**№3-LABORATORIYA ISHI**

**3.1. Qidiruv algoritmlari: chiziqli va binar qidiruv.**

**3.2. Satrda qidiruv algoritmlari: KMP-qidiruv, BM-qidiruv, RK-qidiruv.**

**3.3. Hesh-jadval va funksiyani tuzish.**

**3.4. Heshlash algoritmlari va kolliziyaga qarshi kurashish.**

**Ishdan maqsad:** Talabalar berilgan tuzilmaning shakliga qarab biror kalitga mos elementni qidirishning optimal usulini qo‘llashni o‘rganishlari va qidiruv usullarining samaradorligini taqqoslashlari kerak.

**Qo‘yilgan masala**: Topshiriq variantidagi masalani so‘ralayotgan qidiruv usuli yordamida yechishning Python tilidagi dasturini yaratish ko‘nikmasiga ega bo‘lish.

**Ish bajarish tartibi:**

* Tajriba ishi nazariy ma’lumotlarini o‘rganish;
* Berilgan topshiriqning algoritmini ishlab chiqish;
* Python dasturlash muhitida dasturni yaratish;
* Natijalarni tekshirish;
* Hisobotni tayyorlash va topshirish.

**3.1. Qidiruv algoritmlari: chiziqli va binar qidiruv**

***Qidiruv tushunchasi.*** Kompyuterda ma’lumotlarni qayta ishlashda qidiruv asosiy amallardan biri bo‘lib hisoblanadi. Uning vazifasi berilgan argument bo‘yicha massiv ma’lumotlari ichidan mazkur argumentga mos ma’lumotlarni topishdan iborat. Ixtiyoriy ma’lumotlar majmuasi jadval yoki fayl deb ataladi. Ixtiyoriy ma’lumot (yoki tuzilma elementi) boshqa ma’lumotdan biror bir belgisi orqali farq qiladi. Mazkur belgi kalit deb ataladi. Kalit noyob bo‘lishi, ya’ni mazkur kalitga ega ma’lumot jadvalda yagona bo‘lishi mumkin.

Bunday noyob kalitga boshlang‘ich (birinchi) kalit deyiladi. Ikkinchi kalit bir jadvalda takrorlansada u orqali xam qidiruvni amalga oshirish mumkin. Ma’lumotlar kalitini bir joyga yig‘ish (boshqa jadvalga) yoki yozuv sifatida ifodalab bitta maydonga kalitlarni yozish mumkin. Agar kalitlar ma’lumotlar jadvalidan ajratib olinib alohida fayl sifatida saqlansa, u holda bunday kalitlar tashqi kalitlar deyiladi. Aks holda, ya’ni yozuvning bir maydoni sifatida jadvalda saqlansa ichki kalit deyiladi.

Kalitni berilgan argument bilan mosligini aniqlovchi algoritmga berilgan argument bo‘yicha qidiruv deb ataladi. Qidiruv algoritmi vazifasi kerakli ma’lumotni jadvalda topish yoki yo‘qligi aniqlashdan iboratdir. Agar kerakli ma’lumot yo‘q bo‘lsa, u holda ikkita ishni amalga oshirish mumkin:

* ma’lumot yo‘qligini indikasiya (belgilash) qilish;
* jadvalga ma’lumotni qo‘yish.

Faraz qilaylik, k - kalitlar massivi. Har bir k(i) uchun r(i) - ma’lumot mavjud. ***Key*** - qidiruv argumenti. Unga ***rec*** - informasion yozuv mos qo‘yiladi. Jadvaldagi ma’lumotlarning tuzilmasiga qarab qidiruvni bir necha turlari mavjud.

***Chiziqli qidirish algoritmi.*** Berilgan tartiblanmagan massivdagi biror elementni *(kalitni)* topish uchun chiziqli *(ketma-ket)* qidirish algoritmi ishlatiladi. U ham tartiblanmagan, ham tartiblangan massivlar bilan ishlaydi, lekin ikkinchisi uchun chiziqli qidiruvdan ko'ra samaraliroq algoritmlar mavjud.

Bu samarasizlik algoritmning oddiy bajarilishi va uni tartiblanmagan ketma -ketlikda qo'llash qobiliyati bilan qoplanadi. Bu yerda, shuningdek, boshqa barcha qidirish algoritmlarini ko'rib chiqayotganda, ma'lum bir qiymat, algoritm bajarilganda, qator elementlarining qiymatlari bilan taqqoslaganda, kalit vazifasini bajaradi, deb taxmin qilamiz.

Birinchisidan boshlab, massivning barcha elementlari ketma -ket skanerdan o'tkaziladi va kerakli element bilan solishtiriladi. Agar biror qadamda joriy element kerakli elementga teng bo'lsa, u holda element topilgan hisoblanadi va natijada bu element raqami yoki boshqa ma'lumotlar qaytariladi. (Bundan tashqari, mahsulot raqami chiqish bo'ladi). Aks holda, ketma -ketlikda uning yo'qligini ko'rsatadigan biror narsa qaytarilishi kerak.

Quyida, raqamlar misolida, chiziqli qidirish algoritmining ishlashi aniq ko'rsatilgan. Raqam kerakli element sifatida o'rnatiladi (bu holda bu kvadrat) va u o'z navbatida bir xil raqam topilmaguncha mavjud bo'lgan barcha raqamlar bilan taqqoslanadi yoki berilgan raqamda bunday raqam yo'q ekan o'rnatish

Shunisi e'tiborga loyiqki, kalit qiymatiga mos keladigan bir xil qiymatlarga ega bo'lgan bir nechta elementlar bo'lishi mumkin. Bunday holda, agar aniq shartlar bo'lmasa, siz, masalan, natijada birinchi topilgan elementning sonini olishingiz mumkin (quyida keltirilgan ro'yxatda ko'rsatilgan) yoki barcha bir xil elementlarning raqamlarini qatorga yozishingiz mumkin.

***C ++ dastur kodi:***

| *#include <iostream>*  *#include <cstdlib>*  *#include <ctime>*  *using namespace std;*  *int main() {*  *setlocale(LC\_ALL, "rus");*  *int ans[20]; // barcha indekslarni yozish uchun massiv yaratildi*  *int h = 0;*  *int arr[20]; // 20 ta elementdan iborat massiv yaratildi*  *int key; // kalit o'zgaruvchini yaratildi*  *srand ( time(NULL) );*  *for (int i = 0; i < 20; i++) {*  *arr[i] = 1 + rand() % 20; // tasodifiy sonlarni to'ldirish*  *cout << arr[i] << " ";*  *if (i == 9) {*  *cout << endl;*  *}*  *}*  *cout << endl << endl << "Kalitni kiriting: "; cin >> key; // kalitni o'qing*  *for (int i = 0; i < 20; i++) {*  *if (arr[i] == key) { // arr [i] kalitga tengligini tekshiring*  *ans[h++] = i;*  *} }*  *if (h != 0) { // mosliklar borligini tekshiring*  *for (int i = 0; i < h; i++) {*  *cout << "Kalit " << key << " indeks " << ans[i] <<" da" << endl; //barcha indekslarni ko'rsatish*  *} }*  *else {*  *cout << "Biz kalitni topa olmadik!!! " << key << " massivda"; }*  *system("pause");*  *return 0;*  *}* |
| --- |

**Pythonda dastur kodi:**

*a=map(int,input('massiv elementlarini kiriting: ').split())*

*a=list(a)*

*key=int(input('Qidirilayotgan elementni kiritng: '))*

*def chiz\_qid(a,key):*

*for x in range(len(a)):*

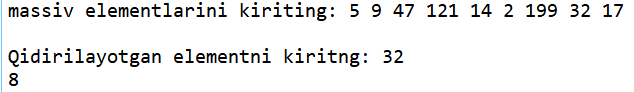
*if a[x]==key:*

*return x+1*

*return -1*

*print(chiz\_qid(a,key))*

**Natija:**



***Bu yerda:***

*List() – ro‘yxat*

*Key – qidirilayotgan kalit*

*Len() – elementlar soni*

***Binar qidiruv algoritmi.*** Faraz qilaylik, o’sish tartibida tartiblangan sonlar massivi berilgan bo’lsin. Ushbu usulning asosiy g’oyasi shundan iboratki, tasodifiy qandaydir element olinadi va u X qidiruv argumenti bilan taqqoslanadi. Agar =X bo’lsa, u holda qidiruv yakunlanadi; agar < X bo’lsa, u holda indekslari M dan kichik yoki teng bo’lgan barcha elementlar kelgusi qidiruvdan chiqarib yuboriladi.

Bu algoritm rekursiv va norekursiv ko’rinishida bo’lishi mumkin. Qidiruvning qadamlar soni

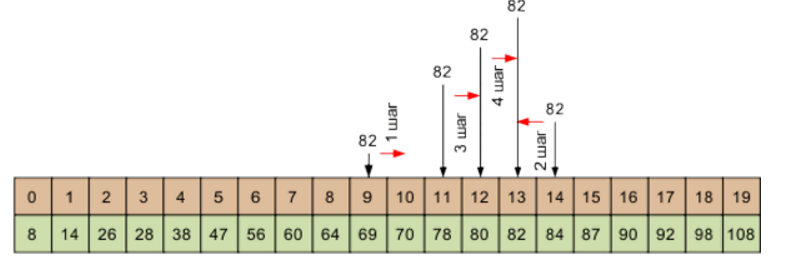
*log2n↑*

Bu yerda, *n* -qadamlar soni, ↑ - yaxlidlanganda kata tomonga o’tish

Har bir qadamni tekshirish formulasi

*mid = (left - right)/2*

Agar qiridilayotgan elementning qiymati *mid*ga teng bo’lsa, qidiruv yakunlanadi.



