Алгоритмын шинжилгээ ба зохиомж

F.CS301

Д. Батмөнх

2024 оны 11-р сарын 14

Хичээлийн тухай I

Сурах бичиг болон бусад

- Introduction to Algorithms (Fourth Edition) by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein
- Online platforms for coding: HackerRank, LeetCode

Хичээлийн тухай II

Үнэлгээ

Ирцийн оноо	15	Лабораторийн ирц 1 оноо
Бие даалтын оноо	10	Нийт 2 бие даалт тусбүр 5 оноо
Явцын сорилын оноо	15	2 удаа
Лабораторийн оноо	30	Цагтаа бодсон бодлого бүр 1 оноо
Шалгалтын оноо	30	

Хэрэв дараа 7 хоногт үзүүлбэл даалгавар бүрийн дээд оноо 0.5. 1 бодлогыг 2 хэл дээр бодвол 1 оноо, 1 хэл дээр бодвол 0.5 оноо.

Хичээлийн тухай III

Лабораторийн хичээлд ашиглах хэл

- C++
- Java
- Python

Оюутан дээрх 3 хэлнээс 2 хэлийг сонгож лабораторийн даалгаврыг хийнэ.

Хичээлийн тухай IV

Unit test

Туршилтын өгөгдөл бэлтгэн оруулахдаа дараах unit testing framework ашиглана:

- C++ бол Catch2
- Java бол JUnit
- Python бол unittest

Алгоритм I

Алгоритм гэж оролтод өгөгдсөн утгыг тогтоосон хугацаанд боловсруулан гаргах нарийвчлан боловсруулсан үйлдэл.

Эрэмбэлэх бодлогын ерөнхий тохиолдол

Оролт: n тооны дараалал $\langle a_1, a_2, \ldots, a_n \rangle$
Гаралт: Оролтын өгөгдлийн $a'_1 \leq a'_2 \leq \cdots \leq a'_n$ байх сэлгэлт $\langle a'_1, a'_2, \ldots, a'_n \rangle$

Тухайн тохиолдол

Оролт: $\langle 31; 41; 59; 26; 41; 58 \rangle$
Гаралт: $\langle 26; 31; 41; 41; 58; 59 \rangle$

Алгоритм II

Хэрэв бодлого, тухайн тохиолдлын оролт бүрд төгсгөлөг хугацаанд тооцоолол хийж дуусан зөв хариу буцааж байвал бодлогын тооцооллын алгоритмыг зөв байна гэж үзнэ. Өөрөөр хэлбэл, буруу алгоритм нь зарим тухайн тохиолдлын оролтын тооцоолол дуусгахгүй буюу буруу хариу буцаадаг.

Ямар төрлийн бодлогууд алгоритмаар бодогдох вэ?

- Биологийн бодлогууд: хүний ДНХ бүрдүүлэгч 30 мянган генийг судалж 3 тэрбум дарааллыг гарган авах бодлогыг Dynamic programming ашиглах бодох
- Өдөр бүр интернетээр хийгдэж буй их хэмжээний хайлт болон хандалтуудыг түргэн шуурхай зохицуулахад Hash tables, Single-Source Shortest Paths, String Matching
- Цахим худалдаа зэрэгт хувийн нууцлалыг хангахад криптографи (Number-Theoretic Algorithms) ашиглах
- Үйлдвэр худалдааны газруудад барааг үр ашигтайгаар түгээхэд Linear Programming ашиглах
- Эмчийн, өвчний оношийг тогтооход бүлэглэх алгоритмыг (Machine-Learning Algorithms) ашиглах
- Huffman coding тусламжтайгаар их хэмжээний өгөгдлийг шахах (LZW compression)

Өгөгдлийн бүтэц

Өгөгдлийн бүтэц (data structure) нь өгөгдлийг санах ойд байршуулан шуурхай хандан өөрчлөлт хийх боломжийг олгодог.

Олон төрлийн бүтэцтэй байх бөгөөд тэдгээр нь өөрийн гэсэн сул болон давуу талтай байдаг. Тиймээс тухайн тохиолдолд тохирсон өгөгдлийн бүтцийг ашиглах нь алгоритмын зохиомжийн чухал хэсэг болдог.

Hard problems

- Бид цаашдаа алгоритмын үр ашгийг түлхүү судлах бөгөөд үүнийг тооцох хэмжигдэхүүн нь хурд юм. Өөрөөр хэлбэл, үр дүнг гаргахын тулд алгоритм хэр удаан ажиллах хэрэгтэй вэ? гэдгийг тооцоолно гэсэн үг.
- Гэвч зарим бодлогын хувьд хугацаа тогтоон бодох алгоритм байдаггүй. Үүнийг NP-complete бодлого гэдэг.
- Жишээлбэл, түгээлтийн компанийн ачааны машинууд өдөр бүр төв агуулахаасаа ачаагаа ачин түгээх цэгүүд дээрээ буулгаад орой төв агуулах дээрээ буцаж ирнэ. Компани энэхүү үйл ажиллагааны зардлыг бууруулахаар түгээлтийн оновчтой цэгүүдийг тодорхойлох хэрэгтэй болсон гэвэл, хэдийгээр богино замыг тодорхойлж болох ч уг асуудлыг (traveling-salesperson problem) шийдэх оновчтой алгоритм байхгүй гэсэн үг.

Технологийн хувьд

Хэдийгээр компьютерын хүчин чадлын өсөлт нэмэгдсээр байгаа боловч түүний хурд хязгаарлагдмал бөгөөд тооцоолол хийх хугацаа мөн хязгаарлагдмал тул дараах 2 тохиолдлын аль нэгийг сонгох хэрэгтэй болно:

- тооцоолол хийх хугацаа бага байх
- санах ойн ашиглалт бага байх

Бүтээмж (efficiency) I

Нэг бодлогыг олон төрлийн алгоритмаар бодоход зарцуулах нөөц боломж харилцан адилгүй байх бөгөөд их хэмжээний өгөгдлийн хувьд энэхүү ялгаа улам бүр ихэсдэг.
Жишээлбэл,

- insertion sort: $c_1 n^2$
- merge sort: $c_2 n \lg n$

n хувьсагчийн багахан утгад:

- insertion sort: 1,000
- merge sort: 10

Өсгөвөл

- insertion sort: 1,000,000
- merge sort: 20

Бүтээмж (efficiency) II

Өөр нэгэн жишээ авъя. 10 сая өгөгдлийг дараах байдлаар бэлтгэж эрэмбэлнэ:

- insertion sort: А компьютер (секундэд 10 тэрбум үйлдэл гүйцэтгэнэ) дээр туршлагатай программ зохиогч доод төвшний хэл дээр insertion sort ашиглан эрэмбэлэлт хийнэ
- merge sort: Б компьютер (секундэд 10 сая үйлдэл гүйцэтгэнэ (1000 дахин удаан)) дээр дундаж төвшний программ зохиогч өндөр төвшний хэл дээр merge sort ашиглан эрэмбэлэлт хийнэ.

Бүтээмж (efficiency) III

Нийт зарцуулах хугацаа:

- insertion sort: 20,000 секунд (5.5 цаг)
- merge sort: 1,163 секунд (20 минут)

Алгоритмын бүтээмжийн ялгаа, их хэмжээний өгөгдөл дээр ийнхүү анзаарагдана.

Insertion sort I

Өмнө эрэмбэлэлтийн талаар (insertion sort) дурдсан билээ.
Оролт: n тооны дараалал $\langle a_1, a_2, \ldots, a_n \rangle$
Гаралт: Оролтын өгөгдлийн $a'_1 \leq a'_2 \leq \cdots \leq a'_n$ байх сэлгэлт $\langle a'_1, a'_2, \ldots, a'_n \rangle$

Insertion sort II

Эрэмбэлэгдэх тоонуудыг түлхүүр (keys) гэдэг бол бусад хэсгийг дагуул өгөгдөл (satellite data) гэдэг. Жишээлбэл, нас, дундаж оноо, үзсэн хичээл зэрэг мэдээллийг агуулсан оюутны хүснэгтэн мэдээллийг эрэмбэлэх үед гол эрэмбэлэх баганыг **түлхүүр** гээд үлдэх баганууудыг **дагуул өгөгдөл** гэнэ.

Insertion sort III

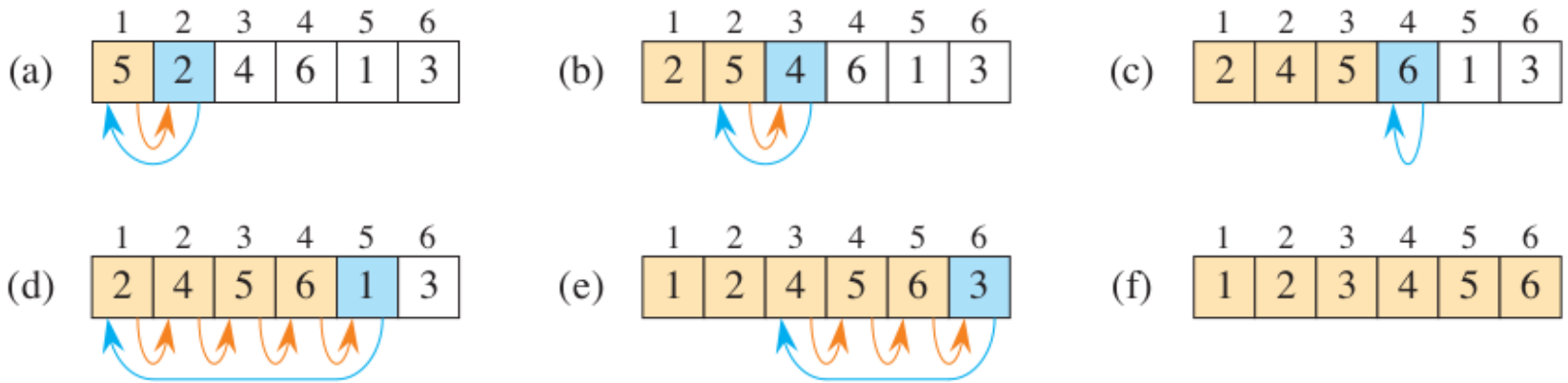
Insertion sort-ийн хийсвэр код (pseudocode):

Insertion-Sort(A, n)

```
1  for i = 2 to n
2        key = A[i]
3        j = i − 1
4        while j > 0 and A[j] > key
5              A[j + 1] = A[j]
6              j = j − 1
7        A[j + 1] = key
```

Insertion sort IV

Дээрх хийсвэр код нь 2 хэмжигдэхүүнтэй (parameter) бөгөөд нэг нь A гэсэн жагсаалт, нөгөө нь эрэмбэлбэл зохих n ширхэг хувьсагч.



Insertion sort V

Дээрх давталтаас дараах 3 зүйлийг онцолж болно:

- Эхлэл: Давталтын эхний алхмын өмнөтгөл үнэн ($i = 2,\ A[1]$)
- Өрнөл: Давталтын эхлэл хэсэг үнэн бол дараагийн давталтын урьтал мөн үнэн байна
- Төгсгөл: Давталт дуусаж, функцийн ажиллагаа зогсож байгаа нь алгоритм зөв байна гэдгийг илтгэнэ

Insertion sort VI

Хийсвэр кодын дүрэм

- Догол мөрөөр бүлэг кодыг тэмдэглэнэ. Жишээлбэл, for (2–7) болон while (5–6) давталтыг догол мөрөөр тус тус ялган бүлэглэсэн байна. Хэрэв хуудас салж бичигдэх бол begin болон end түлхүүр үг ашиглана.
- Давталтыг тэмдэглэхдээ while, for болон repeat-until, нөхцөл тэмдэглэхдээ if-else, elseif ашиглана. Дээрх жишээнд өсөх давталтыг to, for ашиглан тэмдэглэсэн бол буурах тохиолдолд downto, for ашигладаг. Хэрэв өсөлт нь 1-ээс их бол by түлхүүр үг ашиглана.
- Тайлбарыг // түлхүүр ашиглаж бичнэ.
- Дээрх жишээнд ашигласан хувьсагчууд нь (i, j, key) нь дотоод (local) хувьсагч юм. Гадаад хувьсагч бол global хэмээн зааж өгнө.

Insertion sort VII

- Жагсаалтын элементүүдийн товъёгийг дөрвөлжин хаалтад тэмдэглэх бөгөөд энэ нь 1 гэсэн утгаас эхэлнэ. Мөн дэд жагсаалтыг (subarray) : тэмдэг ашиглан бичнэ. Жишээлбэл, $A[i : j]$ гэвэл $A[i], A[i + 1], \ldots, A[j]$ болно.
- Объектын атрибутуудыг дуудахдаа цэг ашиглана. Жишээ нь, x объектын f атрибутыг дуудахдаа $x.f$
- Хэрэв функцэд утга дамжуулах үед дамжуулсан утгын хожмын өөрчлөлт функцэд нөлөөлөхгүй, харин объект дамжуулсан бол дамжуулсан объектын заагчийг хуулбарлах боловч объектын атрибутуудыг хуулбарлахгүй. Жишээлбэл, x нь функцийн хэмжигдэхүүн бөгөөд $x = y$ оноолт нь функцэд нөлөөлөхгүй. Харин $x.f = 3$ утга оноолт нь нөлөөлнө, учир нь x объектын заагчтай ижил тул.
- return нь олон өгөгдлийг, шинээр объект үүсгэж багцлахгүйгээр шууд буцаадаг.

Insertion sort VIII

- and болон or нь богино холбоо (short circuiting) бөгөөд *x and y* гэвэл эхэлж *x* үнэн байж удаах үнэлэгдэнэ.
- error түлхүүр үгээр функцэд алдаа гарах тохиолдлыг бүртгэхэд ашиглана.

Алгоритмын шинжилгээ I

Алгоритмын шинжилгээг, ердийн нэг процессор бүхий зэрэгцээ бус ердийн горимоор ажиллах дурын машин (random-access machine (RAM)) дээр хийсвэрлэн тооцно. RAM загварын хувьд өгөгдлийн төрөл нь integer, floating point болон character байна. Мөн санах ойн шатлал (memory-hierarchy) зэрэг бодит компьютерын хувьд яригдах ойлголтуудаас ангид.

Өмнө үзсэн эрэмбэлэх алгоритмын хувьд оролтын өгөгдлийн хэмжээнээс хамаарч ажиллах хугацааг тооцоолъё. Үүний тулд эхлээд ажиллах хугацаа (running time) болон оролтын хэмжээ (input size) гэсэн ойлголтуудыг авч үзье.

Алгоритмын шинжилгээ II

Оролтын хэмжээ нь тухайн бодлогоос шууд хамаарна. Жишээ нь, графын хувьд энэ нь оройн тоо болон ирмэгийн тоо байх бол эрэмбэлэх бодлогын хувьд энэ нь эрэмбэлэх нэгж өгөгдлийн тоо байна.

Ажиллах хугацаа нь нийт гүйцэтгэх команд болон өгөгдлийн хандалтын тоо байна. Нийт хугацаа нь бидний хийсвэр кодын мөр бүрийг биелүүлэхэд шаардагдах хугацаа байна. Нэг мөр бүр бусдаасаа ахиу буюу богино хугацаанд биелнэ. Хэрэв k гэсэн мөр нь c_k хугацаанд биелэх бөгөөд c_k нь тогтол хэмжигдэхүүнд тооцогдоно.

Алгоритмын шинжилгээ III

Insertion sort

n тооны оролтын хэмжээ бүхий ажиллах хугацааг[1] $T(n)$ хэмээн тэмдэглэнэ гэвэл:

Insertion-Sort(A, n)

1	**for** $i = 2$ **to** n	c_1	n
2	$key = A[i]$	c_2	$n-1$
3	$j = i - 1$	c_3	$n-1$
4	**while** $j > 0$ and $A[j] > key$	c_4	$\sum_{i=2}^{n} t_i$
5	$A[j+1] = A[j]$	c_5	$\sum_{i=2}^{n} (t_i - 1)$
6	$j = j - 1$	c_6	$\sum_{i=2}^{n} (t_i - 1)$
7	$A[j+1] = key$	c_7	$n-1$

Алгоритмын шинжилгээ IV

Дээрх алгоритмын хувьд хугацаа нь:

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{i=2}^{n} t_i + \\ + c_5 \sum_{i=2}^{n} (t_i - 1) + c_6 \sum_{i=2}^{n} (t_i - 1) + c_7(n-1)$$

байна.

Алгоритмын шинжилгээ V

Хамгийн бага хугацаа[2] (best-case running time):

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_7(n-1) = \\ = (c_1 + c_2 + c_3 + c_4 + c_7)n - (c_2 + c_3 + c_4 + c_7)$$

буюу
$an + b$ гэсэн шугаман функц гарна.

Алгоритмын шинжилгээ VI

Хамгийн их хугацаа[3] (worst case) буюу урвуу эрэмбэлэгдсэн:

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4\left(\frac{n(n+1)}{2} - 1\right) +$$
$$+ c_5\left(\frac{n(n+1)}{2} - 1\right) + c_6\left(\frac{n(n+1)}{2} - 1\right) + c_7(n-1) =$$
$$= \left(\frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2}\right) n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4}{2} - \frac{c_5}{2} - \frac{c_6}{2} + c_7\right) n -$$
$$- (c_2 + c_3 + c_4 + c_7)$$

буюу

$an^2 + bn + c$ гэсэн квадрат функц байна.

[1]Энд 1-р мөр n удаа (нөхцөл шалгалт нэмэгдээд), 2-р мөр $n-1$ удаа, 4-р мөр нөхцөл шалгах учир 5, 6-р мөрөөс 1 үйлдэл илүү хийгдэх ба 5, 6 мөрүүд багадаа 1, ихдээ $n-1$ удаа дуудагдана.

[2]Энд 4–6 мөрийн хувьд $t_i = 1$ буюу зөвхөн нөхцөл шалгана (хэдийн эрэмбэлэгдсэн тул 5, 6-р мөрүүдэд очихгүй) гэж үзэв.

[3]$\sum_{i=2}^{n} i = \left(\sum_{i=1}^{n} i\right) - 1 = \frac{n(n+1)}{2} - 1, \sum_{i=2}^{n}(i-1) = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$

Хамгийн их болон дундаж хугацааны шинжилгээ I

Алгоритмын шинжилгээний хувьд ихэвчлэн хамгийн их хугацааг авч үзэх хэрэгтэй болдог. Учир нь

- Хамгийн их хугацааг тогтоосноор алгоритмын ажиллах хязгаар баталгаажна.
- Зарим программын хувьд гүйцэтгэлийн хурд ихэвчлэн удаан байдаг. Жишээлбэл, өгөгдлийн сангаас мэдээлэл хайх гэх мэт.
- Дундаж хугацаа нь мөн хамгийн их хугацааны адилаар түгээмэл ашиглагддаг. Жишээ нь, insertion sort дээр $A[i]$ элементийг $A[1 : i-1]$ дэд жагсаалтад нэмэх тохиолдолд дунджаар $A[i]$ нь дэд жагсаалтын дундаж утга байлаа гэвэл давталтаар гүйх хугацаа нь $i/2$ байх бөгөөд энэ нь квадрат функц байна гэсэн үг.

