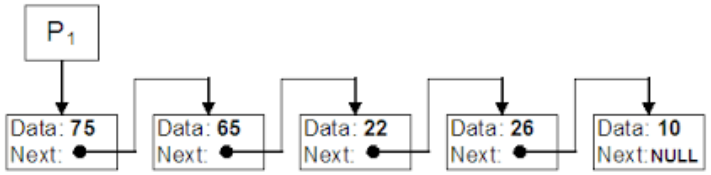
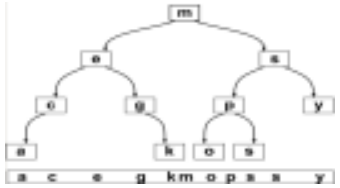


TIQXMMI
Toshkent Irrigatsiya va Qishloq Xo'jaligini Mexanizatsiyalash Muhandislari Instituti

Ma’lumotlar tuzilmasi

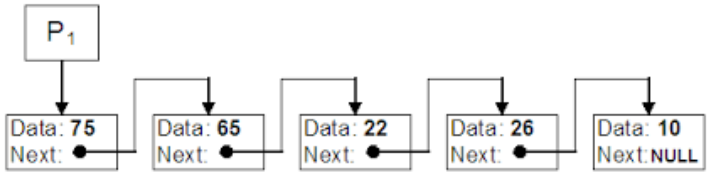
Data structures



4-ma’ruza: Qidiruv va heshlash algoritmlari

Lecture #4. Search and hashing algorithms

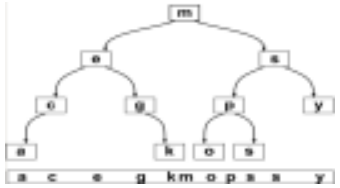
PhD, Sh.A.Toirov



TIQXMMI
Toshkent Irrigatsiya va Qishloq Xo'jaligini Mexanizatsiyalash Muhandislari Instituti

Ma’lumotlar tuzilmasi

Data structures



4-ma’ruza: Qidiruv: chiziqli va binary qidiruv algoritmlari

Lecture #4. Search: linear and binary search algorithms

PhD. Sh.A.Toirov

TIQXMMI
Toshkent Irrigatsiya va Qishloq Xo'jaligini Mexanizatsiyalash Muhandislari Instituti

Ma’ruza rejasi
Plan lecture

- Kalit tushunchasi
- Ketma-ket qidiruv
- Indeksli ketma-ket qidiruv
- Ketma-ket qidiruvni samaradorligi
- Indeksli ketma-ket qidiruvni samaradorligi

Kalit tushunchasi

- Ixtiyoriy ma’lumotlar majmuasi jadval yoki fayl deb ataladi. Ma’lumot (ya’ni, tuzilma elementi) boshqa ma’lumotdan biror ***bir belgisi*** bilan farq qiladi. Mazkur belgi ***kalit*** deb ataladi.
- Tuzilmaning elementlari alohida kalitlarga ega bo’lishi mumkin. Bunday element kaliti boshlang’ich, ya’ni ***birinchi kalit*** deyiladi.
- Elementlarning boshqasidan farq qiluvchi yana bir belgisi va bir nechta elementlarda takrorlanuvchi kaliti ***ikkinchi kalit*** deyiladi.

Kalit tushunchasi

- **Tashqi va ichki kalit tushunchalari**

  Ma’lumotlar kalitini bir joyga yig’ish (ya’ni, alohida boshqa jadvalga yozib qo’yish) yoki yozuvlarda alohida maydonga yozib qo’yish 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.

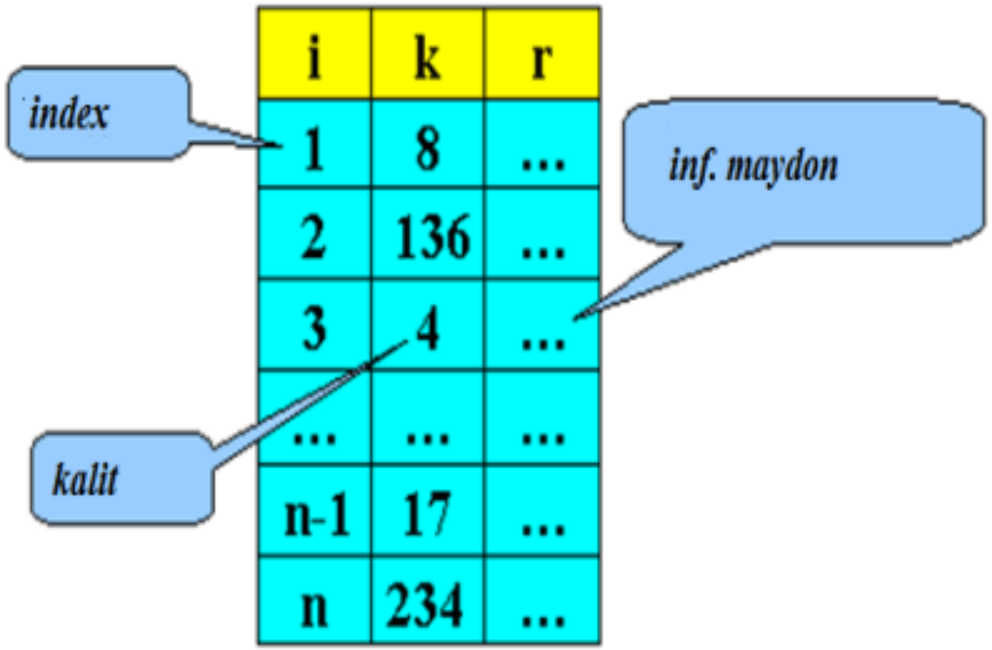
Qidiruv tushunchasi

- Kalitni berilgan argument bilan mosligini aniqlovchi algoritmga berilgan argument bo’yicha ***qidiruv*** deb ataladi.
- Qidiruv algoritmining vazifasi kerakli ma’lumotni tuzilmadan (jadvaldan) topish yoki uning yo’qligi aniqlashdan iborat.
- Agar qidirilayotgan ma’lumot yo’q bo’lsa, u holda quyidagi ikkita vazifani amalga oshirish mumkin:
  - ma’lumot yo’qligini indikatsiya (belgilash) qilish
  - tuzilmaga ushbu ma’lumotni qo’shish.

