Ko‘p o‘lchovli massivlar va ularning funksiyalarini tadqiq qilish, ularning o‘ziga xosligini hisobga olish

Gulbodom Oybek qizi Norqulova
BXU

Annotatsiya: Ushbu maqolada ko‘p o‘lchovli massivlar va ularning funksiyalarini kompleks tadqiq qilish amalga oshirilgan. Massivlarning matematik modellari, tuzilishi, xotira tashkil etilishi va asosiy operatsiyalar to‘liq tahlil qilingan. Tadqiqotda bir, ikki va yuqori o‘lchovli massivlar, ularning xususiyatlari, qo‘llanish sohalari va amaliy dasturlashdagi ahamiyati o‘rganilgan. Maqolada massiv funksiyalarining klassifikatsiyasi, yaratish va initsializatsiya, kirish va o‘zgartirish, matematik operatsiyalar, qidiruv va saralash algoritmlari batafsil ko‘rib chiqilgan. Massivlar bilan ishlashning algoritmik murakkabligi, xotira samaradorligi va optimallash strategiyalari tahlil qilingan. Tadqiqot natijalari turli dasturlash tillarida (Python, Java, C++) massivlarni qo‘llash usullari va zamonaviy hisoblash texnologiyalarida ularning roli haqida keng ma’lumot beradi. Didaktik yondashuv asosida massivlarni o‘qitish metodikasi ishlab chiqilgan.

Kalit so‘zlar: ko‘p o‘lchovli massivlar, massiv funksiyalari, ma’lumotlar strukturasi, algoritmik murakkablik, xotira tashkil etilishi, indekslash tizimlari, matritsali operatsiyalar, massiv algoritmlari, dasturlash, ma’lumotlar qayta ishlash, tensorlar, massiv optimallashuvi, hisoblash samaradorligi

Study of Multidimensional Arrays and Their Functions Considering Their Specific Features

Gulbodom Oybek qizi Norqulova
BXU

Abstract: This article presents a comprehensive study of multidimensional arrays and their functions, taking into account their specific characteristics. The mathematical models of arrays, their structure, memory organization, and fundamental operations are thoroughly analyzed. The research examines one-dimensional, two-dimensional, and higher-dimensional arrays, their properties, areas of application, and importance in applied programming. The paper provides an in-depth discussion of array function classification, creation and initialization, access and modification, mathematical operations, search and sorting algorithms. The algorithmic complexity of array processing, memory efficiency, and optimization strategies are analyzed. The research

findings offer extensive information on the use of arrays in various programming languages (Python, Java, C++) and their role in modern computational technologies. A didactic approach is employed to develop a methodology for teaching arrays.

Keywords: multidimensional arrays, array functions, data structures, algorithmic complexity, memory organization, indexing systems, matrix operations, array algorithms, programming, data processing, tensors, array optimization, computational efficiency

Kirish

Ko‘p o‘lchovli massivlar zamonaviy dasturlash va hisoblash texnologiyalarining asosiy ma’lumotlar strukturasi hisoblanadi. Ular katta hajmdagi tartibli ma’lumotlarni saqlash, tashkil etish va samarali qayta ishlash imkonini beradi. Massivlar tushunchasi dasturlashning dastlabki kunlaridan boshlab mavjud bo‘lsa-da, zamonaviy ilovalarda ularning ahamiyati tobora ortib bormoqda.

Massiv - bu bir xil turdagi elementlarning tartiblangan to‘plami bo‘lib, har bir elementga indeks orqali murojaat qilish mumkin. Bir o‘lchovli massiv chiziqli ketma-ketlik, ikki o‘lchovli massiv jadval yoki matritsa, uch va undan yuqori o‘lchovli massivlar esa tensorlar sifatida qaralishi mumkin. Massivlarning asosiy afzalligi - elementlarga tezkor murojaat, ya’ni O(1) vaqt murakkabligi bilan istalgan elementni olish imkoniyati.

Hozirgi kunda ko‘p o‘lchovli massivlar turli sohalarda keng qo‘llaniladi. Ilmiy hisoblashlarda chiziqli algebra operatsiyalari, matritsali hisoblashlar va raqamli tahlil massivlarsiz tasavvur qilinmaydi. Tasvirlarni qayta ishlashda har bir piksel massiv elementi sifatida saqlanadi va rangli tasvirlar uch o‘lchovli massivlar (tinglik × kenglik × rang kanallari) bilan ifodalanadi. Sun’iy intellekt va chuqur o‘rganishda neyron tarmoqlar og‘irliklari va faollashtirish qiymatlari tensorlar (ko‘p o‘lchovli massivlar) sifatida saqlanadi.

Ma’lumotlar tahlilida katta hajmdagi jadval ko‘rinishidagi ma’lumotlar ikki o‘lchovli massivlarda saqlanadi va turli statistik operatsiyalar qo‘llaniladi. Video qayta ishlashda har bir kadr ikki o‘lchovli massiv bo‘lib, butun video to‘rt o‘lchovli struktura (vaqt × tinglik × kenglik × rang) sifatida qaralishi mumkin. O‘yinlar dasturlashda o‘yin maydoni, xaritalar va ob’ektlar holatlari massivlar orqali ifodalanadi.

Massiv funksiyalari - bu massivlar ustida bajariladigan operatsiyalar majmuasi bo‘lib, ular yaratish, o‘zgartirish, qidiruv, saralash, matematik hisoblashlar va boshqa vazifalarni o‘z ichiga oladi. Har bir dasturlash tili o‘z standart kutubxonasida massiv funksiyalarini taqdim etadi, lekin ularning tuzilishi, ishlash mexanizmi va samaradorligi turlicha bo‘lishi mumkin.