Өсөх хурд[4] I

Өмнө жишээнд дурдсан $an^2 + bn + c$ хугацааг эргэн санавал, энд тогтмол утгууд a, b, c нь n хувьсагчийг бодвол ач холбогдол багатай юм. Учир нь бодит амьдрал дээр тогтмол утга нь хувьсагчийн утгыг бодвол хэд дахин бага байх нь бий.

Өсөх хурдыг грек тета үсгээр (Θ) тэмдэглэх бөгөөд өмнөх алгоритмын хувьд ажиллах хамгийн их хугацаа нь $\Theta(n^2)$ бол хамгийн бага хугацаа нь $\Theta(n)$ байна.

[4] rate of growth буюу order of growth

Алгоритмын зохиомж (design) I

Тухайн бодлогыг олон янзын арга аргачлалаар бодож болдог, жишээлбэл бидний авч үзсэн insertion sort жишээний хувьд өсгөх аргыг (incremental method) ашигласан.

Харин одоо өмнөх аргаа, хэсэгчлэх (divide-and-conquer method) аргаар өөрчилж үзье. Өөрөөр хэлбэл, бодлогыг хэсэгчлэн хувааж боловсруулаад эргүүлэн нэгтгэх маягаар бодно гэсэн үг. Ингэхийн ач холбогдол нь ажиллах хугацааг улам багасгах явдал юм.

Алгоритмын зохиомж (design) II

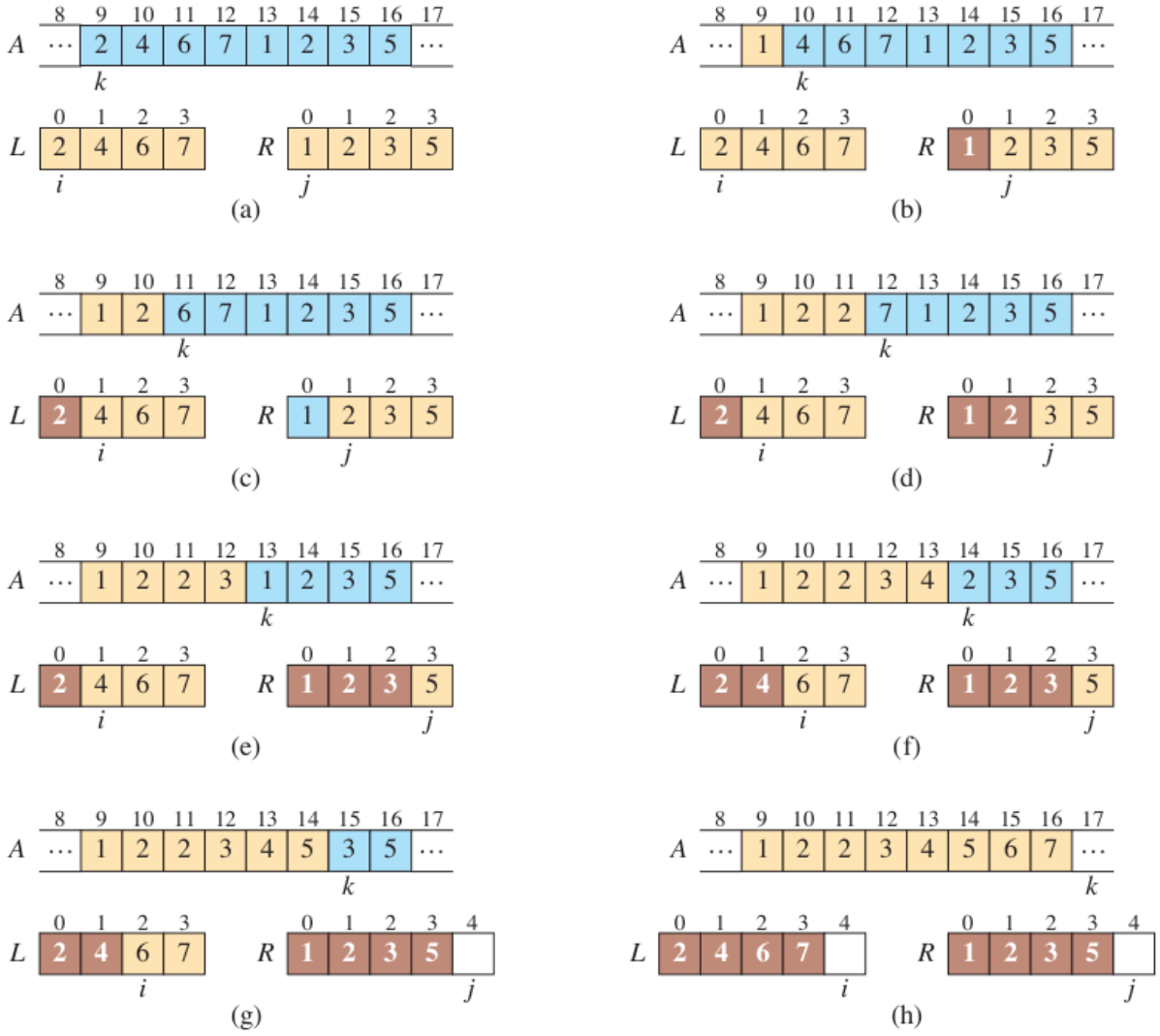
Хэсэгчлэх (divide): бодлогыг ижил төрлийн дэд бодлого болгон задална
Боловсруулах (conquer): дэд бодлогуудыг дахин дуудах (recursively) байдлаар дуудна
Нэгтгэх (combine): ийнхүү хэсэгчлэн задалж боловсруулсан үр дүнгээ нэгтгэнэ

Энэхүү аргыг merge sort алгоритм ашигласан байдаг.

Алгоритмын зохиомж (design) III

```
MERGE(A, p, q, r)
 1  n_L = q − p + 1          // length of A[p : q]
 2  n_R = r − q              // length of A[q + 1 : r]
 3  let L[0 : n_L − 1] and R[0 : n_R − 1] be new arrays
 4  for i = 0 to n_L − 1     // copy A[p : q] into L[0 : n_L − 1]
 5      L[i] = A[p + i]
 6  for j = 0 to n_R − 1     // copy A[q + 1 : r] into R[0 : n_R − 1]
 7      R[j] = A[q + j + 1]
 8  i = 0                    // i indexes the smallest remaining element in L
 9  j = 0                    // j indexes the smallest remaining element in R
10  k = p                    // k indexes the location in A to fill
11  // As long as each of the arrays L and R contains an unmerged element,
    //      copy the smallest unmerged element back into A[p : r].
12  while i < n_L and j < n_R
13      if L[i] ≤ R[j]
14          A[k] = L[i]
15          i = i + 1
16      else A[k] = R[j]
17          j = j + 1
18      k = k + 1
19  // Having gone through one of L and R entirely, copy the
    //      remainder of the other to the end of A[p : r].
20  while i < n_L
21      A[k] = L[i]
22      i = i + 1
23      k = k + 1
24  while j < n_R
25      A[k] = R[j]
26      j = j + 1
27      k = k + 1
```

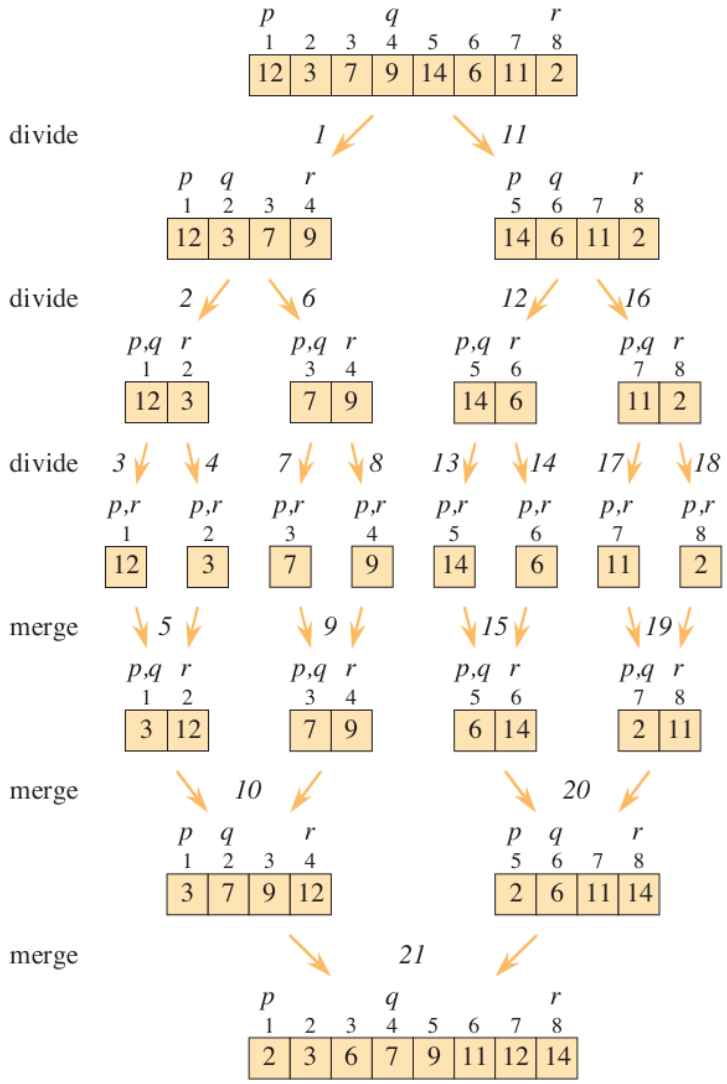
Алгоритмын зохиомж (design) IV



Алгоритмын зохиомж (design) V

```
MERGE-SORT(A, p, r)
1  if p ≥ r                                  // zero or one element?
2      return
3  q = ⌊(p + r)/2⌋                           // midpoint of A[p : r]
4  MERGE-SORT(A, p, q)                       // recursively sort A[p : q]
5  MERGE-SORT(A, q + 1, r)                   // recursively sort A[q + 1 : r]
6  // Merge A[p : q] and A[q + 1 : r] into A[p : r].
7  MERGE(A, p, q, r)
```

Алгоритмын зохиомж (design) VI



Алгоритмын зохиомж (design) VII

Анализ

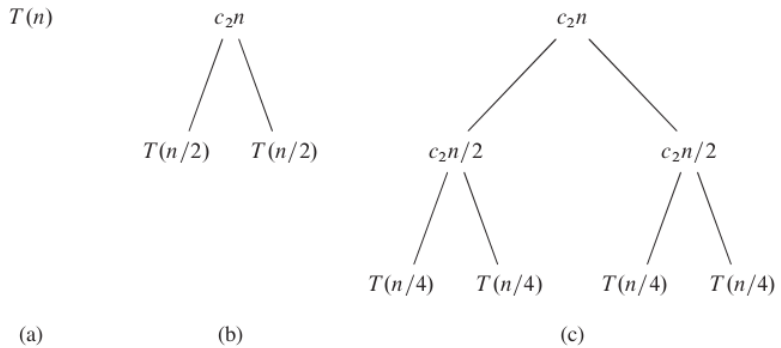
Хэсэгчлэх (divide): Энэ үйлдэл зөвхөн хуваалт хийх тул $D(n) = \Theta(1)$

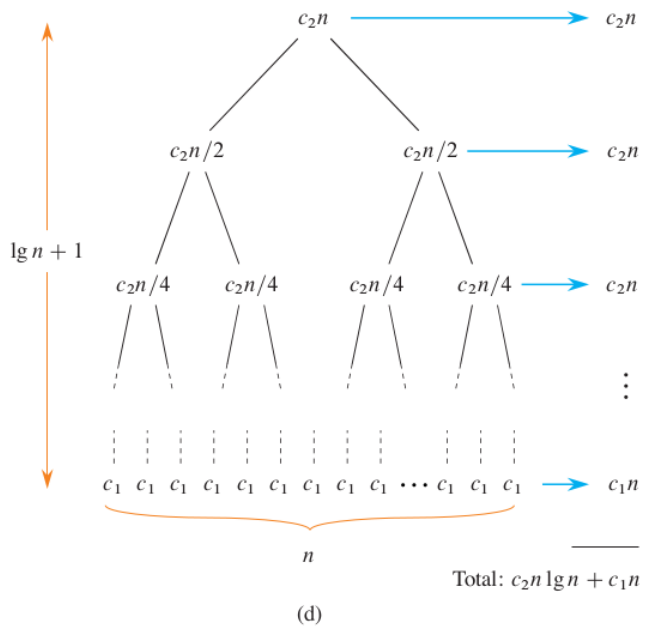
Боловсруулах (conquer): 2 хэсэгт хуваах тул $2T(n/2)$ хугацаа зарцуулна

Нэгтгэх (combine): n элементийг нэгтгэх тул $C(n) = \Theta(n)$ болно.

$T(n) = 2T(n/2) + \Theta(n) = \Theta(n \lg n)$. Энд $\lg n$ гэдэг нь $\log_2 n$. Иймхүү энэ нь insertion-sort-ийн ажиллах хугацаанаас ($\Theta(n^2)$) бага боллоо.

Алгоритмын зохиомж (design) VIII





O, *Omega*, *Theta* тэмдэглэгээ I

Энэ сэдвийн хүрээнд оролтын өгөгдлийн хэмжээ ихсэхэд ажиллах хугацааны өсөлтийн дөхөлтийн (asymptotic) цэгийн хязгаарыг судлах болно.
Өмнөх жишээг эргэн санавал

$$\left(\frac{c_4}{2}+\frac{c_5}{2}+\frac{c_6}{2}\right)n^2+\left(c_1+c_2+c_3+\frac{c_4}{2}-\frac{c_5}{2}-\frac{c_6}{2}+c_7\right)n- \\ -(c_2+c_3+c_4+c_7)$$

бид алгоритмын ажиллах хугацааг тогтоохдоо хамгийн их зэрэгтэй хувьсагчийг бага зэрэгтэй хувьсагч болон тогтмолуудаас онцгойлон авч үзсэн.

O тэмдэглэгээ

O тэмдэглэгээгээр дээд хязгаарыг тэмдэглэнэ. Өөрөөр хэлбэл, $7n^3+100n^2-20n+6$ функцийн хувьд $7n^3$ нь өндөр зэрэг тул ажиллах хугацаа нь $O(n^3)$ болно. Энэ функцийн ажиллагаа нь n^3 хугацаанаас багасахгүй гэсэн үг.

O, *Omega*, *Theta* тэмдэглэгээ II

Ω тэмдэглэгээ

Ω тэмдэглэгээгээр доод хязгаарыг тэмдэглэнэ. Insertion sort алгоритмын хувьд ажиллах хамгийн их хугацааны (worst-case running time) доод хязгаар нь $\Omega(n^2)$ байна.

O, *Omega*, *Theta* тэмдэглэгээ III

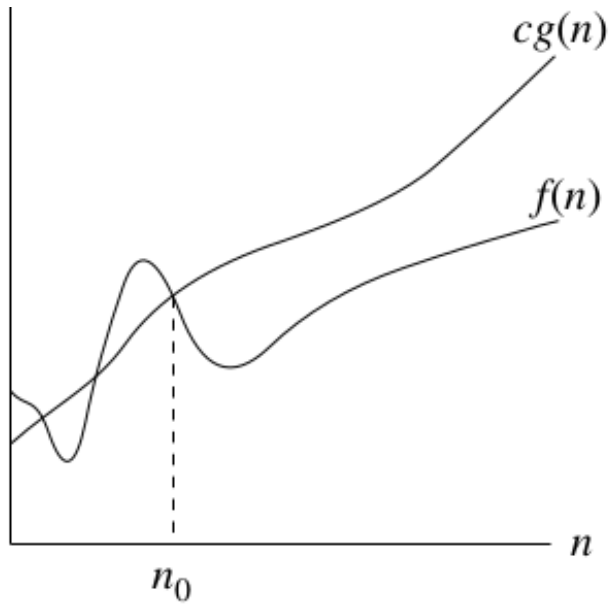
Θ тэмдэглэгээ

Θ тэмдэглэгээгээр дундаж (tight) хязгаарыг тэмдэглэнэ. Хэрэв функц $O(f(n))$ ба $\Omega(f(n))$ бол $\Theta(f(n))$ байна.

Дөхөлтийн тэмдэглэгээ I

O тэмдэглэгээ

$O(g(n)) = \{f(n) : n \geq n_0$ бүрийн хувьд $0 \leq f(n) \leq cg(n)$ байх c ба n_0 эерэг тогтмол тоо олдоно$\}$

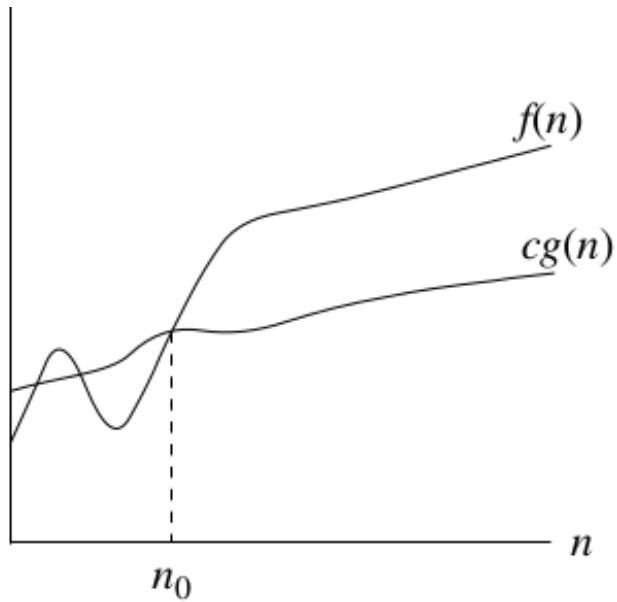


$g(n)$ нь $f(n)$ функцийн хувьд дөхөлтийн дээд хязгаар юм.
Жишээлбэл, $2n^2 = O(n^3)$ үүнд, $c = 1$ ба $n_0 = 2$.

Дөхөлтийн тэмдэглэгээ II

Ω тэмдэглэгээ

$O(g(n)) = \{f(n) : n \geq n_0$ бүрийн хувьд $0 \leq cg(n) \leq f(n)$ байх c ба n_0 эерэг тогтмол тоо олдоно$\}$

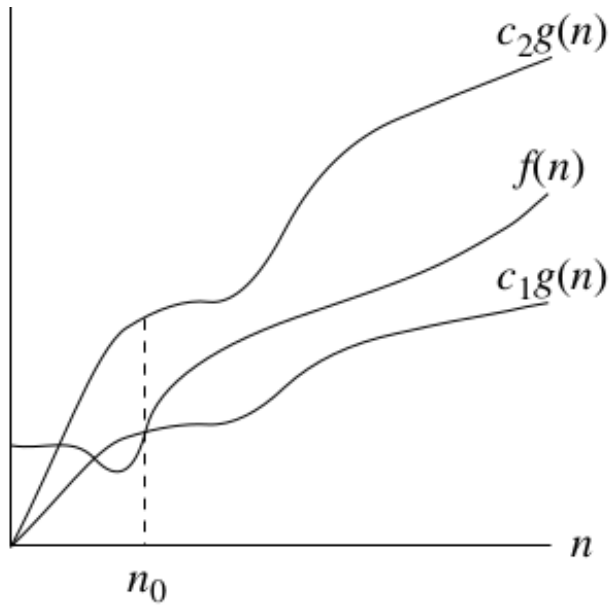


$g(n)$ нь $f(n)$ функцийн хувьд дөхөлтийн доод хязгаар юм.
Жишээлбэл, $\sqrt{n} = \Omega(\lg n)$ үүнд, $c = 1$ ба $n_0 = 16$.

