DOI: 10.24412/2413-2527-2023-335-68-73

Оптимизация образовательного процесса: построение индивидуальных учебных траекторий с помощью вариативной части учебного плана на примере задачи о рюкзаке

Д. Ю. Федоров, к.и.н. А. В. Забродин
Петербургский государственный университет путей сообщения Императора Александра I
Санкт-Петербург, Россия
elfbrus@gmail.com, teach-case@yandex.ru

Аннотация. **Исследуется использование комбинаторных оптимизационных задач для формирования вариативной части учебного плана с целью построения индивидуальных образовательных траекторий обучения. Рассматривается конкретная задача о рюкзаке в контексте оптимизации обучения. Представлен метод формирования индивидуальных учебных планов на основе анализа потребностей и способностей учащихся. Особое внимание уделяется процессу адаптации учебных программ под конкретные потребности студентов, что способствует более эффективному и персонализированному обучению. Результаты исследования могут быть полезными для образовательных учреждений и педагогов, стремящихся улучшить качество образования и поддержать индивидуальный образовательный опыт каждого обучающегося.**

Ключевые слова: **учебный план, дисциплина, алгоритм, задача о рюкзаке, траектория, оптимизация, статистика, сортировка.**

Введение

С увеличением численности студентов современные образовательные учреждения сталкиваются с вызовом поддержания высокого качества обучения.

Особенно это актуально для университетов, где численность студентов часто достигает значительных масштабов. Такие учебные заведения стоят перед задачей организации и планирования обучения для огромного числа обучающихся, каждый из которых имеет свои индивидуальные образовательные потребности. Эта ситуация требует применения более алгоритмичного и автоматизированного подхода к решению организационных задач.

В статье предложен вариант эффективного решения, направленного на автоматизацию процесса подбора дисциплин для вариативной части учебного плана студентами. Этот подход предоставляет студентам большую гибкость в выборе дисциплин, соответствующих их интересам и академическим целям, и при этом не требует активного вмешательства административного персонала или вышестоящего руководства.

Все больше и больше образовательных организаций автоматизируют процесс взаимодействия с обучающимися. В частности, это касается университетов, число студентов в которых составляет несколько тысяч. В таких условиях необходимо более алгоритмичное и автоматизированное взаимодействие по организационным вопросам. В статье рассмотрен способ автоматизации подбора дисциплин вариативной части студентом, облегчающим процесс выбора направлений, не требующий участия вышестоящего руководства.

Предметная область

Образовательная программа подразделяется на два основных компонента: обязательную и вариативную часть.

Обязательная часть представляет собой набор учебных предметов или курсов, которые студенты обязаны изучать в рамках своей учебной программы. Эти предметы могут включать в себя основные дисциплины, необходимые для получения образовательного стандарта или квалификации. Обязательная часть также включает в себя фундаментальные знания и навыки, необходимые для успешного завершения образовательной программы.

Вариативная часть предоставляет студентам возможность выбирать определенное количество учебных предметов или курсов из предложенного учебным заведением перечня. Эта часть программы позволяет студентам настраивать свою образовательную траекторию с учетом их интересов, целей и специализации. Вариативная часть значительно способствует индивидуализации обучения и позволяет студентам глубже и шире исследовать предметы, которые соответствуют их учебным интересам и будущим карьерным планам.

Таким образом, комбинация обязательной и вариативной части образовательной программы обеспечивает баланс между установленными стандартами обучения и индивидуальной гибкостью в выборе учебных материалов, что способствует более качественному и персонализированному обучению.

Дисциплины в вариативной части (ВЧ) представляют собой дополнительные учебные предметы. Обычно учебное заведение предоставляет ограниченное количество часов или времени, в рамках которого студент может выбирать наиболее интересные для него дисциплины из предложенного перечня.

Рассмотрим пример. Имеется 100 часов, в которые необходимо поместить следующие предметы: «Теория вероятности» — 25 часов, «Операционные системы» — 50 часов, «Информатика» — 50 часов, «Линейная алгебра» — 25 часов. Студенту необходимо выбрать наиболее приоритет-

ные для него направления, укладывающиеся в установленные временные рамки. В статье рассматривается идея алгоритмизации этого достаточно нетривиального процесса.