Ketma-ket qidiruv tushunchasi va algoritmi

- Mazkur ko’rinishdagi qidiruv agar ma’lumotlar tartibsiz yoki ular tuzilishi noaniq bo’lganda qo’llaniladi.
- Bunda ma’lumotlar tuzilmasi butun jadval bo’ylab tezkor xotirada kichik adresdan boshlab, to katta adresgacha ketma-ket qarab chiqiladi.

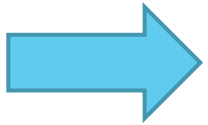
Massivda ketma-ket qidiruv



- Bunda yordamchi ***search*** o’zgaruvchisi topilgan element kalitini saqlaydi
- Massivda ketma-ket qidiruv algoritmining samaradorligini bajarilgan taqqoslashlar soni **M** bilan aniqlash mumkin.
- **Mmin = 1, Mmax = n**.
- Agar elementlar massiv yacheykasida bir xil ehtimollik bilan taqsimlangan bo’lsa, u holda
- **Mo’rt ≈ (n + 1)/2 bo’ladi**

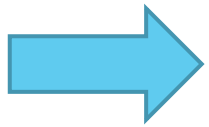
C++ tilida qidiruv algoritmi quyidagicha bo’ladi:

```
int search (int a[], int N, int key)
{
    int i=0;
    while (i!=N)
        if (a[i]==key) return i;
            else i++;
    return -1; }
```



Python tilida qidiruv algoritmi quyidagicha bo’ladi:

```
data=[1,2,3,4,5,6,7]
target=[5]
def Linear_search(data, target):
    for i in range(len(data)):
        if data[i]==target:
            return True
    return False
print(Linear_search(data,target))
```



Ketma-ket qidirish samaradorligi

- Taqqoslashlar soni qanchalik kam bo’lsa, qidirish algoritmining samaradorligi shuncha ***yuqori*** bo’ladi.
- Massivda ketma-ket qidirish samaradorligi quyidagicha:

$C = 1 \div n$, $C = (n + 1)/2$.

- Ro’yxatda ketma-ket qidirish samaradorligi ham xuddi shunday. Ma’lumotlarni massiv va ro’yxat ko’rinishlarda tashkil etishning o’z afzalligi va kamchiliklari mavjud, bog’langan ro’yxatlarda qidirishdagi taqqoslashlar soni ham massivdagi qidirish bilan bir xil bo’ladi.

Ketma-ket qidirish samaradorligi

Qidirish maqsadi quyidagi protseduralarning bajarilishini ta’minlaydi:

1) Topilgan yozuvni o’qish.

2) Yozuvni chiqarish jarayonida uni jadvalga qo’yish.

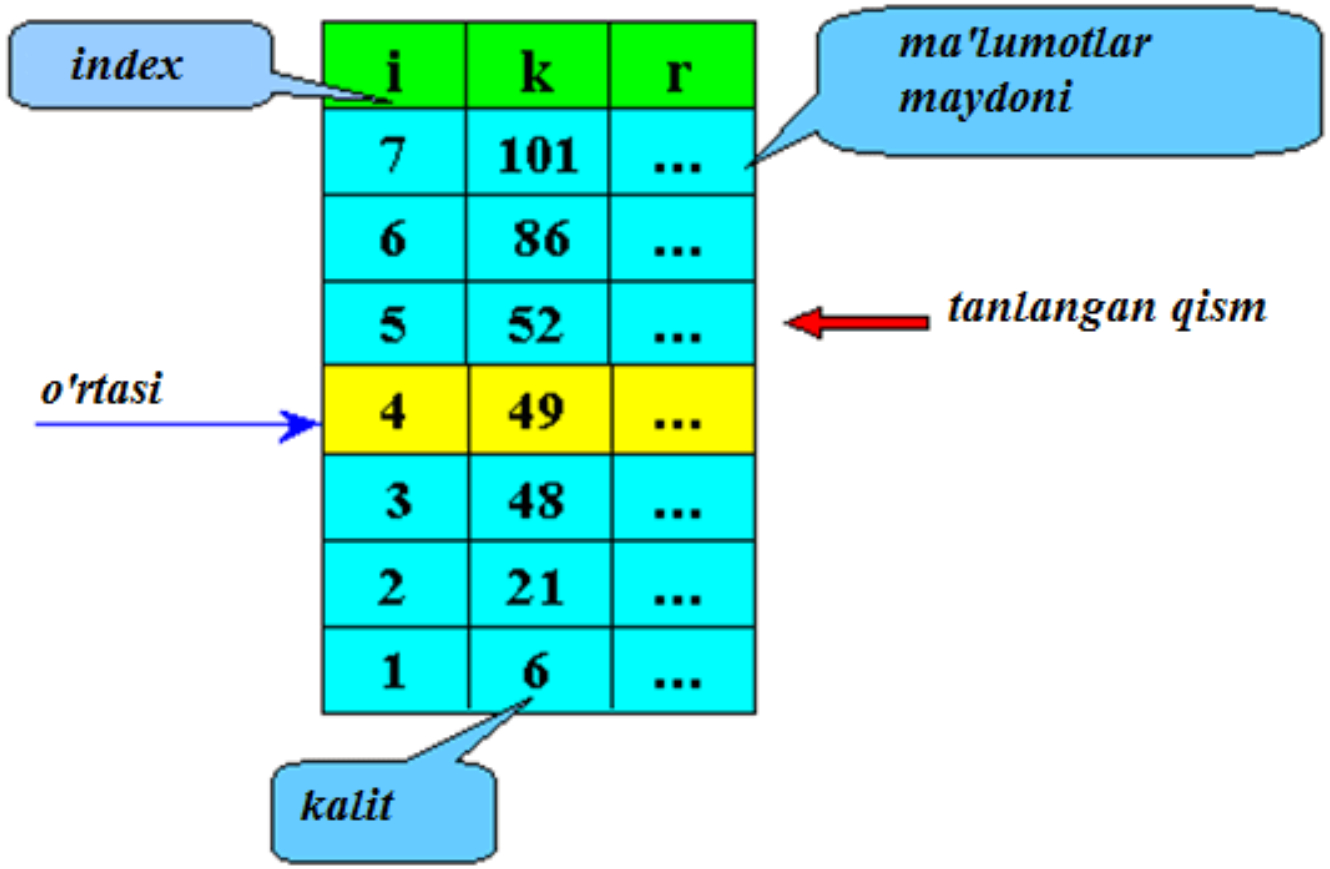
3) Topilgan yozuvni o’chirish.

