Алгоритмын зохиомж бие даалт 2

Г. Алтай
Оюутны код: B222270028
Хичээл: Алгоритмын зохиомж (F.CSM301)

December 4, 2025

Даалгаврын зорилго

Энэхүү бие даалтаар оновчлолын алгоритмуудын (**Greedy** болон **Dynamic Programming**) үндсэн санааг ойлгож, бичвэрийг жигдлэх даалгаврыг гүйцэтгэх болно.

Агуулга

1 Оршил

Энэхүү бие даалтын зорилго нь бичвэрийг жигдлэх асуудлыг хоёр өөр аргаар — **Greedy** ба **Dynamic Programming** (DP) — шийдвэрлэж, алгоритмуудын үр ашиг, оновчтой байдал, санах ойн хэрэглээг харьцуулан дүгнэхэд оршино. Бичвэр жигдлэх (text justification) нь өгөгдсөн үгийн дарааллыг мөр бүрийн өргөнд тааруулан, үсэг хоорондын зайг зөв хуваарилах классик оновчлолын асуудал юм.

2 Асуудлын тодорхойлолт

Өгөгдсөн:

- Үгсийн дараалал $W = [w_1, w_2, \ldots, w_n]$
- Мөрийн дээд өргөн L

Зорилго нь өгөгдсөн үгсийг мөрүүдэд хувааж, мөр бүрийн сул зайг (*slack*) оновчтой түгээж, бичвэрийг жигд хэлбэртэй болгох явдал юм.

Асуудлыг хоёр аргаар шийдэв:

1. **Greedy** — мөрөнд аль болох олон үг багтааж, орхигдсон сул зайг тухайн мөрөнд л засварлана.
2. **DP** — бүх боломжит мөрлөлтүүдийн өртгийг тооцож, нийт хамгийн бага өртөгтэй хуваалтыг олно.

3 Greedy алгоритм

3.1 Алгоритмын ерөнхий ойлголт

Greedy буюу шунахай алгоритм нь бичвэрийг жигдлэх асуудлыг локал түвшний хамгийн оновчтой сонголтуудын нийлбэр нь нийт оновчтой үр дүнг өгнө гэсэн таамаглал дээр үндэслэн шийддэг. Өөрөөр хэлбэл алгоритм нь өгөгдсөн үгийн дарааллаас мөр бүрт багтаж болох хамгийн олон үгийг дарааллын эхнээс нь эхлэн сонгож, тухайн мөрийн сул зайг тухайн мөрөнд нь л шийдвэрлээд дараагийн мөрөнд шилждэг. Мөрлөлт бүрийг тухайн мөчид хамгийн боломжит хувилбараар сонгож байгаа тул глобал оновчлол хийхгүй.

Энэ аргын гол санаа нь "*нэг мөрийн хэмжээнд хамгийн сайн сонголт хийнэ, бүхэл бичвэрийг дахин бодохгүй*" гэх шийдэл юм. Үүний улмаас алгоритм нь маш хурдтай бөгөөд өгөгдсөн текстийн хэмжээнээс үл хамааран $O(n)$ хугацааны төвөгшилтэй ажиллана. Гэхдээ локал хамгийн сайн шийдлүүдийн нийлбэр нь нийт хамгийн сайн хуваалтыг заавал үүсгэнэ гэсэн баталгаа байхгүй тул Greedy арга нь түгээмэл тохиолдолд боломжийн жигдлэл өгдөг ч хамгийн чанартай, оновчтой жигдлэлтийг үргэлж баталгаажуулдаггүй.

Greedy алгоритмийг текст жигдлэх, үгийн урт тэнцүүлэх, хуудасны хэвлэлийн автомат формат зэрэг практик хэрэглээнд өргөн ашигладаг. Учир нь эдгээр системүүдийн ихэнх нь бодит цагийн ажиллагаатай тул удаан ажиллах DP-тэй харьцуулахад Greedy илүү тохиромжтой байдаг.