Дөхөлтийн тэмдэглэгээ III

Θ тэмдэглэгээ

$\Theta(g(n)) = \{f(n) : n \geq n_0$ бүрийн хувьд $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ байх c_1, c_2 ба n_0 эерэг тогтмол тоо олдоно$\}$



$g(n)$ нь $f(n)$ функцийн хувьд дөхөлтийн дундаж хязгаар болно. Жишээлбэл, $n^2/2 - 2n = \Theta(n^2)$ ба $c_1 = 1/4,\ c_2 = 1/2$ ба $n_0 = 8$.

Дөхөлтийн тэмдэглэгээ IV

Теорем

Хэрэв $f = O(g(n))$ ба $f = \Omega(g(n))$ бол $f(n) = \Theta(g(n))$ байна.

Дөхөлтийн тэмдэглэгээ V

Томьёоны баруун талд

$O(n^2)$ нь $O(n^2)$ олонлогийн дурын функцийг илэрхийлнэ.
$2n^2 + 3n + 1 = 2n^2 + \Theta(n)$ гэдэг нь $f(n) \in \Theta(n)$ функцийн хувьд $2n^2 + 3n + 1 = 2n^2 + f(n)$. Үүнд $f(n) = 3n + 1$.
$\sum_{i=1}^{n} O(i)$ илэрхийллийн хувьд i гэсэн дурын функц байна гэж ойлгох бөгөөд энэ нь $O(1) + O(2) + \cdots + O(n)$ гэсэн үг биш.

Дөхөлтийн тэмдэглэгээ VI

Томьёоны зүүн талд

Илэрхийллийн зүүн талд үл мэдэгдэх функцийг хэрхэн сонгосноос үл хамааран баруун талд үл мэдэгдэх функцийг тэнцүүлэн сонгоно: $2n^2 + \Theta(n) = \Theta(n^2)$ гэсэн илэрхийлэл нь $f(n) \in \Theta(n)$ бүх функцийн хувьд $2n^2 + f(n) = g(n)$ байх $g(n) \in \Theta(n^2)$ функц оршин байна.

$$2n^2 + 3n + 1 = 2n^2 + \Theta(n) = \Theta(n^2).$$

Дөхөлтийн тэмдэглэгээ VII

o тэмдэглэгээ

$o(g(n)) = \{f(n) :$ бүх $c > 0$ байх тогтмол бүрийн хувьд $n \geq n_0$ бүрийн хувьд $0 \leq f(n) < cg(n)$ байх $n_0 > 0$ байх тогтмол олдоно$\}$

Өөрөөр хэлбэл,

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0.$$

Жишээлбэл, $n^{1.9999} = o(n^2)$

Дөхөлтийн тэмдэглэгээ VIII

ω тэмдэглэгээ

$\omega(g(n)) = \{f(n)$: бүх $c > 0$ байх тогтмол бүрийн хувьд $n \geq n_0$ бүрийн хувьд $0 \leq cg(n) < f(n)$ байх $n_0 > 0$ байх тогтмол олдоно$\}$

Өөрөөр хэлбэл,

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty.$$

Жишээлбэл, $n^{2.00001} = \omega(n^2)$

Дөхөлтийн тэмдэглэгээ IX

Түгээмэл функцүүд

$$
\begin{aligned}
a &= b^{\log_b a}, \\
\lg n &= \log_2 n, \\
\log_b a^n &= n \log_b a, \\
a^{\log_b c} &= c^{\log_b a}.
\end{aligned}
$$

Хэсэгчлэх алгоритмын шинжилгээ I

Өөрийгөө дахин дуудах (recurrences) бодлогыг дараах 4 аргаар бодно:

- Орлуулах арга: таамаглал дэвшүүлж нэгтгэн дүгнэх байдлаар баталгаа хийнэ.
- Модны арга: Алхам бүрийг модлон зурж төвшин бүрийг тооцоолно.
- Мастер арга: Давталтыг $a > 0,\ b > 1$ байх тогтмол бүхий $T(n) = aT(n/b) + f(n)$ томьёонд орлуулан тооцоолно.
- Акра-Баззийн арга: Өмнөх мастер аргын ерөнхий тохиолдол болно.

Квадрат матрицыг үржүүлэх I

$n \times n$ гурван матриц өгөгджээ: $A = (a_{ij}),\ B = (b_{ij}),\ C = (c_{ij})$.
Тэгвэл $A,\ B$ матрицын үржвэрийг C матрицад бод.

Квадрат матрицыг үржүүлэх II

Энгийн аргаар бодвол

Matrix-Multiply(A, B, C, n)

```
1  for i = i to n
2      for j = 1 to n
3          for k = 1 to n
4              c_ij = c_ij + a_ik · b_kj
```

Ажиллах хугацаа нь: $\Theta(n^3)$

Квадрат матрицыг үржүүлэх III

Хэсэгчлэх аргаар бодвол

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$$\begin{aligned} C_{11} &= A_{11} \cdot B_{11} + A_{12} \cdot B_{21}, \\ C_{12} &= A_{11} \cdot B_{12} + A_{12} \cdot B_{22}, \\ C_{21} &= A_{21} \cdot B_{11} + A_{22} \cdot B_{21}, \\ C_{22} &= A_{21} \cdot B_{12} + A_{22} \cdot B_{22}. \end{aligned}$$

Квадрат матрицыг үржүүлэх IV

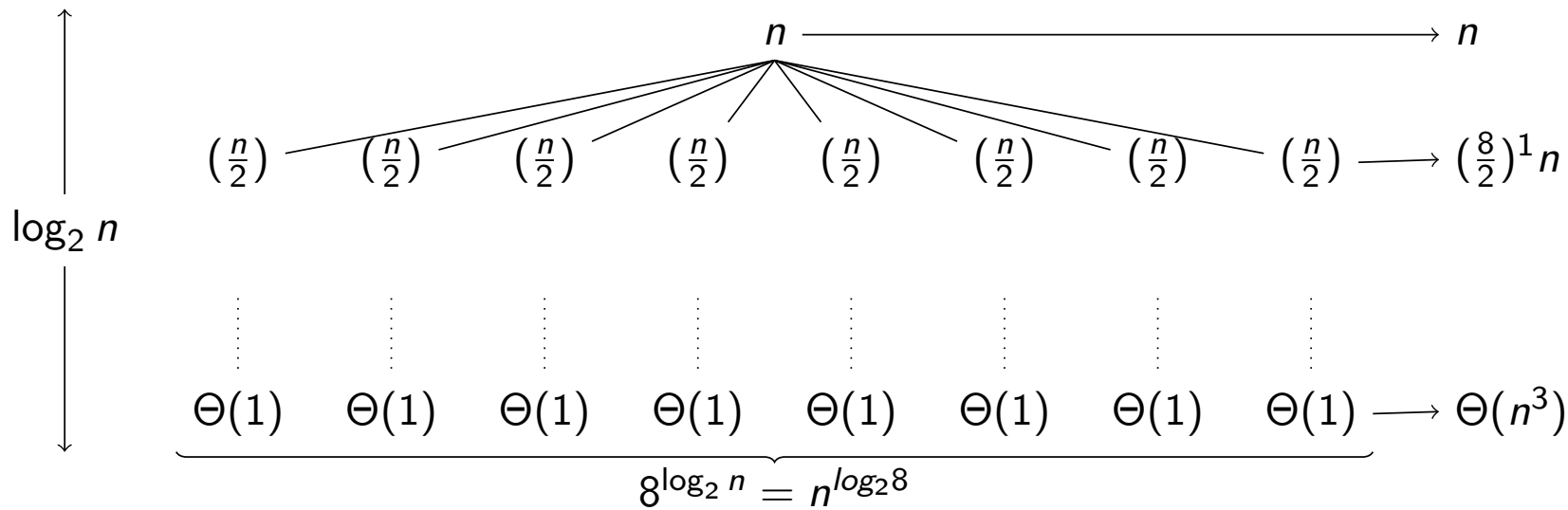
Matrix-Multiply-Recursive(A, B, C, n)

1 **if** $n == 1$
2 $\quad c_{11} = c_{11} + a_{11} \cdot b_{11}$
3 $\quad$ **return**
4 partition A, B and C into $n/2 \times n/2$ submatrices
A_{11}, A_{12}, A_{21}; $B_{11}, B_{12}, B_{21}, B_{22}$; and $C_{11}, C_{12}, C_{21}, C_{22}$
5 Matrix-Multiply-Recursive$(A_{11}, B_{11}, C_{11}, n/2)$
6 Matrix-Multiply-Recursive$(A_{11}, B_{12}, C_{12}, n/2)$
7 Matrix-Multiply-Recursive$(A_{21}, B_{11}, C_{21}, n/2)$
8 Matrix-Multiply-Recursive$(A_{21}, B_{12}, C_{22}, n/2)$
9 Matrix-Multiply-Recursive$(A_{12}, B_{21}, C_{11}, n/2)$
10 Matrix-Multiply-Recursive$(A_{12}, B_{22}, C_{12}, n/2)$
11 Matrix-Multiply-Recursive$(A_{22}, B_{21}, C_{21}, n/2)$
12 Matrix-Multiply-Recursive$(A_{22}, B_{22}, C_{22}, n/2)$

Квадрат матрицыг үржүүлэх V

Шинжилгээ

$$T(n) = 8T(n/2) + \Theta(1) = \Theta(n^3).$$



Квадрат матрицыг үржүүлэх VI

Штрассены алгоритм

Матрицын үржвэрийг олох алгоритмыг хэсэгчлэх аргаар бодох хугацаа $\Theta(n^3)$ байсан билээ. Тэгвэл энэ удаад мөн дээрх хэсэгчлэх аргыг ашиглаад давталтыг 8 биш 7 хүртэл бууруулснаар ажиллах хугацааг $o(n^3)$ хүртэл богиносгох болно. Өөрөөр хэлбэл Штрассены алгоритм нь $\Theta(n^{\lg 7})$ хугацаанд ажиллана.

Квадрат матрицыг үржүүлэх VII

Штрассены алгоритм

Энэхүү аргын гол түлхүүр нь жишээлбэл, $x^2 - y^2$ үржвэрийг тооцохдоо 2 удаа үржүүлэхгүй 1 удаа үржүүлэх үйлдэл хийхэд оршино:

$$x^2 - y^2 = (x + y)(x - y)$$

Орлуулах арга I

Орлуулах аргыг дараах 2 алхмаар гүйцэтгэнэ:

1. Бодолтыг таамаглах
2. Нэгтгэн дүгнэх байдлаар тогтмолыг олж, таамагласан бодолт таарч байгаа эсэхийг шалгана

Энэхүү аргаар ихэвчлэн дээд хязгаар (O) буюу доод хязгаарыг (Ω) тогтоодог.

Жишээ

Дөхөлтийн дээд хязгаарыг тодорхойлох дараах жишээг авч үзье: $T(n) = 2T\lfloor n/2 \rfloor + \Theta(n)$. Үүнийг merge-sort-той төстэй хугацаанд ажиллана гэж таамаглая. Өөрөөр хэлбэл, $T(n) = O(n \lg n)$.

Орлуулах арга II

Нэгтгэн дүгнэх таамаглал

Бүх $n \geq n_0$ хувьд $T(n) \leq cn \lg n$ байна. Хязгаарлалтын нөхцөлийг тогтоомогц $n, n_0 > 0$ тогтмолуудыг олох болно.

Орлуулах арга III

Нэгтгэн дүгнэх алхам

n_0-ээс их буюу тэнцүү n-ээс эрс бага бүх тооны хувьд $T(n) \leq cn \lg n$ гэж таамаглал дэвшүүлье. Хэрэв $n \geq 2n_0$ бол $\lfloor n/2 \rfloor \Rightarrow T(\lfloor n/2 \rfloor) \leq c\lfloor n/2 \rfloor \lg \lfloor n/2 \rfloor$. Өмнөх таамаглалаа орлуулбал:

$$\begin{aligned} T(n) &\leq 2(c\lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)) + \Theta(n) \\ &\leq 2(c(n/2)\lg(n/2)) + \Theta(n) \\ &= cn \ln(n/2) + \Theta(n) \\ &= cn \lg n - cn \lg 2 + \Theta(n) \\ &= cn \lg n - cn + \Theta(n) \\ &\leq cn \lg n. \end{aligned}$$

Энд хэрэв $n \geq 2n_0$ байх c, n_0 хангалттай их утга авах үед $\Theta(n)$ хязгаарыг cn давах болно.

Орлуулах арга IV

Ерөнхий тохиолдол

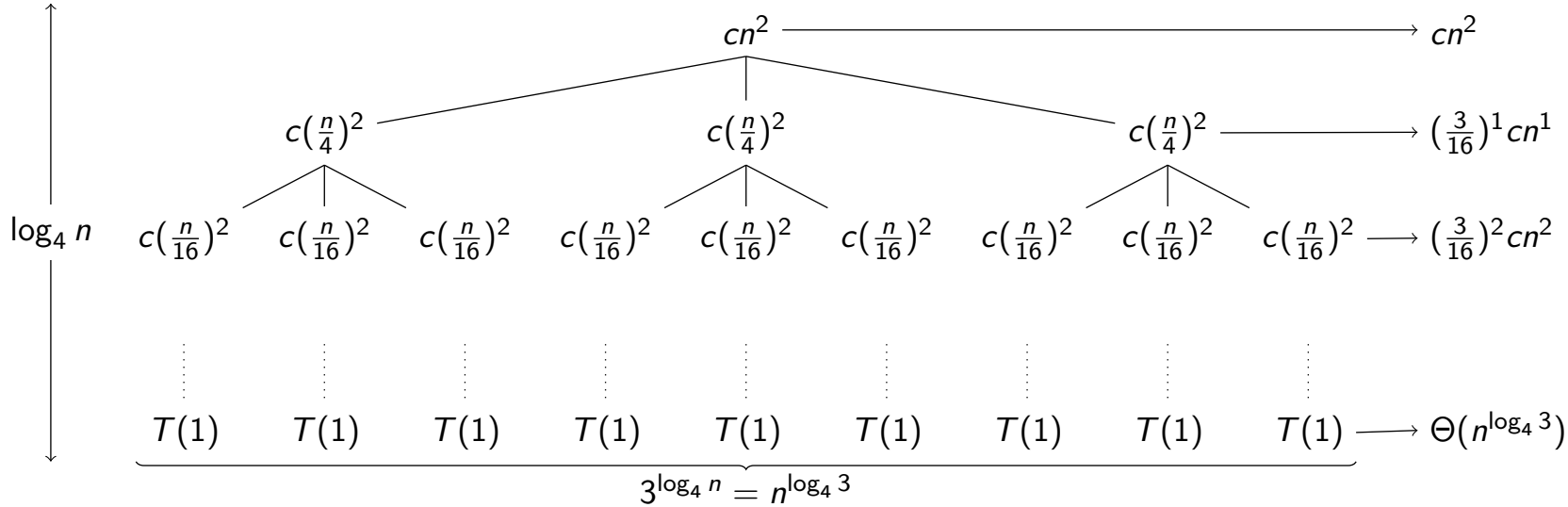
$T(n) \leq cn \lg n$ гэдгийг $n_0 \leq n < 2n_0$ тохиолдолд батлахын тулд хязгаар тогтооё: $n_0 > 1 \Rightarrow \lg n > 0 \Rightarrow n \lg n > 0$. Хэрэв $n_0 = 2$ үед $T(2), T(3)$ тогтмол байна. Хэрэв $c = \max\{T(2), T(3)\}$ сонгох үед $T(2) \leq c < c(2 \lg 2)$ ба $T(3) \leq c < c(3 \lg 3)$ болно.

Дүгнэлт

Бүх $n \geq 2$ хувьд $T(n) \leq cn \lg n$ олдох бол $T(n) = O(n \lg n)$ байна.

Давталтын модны арга I

$$T(n) = 3T(n/4) + \Theta(n^2).$$



$T(1) = \Theta(1)$ гэж бодъё. Тэгвэл i гүнтэй зангилааны нийт тоо нь $n/4^i$ гэдгээс $n/4^i = 1 \Rightarrow n = 4^i \Rightarrow i = \log_4 n$.