Tadqiqotning maqsadi ko‘p o‘lchovli massivlar va ularning funksiyalarini to‘liq tadqiq qilish, nazariy asoslarini yaratish, amaliy qo‘llanishlarni ko‘rsatish va samarali

dasturlash yo‘l-yo‘riqlarini ishlab chiqishdan iborat. Tadqiqot ob’ekti sifatida turli o‘lchovdagi massivlar, ularning xususiyatlari, operatsiyalari va turli dasturlash tillarida amalga oshirilishi tanlab olingan. Tadqiqot metodologiyasi nazariy tahlil, algoritmik baholash, eksperimental o‘lchash va qiyosiy tahlilni o‘z ichiga oladi.

Asosiy qism

Massivlarning nazariy asoslari va matematik modeli

Massiv formal ravishda xaritalash sifatida aniqlanishi mumkin: A: I → T, bu yerda I - indekslar to‘plami, T - elementlar turi. Bir o‘lchovli massiv uchun I = {0, 1, 2, ..., n-1} yoki I = {1, 2, ..., n}. Ikki o‘lchovli massiv uchun I = {0, ..., m-1} × {0, ..., n-1}. Umumiy n-o‘lchovli massiv uchun $I = I_1 \times I_2 \times ... \times I_n$ (Dekart ko‘paytma).

Matematik jihatdan, bir o‘lchovli massiv vektor, ikki o‘lchovli massiv matritsa, yuqori o‘lchovli massiv esa tensor sifatida qaraladi. Vektor $v \in R^n$ n ta haqiqiy sonlardan iborat bo‘lib, $v = (v_1, v_2, ..., v_n)$ ko‘rinishda yoziladi. Matritsa A ∈ R^(m×n) m qator va n ustundan tashkil topgan: $A = [a_{ij}]$, bu yerda i = 1,...,m va j = 1,...,n.

Tensor T ko‘p indeksli ob’ekt bo‘lib, umumiy ko‘rinishda $T = [t_{i1i2...in}]$ ko‘rinishida yoziladi. Masalan, uch o‘lchovli tensor T ∈ R^(l×m×n) uchta indeksga ega: $T = [t_{ijk}]$, bu yerda i = 1,...,l, j = 1,...,m, k = 1,...,n. Tensorlar fizikada (stress-strain tensorlari), sun’iy intellektda (chuqur o‘rganish) va signallar qayta ishlashda keng qo‘llaniladi.

Massivning asosiy xususiyatlari quyidagilardan iborat. O‘lchovlilik (dimensionality) - massivning nechta indeks bilan aniqlanishi. Shakl (shape) - har bir o‘lcham bo‘yicha elementlar soni, masalan, (3, 4, 5) shaklda uch o‘lchovli massiv. Hajm (size) - umumiy elementlar soni, shakl komponentlarining ko‘paytmasi: 3 × 4 × 5 = 60. Ma’lumot turi (dtype) - har bir elementning turi (int, float, char va boshqalar).

Massiv indekslash tizimlari dasturlash tillariga bog‘liq. 0-asosli indekslash (C, Java, Python) 0 dan boshlanadi: A[0], A[1], ..., A[n-1]. 1-asosli indekslash (MATLAB, Fortran, Lua) 1 dan boshlanadi: A(1), A(2), ..., A(n). Ko‘p o‘lchovli massivlarda qator-ustun tartibi (row-major vs column-major) muhim: C/C++/Python da qator-ustun, MATLAB/Fortran da ustun-qator.

Xotira tashkil etilishi va saqlash usullari

Massivlar xotirada uzluksiz blok sifatida saqlanadi, bu tezkor murojaat imkonini beradi. Bir o‘lchovli massiv A[n] uchun xotira tuzilmasi oddiy: ketma-ket joylashgan n ta element. Har bir elementning manzili: address(A[i]) = base_address + i × element_size, bu yerda base_address - massivning boshlang‘ich manzili, element_size - bitta elementning xotira hajmi (baytlarda).

Ikki o‘lchovli massiv A[m][n] uchun ikki asosiy saqlash usuli mavjud. Qator-ustun tartibi (row-major order, C tili) qatorlar ketma-ket saqlanadi: A[0][0], A[0][1], ..., A[0][n-1], A[1][0], A[1][1], ... Element manzili: address(A[i][j]) = base_address + (i ×

n + j) × element_size. Bu tartib qatorlar bo‘yicha iteratsiya qilishda kesh samaradorligini oshiradi.

Ustun-qator tartibi (column-major order, Fortran/MATLAB) ustunlar ketma-ket saqlanadi: A[0][0], A[1][0], ..., A[m-1][0], A[0][1], A[1][1], ... Element manzili: address(A[i][j]) = base_address + (j × m + i) × element_size. Bu tartib ustunlar bo‘yicha iteratsiya qilishda samaraliroq. Saqlash tartibini bilish algoritmlarni optimallashtirish uchun juda muhim.

Yuqori o‘lchovli massivlar uchun umumiy formula: n-o‘lchovli massiv $A[d_1][d_2]...[d_n]$ ning elementi $A[i_1][i_2]...[i_n]$ manzili qator-ustun tartibida: address = base_address + ((...((i₁ × d₂ + i₂) × d₃ + i₃)...) × d_n + i_n) × element_size. Bu rekursiv formula har bir o‘lcham bo‘yicha offset hisoblaydi.

Stride tushunchasi xotira tashkil etilishida muhim. Stride - har bir o‘lcham bo‘yicha keyingi elementga o‘tish uchun zarur baytlar soni. Bir o‘lchovli massivda stride = element_size. Ikki o‘lchovli massivda qator stride’i = n × element_size, ustun stride’i = element_size (row-major tartibda). Stride’lar massiv shaklini o‘zgartirish (reshape) operatsiyasida o‘zgaradi.

