Математичка гимназија

Матурски рад
из програмирања

Хеширање и његова примена у задацима

Ученик
Софија Чебашек, IVд

Ментор
Милош Арсић

Београд, мај 2025.

Садржај

1

Увод

Смештање велике количине података у ограничени меморијски простор је чест проблем у свакодневној употреби рачунарских система.

Посматрајмо познату ситуацију прављења налога за сајт. Незаобилазан корак је и креирање шифре. Због често огромног броја корисника, као и циља да шифра буде јединствена, број различитих шифри је изузетно велик. Структуре у које желимо да сместимо све те податке (нпр. вектор), немају довољно меморијског простора да би се шифре чувале у облику у ком их корисник уноси (нпр. стринг). Зато нам је циљ да пронађемо начин како да запамтимо шифре, тако да се њихова јединственост очува и омогући једноставни приступ сачуваним подацима.

Хеширање нам омогућује да лако приступамо стринговима и ефикасно их поредимо. Поређење стрингова је могуће одрадити помоћу алгоритама грубе силе, поређењем сваког карактера појединачно, из једног стринга, са карактером који се налази на истој позицији у другом стрингу. Сложеност оваквог решења је $O(n)$, где је n дужина краћег стринга. Хеширањем бисмо доделили сваком стрингу вредност (број) на унапред одређен начин, па би поређење два стринга могло да се одради у сложености $O(1)$, једноставним упоређивањем додељених вредности.

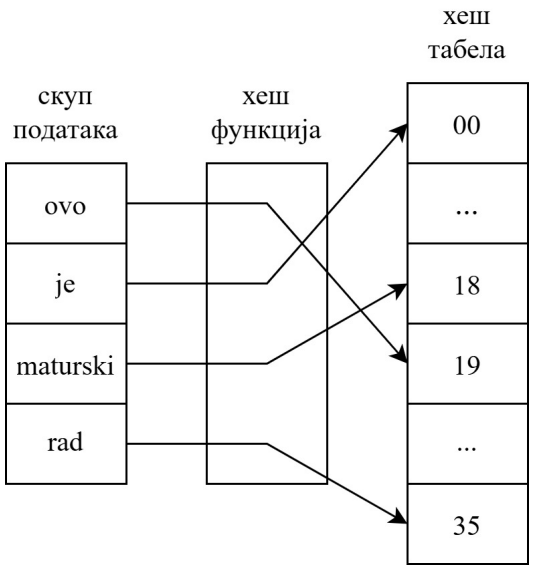
У овом раду представићемо идеју технике хеширања, као и примену кроз примере задатака такмичарског програмирања.

2

Основни појмови

Дефиниција 2.1. Претпоставимо да имамо скуп података величине M и да нам је дато m његових чланова. **Хеш табела** је табела величине n у коју се по унапред одређеним правилима смешта m података, тако да је њима касније могуће приступити у временској сложености $O(1)$.

Дефиниција 2.2. Хеш функција мапира податке из скупа података на позиције $0, \ldots, n-1$ у хеш табели. Вредност коју функција додели податку је његова **хеш вредност**, на основу које одређујемо одговарајућу позицију у табели.



Слика 2.1. Пример хеширања

Дефиниција 2.3. Када хеш функција мапира два различита податка на исту локацију у хеш табели долази до **колизије**.

Циљ је да хеш функција буде таква да хеш вредност сваког податка буде јединствена, вероватноћа да до колизија дође најмања могућа и да равномерно рапспоређује податке у хеш табели.

Претпоставимо да имамо хеш табелу са 100 поља, нумерисаних са $00, 01, \ldots, 99$ и да у њој желимо да чувамо бројеве телефона групе корисника. Нека нам хеш функција враћа последње 2 цифре сваког броја. На основу 2 цифре које функција додељује, податке смештамо у одговарајуће поље хеш табеле, нумерисано тим цифрама. Ова функција је једноставна, али би због великог броја телефонских бројева, дошло до огромног броја колизија, како се много бројева завршава са исте две цифре.

3

Технике хеширања

Хеширање података представља процес додељивања вредности податку уз помоћ хеш функције. У овом поглављу описаћемо неколико начина хеширања стрингова, низова и матрица, чију примену ћемо приказати кроз примере задатака у наставку рада.