Давталтын модны арга II

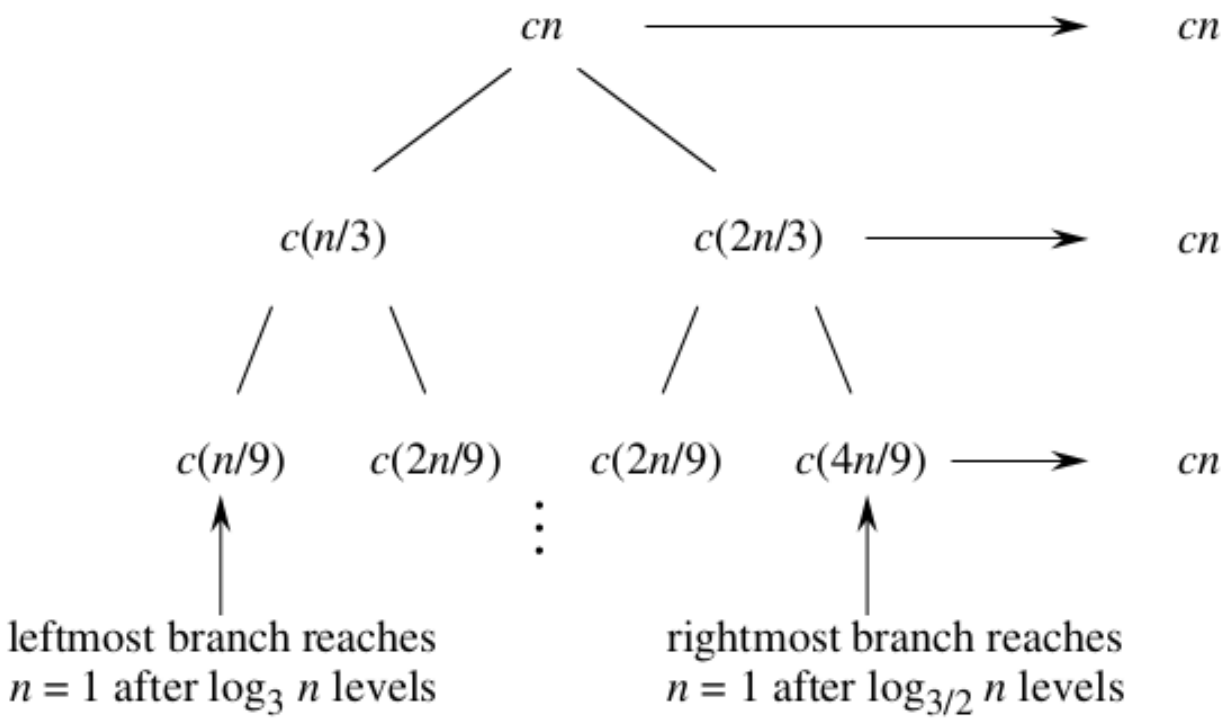
Дээрх модноос нийлбэрийг тооцоолбол:

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{\log_4 n} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\
&< \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\
&= \frac{1}{1-(3/16)} cn^n + \Theta(n^{\log_4 3}) \\
&= \frac{16}{13} cn^2 + \Theta(n^{\log_4 3}) \\
&= O(n^2).
\end{aligned}
$$

Давталтын модны арга III

Өвөрмөц тохиолдол

$$T(n) = T(n/3) + T(2n/3) + \Theta(n).$$



Давталтын модны арга IV

- Мөчир бүрийн орой cn-ээс бага буюу тэнцүү.
- Доод хязгаар нь эерэг d тогтмолын хувьд $dn \log_3 n = \Omega(n \lg n)$-ээс их буюу тэнцүү
- Дээд хязгаар нь эерэг d тогтмолын хувьд $dn \log 3/2n = O(n \lg n)$-ээс бага буюу тэнцүү

Давталтын модны арга V

Дээд хязгаар

$T(n) \leq dn \lg n$ гэж таамаглал дэвшүүлье. Тэгвэл орлуулах аргаар:

$$\begin{aligned}
T(n) &\leq T(n/3) + T(2n/3) + cn \\
&\leq d(n/3)\lg(n/3) + d(2n/3)\lg(2n/3) + cn \\
&= (d(n/3)\lg n - d(n/3)\lg 3) + \\
&\qquad + (d(2n/3)\lg n - d(2n/3)\lg(3/2)) + cn \\
&= dn\lg n - d((n/3)\lg 3 + (2n/3)\lg(3/2)) + cn \\
&= dn\lg n - d((n/3)\lg 3 + (2n/3)\lg 3 - (2n/3)\lg 2) + cn \\
&= dn\lg n - dn(\lg 3 - 2/3) + cn \\
&\leq dn\lg n
\end{aligned}$$

Иймхүү $T(n) = O(n \lg n)$.

Давталтын модны арга VI

Энд

$$-dn(\lg 3 - 2/3) + cn \leq 0 \Rightarrow d \geq \frac{c}{\lg 3 - 2/3}$$

болно.

Давталтын модны арга VII

Доод хязгаар

$T(n) \geq dn \lg n$ гэе. $0 < d \leq \frac{c}{\lg 3 - 2/3}$ гэдгээс $T(n) = \Omega(n \lg n)$ болно.

Ийнхүү $T(n) = O(n \lg n)$ ба $T(n) = \Omega(n \lg n)$ гэдгээс

$$T(n) = \Theta(n \lg n)$$

болно.

Мастер арга I

Хэсэгчлэх аргын ихэнх бодлогуудад мастер арга түгээмэл ашиглагддаг:

$$T(n) = aT(n/b) + f(n).$$

Үүнд, $a \geq 1,\ b > 1$ ба $f(n)$ нь хангалттай их эерэг тооны хувьд дөхөлтийн утга нь сөрөг биш байна.

Энд n тооны оролтын өгөгдөл бүхий бодлогыг n/b хэмжээний a дэд бодлогуудад хувааж бодохыг илэрхийлэх бөгөөд $f(n)$-ийг нөлөөлөгч функц (driving function) гэнэ.

$aT(n/b) = a'T(\lfloor n/b \rfloor) + a''T(\lceil n/b \rceil)$ болно. Үүнд, $a',\ a'' \geq 0$ байх $a = a' + a''$

Одоо $n^{\log_b a}$ гэх гол функцийг (watershed function) $f(n)$ функцтэй харьцуулах замаар бодолт хийнэ:

Мастер арга II

Тохиолдол 1

Зарим $\epsilon > 0$ хувьд $f(n) = O(n^{\log_b a - \epsilon})$ бол хариу нь $T(n) = \Theta(n^{\log_b a})$.

Тохиолдол 2

$k \geq 0$ тогтмолын хувьд $f(n) = \Theta(n^{\log_b a} \lg^k n)$ бол хариу нь $T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$ байна.

Тохиолдол 3

$\epsilon > 0$ байх тогтмолын хувьд $f(n) = \Omega(n^{\log_b a + \epsilon})$ ба $f(n)$ нөлөөлөгч функц нь $c < 1$ байх зарим тогтмолын хувьд $af(n/b) \leq cf(n)$ нөхцөлийг давхар хангаж байвал хариу нь $T(n) = \Theta(f(n))$ байна.

Мастер арга III

Жишээ 1

$T(n) = 5T(n/2) + \Theta(n^2)$ гэдгээс $a = 5,\ b = 2,\ f(n) = n^2$ бөгөөд $n^{\log_2 5}$ гол функцийг n^2 нөлөөлөгч функцтэй харьцуулах замаар бодвол $\log_2 5 - \epsilon = 2$ нөхцөлийг хангах $\epsilon > 0$ байх тогтмол орших учраас тохиолдол 1 ёсоор $T(n) = \Theta(n^{\lg 5})$ болно.

Мастер арга IV

Жишээ 2

$T(n) = 27\,T(n/3) + \Theta(n^3 \lg n)$ гэдгээс $n^{\log_3 27} = n^3$ гол функцийг $n^3 \lg n$ нөлөөлөгч функцтэй харьцуулбал $k = 1$ гэдгээс тохиолдол 2 ёсоор $T(n) = \Theta(n^3 \lg^2 n)$ болно.

Мастер арга V

Жишээ 3

$T(n) = 5T(n/2) + \Theta(n^3)$ гэдгээс $n^{\log_2 5}$ гол функцийг n^3 нөлөөлөгч функцтэй харьцуулбал $\epsilon > 0$ тогтмолын хувьд $\lg 5 + \epsilon = 3$ оршин байх бөгөөд 3-р тохиолдлын нөхцөл $af(n/b) \leq cf(n)$ ёсоор $5(n/2)^3 = 5n^3/8 \leq cn^3$ болох ба энэ нь $c = 5/8 < 1$ тохиолдолд биелэгдэх тул тохиолдол 3 ёсоор $\Theta(n^3)$ болно.

Мастер арга VI

Жишээ 4

$T(n) = 8T(n/2) + \Theta(1)$ өмнөх жишээний хувьд бодвол $a = 8,\ b = 2$ ба гол функц $n^{\log_b a} = n^{\log_2 8} = n^3$ болох ба $n > 0$ тул $\Theta(1) <= O(n^{3-\epsilon})$ байх $0 < \epsilon < 3$ олдох учир 1-р тохиолдол ёсоор $T(n) = \Theta(n^3)$ байна.

Мастер арга VII

Жишээ 5

$T(n) = 27\,T(n/3) + \Theta(n^3/\lg n)$ гэдгээс $n^{\log_3 27} = n^3$ болон $n^3/\lg n = n^3 \lg^{-1} n$ болж 2-р тохиолдолтой ($k \geq 0$ байх ёстой) харшлахад хүрч байна.

Иймхүү энэ тохиолдолд мастер аргыг ашиглах боломжгүй.

Динамик программчлал I

- Энэ нь тодорхой алгоритм биш, харин аргачлал юм.
- Шугаман программчлалын адилаар хүснэгтлэх аргаар гүйцэтгэдэг.
- Оновчлолын бодлогуудад ашиглагддаг. Жишээ нь: хамгийн их, хамгийн бага утгыг олох гэх мэт

Динамик программчлал II

Дөрвөн алхамт арга

1. Бодолтын оновчтой төлөвлөгөөг боловсруулна
2. Бодолтын утгыг өөрийгөө дахин дуудах байдлаар төлөвлөнө
3. Хариуг тооцоолохдоо доороос дээш зарчмыг ихэвчлэн баримтална
4. Өмнө тооцоолсон утгыг дахин ашиглах байдлаар зохион байгуулна

Динамик программчлал III

Ган савааг хуваах бодлого

Ган савааг хамгийн их ашигтайгаар хэрхэн хуваах вэ? Хуваалт буюу тайралт бүрд өртөг тооцохгүй. Хуваалтын уртын хэмжээ нь бүхэл тоогоор илэрхийлэгдэнэ.

Оролт n урт болон p_i үнэ бүхий талбар өгөгдөнө. Үүнд, $i = 1, 2, \ldots, n$

Гаралт Ган савааны уртын нийлбэр нь n байхаар хамгийн өндөр ашиг өгөхөөр хуваана.

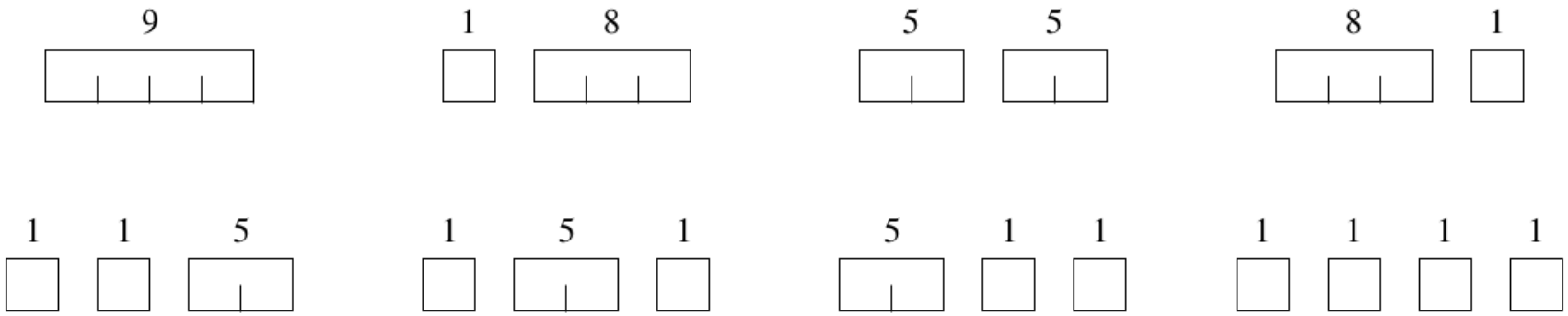
Жишээлбэл,

i урт	1	2	3	4	5	6	7	8
p_i үнэ	1	5	8	9	10	17	17	20

Динамик программчлал IV

Нийтдээ 2^{n-1} ялгаатай янзаар хувааж болно. Учир нь $n-1$ уртын хувьд хуваалт хийгдэх тухай яригдана.

Жишээлбэл, 4 урттай ган савааг дараах байдлаар хувааж болно.



Эндээс хамгийн өндөр ашиг өгөх хуваалт нь $p_2 + p_2 = 5 + 5 = 10$ болно.

Динамик программчлал V

r_i нь i урттай ган сабааны хамгийн их ашиг бол хамгийн оновчтой хуваалт хийх r_i ашгийг дараах байдлаар тооцно:

i	r_i	оновчтой хувилбар
1	1	1 (хуваахгүй)
2	5	2 (хуваахгүй)
3	8	3 (хуваахгүй)
4	10	2+2
5	13	2+3
6	17	6 (хуваахгүй)
7	18	1+6 эсвэл 2+2+3
8	22	2+6

Динамик программчлал VI

Хамгийн их ашиг r_n нь

- p_n: үл хуваах үеийн ашиг
- $r_1 + r_{n-1}$: 1 болон $n-1$ уртаар хуваах үеийн хамгийн их ашиг
- $r_2 + r_{n-2}$: 2 болон $n-2$ уртаар хуваах үеийн хамгийн их ашиг...
- $r_{n-1} + r_1$.

Нийт ашиг $r_n = \max\{p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \ldots, r_{n-1} + r_1\}$.

Динамик программчлал VII

Оновчтой шийд

n хэмжээний өгөгдөл бүхий эл бодлогыг жижиг хэмжээтэй дэд бодлогуудад хувааж бодно.

Бодлогыг задлан хялбарчлах арга

- Зөвхөн үлдэгдлийг дахин хуваана
- 2 бус зөвхөн 1 дэд бодлогыг үлдээнэ
- Хэрэв саваг тайралгүй p_n ашиг олох бол үлдэгдэл хэмжээ нь 0 бөгөөд ашиг нь $r_0 = 0$ болно гэж тооцно
- r_n ашгийг хялбарчилж томьёолбол:
  $r_n = \max\{p_i + r_{n-i} : 1 \leq i \leq n\}$.

Динамик программчлал VIII

Дээрээс доош давтах аргаар бодох

Cut-Rod(p, n)

```
1  if n == 0
2        return 0
3  q = -∞
4  for i = 1 to n
5        q = max{q, p[i] + Cut-Rod(p, n - i)}
6  return q
```

Энэхүү бодолт нь хэдийгээр дэд бодлогуудад хуваагдан ажиллах ч төдийлөн үр ашигтай биш юм. Учир нь нэг тооцооллоо дэд бодлого бүр дахин дахин тооцоолно.

Динамик программчлал IX

$$T(n) = \begin{cases} 1, & \text{хэрэв } n = 0 \\ 1 + \sum_{j=0}^{n-1} T(j), & \text{хэрэв } n \geq 1 \end{cases}$$

Ийнхүү ажиллах хугацаа нь $T(n) = 2^n$ болно.

Динамик программчлал X

Оновчтой шийд

Нэг тооцооллоо давтан тооцоолохгүйн тулд тооцоолсон утгаа хадгалах хэрэгтэй юм. Үүнийг memoizing гэдэг.

Динамик программчлал XI

Memoizing

Memoized-Cut-Rod(p, n)

```
1  let i = 0 be a new array
2  for i = 0 to n
3        r[i] = −∞
4  return Memoized-Cut-Rod-Aux(p, n, r)
```

Динамик программчлал XII

Memoized-Cut-Rod-Aux(p,n,r)

1 **if** $r[n] \geq 0$
2 **return** $r[n]$
3 **if** $n == 0$
4 $q = 0$
5 **else** $q = -\infty$
6 **for** $i = 1$ **to** n
7 $q = \max\{q, p[i] + \text{Memoized-Cut-Rod-Aux}(p, n - i, r)\}$
8 $r[n] = q$
9 **return** q

Динамик программчлал XIII

Доороос дээшээ

Энэхүү арга нь дэд бодлогыг хэмжээгээр нь ангилж, жижгийг нь түрүүлж бодох бөгөөд дэд бодлогыг бодохын тулд түрүүнд бодогдсон жижиг дэд бодлогуудыг ашиглана.

Bottom-Up-Cut-Rod(p,n)

```
1  let r[0 : n] be a new array
2  r[0] = 0
3  for j = 1 to n
4        q = −∞
5        for i = 1 to j
6              q = max{q, p[i] + r[j − i]}
7        r[j] = q
8  return r[n]
```

Динамик программчлал XIV

Ажиллах хугацаа

Дээрээс доош, доороос дээш аргачлалууд аль аль нь $\Theta(n^2)$ хугацаанд ажиллана. Дурдсан жишээнүүдийн хувьд доороос дээшээ аргачлал нь давхар давталтаар ажиллаж байгаа бол дээрээс доошоо аргачлал нь нэгэнт хийгдсэн тооцооллыг дахин хийхгүй учир n тооны бодлогыг n удаа гүйлгэнэ.

Динамик программчлал XV

Ган савааг хуваах бодлогын өртөг тооцох хэсгийг сайжруулсан бол энэ удаад ашигтай хувилбарыг сонгох хэсгийг сайжруулах болно.

Extended-Bottom-Up-Cut-Rod(p,n)

```
1   let r[0 : n] and s[1 : n] be new arrays
2   r[0] = 0
3   for j = 1 to n                  // for increasing rod length j
4         q = −∞
5         for i = 1 to j
6               if q < p[i] + r[j − i]
7                     q = p[i] + r[j − i]
8                     s[j] = i    // best cut location so far for length j
9         r[j] = q                  // for increasing rod length j
10  return r and s
```

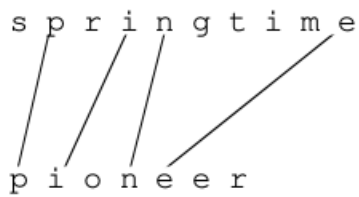
Динамик программчлал XVI

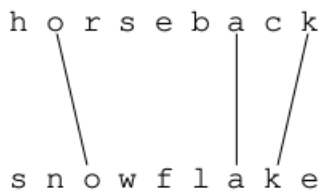
```
Print-Cut-Rod-Solution(p,n)
1   (r,s)=Extended-Bottom-Up-Cut-Rod(p,n)
2   while n > 0
3         print s[n]
4         n = n − s[n]
```

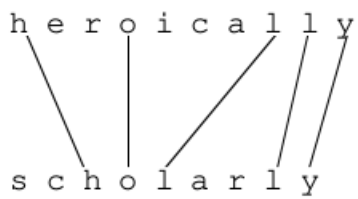
Динамик программчлал XVII

Хамгийн урт нийтлэг дэд дараалал (LCS)

Бодлого: $X = \langle x_1, \ldots, x_m \rangle$ ба $Y = \langle y_1, \ldots, y_n \rangle$ гэсэн хоёр дараалал өгөгджээ. Эдгээр дарааллуудад зэрэг агуулагдах хамгийн урт дэд дарааллыг ол. Энэхүү дарааллын элементүүд нь дараалан орсон байх албагүй ч нийтлэг элементүүдийн байр өмнө хойноо орохгүй.







Динамик программчлал XVIII

Хар аргаар X дарааллын дэд дараалал бүрийг Y дарааллын дэд дараалалтай харьцуулахад ажиллах хугацаа нь: $\Theta(n2^m)$.

- X дарааллын 2^m дэд дарааллыг шалгах
- Дэд дараалал бүрийг шалгахад $\Theta(n)$ хугацаа зарцуулна

Динамик программчлал XIX

Алхам 1: LCS тодорхойлох

Өгөгдсөн $X = \langle x_1, x_2, \ldots, x_m \rangle$ дарааллын i-р угтвар нь $X_i = \langle x_1, x_2, \ldots, x_i \rangle$ байна. Үүнд: $i = 0, 1, \ldots, m$.

Theorem

$X = \langle x_1, x_2, \ldots, x_m \rangle$ ба $Y = \langle y_1, y_2, \ldots, y_n \rangle$ дараалуудын хувьд $Z = \langle z_1, z_2, \ldots, z_k \rangle$ нь X болон Y дарааллын хамгийн урт нийтлэг дэд дараалал байг.