3.2 Алгоритмын ажиллагааны нарийвчилсан тайлбар

Алгоритм дараах үндсэн дарааллыг гүйцэтгэнэ:

1. Эхлэх индексийг i гэж авна.
2. i–ээс эхлэн мөрийн өргөнд багтах хүртэл дараагийн үгийг нэмж шалгана.
3. Хэрэв дараагийн үг багтахгүй бол i–ээс $j-1$ хүртэлх үгсээр нэг мөрийг бүрдүүлнэ.
4. Мөрийг бүрдүүлэхдээ сул зайг нийт зайгаар хуваан, зайг үг хооронд тэнцүү тараана.
5. Сүүлийн мөрийн хувьд зүүнээс баруун тийш энгийн стандарт формат хэрэглэнэ.
6. Дараагийн мөрийг j индекстэйгээр эхлүүлж, ажлыг дуусах хүртэл үргэлжлүүлнэ.

Энэ аргын үе шат бүр нь зөвхөн дараагийн боломжит үйлдлийн тухай шийдвэр гаргадаг тул санах ойн хэрэглээ бага ($O(1)$), өгөгдлийн хэмжээ нэмэгдэхэд цагийн өсөлт маш бага байдаг.

3.3 Давуу тал

- Маш хурдан ($O(n)$)
- Ойролцоо жигдлэл хангана
- Хэрэгжүүлэхэд хамгийн энгийн

3.4 Сул тал

- Нийт бичвэрийн хэмжээнд глобал оновчлол хийдэггүй
- Мөрлөлт зарим үед хамгийн муу хувилбарыг сонгож болдог

3.5 Greedy алгоритмын псевдокод

Доорх псевдо-код нь Greedy текст жигдлэх алгоритмын логикийг товч, ойлгомжтой илэрхийлнэ:

```
procedure JUSTIFY_GREEDY(words, maxWidth):
    result = empty list
    i = 0

    while i < length(words):
        lineLen = length(words[i])
        j = i + 1

        while j < length(words) and
              lineLen + 1 + length(words[j])  maxWidth:
            lineLen = lineLen + 1 + length(words[j])
            j = j + 1

        numWords = j - i

        if j == length(words) or numWords == 1:
```

```
            line = join(words[i..j-1], "␣")
            while length(line) < maxWidth:
                line = line + "␣"
        else:
            totalSpaces = maxWidth - sumLength(words[i..j-1])
            gaps = numWords - 1
            space = totalSpaces div gaps
            extra = totalSpaces mod gaps

            line = ""
            for k = i to j-2:
                line = line + words[k] + repeat("␣", space)
                if extra > 0:
                    line = line + "␣"
                    extra = extra - 1
            line = line + words[j-1]

        append(result, line)
        i = j

    return result
```

3.6 Хугацааны болон санах ойн төвөгшил

Greedy алгоритм нь өгөгдсөн үгийг зүүнээс баруун руу нэг удаа шалган, тухайн мөрөнд багтах эсэхийг тооцоолж дараагийн мөр лүү шилждэг. Энэ үед ямар ч дахин тооцолт, буцаагч давталт (backtracking) хийх шаардлагагүй. Иймээс алгоритмын ажлын хэмжээ үгийн тоотой шууд пропорционал байна.

$$T(n) = O(n)$$

Энэ нь өгөгдлийн хэмжээ өсөх тусам алгоритм зөвхөн нэг л удаагийн шугаман скан хийхээс хэтрэхгүй гэсэн үг юм. Мөн мөр бүрийн хооронд зай тооцоолох үйлдэл нь тогтмол хугацаанд хийгддэг тул нийт гүйцэтгэлийн хурдад нөлөөлөхүйц нэмэлт өсөлт үүсдэггүй.

Санах ойн хувьд Greedy алгоритм нь зөвхөн одоогийн мөрийн үгс, тооцоолсон урт, эсвэл нэмэлт завсрын хувьсагчдыг хадгалах шаардлагатай. Эдгээр хувьсагчид нь оролтын хэмжээнээс үл хамаарч тогтмол тоотой байдаг. Аль ч үед алгоритм оролтын бүх үгийг хадгалах массивыг өөрчилдөггүй бөгөөд нэмэлт массив, динамик бүтэц шаарддаггүй.

$$S(n) = O(1)$$

Иймээс Greedy алгоритм нь маш бага санах ой ашиглаж, том хэмжээний текстэн дээр ч үр ашигтай ажиллах боломжтой.

