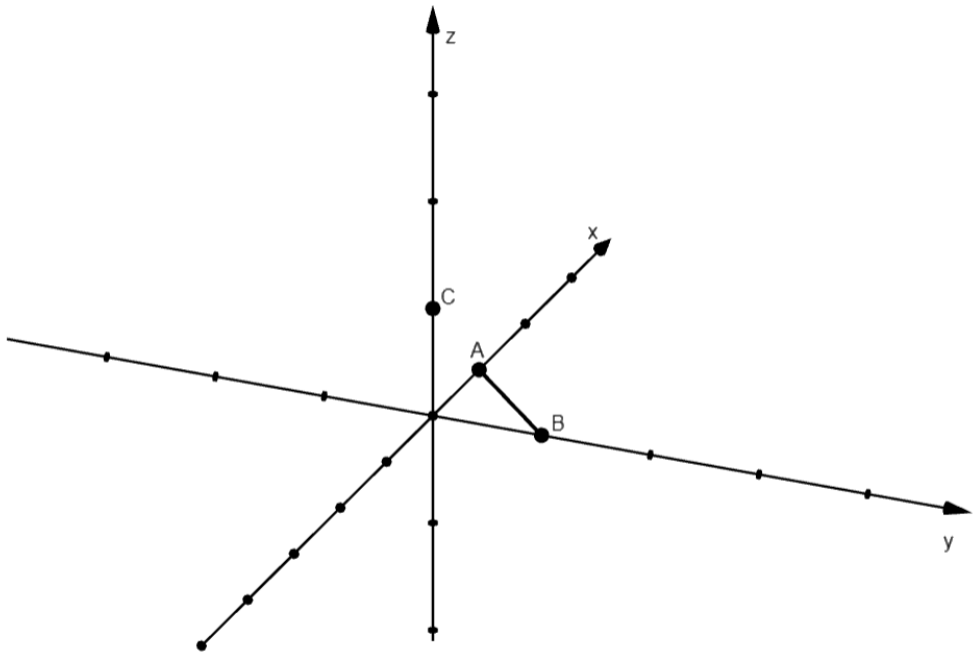
Постоји и дуална косинусна теорема, која даје однос између углова:

$$\cos \alpha = -\cos \beta \cos \gamma + \sin \beta \sin \gamma \cos a,$$
$$\cos \beta = -\cos \gamma \cos \alpha + \sin \gamma \sin \alpha \cos b,$$
$$\cos \gamma = -\cos \alpha \cos \beta + \sin \alpha \sin \beta \cos c.$$

Обим троугао се дефинише као збир свих страница $O = a + b + c$, док се може доказати да је површина сферног троугла $P = \alpha + \beta + \gamma - \pi$.

Потребно је да коначна решења упишите у за то предвиђене правоугаонике. Поступак се не оцењује. Негативне поене на овом задатку не можете да добијете.

(а) [**4 поена**] Дат је круг полупречника r_1 који садржи тачке $A(1,0,0)$, $B(0,1,0)$, $C(0,0,1)$ (видети слику испод). Означимо са d_1 растојање између тачака A и B **мерено по том кругу**. Означимо са d_2 дужину странице која спаја тачке A и B сферног троугла ABC (тј. растојање тачака A и B **мерено по јединичној сфери са центром у координатном почетку** - тачки са координатама $(0,0,0)$). Тада је $d_2 - d_1$ једнако:



(б) [**5 поена**] Сферни троугао ABC који се налази на јединичној сфери има углове $\alpha = \frac{\pi}{3}$, $\beta = \frac{2\pi}{3}$ и површину $P = \frac{\pi}{6}$. Његов обим је:

Задатак 4. Увод у вероватноћу

Вероватноћа је грана математике која се бави проучавањем случајних појава и њихових исхода. Она омогућава квантификацију несигурности и помаже у доношењу одлука у ситуацијама где исходи нису унапред познати. Основни задатак истраживача у области вероватноће је да идентификује основне карактеристике случајног експеримента, дефинише одговарајући простор вероватноће и развије моделе који омогућавају анализу и предвиђање будућих исхода. Вероватноћа има широку примену у свакодневном животу и различитим областима науке. На пример, користи се у играма на срећу за предвиђање шансе за добитак, у спортским статистикама за анализу учинка играча, и у метеорологији за процену вероватноће временских појава попут кише. Такође, вероватноћа игра кључну улогу у алгоритмима друштвених мрежа и видео игара, где помаже у генерисању насумичних догађаја или персонализованих препорука.

Основни појмови у вероватноћи

Простор догађаја

Простор догађаја (Ω) је скуп свих могућих исхода неког случајног експеримента. Сваки елемент овог скупа представља један могући исход.

Пример 1: Код бацања новчића, простор догађаја је $\Omega = \{\text{писмо}, \text{глава}\}$.

Пример 2: Код бацања шестостране коцкице, простор догађаја је $\Omega = \{1, 2, 3, 4, 5, 6\}$.

Случајни догађај

Случајни догађај је подскуп простора догађаја, који обухвата одређене исходе. На пример, ако разматрамо бацање коцкице:

- A: „Коцкица показује паран број” је случајни догађај $A = \{2, 4, 6\}$.
- B: „Коцкица показује број мањи од 4” је случајни догађај $B = \{1, 2, 3\}$.

Ако је догађај једнак целом простору догађаја ($A = \Omega$), то је **сигуран догађај**. Ако је празан скуп ($A = \emptyset$), то је **немогућ догађај**.

Вероватноћа

Вероватноћа је функција која сваком догађају A придружује број из интервала $[0, 1]$. Функција P мора задовољити следеће особине:

- **Ненегативност:** За сваки догађај A важи $0 \leqslant P(A) \leqslant 1$.
- **Вероватноћа сигурног догађаја:** Вероватноћа да се догоди сигуран догађај је $P(\Omega) = 1$.
- **Вероватноћа немогућег догађаја:** Вероватноћа немогућег догађаја је: $P(\emptyset) = 0$.
- **Адитивност за дисјунктне догађаје:** За коначан број дисјунктних догађаја $A_1, A_2, \ldots, A_n$ важи

$$P\left(\bigcup_{i=1}^{n} A_i\right) = \sum_{i=1}^{n} P(A_i).$$

Комплемент догађаја

За сваки догађај A дефинисан у простору догађаја Ω, комплемент догађаја A, означен са A^c, представља скуп исхода који не припадају A. Формално, $A^c = \Omega \setminus A$. Јасно, за комплемент важи $P(A) + P(A^c) = 1$.

Пример: Претпоставимо да бацамо стандардну шестостране коцкицу. Ако је догађај A: „Испао је паран број” ($A = \{2, 4, 6\}$). Тада је A^c: „Испао је непаран број” ($A^c = \{1, 3, 5\}$). Вероватноће су:

$$P(A) = \frac{3}{6} = 0.5, \quad P(A^c) = \frac{3}{6} = 0.5, \quad P(A) + P(A^c) = 1.$$

Условна вероватноћа

Условна вероватноћа $P(A|B)$ представља вероватноћу да се догоди догађај A, под условом да се догодио догађај B. Дефинише се формулом:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}, \quad \text{ако је } P(B) > 0.$$

Формула потпуне вероватноће

Ако је простор догађаја подељен на дисјунктне догађаје $H_1, H_2, \ldots, H_n$ (тзв. хипотезе) који покривају цео простор ($H_1 \cup H_2 \cup \cdots \cup H_n = \Omega$), тада вероватноћа догађаја A може да се изрази као:

$$P(A) = \sum_{i=1}^{n} P(H_i) \cdot P(A|H_i).$$

Пример

Замислимо експеримент: Бацамо шестострану коцкицу. Израчунајмо вероватноћу догађаја: „Број је паран и мањи од 5".

