# JARAYONNING XOTIRA MURAKKABLIGI

*Mo‘minov Bahodir Boltayevich[1], Muxamadiyev Sanjar Isoyevich[2]*

*[1]Toshkent davlat iqtisodiyot universiteti t.f.d., professor, mbbahodir@gmail.com*
*[2]Toshkent davlat iqtisodiyot universiteti Sun’iy intellekt kafedrasi katta o‘qituvchisi, mrmsi2885@gmail.com*

*KEYWORDS*

xotira murakkabligi, funksiya, qoldiqli rekursiv funksiya, qoldiqsiz rekursiv funksiya, maydon, sinf, tuzilma

*ABSTRACT*

Ushbu maqola xotirani moslashtirish va to'ldirishning sinflar va tuzilmalar kabi ma'lumotlar tuzilmalarining xotira murakkabligiga ta'sirini o'rganadi. Xotiraning murakkabligiga maydonlar turi va soni, moslashtirish talablari va xotiraga kirishni optimallashtirish uchun kompilyator tomonidan kiritilgan to'ldirish kabi omillar ta'sir qiladi. Hizalama unumdorlikni oshirsa-da, chegara cheklovlarini qondirish uchun to'ldirishni qo'shish orqali qo'shimcha xotira yukini kiritadi. Matematik formulalar xotira murakkabligi maydon turlari, to'ldirish va tilga xos bo'lgan qo'shimcha xarajatlar asosida qanday hisoblanganligini ko'rsatish uchun ishlatiladi. Ushbu omillarni tushunish algoritmni loyihalash va optimallashtirishda xotirani yanada samarali boshqarish imkonini beradi.

## Kirish.

Jarayonning murakkabligi jarayonostilarning murakkabliklarining yig‘indisiga teng. Jarayonning murakkabligi esa jarayonni amalga oshirish metodiga(algoritmiga) bog‘liq bo‘lib hisoblanadi. Algoritm murakkabligini asimptotik baholash odatiy holdir. Algoritmlarni nazariy tahlil qilishda ularning murakkabligini asimptotik ma’noda baholash, ya’ni ixtiyoriy ravishda katta kiritilgan ma’lumotlar uchun murakkablik funksiyasini baholash odatiy holdir.

Algoritm tahlili hisoblash murakkabligi nazariyasining muhim qismi bo‘lib, u aniq hisoblash muammosini hal qilish uchun algoritmning zarur resurslarini nazariy baholashni ta’minlaydi. Ko‘pgina algoritmlar ixtiyoriy uzunlikdagi kirishlar bilan ishlash uchun mo‘ljallangan. Algoritmlarni tahlil qilish - uni bajarish uchun zarur bo‘lgan vaqt va xotira resurslari miqdorini aniqlash.

Algoritmning murakkabligi va tahlili hisoblash murakkabligi nazariyasi doirasida chuqur o'rganilgan informatika fanining muhim mavzulari bo'lgan. Tahlil odatda kirish hajmi oshgani sayin muammoni hal qilish uchun algoritm uchun zarur bo'lgan vaqt va xotirani baholash atrofida aylanadi. Bu turli xil sharoitlarda turli xil algoritmlarning samaradorligi haqida tushuncha beradi.

Asimptotik tahlil algoritm murakkabligini baholashda, ayniqsa katta hajmdagi kirishlarda qo'llaniladigan asosiy usul hisoblanadi. Knuth (1976) Big O notation tushunchasini kiritdi, u algoritmlar uchun zarur bo'lgan vaqt yoki xotoraning yuqori chegarasini ta'minlaydi, chunki kirish hajmi cheksizlikka intiladi. Ushbu belgi algoritmlarni eng yomon holatlar stsenariysi bo'yicha tasniflash imkonini beradi va har qanday kirish hajmini sinab ko'rmasdan turli usullarni solishtirish imkonini beradi.

