C++ DASTURLASH TILIDA SAMARALI ALGORITMLAR YARATISH USULLARI

METHODS FOR CREATING EFFICIENT ALGORITHMS IN THE C++ PROGRAMMING LANGUAGE

МЕТОДЫ СОЗДАНИЯ ЭФФЕКТИВНЫХ АЛГОРИТМОВ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ C++

Rizoqulova Marjonabonu Zokirovna
Buxoro Davlat Texnika Universiteti 2 kurs talabasi

Annotatsiya: *Ushbu maqolada C++ dasturlash tilida samarali algoritmlar yaratishning asosiy usullari va tamoyillari yoritilgan. Algoritmlarning vaqt va xotira murakkabligini tahlil qilish, mos ma'lumotlar tuzilmalarini tanlash, STL kutubxonasi imkoniyatlaridan foydalanish, dinamik dasturlash, rekursiya va parallel dasturlash texnologiyalarini qo'llash masalalari ko'rib chiqilgan. Shuningdek, algoritmlarni optimallashtirish, testlash va profil qilish usullari tahlil etilgan. Tadqiqot natijalari C++ dasturlash tilida yuqori samaradorlikka ega dasturiy yechimlar yaratish uchun amaliy tavsiyalar beradi.*

Kalit so'zlar: *C++, algoritm, samaradorlik, Big O notatsiyasi, STL, dinamik dasturlash, ma'lumotlar tuzilmasi, optimallashtirish.*

Аннотация: *В данной статье рассматриваются основные методы и принципы создания эффективных алгоритмов на языке программирования C++. Исследуются вопросы анализа временной и пространственной сложности алгоритмов, выбора подходящих структур данных, использования возможностей библиотеки STL, динамического программирования, рекурсии и параллельного программирования. Также рассматриваются методы оптимизации, тестирования и профилирования алгоритмов. Результаты исследования предоставляют практические рекомендации по разработке высокопроизводительных программных решений на языке C++.*

Ключевые слова: *C++, алгоритм, эффективность, нотация Big O, STL, динамическое программирование, структуры данных, оптимизация.*

Abstract: *This article discusses the main methods and principles for developing efficient algorithms in the C++ programming language. It examines the analysis of time and space complexity, the selection of appropriate data structures, the use of STL library features, dynamic programming, recursion, and parallel programming techniques. In addition, methods for algorithm optimization, testing, and profiling are explored.*

The findings provide practical recommendations for developing high-performance software solutions using C++.

Keywords: *C++, algorithm, efficiency, Big O notation, STL, dynamic programming, data structures, optimization.*

KIRISH

Zamonaviy dasturiy ta'minot ishlab chiqishda algoritmlarning samaradorligi muhim ahamiyat kasb etadi. Dastur qanchalik tez ishlashi va kam resurs sarflashi ko‘p jihatdan algoritmning to‘g‘ri tanlanishi va amalga oshirilishiga bog‘liq.

C++ dasturlash tili yuqori unumdorligi, keng kutubxonalari va apparat resurslari bilan samarali ishlash imkoniyatlari tufayli samarali algoritmlar yaratish uchun eng ommabop tillardan biri hisoblanadi.

Ushbu maqolada C++ dasturlash tilida samarali algoritmlar yaratishning asosiy usullari, tamoyillari va amaliy tavsiyalari ko‘rib chiqiladi.

Samarali algoritm tushunchasi

Samarali algoritm — bu masalani minimal vaqt va xotira sarfi bilan yechadigan algoritmdir. Algoritm samaradorligi odatda quyidagi ikki mezon orqali baholanadi:

• Vaqt murakkabligi (Time Complexity) — algoritm bajarilishi uchun ketadigan vaqt.

• Xotira murakkabligi (Space Complexity) — algoritm ishlashi uchun zarur bo‘lgan qo‘shimcha xotira hajmi.

Masalan, element qidirish masalasida oddiy chiziqli qidiruv algoritmi O(n) murakkablikka ega bo‘lsa, saralangan massivda binar qidiruv O(log n) murakkablik bilan ishlaydi.

Big O notatsiyasidan foydalanish

Algoritmlar samaradorligini tahlil qilishda Big O notatsiyasi keng qo‘llaniladi. U algoritm ishlash vaqtining kiruvchi ma'lumot hajmiga bog‘liqligini ifodalaydi.

Eng ko‘p uchraydigan murakkabliklar:

Murakkablik	Tavsif
O(1)	Doimiy vaqt
O(log n)	Logarifmik vaqt
O(n)	Chiziqli vaqt
O(n log n)	Samarali saralash algoritmlari
$O(n^2)$	Ikki siklli algoritmlar
$O(2^n)$	Eksponensial algoritmlar

Dasturchi imkon qadar past murakkablikdagi algoritmlarni tanlashi lozim.

