12-дәріс

АСИМПТОТИКАЛЫҚ ТАЛДАУ. BIG-OH НОТАЦИЯСЫ

Күрделілік

Күрделілік кейбір ресурстарды қолдану өлшемі болып табылады. Бұл ресурсқа жадыда немесе уақыт бойынша сақтау талабы қойылуы мүмкін. Ереже бойынша алгоритм құртатын уақыттың саны қиындық контекстіндегі маңызды ресурс болып табылады.

Асимптотикалық талдау – алгоритм арқылы қолданылатын ресурстардың сандық бейнеленуі. Ресурс көп жағдайда алгоритмнің орындалу уақытын білдіреді. Программаның орындалу уақыты физикалық параметр болып табылады. Негізінде программа орындалуы үшін қанша уақыт кететінін анықтау үшін секундомерді қолдану жеткілікті. Компьютерлік жүйелердің барлығында ішкі сағаттары бар болғандықтан қол сағатын қолданудың қажеті жоқ. Бірақ программаның орындалу уақыты қатаң бекітілген параметр болып табылмайды; программаның орындалу уақыты енгізілген деректердің көлемінен тікелей тәуелді болады. Мысалы, бір миллион саннан тұратын тізбек үшін сұрыптау алгоритмін орындау жүз элементтен тұратын тізбек үшін сұрыптау алгоритмін орындауға қарағанда ұзақ уақытты алады. Программаның орындалу уақыты мәліметтерді енгізу функциясы болып табылады. Программаның орындалу ұзақтығына әсер етуші және бір фактор ол қолданылып отырған компьютердің типіне де байланысты болады. Осы секілді көптеген қиындықтардан құтылу үшін алгоритмдегі қадамдар санын көрсететін ***логикалық орындалу уақыты*** өлшемін қолдану ұсынылады.

Қадамдар санын есептеу алгоритмнің орындалуының тиімді өлшемі болып табылады. Сонымен қатар алгоритм белгілі бір компьютерде орындалуы тиіс болса, онда бірнеше жағдай үшін кететін физикалық уақытты өлшеу қажет. Теориялық талдаулар арқылы деректерді кездейсоқ енгізулер үшін алгоритмдердің орындалуының дәл уақыты жеткілікті болып келеді.

Алгоритмнің орындалуының логикалық уақытын анықтау

Үлкен көлемді есептерде алгоритм қадамдарын қалай санауға болады? Ол үшін ұсақ детальдарға назар аудармай үлкен мәселелерді ғана шешу керек. Мысал ретінде ***selection sort*** алгоритмін қарастыруға болады.

Локальды айнымалылар кішкентай, тұрақты қадамдар санынан тұрады. Сыртқа циклдің денесі ***v.size ()*** – 1 жағдайы туындағанша орындалады, ***v.size ()*** вектордағы элементтер саны. Ішкі циклде орындалатын қадамдар саны сыртқы циклдің әрбір итерациясы сайын төмендейді. Талқылауға тиімді болуы үшін вектордағы элементтер санын n-ге теңестіреміз. Бірінші программаның орындалуы барысында ішкі цикл ***(n–1)*** итерация орындайды, екінші программаның орындалуы барысында ішкі цикл ***(n–2)*** итерация орындайды. Ішкі циклдің денесі "***if***" операторынан тұрады. Бұл оператор тұрақты уақыт шамасында орындалуы мүмкін.

Сонымен бізде ***selection sort*** орындалатын операциялар қосындысы белгілі. Бұл қосындыдағы әрбір n сыртқы циклде орындалатын итерациялардың қадамдарының санын береді.

$$(n - 1) + (n - 2) + (n - 3) + \ldots + 3 + 2 + 1$$

1-мысал

Бұл өрнектің мәні $(n^2 - n) / 2$-ге тең. Сонымен қатар ***selection sort*** қосындысы квадрат алгоритмін береді. Кіріс өлшемдерінің ортаквадраттық көбеюі алгоритмнің орындалу барысында төрт есе көбеюіне алып келеді.