Решење. Означимо A: „Испао је паран број" ($A = \{2, 4, 6\}$), а са B: „Испао је број мањи или једнак 3" ($B = \{1, 2, 3\}$). Јасно, вероватноће тих догађаја су $P(A) = \frac{3}{6} = 0.5$ и $P(B) = \frac{3}{6} = 0.5$.
Приметимо сада, вероватноћа да је испао паран број и да је тај број мањи или једнак 3) је $P(A \cap B) = \frac{1}{6}$ (пошто је $A \cap B = \{2\}$). Слично, вероватноћа да је испао паран број или број који је мањи или једнак 3 се може израчунати принципом укључења и искључења (а може и на друге начине, али овде ради илустрације приказујемо ово решење), чиме добијамо

$$P(A \cup B) = P(A) + P(B) - P(A \cap B) = 0.5 + 0.5 - \frac{1}{6} = \frac{6}{6} - \frac{1}{6} = \frac{5}{6}.$$

Сада, условна вероватноћа $P(A|B)$ се рачуна на следећи начин: ако знамо да је број мањи или једнак 3 (B), тражимо вероватноћу да је број паран (A), што даје $P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{\frac{1}{6}}{\frac{3}{6}} = \frac{1}{3}$.
Коначно, израчунајмо вероватноћу догађаја C: „Број је паран и мањи од 5". Овај догађај можемо изразити помоћу партиције простора дефинисане догађајима H_1 : „Број је мањи или једнак 3" ($H_1 = \{1, 2, 3\}$) и H_2: „Број је већи од 3" ($H_2 = \{4, 5, 6\}$).
Применом формулу потпуне вероватноће добијамо $P(C) = P(C|H_1) \cdot P(H_1) + P(C|H_2) \cdot P(H_2)$. Вероватноћа $P(H_1)$, да је број мањи или једнак 3, је $P(H_1) = \frac{3}{6} = 0.5$, док је $P(H_2) = \frac{3}{6} = 0.5$, јер оба скупа имају по три елемента. Вероватноћа $P(C|H_1)$, да је број паран у H_1, је $P(C|H_1) = P(\{2\}) = \frac{1}{3}$, а $P(C|H_2)$, да је број паран у H_2, је $P(C|H_2) = P(\{4, 6\}) = \frac{2}{3}$. Заменом у формулу потпуне вероватноће добијамо

$$P(C) = \frac{1}{3} \cdot 0.5 + \frac{2}{3} \cdot 0.5 = \frac{1}{6} + \frac{2}{6} = \frac{3}{6} = 0.5.$$

Задатак 1

Нека су A, B, C, D и E догађаји са позитивном вероватноћом, дефинисани у оквиру неког простора догађаја Ω.

(а) [2 поена] Да ли је могуће да су **истовремено** испуњене следеће неједнакости:

$$P(A|C) < P(B|C), \quad P(A|C^c) < P(B|C^c), \quad P(A) > P(B)?$$

а) да, увек б) постоји случај у ком важе све три в) не, никад г) нема довољно информација

(б) [3 поена] Ако је $P(D) < 1$, проверити која од следећих еквиваленција **није тачна** (могуће је заокружити један или више одговора):

1. $P(D|E) > P(D) \iff P(DE) > P(D)P(E)$,
2. $P(DE) > P(D)P(E) \iff P(E|D) < P(E|D^c)$,
3. $P(D|E) > P(D) \iff P(E|D) > P(E|D^c)$.

а) Еквиваленција 1 б) Еквиваленција 2 в) Еквиваленција 3 г) Све еквиваленције су тачне

Задатак 2

На основу спроведеног истраживања, познате су следеће информације о потрошачима у супермаркету:

- A: Потрошач купује млеко.
- B: Потрошач долази ујутру.
- C: Потрошач долази поподне.
- D: Потрошач долази увече.

Подаци:

- Вероватноћа да потрошач купи млеко ако долази ујутру је $P(A|B) = \frac{2}{5}$,
- Вероватноћа да потрошач долази ујутру је $P(B) = \frac{1}{2}$,
- Вероватноћа да потрошач купи млеко ако долази поподне је $P(A|C) = \frac{3}{10}$,
- Вероватноћа да потрошач долази поподне ако купује млеко $P(C|A) = \frac{2}{5}$,
- Вероватноћа да потрошач купује млеко $P(A) = \frac{7}{20}$.

(а) [2 поена] Колико је $P(C)$?

а) $\frac{1}{3}$ б) $\frac{7}{15}$ в) $\frac{21}{80}$ г) $\frac{6}{7}$

(б) [2 поена] Израчунати вероватноћу да потрошач купује млеко ујутру или поподне, тј. наћи $P(A \cap (B \cup C))$. Претпоставити да су B и C дисјунктни догађаји.

а) $\frac{203}{600}$ б) $\frac{7}{10}$ в) $\frac{7}{20}$ г) $\frac{17}{50}$

За радознале

Јесте ли знали? Ако у просторији има 23 особе, постоји преко 50% шансе да двоје од њих имају рођендан истог дана! Ово је познато као **парадокс рођендана**.

Задатак 5. Математичка анализа

Анализа је математичка област која се бави изучавањем граничних процеса и бесконачно малих величина. Један од кључних појмова математичке анализе јесте **лимес** (превод са латинског на српски је граница). Лимес је од изузетног значаја у многим областима. На пример, у машинском учењу користи се за анализу конвергенције градијентног спуста како би се осигурало да низ апроксимација тежи глобалном минимуму функције губитка. Ово је кључно за потврду ефикасности и тачности модела.

Супремум, инфимум, максимум и минимум скупа

Нека је S непразан подскуп скупа реалних бројева $\mathbb{R}$ (може бити и бесконачан).

- **Супремум (најмање горње ограничење)** скупа S, у ознаци $\sup S$, је број који задовољава:
  1. $\forall x \in S, x \leqslant \sup S$,
  2. $\forall y < \sup S, \exists x \in S$ такво да $y < x$. Ако не постоји реалан број који задовољава претходно, али за сваки реалан број M постоји $x \in S$ такво да је $x > M$ тада кажемо да је $\sup S = +\infty$
- **Инфимум (највеће доње ограничење)** скупа S, у ознаци $\inf S$, је број који задовољава:
  1. $\forall x \in S, x \geqslant \inf S$,
  2. $\forall y > \inf S, \exists x \in S$ такво да $y > x$. Ако не постоји реалан број који задовољава претходно, али за сваки реалан број m постоји $x \in S$ такво да је $x < m$, тада кажемо да је $\inf S = -\infty$.
- **Максимум** скупа S, у ознаци $\max S$, је број који задовољава:
  1. $\max S \in S$,
  2. $\forall x \in S, x \leqslant \max S$.
- **Минимум** скупа S, у ознаци $\min S$, је број који задовољава:
  1. $\min S \in S$,
  2. $\forall x \in S, x \geqslant \min S$.

Пример: Нека је $S = \left\{ \frac{1}{n} \mid n \in \mathbb{N} \right\}$. Чланови тог скупа су $1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \ldots$ Тада је:

$$\sup S = 1, \quad \inf S = 0, \quad \max S = 1, \quad \min S \text{ не постоји.}$$

$\min S$ не постоји јер чланови скупа S могу бити произвољно близу 0, а $0 \notin S$.