Massivga chap va o’ng chegaralarini belgilab olamiz *left = 0, right = 19* 1-va oxirgi indeks son qiymatini qo’shib 2 ga bo’lamiz *mid = (19+0)/2=9.* Agar qidirilayotgan element markaziy indeks elementidan katta bo’lsa 69 < 82 qidiruv o’ng tomonga ko’chiriladi va

*left = mid = 9*

Matritsaning o’ng tomonida markaziy indeksga mos element topiladi *mid = (9+19)/2=14* va qidirilayot gan elamaent bilan solishtiriladi:L

*84 > 82*

Bo’lganligi sababli *right = mid = 14*

9 va 14 indekslar bo’yicha markaziy indeksga mos elementni topamiz

*mid = (9+14)/2=11*

11 indeksga mos element qidirilayotgan element bilan solishtiriladi 78 < 82 bo’ladi va left = mid = 11 ga ega bo’lamiz.

11-va 14 indekslar bo’yicha markaziy indeksga mos elementni topamiz

*mid = (11+14)/2=12*

12-indeksga mos element qiymati qidirilayotgan element bilan solishtiriladi 80 < 82

Va jadvalni o’ng tomonini olamiz

*left = mid = 12*

12-va 14 indekslar bo’yicha markaziy indeksga mos elementni topamiz

mid = (12+14)/2=13

markaziy indeksga mos elementni qidirilayotgan element bilan solishtiriladi.

82 = 82

bo’lsa qidiruv yakunlanadi.

Bu algoritmning dasturi C++ dasturlash tilida quyidagicha bo’ladi:

| *#include <iostream>*  *#include <stdio.h>*  *#include <stdlib.h>*  *int main(){*  *int k[20];*  *int r[20];*  *int key, i;*  *k[0] = 8; k[1] = 14;*  *k[2] = 26; k[3] = 28;*  *k[4] = 38; k[5] = 47;*  *k[6] = 56; k[7] = 60;*  *k[8] = 64; k[9] = 69;*  *k[10] = 70; k[11] = 78;*  *k[12] = 80; k[13] = 82;*  *k[14] = 84; k[15] = 87;*  *k[16] = 90; k[17] = 92;*  *k[18] = 98; k[19] = 108;*  *for (i = 0; i < 20; i++) {*  *printf("%2d. k[%2d]=%3d: r[%2d]= ", i, i, k[i], i);*  *scanf("%d", &r[i]); }*  *printf("kalitni kiriting key: ");* | *scanf("%d", &key);*  *int left = 0;*  *int right = 19;*  *int search = -1;*  *while (left <= right) {*  *int mid = (left + right) / 2;*  *if (key == k[mid]) {*  *search = mid;*  *break;*  *}*  *if (key < k[mid])*  *right = mid - 1;*  *else*  *left = mid + 1;*  *}*  *if (search == -1)*  *printf("Element topilmadi!\n");*  *else*  *printf("%d. key= %d. r[%d]=%d", search, k[search], search, r[search]);*  *getchar(); getchar();*  *return 0;}* |
| --- | --- |

**Pythonda dastur kodi:**

*def binar\_qid(lst,key):*

*chap=0*

*ung=len(lst)-1*

*while ung>=chap:*

*mid=(chap+ung)//2*

*if key<lst[mid]:*

*ung=mid-1*

*elif key==lst[mid]:*

*return mid*

*else:*

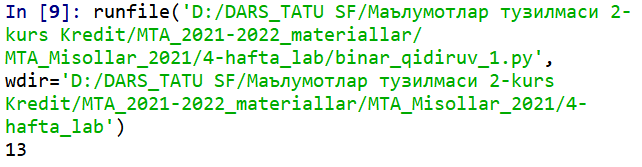
*chap=mid+1*

*return -1*

*lst=[8,14,26,28,38,47,56,60,64,69,70,78,80,82,84,87,90,92,98,108]*

*print(binar\_qid(lst,82))*

**Natija:**



**3.2. Satrda qidiruv algoritmlari: KMP-qidiruv, BM-qidiruv, RK-qidiruv**

***Knut-Morris-Pratt algoritmi*** satrda pastki kalitni (matnni) topish uchun ishlatiladi. Ko'rinib turibdiki, bu oddiyroq bo'lishi mumkin: chiziq bo'ylab harakatlaning va belgilarni ketma -ket matn bilan solishtiring. Mos kelmasa, taqqoslash boshlanishini bir qadam siljiting va qayta solishtiring. Va shunga o'xsxesh, kalit topmaguncha yoki matn oxiriga yetguncha davom ettiramiz.

Funksiya shunga o'xsxesh:

***C++ tilida:***

*// satrdagi matnni oddiy qidirish*

*// satrdagi matn boshining indeksini qaytaradi yoki topilmasa -1*

***int******find****(****char*** *\*nabuna,* ***char*** *\*qator)*

*{*

***for*** *(****int*** *i=0;qator[i];++i) {*

***for*** *(****int*** *j=0;;++j) {*

***if*** *(!namuna[j])* ***return*** *i;*

***if****(qator[i+j]!=namuna[j])* ***break****;*

*}*

*}*

***return*** *-1;*

*}*

Ma'lumot qidirish vazifalarida, eng muhim vazifalardan biri - satrda aniq ko'rsatilgan pastki satrni topishdir. Satrda pastki satrni topishning ibtidoiy algoritmi uzunligi qidiruv matnining uzunligiga teng bo'lgan barcha pastki qatorlarni sanab o'tishga va bunday pastki satrlarni qidiruv namunasi xarakteri bilan taqqoslashga asoslangan. An'anaga ko'ra, qidiruv matnini yoki matnini odatda *needle*, qidiruv chizig'ini esa *haystack* deb atashadi.

***Pythonda*** algoritmi quyidagicha ko'rinishda:

*index = -1*

***for*** *i* ***in*** *xrange(len(haystack)-len(needle)+1):*

*success = True*

***for*** *j* ***in*** *xrange(len(needle)):*

***if*** *needle[j]<>haystack[i+j]:*

*success = False*

***break***

***if*** *success:*

*index = i*

***break***

***print*** *index*

Bu yerda, n = *haystack* , m = *needle* ni bildiradi. Eng oddiy qidirish algoritmi

*n - m + 1* ni eng yaxshi holatda ham taqqoslaydi;

agar bir -biriga o'xsxeshliklar ko'p bo'lsa, *tezlik O (n \* m)* ga tushadi.

Quyida ko'rib chiqilgan algoritm, garchi u "yaxshi" ma'lumotlarning past tezligiga ega bo'lsada, "yomon" ma'lumotlarning regressiyasi yo'qligi bilan qoplanadi. Knuth-Morris-Pratt algoritmi eng yomon chiziqli taxminiy algoritmlardan biridir. Algoritm tavsifiga o'tishdan oldin, prefiks funksiyasi tushunchasini ko'rib chiqish kerak.

π(S, i) satrining *prefiks* funksiyasi bu satrga mos kelmaydigan va shu bilan birga uning qo'shimchasi bo'lgan S [1..i] satrining eng uzun prefiksining uzunligi. Oddiy qilib aytganda, bu mag'lubiyatning eng uzun boshlanishining uzunligi, bu ham ipning oxiri. S qatori uchun prefiks funksiyasini uzunlik | S | -1 vektori sifatida ko'rsatish qulay. | S | uzunlikdagi prefiks funksiyasini π (S, 1) = 0 ni o'rnatish orqali ko'rib chiqishimiz mumkin. "***Abcdabcabcdabcdab***" qatori uchun funksiya prefiksiga misol:

| *S[i]* | *A* | *b* | *c* | *d* | *a* | *b* | *c* | *a* | *b* | *c* | *d* | *a* | *b* | *c* | *d* | *a* | *b* |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *π(S,i)* | *0* | *0* | *0* | *0* | *1* | *2* | *3* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *4* | *5* | *6* |

*Faraz qilaylik*, π (S, i) = k. *Prefiks* funksiyasining quyidagi xususiyatlariga e'tibor bering.

1. Agar S [i + 1] = S [k + 1] bo'lsa, u holda π (S, i + 1) = k + 1.
2. S [1..π (S, k)] - S [1..i] qatorining qo'shimchasi. Haqiqatan ham, agar S [1..i] qatori S [1 ... π (S, i)] = S [1..k] qatori bilan tugasa va S [1..k] qatori bilan tugasa. satr S [1..π (S, k)], keyin S [1..i] qatori ham S [1..π (S, k)] qatori bilan tugaydi.
3. ∀ j∈ (k, i), S [1..j] S [1..i] qatorining qo'shimchasi emas. Aks holda,> (S, i) = k taxminlari noto'g'ri bo'ladi, chunki j> k.

Ko'rib chiqilgan xususiyatlar prefiks funksiyasini hisoblash algoritmini olish imkonini beradi.

*Let (S, i) = k* bo'lsin. *Calculate* *(S, i + 1)* ni hisoblash kerak.

1. Agar S [i + 1] = S [k + 1] bo'lsa, u holda π (S, i + 1) = k + 1.
2. Aks holda, agar k = 0 bo'lsa, u holda (S, i + 1) = 0.
3. Aks holda, k: = π (S, k) qo'ying va 1 -bosqichga o'ting.