Mos ma'lumotlar tuzilmasini tanlash

Samarali algoritm yaratishda ma'lumotlar tuzilmasini to‘g‘ri tanlash juda muhim.

C++ STL kutubxonasidagi asosiy konteynerlar:

- vector — dinamik massiv.
- deque — ikki tomonlama navbat.
- list — bog‘langan ro‘yxat.
- stack — stek.
- queue — navbat.
- priority_queue — ustuvor navbat.
- set va multiset — tartiblangan to‘plamlar.
- map va multimap — kalit-qiymat juftliklari.
- unordered_set va unordered_map — xesh-jadvallar.

Masalan, tezkor qidirish talab qilingan hollarda unordered_map dan foydalanish O(1) o‘rtacha murakkablikni ta'minlaydi.

```
unordered_map<int, string> student;
student[1] = "Ali";
student[2] = "Vali";
```

STL algoritmlaridan foydalanish

C++ Standard Template Library (STL) ko‘plab optimallashtirilgan algoritmlarni taqdim etadi.

Misollar:

```
sort(arr.begin(), arr.end());
binary_search(arr.begin(), arr.end(), x);
reverse(arr.begin(), arr.end());
```

STL algoritmlari odatda qo‘lda yozilgan kodlarga nisbatan tezroq va ishonchliroq ishlaydi.

Rekursiyani ehtiyotkorlik bilan qo‘llash

Rekursiya kodni soddalashtiradi, ammo chuqur chaqiruvlar xotira sarfini oshirishi mumkin.

Masalan:

```
int factorial(int n)
{    if(n <= 1)
       return 1;
   return n * factorial(n - 1);}
```

Katta ma'lumotlar bilan ishlashda iterativ usul ko‘pincha samaraliroq bo‘ladi.

Dinamik dasturlash usulidan foydalanish

Dinamik dasturlash (Dynamic Programming) takrorlanuvchi hisob-kitoblarni saqlab qolish orqali algoritm tezligini sezilarli oshiradi.

Misol sifatida Fibonacci sonlarini hisoblash:

```
vector<long long> dp(n + 1);
dp[0] = 0;
dp[1] = 1;
for(int i = 2; i <= n; i++)
   dp[i] = dp[i-1] + dp[i-2];
```

Bu usul oddiy rekursiv yechimning eksponensial murakkabligini O(n) ga kamaytiradi.

Keraksiz nusxalashlardan qochish

C++ da obyektlarni nusxalash qo‘shimcha vaqt va xotira talab qiladi.

Quyidagi usul samaraliroq:

```
void printVector(const vector<int>& v)
{   for(int x : v)       cout << x << " ";}
```

Bu yerda const & parametrni nusxalamasdan uzatadi.

Xotiradan oqilona foydalanish

Katta hajmdagi ma'lumotlar bilan ishlashda xotira sarfini kamaytirish muhimdir.

Masalan:

```
vector<int> arr;
arr.reserve(1000000);
```

reserve() funksiyasi qayta ajratishlar sonini kamaytiradi va ishlash tezligini oshiradi.

Parallel dasturlash imkoniyatlaridan foydalanish

Zamonaviy protsessorlar ko‘p yadroli bo‘lgani sababli parallel hisoblash samaradorlikni oshirishi mumkin.

C++ da oqimlar (threads) bilan ishlash:

```
#include <thread>
void task()
{    cout << "Hello";}
thread t(task);
t.join();
```

Murakkab hisoblashlarni bir nechta oqimlarga bo‘lish orqali bajarish vaqtini kamaytirish mumkin.

Algoritmlarni testlash va profil qilish

Samaradorlikni oshirish uchun algoritm ishlash vaqtini o‘lchash zarur.

```
#include <chrono>
auto start = chrono::high_resolution_clock::now();
/* kod */
auto end = chrono::high_resolution_clock::now();
```

Profil qilish orqali dasturdagi sekin ishlayotgan qismlar aniqlanadi va optimallashtiriladi.

C++ dasturlash tili yuqori unumdorligi, apparat darajasiga yaqin ishlash imkoniyati, kuchli standart kutubxonalari va moslashuvchanligi sababli samarali algoritmlar ishlab chiqishda keng qo‘llaniladi. Ayniqsa, operatsion tizimlar, ma'lumotlar bazalari, o‘yin dasturlari, sun'iy intellekt tizimlari va ilmiy hisob-kitoblarda C++ ning ahamiyati katta.

Ushbu maqolada C++ dasturlash tilida samarali algoritmlar yaratishning nazariy asoslari, optimallashtirish usullari va amaliy tavsiyalari batafsil yoritiladi.