Turli tadqiqotlar o'rtacha va eng yaxshi holatdagi murakkabliklarga qaratilgan (Kormen va boshq., 2009), ammo asimptotik (eng yomon holat) tahlili ustunlik qiladi, chunki u resurslardan foydalanishning yuqori chegarasini ta'minlaydi. Masalan, chiziqli vaqt murakkabligi (O(n)) bo‘lgan algoritmlar odatda kvadratik ($O(n^2)$) yoki eksponensial algoritmlarga (O(2^n)) qaraganda katta kirishlar uchun samaraliroq hisoblanadi.

## Adabiyotlar tahlili

Hisoblash murakkabligi nazariyasi. Algoritm tahlilining asosini hisoblash murakkabligi nazariyasi tashkil etadi, u muammolarni o'ziga xos qiyinchilik asosida

tasniflash bilan bog'liq. Garey va Jonson (1979) muammolarni P (polinom vaqtida echiladigan) va NP (polinom vaqtida tekshirilishi mumkin) kabi murakkablik sinflariga tasniflab, algoritmlarning cheklovlarini tushunish uchun asos yaratdi. Mashhur P va NP muammosi, Kuk (1971) ta'kidlaganidek, algoritm dizayniga keng ta'sir ko'rsatadigan sohadagi ochiq savollardan biri bo'lib qolmoqda.

Nazariya, shuningdek, deterministik va deterministik bo'lmagan algoritmlarni farqlaydi, deterministik algoritmlarni tahlil qilish osonroq. NP-hard va NP-complete kabi murakkablik sinflari ma'lum polinom-vaqt echimlari mavjud bo'lmagan muammolarni tavsiflaydi, ammo amalda bu qiyin muammolarni samarali hal qilish uchun yaqinlashtirish algoritmlari yoki evristika (Papadimitriou, 1994) ko'pincha ishlatiladi.

Algoritmni loyihalash va baholash. Algoritmlarning amaliy dizayni ko'pincha vaqt murakkabligi va xotira murakkabligi o'rtasidagi kelishuvni o'z ichiga oladi. Masalan, mergesort kabi bo'lin va zabt et algoritmlari muammolarni kichikroq kichik muammolarga bo'lish orqali sodda algoritmlarga qaraganda yaxshiroq ishlashga erishadi. Tarjan (1975) union-find kabi ilg'or ma'lumotlar tuzilmalarini o'rganib chiqdi, ular grafik algoritmlardagi muayyan operatsiyalar uchun vaqt murakkabligini minimallashtiradi.

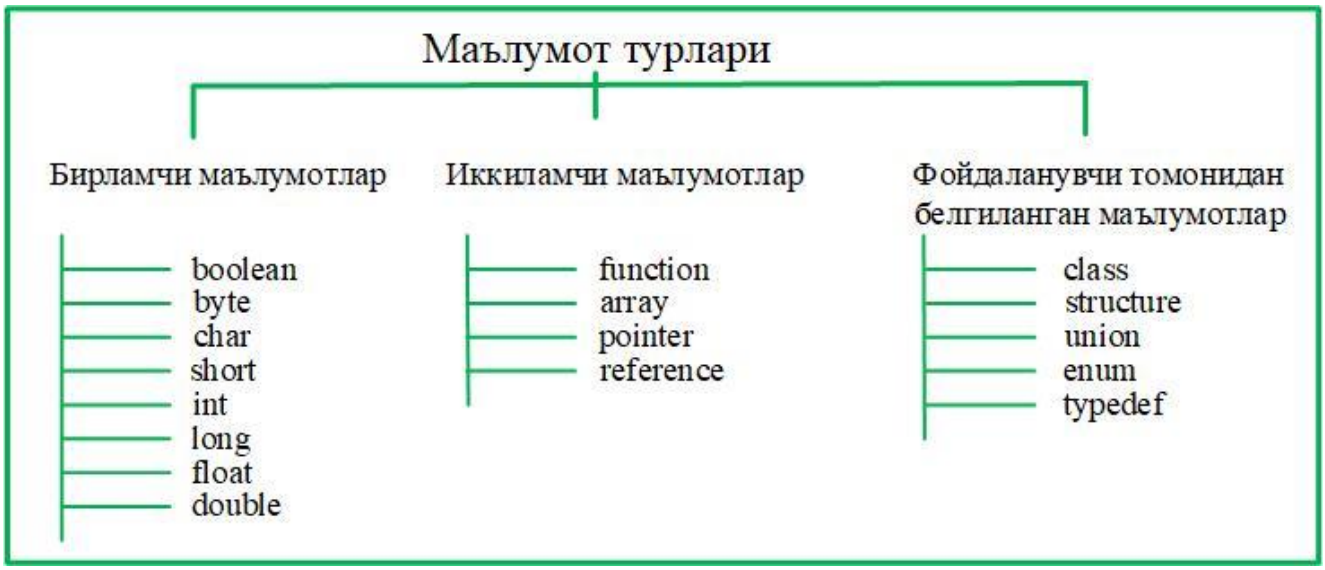
Algoritmlar muayyan holatlar uchun ham optimallashtirilishi mumkin. Misol uchun, Strassenning matritsalarni ko'paytirish algoritmi (Strassen, 1969) standart kub-vaqt usulini yaxshilaydi, bu esa uni katta kirishlar uchun qulayroq qiladi. Shu bilan birga, dinamik dasturlash va ochko'z algoritmlar ko'pincha ortiqcha hisob-kitoblarni kamaytirish orqali murakkab muammolarni samarali hal qilishga erishadi (Bellman, 1957).