- Birinchi protsedura qolganlari bilan bir vaqtda bajariladi. Ikkinchi va uchinchi protseduralar ro’yxatli tuzilmalar uchun ancha samaraliroq (massivda elementlarni siljitish kerak).
- Agar massivda elementlarni siljitishlar soni ***k*** ga teng deb olsak, u holda $k=(n+1)/2$.

Ikkilik (binary) qidiruv algoritmi

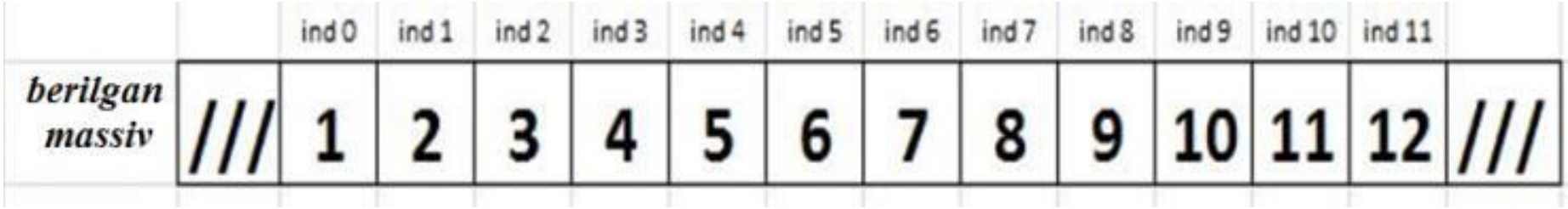
- Biz chiziqli qidiruv algoritmlarini qarab chiqdik. Qidiruvning boshqa algoritmlari ham mavjud.
- Masalan, ikkilik (binar) qidiruv algoritmi.
- Biz yuqorida chiziqli qidiruv algorimtlaridan indeksli ketma-ket qidirish usulini qarab chiqdik. Bunda qayta ishlanayotgan tuzilma elementlarini saralangan bo’lishi kerak edi.
- Xuddi shunday binar qidiruv algoritmi ham saralangan tuzilmalar ustida qo’llanilsa, yuqori samara beradi.

Ikkilik (binary) qidiruv algoritmi



Ikkilik (binary) qidiruv algoritmi

- Faraz qilaylik, bizga 12 ta elementdan iborat, o’sish tartibida saralangan quyidagi massiv berilgan bo’lsin:



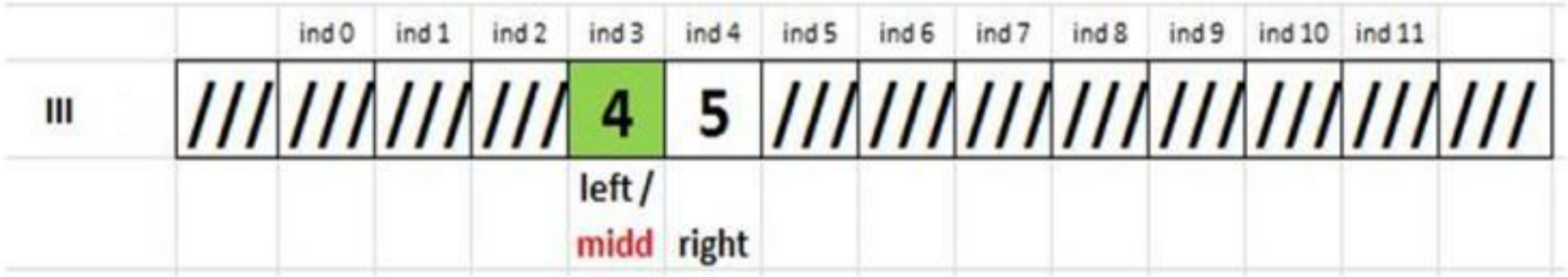
- Foydalanuvchi tomonidan kiritilgan qidiruv kaliti asosida qidirilayotgan elementni topish masalasi qo’yilgan. Masalan, kalit 4 ga teng bo’lsin.

Ikkilik (binary) qidiruv algoritmi

- Birinchi qadamda massiv teng ikkiga bo’linadi (buning uchun o’rta element - **midd** ni aniqlaymiz): (0 + 11) / 2 = 5 (bo’linmaning butun qism olinadi, 0.5 tashlab yuboriladi). Birinchi massiv elementlarining o’rta qiymatini tekshiramiz, agar u kalit bilan mos kelsa, algoritm o’z ishini yakunlaydi va topilgan element haqida axborot beradi. Qaralayotgan misolda o’rta qiymat qidirilayotgan kalitga mos kelmaydi.

Ikkilik (binary) qidiruv algoritmi

- Agar qidirilayotgan kalit qiymatli element o’rta qiymatdan kichik bo’lsa, algoritm o’rta qiymatdan katta elementlar joylashgan qismini tekshirmaydi. Qidiruvning o’ng tomondagi chegarasi (midd - 1) ga joylashadi. Hosil bo’lgan qism massivni yana 2 ga bo’lamiz.
- Qidiruv kaliti yana o’rta elementga teng emas, katta. Endi qidiruvning chap chegarasi (midd + 1) ga joylashadi.

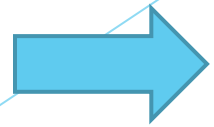


Ikkilik (binary) qidiruv algoritmi

- Agar qidirilayotgan kalit qiymatli element o’rta qiymatdan kichik bo’lsa, algoritm o’rta qiymatdan katta elementlar joylashgan qismini tekshirmaydi. Qidiruvning o’ng tomondagi chegarasi (midd - 1) ga joylashadi. Hosil bo’lgan qism massivni yana 2 ga bo’lamiz.
- Qidiruv kaliti yana o’rta elementga teng emas, katta. Endi qidiruvning chap chegarasi (midd + 1) ga joylashadi.
- Uchinchi qadamda o’rta element 3 indeksli elementga teng:
- (3 + 4) / 2 = 3. U kalitga teng. Algoritm o’z ishini yakunlaydi.