Algoritm va uning samaradorligi

Algoritm — bu ma'lum bir masalani yechish uchun ketma-ket bajariladigan amallar majmui hisoblanadi. Har qanday algoritm quyidagi talablarga javob berishi kerak:

- Aniqlik;
- Tugallanish;
- Natijaviylik;
- Samaradorlik;
- Tushunarlilik.

Samarali algoritm deganda masalani minimal vaqt va minimal xotira sarfi bilan yechadigan algoritm tushuniladi.

Algoritm samaradorligi quyidagi ikki mezon orqali baholanadi:

Vaqt murakkabligi (Time Complexity)

Vaqt murakkabligi algoritm bajarilishi uchun zarur bo‘lgan operatsiyalar sonini ifodalaydi.

Xotira murakkabligi (Space Complexity)

Xotira murakkabligi algoritm ishlashi davomida foydalaniladigan qo‘shimcha xotira hajmini ko‘rsatadi.

Big O notatsiyasi

Algoritmlarni tahlil qilishda Big O notatsiyasi qo‘llaniladi.

Quyidagi murakkabliklar eng samarali hisoblanadi:

Murakkablik	Samaradorlik
O(1)	Juda yuqori
O(log n)	Yuqori
O(n)	Yaxshi
O(n log n)	Qoniqarli
$O(n^2)$	Past
$O(2^n)$	Juda past

Masalan:

```
int x = arr[0];
```

Bu operatsiya O(1) murakkablikka ega.
Chiziqli qidiruv:

```
for(int i=0;i<n;i++)
{    if(arr[i]==x)
      return i;}
```

Bu algoritm O(n) murakkablik bilan ishlaydi.
Ma'lumotlar tuzilmalarining ahamiyati
Algoritm samaradorligi ko‘p hollarda tanlangan ma'lumotlar tuzilmasiga bog‘liq.
Vector

```
vector<int> v;
```

Afzalliklari:
• Tez indekslash O(1)
• Dinamik hajm
• Xotiradan samarali foydalanish
Set

```
set<int> s;
```

Xususiyatlari:
• Avtomatik saralash
• O(log n) qidiruv
Unordered Map

```
unordered_map<int,string> mp;
```

Afzalliklari:
• O‘rtacha O(1) qidiruv
• Kalit bo‘yicha tez murojaat
STL Kutubxonasi imkoniyatlari
STL (Standard Template Library) C++ ning eng kuchli imkoniyatlaridan biridir.
Asosiy qismlari:
1.Containerlar
2. Iteratorlar
3. Algoritmlar
4. Funktorlar
Sort algoritmi

```
sort(v.begin(),v.end());
```

Murakkabligi:
O(n log n)
Binary Search

```
binary_search(v.begin(),v.end(),x);
```

Murakkabligi:

O(log n)

Lower Bound

lower_bound(v.begin(),v.end(),x);

Bu usul elementning birinchi uchrashgan o‘rnini topadi.

Dinamik dasturlash (Dynamic Programming)

Dinamik dasturlash murakkab masalalarni kichik qismlarga bo‘lib yechishga asoslanadi.

Asosiy g‘oya:

Bir marta hisoblangan qiymatni qayta hisoblamaslik.

Fibonacci misoli

Oddiy rekursiya:

```
int fib(int n)
{   if(n<=1) return n;    return fib(n-1)+fib(n-2);}
```

Murakkablik:

$O(2^n)$

Dinamik dasturlash:

```
vector<long long> dp(n+1);
dp[0]=0;
dp[1]=1;
for(int i=2;i<=n;i++)
{   dp[i]=dp[i-1]+dp[i-2];
}
```

Murakkablik:

O(n)

Natijada algoritm yuzlab marta tezlashadi.

Greedy algoritmlar

Greedy (ochko‘z) algoritmlar har qadamda eng yaxshi lokal yechimni tanlaydi.

Qo‘llanilishi:

- Huffman Coding
- Dijkstra algoritmi
- Activity Selection

Misol:

Tangalar yordamida summa yig‘ish.

```
vector<int> coins={100,50,20,10,5,1};
```

Har safar eng katta mos tangani tanlash orqali optimal natija olinadi.