3.1 Хеширање стрингова

Један од начина хеширања стрингова јесте користећи **полиномијалну хеш функцију**:

$$h(s) = (s[0] + s[1] \cdot p + \ldots + s[n-1] \cdot p^{n-1}) \mod M,$$

где је s стринг који хешираме, $h(s)$ његова хеш вредност, n његова дужина, $s[i]$ за $i \in \{0, 1, \ldots, n-1\}$ кодови карактера у стрингу (нпр. ASCII кодови или позиција слова у енглеском алфабету), а бројеви p и M унапред одређени позитивни бројеви.

Уколико су хеш вредности два стринга различите, тада се они *сигурно* разликују. Ако су им хеш вредности једнаке, они су *вероватно* једнаки, а вероватноћа да јесу зависи од вредности константи p и M.

Хеш вредности је могуће израчунати уз помоћ низа q:

$$q[i] = p^i, \quad i = 0, \ldots, n-1.$$

Уколико је H помоћни низ за рачунање хеш вредности стринга s, тада је:

$$H[0] = s[0],$$
$$H[i] = H[i-1] + q[i] \cdot s[i], \quad i = 1, \ldots, n-1.$$

Тада је сваки елемент низа $H[i]$ једнак хеш вредности одговарајућег префикса стринга.

Још један начин рачунања хеш вредности целог стринга:

$$H[n-1] = s[n-1],$$
$$H[i] = H[i+1] \cdot p + s[i], \quad i = 0, \ldots, n-2.$$

Хеш вредност целог стринга једнака је елементу низа $H[0]$.

Проблем 3.1. Дати су нам стрингови s и t. Одредити све позиције на којима се стринг s појављује у оквиру стринга t.

Овај проблем је могуће решити помоћу **Рабин-Карп алгоритма**, из 1987. године. Израчунамо хеш вредност стринга s, као и све префиксне хеш вредности стринга t. Пролазимо кроз све позиције стринга t и проверавамо да ли је могуће да позиција буде почетна за стринг s тј. да ли је $t_i t_{i+1} \ldots t_{i+l-1} = s$, $l = |s|$, користећи израчунате хеш вредности.

За почетну позицију поредимо одговарајући префикс стринга t са h, h је хеш вредности целог стринга s. За свако следеће померање почетне позиције коју проверавамо, неопходно је урадити следећи корак: $h = h \cdot p$. На овај начин смо обезбедили да почетни степен од ког рачунамо хеш вредности неког дела стринга t и стринга s буду једнаки и да је упоређивање могуће одрадити. Уколико смо нашли место на ком се стринг s појављује, неопходно је да важи:

$$H[i+l-1] = h, \quad i = 0,$$
$$H[i+l-1] - H[i-1] = h, \quad i = 1, \ldots, n-l,$$

где је h израчунато на описани начин. Алгоритам је временске сложености $O(n)$, где је $n = |t|$ и избегава се споро проверавање грубом силом у сложености $O(n^2)$.

Вероватноћа да дође до колизије зависи од вредности параметара p и M. Уколико су p и M различити прости бројеви, постоји модуларни инверз броја p по модулу M, који је потребан да би се израчунала нова хеш вредност неког стринга уколико му обришемо неколико првих карактера.

Пример 3.1. Уколико посматрамо два стринга a и b и сматрамо да су све могуће хеш вредности подједнако вероватне, вероватноћа да дође до колизије је $\frac{1}{M}$ када међусобно поредимо a и b. За $M = 10^9 + 7$ или $M = 10^9 + 9$, које су честе вредности које се користе, та вероватноћа је веома мала.

Пример 3.2. Када поредимо стринг a са стринговима $b_1, b_2, ..., b_n$ вероватноћа да се деси бар једна колизија је $1 - (1 - \frac{1}{M})^n$.

Пример 3.3. Имамо скуп стрингова $a_1, a_2, ..., a_n$ и поредимо свака два међусобно. Вероватноћа да дође до колизије у овом случају је $1 - \frac{M(M-1)...(M-n+1)}{M^n}$. У оваквим ситуацијама долази до **рођенданског парадокса** - у просторији са 23 особе, вероватноћа је $\frac{1}{2}$ да бар 2 особе имају исти датум рођења. Последица овог парадокса је да је у оваквим ситуацијама неопходно да M буде велики број како би број колизија био занемарљив [2].

За полиномијално хеширање стрингова, који садрже само мала слова енглеског алфабета, $p = 31$ даје добре резултате.

3.2 XOR хеширање

Проблем 3.2. Дат нам је низ a дужине n и Q упита облика $l \quad r$, $l \leqslant r$. За сваки упит потребно је одредити да ли је подниз $a[l], a[l+1], \ldots, a[r]$ пермутација тј. да ли се сваки од бројева $1, 2, \ldots, r-l+1$ појављује тачно једном.