Ikkilik (binary) qidiruv algoritmi C++da

```
int Search_Binary (int arr[], int left, int right, int key) {
 int midd = 0;
 while (1)      {
        midd = (left + right) / 2;
        if (key < arr[midd])       // agar qidirilayotgan element kichik bo’lsa
              right = midd - 1;      // qidiruvning o’ng chegarasini aniqlash
        else if (key > arr[midd])  // agar qidirilayotgan element katta bo’lsa
              left = midd + 1;      // qidiruvning chap chegarasini aniqlash
        else                       // yoki (qiymat teng)
              return midd;          // funktsiya ushbu yacheyka qiymatini qaytaradi
        if (left > right)          // agar chegara mos kelmasa
              return -1;    }    }
```



Ikkilik (binary) qidiruv algoritmi Pythonda

```
data = [1, 2, 3, 4, 5, 6, 7, 10]
target = 10
def binar_search_itereativ(data, target):
    low = 0
    high = len(data) - 1
    while low <= high:
        mid=(high+low) // 2
        if target == data[mid]:
            return True
        elif target < data[mid]:
            high = mid - 1
        else:
            low = mid + 1
    return False
print(binar_search_itereativ(data, target))
```

Ikkilik (binary) qidiruv algoritmi samaradorligi

- Agar ***C*** – taqqoslashlar soni va ***n*** – jadvaldagi elementlar soni bo’lsa, u holda

$$C = log_2 n$$

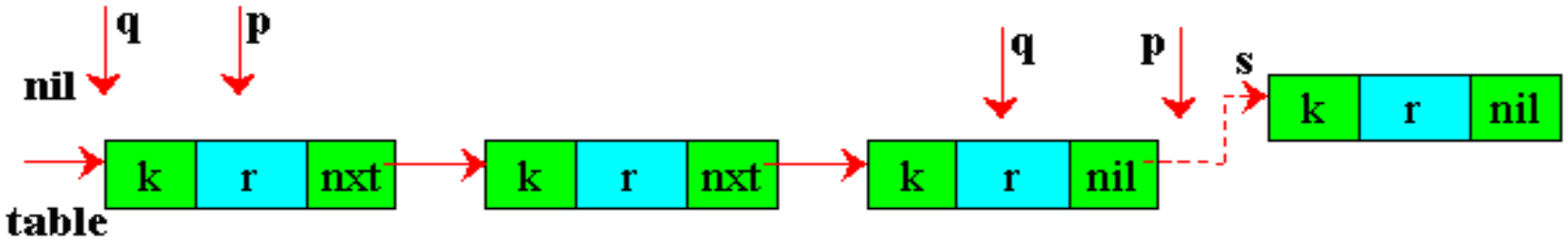
- Masalan, **n = 1024, bo’lsa,**

  Ketma-ket qidiruvda ***C = 512***, binar qidiruvda esa ***C = 10***.

- Agar katta hajmdagi ma’lumotlar ichida qidiruv amalga oshirilayotgan bo’lsa, u holda binar va indeksli ketma-ket qidiruvni umumlashtirib olib borish mumkin. Sababi, har ikkala qidiruv ham tartiblangan massivda amalga oshiriladi.
- Bu algoritmning *kamchiligi* massiv oldindan saralangan bo’lishi talab etilishidir.

Ro’yxatda ketma-ket qidiruv:

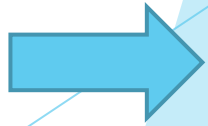
Agar ma’lumotlar tuzilmasi bir bog’lamli ro’yxat ko’rinishida berilgan bo’lsa, u holda ketma-ket qidiruv ro’yxatda quyidagicha amalga oshiriladi



Ro’yxatli tuzilmaning afzalligi shundan iboratki, ro’yxatga elementni qo’shish yoki o’chirish tez amalga oshadi, bunda qo’shish yoki o’chirish element soniga bog’liq bo’lmaydi, massivda esa elementni qo’shish yoki o’chirish taxminan barcha elementlarni yarmini siljitishni talab qiladi. Ro’yxatda qidiruvni samaradorligi taxminan massivniki bilan bir xil bo’ladi.

Bir bog’lamli ro’yxatda ketma-ket qidiruv algoritmi

```
//ro’yxat
struct TNode {    int value;
          TNode* pnext;
          TNode(int val): pnext(0), value(val) {}   };
//qidirish funksiyasi
TNode* Find(TNode *phead, int x)
{
TNode *p=phead;
       while(p)
  if (p->value==x) return p;
          else p = p->pnext;
  return 0;
}
```