Algoritmning mohiyatini tushunishning asosiy nuqtasi shundaki, agar oldingi bosqichda topilgan qo'shimchani keyingi holatga cho'zish mumkin bo'lmasa, biz iloji boricha kichikroq qo'shimchalarni ko'rib chiqishga harakat qilamiz.

Pythonda ***prefiks*** funksiyasini hisoblash algoritmi:

***def******prefix****(s):*

*v = [0]\*len(s)*

***for*** *i* ***in*** *xrange(1,len(s)):*

*k = v[i-1]*

***while*** *k > 0* ***and*** *s[k] <> s[i]:*

*k = v[k-1]*

***if*** *s[k] == s[i]:*

*k = k + 1*

*v[i] = k*

***return*** *v*

Keling, algoritmning ishlash vaqti *O(n)* ekanligini ko'rsataylik, bu erda n=|S|. Shuni esda tutingki, while tsiklining takrorlanishlarining umumiy soni algoritmning asimptotikligini aniqlaydi. Buning sababi shundaki, while tsiklini hisobga olmaganda, for loopning har bir iteratsiyasi doimiy vaqtdan oshmaydi. For loopining har bir iteratsiyasida k birdan oshmaydi, ya'ni maksimal mumkin bo'lgan qiymat k = n - 1. Vaqti -vaqti ichida k ning qiymati kamaygani uchun, k ning jami n - 1 martadan ko'proq kamayishi mumkin emasligi ma'lum bo'ldi. Bu shuni anglatadiki, vaqt tsikli oxirigacha n marta bajariladi, bu esa algoritm O (n) vaqtining yakuniy bahosini beradi.

Prefiks funksiyasidan foydalanishga asoslangan *Knut-Morris-Pratt* algoritmini ko'rib chiqing. Boshlang'ich substring qidirish algoritmida bo'lgani kabi, matn ham moslikni topish uchun chiziq bo'ylab chapdan o'ngga "siljiydi". Biroq, asosiy farq shundaki, prefiks funksiyasidan foydalanib, biz bila turib foydasiz o'zgarishlardan qochishimiz mumkin.

S [0..m - 1] matn, T [0..n - 1] izlanayotgan qator bo'lsin. I pozitsiyadagi satrlarni taqqoslashni ko'rib chiqing, ya'ni S [0..m - 1] matn T [i..i + m - 1] qatorining qismi bilan mos keladi. Faraz qilaylik, birinchi nomuvofiqlik S [j] va T [i + j] belgilari orasida sodir bo'ladi, bu erda i <j <m. P = S [0..j - 1] = T [i..i + j - 1] ni belgilaylik. O'zgartirish paytida, S prefiksi P qatorining bir nechta qo'shimchasi bilan birlashishini kutish mumkin, chunki bir vaqtning o'zida qo'shimchali bo'lgan eng uzun prefiks uzunligi, S indeksining j indeksi uchun prefiks funksiyasi.

Quyidagi algoritmga o'ting:

1. Prefiks S funksiyasi namunasini tuzing, uni F bilan belgilang.
2. $ K = 0, i = 0 $ qo'ying.
3. S [k] va T [i] belgilarini solishtiring. Agar ramzlar teng bo'lsa, k ni 1 ga oshiring. Agar bu holda k matn uzunligiga teng bo'lsa, u holda T chizig'ida S matnining paydo bo'lishi topiladi, paydo bo'lish indeksi i ga teng bo'ladi | S | + 1. Algoritm tugaydi. Agar belgilar teng bo'lmasa, siljishni optimallashtirish uchun prefiks funksiyasidan foydalaning. K> 0 bo'lsa, k = F [k - 1] ni belgilang va 3 -qadamning boshiga o'ting.
4. I <| T | bo'lsa, i ni 1 ga oshiring va 3 -bosqichga o'ting.

Pythonda ***Knut-Morris-Pratt*** algoritmini amalga oshirish mumkin:

***def******kmp****(s,t):*

*index = -1*

*f = prefix(s)*

*k = 0*

***for*** *i* ***in*** *xrange(len(t)):*

***while*** *k > 0* ***and*** *s[k] <> t[i]:*

*k = f[k-1]*

***if*** *s[k] == t[i]:*

*k = k + 1*

***if*** *k == len(s):*

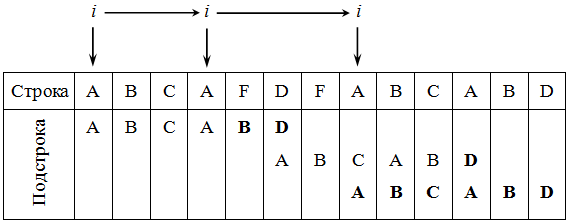
*index = i - len(s) + 1*

***break***

***return*** *index*

***BM-qidiruv algoritmi.*** Boyer-Mur satr qidirish algoritmi umumiy maqsadli algoritmlar qatorida pastki qatorni topishning eng tezkori hisoblanadi. U 1977 yilda Robert Boyer va Jey Mur tomonidan ishlab chiqilgan . Ushbu algoritmning afzalligi shundan iboratki, shablon bo'yicha ma'lum miqdordagi dastlabki hisob -kitoblar uchun (lekin qidirilayotgan satr bo'yicha emas), shablon barcha pozitsiyalardagi manba matn bilan taqqoslanmaydi - ba'zi tekshiruvlar o'tkazib yuboriladi, atayin natija bermaydi.

*Algoritmning asosiy g'oyasi* - qidiruvni boshidan emas, balki pastki satr oxiridan boshlash. Agar biz mos kelmaslikka duch kelsak, biz substringni berilgan belgining eng o'ng tomoniga o'tkazamiz.



Sarlavha rasmidagi misolni ko'rishingiz mumkin. *Knut-Morris-Pratt* misolida biz ham ifodani oldindan kompilyatsiya qilishimiz mumkin.

Buning uchun lug'at juftlariga kalit = belgining raqamli qiymati (ord), qiymat = tartibli pastki satrga kiritish qulay.

Va bu yerda amalga oshirish:

*def bmPredCompil(x):*

*d = {}*

*lenX = len(x)*

*for i in xrange(len(x)):*

*# сколько символов с правого края до этой буквы*

*d[ord(x[i])] = lenX - i*

*return d*

O'zgartirish bilan qidiruvni amalga oshirish qoladi:

*def boyerMurSearch(s, x):*

*d = bmPredCompil(x)*

*# k - s bo'ylab o'tish*

*# j – x bo’ylab o’tish*

*# i - s bo'ylab o'tishning boshlanish joyi*

*lenX = i = j = k = len(x)*

*while j > 0 and i<=len(s):*

*# tasodifan, davom eting (oxiridan boshigacha)*

*if s[k-1] == x[j-1]:*

*k -= 1*

*j -= 1*

*# aks holda, satrni d ga siljiting va pastki satrning o'ng uchidan yana boshlang*

*else:*

*i += d[ord(s[i])]*

*j = lenX*

*k = i*

*if j <= 0:# topildi*

*return k*

return None # *topilmadi*

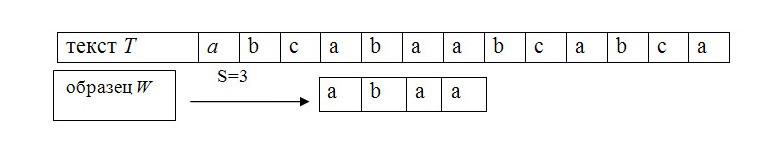
***RK-qidiruv algoritmi.*** Ko'pincha siz ma'lum bir qidiruv bilan shug'ullanishingiz kerak, ya'ni satr qidiruvi (satrda qidirish). Bir nechta T va so'z (yoki rasm) W. bo'lsin. Belgilangan matnda bu so'zning birinchi marta uchrashini topish kerak. Bu harakat har qanday so'zni qayta ishlash tizimiga xosdir. (T va W massivlarining elementlari cheklangan alifbo belgilaridir - masalan, {0, 1} yoki {a, ..., z} yoki {a, ..., z}.)

Bunday topshiriqni bajarishning eng tipik qo'llanilishi hujjatli qidiruvdir: bibliografik ma'lumotlarning ketma -ketligidan iborat hujjatlar to'plami berilgan, har bir murojaatga mos yozuvlar mavzusini ko'rsatuvchi "deskriptor" ilova qilinadi. Ta'riflovchilar orasida topilgan ba'zi kalit so'zlarni topish kerak. Masalan, "Dasturlash" va "Java" so'rovlari bo'lishi mumkin. Bunday so'rovni quyidagicha izohlash mumkin: "Dasturlash" va "Java" deskriptorlari bo'lgan maqolalar bormi.

Ipni qidirish rasman quyidagicha ta'riflanadi. T elementlari T va M elementlari W massivi berilsin, 0 <M≤N. Qatorni qidirish T ning birinchi W ko'rinishini topadi, natijada i indeksi bo'ladi, u satr boshidan (T massivining boshidan) tasvir (so'z) bilan birinchi moslikni ko'rsatadi.

**Misol**. T = abcabaabcabca matnida W = abaa matnining barcha ko'rinishini topmoqchisiz.

Namuna matnga faqat bir marta, S = 3 siljishi, i = 4 indeksi bilan kiritilgan.



To'g'ridan -to'g'ri qidirish algoritmi.