1. *Хэрэв $x_m = y_n$ бол $z_k = x_m = y_n$ байх ба Z_{k-1} нь X_{m-1} болон Y_{n-1} дарааллын хамгийн урт нийтлэг дэд дараалал байна.*
2. *Хэрэв $x_m \neq y_n$ ба $z_k \neq x_m$ бол Z нь X_{m-1} ба Y дарааллын хамгийн урт нийтлэг дэд дараалал байна.*
3. *Хэрэв $x_m \neq y_n$ ба $z_k \neq y_n$ бол Z нь X болон Y_{n-1} дарааллын хамгийн урт нийтлэг дэд дараалал байна.*

Динамик программчлал XX

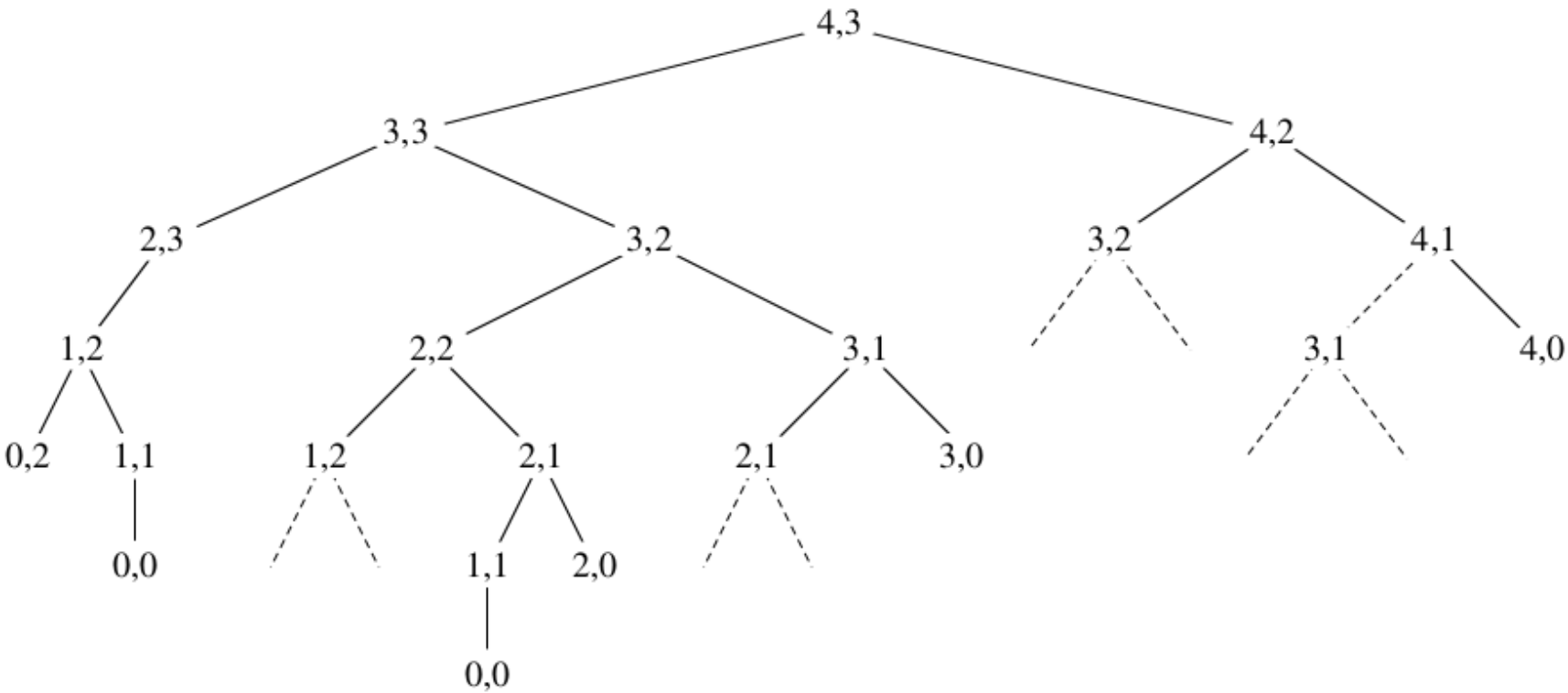
Алхам 2: Дахин дуудах шийдэл

$c[i,j]$ нь X_i ба Y_j дарааллуудын хамгийн урт нийтлэг дэд дараалал нь $c[i,j]$ бол

$$c[i,j] = \begin{cases} 0, & \text{хэрэв } i = 0 \text{ буюу } j = 0, \\ c[i-1,j-1]+1, & \text{хэрэв } i,j > 0 \text{ ба } x_i = y_j, \\ \max(c[i-1,j], c[i,j-1]) & \text{хэрэв } i,j > 0 \text{ ба } x_i \neq y_j. \end{cases}$$

Дээрх мэдээлэлд үндэслэн дахин дуудах алгоритм боловсруулна. Доорх зурагт тасархай шугамаар өмнө тооцоолсон дэд бодлогыг тэмдэглэв.

Динамик программчлал XXI



Динамик программчлал XXII

LCS-Length(X,Y,m,n)

```
 1  let b[1 : m, 1 : n] and c[0 : m, 0 : n] be new tables
 2  for i = 1 to m
 3        c[i, 0] = 0
 4  for j = 0 to n
 5        c[0, j] = 0
 6  for i = 1 to m
 7        for j = 1 to n
 8              if x_i == y_i
 9                    c[i, j] = c[i − 1, j − 1] + 1
10                    b[i, j] = " ↖ "
11              else if c[i − 1, j] ≥ c[i, j − 1]
12                    c[i, j] = c[i − 1, j]
13                    b[i, j] = " ↑ "
14              else c[i, j] = c[i, j − 1]
15                    b[i, j] = " ← "
16  return c and b
```

Динамик программчлал XXIII

i		j	0	1	2	3	4	5	6
		y_j		B	D	C	A	B	A
0	x_i		0	0	0	0	0	0	0
1	A		0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B		0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C		0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B		0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D		0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A		0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B		0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Динамик программчлал XXIV

Сайжруулалт

b хүснэгт шаардлагагүй.

Динамик программчлал XXV

Хоёртын модны оновчтой хайлт

Цөөн тооны зангилааг дайран гарахаар хоёртын модны хайлтыг зохион байгуулна.

- n ялгаатай түлхүүр бүхий $K = \langle k_1, k_2, \ldots, k_n \rangle$ дараалал өгөгдсөн байг. $(k_1 < k_2 < \cdots < k_n)$
- Дээрх түлхүүрээр хоёртын хайлтын модыг босгоно.
- k_i түлхүүрийн хувьд хайлтын утга олдох магадлал p_i
- Хайлтын хамгийн бага өртгөөр хоёртын модны хайлтыг гүйцэтгэх
- k_i түлхүүрийн хувьд өртөг$= depth_T(k_i) + 1$ үүнд, $depth_T(k_i)$ нь T хоёртын хайлтын модны гүн.

Динамик программчлал XXVI

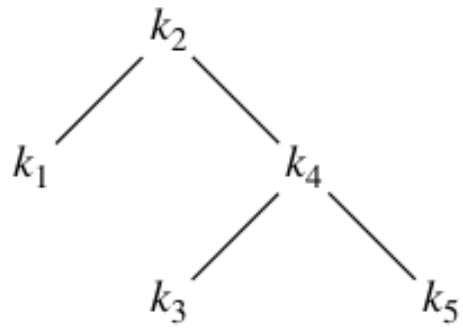
$$
\begin{aligned}
E[T \text{ дэх хайлтын өртөг}] &= \sum_{i=1}^{n} (depth_T(k_i) + 1) \cdot p_i = \\
&= \sum_{i=1}^{n} depth_T(k_i) \cdot p_i + \sum_{i=1}^{n} p_i = \\
&= \sum_{i=1}^{n} depth_T(k_i) \cdot p_i + 1
\end{aligned}
$$

Динамик программчлал XXVII

Жишээ

i	1	2	3	4	5
p_i	.25	.2	.05	.2	.3

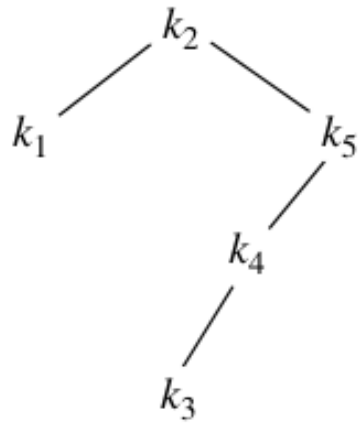
Динамик программчлал XXVIII



i	$\text{depth}_T(k_i)$	$\text{depth}_T(k_i) \cdot p_i$
1	1	.25
2	0	0
3	2	.1
4	1	.2
5	2	.6
		1.15

Ийнхүү хайлтын өртөг нь $E = 2.15$ болно.

Динамик программчлал XXIX



i	$\text{depth}_T(k_i)$	$\text{depth}_T(k_i) \cdot p_i$
1	1	.25
2	0	0
3	3	.15
4	2	.4
5	1	.3
		1.10

Хайлтын өртөг нь $E = 2.10$ болж, оновчтой хувилбар болно.

Динамик программчлал XXX

Ажиглалт

- Хоёртын оновчтой хайлтын модны (Optimal BST) өндөр нь хамгийн бага утга авахгүй
- Хоёртын оновчтой хайлтын модны хамгийн өндөр магадлалтай (давтамжтай) түлхүүр нь оройн зангилаа үл болно.

Динамик программчлал XXXI

Алхам 1: OBST бүтэц

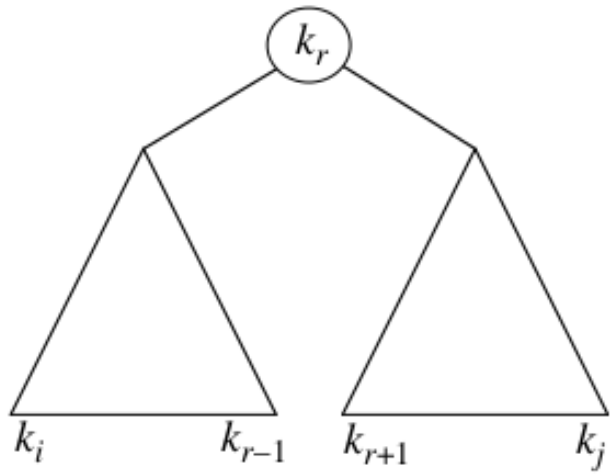
Дурын дэд хоёртын хайлтын модны хувьд энэ нь $1 \leq i \leq j \leq n$ байх $k_i, \ldots, k_j$ хөрш оройнуудад түлхүүрээ агуулна.
Хэрэв T нь хоёртын оновчтой хайлтын мод бол энэ нь $k_i, \ldots, k_j$ түлхүүр бүхий T' дэд мод агуулна гэвэл T' нь $k_i, \ldots, k_j$ түлхүүр бүхий хоёртын оновчтой хайлтын мод байна.

Динамик программчлал XXXII

Оновчтой шийдлийг боловсруулахын тулд оновчтой дэд модыг ашиглана:

- $k_i, \ldots, k_j$ түлхүүр өгөгдсөн байг
- Тэдгээр түлхүүрийн нэг k_r $(i \leq r \leq j)$ нь орой зангилаа байна
- k_r модны зүүн дэд мод нь $k_i, \ldots, k_{r-1}$ түлхүүр агуулна
- баруун дэд мод нь $k_{r+1}, \ldots, k_j$ түлхүүр агуулна
- Хэрэв $i \leq r \leq j$ байх бүх оройн зангилаа k_r бүрийг шалгаад
- $k_i, \ldots, k_{r-1}$ болон $k_{r+1}, \ldots, k_j$ түлхүүр агуулсан бүх хоёртын оновчтой хайлтын моднуудыг тогтоосон бол $k_i, \ldots, k_j$ түлхүүр бүхий хоёртын оновчтой хайлтын мод олдох нь гарцаагүй.

Динамик программчлал XXXIII



Динамик программчлал XXXIV

Алхам 2: Дахин дуудах шийдэл

Хайлтын өртөг:

$$w(i,j) = \sum_{l=i}^{j} p_l$$

Хэрэв k_r нь оройн зангилаа бол хайлтын хүлээсэн өртөг (expected search cost)

$$e[i,j] = p_r + (e[i,r-1] + w(i,r-1)) + (e[r+1,j] + w(r+1,j)).$$

Динамик программчлал XXXV

$$e[i,j] = \begin{cases} 0, & j = i-1, \\ \min\{e[i,r-1] + e[r+1,j] + w(i,j) : i \le r \le j\} & i \le j. \end{cases}$$

Динамик программчлал XXXVI

Алхам 3: Тооцоолол

Динамик программчлал XXXVII

Optimal-BST(p,q,n)

```
 1  let e[1 : n + 1, 0 : n], w[1 : n + 1, 0 : n], and root[1 : n, 1 : n] be new tables
 2  for i = 1 to n + 1
 3        e[i, i − 1] = 0
 4        w[i, i − 1] = 0
 5  for l = 1 to n
 6        for i = 1 to n − l + 1
 7              j = i + l − 1
 8              e[i, j] = ∞
 9              w[i, j] = w[i, j − 1] + p_j
10              for r = i to j
11                    t = e[i, r − 1] + e[r + 1, j] + w[i, j]
12                    if t < e[i, j]
13                          e[i, j] = t
14                          root[i, j] = r
15  return e and root
```

Динамик программчлал XXXVIII

Ажиллах хугацаа: $O(n^3)$ ба $\Omega(n^3)$ учир $\Theta(n^3)$.

Динамик программчлал XXXIX

Алхам 4: Хоёртын оновчтой хайлтын мод бүтээх

Construct-Optimal-BST(root)

1 $r = root[1, n]$
2 print "k_r" "is the root"
3 Construct-Opt-Subtree$(1, r-1, r,$ "left", $root)$
4 Construct-Opt-Subtree$(r+1, n, r,$ "right", $root)$

Construct-Optimal-Subtree(i,j,r,dir,root)

1 If $i \leq j$
2 $\quad t = root[i, j]$
3 $\quad$ print "k_t" "is" dir "child of k_r"
4 $\quad$ Construct-Opt-Subtree$(i, t-1, t,$ "left", $root)$
5 $\quad$ Construct-Opt-Subtree$(t+1, j, t,$ "right", $root)$

Динамик программчлал XL

Динамик программын үндэс

Үүнд:

- Оновчтой дэд бүтэц
- Дэд бодлогуудын давхцал

Динамик программчлал XLI

Оновчтой дэд бүтэц

- Бодлого нь нэг буюу олон шийдэл бүхий сонголтоос бүрдсэн эсэх
- Оновчтой шийдэд хүргэх дэд бодлого сонгогдлоо хэмээн төсөөлье
- Оновчтой шийдэд хүргэх дэд бодлогууд оновчтой эсэхийг шалгана. Үүний тулд: Аль нэгэн дэд бодлогын шийдэл нь оновчгүй бол үүнийг оновчтой шийдлийг хэсэг уруу зөөнө

Динамик программчлал XLII

Дэд бодлогуудын давхцал

Дахин дуудах алгоритмын (дээрээс доошоо) хувьд энэхүү давхцал үүсдэг. Үүнийг динамик программчлалын хувьд өмнө гарган авсан утгаа хадгалах (memoization) замаар шийддэг.

Хомхойлох алгоритм (Greedy algorithms) I

Санаа

Тухайн агшинд хийж байгаа сонголт нь хэтдээ оновчтой сонголт хэмээн найдаж, хамгийн сайн гэсэн сонголтыг хийнэ. Энэ алгоритм нь ямагт оновчтой шийдэд хүргэдэггүй.

Хомхойлох алгоритм (Greedy algorithms) II

Үйл ажиллагааг сонгох

Нийтлэг нөөц боломжийг n үйл ажиллагаанд хуваарилах. Үйл ажиллагааны олонлог: $S = \{a_1, \ldots, a_n\}$. a_i нь $[s_i, f_i)$ хугацааг шаардана. s_i нь эхлэх цаг, f_i нь дуусах хугацаа.

Хомхойлох алгоритм (Greedy algorithms) III

Зорилго

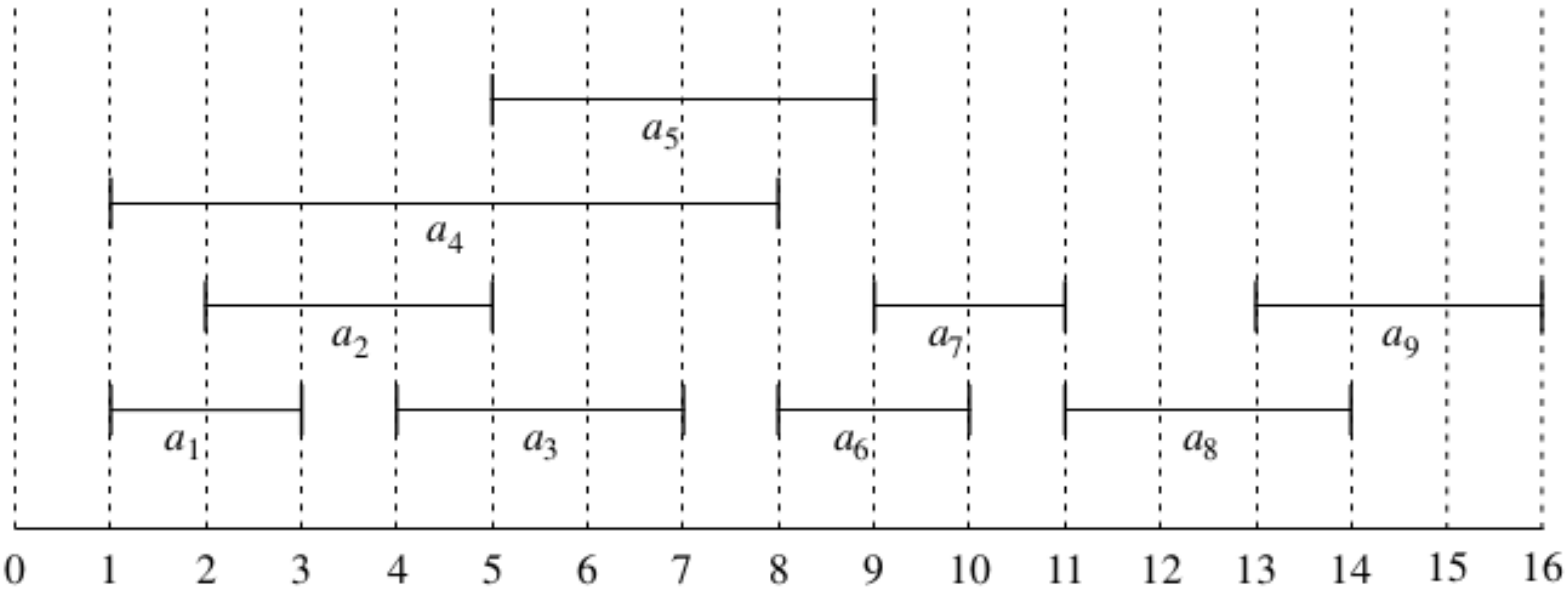
Хоорондоо үл давхцах хамгийн удаан үргэлжлэх үйл ажиллагааг сонгох:

- Хамгийн удаан үргэлжлэх өрөөг захиалах
- Түрээсийн орлогыг хамгийн өндөр хэмжээнд барих

Үйл ажиллагаа нь дуусах хугацаагаараа эрэмбэлэгдсэн гэж төсөөлнө: $f_1 \leq f_2 \leq f_3 \ldots f_{n-1} \leq f_n$.

i	1	2	3	4	5	6	7	8	9
s_i	1	2	4	1	5	8	9	11	13
f_i	3	5	7	8	9	10	11	14	16

Хомхойлох алгоритм (Greedy algorithms) IV



Хамгийн урт хугацаагаар үргэлжлэх олонлог: $\{a_1, a_3, a_6, a_8\}$

Хомхойлох алгоритм (Greedy algorithms) V

Үйл ажиллагааг сонгох оновчтой бүтэц

$S_{ij} = \{a_k \in S : f_i \leq s_k < f_k \leq s_J\}$ байх S_{ij} олонлогт хамаарах үйл ажиллагаа нь:

- Бүх үйл ажиллагаа нь f_i цагаас өмнө дуусна
- Бүх үйл ажиллагаа нь s_J цагт эхэлнэ.