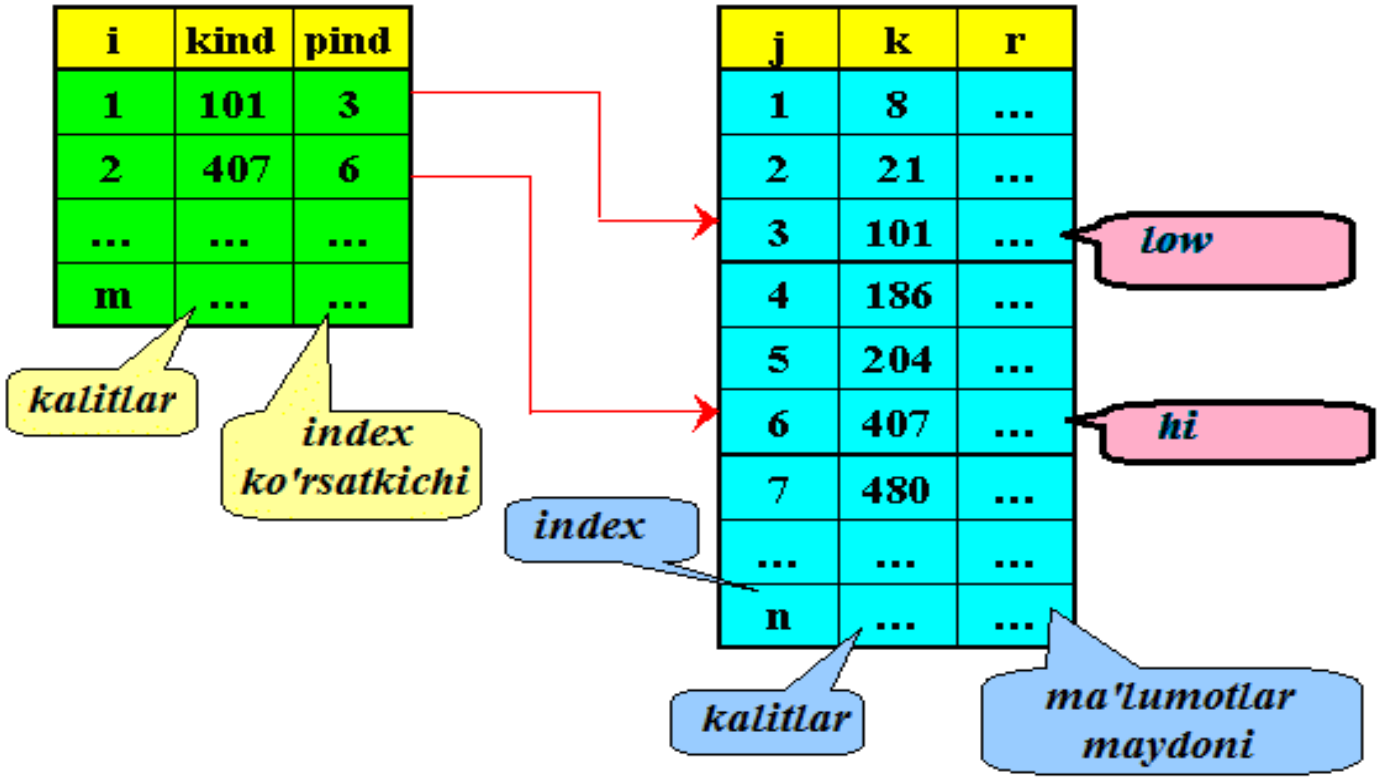


Indeksli ketma-ket qidiruv

- Bu qidiruv amalga oshirilayotganda ikkita jadval tashkil qilinadi: o’z kalitiga ega ma’lumotlar jadvali (o’sish tartibida tartiblangan) va indekslar jadvali.
- Bu yerda birinchi berilgan argument bo’yicha indekslar jadvalidan ketma-ketlikda qidirish amalga oshiriladi. Kalitlarni ko’rib chiqishda berilgan kalitdan kichigi topilsa, u holda ushbu kichik kalitni asosiy jadvaldagi qidirishning eng quyi chegarasi – ***low*** ga joylashtiramiz, xuddi shunday berilgan kalitdan katta deb topilgan kalitni (*kind* > *key*) yuqori ***hi*** ga joylashtiramiz.

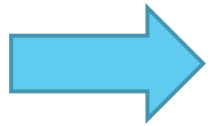
Indeksli ketma-ket qidiruv

- Misol uchun, ***key*** = **101 bo'lsin**. U holda qidiruv butun jadval bo'yicha emas, balki ***low*** dan ***hi*** gacha amalga oshiriladi.



Indeksli ketma-ket qidiruv

```
int InSeqsearch(int realArray[], int N, int kind[2][1000],int m,int key, int *t) {
        int i=0,      low = 0,      hi = 0;
            while ((i<m) && (kind[0][i]<=key)) {  i++;  (*t)++; }  (*t)++;
                  if (i==0)  low=0;   else      low=kind[1][i-1];
                        if (i==m)      hi=N;   else      hi=kind[1][i]-1;
                   for (int j=low; j<=hi; j++)   {
                         (*t)++;  if( key==realArray[j] )
                               {   return j; }
                  }
      return -1;
}
```



Indeksli ketma-ket qidirish samaradorligi

Agar barcha holatlar uchun teng ehtimollik deb hisoblansa, u holda qidirish samaradorligini quyidagicha hisoblash mumkin, n – tuzilmadagi elementlar soni, m – indeks soni, p – qadam uzunligi (o'lchami) deb olsak, $m = \frac{n}{p}$ bo'ladi va:

$$\mathrm{Q} = \frac{m+1}{2} + \frac{p+1}{2} = \frac{\frac{n}{p}+1}{2} + \frac{p+1}{2} = \frac{n}{2p} + \frac{p}{2} + 1;$$

Bunda Q ni p bo'yicha differentsiyallaymiz va nolga tenglashtiramiz:

$$\frac{d\mathrm{Q}}{dp} = \frac{d\left(\frac{n}{2p} + \frac{p}{2} + 1\right)}{dp} = -\frac{n}{2p^2} + \frac{1}{2} = 0;$$

Bu yerda $-\frac{n}{2p^2} + \frac{1}{2} = 0$ dan $p^2 = n; p = \sqrt{n}$ kelib chiqadi.

- $\mathrm{Q} = \sqrt{n} + 1$ qiymatga ega bo'lamiz, bu tuzilmadagi qidiruv uchun sarf qilingan taqqoslashlar sonini bildiradi.
- $\mathrm{Q}(\sqrt{n})$ indeksli ketma-ket qidirish algoritmining samaradorlik darajasini bildiradi.

Qidiruvning mukammal algoritmlari

- Umuman olganda, jadvalda har bir elementni qidirish ehtimolligini qandaydir bir qiymat bilan izohlash mumkin.
- Faraz qilaylik jadvalda qidirilayotgan element mavjud. U holda qidiruv amalga oshirilayotgan barcha jadvalni diskret holatga ega tizim sifatida qarash mumkin hamda unda qidirilayotgan elementni topish ehtimolligini – tizimning ***i-chi*** holati ehtimolligi ***p(i)*** deb olish mumkin, ya’ni:

$$\sum_{i=1}^{n} p(i) = 1$$

Qidiruvning mukammal algoritmlari

- Jadvalni diskret tizim sifatida qaraganimizda, undagi taqqoslashlar soni diskret tasodifiy miqdorlar qiymatlarini matematik kutilmasini ifodalaydi.