Квадраттық алгоритмге қарама-қарсы алгоритм ол ***логарифмдік алгоритм*** болып табылады. Лагорифмдік алгоритмде қадамдар саны аса көбеймейді. Лагорифмдік алгоритм мысалы ретінде екілік іздеу алгоритмін алуға болады.

О-үлкен нотациясы

«О-үлкен» программаның орындалу барысында алгоритмдерді салыстыру тәсілдерімен жабдықтайды. «О-үлкен» негізгі қасиеті ол тұрақтылар мен төменгі ретті

$$500n^3 + 10n^2 + 17n + 121345 = O(n^3)$$

2-мысал

дәрежелерді ескермейді

Жоғарыда келтірілген мысалда теңдеудің ең жоғарғы дәрежесі $500n^3$ -не тең. Тұрақтылар мен төменгі дәрежелерді ескермесек, онда программаның орындалу уақыты $O(n^3)$-ке тең.

Программаның орындалуы барысындағы алгоритмдер төмендегі кластардан алынады:

– O(1) – constant time

– O(log(n)) – logarithmic time

– O(n) – linear time

– O(n log(n)) – "n-log-n" time

– $O(n^2)$ – quadratic time

– $O(n^3)$ – cubic time

– $O(2^n)$ – exponential time

Орындалу уақыты ***O(n log(n))*** болатын алгоритмдер үлкен көлемді есептерді шығару барысында тиімді болып келеді. Өлшемі 1000-ға тең ақпаратты енгізу барысындағы 1 секунд уақытты алатын ***O(n log(n))*** алгоритмін қарастырайық.

input size	running time
1000	1 second
2000	2,2 seconds
5000	6,2 seconds
10000	13,3 seconds
100000	2,77 minutes
10^6	33,3 minutes
10^7	6,48 hours

Салыстыру үшін өлшемі 1000-ға тең ақпаратты енгізу барысындағы 1 секунд уақытты алатын ***квадраттық алгоритмін*** қарастырайық.

$O(n^2)$ алгоритм

input size	running time
1000	1 second
2000	4 seconds
5000	25 seconds
10000	1,66 minutes
100000	2,77 hours
10^6	11,5 days
10^7	3,25 years

Үлкен көлемді деректерді енгізу барысында квадраттық алгоритм тиімсіз болып табылады. Квадраттық алгоритмдер экспоненциалды деректерді енгізу барысымен салыстырғанда жақсырақ алгоритм болып табылады. Салыстыру үшін өлшемі 100-ге тең ақпаратты енгізу барысындағы 1 секунд уақытты алатын ***экспоненциалды алгоритміне*** мысал қарастырайық.

O (2^n) алгоритм

input size	running time
10	1 seconds
15	32 seconds
20	17,1 minutes
25	9,1 hours
30	12,1 days
35	1,09 years
40	34,9 years
50	35700 years
100	4,02 10^{19} years

Алгоритм көріп тұрғанымыздай, 30 өлшемін енгізу кезінде ақ тиімсіз болып табылды. Өкінішке орай, кейбір есептер экспоненциалды алгоритммен шығаруды талап етеді.

Негізгі ережелер

Біз тұрақтыларды ескермегендіктен негізгі тұжырымдардағы кез келген тізбек 1) selection sort меншіктеу немесе 2) selection sort орындалатын операциялар ***(O(1))*** уақытты талап етеді.

Сонымен қатар функцияларды шақыру жеке ескерілуі тиіс. Біріншіден, ***(O(1))*** функциясын шақыруды ұйымдастыру шығыны. Егер шақыру сан мәніне сәйкес жүргізілетін болса, онда енгізу көшірмесін де ескеру қажет.

Ескерілетін уақыт циклі t(0) + t(1) +... + t(n-1) тең. Мұндағы t(k) – i=k болғандағы цикл денесіне қажетті уақыт

```
for (int  i = 0; i < n; ++i)  {
body
}
```

3-мысал

Егер программа денесі O(1) уақытты алса, ол O(n) сәйкес келеді.