Низ је функција $a : \mathbb{N} \to \mathbb{R}$, где сваком природном броју n придружујемо реалан број a_n.

- Низ (a_n) је **монотоно растући** ако важи $a_n \leqslant a_{n+1}$ за свако $n \geqslant n_0$, за $n_0 \in \mathbb{N}$ (почев од неког n_0 важи);
- Низ (a_n) је **монотоно опадајући** ако важи $a_n \geqslant a_{n+1}$ за свако $n \geqslant n_0$, за неко $n_0 \in \mathbb{N}$ (почев од неког n_0 важи);
- Низ (a_n) је **ограничен** ако постоје бројеви $m, M \in \mathbb{R}$ такви да $m \leqslant a_n \leqslant M$ за свако $n \in \mathbb{N}$.

Лимес низа (a_n) је број L за који важи:

$$(\forall \varepsilon > 0)(\exists n_0 \in \mathbb{N})(n \geqslant n_0 \implies |a_n - L| < \varepsilon).$$

Овај појам се користи да би се описао број ка којем чланови низа теже, како n расте, односно неформално говорећи, L је лимес низа (a_n) ако су почевши од некад сви чланови низа a_n произвољно близу L. Ако низ (a_n) има лимес L, кажемо да низ **конвергира** и пишемо:

$$\lim_{n \to \infty} a_n = L.$$

Овај појам се користи да би се описао број ка којем чланови низа теже, како n расте. Ако не постоји овакво L кажемо да (a_n) нема коначан лимес.

Пример за лимес низа

Размотримо низ $a_n = \frac{(-1)^n}{n}, \quad n \in \mathbb{N}$. Ми желимо да покажемо да је $\lim_{n\to\infty} a_n = 0$. Према дефиницији лимеса низа, треба да докажемо да за свако $\varepsilon > 0$ постоји природан број N такав да за сваки $n > N$ важи $|a_n - 0| < \varepsilon$. Погледајмо апсолутну вредност члана низа $|a_n - 0| = \left|\frac{(-1)^n}{n}\right| = \frac{1}{n}$. Тада захтев $|a_n - 0| < \varepsilon$ постаје $\frac{1}{n} < \varepsilon$. Поставимо $N = \left\lceil \frac{1}{\varepsilon} \right\rceil$ (ово је ознака за горњи цео део броја). Тада за сваки $n > N$ имамо $n > \frac{1}{\varepsilon}$ па је и $\frac{1}{n} < \varepsilon$. Дакле, за свако $\varepsilon > 0$ постоји N такав да је $|a_n - 0| < \varepsilon$ за сваки $n > N$. Према дефиницији лимеса, следи да $\lim_{n\to\infty} a_n = 0$.

Теорема. Сваки **монотон** и **ограничен** низ (a_n) конвергира.

Лимес функције ако имамо низ (a_n) који конвергира ка L, тада ће и низ $(f(a_n))$ да конвергира ка L уколико је функција непрекидна. Овај појам се користи како би се описала понашања функције у околини тачке a, чак иако функција није дефинисана у самој тачки a.

Непрекидне функције

Функција $f(x)$ је непрекидна у тачки a ако важи $\lim_{x\to a} f(x) = f(a)$.
То значи да се вредност функције $f(x)$ може произвољно приближити вредности $f(a)$ када x довољно близу тачке a. Примери непрекидних функција би биле квадратна (x^2), линеарна ($mx + b$), експоненцијална (e^x) и тригонометријске ($\sin(x), \cos(x)$). Оне су непрекидне на целом свом домену.

У овом задатку је неопходно да прикажете комплетан поступак. Бодоваће се и парцијална решења.

а) [4 поена] Нека је дат скуп

$$S = \left\{ n + \frac{25}{n+2} \mid n \in \mathbb{N} \right\}.$$

Одредити инфимум ($\inf S$) и супремум ($\sup S$) скупа S. Одредити да ли скуп S има минимум ($\min S$) и максимум ($\max S$). Ако постоји, наћи њихове вредности.

б) [3 поена] Нека је дат скуп:

$$T = \left\{ \frac{n^2 + 2n + 25}{n+2} \cdot \operatorname{tg}\left(\frac{\pi m + 1}{2m}\right) \mid m, n \in \mathbb{N} \right\}.$$

Одредити супремум скупа T.

в) [7 поена] Дат је низ $\{a_n\}$ који задовољава следеће услове:

$$2 < a_1 < 3, \quad \text{и} \quad 5a_{n+1} = a_n^2 + 6 \quad \text{за све} \quad n \in \mathbb{N}.$$

Испитати да ли низ $\{a_n\}$ конвергира. Ако низ конвергира, израчунати његов лимес.

Задатак 6. Дигитални запис података

Приликом комуникације између два уређаја, који могу и не морају бити у оквиру истог рачунарског система, може доћи до непредвиђених грешака. Те грешке могу да се догоде или на самом преносном путу, или на локацијама где се налазе уређаји који учествују у комуникацији. Разлози за настанак грешака су разни, а најчешћи узроци су шумови (проузроковани електромагнетним сметњама), физичка оштећења преносног медијума, преоптерећеност мреже, и слично. Преносни сигнал, односно порука коју један уређај шаље другом, интерпретира се као низ нула и јединица, па се грешке приликом преноса манифестују кроз промену вредности једног или више битова.

Будући да је прецизан пренос података од суштинске важности, развијени су различити алгоритми за детекцију грешака. Неки од познатијих су контрола парности, контрола збира блока, циклична провера редунданси (CRC) и Хамингов SEC код. Последњи поменути алгоритам, Хамингов SEC код, за разлику од осталих, поред откривања постојања грешке, може и да изврши корекцију на биту на којем се појавила грешка.

У овом делу задатка бавићемо се управо тим алгоритмом. Алгоритам објашњавамо на примеру порука које се састоје од 12 битова, где првих 8 битова представља сам садржај поруке, док преостала 4 бита представљају контролне битове. Прималац, на основу добијене поруке, генерише своје контролне битове и упоређује их са контролним битовима које му је проследио пошиљалац. На основу тога ће закључити да ли је дошло до грешке на неком биту, и ако јесте, одрадити потребне корекције. Нека су битови поруке означени са $m_8, ..., m_1$, а контролни битови са $c_4, ..., c_1$, на следећи начин:

m_8	m_7	m_6	m_5	m_4	m_3	m_2	m_1	c_4	c_3	c_2	c_1

Сада је потребно формирати таблицу Хамингових SEC кодова. У првој колони су декадни бројеви од 12 до 1, у другој колони одговарајући бинарни еквиваленти, записани помоћу четири бита, а у трећој се сваком броју (од 12 до 1) додељује неки од битова $m_8, ..., m_1, c_4, ..., c_1$. Битови $c_4, ..., c_1$ се додељују оним бројевима који су степени броја два (тј. бројевима 1, 2, 4 и 8), а на преостала места се додају битови $m_8, ..., m_1$ на начин приказан у следећој табели:

12	1100	m_8
11	1011	m_7
10	1010	m_6
9	1001	m_5
8	1000	c_4
7	0111	m_4
6	0110	m_3
5	0101	m_2
4	0100	c_3
3	0011	m_1
2	0010	c_2
1	0001	c_1