Техника **XOR хеширања** је један од начина решавања овог проблема.

Дефиниција 3.1. Логичка операција **XOR (ексклузивна дисјункција)** над два бита враћа резултат тачно (тј. 1) ако је тачно један бит 1. Означава се са $\oplus$, $\hat{}$ или $\veebar$.

a	b	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

Табела 3.1: Дефиниција XOR-а

Ексклузивна дисјункција два цела броја јесте број који се добије када се операција примени на свим битовима.

Свакој вредности v која се појављује у низу доделимо насумичну вредност $val[v]$ и сваки елемент низа пресликамо у нову вредност тј. $A[i] = val[A[i]]$. Да би подниз $a[l], a[l+1], \ldots, a[r]$ био пермутација неопходно је и довољно да:

$$a[l] \oplus a[l+1] \oplus \ldots \oplus a[r] = 1 \oplus 2 \oplus \ldots \oplus (r-l+1).$$

Десну страну једнакости могуће је израчунати пред обраду упита и чувати све вредности $1 \oplus 2 \oplus \ldots \oplus i$ за све $i = 1, \ldots, n$. За леву страну користићемо

низ $pref[i] = a[0] \oplus a[1] \oplus \ldots \oplus a[i]$, за $i = 0, \ldots, n-1$.

$$\begin{aligned} &pref[0] = a[0], \\ &pref[i] = pref[i-1] \oplus a[i], \quad i = 1, \ldots, n-1. \end{aligned}$$

Тада је:

$$\begin{aligned} a[l] \oplus a[l+1] \oplus \ldots \oplus a[r] &= (a[0] \oplus a[1] \oplus ..a[l-1]) \oplus (a[0] \oplus a[1] \oplus \ldots \oplus a[r]) \\ &= pref[l-1] \oplus pref[r], \end{aligned}$$

јер за свако x важи $x \oplus x = 0$.

Посматрајмо један бит. Вероватноћа да дође до колизије између 2 бита је $\frac{1}{2}$. Ако смо за насумичне вредности у које пресликавамо бројеве користили b битова, вероватноћа је 2^{-b}, јер је операција XOR независна за све битове [5].

3.3 Дводимензионално хеширање

Помоћу **дводимензионалног полиномијалног хеширања** могуће је хеширање матрица:

$$h(i,j) = a[i][j] \cdot p^i \cdot q^j \mod M$$

где је $h(i,j)$ хеш вредност поља у i-том реду и j-тој колони чија је вредност $a[i][j]$, а p, q и M позитивне целобројне константе. Хеш вредности подматрица можемо добити дводимензионалним префиксним сумама.

Вероватноћа да дође до колизије рачуна се аналогно као при хеширању стрингова. Функција ће бити квалитетнија, тј. број колизија ће бити мањи, уколико су p, q и M различити прости бројеви и $p \neq q$.

4

Обрада колизија

Саврешене хеш функције нам обезбеђују да приликом чувања података у хеш табели не долази до колизија. Такве функције су или врло компликоване или захтевају велику количину меморијског простора. Због тога, колизије су углавном неизбежне и потребно је наћи ефикасан алгоритам њихове обраде.

Једноставним повећавањем броја поља у табели, број колизија би био смањен. Како то није увек могуће због меморијског ограничења, колизије обрађујемо користећи **одвојено низање** или **отворено адресирање**. Отворено адресирање подразумева тражење другог слободног поља у табели када је поље одређено хеш функцијом заузето. Постоје три начина отвореног адресирања: *линеарно пробирање*, *квадратно проверавање* и *двоструко хеширање*.