Algoritmning murakkabligi ko'pincha bosqichlar sonining funksiyasi sifatida aniqlanadi, bu vaqt murakkabligi yoki xotiraning murakkabligi (yoki kosmik murakkablik) deb ataladigan iste'mol qilinadigan xotira miqdori . Ushbu ko'rsatkichlar kirish hajmining oshishi bilan algoritmlarning qanday masshtablanishini tushunish uchun zarurdir. Vaqtning murakkabligi algoritmning ishlash vaqti kirish hajmiga bog'liq holda qanday ko'payishini aniqlaydi, odatda quyidagicha belgilanadi. Xuddi shunday, n xotiraning murakkabligi algoritmni bajarish vaqtida talab qiladigan xotira yoki bo'sh joy miqdorini anglatadi.

Vaqt murakkabligi deb nomlanuvchi qadamlar soniga yoki xotira murakkabligi deb nomlanuvchi xotira miqdoriga bog‘liq bo‘lgan funksiya sifatida belgilanadi.

Xotira murakkabligini hisoblash uchun dasturda ishlatilayotgan o‘zgaruvchilarning tipi va soni ahamiyatga ega. Umaman olganda foydalanuvchi tomonidan belgilangan ma’lumot tiplari 3 xil bo‘ladi.

- Birlamchi ma’lumotlar tiplari
- Ikkilamchi ma’lumotlar tiplari
- Foydalanuvchi tomonidan belgilangan ma’lumotlar tiplari



*1-rasm. Foydalanuvchi tomonidan belgilangan ma'lumot tiplari*

Birlamchi ma'lumotlar tiplari dasturlash tili tomonidan taqdim etilgan va o'rnatilgan ma'lumotlar tiplaridir. Birlamchi ma'lumotlar tiplari bir necha turda bo'lishi mumkin, masalan, int, unsigned int, short int, unsigned long int va boshqalar.

Ma'lumotlarning birlamchi ma'lumot turlari $T=\{t_1,t_2, \ldots,t_m\}$ va ularning xotiradan egallaydigan xotira hajmi mos ravishda $H=\{h_1,h_2, \ldots,t_m\}$, hamda ularning dasturda foydalanishlar soni $N=\{n_1,n_2, \ldots,n_m\}$, lar bo'lsa dasturning xotira murakkabligi

$$O = h_1 n_1 + h_2 n_2 + \cdots + h_m n_m = \sum_{i=1}^{m} h_i \cdot n_i \qquad (1)$$

Ikkilamchi ma'lumotlar turlari asosan birlamchi ma'lumotlar turlaridan kelib chiqadi. Masalan massiv, funksiya, ko'rsatkichlar va boshqalar.