Xotira joylashuvining ahamiyati kesh xotiradan samarali foydalanishda namoyon bo‘ladi. Zamonaviy protsessorlar ma’lumotlarni kichik bloklarda (kesh chiziqlari, odatda 64 bayt) yuklaydi. Agar massiv elementlariga xotirada joylashgan tartibda murojaat qilsak, kesh hit rate yuqori bo‘ladi. Aksincha, tasodifiy murojaat yoki noto‘g‘ri tartibda murojaat kesh miss’larni oshiradi va sekinlashtiradi.

Asosiy massiv funksiyalari va ularning tasnifi

Massiv funksiyalari maqsadiga ko‘ra bir nechta asosiy toifalarga bo‘linadi. Yaratish va initsializatsiya funksiyalari yangi massiv ob’ektlarini hosil qiladi va boshlang‘ich qiymatlar bilan to‘ldiradi. Kirish va o‘zgartirish funksiyalari elementlarga murojaat va ularni yangilash imkonini beradi. Matematik operatsiyalar funksiyalari arifmetik va algebraik amallarni bajaradi.

Qidiruv funksiyalari ma’lum elementni yoki shartga mos elementlarni topadi. Saralash funksiyalari elementlarni tartibda joylashtiradi. Transformatsiya funksiyalari massiv shaklini o‘zgartiradi. Agregatsiya funksiyalari statistik hisoblashlarni amalga oshiradi (yig‘indi, o‘rtacha, maksimum). Filtratsiya funksiyalari shartga mos elementlarni tanlaydi. Birlashtirish funksiyalari bir nechta massivlarni birlashtiriladi.

Python NumPy kutubxonasida massiv yaratish funksiyalari: numpy.array() - listdan massiv yaratish, numpy.zeros(shape) - nollar bilan, numpy.ones(shape) - birlar bilan, numpy.empty(shape) - initsializatsiyasiz, numpy.full(shape, value) - berilgan qiymat bilan, numpy.arange(start, stop, step) - ketma-ketlik, numpy.linspace(start, stop, num) - chiziqli bo‘lingan, numpy.eye(n) - birlik matritsasi, numpy.random.random(shape) - tasodifiy sonlar.

Java tilida massiv yaratish: int[] array = new int[n] - bir o‘lchovli, int[][] matrix = new int[m][n] - ikki o‘lchovli, int[] array = {1, 2, 3} - boshlang‘ich qiymatlar bilan. Arrays sinfi qo‘shimcha funksiyalarni taqdim etadi: Arrays.fill(array, value), Arrays.copyOf(array, length), Arrays.sort(array), Arrays.binarySearch(array, key).

C++ da massiv yaratish: int array[n] - statik, int* array = new int[n] - dinamik, std::vector<int> vec(n) - vektor konteyner, std::array<int, n> arr - std::array. Vektorlar o‘lchamni dinamik o‘zgartirish imkonini beradi: vec.push_back(value), vec.pop_back(), vec.resize(new_size). STL algoritmlari (std::sort, std::find, std::copy) massivlar va konteynerlar uchun umumiy.

Indekslash va slicing operatsiyalari

Indekslash massiv elementlariga murojaat qilishning asosiy usuli. Oddiy indekslash: array[i] bir o‘lchovli massivda, matrix[i][j] ikki o‘lchovli massivda. Manfiy indekslash (Python) oxiridan hisoblash: array[-1] oxirgi element, array[-2] oxiridan ikkinchi. Ko‘p o‘lchovli indekslash: array[i, j] yoki array[i][j] sintaksisi.

Slicing (kesish) massivning bir qismini ajratib olish. Oddiy slicing: array[start:stop] start dan stop gacha (stop kiritmay). Qadamli slicing: array[start:stop:step] har step-elementni olish. To‘liq o‘lcham: array[:] barcha elementlar. Boshidan kesish: array[:n] birinchi n ta element. Oxiridan kesish: array[n:] n-elementdan oxirigacha. Teskari tartiblash: array[::-1].

Ko‘p o‘lchovli slicing: matrix[start:stop, :] qatorlarni kesish, matrix[:, start:stop] ustunlarni kesish, matrix[::2, ::2] har ikkinchi qator va ustun. Numpy da advanced indexing imkoniyatlari keng: boolean indexing (array[array > 0] musbat elementlar), fancy indexing (array[[0, 2, 4]] tanlangan indekslar), masking (array[mask] boolean mask bo‘yicha).

Slicing operatsiyasining ichki mexanizmi: NumPy da slicing ko‘pincha view (ko‘rinish) qaytaradi - yangi xotira ajratilmaydi, faqat stride va offset o‘zgaradi. Bu xotirani tejaydi va tezroq ishlaydi. Lekin o‘zgartirish asl massivga ta’sir qiladi. Copy (nusxa) yaratish uchun .copy() metodidan foydalanish kerak. Java va C++ da slicing tabiiy qo‘llab-quvvatlanmaydi, lekin kutubxonalar orqali amalga oshirilishi mumkin.

Ellipsis (...) ko‘p o‘lchovli massivlarda barcha o‘lchamlarni qamrab olish uchun: array[..., 0] oxirgi o‘lchamning birinchi elementi, array[0, ...] birinchi o‘lchamning birinchi elementi. Newaxis (None) yangi o‘lcham qo‘shish uchun: array[np.newaxis, :] (1, n) shaklga o‘zgartiradi. Bu broadcasting uchun foydali.