4 Dynamic Programming алгоритм

4.1 Алгоритмын санаа

4.2 Dynamic Programming алгоритм

Dynamic Programming (DP) арга нь текст жигдлэх асуудлыг глобал оновчлолын хувьд шийддэг. Өөрөөр хэлбэл, алгоритм нь мөр бүрийг хэрхэн хуваах боломжтой бүх хувилбаруудыг тооцож, нийт өртөг (*badness*) хамгийн бага болох хуваалтыг олдог.

DP нь локал хамгийн сайн шийдлүүдийг дараалалд зүгээр нэмэх бус, бүх мөрлөлтийн хувилбаруудыг харгалзан нийт системийн оновчтой шийдлийг тодорхойлдог. Ингэснээр мөр бүрийн сул зайг хамгийн оновчтой хуваарилж, бүхэл бичвэрийн жигдлэлт хамгийн сайн болно.

Энэ алгоритмд "*badness*" буюу мөрийн сул зайны өртөгийн функц хэрэглэгддэг. Жишээлбэл:

$$badness(slack) = slack^3$$

Энэ нь том сул зайтай мөрт өртөг их, жижиг сул зайтай мөрт өртөг бага байхыг баталгаажуулна. DP алгоритм нь дараах үндсэн зарчмаар ажиллана:

1. Бүх боломжит мөрийн эхлэл, төгсгөлийн индексуудын өртгийг (*cost[i][j]*) тооцох
2. Эцсийн индексээс буцаан шугаман байдлаар dp массивыг бөглөн, хамгийн бага нийт өртөгтэй хуваалтыг олох
3. Өөрт тохирсон мөрүүдийг дарааллаар гаргаж, зайг тэнцүү тараан бичвэрийг жигдлэх

DP алгоритм нь Greedy-тэй харьцуулахад илүү удаан ажилладаг ($O(n^2)$), мөн санах ой их шаардагддаг ($O(n^2)$) боловч глобал оновчлолын баталгаатай. Иймээс бичвэрийн чанарыг нэн тэргүүнд тавих шаардлагатай үед DP илүү тохиромжтой байдаг.

4.3 Давуу тал

- Глобал оновчлол хийдэг — хамгийн сайн боломжит бичвэр гарна
- Олон төрлийн жигдлэлийн функц дээр ажиллуулж болох уян хатан бүтэцтэй

4.4 Сул тал

- Greedy-ээс мэдэгдэхүйц удаан
- Санах ойн хэрэглээ өндөр

4.5 DP алгоритмын pseudocode

Доорх pseudocode нь Greedy текст жигдлэх алгоритмын логикийг товч, ойлгомжтой илэрхийлнэ:

```
procedure JUSTIFY_DP(words, maxWidth):
    n = length(words)
    lens = array of length n
    for i = 0 to n-1:
        lens[i] = length(words[i])

    INF = large number
    cost = array[n][n] filled with INF

    for i = 0 to n-1:
        length = lens[i]
        for j = i to n-1:
            if j > i:
                length = length + 1 + lens[j]
            if length   maxWidth:
                slack = maxWidth - length
                if j == n-1:
                    cost[i][j] = 0
                else:
                    cost[i][j] = slack^3
            else:
                break

    dp = array[n+1] filled with INF
    nextBreak = array[n]
    dp[n] = 0

    for i = n-1 downto 0:
        for j = i to n-1:
            if cost[i][j] == INF:
                break
            val = cost[i][j] + dp[j+1]
            if val < dp[i]:
                dp[i] = val
                nextBreak[i] = j+1

    result = empty list
    i = 0
    while i < n:
        j = nextBreak[i]
        lineWords = words[i..j-1]
        totalSpaces = maxWidth - sumLength(lineWords)
        gaps = length(lineWords) - 1

        line = ""
        if j == n or gaps == 0:
            line = join(lineWords, "␣")
            while length(line) < maxWidth:
                line = line + "␣"
        else:
            space = totalSpaces div gaps
            extra = totalSpaces mod gaps
            for k = 0 to length(lineWords)-2:
                line = line + lineWords[k] + repeat("␣", space)
                if extra > 0:
                    line = line + "␣"
                    extra = extra - 1
```

```
            line = line + lineWords[length(lineWords)-1]

        append(result, line)
        i = j

    return result
```