*Algoritm g'oyasi:*

1. I = 1,

2. T massivining I-belgisini W massivining birinchi belgisi bilan

solishtiring ,

3. mos → ikkinchi belgilarni solishtiring va hokazo,

4. mos kelmaslik → I: = I + 1 va 2 -bosqichga o'ting

Algoritm tugashining sharti:

1. ketma -ket M taqqoslash muvaffaqiyatli,

2. I + M> N, ya'ni so'z topilmadi.

*Algoritm murakkabligi****.*** Eng yomon holat. T → {AAA… .AAAB} qatorini, uzunligi │T│ = N, namuna W → {A… .AB}, uzunligini │W│ = M bo'lsin. Shubhasiz, satr oxirida moslikni topish uchun taxminan N \* M taqqoslash kerak bo'ladi, ya'ni O (N \* M).

*Algoritmning kamchiliklari:*

* yuqori murakkablik - O (N \* M), eng yomon holatda - Θ ((N -M + 1) \* M);
* nomuvofiqlikdan so'ng, skanerlash har doim matnning birinchi belgisidan boshlanadi va shuning uchun ilgari ko'rib chiqilgan T belgilarini o'z ichiga olishi mumkin (agar mag'lubiyat ikkinchi darajali xotiradan o'qilgan bo'lsa, unda bunday qaytarish uzoq davom etadi);
* berilgan S smenasini tekshirishda olingan T matni haqidagi ma'lumot keyingi smenalarni tekshirishda hech qanday tarzda ishlatilmaydi.

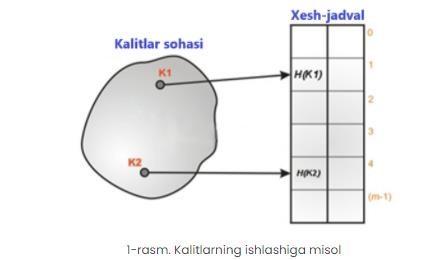
**3.3. Hesh-jadval va funksiyani tuzish**

"Xesh" so'zi ingliz tilidagi «xesh» so’zidan olingan bo’lib, uning ma'nosi “shovqin” yoki “aralash” kabi ta'riflanadi. Aslida, bular atamaning haqiqiy ma'nosini to'liq ifodalaydi.

1. Xesh jadvalni tashkil etish: *asosiy tushunchalari*

*Xeshlash* – bu ma'lum bir turdagi va ixtiyoriy uzunlikdagi [kirish](http://moodle.fbtuit.uz/mod/resource/view.php?id=16223) ma'lumotlari massivini fiksirlangan uzunlikdagi chiquvchi bitlar qatoriga (butun son) aylantirish. Bunday akslantirish (aylantirish) *xesh-funksiya* deb ham ataladi. Xeshfunksiya – bu [kirish](http://moodle.fbtuit.uz/mod/resource/view.php?id=16223) ma’lumotlarini sonlarga aylantiruvchi funksiya bo’lib, bir xil ma’lumotlar to’plami hamma vaqt bir xil natija beradi.

Xesh-jadval – bu elementlari “kalit-qiymat” juftliklari bo'lgan assotsiativ massiv shaklidagi ma'lumotlar tuzilmasi.



*1-rasm. Kalitlarning ishlashiga misol*

Xeshlash assotsiativ massivlarni tashkil qilish uchun qo’llaniladi, bunda indekslari sonlar emas, balki ixtiyoriy qiymatlar bo’ladi. Xeshlashdan odatda matnlardan nusxalarning takrorlanishini qidirishda, ya'ni xesh-funksiyalarining bir xil qiymatiga ega bo'laklarni qidirishda foydalaniladi.

Bundan tashqari, xeshlash ko'pincha parollarni saqlash uchun ishlatiladi; shu bilan birgalikda noyob identifikatorlarni yaratish uchun, masalan, agar fayl o'ziga xos nomni talab qilsa, siz ushbu faylni xeshlash natijasini hisoblab chiqishingiz va natijani faylga nom sifatida ishlatishingiz mumkin.

Ro’yxat:

* Olma - 5000
* Gilos - 20000
* Shaftoli - 5000
* Hurmo - 8000
* Banan - 23000
* Qovun - 15000
* Tarvuz - 10000

Keling bizga kerakli bo’lgan elementni oldingi darsimizda o’rgangan algoritmni tahlil qilib ko’ramiz.

Binary search(ikkilik qidiruv algoritmi) bu algoritm asosida qidirayotgan ma’lumotlar saralangan bo’lishi zarur, saralangan ma’lumotlarni log2(n) vaqtda bajariladi, lekin ma’lumotlar yuqoridagidek saralanmagan bo’lsa chiziqli qidiruv algoritmiga murojaat qilishimizga to’g’ri keladi.

Liner search(Chiziqli algoritm: Bu algoritm asosida har bir elementni solishtirib ko’rishimizga O(n) vaqt ketadi.

Bunday holatlarda Hesh jadvallarga murojaat qilishimizga to’g’ri keladi.

Tasavur qiling siz ro’yxatni yodlab olgansiz, har bir savolga birdaniga javob bera olasiz.Bu amalni bajarish uchun qancha vaqt ketgan deb o’ylaysiz O(1) vaqt.Xuddi shunday qilaylikki kompyuterimiz ham bunday ma’lumotlarni yodlab olsin va biz biror ma’lumot so’raganimizda birdaniga ma’lumotlarni chop etsin ya’ni qidiruvni O(1) vaqtda bajarsin.

Hesh funksiya –matnni biror takrorlanmas kodga aylantirib beradi.Hesh funksiyalarning turi ko’p.

Yaxshi hesh funksiyalari shartlari:

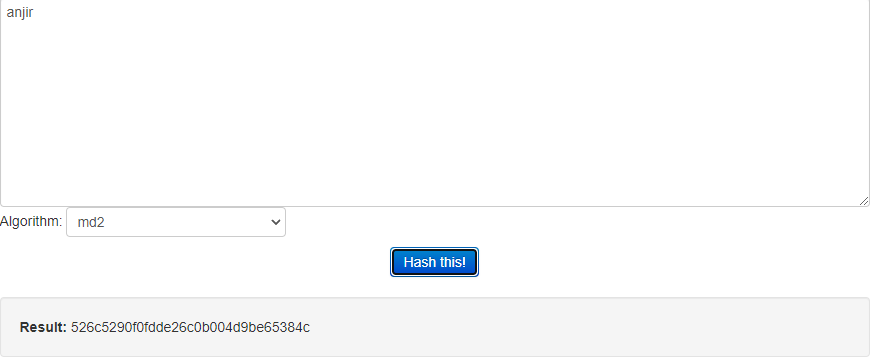
Bir xil matn uchun bir xil kod qaytaradi.



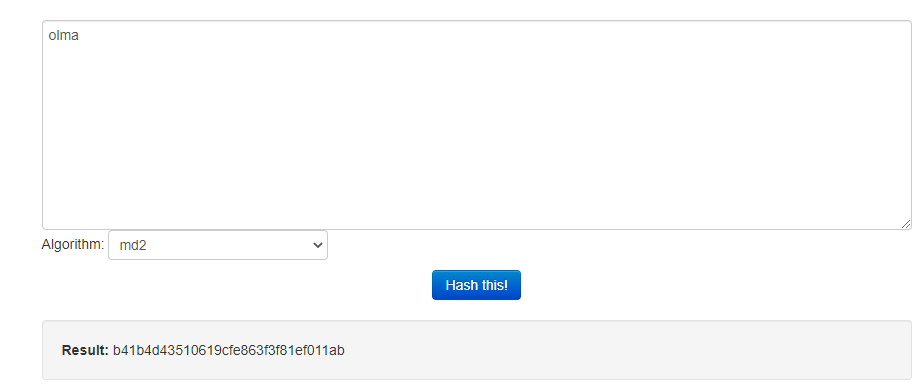
Bunda qachon Hurmo so’zini qidirsak 8000 qiymatni chop etadi ya’ni doim bir xil qiymat qaytaradi.

Xar xil matn uchun xar xil qiymat qaytarishi kerak.

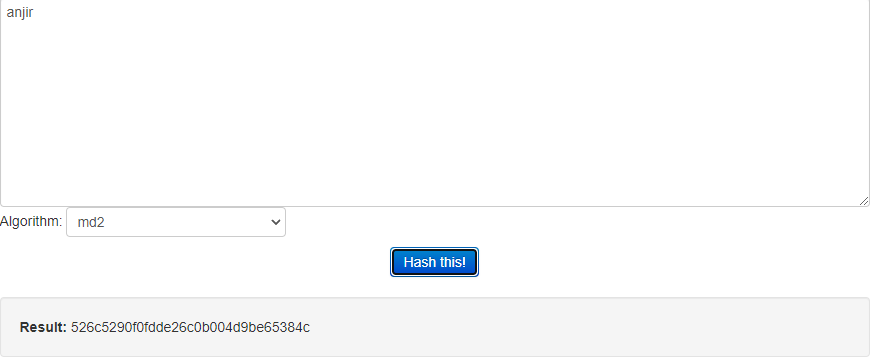
Tushinish uchun <https://www.tools4noobs.com/online_tools/hash/> manzilni ishlatib ko’rishimiz mumkin.



Yuqoridan ko’rinib turibdiki anjir sozini yozganimizdan keyin Result bo’limida matnga mos ravishda 16 sanoq sistemasida son qaytardi.Endi anjir so’zini olma so’ziga o’zgartiramiz.



Endi bunga ham mos ravihda son qaytardi.Agar matnni yana anjir deb yozsak yana yuqoridagi qiymat qaytariladi.



Keling endi hesh jadval tuzishni o’ragamiz. Python dasturlash tilida dict ma’lumot turlari mavjud.

