Ткаченко Кирилл Станиславович
Специалист по учебно-методической работе 1-й категории, аспирант, ассистент Севастопольского государственного университета.

Как уложить ранец и рюкзак?

В публикации рассматривается классическая задача динамического программирования об укладке ранца (рюкзака), вариант 0-1.

Достаточно часто при решении вычислительных задач прикладной математики возникают ситуации, связанные с загрузкой определённых предметов в один. А именно, происходит выбор некоторого подмножества из конечного счётного множества элементов. Элементы характеризуются условными признаками веса и ценности. Необходимо, чтобы элементы выбранного подмножества обладали наибольшей суммарной ценностью, при этом не превосходя некоторой предельной вместимости ранца.

Наиболее хорошо изученной является задача о ранце 0-1. В этой задаче [1], [2] предполагается, что для каждого предмета происходит выбор не более одного экземпляра. При этом в статье [1] приводится фрагмент псевдокода для решения с взрывной, быстрорастущей длительностью работы, а в учебном пособии [2] – решение задачи о вычислении элементов двумерной таблицы с дополнительными ограничениями с фрагментами программ на школьном алгоритмическом языке и языке Pascal.

Пусть имеется N предметов, а максимальная вместимость ранца есть M. N и M – целые положительные числа. Для каждого предмета, i от 1 до N, известны вес и ценность. Удобно организовать два вектора целых положительных чисел для их хранения, а именно, вес [1 : N] и ценность [1 : N], при этом вес [i] и ценность [i] – соответственно вес и ценность i-го предмета.

Удобно ввести целые положительные переменные: номер_предмета – для хранения порядкового номера рассматриваемого в данный момент предмета и вместимость –

для хранения текущей возможной вместимости.

При решении задачи с использованием динамического программирования необходимо использовать таблицы. В данном случае будем использовать двумерный массив целых неотрицательных чисел ДП[0 : N, 0 : M] – массив ценностей. Строка таблицы (первый индекс массива) – это номер рассматриваемого элемента, столбец таблицы (второй индекс массива) – предельное значение вместимости.

В начальном случае, когда первый или второй индексы равны нулю, значение ДП[] равно нулю. Это очевидно, поскольку если не рассматриваются никакие элементы либо невозможно ничего положить в ранец, то ценность нулевая.

Начальные значения других элементов ДП[] не определены, но могут быть для удобства заданы равными нулю.

Так же, как и при решении любых задач динамического программирования, необходимо построить рекуррентные соотношения, на основании которых будет производиться переход от частных подзадач к более общим.

Будем рассматривать подзадачу в порядке по строкам – номер предмета от 1 до N, по столбцам – вместимость от 1 до M – как типовую задачу по обходу двумерного массива.

При этом при рассмотрении элемента ДП[номер_предмета, вместимость] необходимо присвоить ему некоторое значение. При этом возникает две ситуации выбора из подзадач, а именно, (1) предмет с номером номер_предмета не помещается в ёмкость; (2) предмет с номером номер_предмета помещается в ёмкость.

Ситуация (1) возникает тогда, когда вес рассматриваемого предмета строго больше, чем текущее значение вместимости, то есть если вес[номер_предмета] > вместимость. При этом очевидно, что значение в рассматриваемой ячейке ДП[] будет равно значению в ячейке ДП[] того же столбца, но на одну строку выше, то есть ДП[номер_предмета, вместимость] := ДП[номер_предмета – 1, вместимость].

Ситуация (2) возникает тогда, когда вес рассматриваемого предмета не превышает текущего значения вместимости, то есть если вес[номер_предмета] ≤ вместимость. В этой ситуации можно либо (2.1) – использовать предмет, либо (2.2) – не использовать его, в зависимости от того, в каком случае значение ДП[] больше, а именно необходимо найти максимальное значение из ДП[(2.1)] и ДП[(2.2)].

Расчёт для ДП[(2.1)] соответствует (1), как ДП[номер_предмета, вместимость] := ДП[номер_предмета – 1, вместимость].