Matematik operatsiyalar va chiziqli algebra

Elementma-element operatsiyalar (element-wise) har bir elementga alohida qo‘llaniladi. Arifmetik: A + B, A - B, A * B (element-wise ko‘paytirish), A / B, A ** n (darajaga ko‘tarish). Taqqoslash: A > B, A == B, A != B (boolean massiv qaytaradi). Mantiqiy: A & B (and), A | B (or), ~A (not).

Broadcasting mexanizmi turli shaklдagi massivlarni operatsiyalarda ishlatish imkonini beradi. Qoidalar: (1) O‘lchamlar soni tenglanadi (kichikroq massivga 1 o‘lchamlari qo‘shiladi). (2) Har bir o‘lcham bo‘yicha o‘lchamlar teng bo‘lishi yoki biri 1 bo‘lishi kerak. (3) Natija shakli - har bir o‘lcham bo‘yicha maksimum. Masalan, (3, 1) va (1, 4) → (3, 4).

Chiziqli algebra operatsiyalari: Matritsalarni ko‘paytirish: C = A @ B yoki np.matmul(A, B), bu yerda A(m×n), B(n×p) → C(m×p). Transpozitsiya: A.T yoki np.transpose(A). Determinant: np.linalg.det(A). Teskari matritsa: np.linalg.inv(A). Xos qiyatlar: np.linalg.eig(A). Singular qiymatlar: np.linalg.svd(A).

Vektor operatsiyalari: Nuqtali ko‘paytma (dot product): np.dot(a, b) yoki a @ b. Ichki ko‘paytma: np.inner(a, b). Tashqi ko‘paytma (outer product): np.outer(a, b) - matritsa hosil qiladi. Xoch ko‘paytma (cross product): np.cross(a, b) (faqat 3D vektorlar uchun). Norma: np.linalg.norm(a) Evklid normasi.

Matritsali yoyilmalar: LU yoyilmasi: A = LU (scipy.linalg.lu). QR yoyilmasi: A = QR (np.linalg.qr). Cholesky yoyilmasi: A = LL^T musbat aniq matritsalar uchun (np.linalg.cholesky). SVD: A = UΣV^T (np.linalg.svd). Bu yoyilmalar chiziqli tizimlarni yechish, eng kichik kvadratlar masalasi va boshqa qo‘llanmalarda ishlatiladi.

Tensorli operatsiyalar: Tensorni qisqartirish (contraction): np.tensordot(A, B, axes). Einstein yozuvi (np.einsum) murakkab tensorli operatsiyalarni qisqa ifoda qilish uchun. Masalan, np.einsum(‘ij,jk->ik’, A, B) matritsalarni ko‘paytirish. Tensorli ko‘paytma: np.kron(A, B) Kroneker ko‘paytmasi.

Qidiruv va saralash algoritmlari

Chiziqli qidiruv (linear search) eng oddiy usul: ketma-ket barcha elementlarni tekshirish. Vaqt murakkabligi O(n). Python da: element in array, array.index(element). NumPy da: np.where(array == value) barcha indekslarni qaytaradi. Afzalligi: massiv tartiblangan bo‘lishi shart emas. Kamchiligi: katta massivlarda sekin.

Ikkilik qidiruv (binary search) tartiblangan massivlarda ishlaydi: o‘rtadagi element bilan taqqoslash va qidirish oralig‘ini ikki baravarga kamaytirish. Vaqt murakkabligi O(log n). Python da: bisect modulidan bisect_left, bisect_right. NumPy da: np.searchsorted(array, value). Shartlar: massiv tartiblangan bo‘lishi kerak.

Saralash algoritmlari turli murakkablik va xususiyatlarga ega. Bubble sort: $O(n^2)$, oddiy lekin sekin. Insertion sort: $O(n^2)$ eng yomon holat, O(n) yaxshi tartiblangan uchun. Selection sort: $O(n^2)$, minimal almashtirishlar. Merge sort: O(n log n), barqaror, qo‘shimcha xotira talab qiladi. Quick sort: O(n log n) o‘rtacha, $O(n^2)$ eng yomon, joyida, barqaror emas. Heap sort: O(n log n), joyida, barqaror emas.

Python da: sorted(array) yangi tartiblangan list qaytaradi, array.sort() joyida saralaydi. NumPy da: np.sort(array) yangi massiv, array.sort() joyida. Tartib: ascending (default) yoki descending ([::-1] yoki sort_order parametr). Maxsus kalit funksiya: sorted(array, key=lambda x: custom_key(x)).

Ko‘p o‘lchovli massivlarni saralash: axis parametri bo‘yicha. np.sort(matrix, axis=0) ustunlarni saralaydi, np.sort(matrix, axis=1) qatorlarni saralaydi. Leksikografik saralash: np.lexsort((array2, array1)) birinchi array1, keyin array2 bo‘yicha. Structured array’lar bir nechta maydon bo‘yicha saralash imkonini beradi.

Partial sort (qisman saralash): faqat k ta eng kichik/katta elementni topish to‘liq saralashdan tezroq. np.partition(array, k) k-indeksga nisbatan partitsiya qiladi: kichiklar chapda, kattalar o‘ngda, lekin har bir qism ichida tartib yo‘q. np.argpartition() indekslarni qaytaradi. Vaqt murakkabligi O(n), to‘liq saralashdan (O(n log n)) tezroq.

Agregatsiya va statistik funksiyalar

Yig‘indi va o‘rtacha: np.sum(array) barcha elementlar yig‘indisi, np.mean(array) o‘rtacha qiymat, np.average(array, weights) og‘irlikli o‘rtacha. Axis parametri: np.sum(matrix, axis=0) har bir ustun yig‘indisi, np.sum(matrix, axis=1) har bir qator yig‘indisi. keepdims=True o‘lchamlarni saqlaydi.