Divide and Conquer usuli

Masalani kichik qismlarga ajratib yechish tamoyiliga asoslanadi.
Misollar:
• Merge Sort
• Quick Sort
• Binary Search
Merge Sort

```
void mergeSort(int l,int r)
{   if(l>=r) return;
   int mid=(l+r)/2;
   mergeSort(l,mid);
   mergeSort(mid+1,r);
   merge(l,mid,r); }
```

Murakkabligi:
O(n log n)
Rekursiya va Iteratsiya
Rekursiya kodni qisqartiradi.
Kamchiliklari:
• Stack overflow xavfi
• Qo‘shimcha xotira sarfi
Masalan factorial:

```
int fact(int n)
{   if(n==0) return 1;
   return n*fact(n-1);}
```

Katta qiymatlar uchun iterativ usul samaraliroq bo‘ladi.
Xotirani optimallashtirish
Katta loyihalarda xotira sarfi juda muhim.
Reserve funksiyasi

```
vector<int> v;
v.reserve(1000000);
```

Afzalliklari:
• Qayta xotira ajratish kamayadi
• Dastur tezlashadi
Move Semantics
C++11 dan boshlab:

```
vector<int> a={1,2,3};
vector<int> b=move(a);
```

Bu nusxalash o‘rniga ma'lumotni ko‘chiradi.

Parallel dasturlash

Ko‘p yadroli protsessorlardan samarali foydalanish uchun multithreading qo‘llaniladi.

```
#include ‹thread›
void task()
{    cout‹‹"Thread ishladi"; }
thread t(task);
t.join();
```

Afzalliklari:

- Hisoblash tezligi oshadi
- Katta ma'lumotlar tezroq qayta ishlanadi

Algoritmlarni testlash va profil qilish

Har qanday optimallashtirishdan oldin algoritmni o‘lchash kerak.

Chrono kutubxonasi

```
#include ‹chrono›
auto start=
chrono::high_resolution_clock::now();
// kod
auto stop=
chrono::high_resolution_clock::now();
```

Bu usul bajarilish vaqtini aniqlash imkonini beradi.

Zamonaviy C++ imkoniyatlari

C++17 va C++20 versiyalarida samaradorlikni oshiruvchi ko‘plab yangiliklar mavjud:

- Smart Pointerlar (unique_ptr, shared_ptr)
- Lambda funksiyalar
- Parallel STL
- constexpr
- Structured Bindings
- Concepts

Misol:

```
auto square=[](int x)
{    return x*x;};
```

Lambda funksiyalar kodni ixchamlashtiradi va tez ishlaydi.

Samarali algoritm yaratish bo‘yicha tavsiyalar

1. Masalani to‘liq tahlil qiling.
2. Eng mos ma'lumotlar tuzilmasini tanlang.
3. STL algoritmlaridan foydalaning.
4. Keraksiz sikllarni kamaytiring.

5. Rekursiyani ehtiyotkorlik bilan qo‘llang.
6. Dinamik dasturlashdan foydalaning.
7. Xotira sarfini nazorat qiling.
8. Kodni profil qiling.
9. Parallel hisoblashlardan foydalaning.
10. Zamonaviy C++ standartlarini o‘rganing.

Xulosa

C++ dasturlash tilida samarali algoritmlar yaratish yuqori unumdor dasturiy ta'minot ishlab chiqishning asosiy shartlaridan biridir. Algoritm murakkabligini tahlil qilish, to‘g‘ri ma'lumotlar tuzilmasini tanlash, STL kutubxonasidan foydalanish, dinamik dasturlash va parallel hisoblash texnologiyalarini qo‘llash orqali dasturlar tezligi va ishonchliligini sezilarli darajada oshirish mumkin. Zamonaviy C++ imkoniyatlaridan oqilona foydalanish esa katta hajmdagi ma'lumotlarni qayta ishlash va murakkab masalalarni samarali yechishga yordam beradi. Ushbu usullarni amaliyotda qo‘llash har bir dasturchining professional mahoratini oshiradi va sifatli dasturiy mahsulotlar yaratishga xizmat qiladi.

FOYDALANILGAN ADABIYOTLAR

1.The C++ Programming Language. 4th Edition. Boston: Addison-Wesley Professional, 2013.

1.Programming: Principles and Practice Using C++. 2nd Edition. Boston: Addison-Wesley Professional, 2014.

2. Effective Modern C++. Sebastopol: O'Reilly Media, 2014.
3. C++ Primer. 5th Edition. Addison-Wesley Professional, 2012.
4. Introduction to Algorithms. 4th Edition. Cambridge: MIT Press, 2022.
5. Algorithm Design. Boston: Addison-Wesley, 2005.
6. Competitive Programming 4. CP4 Book Series, 2022.
7. cplusplus.com Documentation — C++ sintaksisi, STL va standart kutubxonalar bo‘yicha ma’lumotlar.
8. cppreference.com Documentation — C++ standartlari va STL bo‘yicha eng to‘liq texnik manba.
9. ISO C++ Foundation — C++ standarti va zamonaviy C++ texnologiyalari bo‘yicha rasmiy ma’lumotlar.
10. Data Structures and Algorithm Analysis in C++. 4th Edition. Pearson Education, 2013.
11. Algorithms. 4th Edition. Addison-Wesley Professional, 2011.