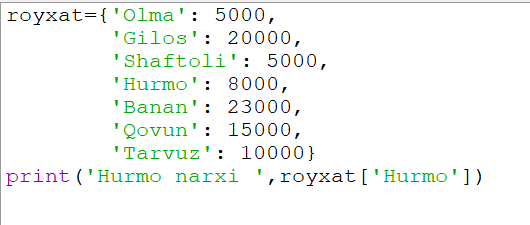
*Royxat=dict()*

Royxatda ma’umotlar quyidagi ko’rinishda saqlanadi.

*Royxat={ ‘Olma’: 5000, ‘Gilos’: 20000, ‘Shaftoli’: 5000,’Hurmo’: 8000, ‘Banan’: 23000, ‘Qovun’: 15000, ‘Tarvuz’: 10000}*

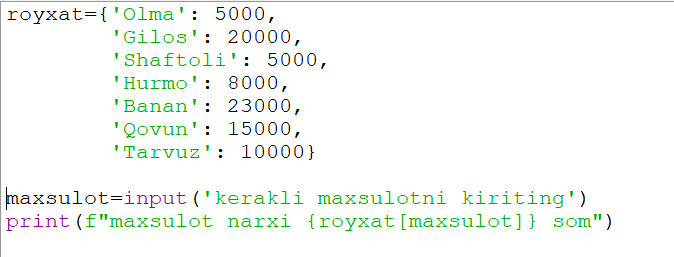
Bu jadvaldagi bizga kerakli ma’lumotni chop etmoqchi bo’lsak quyidagicha amalga oshirmoqchi bo’lsak quyidagicha amalga oshiramiz.

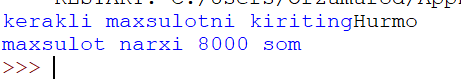
*Print(royxat[‘hurmo’])*



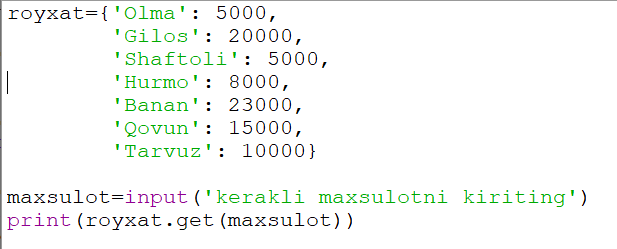


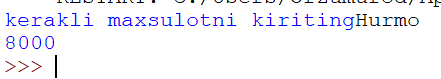
bizga kerakli maxsulotni foydalanuvchi kiitishi kerak bo'lsa,



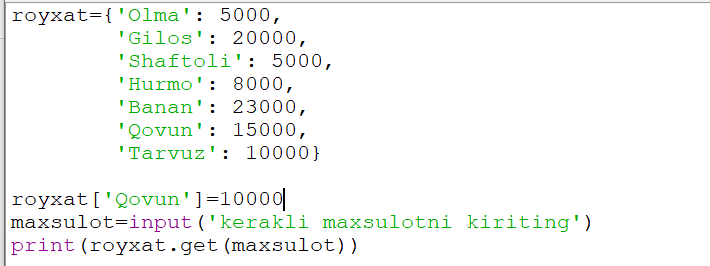


Yoki





Ko’rinishda bajarish mumkin. Jadvalga ma’lumot qo’shish ham quyidagicha amalga oshiriladi. Masalan qovunni jadvalga qo’shmoqchi bo’lsak, *royxat\_nomi[‘maxsulot\_nomi’]=qiymat* kabi qo’shishimiz mumkin bo’ladi.





**3.4. Heshlash algoritmlari va kolliziyaga qarshi kurashish**

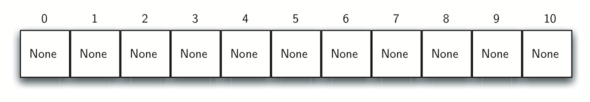
## Oldingi bo'limlarda biz obyektlar bir -biriga nisbatan saqlanadigan joy haqidagi ma'lumotlardan foydalanib, qidiruv algoritmlarini takomillashtirishga muvaffaq bo'ldik. Masalan, ro'yxat tartiblanganligini bilib, biz logarifmik vaqtda ikkilik algoritm yordamida qidirishimiz mumkin. Bu bo'limda biz O (1) vaqt ichida qidirish mumkin bo'lgan ma'lumotlar tuzilmasini yaratish orqali bir qadam oldinga borishga harakat qilamiz. Bu tushuncha hesh deyiladi.

## Endi biz elementni kollektsiyadan qidirganimizda uning o'rnini bilishimiz kerak emas. Agar har bir element bo'lishi kerak bo'lgan joyda bo'lsa, u holda qidiruv faqat taqqoslash yordamida qidirilayotgan elementning mavjudligini aniqlay oladi.

***Hesh jadvali*** - bu ularni keyinchalik osongina topish mumkin bo'lgan tarzda saqlanadigan narsalar to'plami. Xesh-jadvaldagi har bir pozitsiya (ko'pincha slot deb ataladi) o'z elementini va nolga asoslangan butun sonni o'z ichiga olishi mumkin.

Masalan, bizda 0 slot, 1 slot, 2 slot va boshqalar bor. Dastlab, xesh -jadvalda hech qanday element yo'q, shuning uchun ularning har biri bo'sh. Biz har bir element Pythonning maxsus qiymati bilan ishga tushirilgan ro'yxat yordamida hesh -jadvalni amalga oshirishimiz mumkin.

1 -rasmda m = 11 o'lchamdagi hesh jadvali ko'rsatilgan. Boshqacha aytganda, u 0 dan 10 gacha raqamlangan m slotlarga ega.



*1-rasm. 11 ta bo'sh slotga ega bo'lgan hesh jadvali*

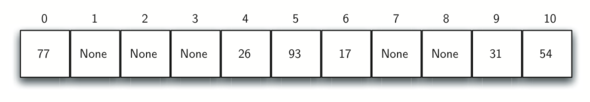
Obyekt va u joylashtirilgan slot o'rtasidagi munosabatlar *hesh funksiyasi* deb ataladi. U to'plamdan istalgan elementni oladi va uyalar nomlari oralig'idagi elementlarni qaytaradi *(0 dan m - 1 gacha).*

Faraz qilaylik, bizda 54, 26, 93, 17, 77 va 31 elementlar to'plami bor. Bizning birinchi hesh funksiyamiz, ba'zida ***"qoldiq usuli"*** deb ataladi, shunchaki elementni oladi va uni jadvalning kattaligiga bo'linadi va qoldiqni qaytaradi. Hesh sifatida qiymatlar *(h(item)=item%11).*

1 -jadvalda bizning misolimizdagi raqamlarning barcha hesh qiymatlari ko'rsatilgan. Qolgan usul (modulli arifmetika) odatda barcha xesh funksiyalarida mavjud, chunki natija slot nomlari oralig'ida bo'lishi kerak.

| **Element** | **Hesh ma’lumoti** |
| --- | --- |
| 54 | 10 |
| 26 | 4 |
| 93 | 5 |
| 17 | 6 |
| 77 | 0 |
| 31 | 9 |

*1-jadval. Qoldiqlardan foydalangan holda oddiy hesh funksiyasi*

Hesh qiymatlarini hisoblash mumkin bo'lganligi sababli, biz 2 -rasmda ko'rsatilgandek, har bir elementni hesh jadvaliga ma'lum bir joyga qo'shishimiz mumkin. E'tibor bering, hozirda 11 ta uyadan 6 tasi band. Bunga yuk koeffitsienti deyiladi va odatda bilan belgilanadi. Bu misolda 

*2-rasm. Oltita elementli hesh jadvali*

Endi, biz elementni topmoqchi bo'lganimizda, biz elementning uyasi nomini hisoblash uchun hesh funksiyasidan foydalanamiz va keyin uning mavjudligini jadvalda tekshiramiz. Bu qidiruv vaqti *O (1),* chunki hesh qiymatini hisoblash va topilgan indeks bo'yicha harakat qilish uchun doimiy vaqt kerak bo'ladi. Agar hamma narsa kerakli joyda bo'lsa, biz qidirish algoritmini doimiy ravishda olamiz.

Siz hozircha sezgan bo'lishingiz mumkinki, bu usul faqat har bir element xesh -jadvaldagi o'ziga xos pozitsiya bilan bog'langanida ishlaydi.

Misol uchun, agar 44 -band bizning to'plamimizda keyingi bo'lsa, u hesh qiymatiga 0 bo'ladi *(44% 11 == 0).* Va 77 hesh qiymati 0 bo'lgani uchun biz muammoga duch keldik. Hesh funksiyasiga ko'ra, ikki yoki undan ortiq elementlar bitta uyaga ega bo'lishi kerak. Bunga **kolleziya** *(ba'zan "to'qnashuv")* deyiladi. Shubhasiz, kolleziyalar heshlash jarayoni uchun muammo tug'diradi. Biz ularni keyinroq batafsil muhokama qilamiz.

## ***Hesh funksiya.*** Berilgan elementlar to'plami uchun har bir elementni o'ziga xos slot bilan bog'laydigan **hesh funksiyasi *ideal hesh funksiyasi*** deb ataladi. Agar bilsak, to'plamdagi hech qanday element o'zgarmaydi, shuning uchun mukammal hesh funksiyasini yaratish mumkin (bu haqda ko'proq ma'lumot olish uchun mashqlarga qarang). Afsuski, ixtiyoriy elementlar majmui uchun mukammal hesh funksiyasini tuzishning tizimli usuli yo'q. Yaxshiyamki, bu bizga samarali ishlashi uchun kerak emas.