Затим се израчунавају контролни битови $c'_4, ..., c'_1$ преко битова $m_8, ..., m_1$, тако што се редом у четвртој, трећој, другој и првој колони свих бинарних записа из табеле у одговарајућу формулу убаце они битови m_i чија је вредност 1 и изврши се њихова ексклузивна дисјункција. Другим речима, важи:

$$c'_4 = m_5 \oplus m_6 \oplus m_7 \oplus m_8$$

$$c'_3 = m_2 \oplus m_3 \oplus m_4 \oplus m_8$$

$$c'_2 = m_1 \oplus m_3 \oplus m_4 \oplus m_6 \oplus m_7$$

$$c'_1 = m_1 \oplus m_2 \oplus m_4 \oplus m_5 \oplus m_7$$

Коначно, израчунава се вредност $c_4c_3c_2c_1 \oplus c'_4c'_3c'_2c'_1$, чиме се добија четворобитна бинарна ниска $r_4r_3r_2r_1$. Уколико се овај број не налази у горе наведеној табели, или се пак налази, али се у одговарајућој врсти налази нека од вредности $c_4, ..., c_1$, није дошло до грешке. У супротном, добијена вредност упућује на ком од контролних битова $m_8, ..., m_1$ је дошло до грешке, те је ту вредност потребно инвертовати, како би се добио тачан садржај поруке.

(а) [**4 поена**] Прималац је добио три наредне дванаестобитне поруке:

1. 010011001110
2. 010001111100
3. 010011101010

Свака од примљених порука носи информацију о једном ASCII карактеру. Знајући да је ASCII код карактера 'A' 65, која трословна реч је састављена од карактера који су кодирани наведеним порукама?

а) LGN

б) LON

в) LOL

г) Ништа од наведеног

Размотримо сада поменути алгоритам цикличне провере редунданси (односно CRC методу), који, за разлику од Хамингового SEC кода, не може да детектује позицију бита на ком се догодоила грешка, већ само да установи да је до исте дошло. Прималац сада, поред кодиране поруке, добија и *полином генератор*, на основу којих може или да установи да је дошло до грешке и да затражи поновно слање поруке, или да издвоји оригиналну поруку, ако се испостави да није дошло до грешке.

Полином генератор је произвољан полином чији су коефицијенти или 1, или 0. *Бинарни запис* полинома генератора $G(x) = a_nx^n + a_{n-1}x^{n-1} + ... + a_1x + a_0$ је бинарна ниска састављена од коефицијената полинома $G(x)$, тј. ниска $a_na_{n-1}...a_1a_0$. Имајући ове дефиниције у виду, поступак CRC методе можемо описати следећим корацима:

1. Пошиљалац бира произвољан полином генератор. Нека је степен изабраног полинома n.

2. Пошиљалац дописује n нула на крај поруке коју жели да пошаље (што представља множење поруке са x^n).

3. Извршава се дељење новодобијене поруке бинарним записом полинома генератора. Дељење бинарних бројева је лакше него дељење декадних, јер се уместо одузимања одговарајућих цифара врши њихова ексклузивна дисјункција, па нема ни позајмица. Водеће нуле које се добијају у међурезултатима се игноришу, а потом се спушта онолико цифара дељеника колико је неопходно да се добије број чија је дужина једнака дужини делиоца (под условом да постоји толико цифара у дељенику). Приликом овог дељења, од интереса је само остатак који се добија, не и количник. Пример оваквог дељења приказан је у наставку:

```
111001100000 / 11001
11001
  10111
  11001
   11100
   11001
     10100
     11001
      11010
      11001
         110
```

4. Порука која се шаље примаоцу добија се када се на оригиналну поруку надовеже остатак добијен претходним дељењем. Тај остатак је, евентуално, потребно допунити водећим нулама тако да буде записан на укупно n места.

5. Тако добијена порука се шаље примаоцу, заједно са полиномом генератором.

6. Прималац врши дељење примљене поруке одговарајућим бинарним записом добијеног полинома генератора, на исти начин који је описан у кораку 3.

7. Уколико је добијени остатак различит од 0, констатује се грешка у преносу и захтева се поновно слање поруке. У супротном, порука је исправно примљена, а оригинални садржај добија се одбацивањем последњих n битова примљене поруке.

(б) [**2.5 поена**] Прималац је добио поруку 101110001010111 и полином генератор $G(x) = x^4 + x^2 + 1$. Оригинални садржај поруке је:

а) немогуће одредити, јер је CRC алгоритам установио постојање грешке

б) 1001010011

в) 1001

г) ништа од наведеног

(в) [**2.5 поена**] Пошиљалац жели да као садржај поруке пошаље осмобитни бинарни запис броја 214 (записаног у декадном бројном систему). За полином генератор изабрао је полином $G(x) = x^4 + x^2 + x$. Облик за слање ове поруке је:

а) 110101101110

б) 1110
в) 10110
г) ништа од наведеног
Напомена: Таблица ексклузивне дисјункције битова приказана је испод:

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

Задатак 7. Машинско учење

Кластеровање је метода машинског учења која се користи за груписање података у мање групе, познате као **кластери**. Подаци унутар истог кластера имају сличне особине и међусобно су ближи, док су у односу на податке из осталих кластера различитији и удаљенији. Ова техника представља **ненадгледану методу**, што значи да за податке унапред није познато којој групи припадају. Уместо тога, алгоритми кластеровања анализирају особине података и на основу њих одлучују којој групи (кластеру) припадају. Циљ је идентификовати природне групе и обрасце који постоје у подацима, чиме се омогућава боље разумевање сложених скупова података.

Алгоритми кластеровања су незаменљив алат у савременој науци и индустрији, захваљујући способности да групишу сличне податке и открију скривене обрасце. На пример, у биологији се користе за груписање организама на основу генетских сличности, док у друштвеним мрежама омогућавају идентификацију заједница корисника са заједничким интересовањима – замислите, на пример, груписање корисника који прате исте теме или инфлуенсере. Када се ради о сликама, кластеровање може да разврста фотографије према визуелним сличностима – на пример, све слике са истим лицима могу бити груписане у један фолдер, или слике са исте локације аутоматски препознате и постављене заједно. Такође, кластеровање може омогућити ефикасно претраживање огромних збирки чланака, вести или књига. На пример, у великим новинским архивама, алгоритми могу аутоматски груписати чланке који говоре о истим темама, као што су политика, спорт или култура, чинећи претраживање бржим и једноставнијим.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) је популаран алгоритам за кластеровање података заснован на густини тачака (података) у простору. Његова главна идеја је да идентификује густе просторе тачака као кластере, док тачке које су изоловане или се налазе у ретким областима означава као шум. Овај алгоритам користи два основна параметра:

- r - максимално растојање између тачака како би се сматрале блиским.
- n - минималан потребан број тачака блиских некој тачки x како би се простор око тачке x сматрао кластером.