Расчёт для ДП[(2.2)] несколько сложнее. Это вызвано тем, что при этом (2.2.а) вместимость подзадачи должна быть меньше рассматриваемой вместимости на величину веса предмета и (2.2.б) рассматриваемая ДП[] должна быть больше ДП[] подзадачи на величину ценности пред-

мета. При объединении (2.2.а) и (2.2.б) получается ДП[номер_предмета, вместимость] := ДП[номер_предмета - 1, вместимость - вес[номер_предмета]] + ценность[но-мер_предмета].

Окончательно приходим к рекуррентному соотношению (1) для заполнения таблицы в порядке по строкам номер_предмета от 1 до N, по столбцам вместимость от 1 до M:

$$ДП_{номер\_предмета,вместимость} = \begin{cases} ДП_{номер\_предмета-1,вместимость}, \text{если } вес_{номер\_предмета} > вместимость \\ \max\begin{cases} ДП_{номер\_предмета-1,вместимость}, \\ ДП_{номер\_предмета-1,вместимость-вес_{номер\_предмета}} + \\ + ценность_{номер\_предмета} \end{cases}, \\ \text{если } вес_{номер\_предмета} \le вместимость. \end{cases} \quad (1)$$

Соотношение (1) может быть удобно переписано для языков программирования, в которых отсутствует стандартная библиотечная функция нахождения максимума двух чисел, в форме из двух последовательных шагов (2):

$$1.ДП_{номер\_предмета,вместимость} := ДП_{номер\_предмета-1,вместимость};$$

$$2.ДП_{номер\_предмета,вместимость} := ДП_{номер\_предмета-1,вместимость-вес_{номер\_предмета}} + \quad (2)$$

$$+ ценность_{номер\_предмета}, \text{если } \left(вес_{номер\_предмета} \le вместимость\right) \wedge$$

$$\wedge \left(\begin{array}{l} ДП_{номер\_предмета-1,вместимость-вес_{номер\_предмета}} + \\ + ценность_{номер\_предмета} > ДП_{номер\_предмета-1,вместимость} \end{array} \right).$$

При этом наибольшей ценностью предметов, которые можно поместить в ёмкость, является величина в ДП[N, M].

Теперь необходимо получить номера предметов, которые в этом решении должны быть помещены в ранец. На этом этапе номер рассматриваемого предмета изменяется от N до 1, а вместимость уменьшается, начиная с M.

1. Вместимость := M.

2. В цикле изменяем номер_ предмета от N до 1.

3. Пусть рассматривается ДП[номер_предмета, вместимость].

4. Если предмет с номером номер_предмета должен быть помещён в ёмкость, то ДП[] в строке выше должна отличаться от рассматриваемой, то есть ДП[номер_предмета, вместимость] <> ДП[номер_предмета – 1, вместимость]. При этом необходимо зафиксировать

факт помещения предмета в ранец и уменьшить вместимость на величину веса предмета как вместимость := вместимость – вес [номер_предмета].

5. Переход к 2.

Указанные соотношения реализуются как алгоритм на школьном алгоритмическом языке. В листинге 1 приводится решение задачи для N = 5, M = 10, вес = [1, 2, 3, 4, 5], ценность = [4, 5, 6, 7, 8], что для системы КуМир приводится в листинге № 1.