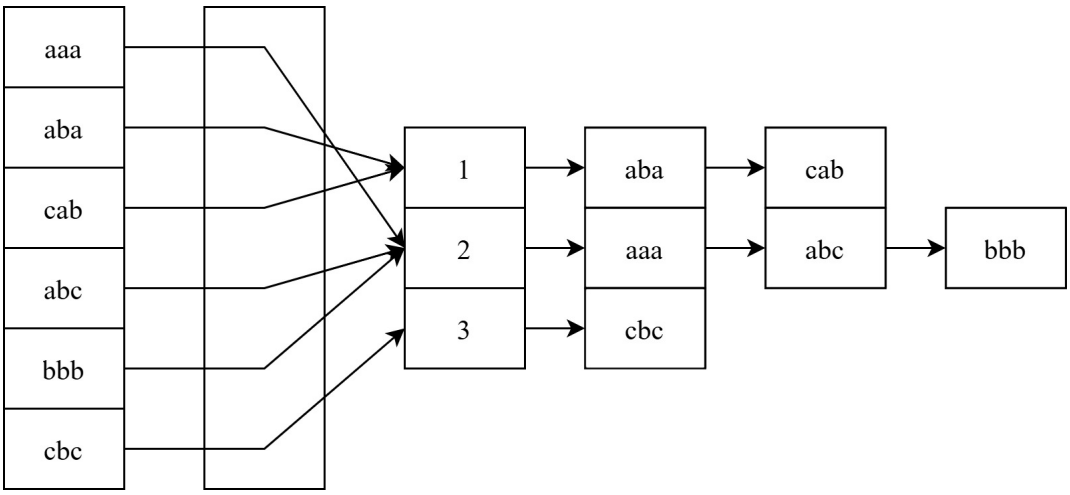
4.1 Одвојено низање

У **одвојеном низању**, као једном од начина обраде колизија, свако поље хеш табеле садржи показивач на уланчану листу у којој се чувају сви подаци који треба да буду смештени у то поље на основу њихове хеш вредности (слика 4.1).

На основу израчунате хеш вредност податка који треба да буде уписан у хеш табелу, налазимо одговарајуће поље. Како би се избегло уписивање исте вредности два пута, линеарно се претражује листа на коју поље има показивач. Податак ће бити уписан само уколико иста вредност није већ сачувана у листи.

Овакав начин обраде колизија је једноставан и прецизан. Једна од главних мана ове технике је што уколико листе у хеш табели садрже велики број елемената, њихово претраживање је значајно спорије, а самим тим и уписивање податка. До овакве ситуације долази када је број података који се чувају велики или је одабрана лоша хеш функција која често ствара колизије. Додатно,

одвојено низање захтева и доста меморијског простора због показивача који се чувају.



Слика 4.1. Пример одвојеног низања

4.2 Линеарно попуњавање

Уколико податак x желимо да сместимо у хеш табелу на позицију $h(x)$ која је већ заузета, **линеарним попуњавањем** податак смештамо на прву следећу слободну позицију хеш табеле тј. $f(x,i) = h(x) + i \mod n$, где је n величина хеш табеле, i најмањи број за који је одговарајуће поље слободно, а $h(x)$ хеш функција.

Овакав начин обраде колизија је ефикасан уколико је табела ретко попуњена, јер би у супротном долазило до *секундардних колизија*. Приликом секундардних колизија долази до колизија између података са различитим вредностима хеш функције $h(x)$. Након великог броја секундарних колизија на неком одсечку табеле, тај део табеле постаје густо попуњен и долази до *ефекта груписања*, што касније чешће доводи до секундарних колизија.

Један од начина да се ублажи ефекат груписања јесте промена вредности корака који се прави приликом тражења првог слободног поља. У i-том кораку проверавамо да ли је слободно поље са индексом $f(x,i) = h(x) + i \cdot c \mod n$. Да би се сигурно обишла сва поља табеле, број c се бира тако да је $(c,n) = 1$. Уколико се не би обилазила сва поља табеле, могуће је да међу пољима која се обиђу не постоји слободно, али је неко од поља која нисмо проверили слободно.

Када табела више није ретко попуњена, тражење слободних поља табеле је спорије и обавља се у линеарној временској сложености.

4.3 Квадратно проверавање

Још један од начина обраде колизије је **квадратно проверавање**. Уводимо функцију $p(x, i) = c_0 i^2 + c_1 i + c_2$. Уколико је поље $h(x)$ заузето и податак x се не може сместити у то поље, тражимо прво следеће слободно поље чији је индекс $h(x) + p(x, i) \mod n$ за $i \in \{1, 2, \ldots, n\}$, n је величина табеле. Оваквим алгоритмом не долази до ефекта груписања, као при линеарном попуњавању.

Када уписујемо нови податак, при квадратном проверавању се не проверавају сва поља у табели и постоји могућност да сва слободна поља буду прескочена. Посматрајмо ситуацију у којој је одабрана функција $p(x, i) = i^2$, а хеш табела је величине 3 тј. њена поља су индексирана бројевима 0, 1 и 2. Уколико неки податак треба да буде уписан на поље 0, које је заузето, на основу функције $p(x, i)$ биће проверавана само поља 0 и 1, док поље са индексом 2 никада неће бити опција за нови податак. До оваквих проблема може доћи и у већим хеш табелама, уколико није изабрана добра функција $p(x, i)$.