Егер программа денесі O(i) уақытты алса, ол $O(n^2)$ сәйкес келеді.

Егер программа денесі O (i^2 уақытты алса, ол $O(n^3)$ сәйкес келеді.

Циклдің өзге құраушылары осыған ұқсас құрылады.

Хеш функциялар

Хэштеу – бұл анықталған типтегі және кез келген ұзындықтағы мәліметтердің кіріс массивін белгіленген ұзындықтағы шығыс биттік қатарға түрлендіру. Сонымен қатар мұндай түрлендірулерді ***хэш функциялар*** деп, ал олардың нәтижелерін ***хэштер, хэш код, хэш кестелер*** деп атаймыз.

Хэштеу мәліметтерді салыстыру үшін қолданылады: егер екі массивтің хэш коды әртүрлі болса, онда массивтер әртүрлі болады, ал егер бірдей массивтер болса, олар бірдей болады.

Әртүрлі сипаттамадағы көптеген алгоритмдер бар. Хэш функцияларды таңдау орындалатын есептің спецификасына байланысты анықталады.

Хэштеу идеясын бірінші рет 1953 жылы қаңтарда ішкі IBM меморандумын құру барысында Г.П. Ханс Петер Лун тізбектер әдісімен шешу үшін айтқан болатын. Осы уақытта IBM-нің тағы бір қызметкері ашық сызықты адрестеуді қолдану идеясын айтқан болатын. Ең алғаш рет ашық баспадағы хэштеуді хэш адресі ретінде қарапайым санға бөлу қалдығын қолдану ыңғайлы екенін көрсете отырып, Арнольд Думи (1956 ж.) сипаттап берген болатын.

Хэш кесте – бұл ассоциативті массивтің интерфейсін жүзеге асыратын мәліметтер құрылымы, яғни ол «кілт-мағына» түріндегі жұпты сақтауға мүмкіндік береді және келесі үш операцияны орындайды:

– жаңа жұпты қосу операциясы;

– іздеу операциясы;

– кілті бойынша жұптарды өшіру операциясы.

Хэш кестелер хэш функциялардың белгілі бір тәртібімен қалыптасатын массив болып табылады. Келесі шарттарды қанағаттандыратын функцияларды ***жақсы хэш функциялар*** деп атаймыз:

– есептеуіш көзқараспен қарағанда функция қарапайым болуы керек;

– функция кілттерді хэш кестелерге бірқалыпты үлестіруі керек;

– функция кілттердің мәні мен адрестердің мәнінің арасында ешқандай байланысты бейнелемеуі қажет;

– әртүрлі кілтке хэш функцияның бір мәні сәйкес келген жағдайда функция коллозия санын минимумдауы қажет (кілттер мұндай жағдайда синоним деп аталады).

Егер барлық мәліметтер кездейсоқ шамалар болса онда хэш функциялар қарапайым болар еді. Бірақ тәжірибе кездейсоқ мәліметтер сирек кездеседі және бүкіл кілттен тұратын функцияны құруға тура келеді.

Егер хэш функция мүмкін болатын кілттер жиынтығын индекстер жиынтығы бойынша бірқалыпты үлестірсе, онда хэштеу кілттер жинтығын тиімді бөледі.

Data	Letter	Ascii	Index
Hanson, Bob	a	97	7
Adderly, Maria	d	100	0
Cruz, Jose	r	114	4
Song, Albert	o	111	1
Appleton, Mary	p	112	2

Жоғарыдағы суретте кілттердің индекстердегі бейнесі келтірілген. Бұл жерде хеш функция ASCII кодының сан мәнінің оң жағындағы цифрға, адамның атының екінші әрпіне негізделген индексті генерациялайды.

Мысалы, "Hanson, Bob" атында екінші әріп «a» болып табылады. ASCII кодтағы «a» мәні 97-ге тең. 97 оң жағындағы сан 7. Сондықтан "Hanson, Bob" хеш кестеде 7 индексімен сақталған.