Massivlar odatda statik, dinamik hamda bir va ko'p o'lchamli bo'ladi. Massivlarning xotira marakkabligi elementlar soni va elementlarining tipiga bog'liq. $O = h * n$

Dastur bajarilishi davomida xotira murakkabligi o'zgarib turadi. Funksiya va operatorlar tarkibidagi lokal o'zgaruvchilar ular foydalanilmayotgan vaqtda xotiridan joy olmaydi. Funksiyaning xotira murakkabligi funksiyada e'lon qilingan lokal o'zgaruvchilar va parametr sifatida uzatilgan o'zgaruvchilar hajmining yig'indisi bilan hisoblanadi. Funksiyaga parametr sifatida havolalar berilsa funksiya xotira murakkabligiga ta'sir qilmaydi. Funksiyaning xotira murakkabligi quyidagi formula yordamida hisoblanadi.

$$O_f = O_p + O_i \qquad (2)$$

Bu yerda $O_p$ funksiya parametrlarining xotira murakkabligi, $O_i$ funksiyaning lokal o'zgaruvchilarining xotira murakkabligi.

Rekursiv funksiyalarning xotira murakkabligi rekursiyaning qoldiq-rekursiv yoki yo'qligiga bog'liq.

Qoldiqli rekursiv funksiyada rekursiv chaqiruv funksiyadagi oxirgi amal bo'lib, rekursiv chaqiruv natijasi hech qanday boshqa hisob-kitoblarsiz bevosita qaytariladi.

Qoldiqli rekursiv funksiyalar kompilyatorlar tomonidan qoldiq chaqiruvini optimallashtirish yoki qoldiq chaqiruvini yo'q qilish deb nomlangan texnika orqali optimallashtirilishi mumkin. Ushbu optimallashtirish har bir rekursiv chaqiruv uchun alohida stek ramkasini saqlash zaruratini yo'q qiladi, rekursiyani samarali ravishda siklga aylantiradi.

Qoldiqli rekursiv bo'lmagan rekursiv funksiyalarning xotira murakkabligi doimiy, O(1), chunki chaqiruvlar to'plami har bir rekursiv chaqiruv bilan o'smaydi.

Qoldiqli rekursiv funksiyaga misol:

```
def factorial(n, acc=1):
  if n == 0:
        return acc
  else:
        return factorial(n-1, n*acc)
```

Qoldiqsiz rekursiv funksiyada rekursiv chaqiruv qaytgandan keyin bajarilishi kerak bo'lgan amallar mavjud bo'lib, har bir rekursiv chaqiruv chaqiruvlar stekiga yangi stek ramkasini qo'shadi va agar rekursiya yetarlicha chuqur bo'lsa, bu stekning to'lib ketishiga olib kelishi mumkin.Qoldiqsiz rekursiv funksiyalarning xotira murakkabligi odatda O(n) dir, bu yerda n - rekursiyaning chuqurligi (rekursiv chaqiruvlar soni). Qoldiqsiz rekursiv funksiyaga misol:

```
def factorial(n):
  if n == 0:
        return 1
  else:
        return n*factorial(n-1)
```

Ko'rinib turibdiki rekursiv funksiyalarning xotira murakkabligi qoldiqli rekursiv yoki qoldiqsiz rekursivligiga qarab o'zgarishi mumkin. Qoldiqli rekursiv funksiyalar odatda doimiy xotira murakkabligiga ega, qoldiqsiz rekursiv funksiyalar esa chiziqli xotira murakkabligiga ega.

Foydalanuvchi tomonidan belgilangan ma’lumotlar turi bu ma’lumotlarni dasturchi tanlagan tarzda belgilaydi. Bunga misol qilib classlar, tuzulmalar, enumlar va boshqalarni aytish mumkin.

Dasturlash tillaridagi sinflarning xotira murakkabligi bir necha omillarga, jumladan xususiy o‘zgaruvchilari, usullari va sinf bilan bog‘liq har qanday qo‘shimcha xotira yukiga bog‘liq. Bu yerda ba’zi fikrlar mavjud:

Sinfning xotira murakkabligiga uning o‘zgaruvchilari (maydonlar yoki atributlar deb ham yuritiladi) uchun zarur bo‘lgan xotira ta’sir qiladi.