Листинг 1 – реализация на школьном алгоритмическом языке

```
алг Ранец
нач
  цел таб вес[1:5], ценность[1:5]
  цел таб ДП[0:5, 0:10]
  цел номер_предмета, вместимость

  вес[1] := 1; вес[2] := 2
  вес[3] := 3; вес[4] := 4
  вес[5] := 5

  ценность[1] := 4; ценность[2] := 5
  ценность[3] := 6; ценность[4] := 7
  ценность[5] := 8

  ДП[0, 0] := 0
  нц для номер_предмета от 1 до 5
    ДП[номер_предмета, 0] := 0
  кц
  нц для вместимость от 1 до 10
    ДП[0, вместимость] := 0
  кц

  нц для номер_предмета от 1 до 5
    нц для вместимость от 1 до 10
      ДП[номер_предмета, вместимость] := ДП[номер_предмета - 1,
вместимость]
      если вес[номер_предмета] <= вместимость
      то
        если ДП[номер_предмета - 1, вместимость - вес[номер_ предме-
та]] + ценность[номер_предмета] > ДП[номер_предмета - 1, вместимость]
        то
          ДП[номер_предмета, вместимость] := ДП[номер_предмета - 1,
вместимость - вес[номер_предмета]] + ценность[номер_ предмета]
        все
      все
    кц
  кц

  вывод "Максимум: ", ДП[5, 10]
  вывод ". Предметы:"

  вместимость := 10
  нц для номер_предмета от 5 до 1 шаг -1
    если ДП[номер_предмета, вместимость] <> ДП[номер_предмета - 1,
вместимость]
    то
      вывод " ", номер_предмета
      вместимость := вместимость - вес[номер_предмета]
    все
  кц
кон
```

Протокол отладки программы из листинга 1 приводится в листинге 2, а результат выполнения за 525 шагов – в листинге 3.

Листинг 2 – протокол отладки

```
вес[1]=1; вес[2]=2
вес[3]=3; вес[4]=4
вес[5]=5

ценность[1]=4; ценность[2]=5
ценность[3]=6; ценность[4]=7
ценность[5]=8

ДП[0,0]=0
номер_предмета=5
ДП[5,0]=0

вместимость=10
ДП[0,10]=0

номер_предмета=5
вместимость=10
ДП[5,10]=22
да

нет

ДП[4,10]=22

вместимость=10
номер_предмета=1
да

вместимость=0
```

Листинг 3 – результат выполнения

```
Максимум: 22. Предметы: 4 3 2 1
```

Затем этот алгоритм реализуется как программа на языках программирования высокого уровня Java, Pascal, Python (с переводом идентификаторов переменных на английский язык), соответственно, в листингах 4, 5, 6.

Листинг 4 – исходный текст программы на языке программирования Java

```
public class Knapsack {
    public Knapsack() {
        final int NUMBER_OF_ITEMS = 5;
        final int KNAPSACK_CAPACITY = 10;

        int[] weight = new int[] { 1, 2, 3, 4, 5 };
        int[] value = new int[] { 4, 5, 6, 7, 8 };

        int[][] dp = new int[NUMBER_OF_ITEMS + 1][KNAPSACK_CAPACITY
        + 1];

        for (int itemIndex = 1; itemIndex <= NUMBER_OF_ITEMS;
        itemIndex++) {
            for (int capacity = 1; capacity <= KNAPSACK_CAPACITY;
            capacity++) {
                if (weight[itemIndex - 1] > capacity) {
```

```
                    dp[itemIndex][capacity] = dp[itemIndex - 1]
                    [capacity];
                } else {
                    dp[itemIndex][capacity] = Math
                            .max(dp[itemIndex - 1][capacity
                                    - weight[itemIndex - 1]]
                                    + value[itemIndex - 1],
                                    dp[itemIndex - 1][capacity]);
                }
            }
        }

        System.out.print("Maximum: ");
        System.out.println(dp[NUMBER_OF_ITEMS][KNAPSACK_CAPACITY]);

        java.util.HashSet<Integer> items_in_the_knapsack = new ja
        va.util.HashSet<Integer>();

        for (int itemIndex = NUMBER_OF_ITEMS, capacity = KNAP
        SACK_CAPACITY; itemIndex > 0; itemIndex--) {
            if (dp[itemIndex][capacity] != dp[itemIndex - 1]
            [capacity]) {
                items_in_the_knapsack.add(itemIndex);
                capacity -= weight[itemIndex - 1];
            }
        }

        System.out.print("Items:");
        for (int item : items_in_the_knapsack) {
            System.out.print(' ');
            System.out.print(item);
        }
        System.out.println();
    }

    public static void main(String[] args) {
        new Knapsack();
    }
}
```