Хомхойлох алгоритм (Greedy algorithms) VI

A_{ij} нь S_{ij} олонлогт харьяалагдах хамгийн урт хэмжээтэй дэд олонлог байг. $a_k \in A_{ij}$ нь A_{ij} олонлогт харьяалагдах үйл ажиллагаа бол дараах 2 дэд бодлого олдоно:

- S_{ik} олонлогт харьяалагдах (a_i хугацаанд эхлээд a_k хугацаанаас өмнө дуусах)
- S_{kj} олонлогт харьяалагдах (a_k хугацаанд эхлээд a_j хугацаанаас өмнө дуусах)

Хомхойлох алгоритм (Greedy algorithms) VII

A_{ij} олонлогт харьяалагдах, a_k эхлэхээс өмнө дуусах $A_{ik} = A_{ij} \cap S_{ik}$.
a_k дууссаны дараа эхлэх $A_{kj} = A_{ij} \cap S_{kj}$ байг. Тэгвэл
$A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj} \Rightarrow |A_{ij}| = |A_{ik}| + |A_{kj}| + 1$

Дүгнэлт

A_{ij} оновчтой шийд нь S_{ik} ба S_{kj} олонлогуудын хоёр дэд бодлогын оновчтой шийдлийг агуулна.

Хомхойлох алгоритм (Greedy algorithms) VIII

Дахин дуудах шийдэл

A_{ij} оновчтой шийд нь S_{ik} ба S_{kj} олонлогуудын дэд бодлогын оновчтой шийдлийг агуулах тул үүнийг динамик программчлалын аргаар бодож болно.
$c[i,j]$ нь S_{ij} оновчтой шийдлийн хэмжээ бол
$c[i,j] = c[i,k] + c[k,j] + 1$ байна.

$$c[i,j] = \begin{cases} 0, & \text{хэрэв } S_{ij} = \emptyset \\ \max\{c[i,k] + c[k,j] + 1 : a_k \in S_{ij}\}, & \text{хэрэв } S_{ij} \neq \emptyset \end{cases}$$

Хомхойлох алгоритм (Greedy algorithms) IX

Хомхой сонголт хийх

Үйл ажиллагааг сонгох бодлогын хувьд бид хомхой сонголтыг чухалчилж болохгүй, бусад үйл ажиллагаанд нөөц боломжийг үлдээх хэрэгтэй.
Асуулт: Аль үйл ажиллагаа нь бусад үйл ажиллагаанд нөөц үлдээдэг вэ?
Хариулт: Хамгийн эхний дуусах үйл ажиллагаа.

Теорем

Хэрэв S_k нь хоосон биш олонлог бөгөөд a_m нь хамгийн эрт дуусах хугацаа бол a_m нь зарим оновчтой шийдэд агуулагдана.

Хомхойлох алгоритм (Greedy algorithms) X

Дээрээс доошоо аргаар дэд бодлогуудыг шийднэ. Сонголт хийхээс өмнө дэд бодлогуудыг шийдэх шаардлагагүй.

Хомхойлох алгоритм (Greedy algorithms) XI

Дахин давталтат хомхойлох алгоритм

Recursive-Activity-Selector(s, f, k, n)

```
1  m = k + 1
2  while m ≤ n and s[m] < f[k]
3        m = m + 1
4  if m ≤ n
5        return {a_m} ∪ Recursive-Activity-Selector(s, f, m, n)
6  else return ∅
```

Эхлэн дуудахдаа: Recursive-Activity-Selector($s, f, 0, n$).
Санаа нь: $a_{k+1}, a_{k+2}, \ldots, a_n$ хүртэл a_k үйл ажиллагаатай $s_m \geq f_k$ байх a_m үйл ажиллагааг хайж олно.

- a_m олдоод дахин дуудалтаар S_m үйл ажиллагааг шийднэ
- Хэрэв a_m ($m > n$) олдохгүй бол хоосон олонлог буцаана.

Ажиллах хугацаа: $\Theta(n)$

Хомхойлох алгоритм (Greedy algorithms) XII

Давталтат хомхойлох алгоритм

Дахин дуудах алгоритмыг давтах алгоритм уруу шилжүүлж болно.

Greedy-Activity-Selector(s, f, n)

```
1  A = {a_1}
2  k = 1
3  for m = 2 to n
4        if s[m] ≥ f[k]
5              A = A ∪ {a_m}
6              k = m
7  return A
```

Ажиллах хугацаа: $\Theta(n)$

Хомхойлох алгоритм (Greedy algorithms) XIII

Хомхойлох аргачлалын үндэс

Энэхүү аргачлал нь тухайн агшинд хамгийн сайн гэсэн сонголтыг хийнэ. Улмаар

1. Оновчтой дэд шийдлийг тодорхойлно.
2. Дахин дуудах шийдлийг боловсруулна.
3. Хэрэв хомхой сонголт хийвэл зөвхөн ганц дэд бодлого үлдэнэ гэдгийг харуулна.
4. Хомхой сонголт хийх нь хожим алдагдалгүй гэдгийг батална.
5. Дахин дуудах хомхойлох алгоритм боловсруулах
6. Үүнийгээ давталтын алгоритм уруу хөрвүүлэх

Хомхойлох алгоритм (Greedy algorithms) XIV

Хомхойлох аргачлалын үндэс

Хомхойлох алгоритм нь оновчтой эсэхийг харуулах аргачлал байхгүй ч дараах хоёр зүйлийг шалгуур болгоно:

1. хомхой сонголтын шинж чанар
2. оновчтой дэд бүтэц

Хомхойлох алгоритм (Greedy algorithms) XV

Хомхой сонголтын шинж чанар

Тухайн тохиолдолд оновчтой байхаар хомхой сонголт хийж улмаар ерөнхий тохиолдолд оновчтой шийдэд хүрч болно. Динамик программчлалын аргаас ялгарах онцлогийг нь авч үзье.

Хомхойлох алгоритм (Greedy algorithms) XVI

Ялгаа

Динамик программчлал:

- Алхам тутамд сонголт хийнэ
- Сонголт нь дэд бодлогын хувьд оновчтой шийд байх эсэхээс хамаарна. Өөрөөр хэлбэл, дэд бодлогыг эхэлж бодно.
- Доороос дээш аргачлалаар бодно.

Хомхойлох:

- Алхам тутамд сонголт хийнэ
- Дэд бодлогыг бодохоос өмнө сонголт хийнэ
- Дээрээс доош аргачлалаар бодно

Хомхойлох алгоритм (Greedy algorithms) XVII

Оновчтой дэд бүтэц

Үүргэвчтэй бодлогын хувьд хомхойлох болон динамик программчлалын аргуудын ялгааг авч үзье.

Хомхойлох алгоритм (Greedy algorithms) XVIII

Үүргэвчтэй бодлого (0-1)

- n бараа
- i -р бараа v_i үнэтэй, w_i жинтэй
- Нийт жин нь W -ээс бага буюу тэнцүү, нийт үнэ нь хамгийн өндөр байх бараануудыг сонгоно.

Хомхойлох алгоритм (Greedy algorithms) XIX

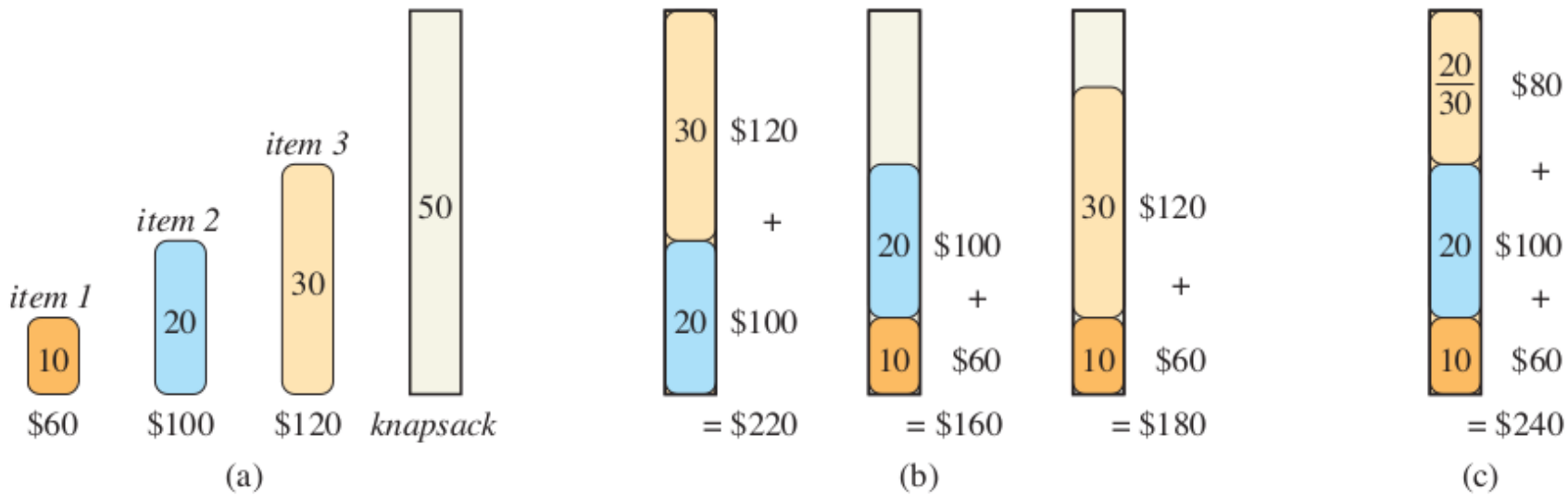
Үүргэвчтэй бодлого (бутархай)

Өмнөх бодлогоос ялгаатай нь барааг хэсэгчлэн хуваааж болно. Энэ бодлогын хувьд хомхой сонголтын шинж чанарыг агуулсан бол өмнөх бодлого тийм биш. Өөрөөр хэлбэл, бараа тус бүрийг, үнийг нь жинд нь харьцуулах замаар үнэлж хамгийн өндөр үнэтэй барааг эхэлж сонгох зарчмыг баримтална.

Хомхойлох алгоритм (Greedy algorithms) XX

Үүргэвчтэй бодлого

Үүргэвчтэй бодлогыг (бүхэл) хомхойлох аргачлалаар яагаад бодож болохгүй вэ гэдгийг жишээгээр тайлбарлая.



Хомхойлох алгоритм (Greedy algorithms) XXI

Үүргэвчтэй бодлого

Энд, a нь анхны өгөгдөл, b нь 0-1 бодлогын оновчтой хувилбар бол c нь бутархай бодлогын хувьд оновчтой хувилбар. 0-1 бодлогын (b) хувьд хомхойлох аргачлалыг ашиглавал эхэлж хамгийн үнэтэйг сонгох тул хариу нь $220-оос бага байна.

Хомхойлох алгоритм (Greedy algorithms) XXII

Үүргэвчтэй бодлого (бутархай)

Fractional-Knapsack(v, w, W)

1 $load = 0$
2 $i = 1$
3 **while** $load < W$ and $i \leq n$
4 **if** $w_i \leq W - load$
5 take all of item i
6 **else** take $(W - load)/w_i$ of item i
7 add what was taken to $load$
8 $i = i + 1$

Ажиллах хугацаа: $O(nlgn)$

Хомхойлох алгоритм (Greedy algorithms) XXIII

Хаффманы код

Хаффманы код нь өгөгдлийг, тэмдэгтээс хамаарч 20%–90% хүртэл шахдаг. Хаффманы хомхойлох алгоритм нь хүснэгт ашиглан тэмдэгтийн давтамжийг тооцоолж, хоёртын мод байгуулах замаар хоёртын тэмдэгт уруу хөрвүүлдэг.

Хомхойлох алгоритм (Greedy algorithms) XXIV

Хаффманы код

Жишээлбэл, 100,000 тэмдэгт бүхий мэдээллийг байна гэж саная. Тэгвэл үүнд, a тэмдэгт 45,000 удаа, b тэмдэгт 13,000 удаа гэх мэт давтагдан оржээ.

	a	b	c	d	e	f
Давтамж (мянгаар)	45	13	12	16	9	5
Ижил урттай код	000	001	010	011	100	101
Ялгаатай урттай код	0	101	100	111	1101	1100

Мэдээллийг олон төрлийн хэлбэрээр тэмдэглэж болно: хоёртын тэмдэгт, ижил урттай, ялгаатай урттай код

Хомхойлох алгоритм (Greedy algorithms) XXV

100,000 тэмдэгтийг ижил урттай кодоор тэмдэглэвэл 300,000 бит, харин ялгаатай урттай кодоор тэмдэглэвэл 224,000 бит буюу 25%-аар бага зай эзэлнэ.

Хомхойлох алгоритм (Greedy algorithms) XXVI

Угтваргүй код

Мэдээллийг шахахад угтваргүй код (prefix-free codes) ашиглах нь ямагт хамгийн бага зай эзэлдэг.
Кодлохдоо хоёртын кодуудыг өөр хооронд нь залгах байдлаар угсардаг. Жишээлбэл: face гэсэн 4 тэмдэгтийг
$1100 \cdot 0 \cdot 100 \cdot 1101 = 110001001101$
Кодоо тайлахдаа жишээлбэл, 100011001101 кодын хувьд 100=c гэвэл үлдэх 011001101 тэмдэгтийн эхний 0=a, харин 1100=f, сүүлийн 1101=e болж харгалзаж, cafe гэсэн үг гарна.

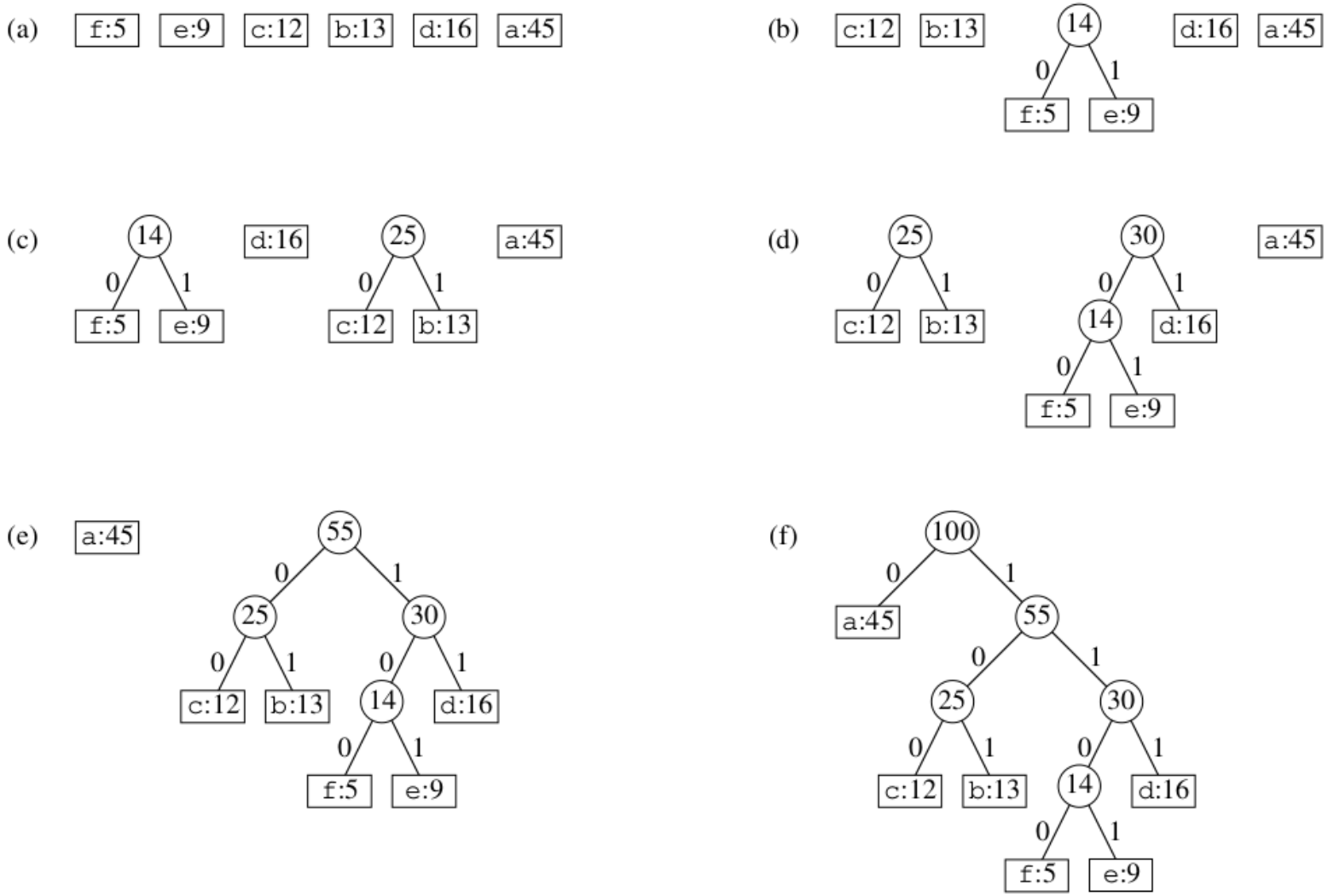
Хомхойлох алгоритм (Greedy algorithms) XXVII

Хаффманы кодыг байгуулах

Дэвид Хаффманы боловсруулсан, угтваргүй кодыг хоёртын мод ашиглан угсрахдаа хамгийн цөөн удаа давтагдан орсон тэмдэгтүүдийг баруун зүүн зангилаа болгон сонгоно.

Хомхойлох алгоритм (Greedy algorithms) XXVIII

Хаффманы кодыг байгуулах



Хомхойлох алгоритм (Greedy algorithms) XXIX

Huffman(C)

```
1   n = |C|
2   Q = C
3   for i = 1 to n − 1
4         allocate a new node z
5         x=Extract-Min(Q)
6         y=Extract-Min(Q)
7         z.left = x
8         z.right = y
9         z.freq = x.freq + y.freq
10        Insert(Q, z)
11  return Extract-Min(Q, z)
```

Үүнд, Q min-priority queue, C цагаан толгой

Хомхойлох алгоритм (Greedy algorithms) XXX

Баталгаа

Хаффманы хомхойлох алгоритм нь зөв эсэхийг дараах лемм харуулна.