Sinfning har bir obyekti xususiy o‘zgaruvchilari to‘plamiga ega va bu o‘zgaruvchilar uchun zarur bo‘lgan xotira umumiy xotira murakkabligiga hissa qo‘shadi.

Sinf usullari ham xotiradan foydalanishi mumkin, lekin bu odatda xususiy o‘zgaruvchilari bilan solishtirganda kichik doimiy omil hisoblanadi.

Usullar uchun kod odatda sinfning barcha maydonlari orasida taqsimlanadi, shuning uchun u maydonlar soni bilan o‘smaydi.

Agar sinfda statik o‘zgaruvchilar mavjud bo‘lsa, ular sinfning barcha misollari orasida taqsimlanadi va xotira murakkabligiga hissa qo‘shadi.

Dasturlash tiliga qarab, har bir usul chaqiruvi bilan bog‘liq ba’zi xotira yuklari bo‘lishi mumkin. Bu usulning lokal o‘zgaruvchilari, chaqiruvlar to‘plamini va har qanday hisoblash ma’lumotlari uchun zarur bo‘lgan xotirani o‘z ichiga oladi.

Agar sinf boshqa sinfdan meros bo‘lsa yoki interfeyslarni amalga oshirsa, vtables (virtual funksiya jadvallari) yoki usullarni yuborish mexanizmlari bilan bog‘liq qo‘shimcha xotira yuki bo‘lishi mumkin.

**Muhokama.**

Ba’zi dasturlash tillari va kompilyatorlar xotiradagi ma’lumotlar tuzilmalarini tekislash uchun to‘ldirishni kiritishi mumkin, bu sinfning xotira murakkabligiga ta’sir qilishi mumkin.

Tuzilmalarning xotira murakkabligi sinflarga o‘xshab turli omillar, jumladan, ularning maydonlarining soni va turlari, moslashtirish talablari va dasturlash tili yoki kompilyator tomonidan yuklangan har qanday qo‘shimcha xotira yuki ta’sir qiladi. Keling, ba’zi asosiy fikrlarni ajratib ko‘rsatamiz:

Tuzilmaning xotira murakkabligiga ta’sir qiluvchi asosiy omil - bu uning maydonlarining soni va turi (maydonlar yoki atributlar). Har bir maydon umumiy xotiradan foydalanishga hissa qo‘shadi. Butun sonli maydonlarga ega tuzilma odatda belgilar kabi kichikroq o‘lchamli maydonlarga ega bo‘lgan tuzilmaga qaraganda ko‘proq xotira talab qiladi.

Ko'pgina dasturlash tillarida xotirani tekislash va to'ldirish sinflar va tuzilmalar kabi ma'lumotlar tuzilmalarining xotiradan haqiqiy foydalanishini aniqlashda hal qiluvchi rol o'ynaydi. Xotirani tekislash ma'lumotlar turining o'lchamiga yoki protsessor tomonidan o'rnatilgan maxsus chegara talablariga ko'payadigan manzillarda xotiradagi ma'lumotlarni joylashtirishni anglatadi. Boshqa tomondan, **padding** - bu ma'lumotlar strukturasidagi maydonlarning to'g'ri tekislanganligini ta'minlash uchun kompilyator tomonidan kiritilgan qo'shimcha xotira.

Xotirani moslashtirishning asosiy maqsadi kirish tezligini optimallashtirish va apparatni xotiradan samarali o'qish va xotiraga yozishni ta'minlashdir. Noto'g'ri moslashtirilgan ma'lumotlar xotiraga samarasiz kirishga olib kelishi mumkin, chunki protsessor turli xil xotira chegaralari bo'ylab ma'lumotlarni qayta ishlash uchun bir nechta o'qish yoki yozishni amalga oshirishi kerak bo'lishi mumkin.

Xotiraning murakkabligiga ta'siri. Ma'lumotlar strukturasining xotira murakkabligiga quyidagi omillar ta'sir qiladi :

- Uning maydonlarining soni va turlari.
- Uskuna tomonidan qo'yiladigan hizalama talablari.

- Ushbu moslashtirish talablariga javob berish uchun kompilyator tomonidan kiritilgan har qanday to'ldirish.
- C++ kabi ob'ektga yo'naltirilgan tillarda virtual jadval ko'rsatkichlari kabi kompilyatorlar tomonidan kiritilgan tilga xos qo'shimcha xarajatlar.

