Министерство науки и высшего образования Российской Федерации
ФГАОУ ВО «УрФУ имени первого Президента России Б.Н. Ельцина»

Оценка работы_____

Динамическое программирование

ОТЧЕТ
Вид практики: Учебная практика
Тип практики: Учебная практика, научно-исследовательская работа
(получение первичных навыков научно-исследовательской работы)

Руководитель практики от предприятия (организации):

Студент:

Специальность (направление подготовки):

Группа:

Екатеринбург 20 ____

Теоретическая часть

Динамическое программирование – это метод решения оптимизационных задач, основанный на разбиении исходной задачи на более мелкие подзадачи, рекуррентно связанные между собой.

Динамическое программирование может быть применено к широкому кругу задач, включая нахождение оптимального маршрута в графе, поиск наибольшей общей подпоследовательности двух строк, нахождение наибольшего подмассива в массиве чисел и многие другие.

Существует два основных метода вычислений результата в динамическом программировании.

1. Рекурсивный - для каждой задачи можно составить рекуррентную формулу, которая будет описывать её решение.
2. Последовательный

В обоих случаях для уменьшения числа вычислений используют мемоизацию – сохранение результатов выполнения функции.

Для успешного применения динамического программирования необходимо

- разбить исходную задачу на подзадачи
- определить оптимальную подструктуру
- разработать алгоритм для вычисления результатов.

Задача имеет оптимальную подструктуру, если её оптимальное решение может быть рационально составлено из оптимальных решений её подзадач. При правильном применении этот метод может значительно ускорить решение сложных задач и существенно повысить эффективность вычислений.

Практическая часть

Для демонстрации применения метода динамического программирования приведу следующую задачу.

Исходные данные

В первой строке через пробел даны два целых числа n и k — количество рядов в салоне самолёта и количество людей, купивших билеты вместе, соответственно ($1 \le n \le 100, 1 \le k \le 6 \cdot n$).

Далее следуют n строк, каждая из которых содержит описание очередного ряда — строку длины 9 символов. Символ «.» (код 46) означает, что кресло свободно, «*» (код 42) — что занято. Первые три символа описывают занятость мест *A,B,C*. Затем следуют ещё три символа «|_|» (коды 124, 95, 124; без кавычек) — описание прохода между креслами. Последние три символа описывают занятость мест *D,E,F*.

Гарантируется, что число k не превосходит количество свободных мест в салоне.

Результат

Если рассадить всех людей нужным способом невозможно, в единственной строке выведите слово «`PORAZHENIE`». Иначе в первой строке выведите слово «`POBEDA`», затем в следующих k строках выведите номера мест, на которые нужно посадить людей, в любом порядке. Номера мест следует выводить в формате, описанном в условии. Если возможных ответов несколько, выведите любой.

Неформальное изложение алгоритма

1. Для начала считываем данные. Данные о ряде удобно хранить в виде десятичного целого числа *s* ($0 \le s \le 64$) – двоичное представление которого является “маской” ряда (1 – место занято, 0 - свободно). Имеем три массива.

- seats[i] – схема i-ого ряда

- dp[i][mask] - максимальное число пассажиров, мы можем посадить на первых i рядах, где mask – конкретное расположение пассажиров в ряду i
- prev[i][mask] –оптимальная маска для ряда i-1

2. Задача – найти максимум пассажиров, который можно усадить в самолет. Подзадача – найти максимум пассажиров (и их расположение), который можно усадить на i-ый ряд с учетом известного расположения для (i-1)-ого ряда. Именно в этом и заключается подход динамического программирования – результат задачи находится из решения большого количества однотипных подзадач, ведущих к нему.