DBSCAN се састоји из следећих корака:

Фаза означавања

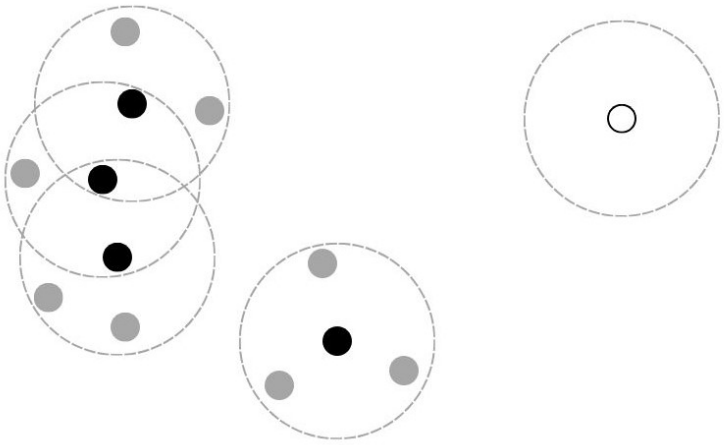
1. Бира се тачка x из скупа података који је доступан.
2. За тачку x се израчунава број N − колико тачака се налази на растојању мањем или једнаком од r, не убрајајући тачку x.
   - Уколико важи $N >= n$, тачка x чини **језгро кластера**.
   - Уколико важи $N < n$, и уколико се x налази на максималном растојању r од неке тачке која чини језгро кластера, тада тачка x представља **граничну тачку кластера**.
   - Уколико важи $N < n$, и уколико се x не налази на максималном растојању r од неке тачке која чини језгро кластера, тада тачка x представља **шум** (изоловану тачку).
3. Кораци 1. и 2. се понављају док се не обраде све тачке из скупа података.

Фаза формирања кластера

Свака тачка која је означена као језгро служи као почетна тачка за формирање кластера. Све тачке које се налазе на растојању мањем од r од ње, припадаће истог кластеру као и она. Уколико се међу тим тачкама налази још нека тачка језгра, тада ће се кластер проширити и на тачке које су на растојању мањем од r и од те тачке језгра.

Пример. На слици 1 може се видети пример DBSCAN алгоритма над конкретним подацима. Црном бојом означене су тачке које чине језгро, сивом бојом граничне тачке а тачке које су непопуњене представљају шум. На левом делу слике се уочава један кластер, који је настао надовезивањем три тачке језгра са њиховим околним граничним тачкама. У средишњем делу уочава је још један одвојени кластер, који чини једна тачка језгра са своје три граничне тачке. У горњем десном углу налази се шум, и он није део ни једног кластера. Закључујемо да се у подацима налазе два кластера, тј. два простора густих тачака који су

раздвојени ретким простором. Управо смо уз помоћ параметара r и n дефинисали појам густине простора. Са другачијим избором вредности тих параметара, број кластера би био другачији, јер би самим тим и појам густине простора био другачији. У овом примеру су испрекиданом линијом исцртане области на растојању r (r-околине) од тачака језгра и шума, а за параметар n је узета вредност 3.



Слика 1: DBSCAN алгоритам - пример

Задатак. Замислите да радите на пројекту развоја туризма у Републици Србији. Ваш задатак је да групишете градове у туристичке кластере тако да градови унутар истог кластера треба да буду довољно близу како би могли ефикасно сарађивати на промоцији заједничких атракција, развоју смештајних капацитета и организацији међуградских тура. За овај задатак ћете управо користити DBSCAN алгоритам са параметрима $r = 80km$ и $n = 2$ и матрицу растојања између градова која је наведена у табели 1.

(а) [**2 поена**] Број кластера добијен кластеровањем са задатим параметрима је:

а) 1 б) 2 в) 3 г) 4

(б) [**2 поена**] Број градова који представљају језгро кластера је:

а) 6 б) 7 в) 8 г) 9

(в) [**2 поена**] Број градова који неће сарађивати са осталим градовима у реализацији пројекта је:

а) 0 б) 1 в) 2 г) 3

(г) [**3 поена**] Заокружи тврдњу која је тачна:

а) Повећањем вредности параметра r, број кластера расте.

б) Све тачке унутар истог кластера морају бити на међусобном растојању мањем од r.

ц) Тачка која је шум у својој r-околини може имати тачку која је гранична.

д) Све претходне тврдње су нетачне.

Табела 1: Растојања између градова

	БГ	СД	ЈА	КИ	КШ	НИ	УЕ	НС	ПА	ЗР	ВР	ББ
БГ	0	70	140	130	200	240	200	80	160	90	340	170
СД	70	0	100	160	150	190	230	140	110	120	290	240
ЈА	140	100	0	310	60	100	150	230	20	210	200	180
КИ	130	160	310	0	350	400	340	100	330	50	550	330
КШ	200	150	60	350	0	80	130	270	40	250	160	150
НИ	240	190	100	400	80	0	250	330	90	310	120	270
УЕ	200	230	150	340	130	250	0	250	170	270	290	30
НС	80	140	230	100	270	330	250	0	260	60	450	230
ПА	160	110	20	330	40	90	170	260	0	240	180	190
ЗР	90	120	210	50	250	310	270	60	240	0	430	285
ВР	340	290	200	550	160	120	290	450	180	430	0	300
ББ	170	240	180	330	150	270	30	230	190	285	300	0

Задатак 8. Алгоритми и структуре података

Велики број проблема са којим се сусрећемо је индуктивне природе, и у њиховом решавању се јављају рекурзивне функције, то јест функције које позивају саме себе. У многим случајевима се дешава да током извршавања рекурзивне функције долази до преклапања рекурзивних позива тј. да се идентични рекурзивни позиви (рекурзивни позиви са идентичним параметрима) извршавају више пута. Ако се то дешава често, програми су по правилу веома неефикасни (у многим случајевима број рекурзивних позива, па самим тим и сложеност бива експоненцијална у односу на величину улаза). До ефикаснијег решења се често може доћи техником динамичког програмирања. Оно често временску ефикасност поправља ангажовањем додатне меморије у којој се бележе резултати извршених рекурзивних позива. Динамичко програмирање долази у два облика.

- Техника **мемоизације** или **динамичког програмирања наниже** задржава рекурзивну дефиницију али у додатној структури података (најчешће низу или матрици, ређе мапи тј. речнику) бележи све резултате рекурзивних позива, да бих их у наредним позивима у којима су параметри исти само очитала из те структуре.
- Техника **динамичког програмирања навише** у потпуности уклања рекурзију и ту помоћну структуру података попуњава исцрпно у неком систематичном редоследу. Дакле, рекурзивна конструкција се замењује индуктивном, тј. итеративном.

Док се код мемоизације може десити да се рекурзивна функција не позива за неке вредности параметара, код динамичког програмирања навише се израчунавају вредности функције за све могуће вредности њених параметара мањих од вредности која се заправо тражи у задатку. Иако се на основу овога може помислити да је мемоизација ефикаснија техника, у пракси је чешћи случај да је током одмотавања рекурзије потребно израчунати вредност рекурзивне функције за баш велики број различитих параметара, тако да се ове предности мемоизаиције у пракси ретко сређе.

Пример. Најбољи начин да разјаснимо технику динамичког програмирања је да је илуструјемо на погодно одабраном примеру. За то ћемо користити Фибоначијев низ, који је опште познати проблем и кроз чије се решавање могу илустровати већина основних концепта динамичког програмирања.

Основно рекурзивно решење Имплементацију можемо направити рекурзивно, директно из дефиниције (за $n = 0$ и $n = 1$ функција враћа 1, а у супротном враћа збир рекурентних позива за $n - 1$ и $n - 2$).

```
function fib(n):
  if n == 0 or n == 1:
    return 1
  else:
    return fib(n - 1) + fib(n - 2)
```

Неефикасност овакве имплементације се може видети у томе што се рекурзивни позиви за исте вредности врше више пута. Зато ћемо видети како динамичким програмирањем можемо доћи до ефикасније имплементације.

