Шинжлэх Ухаан Технологийн Их Сургууль
Мэдээлэл, Холбооны Технологийн Сургууль

Бие Даалтын Тайлан

F.CSM301 Алгоритмын шинжилгээ ба зохиомж

Шалгасан багш: Д. Батмөнх
Гүйцэтгэсэн оюутан: О.Дэлгэрмаа /B232270024/

Улаанбаатар хот
2025 он

Гарчиг

Зургийн жагсаалт

1. Удиртгал

1.1 Программын танилцуулга, зорилго

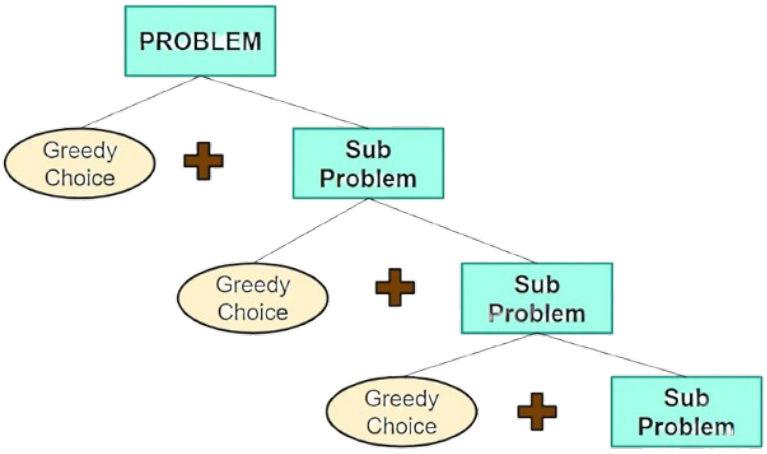
Энэ бие даалтаар гарнаас өгөгдсөн текстийг тогтоосон мөрний өргөн (line width) дотор багтаах зорилготойгоор хуваарилж, мөр бүрийг justified ашиглан тэгшлэн, цэгцтэй текстийг өгөх ажлыг гүйцэтгэдэг. Энэхүү асуудал нь текст боловсруулалт, хэвлэлийн систем зэрэгт чухал бөгөөд шийдлийн оновчтой байдлыг хангахын тулд хоёр өөр алгоритмын арга зүйг харьцуулан ашигласан.

Үндсэн зорилго: Текстийн үгсийг W өргөнтэй мөрүүдэд хуваарилах, ингэхдээ мөрүүдийн төгсгөлд үүсэх хоосон зайг багасгах.

Нэмэлт боломж: Монгол болон Англи хэлний үгсийг ялган таньж, шаардлагатай үед үеэр таслах (*hyphenation*) боломжийг нэвтрүүлсэн.

1.2 Greedy Algorithm

Greedy Algorithm нь optimization problems-ийг шийдэхэд өргөн ашиглагддаг энгийн бөгөөд ойлгомжтой арга юм. Энэхүү алгоритм нь асуудлыг үе шаттайгаар шийдэж, алхам бүрт тухайн агшинд хамгийн оновчтой гэж үзсэн сонголтыг (locally optimal choice) хийдэг. Нэгэнт сонголт хийгдсэн бол буцаан өөрчлөхгүй бөгөөд энэ нь алгоритмыг хурдан, хэрэгжүүлэхэд хялбар болгодог.



Зураг 1.1: Greedy алгоритм нь алхам бүрд сонголт хийж, дэд шийдэлд нэмдэг(Эх сурвалж: Manali Teke, "Understanding Greedy Algorithms)

Greedy Algorithm-ийн үндсэн шинж чанарууд

1. Greedy Choice Property

   Хэрэв асуудлыг шийдэх явцад алхам бүрд хийсэн locally optimal choice-уудын үр дүнд

globally optimal solution-д хүрэх боломжтой бол уг асуудал greedy choice property-тэй гэж үзнэ.