4.4 Двоструко хеширање

Нека је $h(x)$ хеш функција и уведимо још једну помоћну $h'(x)$. Када дође до колизије за вредност $h(x)$, проверавају се редом поља $h(x) + i \cdot h'(x) \mod n$, где је n величина табеле, а $i \in \{1, 2, \ldots, n\}$. Податак се уписује на прво слободно поље. Овакав начин обраде колизија назива се **двоструко хеширање** и ефикасније решава ефекат груписања од квадратног проверавања.

Неопходно је да за функцију $h'(x)$ никада не важи $h'(x) = 0$. Како би се проверила сва поља хеш табеле, функција $h'(x)$ се бира тако да вредности која та функција узима буду узајамно просте са величином табеле (често се за величину табеле n бира прост број).

5

Примене хеширања

Хеширање има веома широку примену, а у овом поглављу ћемо укратко описати неколико најважнијих.

Једна од најчешћих примена хеширања јесте приликом чувања лозинки. Свака лозинка се хешира и чува се њена хеш вредност. На тај начин су лозинке безбедније сачуване, јер је само од хеш вредности тешко доћи до почетне лозинке.

Хеширање може да се користи када је неопходно да податке чувамо и омогућимо брз приступ њима касније, као нпр. у различитим базама података. Додатно, хеширана вредност података је често меморијски мање захтевна од самих података.

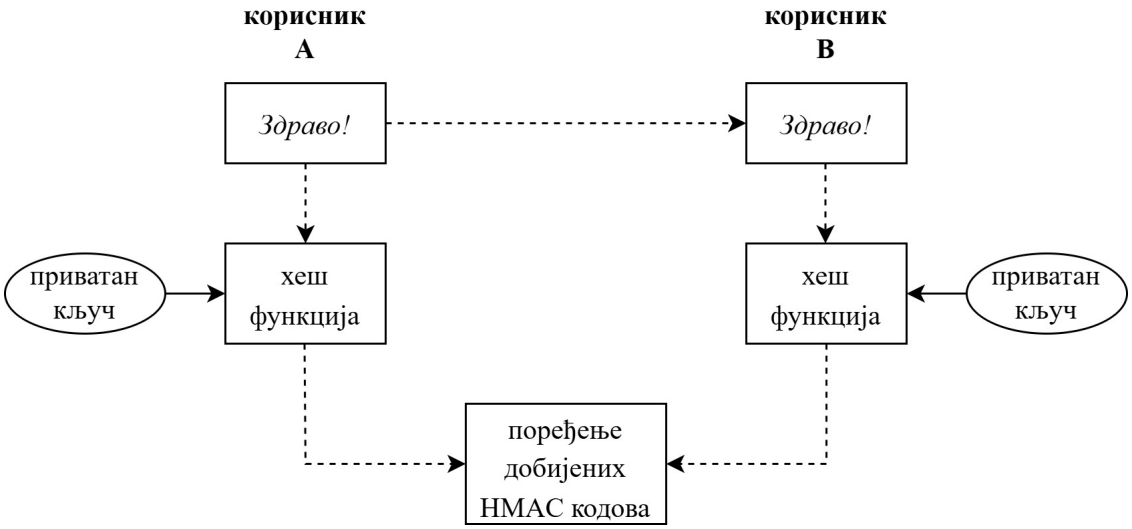
Хеширање се користи и за имплементацију уграђених структура података као што су unordered_set и unordered_map у програмском језику C++. На овај начин је омогућено да се додавање, брисање и тражење елемената у структурама обавља у временској сложености $O(1)$.

Криптографија је наука која се бави начинима чувања и измене доступних података тако да се прикрије њихов садржај. Постоје **криптографске хеш функције** које се користе за хеширање порука приликом провере њихове аутентичности. Неке познатије криптографске хеш функције су SHA-1, SHA-3, SHA-256, MD5...

Дигитални потпис се користи приликом слања порука као доказ да иза поруке која је стигла особи B стоји пошиљалац A. Порука која се шаље се прво хешира и додели јој се приватан кључ који је доступан само особи A. Помоћу хеш вредности поруке и приватног кључа, добијамо дигитални потпис.