Мемоизација уз коришћење статичког низа Прва могућност је да применимо мемоизацију. Резултате рекурзивних позива ћемо памтити у некој помоћној структури (у овом случају, статичком низу), и онда ћемо на почетку сваког рекурзивног позива проверавати да ли је резлтат за текућу вредност већ израчунат. Пошто морамо некако обележити вредост у низу која још није рачуната, то радимо помоћу неке специјалне вредности, у овом случају, како су сви фибоначијеви бројеви позитивни, бирамо -1/

```
function fib(n, memo):
    // Ako je vrednost već izračunata, vraćamo je
    if memo[n] != -1:
        return memo[n]

    // Bazni slučajevi
    if n == 0 or n == 1:
        memo[n] = 1
        return memo[n]

    // Inače, računamo n-ti element niza memo i vraćamo ga
    memo[n] = fib(n - 1, memo) + fib(n - 2, memo)
    return memo[n]

function fin(n):
    // Pravimo niz memo od n+1 elemenata
    memo[n+1]

    // Inicijalizujemo elemente niza memo na -1
    init(n, memo)

    return fib(n, memo)

function init(n, memo):
    for i = 0 to n:
        memo[i] = -1
```

Динамичко програмирање навише Друга могућност је да применимо динамичко програмирање навише. Техника динамичког програмирања навише подразумева уклањање рекурзије (она је подложна многим техничким ограничењима) и да се све вредности у низу попуне неким редоследом. Пошто вредности на вишим позиција зависе од вредности на нижим, то радимо са лева на десно. Прво попуњавамо прва два елемента (базне случајеве) па остале попуњавамо у петљи на основу претходне две.

```
function fib(n):
    // Pravimo niz dp od n+1 elemenata
    dp[n+1]

    //Bazni slučajevi
    dp[0] = 1
    dp[1] = 1

    // popunjavamo dp niz od 2 do n
    for i = 2 to n:
        dp[i] = dp[i - 1] + dp[i - 2]

    print(dp[n])
```

Задатак.

Дате су вредности n ($n \leq 100$) различитих новчића. Наћи најмањи број новчића потребан да се добије укупна сума s ($s \leq 1000$).

(а)[4 поена] Допунити празнине у коду следећег рекурзивног решења овог задатка:

```
//funkcijom računamo najmanji broj novčića potreban da se
//naplati suma s kad su nam na raspolaganju n novčića u nizu novčići
function minBrojNovcica(novcici[], n, s):
    //ako je potrebna suma za naplatu 0 to radimo sa 0 novčića
    if s == 0:
        return 0

    //ako nemamo više novčića ne možemo naplatiti, INF predstavlja beskonačno tj da nema rešenja
    if n == 0:
        return INF


    //posmatramo poslednju vrstu novčića, želimo da proverimo da li
    //postižemo minimum kad koristimo novčić poslednje vrste ili
    //bez njega. promenljiva br predstavlja traženi broj

    br = 0

    1._____________________________________________________

    if s >= novcici[n - 1]:
       2._____________________________________________________


    return br
```

а) 1. $br = minBrojNovcica(novcici, n-1, s)$ 2. $br = minBrojNovcica(novcici, n, s - novcici[n-1])$

б) 1. $br = minBrojNovcica(novcici, n-1, s)$ 2. $br = min(br, minBrojNovcica(novcici, n, s - novcici[n-1]) + 1)$

в) 1. $br = minBrojNovcica(novcici, n, s - novcici[n-1])$ 2. $br = min(br, minBrojNovcica(novcici, n-1, s) + 1)$

г) 1. $br = minBrojNovcica(novcici, n, s - novcici[n-1])$ 2. $br = minBrojNovcica(novcici, n-1, s)$

(б)[5 поена] Допунити празнине у коду решења које користи динамичко програмирање навише:

```
//funkcijom računamo najmanji broj novčića potreban da se
//naplati suma s kad su nam na raspolaganju n novčića u nizu novčići
//niz dp predstavlja najmanji broj novčića koji je potreban za naplatu sume s
//za svaki njegov element, tj. vrednost d[k] predstavlja najmanji broj novčića
//potreban za naplatu sume k

function minBrojNovcica(novcici[], n, sum):

    //Pravimo niz za dinamičko programiranje
    dp[MAX_SUMA + 1]

    //počinjemo analizu sa 0 vrsta novčića

    //vrednost 0 naplaćujemo sa 0 novičića
    dp[0] = 0

    //ostale vrednosti ne možemo naplatiti
    for i = 1 to sum:
        dp[i] = INF

    //popunjavamo niz dp tako što uključujemo novčiće vrstu po vrstu,
    //za svaku vrstu ažuriramo minimume tako što proverimo da li je moguće
    //uključiti novčić tekuće vrste, i ažuriramo minimum ako je potrebno


    for i = 1 to n:
        for j = 0 to sum:
            if j >= novcici[i - 1]:
                1._____________________________________________


    //vraćamo najmanji broj novčića za sumu s
    return dp[s]
```

а) 1. $dp[s] = min(dp[j], dp[s - novcici[i-1]] + 1)$

б) 1. $dp[j] = min(dp[j], dp[j - novcici[n-1]] + 1)$

в) 1. $dp[j] = min(dp[j], dp[j - novcici[i-1]] + 1)$

г) 1. $dp[j] = min(dp[j], dp[s - novcici[n-1]] + 1)$

СТУДЕНТСКА УНИЈА МАТЕМАТИЧАРА

Финално такмичење
Б категорија
HS Algorithm 2024

шифра

Задатак 9. - Лексичка анализа

Програмски језици, попут природних језика, имају **синтаксу** и **семантику**. У природном језику, синтакса одређује правила за слагање речи у реченице (нпр. редослед речи), док семантика одређује значење тих реченица. Слично, у програмирању, синтакса су правила која одређују како се пишу изрази и команде, док семантика одређује шта ти изрази и команде значе и како ће рачунар реаговати на њих. Регуларни изрази у програмским језицима баве се провером синтаксе програмског језика.

Правила регуларних израза

- `[`$abcde$`]` - У регуларном изразу се може појавити слово a, b, c, d или e.
- `[`$a-zA-Z$`]` - У регуларном изразу се може појавити било које латинично слово.
- `[`$0-9$`]` - У регуларном изразу се може појавити било која цифра.
- $[a-zA-Z0-9\_]$ - Означава било који знак који може бити слово, број или доња црта.
- $+$ - Означава "један или више претходних знакова". На пример, $[0-9]+$ значи да се може појавити било која комбинација цифара (нпр. `1`, `123`, `01234`, итд.).
- $*$ - Означава "нула или више претходних знакова". На пример, `[0-9]*` значи да се може појавити било која комбинација цифара (нпр. `1`, `123`, `01234`), као и празна реч (реч без иједног карактера).
- `.` - Ознака за било који карактер (осим краја линије).
- $\backslash s$ - Ознака за било који размак (*space*, таб, нови ред).
- $|$ - Ознака за или.$(a|b$ означава да реч може да буде a или $b)$.

Специјални карактери

Неки карактери у регуларним изразима имају специјално значење, па их не можемо користити тек тако. На пример, уколико желимо да напишемо регуларни израз за децимални број, следећи запис би био погрешан:

$$[0-9]+.[0-9]+$$

Разлог је што симбол `.` има специјално значење у регуларим изразима и означава "било који карактер". Тако би овај израз, поред децималног броја као што је `123.456`, прихватио и низ као што је $123m456$.