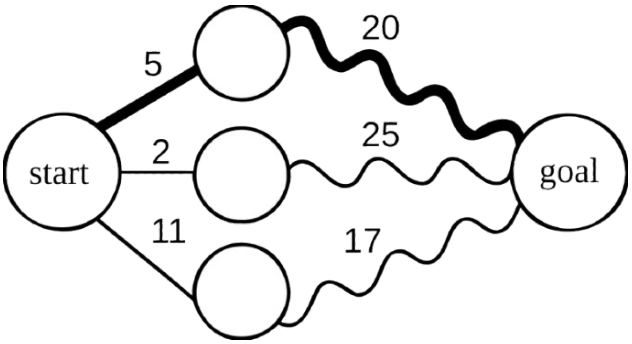
2. Optimal Substructure
   Хэрэв асуудлын global optimal solution нь түүний бүх дэд асуудлуудын (subproblems) optimal solution-уудыг агуулж байвал уг асуудал optimal substructure шинж чанартай байна.

Text justification-ийн асуудалд, Greedy алгоритм нь мөр бүрд багтах үгсийг дараалалтайгаар нэмэх замаар мөрийг бүрдүүлдэг. Хэрэв дараагийн үг мөрт багтахгүй бол шаардлагатай үед үгийг таслан (hyphenation) одоогийн мөрт байрлуулж, үлдсэн хэсгийг дараагийн мөрт оруулна.

Жишээ нь, хэрэв мөрийн өргөн 20 тэмдэгт байвал "Programming"гэдэг үгийг "Program-"ба "ming"гэж тасалж хоёр мөрт байрлуулж болно. Ингэснээр мөр бүр тэнцвэртэй болж, хоосон зай багасна. Greedy алгоритм нь энгийн, хурдан бөгөөд ихэвчлэн ойролцоо оновчтой шийдлийг гаргадаг ч зарим тохиолдолд дэлхийд хамгийн оновчтой шийдлийг гаргахгүй байж болдог.

1.3 Dynamic Programming

Dynamic Programming (DP) нь optimization problems-ийг шийдэх алгоритмын арга бөгөөд асуудлыг хоорондоо давхцаж буй (overlapping) жижиг дэд асуудлууд (subproblems) болгон задлаж, тэдгээрийн шийдлийг хадгалснаар давхардсан тооцооллыг зайлсхийдэг. **Dynamic**



Зураг 1.2: Dynamic Programming (DP) алгоритм (Эх сурвалж: Dynamic programming — Wikipedia)

Programming бодлогыг дараах дөрвөн үндсэн алхмаар шийддэг.

1. Subproblem тодорхойлох Энэ нь хамгийн чухал алхам бөгөөд нэг subproblem-ийн шийдлийг илэрхийлэх хувьсагчийг тодорхойлно.

2. Recurrence Relation гаргах Том subproblem-ийг аль хэдийн бодогдсон жижиг subproblem-уудтай холбох математик илэрхийллийг тодорхойлно.

3. Dependency ба бодох дарааллыг тодорхойлох Recurrence relation нь аль subproblem аль дээрээ хамааралтайг (dependency) харуулна. Үүний дагуу бодох зөв дарааллыг сонгоно.

4. Хадгалах ба бодох Subproblem бүрийн үр дүнг хадгалах өгөгдлийн бүтэц (array эсвэл map) ашиглана. Үүнийг memoization эсвэл tabulation гэж нэрлэдэг.