Листинг 5 – исходный текст программы на языке программирования Pascal

```
program Knapsack;

procedure Main;
const
  number_of_items = 5;
  knapsack_capacity = 10;
var
  weight, value : array [1 .. number_of_items] of integer;
  dp : array [0 .. number_of_items, 0 .. knapsack_capacity] of integer;
  items_in_the_knapsack : set of 1 .. number_of_items;
  item_index, capacity : integer;
begin
  weight[1] := 1; weight[2] := 2;
  weight[3] := 3; weight[4] := 4;
  weight[5] := 5;
  value[1] := 4; value[2] := 5;
```

```
    value[3] := 6; value[4] := 7;
    value[5] := 8;

    dp[0][0] := 0;
    for item_index := 1 to number_of_items do
    begin
      dp[item_index][0] := 0;
    end;
    for capacity := 1 to knapsack_capacity do
    begin
      dp[0][capacity] := 0;
    end;

    items_in_the_knapsack := [];

    for item_index := 1 to number_of_items do
    begin
      for capacity := 1 to knapsack_capacity do
      begin
        dp[item_index][capacity] := dp[item_index - 1][capacity];
        if weight[item_index] <= capacity then
        begin
          if dp[item_index - 1][capacity - weight[item_index]] + value
          [item_index] >
              dp[item_index - 1][capacity] then
          begin
            dp[item_index][capacity] := dp[item_index - 1][capacity -
            weight[item_index]] + value[item_index];
          end;
        end;
      end;
    end;

    writeln('Maximum: ', dp[number_of_items][knapsack_capacity]);

    capacity := knapsack_capacity;
    for item_index := number_of_items downto 1 do
    begin
      if dp[item_index][capacity] <> dp[item_index - 1][capacity] then
      begin
        items_in_the_knapsack := items_in_the_knapsack + [item_index];
        capacity := capacity - weight[item_index];
      end;
    end;

    write('Items: [');
    for item_index := 1 to number_of_items do
    begin
      if item_index in items_in_the_knapsack then
      begin
        write(item_index, ', ');
      end;
    end;
    writeln(']');
  end;

  begin
      Main;
  end.
```

Листинг 6 – исходный текст программы на языке программирования Python

```
number_of_items = 5
knapsack_capacity = 10

weight = list(range(1, 5 + 1))
value = list(range(4, 8 + 1))

dp = [[0 for j in range(knapsack_capacity + 1)] for i in range
(number_of_items + 1)]

for item_index in range(1, number_of_items + 1):
    for capacity in range(1, knapsack_capacity + 1):
        if weight[item_index - 1] > capacity:
            dp[item_index][capacity] = dp[item_index - 1][capacity]
        else:
            dp[item_index][capacity] = max(dp[item_index - 1]
            [capacity - weight[item_index - 1]] + value[item_index - 1],
            dp[item_index - 1][capacity])

print('Maximum: %d' % dp[number_of_items][knapsack_capacity])

items_in_the_knapsack = []

capacity = knapsack_capacity

for item_index in range(number_of_items, 0, -1):
    if dp[item_index][capacity] != dp[item_index - 1][capacity]:
        items_in_the_knapsack += [item_index]
        capacity -= weight[item_index - 1]

print('Items: %s' % sorted(items_in_the_knapsack))
```

Таким образом, в публикации рассматривается решение классической задачи динамического программирования, выполняется построение рекуррентных соотношений и алгоритма.

Литература

1. Knapsack problem. – www.en.wikipedia.org/wiki/Knapsack_problem

2. *Котов В.М.* Информатика. Методы алгоритмизации / В.М. Котов, И.А. Волков, А.И. Лапо. – Минск: Народная асвета, 2000. – 300 с.

Юмор Юмор Юмор Юмор Юмор Юмор

Неожиданные определения

Интуиция – это ум воображения.

Р. Декарт

Идеи – редкая дичь в лесу слов.

В. Гюго

Время – лучший учитель, а практика – лучшая наука.

Пословица

Мысль – это путь от вопроса к ответу.

Г. Бокль