Ekstremal qiymatlar: np.min(array), np.max(array) minimal va maksimal elementlar. np.argmin(array), np.argmax(array) minimal/maksimal elementlarning indekslari. np.nanmin(array), np.nanmax(array) NaN qiymatlarni e’tiborsiz qoldiradi. np.ptp(array) (peak-to-peak) diapazon: max - min.

Dispersiya va standart og‘ish: np.var(array) dispersiya, np.std(array) standart og‘ish. ddof parametri (delta degrees of freedom): ddof=0 populatsiya dispersiya (default), ddof=1 tanlanma dispersiya. Formula: var = $\Sigma(x - mean)^2 / n$, std = $\sqrt{var}$. Dispersiya ma’lumotlarning tarqalishini o‘lchaydi.

Mediana va kvantillar: np.median(array) mediana (50-percentil). np.percentile(array, q) q-percentil. np.quantile(array, q) kvantil ($0 \leq q \leq 1$). Mediana chetki qiymatlarga o‘rtachadan kam ta’sirlanadi. Interquartile range (IQR) = Q3 - Q1 markaziy 50% diapazon.

Korrelyatsiya va kovariatsiya: np.corrcoef(x, y) korrelyatsiya koeffitsiyenti (-1 dan 1 gacha, 0 - bog‘liqlik yo‘q). np.cov(x, y) kovariatsiya matritsasi. np.correlate(x, y) diskret korrelyatsiya (signal processing). Korrelyatsiya ikki o‘zgaruvchi orasidagi chiziqli bog‘liqlikni o‘lchaydi.

Kumulyativ operatsiyalar: np.cumsum(array) kumulyativ yig‘indi, np.cumprod(array) kumulyativ ko‘paytma. np.diff(array) qo‘shni elementlar ayirmasi. Gradient: np.gradient(array) raqamli gradient. Bu funksiyalar vaqt qatorlari va signallarni tahlil qilishda foydali.

Transformatsiya va shakl o‘zgartirish

Reshape (shakl o‘zgartirish): np.reshape(array, new_shape) yoki array.reshape(new_shape). Umumiy elementlar soni bir xil bo‘lishi kerak: $m \times n = p \times q$. -1 parametri avtomatik hisoblash: array.reshape(-1) bir o‘lchovli, array.reshape(3, -1) 3 qator, ustunlar avtomatik. Reshape ko‘pincha view qaytaradi (xotira ko‘chirilmaydi).

Flatten va ravel: array.flatten() massivni bir o‘lchovli qiladi va copy qaytaradi. array.ravel() ham bir o‘lchovli qiladi, lekin view qaytaradi (mumkin bo‘lsa). ravel tezroq, lekin o‘zgartirish asl massivga ta’sir qiladi. Farq: flatten - doimo copy, ravel - view (agar mumkin bo‘lsa).

Transpose: array.T yoki np.transpose(array) matritsani transpozitsiya qiladi (qator ↔ ustun). Ko‘p o‘lchovli uchun axes parametri: np.transpose(array, (2, 0, 1)) o‘lchamlar tartibini o‘zgartiradi. Transpozitsiya view qaytaradi, xotira ko‘chirilmaydi, faqat stride’lar o‘zgaradi.

Squeeze va expand_dims: np.squeeze(array) 1 o‘lchamdagi o‘lchamlarni olib tashlaydi, (1, 3, 1, 5) → (3, 5). np.expand_dims(array, axis) yangi 1 o‘lchamli o‘lcham qo‘shadi. array[:, np.newaxis] ham shu maqsadda ishlatiladi. Bu broadcasting uchun kerakli shaklni yaratishda foydali.

Concatenate va stack: np.concatenate([array1, array2], axis) massivlarni birlashtiradi mavjud o‘lcham bo‘yicha. np.vstack([array1, array2]) vertikal (qatorlar bo‘yicha), np.hstack([array1, array2]) gorizontal (ustunlar bo‘yicha). np.stack([array1, array2], axis) yangi o‘lcham yaratib birlashtiradi. np.column_stack(), np.row_stack() qulaylik funksiyalari.

Split: np.split(array, indices_or_sections, axis) massivni bo‘laklarga ajratadi. np.array_split() teng bo‘lmagan bo‘laklar uchun. np.vsplit(), np.hsplit() vertikal va gorizontal bo‘lish uchun maxsus. Split concatenate ga teskari operatsiya.

Xotira samaradorligi va optimallash

View vs copy farqi juda muhim. View - asl massiv bilan xotirani bo‘lishadi, copy - mustaqil massiv yaratadi. View operatsiyalari: slicing (array[1:5]), transpose (array.T), reshape (ba’zi hollarda). Copy operatsiyalari: fancy indexing (array[[1,3,5]]), fancy slicing ba’zi hollarda. .copy() metodi aniq copy yaratadi.

Kontiguity (uzluksizlik): C-contiguous - qator-ustun tartibda uzluksiz, Fortran-contiguous - ustun-qator tartibda. np.ascontiguousarray(array) C-contiguous qiladi, np.asfortranarray(array) Fortran-contiguous. Uzluksiz massivlar ba’zi operatsiyalarda tezroq. array.flags[‘C_CONTIGUOUS’] tekshirish.

In-place operatsiyalar xotirani tejaydi: array += 1 (yangi massiv yaratmaydi), array *= 2. out parametri: np.add(a, b, out=result) natijani mavjud massivga yozadi. Ufunc metodlari: np.add.accumulate(), np.add.reduce(), np.add.at() maxsus operatsiyalar uchun.