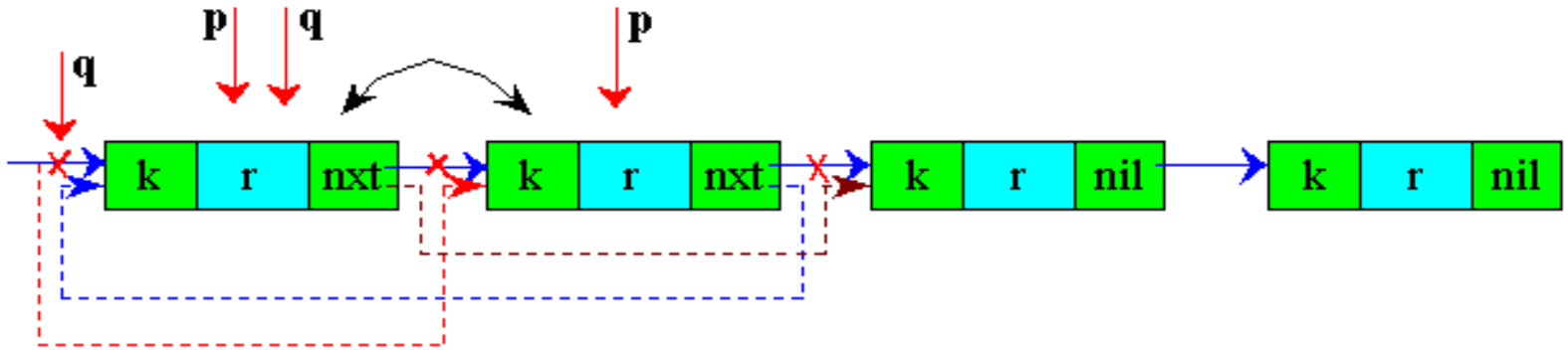
$$Q = 1p(1) + 2p(2) + 3p(3) + \cdots + np(n) = \sum_{i=1}^{n} ip(i)$$

Bu kutilmada $p(1) \geq p(2) \geq p(3) \ldots \geq p(n)$ bo'lsa, maqsadga muvofiq bo'ladi.

- Bu shart taqqoslashlar sonini kamaytirib, samaradorlikni oshiradi. Sababi, ketma-ket qidiruv birinchi elementdan boshlanganligi uchun, eng ko'p murojaat qilinadigan elementni birinchiga qo'yish lozim.
- Qidiruv jadvalini qayta tartiblashni eng ko'p ishlatiladigan ikkita usuli mavjud. Ularni bir bog'lamli ro'yxatlar misolida ko'rib chiqamiz.

Topilgan elementni ro’yxat boshiga qo’yish

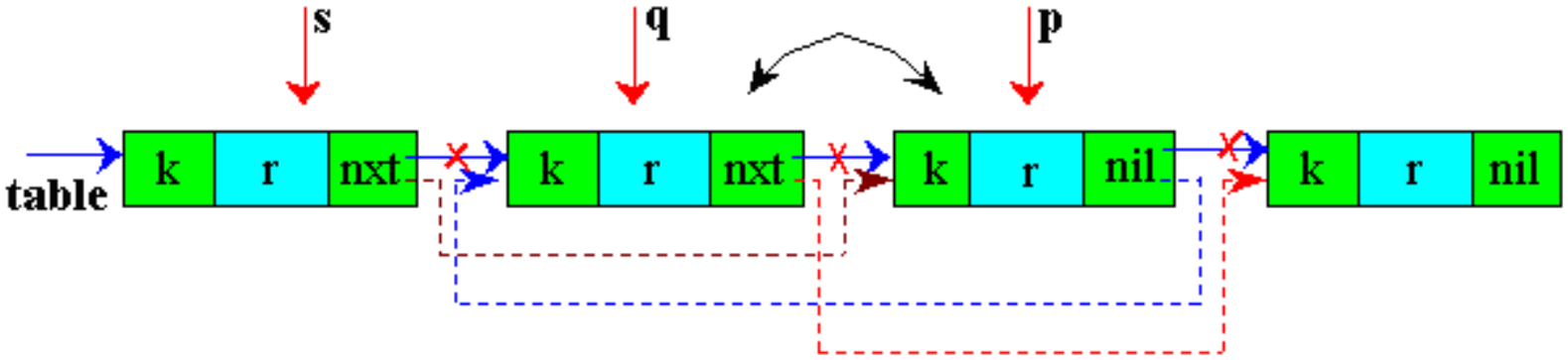
- Mazkur usulning mazmuni shundan iboratki, berilgan kalitga teng qiymatli element ro’yxatda birinchi element sifatida qaraladi, qolgan elementlari esa ushbu elementdan keyingi o’rin (pozitsiya)larga siljitiladi:



- Keltirilgan algoritm ro’yxat uchun ham massiv uchun ham o’rinli. Biroq bu algoritm massiv uchun tavsiya etilmaydi, sababi massiv elementlarining o’rnini almashtirishga ro’yxatdagi ko’rsatkichlar o’rnini almashtirishdan ko’ra ko’p vaqt talab qiladi.

Transpozitsiya usuli

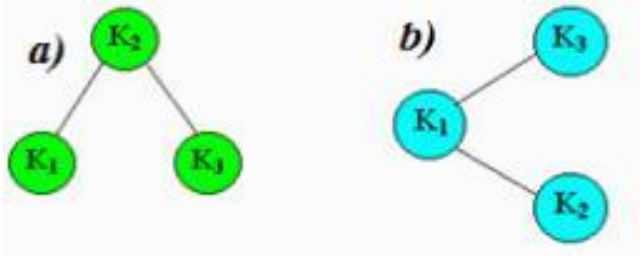
- Bu usulda topilgan element ro’yxatda o’zidan bitta oldingi element bilan o’rin almashtiriladi. Agarda mazkur elementga ko’p murojaat qilinsa, bittadan oldinga surilib borib natijada ro’yxat boshiga o’tkaziladi.



- Bu yerda, ***r*** – ishchi ko’rsatkich, ***q*** – yordamchi ko’rsatkich (***r*** dan bitta qadam keyingi), ***s*** - yordamchi ko’rsatkich (***q*** dan ikkita qadam keyingi)
- Ushbu usul nafaqat ro’yxatda, balki massivda ham qulay (sababi faqatgina ikkita yonma-yon turgan element o’rin almashtiriladi).

Mukammal qidiruv daraxti

- Agar ajratib olingan elementlar qandaydir o’zgarmas to’plamni tashkil qilsa, keyingi qadamdagi qidiruv samaraliroq bo’lishi uchun ularni binar daraxt ko’rinishida ifodalash maqsadga muvofiq bo’ladi.
- Quyida keltirilgan daraxtlarda binar qidiruvni ko’rib chiqaylik (a)va b) chizma). Ikkala daraxt ham uchtadan elementga ega - *k1*, *k2*, *k3* bo’lib, bu yerda ***k1<k2<k3***. *k3* elementni qidirish a) chizmada ikkita taqqoslashni talab qilsa, b) chizmada esa bitta.
- Biror bir yozuvni ajratib olish uchun zarur bo’lgan kalitlarni taqqoslashlar soni binar qidiruv daraxtidagi ushbu yozuv bosqichiga birni qo’shganiga teng.