Lemma (хомхойлох шинжүүр)

C цагаан толгойн хувьд x болон y нь хамгийн бага давтамжтай тэмдэгтүүд байг. Тэгвэл x болон y нь ижил урттай, гагцхүү сүүлийн битээрээ ялгаатай байх C цагаан толгойн хувьд оновчтой угтваргүй код оршин байна.

Хомхойлох алгоритм (Greedy algorithms) XXXI

Баталгаа

Lemma (оновчтой дэд бүтцийн шинжүүр)

C цагаан толгойн хувьд x болон y нь хамгийн бага давтамжтай тэмдэгтүүд бөгөөд $z.freq = x.freq + y.freq$ байх шинэ z тэмдэгтийн хувьд $C' = (C - \{x, y\}) \cup \{z\}$ байг. T' нь C' цагаан толгойн хувьд оновчтой угтваргүй кодоор байгуулсан мод бөгөөд T нь x болон y зангилааг орлох z навч бүхий T' байг. Тэгвэл T нь C цагаан толгойн хувьд оновчтой угтваргүй код болно.

Хомхойлох алгоритм (Greedy algorithms) XXXII

Баталгаа

Theorem

Хаффманы код нь оновчтой угтваргүй кодыг байгуулна.

Теоремын баталгаа нь дээрх леммүүдээс илэрхий.

Хомхойлох алгоритм (Greedy algorithms) XXXIII

Offline caching

Компьютерт cache нь үндсэн санах ойгоос хурдан бөгөөд цомхон байдаг (CPU ашиглана). Энэ нь өгөгдлийг 32, 64 болон 128 байтын cache line гэх үүрүүдэд хуваарилах замаар хадгалдаг. Программ нь үүрүүдэд санах ойн дараалал үүсгэдэг. Үүр бүр хэд хэдэн хүсэлт хүлээн авдаг.

Хомхойлох алгоритм (Greedy algorithms) XXXIV

Offline caching

Cache хэмжээ нь k үүрээр хязгаарлагдах бөгөөд энэ нь эхний хүсэлт хүлээн авахаас өмнө хоослогдсон байна. Хүсэлт бүр cache-д орох 0 буюу 1 үүрийг эзэмшинэ. b үүр дэх хүсэлт нь дараах гурван тохиолдлын нэг нь байна:

1. b нь өмнөх хүсэлтийн улмаас cache-д байвал cache нь үл өөрчлөгдөнө.
2. b нь cache-д байхгүй бөгөөд cache нь хараахан дүүрээгүй бол (k үүрээс бага) b үүр нь cache-д орох бөгөөд cache нь өмнөх хүсэлтээс үүрийн тоогоор олон болно.
3. b нь cache-д ороогүй бөгөөд cache нь дүүрсэн бол cache дахь зарим үүр чөлөөлөгдөж, b cache-д орно.

Хомхойлох алгоритм (Greedy algorithms) XXXV

Offline caching

Зорилго: Сүүлийн 2 тохиолдлын (cache-д ороогүй) тоог багасгах нь ажиллагааг хурдасгана. Ингэхийн тулд дараах төлөвлөгөөний дагуу ажиллана: үүр болон хүсэлт тэдгээрийн cache-д орж буй тоог харгалзан үзэж, тухайн хүсэлт хэр ойрхон дуудагдаж байгаагаас хамаарч тухайн үүрийг чөлөөлөх эсэхийг шийднэ. Өөрөөр хэлбэл, ойрхон дуудагдах үүрийг үлдээж, хол давтамжтай дуудагдах үүрийг чөлөөлнө.

Хомхойлох алгоритм (Greedy algorithms) XXXVI

Оновчтой дэд бүтэц

$|C| \leq k$ байх $b_i, \ldots, b_n$ нийт үүрийн олонлог бөгөөд (C, i) бодлогын оновчтой шийд нь cache-д ороогүй тоог багасгах явдал.

S нь (C, i) бодлогын оновчтой шийд байг. C' нь b_i үүрэн дэх хүсэлтийг боловсруулсны дараах үүрийн олонлог, S' нь $(C', i+1)$ дараагийн дэд бодлогын шийдэл байг. Хэрэв b_i үүр дэх хүсэлт нь cache-д байвал $C' = C$, байхгүй бол $C' \neq C$.

Хомхойлох алгоритм (Greedy algorithms) XXXVII

Оновчтой дэд бүтэц

Тэгвэл S' нь $(C', i+1)$ бодлогын оновчтой шийд байх нь гарцаагүй. Үгүй бол S' шийдлээс cache-д ороогүй тоогоор цөөн S'' нь оновчтой шийд болно. S'' шийдлийг S шийдтэй нэгтгэвэл (C, i) бодлогын S шийдээс cache-д ороогүй тоогоор цөөн шийдэл нь гарна гэдгээс S нь оновчтой шийдэл болохгүй.

Хомхойлох алгоритм (Greedy algorithms) XXXVIII

Дахин дуудах шийдэл

b_i үүрэн дэх хүсэлтийг боловсруулсны дараа C үүрийн олонлогийн тохиргоог $R_{C,i}$ гэе.

- *Cache*-д өөрчлөлт ороогүй бол $R_{C,i} = \{C\}$
- Cache-д байхгүй бол b_i үүрэнд нэмэх $R_{C,i} = \{C \cup \{b_i\}\}$
- Cache дүүрсэн бөгөөд b_i үүрэнд хүсэлт нэмэх бол $R_{C,i} = \{(C - \{x\}) \cup \{b_i\} : x \in C\}$

Хомхойлох алгоритм (Greedy algorithms) XXXIX

Дахин дуудах шийдэл

$miss(C, i)$ нь (C, i) дэд бодлогын cache-д ороогүй хамгийн цөөн тохиолдол байг. Тэгвэл $miss(C, i)$ шийдийн хувьд дараах дахин дуудалт хийгдэнэ:

$$miss(C, i) = \begin{cases} 0, & i = n \ b_n \in C \\ 1, & i = n \ b_n \notin C \\ miss(C, i+1), & i < n \ b_i \in C \\ 1 + \min\{miss(C', i+1) : C' \in R_{C,i}\}, & i < n \ b_i \notin C \end{cases}$$

Хомхойлох алгоритм (Greedy algorithms) XL

Хомхойлох чанар

Offline cache бодлогыг доороос дээш буюу дээрээс доош аргыг memoization техниктэй хослуулан хэрэглэхэд их хэмжээний санах ой шаардагдана. Үүний оронд хомхойлох аргыг хол давтамжтай дуудагдах үүрийг чөлөөлөх зарчмаар ашиглах нь тохиромжтой.

Хомхойлох алгоритм (Greedy algorithms) XLI

Хомхойлох чанар

Theorem

C олонлогийн cache дүүрсэн гэж саная. b_i үүр дуудагдах үед $z = b_m$ үүр чөлөөлөгдөхөөр сонгогдлоо гэж бодъё. Тэгвэл (C, i) дэд бодлогын оновчтой шийдэл агуулагдах b_i үүрэн дэх хүсэлтийг гүйцэлдүүлэхийн тулд z үүрийг чөлөөлнө.

Энэ теоремоор хол зайтай дуудагдах хүсэлтийг чөлөөлөх нь хомхойлох чанар мөн гэдгийг илэрхийлж байна.

Нэгжийн (amortized) шинжилгээ I

Тойм

- Өгөгдлийн бүтцийн хувьд үйлдлийн дараалалд дүн шинжилгээ хийж
- Зарим үйлдлийн хувьд өртөг өндөр байж болох ч, үйлдэл бүрийн дундаж өртөгт бага эсэхийг шалгана.
- Үүнд магадлал тооцохгүй
- Дундаж өртгийг хамгийн муу тохиолдолд тооцно

Нэгжийн (amortized) шинжилгээ II

Тойм

Дараах 3 аргыг судлах болно:

- Нэгтгэсэн шинжилгээ (aggregate analysis)
- Санхүүгийн арга
- Боломжит арга

Нэгтгэсэн (aggregate) шинжилгээ

Олонлогт (stack) хийгдэх үйлдэл:

- $Push(S, x)$: нэг бүр нь $O(1)$ гэвэл n үйлдлийн хувьд $O(n)$
- $Pop(S)$: нэг бүр нь $O(1)$ бол n тохиолдолд $O(n)$ байна

Нэгжийн (amortized) шинжилгээ III

Олонлогийн хувьд

Multipop(S, k)

```
1  while not Stack-Empty(S) and k > 0
2        Pop(S)
3        k = k − 1
```

Дээрх тохиолдолд ажиллах хугацаа нь:

- *Pop* үйлдэл нь шугаман өсөлттэй
- *Push*/*Pop* бүрийн өртөг нь 1
- *while* үйлдлийн хувьд $min(s, k)$ удаа давтана
- Эндээс нийт өртөг нь $min(s, k)$ байна

Нэгжийн (amortized) шинжилгээ IV

Олонлогийн хувьд

n ширхэг *Push*, *Pop*, *Multipop* функцийн хувьд

- хамгийн их өртөг нь $O(n)$
- n үйлдэлтэй
- Хамгийн их өртөг нь $O(n^2)$ байна

Нэгжийн (amortized) шинжилгээ V

Дүгнэлт

- Объект бүрийн хувьд тухайн агшинд 1 удаа олонлогоос хасагдана
- *Push* нь n-ээс бага бол *Pop* мөн бага байна
- Иймээс нийт өртөг нь $O(n)$ байна
- n үйлдлийн хувьд үйлдэл бүрийн дундаж өртөг нь $O(1)$

Энэ аргыг нэгтгэсэн шинжилгээ гэдэг.

Нэгжийн (amortized) шинжилгээ VI

Хоёртын тоолуур

- $A[0 : k-1]$ гэсэн k бит хоёртын тоолуурын хувьд $A[0]$ нь хамгийн ач холбогдол багатай, харин $A[k-1]$ нь хамгийн ач холбогдол өндөртэй битүүд болно.
- 0-ээс өгсгөж тоолно
- Тоолуурын өртөг нь $\sum_{i=0}^{k-1} A[i] \cdot 2^i$
- Эхлэх утга нь 0 буюу $A[0 : k-1] = 0$

Нэгжийн (amortized) шинжилгээ VII

Хоёртын тоолуур

- add 1 (mod 2^k):

```
Increment(A, k)
1   i = 0
2   while i < k and A[i] == 1
3         A[i] = 0
4         i = i + 1
5   if i < k
6         A[i] = 1
```

Нэгжийн (amortized) шинжилгээ VIII

Хоёртын тоолуур

Жишээлбэл: $k = 3$ үед

тоолуурын утга	A(210)	өртөг
0	00$\underline{0}$	0
1	0$\underline{01}$	1
2	01$\underline{0}$	3
3	$\underline{011}$	4
4	10$\underline{0}$	7
5	1$\underline{01}$	8
6	11$\underline{0}$	10
7	$\underline{111}$	11
0	00$\underline{0}$	14

Ажиллах хугацаа нь $O(nk)$

Нэгжийн (amortized) шинжилгээ IX

Хоёртын тоолуур

Дүгнэлт

бит	хөрвөлт	тоо
0	цаг ямагт	n
1	1/2 удаа	$\lfloor n/2 \rfloor$
2	1/4 удаа	$\lfloor n/4 \rfloor$
	$\vdots$	
i	$1/2^i$ удаа	$\lfloor n/2^i \rfloor$
	$\vdots$	
$i \geq k$	огт хөрвөхгүй	0

Нэгжийн (amortized) шинжилгээ X

Хоёртын тоолуур

Нийт хөрвөх тоо нь:

$$\sum_{i=0}^{k-1} \left\lfloor \frac{n}{2^i} \right\rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} = n \cdot \frac{1}{1-1/2} = 2n$$

Ийнхүү ажиллах хугацаа нь $O(n)$,
үйлдэл бүрийн дундаж өртөг: $O(1)$

Нэгжийн (amortized) шинжилгээ XI

Санхүүгийн арга

Зарим үйлдэл харилцан биеэсээ өртгийн хувьд ялгаатай байж болно. Хэрэв нэгжийн өртөг нь бодит өртгөөс их бол хожим, нэгжийн өртгөөс давсан бодит өртөгт шингээхээр дараа зээл (credit) болгон хадгална.

Нэгжийн (amortized) шинжилгээ XII

Санхүүгийн арга

Нэгтгэх аргаас ялгаатай нь:

- Зарим үйлдэл харилцан адилгүй өртөгтэй байж болно
- Нэгтгэсэн шинжилгээгээр бүх үйлдэл ижил өртөгтэй байдаг

Нэгжийн (amortized) шинжилгээ XIII

Санхүүгийн арга

Зээл нь сөрөг утга авч болохгүй, эс бөгөөс дараалсан үйлдлийн нэгж өртөг нь бодит өртгийнхөө дээд хязгаарт хүрэхгүй.

Нэгжийн (amortized) шинжилгээ XIV

Санхүүгийн арга

c_i нь i-р үйлдлийн бодит өртөг, харин $\hat{c}_i$ нь i-р үйлдлийн нэгж өртөг. Нийт зээл нь:

$$\sum_{i=1}^{n} \hat{c}_i - \sum_{i=1}^{n} c_i \geq 0$$

Нэгжийн (amortized) шинжилгээ XV

Stack

үйлдэл	бодит өртөг	нэгж өртөг
Push	1	2
Pop	1	0
Multipop	$\min(k, s)$	0

Нэгжийн (amortized) шинжилгээ XVI

Төсөөлөл

Stack-д объект нэмэхэд 2 доллар төлнө:

- 1 долларыг Push үйлдэлд
- 1 долларыг Pop буюу Multipop үйлдэл
- Объект бүр 1 долларын өртөг бүхий зээл үүсгэх тул сөрөг утга авахгүй
- Нийт бодит өртгийн дээд хязгаар нь нийт нэгж өртөгтэй тэнцүү буюу $O(n)$ байна.

Нэгжийн (amortized) шинжилгээ XVII

Хоёртын тоолуур

0-ээс 1 болгоход 2 долларын өртөг тооцно:

- 1 болгоход 1 доллар
- буцаагаад 0 болгоход 1 долларыг урьдчилан төлнө
- тоолуурын хувьд 1 бүрд 1 долларын зээл тооцно

Иймд зээлийн өртөг нь тэгээс их буюу тэнцүү байна.

Нэгжийн (amortized) шинжилгээ XVIII

Хоёртын тоолуур

Нэмэх үйлдлийн нэгжийн өртөг:

- 0 болгон буцаах өртгийг зээлээр төлнө
- 1 битэд хамгийн ихдээ 1 оногдоно
- Иймд нэгж өртөг нь хамгийн ихдээ 2 доллартой тэнцэнэ
- n үйлдлийн хувьд нэгж өртөг нь $O(n)$ байна.

Нэгжийн (amortized) шинжилгээ XIX

Боломжит арга

- Санхүүгийн арга нь зээлийг объектод оноодог бол
- Боломжит арга нь өгөгдлийн бүтцийн хувьд бүхэлд нь оноодог
- Улмаар ирээдүйд хийгдэх өртөгт үйлдлийн төлбөрийг базаана
- Нэгжийн шинжилгээний хувьд хамгийн уян хатан арга байдаг

Нэгжийн (amortized) шинжилгээ XX

Боломжит арга

- D_i нь өгөгдлийн бүтцийн i-р үйлдэл
- D_0 нь үүсгэсэн өгөгдлийн бүтэц
- c_i нь i-р үйлдлийн өртөг
- $\hat{c}_i$ нь i-р үйлдлийн нэгжийн өртөг

Тэгвэл боломжит функц $\Phi : D_i \to \mathbb{R}$
$\Phi(D_i)$ нь D_i өгөгдлийн бүтцэд хамаарах боломжит функц.

$$\begin{aligned}\hat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &= c_i + \Delta\Phi(D_i).\end{aligned}$$

Нэгжийн (amortized) шинжилгээ XXI

Боломжит арга

Нийт нэгжийн өртөг нь:

$$\sum_{i=1}^{n} \hat{c}_i =$$

$$= \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1})) =$$

$$= \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0).$$

Нэгжийн (amortized) шинжилгээ XXII

Эгэх дараалал (stack)

Φ нь эгэх дарааллын объектууд, D_0 нь хоосон эгэх дараалал бол $\Phi(D_0) = 0$ байна.
Эгэх дараалал дахь объектын тоо нь тэгээс их байна гэвэл дурын i тооны хувьд $\Phi(D_i) \geq 0 = \Phi(D_0)$ байна.

үйлдэл	бодит өртөг	$\Delta\Phi$	нэгж өртөг
Push	1	$(s+1) - s = 1$	$1 + 1 = 2$
Pop	1	$(s-1) - s = -1$	$1 - 1 = 0$
Multipop	$k' = \min(k, s)$	$(s - k') - s = -k'$	$k' - k' = 0$

n дарааллын хувьд нэгжийн өртөг нь $O(n)$ байна.

Нэгжийн (amortized) шинжилгээ XXIII

Динамик хүснэгт

- Хэчнээн тооны объект хадгалагдахыг нь үл мэдэх hash хүснэгт өгөгдсөн байг.
- Хэрэв анх нөөцөлсөн хэмжээ дүүрвэл, өмнөхөөс илүү том хүснэгт шинээр үүсгэж, дахин бүх объектыг хуулна.

Нэгжийн (amortized) шинжилгээ XXIV

Динамик хүснэгт

- Үйлдэл бүрд $O(1)$ нэгж хугацаа зарцуулна
- Хоосон зай нь анх үүсгэсэн зайнаас ямагт бага байна
- Ачаалал (load factor): $\alpha = num/size$, үүнд *num* нь хадгалагдсан объектууд, *size* анх хуваарилсан хэмжээ.
- Хэрэв $size = 0$ бол $num = 0$ байх ба $\alpha = 1$ байна хэмээн үзнэ.

Нэгжийн (amortized) шинжилгээ XXV

Хүснэгтийг өргөтгөх

- Хэрэв хүснэгт дүүрвэл, хэмжээг нь 2 дахин ихэсгээд, бүх өгөгдлийг дахин байршуулна
- $\alpha \geq 1/2$ гэсэн хязгаарлалт тогтооно

Нэгжийн (amortized) шинжилгээ XXVI

Ажиллах хугацаа

c_i нь i-р үйлдлийн бодит өртөг гэвэл

- Хэрэв хүснэгт дүүрээгүй бол $c_i = 1$.
- Дүүрсэн бол i-р өгөгдлийг нэмэхийн тулд $i - 1$ өгөгдлийг бүгдийг дахин хуулах тул өртөг нь $c_i = i$ болно.

Нэгжийн (amortized) шинжилгээ XXVII

Динамик хүснэгт

$$c_i = \begin{cases} i, & \text{хэрэв } i - 12\text{-ын зэрэг} \\ 1, & \text{бусад тохиолдолд} \end{cases}$$

Нийт өртөг:

$$\sum_{i=1}^{n} c_i \leq n + \sum_{j=0}^{\lfloor \lg n \rfloor} 2^j = n + \frac{2^{\lfloor \lg n \rfloor + 1} - 1}{2 - 1} < n + 2n = 3n$$

Ийнхүү нэгтгэх шинжилгээгээр үйлдэл бүрийн нэгжийн өртөг нь 3 болов.