Да бисмо одузели симболима њихово специјално значење, користимо симбол $\backslash$. Правилно написан регуларни израз за децималне бројеве би био:

$$[0-9]+\backslash.[0-9]+$$

Специјални карактери у регуларним изразима

Списак најчешће коришћених специјалних карактера: $., *, +, ?, \{, \}, ', \$, (,), [,]$

Пример регуларног израза за илустрацију

Пример 1: Регуларни израз: $[ab]*$

Ово значи:

- У речи се могу појављивати слова a или b a и b (`[`ab`]`).
- Можеш имати нула или више понављања ($*$).

Шта **не** би било прихваћено овим регуларним изразом: *abc*, *abd*...
Шта би било прихваћено: празна реч, *aaabbb*, *ababa*, *bbb*, *aaa*...

Пример 2: Регуларни израз: $(ab)*$

Ово значи:

- Дозволи реч *ab*.
- Можеш имати нула или више понављања речи (*).

Шта **не** би било прихваћено овим регуларним изразом: *aaa*, *bbbb*, *abaa*
Шта би било прихваћено: празна реч, *ab*, *abab*, *abab*...

Пример 3: Регуларни израз: $\backslash s * primer$

Ово значи:

- $\backslash s$ значи да је испред белина.
- $\backslash s*$ значи да може да има нула или више белина.

Шта не би било прихваћено овим регуларним изразом: празна реч, `нестопример`
Шта би било прихваћено: *primer*, *primer*, *primer* (обратити пажњу на белине)

Задатак - Променљиве у програмском језику C

Један од важних концепата у програмирању су променљиве. У језику C, променљиве означавају места у меморији где се чувају подаци. Да би име променљиве било валидно, мора да задовољава следећа правила:

- Може садржати само слова (мала и велика), бројеве и доњу црту (_).
- Мора почети словом или доњом цртом, али не бројем.
- Не сме садржати специјалне знакове као што су @, %, # - итд.

Примери валидних имена променљивих у C језику:

- x
- $broj\_1$
- $\_privatna$
- $temp2$

Примери невалидних имена променљивих:

- $3broj$ (почиње бројем)
- $\_\_nijeTacnaPromenljiva$ (почиње са две доње црте)
- $123netacno$ (почиње бројем)
- $ime - varijable$ (садржи црту -)
- $@ovonijepromenljiva$ (садржи знак @)

Напомена о резервисаним речима:
Регуларни израз сам по себи не може препознати резервисане речи у језику C. Ово значи да би, на пример, реч *int* била валидна према регуларном изразу, али додатна провера семантичке исправности (нпр. поређење са листом кључних речи) мора бити изведена у програму који користи овај регуларни израз. Током израде задатака дозвољено је да регуларни израз прихвата кључне речи програмског језика Ц (*int*, *void*...).

(а) [4 поена] Написати регуларни израз који препознаје променљиву C програмског језика. :

Задатак - *#include* **директиве у језику** *C*

#include директива у језику *C* користи се за укључивање спољног кода у програм. Она омогућава повезивање са спољним фајловима, као што су:

Стандардне библиотеке (нпр. *stdio.h*, *stdlib.h*) које пружају функције за улаз/излаз, математичке операције и друге основне функционалности.

Кориснички хеадер фајлови (нпр. "*mylib.h*") који могу садржавати дефиниције функција, структура, константи, итд., које користиш у свом пројекту.

Да би нека реч била *#include* директива, мора да поштује следећа правила:

- Библиотеке са <> синтаксом (нпр. $\#include < stdio.h >$).
- Библиотеке са "" синтаксом (нпр. $\#include"myheader.h"$).
- Име билиотеке може бити састављено искључиво од слова.
- На крају имена библиотеке мора постојати ".*h*" екстензија.
- Белине се могу наћи пре и после кључне речи *#include*.

Примери валидних директива:

- $\#include \quad < stdio.h >$
- $\#include"utils.h"$
- $\#include < math.h >$
- $\#include"myheader.h"$
- $\#include < lib.h >$

Примери невалидних директива:

- $\#include < stdio >$ // фали екстензија *.h*
- $\#include"utils"$ // фали екстензија *.h*
- $\#include < 123file.h >$ // име фајла не може почети бројем
- $\#include"my-header.h"$ // садржи црту - у имену библиотеке
- $\#include < stdio.h >$ // ако није баш у облику «...» или "... ", већ нпр. има неки размак или други карактер
- $\#include"myheader.hh"$ // не одговара јер екстензија није *.h*
- $nesto < stdio.h >$ // не почиње са *#include*, и потенцијалним белинама

(б) [2 поена] Који од следећих регуларних израза препознаје све облике *#include* директива у *C* коду

Решење

а) $\backslash s * \#include\backslash s * ((< [a-zA-Z]+\backslash.h >)|("[a-zA-Z]+\backslash.h"))$

б) $\backslash s * \#include\backslash s * [< "](([a-zA-Z]+\backslash.h)[> "]$

ц) $\backslash s * \#include\backslash s * [< "](([a-zA-Z]*\backslash.h)[> "]$

д) ништа од наведеног

Задатак - Коментари у језику C

Коментари у језику C користе се за објашњавање кода и олакшавање читања програма. Коментари се игноришу током компилације, па не утичу на извршавање програма. Једнолинијски коментар је у форми такав да на почетку има две косе црте // па затим следи низ карактера, и коментар се завршава карактером новог реда.Коментар такође може бити и празна линија (након црта које означавају почетак коментара облика "//").

// Ovo je komentar

/Ovo nije komentar jer nema dve kose crte na pocetku komentara

// Ovo nije jednolinijski komentar
jer se nalazi na vise linija

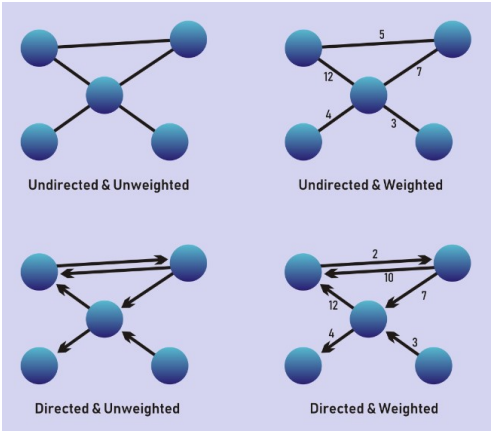
(в) [**3 поена**] Напиши регуларни израз који препознаје једнолинијски коментар у језику C.

Решење

Задатак 10. Вештачка интелигенција

Граф је структура која се састоји од скупа чворова и грана које повезују те чворове. Формално, граф се дефинише као $G = (V, E)$, где је:

- **V скуп чворова,**
- **E скуп грана које повезују парове чворова.**



Графови могу бити усмерени (гране имају смер) или неусмерени (гране немају смер). Такође, графови могу бити тежински или нетежински:

- **У тежинским графовима свакој грани је придружена одређена тежина (нпр. удаљеност, време, трошак...).**
- **У нетежинским графовима све гране имају једнаку тежину.**

Графови су корисни у многим свакодневним ситуацијама, попут планирања путева у навигационим апликацијама, оптимизације мрежа у телекомуникацијама и анализе друштвених односа на платформама попут *Facebook*-а. Такође се користе у логистици за планирање рута испоруке и у анализи података за повезивање сложених система. Њихова примена омогућава ефикасније доношење одлука и решавање сложених проблема.