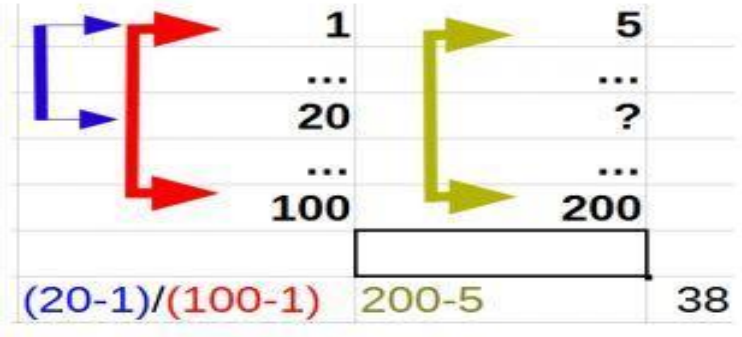


Interpolyatsiya algoritmi

- **Interpolyatsiya** — bu butun soha va qidirilayotgan qiymatga o’xshash elementlar joylashgan masofani hisoblash orqali qidiruv sohasini aniqlash usuli hisoblanadi. Bunga misol sifatida geometriyadagi o’xshash uchburchaklarni olish mumkin, bunda burchaklar qiymati bir xil, lekin proportsiyasi har xil bo’ladi. Interpolyatsiya usulida ham aynan shunday printsipdan foydalaniladi.
- Qidiruv sohasi uzunligi soha boshidan kerakli songacha (masalan, markazdagi elementgacha) masofa hisoblanadi. Hisoblash element nomeri va qiymatlari bo’yicha amalga oshiriladi, undan keyin aniqlangan soha uzunligi bilan qiymatlar orasidagi uzunlik ko’paytiriladi va natijaga soha boshining qiymati qo’shilib, qidirilayotgan qiymat aniqlanadi.

Interpolyatsiya algoritmi

- Buni tushinib olish uchun quyidagicha chizmani olamiz. Bunda 100 ta elementdan iborat massiv berilgan:



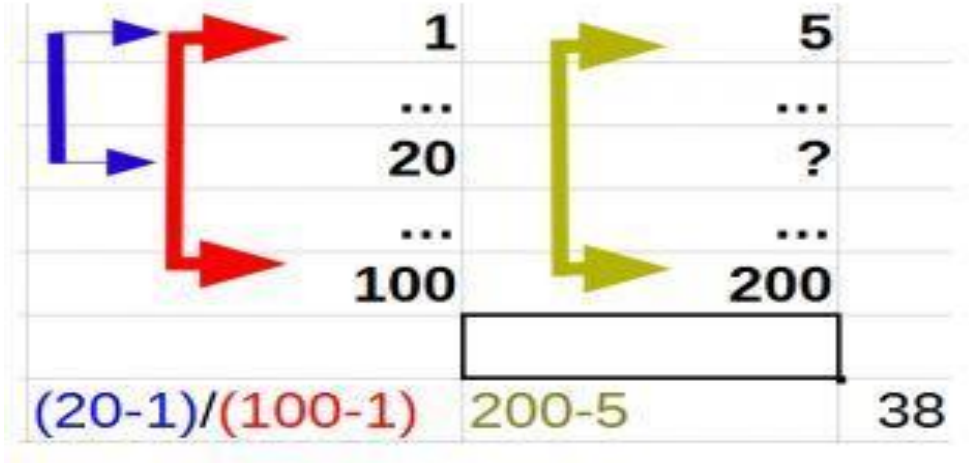
- Buni hisoblash formulasi juda sodda bo’lib, qidirilayotgan element va birinchi element orasidagi masofa (uzunlik)ni hisoblaydi (misolda 20-1). Xuddi shunday birinchi va oxirgi elementlar orasidagi uzunlik hisoblanadi (misoldagi 100-1). Aniqlangan uzunliklar o’zaro bo’linadi (misoldagi (20-1)/(100-1)) (o’xshashlik sohasi uzunligi va birinchi va oxirgi elementlar orasidagi uzunlikka). Xuddi shunday qiymatlarning chegaralari orasidagi uzunlik ham hisoblanadi (misoldagi 200-5).

Interpolyatsiya algoritmi

- Olingan natijalar o’zaro ko’paytriladi va birinchi yacheyka nomeriga ko’paytiriladi. Ya’ni,

$$1 + (20-1)/(100-1) * (200-5) = 38 \text{ (qoldiq bilan)}.$$

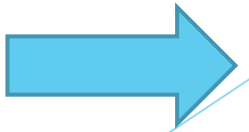
- Olingan natija aynan qidirilayotgan qiymatni beradi. Ya’ni misoldagi №20 nomerda 38 qiymati joylashgan.



Interpolyatsiya algoritmi

Yuqorida keltirilgan algoritmning C++ dagi ko’rinishi quyidagicha:

```
#include <iostream>
using namespace std;
int main() {
 int A[] ={ 1, 2, 4, 6, 7, 89, 123, 231, 1000, 1235 };
 int x = 0;      int a = 0;      int b = 9;
 int d = 1235; //izlanayotgan element
 bool found;
 for (found = false; (A[a] < d) && (A[b] > d) && !found; )
 {          x = a + ((d - A[a]) * (b - a)) / (A[b] - A[a]);
      if (A[x] < d)         a = x + 1;
      else if (A[x] > d)   b = x - 1;
      else found = true;}
 if (A[a] == d) cout << d << "  element topildi indeksi " << a << "  ga teng" << endl;
 else if (A[b] == d) cout << d << " element topildi indeksi " << b << "  ga teng" << endl;
 else cout << "Topilmadi" << endl;
 return 0;  }
```