Ma'lumotlar strukturasidagi har bir o’zgaruvchi uning umumiy xotiradan foydalanishiga hissa qo'shadi. Turli xil ma'lumotlar turlari turli xil xotira talablariga ega:

Butun sonlar (masalan, 32-bit int): odatda 4 bayt talab qiladi.

Belgilar (masalan, char): odatda 1 bayt talab qiladi.

Haqiqiy sonlar (masalan, float): odatda 4 baytni talab qiladi, ikki tomonlama aniqlik (masalan, double) uchun 8 bayt kerak bo'lishi mumkin.

Masalan, C++ tilidagi butun son va belgilar maydoniga ega struktura quyidagicha ko'rinishi mumkin:

```
struct Example {
   int a;   // 4 bayt
   char b;  // 1 bayt
};
```

Bir qarashda, bu struktura 5 bayt xotirani egallashi kerakdek tuyulishi mumkin (butun son uchun 4 bayt va belgi uchun 1 bayt). Biroq, xotira moslashuvi tufayli foydalanilgan umumiy xotira kattaroq bo'lishi mumkin. Bu bizni padding roliga olib keladi .

Xotiraga samarali kirish uchun ko'pgina tizimlar butun sonlar, floatlar va juftlar kabi ma'lumotlar turlarini aniq xotira chegaralariga moslashtirishni talab qiladi. Misol uchun, 4 baytlik butun sonni 4 baytlik chegaraga moslashtirish kerak bo'lishi mumkin, ya'ni uning xotiradagi boshlang'ich manzili 4 ga bo'linishi kerak. Agar ma'lumotlar tuzilmasi a kabi kichikroq maydonlarni o'z ichiga olsa char(bu faqat 1 baytni talab qiladi), kompilyator butun sonlar kabi kattaroq maydonlarni to'g'ri tekislanganligini ta'minlash uchun to'ldirishni qo'shishi mumkin.

Turli xil dasturlash tillari xotiradan foydalanishga kelganda o'zlarining qo'shimcha xarajatlarini kiritadilar. Masalan, C++ kabi ob'ektga yo'naltirilgan tillarda meros yoki virtual usullardan foydalanadigan sinflar ko'pincha virtual jadvallar (v-jadvallar) uchun qo'shimcha ko'rsatkichlarni o'z ichiga oladi va har bir ob'ektga ko'proq xotira yukini qo'shadi.

Bunday hollarda xotira murakkabligi quyidagicha ifodalanishi mumkin:

$$S(n) = \sum_{i=1}^{k} sizeof(field_i) + padding + overhead$$

Bu yerda $k$ kmaydonlar soni va qo'shimcha yuk meros va virtual jadvallar kabi xususiyatlar uchun zarur bo'lgan qo'shimcha xotirani ifodalaydi.

Xotirani moslashtirish tuzilmalar uchun muhim ahamiyatga ega. Ko‘pgina dasturlash tillari va kompilyatorlar xotiraga kirish samaradorligini oshirish uchun tuzilma maydonlarini moslashtiradi.

Moslashtirish talablari tuzilma maydonlari o‘rtasida to‘ldirishni kiritishi mumkin, bu esa qo‘shimcha xotiradan foydalanishga olib keladi.

Aniq moslashtirish qoidalari dasturlash tiliga, kompilyatorga va maqsadli arxitekturaga bog‘liq.

To‘ldirish tuzilma maydonlarini ularning o‘lchamiga karrali xotira manzillariga moslashtirish uchun ishlatiladi. Kiritilgan to‘ldirish miqdori umumiy xotira murakkabligiga ta’sir qilishi mumkin.

To‘ldirishning samaradorligi kompilyatorning bo‘sh joyni minimallashtirish uchun tuzilmaning tartibini qanchalik optimallashtirishiga bog‘liq.

Tuzilmalar ichidagi massivlar: Agar tuzilmada massivlar bo‘lsa, xotira murakkabligiga massiv elementlarining o‘lchami va turi, shuningdek, massivdagi elementlar soni ta’sir qiladi.