Har doim mukammal hesh funksiyasiga ega bo'lishning bir usuli - bu elementlarning mumkin bo'lgan har bir qiymatiga mos keladigan hesh jadvalining hajmini oshirish. Shunday qilib, slotlarning o'ziga xosligi kafolatlanadi. Garchi bu yondashuv oz sonli elementlar uchun amaliy bo'lsada, ularning soni oshgani sari amalga oshishni to'xtatadi. Masalan, to'qqiz xonali ijtimoiy ta'minot slotlari milliardga yaqin slotlarni talab qiladi. Agar biz faqat 25 o'quvchidan iborat ma'lumotni saqlamoqchi bo'lsak ham, biz bunga juda katta xotira sarflaymiz.

Bizning maqsadimiz: kolleziyalarni kamaytiradigan, o'qishni osonlashtiradigan va hesh jadvalidagi elementlarni teng taqsimlaydigan hesh funksiyasini yaratish. Oddiy qoldiq usulini kengaytirishning bir qancha umumiy usullari mavjud. Keling, ulardan ba'zilarini ko'rib chiqaylik.

Hesh funksiyalarni yaratishning ***konvolyutsion usuli***elementni bir xil kattalikdagi komponentlarga bo'lishdan boshlanadi (boshqasi boshqa o'lchamli bo'lishi mumkin). Bu qismlar bir -biriga qo'shiladi va natijada hesh qiymatini beradi.

Misol uchun, agar bizning elementimiz **436-555-4601** telefon raqami bo'lsa, unda biz raqamlarni olib, ularni ikkiga bo'lishimiz mumkin *(43, 65, 55, 46, 01)*. Qo'shishdan keyin *43 + 65 + 55 + 46 + 01* deb olamiz, agar hesh jadvalida 11 sloti bor deb hisoblasak, bu raqamni 11 ga bo'lish va qoldig’ini olish orqali qo'shimcha qadamni bajarishimiz kerak.

Ushbu holatda *210 % 11=1* ga teng, shuning uchun 436-555-4601 telefon raqami 1-slotga joylashtirilgan. Ba'zi konvulsion usullar bir qadam oldinga siljiydi va qo'shishdan oldin bo'lakning har bir qismini aylantiradi. Yuqoridagi misol uchun biz olamiz *43 + 56 + 55 + 64 + 01 = 219* ni beradi *219 % 11 = 10*

Hesh funksiyalarni yaratishning boshqa raqamli usuli ***o'rtacha kvadrat usuli*** deb ataladi. Birinchidan, elementning qiymati kvadratga bo'linadi, so'ngra olingan sonlardan bir qismi chiqariladi. Masalan, agar element 44 bo'lsa, biz avval  ni hisoblaymiz.Ushbu qiymatlar qanday olinganligini tushunganingizga ishonch hosil qiling.

| **Element** | **Qo’shish** | **O'rta kvadrat** |
| --- | --- | --- |
| 54 | 10 | 3 |
| 26 | 4 | 7 |
| 93 | 5 | 9 |
| 17 | 6 | 8 |
| 77 | 0 | 4 |
| 31 | 9 | 6 |

*2-jadval: Ostak va RMS usullarini solishtirish*

Shuningdek, biz belgilar elementlari (satrlar kabi) uchun hesh funksiyasini yaratishimiz mumkin. "Cat" so'zini uning harflari kodlari ketma -ketligi deb hisoblash mumkin.

***>>>*** *ord ( 'c' )*

*99*

***>>>*** *ord ( 'a' )*

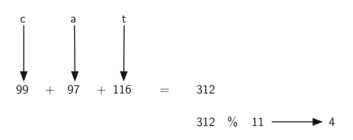
*97*

***>>>*** *ord ( 't' )*

*116*

Keyin siz ushbu uchta kodni olishingiz, ularni qo'shishingiz va hesh qiymatini olish uchun qolgan usuldan foydalanishingiz mumkin ([*3*](http://aliev.me/runestone/SortSearch/Hashing.html#fig-stringhash) -rasmga qarang ).

[*1*](http://aliev.me/runestone/SortSearch/Hashing.html#lst-hashfunction1) -ro'yxat hesh jadval satrini va hajmini oladigan va 0 dan tablesize-1 gacha hesh qiymatini qaytaradigan funksiyani ko'rsatadi .



*3 -rasm: Belgilar kodlari yordamida satrni heshlash*

**Ro'yxat\_1**

***def*** *hesh(astring, tablesize):*

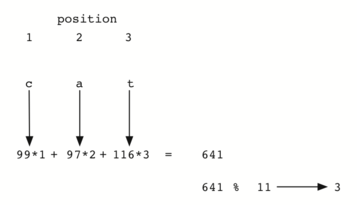
*sum = 0*

***for*** *pos* ***in*** *range(len(astring)):*

*sum = sum + ord(astring[pos])*

***return*** *sum%tablesize*

Qiziqarli kuzatish: bu hesh funksiyasidan foydalanganda, anagramlar har doim bir xil hesh qiymatiga ega bo'ladi. Buni tuzatish uchun belgining pozitsiyasi og'irlik sifatida ishlatilishi kerak.



*4 -rasm: Belgilar kodlari va og'irliklardan foydalangan holda satrni heshlash*

To'plam elementlari uchun hesh qiymatlarini hisoblashning boshqa raqamli usullari haqida o'ylashingiz mumkin. Shuni esda tutish kerakki, samarali hesh funksiyasi saqlash va qidirish jarayonining asosiy qismi bo'lmasligi kerak. Agar u juda murakkab bo'lsa, slotning nomini aniqlash uchun ko'p mehnat talab etiladi. Bunday holda, yuqorida tavsiflangan ketma -ket yoki ikkilik qidiruvdan foydalanish osonroq bo'ladi. Shunday qilib, hesh qilish g'oyasi mag'lubiyatga uchradi.

**Kolliziyaga qarshi kurashish.**

Keling, kolleziyalar muammosiga qaytaylik. Ikkita element bitta slotga joylashtirilganda, biz ikkinchi elementni hesh jadvaliga joylashtirishning tizimli usuliga muhtojmiz. Bu jarayon ***kolleziyani hal*** ***qilish (разрешение коллизий)*** deb ataladi.

Yuqorida aytib o'tganimizdek, agar hesh funksiyasi mukammal bo'lsa, kolleziyalar hech qachon bo'lmaydi. Biroq, bu ko'pincha imkonsiz bo'lgani uchun, kolleziyani hal qilish heshlashning muhim qismiga aylanadi.

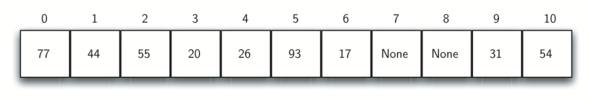
Kolleziyani hal qilish usullaridan biri - bu hesh jadvalga qarash va muammoni keltirib chiqargan elementni joylashtirish uchun boshqa bo'sh slotni qidirishdan iborat. Buning eng oson yo'li - hesh qiymatining asl joyidan boshlash va bo'sh joy topilmaguncha, slotlarni ma'lum bir yo'l bilan aylantirish.

Eslatma: hesh -jadvalni to'liq yopish uchun bizga birinchi slotga (davriy) qaytish kerak bo'lishi mumkin. Bu kolleziyani hal qilish jarayoni ***ochiq manzillar*** deb ataladi, chunki u hesh -jadvaldagi keyingi bo'sh joyni (yoki manzilni) topishga harakat qiladi.

Tizimli ravishda har bir slotga bir marta tashrif buyurib, biz ***chiziqli zondlash*** deb nomlangan ochiq murojaat qilish texnikasiga muvofiq harakat qilamiz .

***5-rasmda*** qolgan hesh funksiyasini qo'llaganidan keyin butun sonli elementlarning kengaytirilgan to'plami ko'rsatilgan ***(54,26,93,17,77,31, 44,55,20).***  Yuqoridagi 1-jadvalda original elementlar hesh qiymatini o'z ichiga oladi, va[*5-shakl,*](http://aliev.me/runestone/SortSearch/Hashing.html#fig-hashtable2) hesh suhbati sovg'alar original mazmunini. 44 -ni 0 -slotga joylashtirmoqchi bo'lsak, kolleziya sodir bo'ladi. Chiziqli tekshirish yordamida biz ketma -ket - slotlar bo'yicha - jadvalni ochiq joyni topguncha skanerlaymiz. Bu holda, bu 1 -slot bo'lib chiqdi.

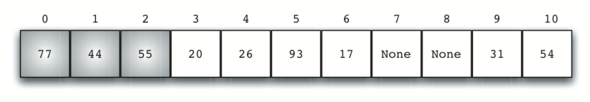
Keyingi safar 0 slotsiga mos keladigan 55, 2 -slotga joylashtiriladi - keyingi bo'sh joy. Oxirgi 20 -raqam 9 -slotga joylashtirilgan. Lekin u band bo'lgani uchun biz chiziqli tekshiruv o'tkazamiz. Biz 10, 0, 1, 2 slotlariga tashrif buyuramiz va nihoyat 3 -pozitsiyada bo'sh slotni topamiz.