Дигитални потпис и порука стижу до особе B. Уз сваки приватан кључ постоји и одговарајући јавни кључ. Различити алгоритми уз помоћ јавног кључа и дигиталног потписа израчунају хеш вредност поруке корисника A и пореде је са хеш вредношћу коју израчунају за испоручену поруку. На тај начин се проверава аутентичност поруке.

Још један начин аутентификације порука и провере интегритета њиховог садржаја, је уз помоћ стандарда **Hash-based Message Authentication Codes (HMAC)**. Корисник A уз помоћ хеш вредности и приватног кључа прави код поруке. Корисник B, који прима поруку, има приступ истом приватном кључу, рачуна хеш вредност исте поруке и добија код који пореди са оним који је добио од A. Приступ приватном кључу имају само особе A и B како би се поруке безбедно преносиле, јер свако ко има приватан кључ може правити кодове порука.



Слика 5.1. Приказ креирања и поређења HMAC кодова

6

Примери задатака

Кроз наредних пар задатака са разних информатичких такмичења, биће приказан начин како да се задатак реши употребом техинке хеширања. За неке задатке постоје и друга решења. За све задатке подразумевају се стандардна временска и меморијска ограничења.

6.1 Задатак 1 [8]

(*Codeforces*)
Дат је стринг s. Свако слово енглеског алфабета је или *добро* или *лоше*. Потребно је наћи број различитих подстрингова стринга s који не садрже више од k *лоших* слова.

Улаз

У првој линији стандардног улаза уноси се стринг s. У другој линији се уноси бинарни стринг који се састоји од 26 карактера. Уколико је i-ти карактер 0, i-то слово енглеског алфабета је *лоше*, а у супротном је *добро*. У последњој линији улаза се налази број $k, 0 \leqslant k \leqslant |s|$.

Излаз

Потребно је у једном реду исписати број различитих тражених подстрингова.

Ограничења

$1 \leqslant |s| \leqslant 1500$

Анализа решења

За сваку позицију памтићемо колико има *лоших* карактера у префиксу стринга до те позиције. Такође, памтићемо префиксне хеш вредности за све позиције на 2 начина, тј. рачунаћемо их користећи 2 различита фактора. За све подстрингове стринга s проверавaћемо да ли садрже највише k *лоших* слова користећи информације о томе колико их има у префиксима стринга. Уколико подстринг испуњава услов, у вектору памтимо пар његових хеш вредности. Ако је p фактор са којим рачунамо хеш вредност, i позиција на којој почиње посматрани подстринг, а n дужина стринга s, хеш вредност коју чувамо ћемо да помножимо са p^{n-i}. Након тога, можемо да поредимо хеш вредности 2 различита подстринга, слично Рабин-Карп алгоритму. Вектор у коме смо памтили парове хеш вредности за све одговарајуће подстрингове сортирамо и налазимо колико различитих парова садржи.

Временска сложеност: $O(n^2 \log n)$.

6.2 Задатак 2 [9]

(*Codeforces*)
Дат је граф са n чворова и m грана. Потребно је одредити број парова чворова i и j који испуњавају следећи услов: сваки чвор k ($k \neq i, k \neq j$) је или сусед и чвору i и чвору j или није суседан ни i ни j.
Између свака два различита чвора постоји највише једна грана и не постоји грана чија су оба краја у истом чвору.

Улаз

У првој линији стандардног улаза дати су бројеви n и m. У наредних m редова дато је по два броја u, v који представљају грану између та два чвора.

Излаз

Потребно је исписати тражени број парова.

Ограничења

$1 \leqslant n \leqslant 10^6, 0 \leqslant m \leqslant 10^6$

Анализа решења

Сваком чвору i ћемо доделити насумичан број $a[i]$. За сваки чвор ћемо израчунати и вредност $val[i]$ која представља број који се добије примењивањем операције XOR над свим вредностима $a[j]$ свих суседа j чвора i. Да би пар чворова (i, j) испуњавао услов задатка, неопходно је да важи:

$$val[i] \oplus a[j] = val[j] \oplus a[i],$$

уколико постоји грана између та два чвора. Уколико не постоји грана између њих, онда мора важити $val[i] = val[j]$. Помоћу структура података као што је нпр. мапа, могуће је наћи број ових парова.

Временска сложеност: $O(n \log n + m)$.

6.3 Задатак 3 [10]

(Румунски мастер из информатике 2017)
Дато је n стрингова дужине k. За сваки од њих, потребно је одредити да ли постоји неки стринг од осталих $n-1$, који се разликује за највише 2 карактера од тренутног. Дато је t независних тест примера.