Xotira bo‘shatish: del array Python da referencelarni olib tashlaydi, lekin xotirani darhol bo‘shatmaydi (garbage collector hal qiladi). NumPy da katta massivlardan foydalangandan keyin del qilish yaxshi amaliyot. Memmap (memory-mapped) fayllar juda katta massivlar uchun: np.memmap() disk bilan integratsiya.

Dtype optimalasi: to‘g‘ri ma’lumot turini tanlash xotirani tejaydi. float64 o‘rniga float32 (agar aniqlik yetarli bo‘lsa) xotirani ikki baravarga kamaytiradi. int8 (-128 to

127) kichik butun sonlar uchun, uint8 (0 to 255) musbat sonlar uchun. array.astype(new_dtype) tur o‘zgartirish, lekin copy yaratadi.

Vektorizatsiya Python loop’larini NumPy operatsiyalari bilan almashtirish orqali tezlashtiradi. Yomon: for i in range(n): result[i] = array[i] * 2. Yaxshi: result = array * 2. Vektorizatsiya C darajasida bajariladigan optimallashtirilgan koddan foydalanadi va 10-100x tezroq bo‘lishi mumkin.

Broadcasting to‘g‘ri qo‘llash ham samaradorlikni oshiradi: explicit loop o‘rniga broadcasting ishlatish. Misol: matrix + vector (vector matritsa qatorlar soniga broadcasting qilinadi). Lekin ehtiyot bo‘lish kerak: haddan tashqari broadcasting xotirani ortiqcha ishlatishi mumkin.

Turli dasturlash tillarida massivlar

Python (NumPy): Eng mashhur ilmiy hisoblashlar kutubxonasi. array obyekti ndarray sinfidan. Afzalliklari: qulay sintaksis, boy funksiyalar to‘plami, C/Fortran bilan integratsiya (tezlik). Kamchiliklari: Python interpretatori overhead, GIL parallel hisoblashni cheklaydi. NumPy massivlari immutable emas - o‘zgartirilishi mumkin.

Python (Pandas): Ma’lumotlar tahlili uchun, DataFrame (ikki o‘lchovli, ustunlar turli turdagi). Series (bir o‘lchovli, indeksli). NumPy ustiga qurilgan, lekin yuqori darajadagi abstraktsiya. SQL-ga o‘xshash operatsiyalar: groupby, merge, join. Time series uchun maxsus qo‘llab-quvvatlash.

Java: Primitive massivlar (int[], double[][]) va object massivlari (String[]). Arrays utility sinfi: Arrays.sort(), Arrays.binarySearch(), Arrays.fill(), Arrays.equals(). ArrayList - dinamik o‘lcham. Multidimensional: Array of arrays (int[][]) - jagged arrays (qatorlar turli uzunlikda bo‘lishi mumkin). Apache Commons Math - ilmiy hisoblashlar uchun.

C++: C-style massivlar (int array[10]), std::array<int, 10> (fixed size), std::vector<int> (dynamic). STL algoritmlari: std::sort, std::find, std::copy. Eigen kutubxonasi - chiziqli algebra, matritsali operatsiyalar. Armadillo - MATLAB-ga o‘xshash sintaksis. Template programming - umumiy kod yozish imkoniyati.

MATLAB: Matritsa-markaziy muhit, har narsa matritsa. Built-in funksiyalar juda ko‘p: matrix operatsiyalari, visualizatsiya, signallar qayta ishlash. 1-based indexing. Colon operator: A(:, 1) birinchi ustun, A(1:3, :) birinchi uchta qator. Broadcast avtomatik. Afzalligi: qulay, kuchli visualizatsiya. Kamchiligi: litsenziya to‘lovi, umumiy dasturlash uchun cheklangan.

JavaScript (Typed Arrays): Float32Array, Int32Array va boshqalar - binary data bilan ishlash uchun. Web Workers - parallel hisoblash. Math.js - ilmiy hisoblashlar kutubxonasi. TensorFlow.js - mashinali o‘rganish browser’da. WebGL - GPU hisoblashlari.

Parallel va GPU hisoblashlari

NumPy parallelizatsiya: Ko‘plab NumPy operatsiyalari avtomatik parallel (BLAS/LAPACK orqali). Threadlar soni: os.environ["OMP_NUM_THREADS"] = "4". Lekin Python GIL multiprocessing talab qilishi mumkin. Joblib, multiprocessing modullari parallel loop’lar uchun.

GPU hisoblashlari CuPy bilan: NumPy-ga o‘xshash API, lekin GPU da. import cupy as cp, cp.array(), cp.matmul(). Host (CPU) dan device (GPU) ga ma’lumotlarni ko‘chirish: cp.asarray(numpy_array). Natijani qaytarish: cp.asnumpy(cupy_array). Afzalligi: katta massivlar uchun 10-100x tezroq.

TensorFlow va PyTorch: Chuqur o‘rganish framework’lari, lekin umumiy tensor operatsiyalari uchun ham. Avtomatik differentsiatsiya, GPU qo‘llab-quvvatlash. Computational graph - operatsiyalar grafiki, optimallash imkoniyati. Eager execution vs graph mode: PyTorch eager (Python kabi), TensorFlow ikkalasi ham (TF 2.0 dan boshlab eager default).

Dask: Parallel NumPy - katta massivlar uchun. dask.array - NumPy API, lekin lazy evaluation va chunking. Task scheduler - parallel bajarish. Distributed computing - bir nechta mashinalar. Out-of-core computation - xotiradan kattaroq ma’lumotlar bilan ishlash. Pandas integration (dask.dataframe).