Нэгжийн (amortized) шинжилгээ XXVIII

Санхүүгийн арга

x элементийн нэмэх үйлдэлд \$3 зарцуулна:

- Нэмэх үйлдэлд \$1 өртөг
- Ирээдүйн зөөх үйлдэлд \$1 өртөг
- Бусад элементийг зөөх үйлдэлд \$1 өртөг тус тус зарцуулна

Тэгвэл $size = num = m$ үед хүснэгт тэлнэ гэвэл $size = 2m$ ба $num = m$ болно. Дараагийн тэлэлтэд $size = num = 2m$ болно.

Нэгжийн (amortized) шинжилгээ XXIX

Санхүүгийн арга

- Тэлэлтэд бүх зээл зарцуулагдсан гэж саная
- Дахин m элемент нэмсний дараа дахин тэлнэ
- Элементийг дахин шилжүүлэх бүрд \$1, шинээр нэмэх бүрд \$1 өртөг орсон
- Дараагийн тэлэлтэд $2m$ элемент шилжүүлэхэд зориулж яг \$$2m$ өртөг бүхий зээл үлдэнэ.

Хялбар Графын Алгоритм I

Графыг илэрхийлэх

$$G = (V, E)$$

Үүнд: V–оройнууд, E–ирмэгүүд.

Хялбар Графын Алгоритм II

Графыг илэрхийлэх

G граф нь чиглэлт эсвэл чиглэлгүй байж болох бөгөөд алгоритмд:

1. хөрш оройн жагсаалт
2. хөрш оройн матриц

ашиглан илэрхийлдэг.

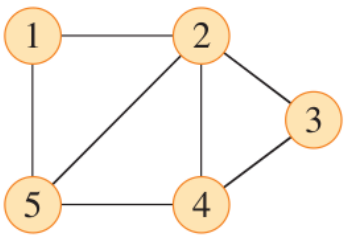
Хялбар Графын Алгоритм III

Графыг илэрхийлэх

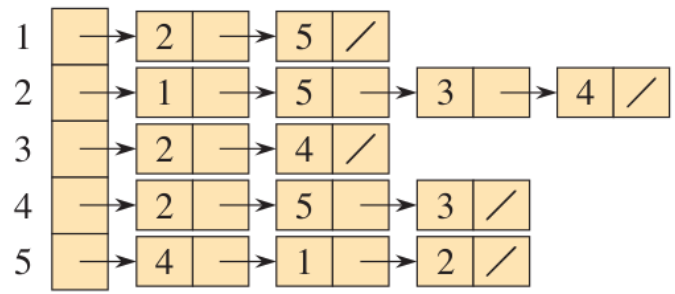
Ажиллах хугацааг тооцохдоо: $O(V + E)$ буюу $O(|V| + |E|)$

Хялбар Графын Алгоритм IV

Графыг илэрхийлэх



(a)



(b)

	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

(c)

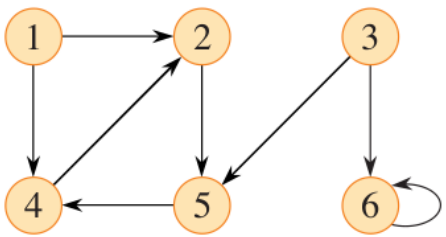
(a) 5 орой болон 7 ирмэгтэй, чиглэлт биш G граф

(b) G графын хөрш оройн жагсаалт

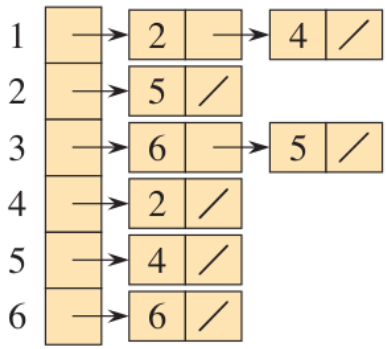
(c) G графын хөрш оройн матриц

Хялбар Графын Алгоритм V

Графыг илэрхийлэх



(a)



(b)

	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

(c)

(a) 6 орой болон 8 ирмэгтэй чиглэлт G граф

(b) G графын хөрш оройн жагсаалт

(c) G графын хөрш оройн матриц

Хялбар Графын Алгоритм VI

Графыг жагсаалтаар илэрхийлэх

V орой бүр нь хөрш оройн жагсаалтыг агуулна. Өөрөөр хэлбэл, $u \in V$ орой бүр нь $(u, v) \in E$ байх ижил ирмэгээр холбогдох v оройг $Adj[u]$ жагсаалтад агуулна.

Хялбар Графын Алгоритм VII

Графыг жагсаалтаар илэрхийлэх

Хийсвэр кодод хөрш оройн жагсаалтыг: $G.Adj[u]$ хэмээн тэмдэглэнэ.

Хялбар Графын Алгоритм VIII

Графыг жагсаалтаар илэрхийлэх

Хэрэв ирмэгт жин өгөгдсөн бол: $w : E \to \mathbb{R}$
Зай: $\Theta(V + E)$
Хугацаа: u оройн хөрш оройн жагсаалт $u : \Theta(degree(u))$
Хугацаа: $(u, v) \in E : O(degree(u))$

Хялбар Графын Алгоритм IX

Графыг матрицаар илэрхийлэх

Хэрэв $G = (V, E)$ граф нь $1, 2, \ldots, |V|$ тооны орой агуулсан бол түүнийг хөрш оройн матрицад $|V| \times |V|$ хэмжээтэй,

$$a_{ij} = \begin{cases} 1, & \text{хэрэв } (i, j) \in E \\ 0, & \text{бусад тохиолдолд} \end{cases}$$

байх $A = (a_{ij})$ хэмээн илэрхийлнэ.

Хялбар Графын Алгоритм X

Графыг матрицаар илэрхийлэх

Зай: $\Theta(V^2)$

Хугацаа: u оройн хөрш оройн жагсаалт $\Theta(V)$

Хугацаа: $(u, v) \in E : \Theta(1)$

Жинг жагсаалттай адилаар илэрхийлж болно

Хялбар Графын Алгоритм XI

Графын шинж чанарыг илэрхийлэх

- Хэрэв v орой нь d гэсэн шинж чанартай (attribute) бол $v.d$
- Хос оройтой ирмэгийн хувьд шинж чанарыг тодорхойлбол $(u, v).f$

Хялбар Графын Алгоритм XII

Графын шинж чанарыг кодод ашиглах

- Программын кодод $d[1 : |V|]$ нэмэлт жагсаалт үүсгэж ашиглана
- Оройн $u.d$ шинж чанарыг $d[u]$ хэмээн бичнэ

Хялбар Графын Алгоритм XIII

Гүехэн хайлт (Breadth-first search)

Өгөгдсөн $G = (V, E)$ графын хувьд s оройгоос v орой уруу хүрч болох боломжит бүх ирмэгүүдээр явахдаа хамгийн цөөн тооны ирмэгийг агуулсан богино замыг тогтоох бөгөөд уг алгоритм нь чиглэлт болон чиглэлт биш граф дээр ашиглагддаг.

Хялбар Графын Алгоритм XIV

Гүехэн хайлт (BFS)

Эхлэх орой: $s \in V$

- $v.d$ нь s оройгоос $v \in V$ байх v орой хүртэлх зай
- $v.\pi$ нь s оройгоос v орой хүртэлх богино зам
- (u, v) нь $s \rightsquigarrow v$ богино замын төгсгөлийн ирмэг
- Богино зам нь (u, v) ирмэг агуулсан бол $v.\pi = u$

Хялбар Графын Алгоритм XV

Гүехэн хайлт (BFS)

- s оройгоос эхэлж 1 ирмэг зайд орших бүх оройнуудад хүрнэ
- улмаар тэдгээр оройнуудаас цааш дахин 1 ирмэг зайд орших оройнуудад хүрнэ
- гэх мэт

Хялбар Графын Алгоритм XVI

Гүехэн хайлт (BFS)

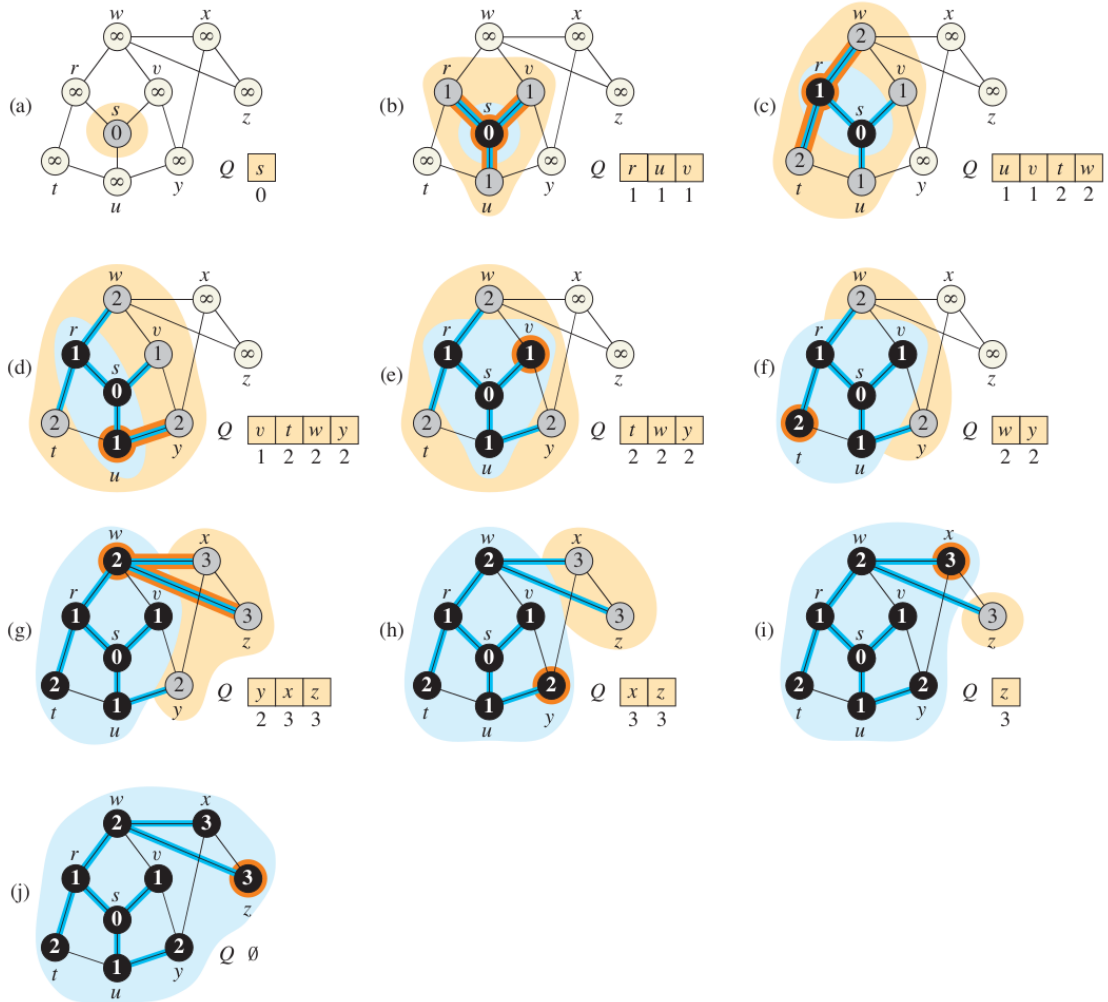
Хялбар Графын Алгоритм XVII

BFS(V, E, s)

```
 1  for each vertex u ∈ V − {s}
 2        u.d = ∞
 3        u.π = NIL
 4  s.d = 0
 5  Q = ∅
 6  Enqueue(Q, s)
 7  while Q ≠ ∅
 8        u = Dequeue(Q)
 9        for each vertex v in G.Adj[u]
10              if v.d == ∞
11                    v.d = u.d + 1
12                    v.π = u
13                    Enqueue(Q, v)
```

Хялбар Графын Алгоритм XVIII

Гүехэн хайлт (BFS)



Хялбар Графын Алгоритм XIX

Гүехэн хайлт (BFS)

BFS нь бүх оройнуудад хүрэхгүй байж болно.

Хялбар Графын Алгоритм XX

Гүехэн хайлт (BFS)

Ажиллах хугацаа: $O(V + E)$

- Орой бүрд хамгийн ихдээ нэг удаа хүрэх тул $O(V)$
- Хэрэв чиглэлт граф бол ирмэг бүрийг хамгийн ихдээ 1 удаа дайрна
- Хэрэв чиглэлт биш граф бол ирмэг бүрийг хамгийн ихдээ 2 удаа дайрна

Хялбар Графын Алгоритм XXI

Гүн хайлт (Depth-first search)

Гүн хайлт нь v оройн сүүлд хайхад үлдсэн ирмэгүүдийг үргэлжлүүлэн хайж дуусмагц үлдэх оройнуудыг буцаж самнана. Ийнхүү эхэлсэн оройгоос хүрч болох бүх оройнуудыг самнаж дуусгана. Хэрэв самнагдаагүй орой үлдвэл тэдгээрийг шинээр самнаж эхэлнэ.

Хялбар Графын Алгоритм XXII

Гүн хайлт (DFS)

DFS(G)

```
1  for each vertex u ∈ G.V
2        u.color = White
3        u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6        if u.color == White
7              DFS − Visit(G, u)
```

Хялбар Графын Алгоритм XXIII

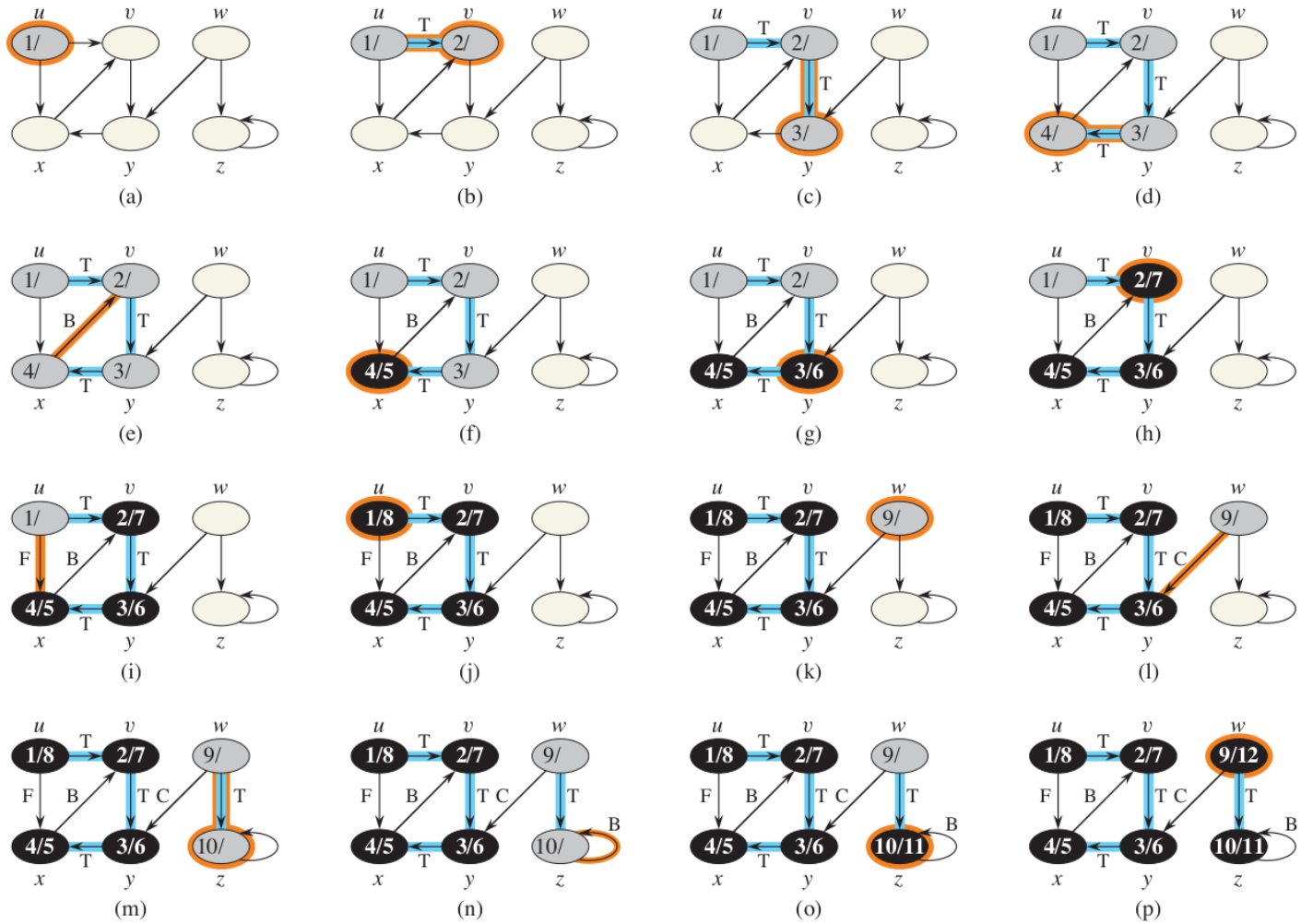
Гүн хайлт (DFS)

DFS-Visit(G,u)

```
1   time = time + 1
2   u.d = time
3   u.color = Gray
4   for each vertex v in G.Adj[u]
5         if v.color == White
6               v.π = u
7               DFS − Visit(G, u)
8   time = time + 1
9   u.f = time
10  u.color = Black
```

Хялбар Графын Алгоритм XXIV

Гүн хайлт (DFS)



Хялбар Графын Алгоритм XXV

Гүн хайлт (DFS)

Ажиллах хугацаа: $\Theta(V + E)$

Бүх орой болон ирмэгээр дайран өнгөрөх тул O биш Θ байна.

Хялбар Графын Алгоритм XXVI

Гүн хайлтын онцлог чанар

- Гүн хайлтын граф нь ой мод хэлбэрийг үүсгэдэг
- Самналт нь хаалт нээх, хаах бүтцийг үүсгэдэг

Хялбар Графын Алгоритм XXVII

Гүн хайлтын шинж чанар–Хаалт

Theorem (хаалтын тухай теорем)

Бүх u, v оройнуудын хувьд дараах тохиолдлын аль нэг нь биелнэ:

1. $u.d < u.f < v.f$ *буюу* $v.d < v.f < u.d < u.f$ *(өөрөөр хэлбэл,* $[u.d, u.f]$ *болон* $[v.d, v.f]$ *завсрууд нь огтлолцоогүй) бөгөөд* u *болон* v *оройнууд нь хоорондоо холбогдоогүй*
2. $u.d < v.d < v.f < u.f$ *ба* v *нь* u *оройн өмнөх орой*
3. $v.d < u.d < u.f < v.f$ *ба* u *нь* v *оройн өмнөх орой*

Хялбар Графын Алгоритм XXVIII

Гүн хайлтын жишээ–Хаалт

Зөв ( ) [ ] ([ ]) [( )]

Буруу ([ )) [( ])