*5 -rasm: Chiziqli prob bilan kolleziyani hal qilish*

Biz hesh -jadvalni ochiq manzil (yoki chiziqli tekshirish) yordamida qurganimiz uchun, ob'ektni qidirishda xuddi shu usuldan foydalanish muhim. Faraz qilaylik, biz 93 raqamini topmoqchimiz. Uning hesh qiymatini kiritish 5 ni beradi. Beshinchi slotdan 93 ni topamiz, biz qaytamiz True. Ammo agar biz 20 ni qidirsak nima bo'ladi? Endi hesh qiymati 9 va 9 -slotda 31 bor. Qaytarib bo'lmaydi, False chunki kolleziya bo'lishi mumkin. Shunday qilib, biz 20 -raqam yoki bo'sh slot bo'lganda tugaydigan o'ninchi pozitsiyadan boshlab ketma -ket qidiruvni amalga oshirishga majbur bo'lamiz.

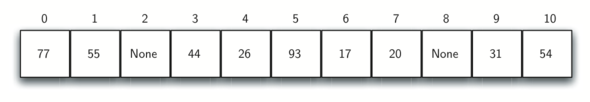
Chiziqli tekshirishning nochorligi uning klasterlash tendentsiyasidir: jadvaldagi elementlar birgalikda guruhlangan. Bu shuni anglatadiki, agar bitta hesh qiymatiga ega bo'lgan kolleziyalar ko'p bo'lsa, chiziqli tekshirish paytida uning atrofidagi bo'shliqlar to'ldiriladi. Bu boshqa elementlarning kiritilishiga ta'sir qila boshlaydi, biz 20 -raqamni jadvalga kiritmoqchi bo'lganimizda, yuqorida ko'rib turganimizdek, bo'sh joyni topish uchun 0 ga qo'yilgan qiymatlar klasterini o'tkazib yuborish kerak. Bu klaster [*6*](http://aliev.me/runestone/SortSearch/Hashing.html#fig-clustering) -rasmda ko'rsatilgan .



*6 -rasm: 0 slot uchun elementlar klasteri*

Klasterlash bilan shug'ullanishning bir usuli - navbatdagi bo'sh joyni ketma -ket qidirish o'rniga, biz bo'sh joylarni o'tkazib yuborish uchun kolleziyani keltirib chiqargan elementlarning yanada tekis taqsimlanishini olish uchun chiziqli tekshirishni kengaytirish. Bu potentsial hosil bo'ladigan klasterni kamaytiradi.

[*7*](http://aliev.me/runestone/SortSearch/Hashing.html#fig-linearprobing2) -rasmda kolleziya aniqlangandan so'ng elementlar plus 3 prob yordamida ko'rsatilgan. Bu shuni anglatadiki, kolleziya sodir bo'lganda, biz bo'sh joy topilmaguncha har uchinchi slotni ko'rib chiqamiz.



*7 -rasm: Plus 3 texnikasi yordamida kolleziyani hal qilish*

Kolleziyadan keyin boshqa slotni topish jarayonining umumiy nomi ***qayta heshlanadi****.* Oddiy chiziqli tekshirishda takrorlangan hesh funksiyasi shunday ko'rinadin ***newheshvalue=rehesh(oldheshvalue)newheshvalue=rehesh(oldheshvalue****bu yerda,****rehesh(pos)=(pos+1)%sizeoftablerehesh(pos)=(pos+1)%sizeoftable.*** "Plyus 3" ning takroriy xeshini quyidagicha ta'riflash mumkin:

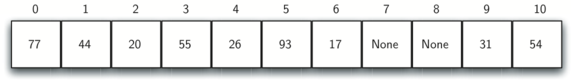
***rehesh(pos)=(pos+3)%sizeoftablerehesh(pos)=(pos+3)%sizeoftable.***

Umuman olganda*:*

***rehesh(pos)=(pos+skip)%sizeoftablerehesh(pos)=(pos+skip)%sizeoftable***

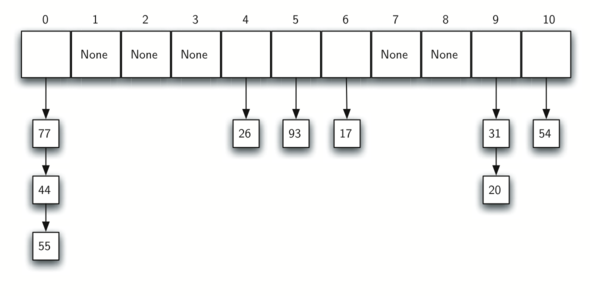
Shuni ta'kidlash kerakki, "o'tkazib yuborish" miqdori oxir -oqibat barcha slotlardan o'tib ketishi kerak. Aks holda, jadvalning bir qismi ishlatilmay qoladi. Bu shartni ta'minlash uchun, odatda, jadvalning kattaligi asosiy raqam deb hisoblanadi. Shuning uchun biz misolda 11 dan foydalanganmiz.

Chiziqli tekshirish uchun yana bir variant - ***kvadratik tekshirish****.* Doimiy "o'tkazib yuborish" qiymatini ishlatishning o'rniga, biz hesh qiymatini *1, 3, 5, 7, 9* va boshqalarga ko'paytiradigan takrorlanadigan hesh funksiyasidan foydalanamiz. Bu shuni anglatadiki, agar birinchi hesh qiymati bo'lsahh, keyin keyingilar bo'ladi *h + 1, h+4, h + 9, h + 16* va boshqalar. Boshqacha qilib aytganda, kvadratik tekshirishda ketma -ket to'liq kvadratlardan iborat bo'shliq ishlatiladi. [*8*](http://aliev.me/runestone/SortSearch/Hashing.html#fig-quadratic) -rasmda ushbu texnikani qo'llaganimizdan keyin bizning misolimizdagi qiymatlar ko'rsatilgan.



*8 -rasm: Kvadratik o'lchov yordamida kolleziyani hal qilish*

Kolleziyalar muammosini hal qilishning muqobil usuli - bu har bir slotga qiymatlar to'plamiga (yoki zanjiriga) havola berishga ruxsat berish. ***Zanjirlar*** bir nechta elementlarga hesh -jadvalda bir xil pozitsiyani egallashga imkon beradi. Qancha ko'p narsalar bir joyga to'plangan bo'lsa, to'plamdagi elementni topish shunchalik qiyin bo'ladi. [*9*](http://aliev.me/runestone/SortSearch/Hashing.html#fig-chaining) -rasmda kolleziya zanjiri yordamida hesh 2-jadvalga elementlar qanday qo'shilishi ko'rsatilgan.



*12 -rasm: Zanjir yordamida kolleziyalarni hal qilish*

Agar biz biror narsani topmoqchi bo'lsak, biz hesh funksiyasidan foydalanib, u joylashtiriladigan slotning raqamini yaratamiz. Har bir slotda to'plam borligi sababli, biz uning mavjudligini aniqlash uchun turli xil qidiruv usullaridan foydalanamiz. Ushbu yondashuvning afzalligi shundaki, har bir slotda kamroq element olish ehtimoli bor, shuning uchun qidiruv samaraliroq bo'ladi. Biz ushbu bobning oxirida batafsil tahlil qilamiz.

Pythonda eng ko'p ishlatiladigan to'plamlardan biri lug'atlardir. Eslatib o'tamiz, lug'at-bu kalit-qiymat juftlarini saqlashi mumkin bo'lgan assotsiativ ma'lumotlar turi. Kalitlar assotsiativ ma'lumotlar qiymatlarini topish uchun ishlatiladi. Biz ko'pincha bu fikrni ***xaritalash*** debataymiz .

Ma'lumotlarning mavhum turini *Map* quyidagicha ta'riflash mumkin. Uning tuzilishi kalitlar va qadriyatlar orasidagi assotsiatsiyalarning tartibsiz to'plamidir. Barcha kalitlar noyobdir, shuning uchun kalitlar va qadriyatlar o'rtasidagi yakka munosabatni saqlab qoladi. Ushbu turdagi ma'lumotlar uchun operatsiyalar quyida keltirilgan:

* *Map()*- Yangi turdagi bo'sh namunani yaratadi. Bo'sh xaritalar to'plamini qaytaradi.
* *Put(key, val)-* Ekranga yangi kalit-qiymat juftligini qo'shadi. Agar bunday kalit allaqachon mavjud bo'lsa, u eski qiymatini yangisiga almashtiradi.
* *Get(key)-* Kalitni qabul qiladi, mos keladigan qiymatini to'plamdan qaytaradi yoki None.
* Del - Ko'rish operatori yordamida displeydan kalit-qiymat juftligini olib tashlaydi del map[key].
* *Len()* - To'plamda saqlangan kalit-qiymat juftlari sonini qaytaradi.
* In – agar berilgan kalit to'plamda key in map bo'lsa yoki boshqa bo'lsa, True ko'rish operatoriga qaytadi  False.

Lug'atlarning katta afzalliklaridan biri shundaki, bizda kalit bo'lsa, biz u bilan bog'liq bo'lgan qiymatni juda tez topa olamiz. Etarli tezlik samarali qidiruvni qo'llab -quvvatlaydigan dasturni talab qiladi. Biz ro'yxatni ketma -ket yoki ikkilik qidiruvdan foydalanishimiz mumkin, lekin yuqorida tavsiflangan hesh jadvalidan foydalanish to'g'ri bo'lar edi, chunki undagi elementni qidirish ishlashga yaqinlashishi mumkin.