Для каждого ряда последовательно находим максимум пассажиров перебором всех масок. Тогда

dp[i][mask] = max(dp[i][mask], dp[i-1][front] + count(mask))

- front – оптимальная маска для переднего (i - 1) ряда
- count(mask) – количество занятых мест новыми пассажирами

prev[i][mask] = front

- запоминаем оптимальную маску предыдущего ряда для сбора ответа

3. Выводим ответ:

- если максимальное число пассажиров, что могут сесть в самолете меньше числа пассажиров k –
  if dp[n + 1][0] < mask
  then “PORAZHENIE”
- иначе “POBEDA”, в обратном порядке собираем ответ – маски для ряда i, каждая из которых является индексом к маске (i - 1)

Алгоритм решения на языке Kotlin:

```
fun main() {
    val solver = VictoriaSolver()
    solver.complete()
}

class VictoriaSolver {
    private lateinit var seats: IntArray
    private lateinit var prev: Array<IntArray>
    private lateinit var dp: Array<IntArray>

    fun complete() {
        val (rowNumber, peopleNumber) = readLine()!!.split(" ").map
{ it.toInt() }

        seats = IntArray(rowNumber + 2) { 0 }
        prev = Array(rowNumber + 2) { IntArray(64) { 0 } }
        dp = Array(rowNumber + 2) { IntArray(64) { 0 } }

        for (i in 1..rowNumber) {
            val line = readLine()!!.split("|_|").joinToString("").reversed()
            line.forEachIndexed { index, char ->
                val bit = if (char == '*') 1 else 0
                seats[i] = seats[i] or (bit shl index)
            }
        }


        for (i in 1..rowNumber + 1) {
            for (front in 0..63) {
                for (mask in 0..63) {
                    if (isValid(mask, front, i, seats)
                        && dp[i][mask] < dp[i - 1][front] + count(mask)
                    ) {
                        dp[i][mask] = dp[i - 1][front] + count(mask)
                        prev[i][mask] = front
                    }
                }
            }
        }


        if (dp[rowNumber + 1][0] < peopleNumber) {
            println("PORAZHENIE")
        } else {
            println("POBEDA")
            if (peopleNumber != 0) {
                var counter = 0
                var m = 0
                for (i in rowNumber + 1 downTo 1) {
                    m = prev[i][m]
                    for (j in 0..5) {
                        if (isSet(j, m) && counter < peopleNumber) {
                            println("${i - 1}${'F' - j}")
                            counter++
                        }
                    }
                }
            }
        }
    }


    fun isValid(mask: Int, frontMask: Int, row: Int, seats: IntArray):
Boolean {
```

```
        val notOccupied = mask and seats[row] == 0
        val hasNeighboursNextTo = hasNeighboursNextTo(mask)
        val hasNeighboursFront = hasNeighboursRow(mask, frontMask)
        return notOccupied && !hasNeighboursNextTo && !hasNeighboursFront
    }

    /* Проверяем, сидят ли 2 человека в [mask] рядом, не считая прохода. */
    fun hasNeighboursNextTo(mask: Int): Boolean {
        for (k in 0..4) {
            if (k != 2 && (mask and (3 shl k) == (3 shl k)))
                return true
        }
        return false
    }

    /* Проверяем каждое потенциальное место на наличие соседей в соседнем
ряду. */
    fun hasNeighboursRow(mask: Int, frontMask: Int): Boolean {
        for (j in 0..5) {
            if (isSet(j, mask) && checkFront(j, frontMask)) return true
        }
        return false
    }

    /* Считает единички в [mask], т.е. занятые места. */
    fun count(mask: Int): Int {
        var counter = 0
        for (i in 0..5) {
            val hasBit = mask and (1 shl i) > 0
            counter += if (hasBit) 1 else 0
        }
        return counter
    }

    /* Возвращает true, если есть соседи в переднем ряду. */
    fun checkFront(seat: Int, frontMask: Int): Boolean {
        return isSet(seat, frontMask)
                || isSet(seat - 1, frontMask) && seat != 3
                || isSet(seat + 1, frontMask) && seat != 2
    }

    /** Проверяет установлен ли i-ый бит маски m */
    val isSet: (i: Int, m: Int) -> Boolean = { i, m -> m and (1 shl i) != 0 }
}
```

Используя подход динамического программирования удалось добиться сложности O(n).

ID	Дата	Автор	Задача	Язык	Результат проверки	№ теста	Время работы	Выделено памяти
10224019	10:45:19 30 мар 2023	Lipatnikov Alexander	2143. Victoria!	Kotlin 1.4.0	Accepted		0.25	4 036 КБ