Ichma ich tuzilmalar: Agar tuzilma boshqa tuzilmalarni (ichiga o‘rnatilgan tuzilmalar) o‘z ichiga olsa, xotira murakkabligiga ichki

o‘rnatilgan tuzilmalarning xotira murakkabligi ta’sir qiladi.

Turli xil dasturlash tillari va kompilyatorlar tuzilmalarni boshqarish va xotirani moslashtirish uchun turli qoidalarga ega bo‘lishi mumkin. Ba’zi tillar ishlab chiquvchilarga direktivalar yoki izohlar orqali tuzilma maydonlarining ishlov berilishi va moslashishini nazorat qilish imkonini beradi.

**Xulosa.**

Xulosa qilib aytganda, tuzilmalarning xotira murakkabligi ularning maydonlarining soni va turlari, xotira moslashuvi, to‘ldirish va boshqa til/kompilyatorga xos bo‘lgan fikrlar bilan belgilanadi. Tuzilmalarning xotira murakkabligini tahlil qilish ushbu tuzilmalarning nusxalarini saqlash uchun qancha xotira kerakligini tushunish imkonini beradi.

**Foydalanilgan adabiyotlar.**

1. Levitin, Anany. Introduction to the design & analysis of algorithms / Anany Levitin. — 3rd ed.
   Muminov B. B., Kh E. Modelling asynchronous parallel process with Petri net //International Journal of Engineering and Advanced Technology (IJEAT). – 2019. – Т. 8. – С. 400-405.
2. Мўминов Б. Б. и др. МЕТОДЫ И АЛГОРИТМЫ ВВОДА, РЕДАКТИРОВАНИЯ И СОХРАНЕНИЯ ДАННЫХ НА ПЛАТФОРМЕ IS-ICT //Евразийский Союз Ученых. – 2019. – №. 9-1 (66).
3. Kushwah S. P. S., Rawat K., Gupta P. Analysis and comparison of efficient techniques of clustering algorithms in data mining //International Journal of Innovative Technology and Exploring Engineering (IJITEE). – 2012. – Т. 1. – №. 3.
4. Hofman T., Dai C. H. Energy efficiency analysis and comparison of transmission technologies for an electric vehicle //2010 IEEE vehicle power and propulsion conference. – IEEE, 2010. – С. 1-6.
5. Kanade T. et al. Cell image analysis: Algorithms, system and applications //2011 IEEE Workshop on Applications of Computer Vision (WACV). – IEEE, 2011. – С. 374-381.
6. **Knuth, D. E.** (1976). *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. Addison-Wesley.
7. **Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C.** (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.
8. **Garey, M. R., & Johnson, D. S.** (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.
9. **Cook, S. A.** (1971). *The complexity of theorem-proving procedures*. *Proceedings of the Third Annual ACM Symposium on Theory of Computing* (pp. 151–158). ACM.
10. Rowley, J.E. (1993). Information systems methodologies: A review and assessment of their applicability to the selection, design and implementation of library and information systems. Journal of Information Science, 19(4), 291-301.
11. Armstrong, D. & Hardgrave, B. (2007). Understanding mindshift learning: The transition to object-oriented development," MIS Quarterly, 31(3).
12. Zhao, L. & Siau, K. (2002). Component-based development using UML. Communications of the Association for Information Systems, 9, 207-222.
13. Erickson, J. & Siau, K. (2008). Web services, service oriented computing, and service oriented architecture: Separating hype from reality. Journal of Database Management, 19(3), 42-54.
14. Erickson, J., Lyytinen, K., & Siau, K. (2005). Agile modeling, agile software development, and extreme programming: The state of research. Journal of Database Management, 16(4), 88-100.
15. Siau, K. & Tan, X. (2005a). Improving the quality of conceptual modeling using cognitive mapping techniques. Data And Knowledge Engineering, 55(3), 343-365.
16. Wei, C., Chiang, R., & Wu, C. (2006). Accommodating individual preferences in the categorization of documents: A personalized clustering approach. Journal of Management Information Systems, 23(2), 2006, 173-201.