Yilda [*ro'yxatlari 2,*](http://aliev.me/runestone/SortSearch/Hashing.html#lst-hashtablecodeconstructor) biz bir sinf yaratish uchun ikki ro'yxatlarini foydalanish *HeshTable* mavhum ma'lumotlar turini o'zida mujassam *Map*. Bitta ro'yxat *slots* elementlarning kalitlarini, parallel ro'yxatda esa *data* ma'lumotlar qiymatlarini o'z ichiga oladi . Biz kalitni topganimizda, u bilan bog'liq qiymat ma'lumotlar ro'yxatidagi tegishli pozitsiyada bo'ladi. Biz ilgari berilgan g'oyalar yordamida hesh -jadval sifatida kalitlar ro'yxati bilan ishlaymiz. E'tibor bering, hesh jadvalining asl o'lchami 11 ga o'rnatiladi. Garchi bu raqam o'zboshimchalik bilan bo'lsa ham, uning oddiy bo'lishi muhim. Bu kolleziyani hal qilish algoritmini iloji boricha samarali qiladi.

**Ro'yxat\_2**

***class******HeshTable****:*

***def*** *\_\_init\_\_(self):*

*self.size = 11*

*self.slots = [None] \* self.size*

*self.data = [None] \* self.size*

*Heshfunction* oddiy qoldiq usulini amalga oshiradi. Kolleziyani hal qilish usuli sifatida "Plyus 1" qayta heshlash funksiyasi bilan chiziqli o'lchesh qo'llaniladi. Funksiya put ( [*3*](http://aliev.me/runestone/SortSearch/Hashing.html#lst-hashtablecodestore) -ro'yxatga qarang ) oxirida bo'sh slotsi borligini yoki bunday kalit allaqachon mavjudligini taxmin qiladi self.slots. U asl hesh qiymatini hisoblab chiqadi va agar bo'sh joy bo'sh bo'lmasa, bo'sh joy topilmaguncha funksiyani qo'llaydi rehesh. Agar bo'sh bo'lmagan slotda kalit bo'lsa, eski ma'lumotlar qiymati yangisiga yoziladi.

***Ro'yxat 3:***

***def*** *put(self,key,data):*

*heshvalue = self.heshfunction(key,len(self.slots))*

***if*** *self.slots[heshvalue] == None:*

*self.slots[heshvalue] = key*

*self.data[heshvalue] = data*

***else****:*

***if*** *self.slots[heshvalue] == key:*

*self.data[heshvalue] = data #replace*

***else****:*

*nextslot = self.rehesh(heshvalue,len(self.slots))*

***while*** *self.slots[nextslot] != None* ***and*** *\*

*self.slots[nextslot] != key:*

*nextslot = self.rehesh(nextslot,len(self.slots))*

***if*** *self.slots[nextslot] == None:*

*self.slots[nextslot]=key*

*self.data[nextslot]=data*

***else****:*

*self.data[nextslot] = data #replace*

***def*** *heshfunction(self,key,size):*

***return*** *key%size*

***def*** *rehesh(self,oldhesh,size):*

***return*** *(oldhesh+1)%size*

Xuddi shunday, funksiya get ( [*4 -ro'yxatga*](http://aliev.me/runestone/SortSearch/Hashing.html#lst-hashtablecodesearch) qarang )boshlang'ich hesh qiymatini hisoblashdan boshlanadi. Agar kerakli qiymat bu slotda bo'lmasa, u *rehesh* keyingi pozitsiyani aniqlash uchun ishlatiladi . E'tibor bering, 15 -satr qidiruvning boshlang'ich slotsiga qaytganimizni tekshirish orqali tugashini ta'minlaydi. Agar bu ro'y bersa, unda barcha mumkin bo'lgan bo'shliqlar tugagan va element to'plamda ko'rsatilmagan.

Sinfning oxirgi usuli*HeshTable* lug'atlar uchun qo'shimcha funksiyalarni ta'minlaydi. Biz elementlarga kirish usullari *\_\_getitem\_\_* va usullarini ortiqcha yuklaymiz . Bu shuni anglatadiki, tanish slot operatori misol uchun mavjud bo'ladi. Qolgan usullarni mashq qilib qoldiramiz.*\_\_setitem\_\_[]HeshTable*

**Ro'yxat 4**

| ***def*** *get(self,key):*  *startslot = self.heshfunction(key,len(self.slots))*  *data = None*  *stop = False*  *found = False*  *position = startslot*  ***while*** *self.slots[position] != None* ***and*** *\*  ***not*** *found* ***and******not*** *stop:*  ***if*** *self.slots[position] == key:*  *found = True*  *data = self.data[position]*  ***else****:*  *position=self.rehesh(position,len(self.slots))*  ***if*** *position == startslot:*  *stop = True*  ***return*** *data*  ***def*** *\_\_getitem\_\_(self,key):*  ***return*** *self.get(key)*  ***def*** *\_\_setitem\_\_(self,key,data):*  *self.put(key,data)* |
| --- |

Keyingi mashg'ulot sinfning HeshTable amaldagi harakatini ko'rsatadi. Birinchidan, biz hesh jadvali yaratamiz va unda bir nechta elementlarni butun sonli kalitlar va mag'lubiyatli ma'lumotlar qiymatlari bilan saqlaymiz.

***>>>*** *H = HeshTable ()*

***>>>*** *H [ 54 ] = "mushuk"*

***>>>*** *H [ 26 ] = "it"*

***>>>*** *H [ 93 ] = "sher"*

***>>>*** *H [ 17 ] = " yo'lbars "*

***>>>*** *H [ 77 ] = " qush "*

***>>>*** *H [ 31 ] = " sigir "*

***>>>****H [ 44 ] = "echki"*

***>>>*** *H[ 55 ] = "cho'chqa"*

***>>>*** *H [ 20 ] = "tovuq"*

***>>>*** *H.slots*

*[77, 44, 55, 20, 26, 93, 17, None, None, 31, 54]*

***>>>*** *H.data*

*["qush", "echki", "cho'chqa", "tovuq", "it", "sher",*

*"yo'lbars", hech biri, hech kim, "sigir", "mushuk")]*

**Topshiriqlar**

1. a) Ketma-ket qidiruv usulidan foydalanib, ro’yxat eng kichik elementini toping?

b) Heshlashning “***метод свёртки”*** algoritmi qanday ishlashini tahlil qiling, o’zingizni F.I.SH. ni hesh qiymatini qaytaruvchi dastur tuzing?

1. a) Ketma-ket qidiruv usulidan foydalanib, ro’yxatda berilgan kalitdan katta elementlarni toping?

b) Heshlashning “***o'rtacha kvadrat”*** algoritmi qanday ishlashini tahlil qiling, o’zingizni F.I.SH. ni hesh qiymatini qaytaruvchi dastur tuzing?

3. a) Mashina raqamlari ro’yxati berilgan: 069, 070, 345, 368, 876, 945, 564, 387, 230, 701, 926, 937, 955. Binar qidiruvdan foydalanib berilgan raqamli mashina qaysi joyda turganini toping?

b) Heshlashda kolleziyani hal qilish ***(разрешение коллизий)*** nima***,*** o’zingizni F.I.SH. ni hesh qiymatini qaytaruvchi dastur tuzing?

4. a) Binar qidiruvdan foydalanib massivdan berilgan kalitga karrali kalitli elementlar sonini toping?

b) Kolleziyalar muammosini hal qilishning muqobil usuli haqida ma’lumot bering, o’zingizni F.I.SH. ni hesh qiymatini qaytaruvchi dastur tuzing?

5. a) Binar qidiruv usuli orqali 1 dan n gacha bo’lgan sonlar ichidan ixtiyoriy elenementni topish dasturini tuzing?

b) Python ***hash()*** funksiyasi haqida ma’lumot bering va misollar keltiring, o’zingizni F.I.SH. ni hesh qiymatini qaytaruvchi dastur tuzing?

6. a) Ketma-ket qidiruv usuli orqali 1 dan n gacha bo’lgan sonlar ichidan ixtiyoriy elenementni topish dasturini tuzing?

b) Pythonda MD5 heshlash haqida ma’lumot bering va misollar keltiring, o’zingizni F.I.SH. ni hesh qiymatini qaytaruvchi dastur tuzing?

7. a) Ketma-ket va binar qidiruv usullari orqali 1 dan n gacha bo’lgan sonlar ichidan ixtiyoriy elenementni topish dasturini tuzing va samaradorligini tahlil qiling?

b) ***Hesh jadvali*** haqida ma’lumot bering va misollar keltiring, o’zingizni F.I.SH. ni hesh qiymatini qaytaruvchi dastur tuzing?

8. a) Ketma-ket qidiruv usuli orqali 069, 070, 345, 368, 876, 945, 564, 387, 230, 701, 926, 937, 955 ixtiyoriy raqamli mashina qaysi joyda turganini toping?

b) Hesh funksiyani yaratish usullarihaqida ma’lumot bering va misollar keltiring, o’zingizni F.I.SH. ni hesh qiymatini qaytaruvchi dastur tuzing?

9. a) Ketma-ket va binary qidiruv usullari orqali 069, 070, 345, 368, 876, 945, 564, 387, 230, 701, 926, 937, 955 ixtiyoriy raqamli mashina qaysi joyda turganini toping va samaradorligini tahlil qiling?

b) Heshlashning ***"qoldiq usuli"*** haqida ma’lumot bering va misollar keltiring, o’zingizni F.I.SH. ni hesh qiymatini qaytaruvchi dastur tuzing?

10. a) Ketma-ket va binary qidiruv usullari orqali ro’yxatdagi eng katta elementni topish dasturini tuzing va samaradorligini tahlil qiling?

b) Heshlashda ***ideal hesh funksiyasi*** haqida ma’lumot bering va misollar keltiring, o’zingizni F.I.SH. ni hesh qiymatini qaytaruvchi dastur tuzing?