Улаз

У првој линији стандардног улаза уноси се број t који представља број различитих тест примера. За сваки тест пример се прво уносе бројеви n и k редом. У наредних n редова дат је по један стринг, дужине k.

Излаз

За сваки тест пример, исписати у једном реду бинарни стринг дужине n. У сваком од t стрингова, i-ти карактер има вредност 1 уколико постоји неки други стринг који се разликује од i-тог стринга за највише 2 карактера, а 0 у супротном.

Ограничења

$1 \leqslant t \leqslant 10$
$1 \leqslant n \cdot k \leqslant 3 \cdot 10^4$

Анализа решења

Задатак ћемо решавати одвојено за 2 случаја. Прво ћемо посматрати случај када је $n < k$, у ком задатак можемо решити техником грубе силе. За свака 2 различита стринга, прођемо кроз свих k позиција и бројимо на колико њих се стрингови разликују. За сваки стринг памтимо да ли смо нашли неки који се разликује на највише 2 позиције.
Како је $n < k$, знамо да је $n^2 < n \cdot k$. Одатле следи да је $n < \sqrt{n \cdot k}$.

Временска сложеност за један тест пример: $O(n^2 k)$.

У другом случају је $n \geqslant k$. Можемо унапред фиксирати 2 позиције као једине на којима ће се стрингови разликовати, ако разлике постоје. Уколико су позиције за које тренутно проверавамо услов, l и r, а карактери у стринговима на позицијама $0, \ldots, k-1$, сваки стринг ћемо поделити на 3 дела који садрже узастопне позиције: $\{0, \ldots, l-1\}$, $\{l+1, \ldots, r-1\}$ и $\{r+1, \ldots, k-1\}$. За два стринга потребно је да се поклапају хеш вредности за сва 3 скупа узастопних позиција како би услов био испуњен. За сваки стринг правимо тројке наведених хеш вредности и чувамо их у вектору, заједно са индексом стринга. У вектору, сортираном по тројкама хеш вредности, проверавамо колико има узастопних чланова са једнаким тројкама вредности. Уколико је нека тројка јединствена, за стринг са индексом ind_1 који иде уз ту тројку не постоји стринг који се разликује само на позицијама l и r. У супротном, уколико има више индекса $ind_1, \ldots, ind_m$ са једнаким вредностима одговарајућих тројки, знамо да се сви ти стрингови међусобно разликују на највише 2 позиције. Слично као у првом случају, овде је $k < \sqrt{n \cdot k}$.

Временска сложеност за један тест пример: $O(nk^2 \log n)$.

6.4 Задатак 4 [11]

(Балтичка олимпијада из информатике 2018)
Дато је n стрингова дужине m и сваки карактер стринга је из скупа $\{'A', 'C', 'G', 'T'\}$. Потребно је наћи индекс стринга који се од сваког од преосталих $n-1$ разликује на тачно k позиција.

Улаз

У првој линији стандардног улаза уносе се бројеви n, m и k. У сваком од наредних n редова дат је стринг дужине m.

Излаз

Потребно је у једном реду исписати индекс траженог стринга.

Ограничења

$3 \leqslant n, m \leqslant 4100$

Анализа решења

Сваком од n стрингова доделићемо јединствену, хеш вредност $val[1]$,$val[2]$, $\ldots, val[n]$, која ће бити насумичан број. Сваки стринг ћемо посебно проверити да ли је тражени. Пролазимо кроз свих m позиција у стрингу и нека се тренутно налазимо на позицији i. На укупан збир s ћемо додати вредност сваког стринга који на i-тој позицији има карактер различит од оног у стрингу који тренутно проверавамо. Ову проверу је могуће одрадити грубом силом у временској сложености $O(n)$ за сваку позицију, али је то превише споро. Уместо тога, на самом почетку, пред проверу стрингова, направићемо матрицу $sum[m][4]$, где $sum[i][j]$ представља збир вредности стрингова који на позицији i имају карактер са кодом j (нпр. код $'A'$ је 0, код $'C'$ је 1, код $'G'$ је 2 и код $'T'$ је 3). Уз помоћ ове матрице тражену суму s је могуће наћи у временској сложености $O(1)$ за сваку позицију.

Уколико је тренутни стринг i тражени тј. од сваког од преосталих $n-1$ се разликује на тачно k позиција, неопходно је да важи:

$$s = \sum_{\substack{j=1 \\ j \neq i}}^{n} k \cdot val[j] = k \cdot (S - val[i]), \quad S = \sum_{i=1}^{n} val[i].$$

Како сви стрингови имају насумичне вредности овај услов је и довољан да би стринг i био и тражени, јер је у супротном вероватноћа да овај услов буде испуњен мала.