OpenMP (C/C++): Parallel loop’lar uchun. #pragma omp parallel for - loop parallel bajariladi. Shared memory parallelism. Threadlar soni: omp_set_num_threads(). Reduction: #pragma omp parallel for reduction(+:sum) - har bir thread o‘z summani hisoblaydi, keyin birlashtiriladi.

MPI (Message Passing Interface): Distributed memory parallelism - bir nechta mashinalar. mpi4py - Python uchun. Scatter, gather, broadcast - ma’lumotlarni taqsimlash. Point-to-point va collective communication. Supercomputer’larda keng qo‘llaniladi.

Amaliy qo‘llanmalar va misollar

Tasvirlarni qayta ishlash: Tasvir = (tinglik × kenglik × 3) massiv, RGB kanallari. Grayscale konvertatsiya: gray = 0.299*R* + *0.587*G + 0.114*B. Filtratsiya: konvolyutsiya operatsiyasi, masalan, Gaussian blur. Edge detection: Sobel, Canny algoritmlari. Transformatsiya: rotate, crop, resize. PIL/Pillow, OpenCV kutubxonalari.

Vaqt qatorlari tahlili: Vaqt qatorlari = bir o‘lchovli massiv vaqt bo‘yicha. Moving average: np.convolve(data, weights, mode=‘valid’). Trend extraction: linear regression. Seasonality: Fourier transformatsiya. Autocorrelation: np.correlate(). Pandas - time series uchun maxsus qo‘llab-quvvatlash: rolling, resampling.

Ma’lumotlar tahlili: Jadval = ikki o‘lchovli massiv (qatorlar = ob’ektlar, ustunlar = xususiyatlar). Normalizatsiya: (x - mean) / std. Missing values: np.isnan(), interpolation. Outliers: IQR usuli, Z-score. Feature engineering: polynomial features, binning. Sklearn - machine learning, preprocessing.

Chuqur o‘rganish: Neyron tarmoq og‘irliklari = matritsalar. Forward pass: z = Wx + b, a = activation(z). Backpropagation: gradientlar hisoblash. Mini-batch: (batch_size × features) massiv. Convolutional layers: 4D tensorlar (batch × channels × height × width). PyTorch, TensorFlow framework’lari.

Simulatsiya va Monte-Carlo: Tasodifiy massivlar generatsiya: np.random.random(), np.random.normal(). Physics simulation: zarrachalar pozitsiyalari massivda. Monte Carlo integration: tasodifiy nuqtalar, o‘rtachani hisoblash. Financial modeling: stock prices simulation. Agent-based models: har bir agent - massiv elementi.

Xatolar va debugging

Index out of bounds: Eng keng tarqalgan xato - indeks massiv chegarasidan tashqarida. Python: IndexError. Java: ArrayIndexOutOfBoundsException. C++: undefined behavior (segmentation fault). Yechim: indekslarni tekshirish, assert statement, boundary checks.

Shape mismatch: Operatsiyalarda massivlar shakli mos kelmasa. Broadcasting qoidalarini buzish. Misol: (3, 4) + (5,) - xato. Yechim: reshape, broadcast_to, shaklni tekshirish. ValueError: operands could not be broadcast together.

Memory errors: Juda katta massiv yaratishga urinish. MemoryError (Python). Segmentation fault (C++). Yechim: ma’lumot turini optimallashtirish, chunking (qismlarga bo‘lish), generator funksiyalar, memory-mapped files.

Type errors: Noto‘g‘ri ma’lumot turi. String va int qo‘shish. Implicit conversion ba’zan kutilmagan natija beradi. Yechim: explicit type conversion, type hints (Python), strong typing (Java, C++).

NaN va Inf: Not a Number (0/0, √-1) va Infinity (1/0). np.isnan(), np.isinf() tekshirish. np.nan_to_num() NaN va Inf ni almayshtirish. Warning‘lar: RuntimeWarning: invalid value encountered.

Performance issues: Sekin ishlash. Sabablari: Python loop’lari vektorizatsiya o‘rniga, noto‘g‘ri xotira access pattern (kesh miss’lar), keraksiz nusxa (copy) yaratish. Yechim: profiling (cProfile, line_profiler), vektorizatsiya, in-place operatsiyalar, to‘g‘ri algoritm tanlash.

Xulosa

Ko‘p o‘lchovli massivlar va ularning funksiyalarini tadqiq qilish natijasida quyidagi asosiy xulosalarga kelamiz.

Massivlar zamonaviy dasturlashning asosiy ma’lumotlar strukturasi bo‘lib, tartibli ma’lumotlarni saqlash va tezkor murojaat qilish imkoniyatini beradi. Matematik jihatdan massivlar vektorlar, matritsalar va tensorlar sifatida qaraladi va chiziqli algebra apparati qo‘llaniladi. O‘lchovlilik, shakl, hajm va ma’lumot turi massivning asosiy xarakteristikalaridir.

Xotira tashkil etilishi massiv samaradorligining asosiy omili hisoblanadi. Qator-ustun va ustun-qator saqlash tartiblari turli tillarda turlicha qo‘llaniladi. Xotirada uzluksiz joylashish tezkor murojaat va kesh samaradorligini ta’minlaydi. Stride tushunchasi xotira tuzilmasini to‘liq tavsiflaydi va shakl o‘zgartirish operatsiyalarida muhim.

Massiv funksiyalari keng spektrni qamrab oladi va turli vazifalarga mo‘ljallangan. Yaratish va initsializatsiya, kirish va o‘zgartirish, matematik operatsiyalar, qidiruv va saralash, transformatsiya va agregatsiya - har bir toifa o‘z algoritmik murakkabligi va qo‘llanish sohalariga ega.