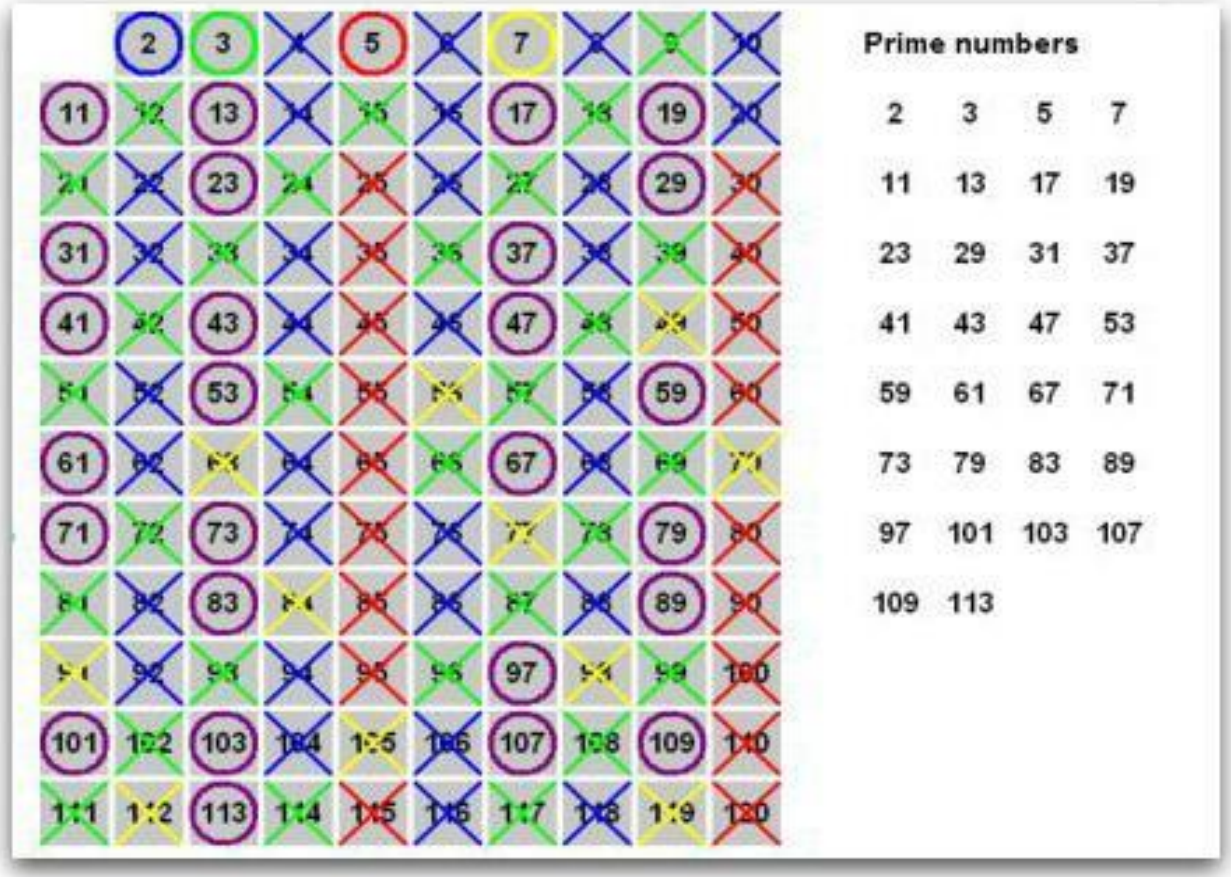


Eratosfen to’ri (panjarasi)

- **Eratosfen to’ri (panjarasi)** — sonlar to’plamidan “tub” sonlarni ajratib olishda qo’llaniladigan algoritm.
- Bu algoritm bizning eramizdan oldin grek matematigi, astronom va geograf Eratosfen Kirenskiy tomonidan ixtiro qilingan.
- Bu algoritm juda sodda bo’lib, sonlar to’plamida ikki marta takrorlanuvchi tsikldan iborat. Birinchi tsikl tub sonlarni, ikkinchisi esa, ushbu tub sondan keyin keluvchi murakkab sonlarni, aniq formula yordamida ajratib beradi. Bu yerda ikkinchi tsiklda tub sondan keyingi murakkab sonlarni ketma-ket o’chirmaydi, balki quyidagi formula yordamida topilgan birinchi tub sondan keyingi murakkab sonlarni ajratadi, masalan, n-birinchi tub son bo’lsin:
- 1-qadam: n*n
- 2-qadam: n*n+n
- 3-qadam: n*n+n+n va h.k. ushbu qadamlar ketma-ketligida topilgan tub songa karrali bo’lgan barcha sonlarni ajratish (belgilash).

Eratosfen to’ri (panjarasi)

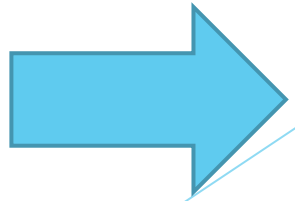
- Buni quyidagicha grafik tasvirini hosil qilish mumkin:



Eratosfen to’ri (panjarasi)

- Ushbu algoritmning kodi quyidagicha:

```
for (int i = 2; i * i <= n; i++)
 { if (a[i])
 //Agar joriy son 0 ga teng bo'lmasa - undan boshlab murakkab sonni qidirish
            for (int j = i*i; j <= n; j += i)
 //va uni qayta ko'rmaslik uchun nolga tenglashtirish
                  a[j] = 0; }
 // sikl oxiri, 0 ga tenglashtirilmagan tub sonni chiqarish
 for (int i = 2; i < n; i++)
 {    if (a[i])  {
            cout <<a[i] << ' ';            }
 }
```



Adabiyotlar

- **[RU]** Алфред В. Ахо., Джон Э. Хопкрофт, Джефри Д. Ульман. Структура данных и алгоритмы. //Учеб.пос., М.: Изд.дом: "Вильямс", 2000, — 384 с.
- **[EN] Adam Drozdek. Data structures and algorithms in C++. Fourth edition.Cengage Learning, 2013.**
- **[UZ] Narzullaev U.X., Qarshiev A.B., Boynazarov I.M. Ma’lumotlar tuzilmasi va algoritmlar.** //O’quv qo’llanma. Toshkent: Tafakkur nashriyoti, 2013 y. – 192 b.
- **[RU]** Лойко В.И. Структуры и алгоритмы обработки данных. Учебное пособие для вузов. - Краснодар: КубГАУ. 2000. - 261 с., ил.

Mustaqil ishlash uchun topshiriqlar:

- Binar daraxtlar ustida bajariladigan amallarga doir misollar yechish
- *Izoh: dars mashg'ulotida berilgan bilimlarga qo'shimcha ma'lumotlarni to'plash-konspekt qilish*