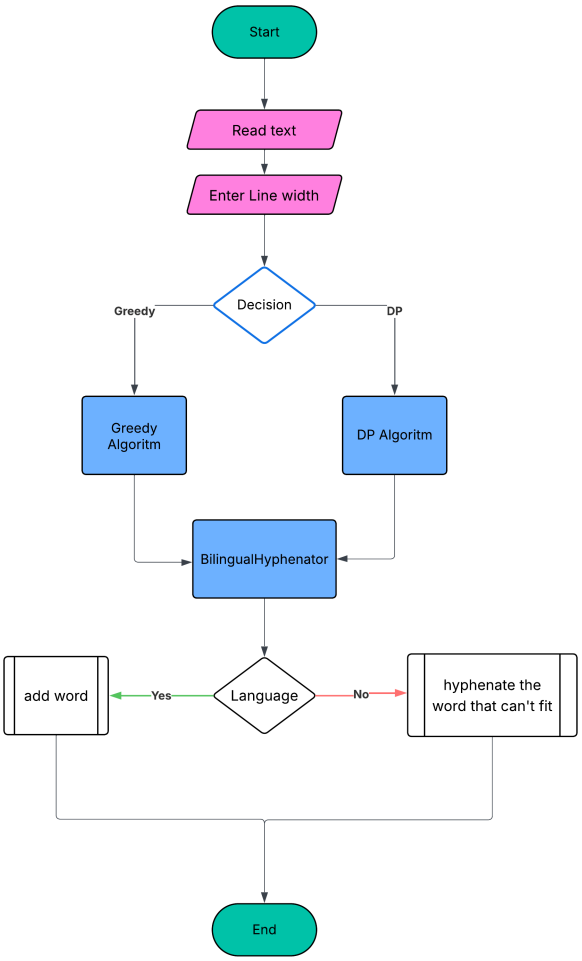
Dynamic Programming-ийн хоёр арга

Memoization (Top-Down)	Tabulation (Bottom-Up)
• Анхны асуудлаас эхэлнэ	• Хамгийн жижиг subproblem-оос эхэлнэ
• Рекурсив байдлаар subproblem-ууд болгон задална	• Давталт ашиглан үндсэн асуудал хүртэл бодож явна
• Үр дүнг cache (array эсвэл hash table)-д хадгална	• Хүснэгтийг (table) тодорхой дараалаар дүүргэнэ
• Бодохоосоо өмнө хадгалсан эсэхийг шалгана	• Рекурсив дуудлага шаардлагагүй

Text justification-д DP алгоритм нь мөрт багтах үгсийн зайны алдагдлыг (width - length^2) тооцоолон, бүх боломжит хуваалтыг үнэлж, хамгийн бага алдагдлыг бий болгох мөрүүдийн хуваалтыг сонгодог.

Жишээ нь, "This is an example of text justification"гэсэн өгүүлбэрийг 16 тэмдэгт өргөнтэй мөрүүдэд зөвтгөхдөө DP алгоритм нь бүх боломжит үгсийн хуваалтыг харьцуулан хамгийн оновчтой хувилбарыг олно. Энд *overlapping subproblems* буюу дахин давтагдах дэд асуудлуудыг хадгалах замаар тооцоог хурдан явуулдаг. Энэ арга нь greedy-ээс удаан боловч эцсийн шийдэл нь дэлхийд хамгийн оновчтой байдаг.

1.4 Ажиллагааны диаграмм



Зураг 1.3: Кодын flowchart диаграмм (Эх сурвалж: Зохиогч)

- **Feasible Solution** — асуудлын бүх нөхцөл болон хязгаарлалтыг (constraints) хангаж буй шийдэл.
- **Optimal Solution** — feasible solution-уудын дотроос objective function-ийн хамгийн их эсвэл хамгийн бага утгыг хангаж буй хамгийн сайн шийдэл.
- **Optimization Problems** — objective function-ийн maximum эсвэл minimum утгыг олох шаардлагатай асуудлууд.

2. Кодын тайлбар, ажиллагаа

2.1 Python хэл дээр

2.1.1 BilingualHyphenator, Hyphenator функц

```
class BilingualHyphenator:
    def __init__(self):
        current_dir = pathlib.Path(__file__).parent
        mn_dic_path = current_dir / "hyph_mn_MN.dic"
        self.mn_dic = pyphen.Pyphen(filename=str(mn_dic_path))
        self.en_dic = pyphen.Pyphen(lang='en_US')
        def hyphenate(self, word):
        if re.match(r'^[a-zA-Z-]+$', word):
            dic = self.en_dic
        else:
            dic = self.mn_dic
        inserted = dic.inserted(word)
        cuts = [i for i, c in enumerate(inserted) if c == "-"]
        return cuts
```

BilingualHyphenator: Монгол, Англи хэлний үгийг ялгаж, шаардлагатай үед үгийг таслан (*hyphenation*) мөрт байрлуулах боломжийг олгодог.