4.6 Хугацааны болон санах ойн төвөгшил

Dynamic Programming (DP) алгоритм нь бүх боломжит мөрлөлтийн хувилбаруудыг харгалзан нийт өртөгийг тооцдог тул Greedy-ээс илүү нарийн төвөгтэй ажиллагаатай. Өөрөөр хэлбэл, алгоритм нь эхний үгнээс сүүлийн үг хүртэлх бүх боломжит төгсгөлийг шалгаж, дараагийн мөрүүдийн өртөгийг тооцоолно.

$$T(n) = O(n^2)$$

Энэ нь алгоритм n үгтэй текстэд $n \times n$ хэмжээтэй өртгийн хүснэгтийг бөглөн, дараа нь буцаан dp массивыг нэг бүрчлэн тооцдог гэсэн үг юм. Иймээс өгөгдлийн хэмжээ нэмэгдэх тусам хугацааны өсөлт квадрат хэлбэртэй гарна.

Санах ойн хувьд DP нь бүх боломжит мөрүүдийн өртгийг хадгалах шаардлагатай байдаг. Өөрөөр хэлбэл:

$$S(n) = O(n^2)$$

`cost[i][j]` хүснэгт нь i–ээс j хүртэлх үгсээр мөрийг хэрхэн бүрдүүлэхэд ямар өртөг гарч болохыг хадгална. Мөн dp массив болон nextBreak массив нь n хэмжээтэй. Энэ бүх бүтэц нийлээд санах ойг квадратик хэлбэрээр шаарддаг.

Ийм учраас DP алгоритм нь маш оновчтой бичвэр гаргах чадвартай боловч том хэмжээний текстэнд Greedy-тэй харьцуулахад илүү их хугацаа болон санах ой шаарддаг. Тиймээс практикт том хэмжээний текстийг хурдан боловсруулах шаардлагатай үед Greedy-ийг, харин чанарт илүү төвлөрөх үед DP-г сонгодог.

5 Харьцуулалт

Арга	Хугацаа	Санах ой
Greedy	$O(n)$	$O(1)$
DP	$O(n^2)$	$O(n^2)$

5.1 Дүгнэлт

Greedy нь хурдан, энгийн боловч зарим тохиолдолд тэнцвэргүй жигдлэлттэй байдаг. Харин DP нь хамгийн чанартай, глобал оновчтой үр дүн өгдөг ч их нөөц хэрэглэдэг.

6 Аль алгоритм энэ даалгаварт илүү тохиромжтой байсан бэ?

Энэ даалгаврын зорилго нь:

- Оновчлолын алгоритмуудын ялгааг ойлгох
- Жигдлэлтийн чанарын ялгааг харах

Учир нь бичвэрийн жигдлэлтийн чанар хамгийн чухал тул **Dynamic Programming** арга нь илүү тохиромжтой алгоритм гэж үзэв.

DP алгоритм бичвэрийг бүхэлд нь харж хамгийн сайн хэлбэрийг сонгодог тул жинхэнэ оновчтой гаргалгаа авах боломжтой. Greedy нь практикт хурдан боловч чанарын хувьд DP-с дутуу.

7 Дүгнэлт

Энэхүү даалгавраар Greedy болон Dynamic Programming аргуудын ялгаа тодорхой харагдлаа. Greedy нь хурд, энгийн байдлаараа давуу боловч DP нь илүү төгс, оновчтой бичвэр гаргадаг. Бичвэрийн жигдлэлтийн чанарыг нэн тэргүүнд тавих тохиолдолд DP арга давуу, харин их хэмжээний урт бичвэрийг хурдтай боловсруулах шаардлагатай үед Greedy илүү тохиромжтой юм.