Временска сложеност: $O(nm)$

6.5 Задатак 5 [12]

(*Hrvatsko otvoreno natjecanje u informatici* 2020/21)
Дата је матрица димензија $n \times m$ која се састоји само од карактера $'.'$ и $'*'$. Потребно је одредити лексикографски најмању матрицу која се добија цикличним померањима свих редова и колона матрице произвољан број пута.

Улаз

У првој линији стандардног улаза дати су бројеви n и m. У наредних n редова дато је по m карактера и сваки је из скупа $\{'.', '*'\}$.

Излаз

Потребно је у n редова исписати по m карактера који представљају тражену матрицу.

Ограничења

$1 \leqslant n, m \leqslant 1000$

Анализа решења

Све матрице које се добијају цикличним померањем дате матрице су подматрице матрице c димензија $2n \times 2m$, која се добије када спојимо 4 копије почетне матрице тј. када:

$$c[i][j] = c[i+n][j] = c[i][j+m] = c[i+n][j+m], \quad 1 \leqslant i \leqslant n, \quad 1 \leqslant j \leqslant m,$$

где је $c[i][j]$ карактер дат на улазу у i-том реду и j-тој колони. Израчунаћемо хеш вредности матрица помоћу дводимензионалног хеширања.

Проверавамо све подматрице димензија $n \times m$ и памтимо позицију лексикографски најмање матрице до сада. Када желимо да проверимо да ли је тренутна матрица лексикографски мања од тренутно најмање, тражимо први ред у ком се матрице разликују помоћу бинарне претраге и израчунатих хеш вредности. Након тога, на сличан начин пронађемо која је прва колона у којој се матрице разликују и упоредимо карактере у одговарајућим пољима.

Временска сложеност: $O(nm \log(nm))$.

Линк ка кодовима за решења задатака.

7

Закључак

У овом раду је описано неколико техника хеширања заједно са честим проблемима у информатици које решавају. Најчешћи проблем ове технике јесу неизбежне колизије. Приказана су четири најпознатија начина њихове обраде и решавања, као и главне предности и мане тих начина.

Хеширање има изузетно широку примену и једна је од најкориснијих техника која се користи у савременим рачунарским системима. Помоћу различитих алгоритама, чувају се подаци који су нам свакодневно потребни. Овом методом омогућена је безбедна комуникација, при којој се чувају интегритет и аутентичност информација које се преносе. У раду је издвојено и укратко описано неколико најчешћих примена методе хеширања.

Кроз примере задатака такмичарског програмирања приказани су начини како да се побољшају хеш функције и повећа вероватноћа да је решење задатка тачно.

Услед убрзаног развоја нових технологија, подаци корисника постају лако доступни и често коришћени без њиховог знања. Развој ове технике би у будућности могао да допринесе безбедности личних података, којој све већи број људи тежи.

Овом приликом се захваљујем професору Милошу Арсићу, професорки Нини Алимпић и свим осталим професорима за све што су ме научили, а што не може стати у овај матурски рад.

Литература

[1] М. Живковић, *Алгоритми*, Математички факултет, Београд

[2] A. Laaksonen, *Competitive Programmer's Handbook*, 2019.

[3] https://cp-algorithms.com/string/string-hashing.html, прегледано 25. 5. 2025.

[4] J. Sannemo, *Principles of Algorithmic Problem Solving*, 2018.

[5] https://codeforces.com/blog/entry/85900, прегледано 25. 5. 2025.

[6] https://research.cs.vt.edu/AVresearch/hashing/index.php, прегледано 25. 5. 2025.

[7] https://usaco.guide/gold/hashing, прегледано 25. 5. 2025.

[8] https://codeforces.com/contest/271/problem/D, прегледано 25. 5. 2025.

[9] https://codeforces.com/contest/154/problem/C, прегледано 25. 5. 2025.

[10] Румунски мастер из информатике 2017: https://csacademy.com/contest/rmi-2017-day-1/task/hangman2/, прегледано 28. 5. 2025.

[11] Балтичка олимпијада из информатике 2018: https://open.kattis.com/problems/genetics2, прегледано 25. 5. 2025.

[12] *Hrvatsko otvoreno natjecanje u informatici* 2020/21: https://oj.uz/problem/view/COCI20_satellti, прегледано 25. 5. 2025.