2.1.2 justify() функц

```
    def justify_line(words, width):
    if len(words) == 1:
        return words[0]

    total_chars = sum(len(w) for w in words)
    spaces_needed = width - total_chars
    gaps = len(words) - 1

    base = spaces_needed // gaps
    extra = spaces_needed % gaps

    line = ""
    for i, w in enumerate(words):
        line += w
```

```
        if i < gaps:
            line += " " * (base + (1 if i < extra else 0))
    return line
```

justify_line(): Мөр дүүрэх үгсийг тэнцвэртэй байрлуулж, хоосон зайг жигд болгоно.

2.1.3 Greedy алгоритм

```
    def greedy(text, width, hyph):
    words = text.split()
    result = []
    line_words = []
    current_len = 0
    for w in words:
        if current_len + len(w) + len(line_words) <= width:
            line_words.append(w)
            current_len += len(w)
        else:
            cuts = hyph.hyphenate(w)
            placed = False
            for c in reversed(cuts):
                left = w[:c] + "-"
                right = w[c:]
                if current_len+len(left)+len(line_words) <= width:
                    line_words.append(left)
                    result.append(justify_line(line_words, width))
                    line_words = [right]
                    current_len = len(right)
                    placed = True
                    break
            if not placed:
                result.append(justify_line(line_words, width))
                line_words = [w]
                current_len = len(w)
    if line_words:
        result.append(justify_line(line_words, width))

    return result
```

greedy(): Мөр бүрд тухайн үеийн хамгийн тохиромжтой үгийг нэмнэ. Хэрэв мөрт багтахгүй

бол үгийг таслан одоогийн мөрт байрлуулна. Эцэст нь мөрүүдийг тэнцвэртэй болгоно.

2.1.4 Dynamic Programming алгоритм

```
    def dp(text, width):
    words = text.split()
    n = len(words)
    def badness(i, j):
        line_words = words[i:j]
        total_chars = sum(len(w) for w in line_words)
        spaces = j - i - 1
        length = total_chars + spaces
        if length > width:
            return math.inf
        return (width - length) ** 2
    dp_arr = [math.inf] * (n + 1)
    nxt = [0] * (n + 1)
    dp_arr[n] = 0
    for i in range(n - 1, -1, -1):
        for j in range(i + 1, n + 1):
            cost = badness(i, j)
            if cost == math.inf:
                break
            if dp_arr[j] + cost < dp_arr[i]:
                dp_arr[i] = dp_arr[j] + cost
                nxt[i] = j
    lines = []
    i = 0
    while i < n:
        j = nxt[i]
        lines.append(justify_line(words[i:j], width))
        i = j
    return lines
```

dp(): Бүх боломжит үгсийн хуваалтыг үнэлж, хамгийн бага алдагдлыг бий болгох мөрүүдийн хуваалтыг олдог. Дахин давтагдах дэд асуудлуудын шийдлийг `dp_arr` массивт хадгална.

2.2 Java хэл дээр

2.2.1 Hyphenator класс

```
public class Hyphenator {
    private final List<String> patterns = new ArrayList<>();

    public Hyphenator(String dicPath) throws IOException {
        InputStream is = getClass().getClassLoader().
           getResourceAsStream(dicPath);
        if (is == null) throw new FileNotFoundException(dicPath);
        BufferedReader br = new BufferedReader(new InputStreamReader(
           is));
        String line;
        while ((line = br.readLine()) != null) {
            line = line.trim();
            if (!line.isEmpty() && !line.startsWith("%")) {
patterns.add(line);             } [
}
```

Үг таслах (*hyphenation*) дүрмүүдийг `.dic` файлаас уншиж, өгөгдсөн үгний аль хэсэгт таслалт хийх боломжтойг тодорхойлдог class юм. Үгийн үсгүүдэд оноо оноож, сондгой оноотой байрлалуудад таслалт хийх логик хэрэгжүүлсэн.

2.2.2 BilingualHyphenator class

```
public class BilingualHyphenator {
    private Hyphenator mn;
    private Hyphenator en;
    public BilingualHyphenator() throws Exception {
        mn = new Hyphenator("hyph_mn_MN.dic");
        en = new Hyphenator("hyph_en_US.dic");
    }
    public List<Integer> hyphenate(String word) {
        if (word.matches("^[a-zA-Z-]+$"))
            return en.hyphenate(word);
        return mn.hyphenate(word);
    }
}
```

BilingualHyphenator: Монгол болон Англи хэлний үгийг автоматаар ялгаж, тухайн хэлэнд тохирсон `Hyphenator`-ыг ашиглан үгийг тасалдаг. Англи үгийг латин үсгээр, бусдыг Монгол хэл гэж үзэн ялгадаг.