Indekslash va slicing massivlar bilan ishlashning asosiy usullari bo‘lib, ma’lumotlarga moslashuvchan murojaat imkonini beradi. NumPy da advanced indexing (boolean, fancy) kuchli filtrlash va tanlash imkoniyatlarini taqdim etadi. View va copy mexanizmlari xotira samaradorligini ta’minlaydi.

Matematik operatsiyalar massivlarning asosiy qo‘llanish sohasini tashkil etadi. Elementma-element operatsiyalar, broadcasting, chiziqli algebra (matritsalarni ko‘paytirish, transpozitsiya, yoyilmalar) ilmiy hisoblashlar va ma’lumotlar tahlilida muhim. Vektorizatsiya Python loop’laridan ancha tezroq ishlaydi.

Qidiruv va saralash algoritmlari turli murakkablik va xususiyatlarga ega. Chiziqli qidiruv oddiy lekin sekin, ikkilik qidiruv tartiblangan massivlarda samarali. Saralash algoritmlari (merge sort, quick sort) O(n log n) murakkablikka ega. Partial sort faqat k ta element kerak bo‘lganda samaraliroq.

Agregatsiya va statistik funksiyalar ma’lumotlar tahlilida keng qo‘llaniladi. Yig‘indi, o‘rtacha, dispersiya, korrelyatsiya kabi ko‘rsatkichlar ma’lumotlarni xarakteristikalaydi. Axis parametri ko‘p o‘lchovli massivlarda operatsiyalarni moslashuvchan qo‘llash imkonini beradi.

Transformatsiya operatsiyalari massiv shaklini o‘zgartirish va manipulyatsiya qilish imkonini beradi. Reshape, transpose, concatenate, split - asosiy transformatsiyalar. View yaratish xotirani tejaydi, lekin asl massivga ta’sir qilish xavfini yaratadi.

Xotira samaradorligi va optimallash dastur tezligi uchun muhim. In-place operatsiyalar, to‘g‘ri dtype tanlash, contiguous massivlar, vektorizatsiya - asosiy optimallash strategiyalari. Profiling xatoliklar va sekinliklar manbalarini aniqlashga yordam beradi.

Turli dasturlash tillarida massivlarning o‘ziga xos xususiyatlari mavjud. Python/NumPy - ilmiy hisoblashlar uchun qulay, Java - korporativ ilovalar uchun mustahkam, C++ - maksimal nazorat va tezlik, MATLAB - matritsa operatsiyalari uchun maxsus. Har bir til o‘z ekosistemasiga va qo‘llanish sohalariga ega.

Parallel va GPU hisoblashlari katta massivlar bilan ishlashni sezilarli tezlashtiradi. NumPy ning avtomatik parallelizatsiyasi, CuPy ning GPU qo‘llab-quvvatlashi,

TensorFlow va PyTorch ning kuchli imkoniyatlari zamonaviy hisoblash texnologiyalarining asosini tashkil etadi.

Amaliy qo‘llanmalar juda keng: tasvirlarni qayta ishlash, vaqt qatorlari tahlili, ma’lumotlar tahlili, chuqur o‘rganish, simulatsiya - barchasida massivlar markaziy rol o‘ynaydi. Har bir soha o‘z maxsus kutubxonalari va usullariga ega.

Tadqiqot ko‘rsatdiki, ko‘p o‘lchovli massivlarni to‘liq tushunish nazariy bilim (matematik asoslar), algoritmik fikrlash (murakkablik tahlili) va amaliy ko‘nikmalarni (dasturlash, debugging) birlashtiradi. Bu bilimlar zamonaviy dasturchi uchun zarurdir.

Foydalanilgan adabiyotlar

1. Cormen T.H., Leiserson C.E., Rivest R.L., Stein C. Introduction to Algorithms, 3rd Edition. MIT Press, 2009. 1312 p.

2. McKinney W. Python for Data Analysis, 2nd Edition. O‘Reilly Media, 2017. 544 p.

3. VanderPlas J. Python Data Science Handbook. O‘Reilly Media, 2016. 541 p.

4. Goodfellow I., Bengio Y., Courville A. Deep Learning. MIT Press, 2016. 775 p.

5. Press W.H., Teukolsky S.A., Vetterling W.T., Flannery B.P. Numerical Recipes, 3rd Edition. Cambridge University Press, 2007. 1235 p.

6. Stroustrup B. The C++ Programming Language, 4th Edition. Addison-Wesley, 2013. 1376 p.

7. Bloch J. Effective Java, 3rd Edition. Addison-Wesley, 2018. 416 p.

8. Harris C.R., Millman K.J., van der Walt S.J. et al. Array programming with NumPy // Nature. 2020. Vol. 585. P. 357-362.

9. Strang G. Introduction to Linear Algebra, 5th Edition. Wellesley-Cambridge Press, 2016. 584 p.

10. Golub G.H., Van Loan C.F. Matrix Computations, 4th Edition. Johns Hopkins University Press, 2013. 756 p.

11. Седжвик Р. Фундаментальные алгоритмы на C++. Анализ/Структуры данных/Сортировка/Поиск. СПб: ДиаСофтЮП, 2002. 688 с.

12. Кнут Д. Искусство программирования, том 1. Основные алгоритмы. М.: Вильямс, 2006. 720 с.

13. Skiena S.S. The Algorithm Design Manual, 3rd Edition. Springer, 2020. 793 p.

14. Lutz M. Learning Python, 5th Edition. O‘Reilly Media, 2013. 1648 p.

15. Ramalho L. Fluent Python, 2nd Edition. O‘Reilly Media, 2022. 1012 p.