Проблеми на графовима често укључују проналажење одређених путања, као што је најкраћа путања између два чвора. Један од најефикаснијих алгоритама за овај задатак и уједно један од најважнијих алгоритама вештачке интелигенције је алгоритам A* (А звезда). Алгоритам A* користи хеуристике (процене које воде претрагу ка циљу на основу приближних, али информативних података о могућем растојању или трошковима) како би усмерио претрагу ка циљу, испуњавајући, при одређеним условима, два важна својства:

- Потпуност – алгоритам ће увек пронаћи решење ако оно постоји.
- Оптималност – алгоритам гарантује решење са најмањом ценом (нпр. минимална удаљеност, најнижи трошак итд.). Оптимално решење не мора увек бити јединствено!

Алгоритам A* је варијанта алгоритма "Прво најбољи", која даје предност чворовима са најбољом оценом. Функција евалуације f (функција која оцењује квалитет стања) у овом алгоритму има следећу форму:

$$f(n) = g(n) + h(n),$$

где је:

- $g(n)$: тренутно позната цена пута од почетног чвора до чвора н,
- $h(n)$: процењена (хеуристичка) цена најјефтинијег пута од н до циљног чвора.

Док се претрага одвија:

- Вредност $g(n)$ се ажурира како алгоритам открива нове и јефтиније путеве.
- Вредност $h(n)$ остаје непромењена, јер представља процену засновану на унапред одређеној хеуристици.

Одабир добре хеуристике је пресудан за ефикасност алгоритма!

Основни појмови алгоритма A*:

- Затворена листа – листа посећених чворова за које су сигурно већ посећени сви суседи (односно сва непосредно доступна стања).
- Отворена листа – листа посећених чворова за које можда нису посећени сви њихови суседи.
- Родитељски чвор – представља чвор из којег се долази до тренутног чвора.

ОПИС АЛГОРИТМА А*

Улаз: Граф G, полазни чвор, циљни чвор и хеуристичка функција h.
Излаз: Пут од полазног до циљног чвора или неуспех.

- Иницијализуј:
  - Отворену листу са полазним чвором.
  - Затворену листу као празну.
- Докле год има чворова у отвореној листи:
  - Изабери чвор n из отворене листе са најбољом оценом $f(n)$.
  - Ако је n циљни чвор:
    - Врати најкраћи пут реконструишући га уназад од n до полазног чвора.
  - За сваког суседа m чвора n:
    - Ако m није ни у отвореној ни у затвореној листи:
      - Додај m у отворену листу.
      - Означи n као родитеља чвора m.
      - Израчунај $g(m)$ и $f(m)$ и придружи их чвору m.
    - Иначе (ако m већ припада некој листи), али је пут преко n јефтинији:
      - Промени информацију о родитељу чвора m на n.
      - Ажурирај вредности $g(m)$ и $f(m)$.
      - Ако је m у затвореној листи, пребацуј га у отворену.
  - Избаци n из отворене листе и убаци n у затворену листу.
- Ако је отворена листа празна, а циљ није достигнут, закључи да решење не постоји.

а) [8 поена] Претпоставимо да планирате пут од Београда до Бањалуке, али због бројних временских непогода морате да узмете хеуристичке процене са сајта министарства саобраћаја и трговине како бисте прилагодили план пута тренутним условима. Уколико су дате удаљености између путем повезаних градова, као и хеуристичка процена удаљености датих градова до Бањалуке, одреди:
Пут од Београда до Бањалуке _____
Редослед додавања градова у затворену листу. _____

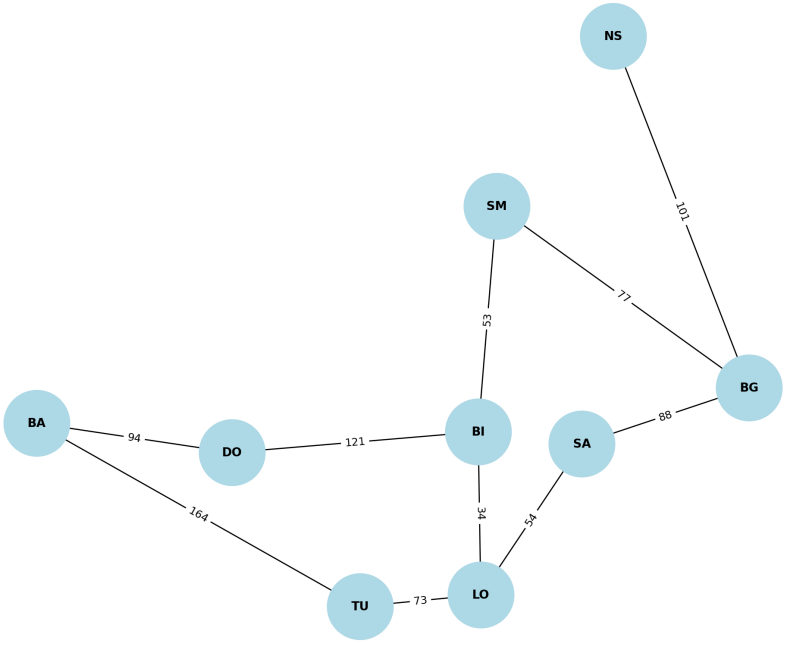
Градови: Београд (БГ), Нови Сад (НС), Сремска Митровица (СМ), Шабац (СА), Лозница (ЛО), Бијељина (БИ), Добој (ДО), Тузла (ТУ), Бањалука (БЛ).

Граф

Чворови	Удаљеност
БГ ↔ НС	101
БГ ↔ СМ	77
БГ ↔ ША	88
ША ↔ ЛО	54
ЛО ↔ БИ	34
ЛО ↔ ТУ	73
ТУ ↔ БЛ	164
СМ ↔ БИ	53
БИ ↔ ДО	121
ДО ↔ БЛ	94

Хеуристичке процене до Бањалуке

Град	Хеуристичка процена
БГ	350
НС	270
СМ	250
ША	210
ЛО	180
БИ	150
ДО	100
ТУ	200
БЛ	0



ХЕУРИСТИКА

Допустива хеуристика - хеуристика (h) која никада не прецењује стварно растојање између текућег чвора и циљног чвора, односно за сваки чвор важи:

$$h(n) \leq h^*(n),$$

где је $h^*(n)$ оптимална хеуристика (цена најкраћег пута од чвора н до циљног чвора).

Уколико се у алгоритму A* користи допустива хеуристика, ако је пронађен пут до циљног чвора, онда је он сигурно оптималан.

Конзистентна хеуристика - хеуристика (h) је конзистентна ако има вредност 0 за циљни чвор и за било која два суседна чвора n и m важи:

$$c(n, m) + h(m) \geq h(n),$$

где је $c(n, m)$ цена придружене (усмерене или неусмерене) грани (n, m).

Ако је хеуристика конзистентна, онда је она и допустива хеуристика.

Ако се у алгоритму A* користи конзистентна хеуристика h, онда:

- ако је пронађен пут до циљног чвора, онда је он сигурно оптималан,
- за чворове доступне из текућег чвора који су већ у затвореној листи, не треба се проверавати да ли њихова вредност g може да се ажурира.

б) [6 поена] Често се као хеуристичка процена удаљености 2 града користи ваздушна удаљеност, јер је сигурно мања од стварног пута и тиме обезбеђује својство оптималности, док је конзистентност задовољена из неједнакости троугла (јер је збир две странице троугла увек већи од треће). Под претпоставком да су следеће хеуристичке вредности заправо ваздушне удаљености између градова и Бањалуке, одреди нови редослед додавања градова у затворену листу.

Град	Хеуристичка удаљеност (км)
БГ	258
НС	215
СМ	187
ША	198
ЛО	164
БИ	160
ДО	71
ТУ	120