2.2.3 justify() функц

```
public static String justify(List<String> words, int width) {
    if (words.size() == 1) return words.get(0);
    int totalChars = words.stream().mapToInt(String::length).sum();
    int spacesNeeded = width - totalChars;
    if (spacesNeeded < 0) spacesNeeded = 0;
    int gaps = words.size() - 1;
    int base = spacesNeeded / gaps;
    int extra = spacesNeeded % gaps;
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < words.size(); i++) {
        sb.append(words.get(i));
        if (i < gaps) sb.append(" ".repeat(base + (i < extra ? 1 : 0)
           ));
    }
    return sb.toString();
}
```

justify(): Нэг мөрт багтах үгсийг өгөгдсөн өргөнд (`width`) тэнцвэртэй байрлуулж, хоосон зайг үг хооронд жигд хуваарилдаг функц юм. Илүүдэл зай байвал зүүн талаас нь эхлэн нэмдэг.

2.2.4 Greedy алгоритм

```
public static List<String> greedy(String text, int width,
   BilingualHyphenator hy) {
    List<String> lines = new ArrayList<>();
    List<String> cur = new ArrayList<>();
    int curLen = 0;
    for (String w : text.split("\\s+")) {
        if (curLen + w.length() + cur.size() <= width) {
            cur.add(w);
            curLen += w.length();
        } else {
            List<Integer> cuts = hy.hyphenate(w);
            boolean placed = false;
```

```
            for (int i = cuts.size() - 1; i >= 0; i--) {
                int c = cuts.get(i);
                String left = w.substring(0, c) + "-";
                String right = w.substring(c);
                if (curLen + left.length() + cur.size() <= width) {
                    cur.add(left);
                    lines.add(justify(cur, width));
                    cur = new ArrayList<>();
                    cur.add(right);
                    curLen = right.length();
                    placed = true;
                    break;
                }
            }
            if (!placed) {
                lines.add(justify(cur, width));
                cur = new ArrayList<>();
                cur.add(w);
                curLen = w.length();
            }
        }
    }
    if (!cur.isEmpty()) lines.add(justify(cur, width));
    return lines;
}
```

greedy(): Greedy алгоритмын зарчмаар мөр бүрд аль болох олон үг багтаахыг оролддог. Хэрэв дараагийн үг мөрт багтахгүй бол үгийг тасалж (*hyphenation*) тухайн мөрт байрлуулахыг оролддог.

2.2.5 Dynamic Programming алгоритм

```
public static List<String> dp(String text, int width,
   BilingualHyphenator hy) {
    String[] words = text.split("\\s+");
    int n = words.length;
    int[] dp = new int[n + 1];
    int[] nextBreak = new int[n + 1];
    String[] nextLine = new String[n + 1];

```

```
8       dp[n] = 0;
9
10      for (int i = n - 1; i >= 0; i--) {
11          dp[i] = Integer.MAX_VALUE;
12
13          List<String> curLine = new ArrayList<>();
14          int curLen = 0;
15
16          for (int j = i; j < n; j++) {
17              String w = words[j];
18
19              if (curLen + w.length() + curLine.size() <= width) {
20                  curLine.add(w);
21                  curLen += w.length();
22                  int spaces = width - (curLen + curLine.size() - 1);
23                  int cost = spaces * spaces + dp[j + 1];
24
25                  if (cost < dp[i]) {
26                      dp[i] = cost;
27                      nextBreak[i] = j + 1;
28                      nextLine[i] = justify(new ArrayList<>(curLine),
                            width);
29                  }
30              } else {
31                  List<Integer> cuts = hy.hyphenate(w);
32                  boolean placed = false;
33                  for (int k = cuts.size() - 1; k >= 0; k--) {
34                      int c = cuts.get(k);
35                      String left = w.substring(0, c) + "-";
36                      String right = w.substring(c);
37                      if (curLen + left.length() + curLine.size() <=
                            width) {
38                          curLine.add(left);
39                          int spaces = width - (curLen + left.length()
                                + curLine.size() - 1);
40                          int cost = spaces * spaces + dp[j + 1];
41                          if (cost < dp[i]) {
42                              dp[i] = cost;
43                              nextBreak[i] = j + 1;
```

```
                                nextLine[i] = justify(new ArrayList<>(
                                    curLine), width);
                            }
                            words[j] = right;
                            placed = true;
                            break;
                        }
                    }
                    if (!placed) break;
                    break;
                }
            }

            List<String> lines = new ArrayList<>();
            int idx = 0;
            while (idx < n) {
                lines.add(nextLine[idx]);
                idx = nextBreak[idx];
            }

            return lines;
        }
```

dp(): Dynamic Programming ашиглан бүх боломжит мөрийн хуваалтыг үнэлж, мөрийн төгсгөлд үлдсэн зайны квадратаар алдагдлыг (*cost*) тооцон хамгийн бага алдагдалтай шийдлийг олдог. Дэд бодлогуудын үр дүнг `dp[]` массивт хадгалж, давхардсан тооцооллыг багасгадаг.

3. Дүгнэлт

Энэхүү бие даалтаар текстийг мөрт багтаан тэгшлэх, үг таслах процессыг Java, Python хэл дээр хэрэгжүүлж, хоёр төрлийн алгоритмыг харьцуулсан. Greedy алгоритм нь энгийн, хурдан бөгөөд мөр бүрд аль болох олон үг багтаах зарчмаар ажилладаг тул бодлогын хувьд шуурхай бөгөөд ойлгомжтой үр дүн гаргадаг. Гэвч энэ арга нь зөвхөн тухайн мөрийн орчин үеийн хамгийн оновчтой байдалд тулгуурладаг учир нийт текстийн тэгшлэлт, хоосон зайны алдагдлыг хамгийн бага болгоход хэцүү байдаг.

Үүний эсрэг, Dynamic Programming (DP) алгоритм нь бүх боломжит мөрийн хуваалтыг үнэлж, үлдсэн зайны квадратын нийлбэрээр алдагдлыг тооцоолдог тул эцсийн үр дүнд хамгийн оновчтой тэгшлэлт, уншихад илүү тааламжтай текстийн бүтэц үүсгэдэг. DP алгоритмын нэг сул тал нь тооцооллын хурд илүү удаан, их хэмжээний текстэд хэрэглэхэд их нөөц шаардаг гэдэгт оршино. Гэхдээ энэ нь хэрэглэгчид илүү нарийн зохион байгуулалттай, уншихад тааламжтай формат санал болгодог.

Мөн бие даалтаар Монгол болон Англи хэлний үгийг автоматаар ялган, хэл бүрд тохирсон таслах дүрмийг ашигласан. Hyphenation-ийн ашиглалт нь зөвхөн мөрт багтах боломжийг нэмэгдүүлэхээс гадна текстийн унших ур чадварыг сайжруулж, виртуал зохион байгуулалтыг оновчтой болгодог давуу талтай.

Ерөнхийд нь хэлбэл, Greedy арга нь хурдан шийдэл, DP арга нь оновчтой шийдэл гэсэн зарчимд тулгуурлан, хэрэглэгчийн шаардлага, текстийн хэмжээ, хурдны шаардлагаас шалтгаалан сонголт хийх боломжтой. Энэ бие даалтын ажил нь текст боловсруулалт, программчлалын алгоритмын практик хэрэглээ, мөн олон хэлний тексттэй ажиллах аргачлалыг нэгтгэсэн жишээ болсон.

Номзүй

[1] Prajun T, "Dynamic Programming," Medium, 2023, https://medium.com/@prajun_t/dynamic-programming-f979106ae7c8, accessed December 2025.

[2] Manali Teke, "Understanding Greedy Algorithms," Medium, 2023, https://medium.com/@manali.teke19/understanding-greedy-algorithms-39a8abe26b83, accessed December 2025.

[3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, *Introduction to Algorithms*, 3rd edition, MIT Press, 2009.

[4] Wikipedia contributors, "Dynamic programming — Wikipedia," https://en.wikipedia.org/wiki/Dynamic_programming, accessed December 2025.