Вводя концепцию рейтинга, мы определяем приоритеты различных специальностей, оценивая их на шкале, например от 1 до 3. Здесь «1» обозначает менее привлекательные направления, «2» — более привлекательные, «3» — наиболее интересные. Путем установления приоритетов для перечисленных дисциплин можно выявить наиболее оптимальный набор предметов, который наберет наибольшее количество баллов рейтинга и при этом уложится в заданный временной интервал.

Задача решается в рамках построения индивидуальной образовательной траектории вариативной части учебного плана одного из направлений подготовки Петербургского государственного университета путей сообщения.

Методология решения

Существует алгоритм, позволяющий автоматизировать решение этой проблемы, так называемая задача о рюкзаке (knapsack problem), интенсивное изучение которой началось после из публикации книги математик Джорджа Данцига (G. Dantzig) в 1957 году. Она заключается в поиске оптимального набора предметов из заданного списка с ограничением по суммарному весу при условии максимизации их суммарной стоимости. Задачу о рюкзаке начать следует с решения задачи для меньшего рюкзака («подрюкзака»), а потом на этой основе попытаться решить исходную задачу [1].

Требуется вычислить подмножество предметов с максимально возможной суммарной стоимостью при условии наличия суммарного размера, не превышающего вместимость рюкзака [2].

Условия задачи: дано N предметов, W — вместимость рюкзака, $w = \{w_1, w_2, \ldots, w_n\}$ — соответствующий ему набор положительных целых весов, $p = \{p_1, p_2, \ldots, p_n\}$ — соответствующий ему набор положительных целых стоимостей. Нужно найти набор бинарных величин $B = \{b_1, b_2, \ldots, b_n\}$, где $b_i = 1$, если предмет n_i включен в набор $b_i = 0$, если предмет не включен, и такой, что:

1) $b_1w_1 + \ldots + b_nw_n \le W$;

2) $b_1p_1 + \ldots + b_np_n = \max$.

Наиболее оптимальный метод решения данной задачи — метод динамического программирования, подразумевающий деление задачи на более мелкие подзадачи.

Пусть $A(k, s)$ есть максимальная стоимость предметов, которые можно уложить в рюкзак вместимости s, если можно использовать только первые k предметов, то есть $\{n_1, n_2, \ldots, n_{k-1}\}$. Назовем его набором допустимых предметов для $A(k, s)$.

$A(k, 0) = 0$, $A(0, s) = 0$. Найдем $A(k, s)$. Возможны 2 варианта:

1. Предмет k не попал в рюкзак. Тогда $A(k, s)$ равно максимальной стоимости рюкзака с такой же вместимостью и набором допустимых предметов $\{n_1, n_2, \ldots, n_{k-1}\}$.

2. Предмет k попал в рюкзак. Тогда $A(k, s)$ равно максимальной стоимости рюкзака, где вес s уменьшаем на вес k-го предмета и набор допустимых предметов $\{n_1, n_2, \ldots, n_{k-1}\}$, прибавляя стоимость k.

$$A(k,s) = \begin{cases} A(k-1,s), & b_k = 0; \\ A(k-1,s-w_k) + p_k, & b_k = 1. \end{cases}$$

То есть:

$$A(k,s) = \max\left(A(k-1,s), A(k-1,s-w_k) + p_k\right).$$

Стоимость искомого набора равна $A(N, W)$, так как нужно найти максимальную стоимость рюкзака вместимостью W, в который входят все предметы [3, 4].

Восстановим набор предметов, входящих в рюкзак. Определим, входит ли предмет n_i в искомый набор. Начинаем с элемента $A(i, w)$, где $i = N$, $w = W$. Для этого сравниваем $A(i, w)$ со следующими значениями:

1. Максимальной стоимостью рюкзака с такой же вместимостью и набором допустимых предметов $\{n_1, n_2, \ldots, n_{i-1}\}$, то есть $A(i - 1, w)$.

2. Максимальной стоимостью рюкзака с вместимостью на w_i меньше и набором допустимых предметов $\{n_1, n_2, \ldots, n_{i-1}\}$ плюс стоимость p_i, то есть $A(i - 1, w - w_i) + p_i$.

Заметим, что при построении A мы выбирали максимум из этих значений и записывали в $A(i, w)$. Если $A(i, w) = A(i - 1, w)$, то n_i не входит в искомый набор, иначе входит.

Путем применения рассмотренного алгоритма возможно определить оптимальный состав набора дисциплин для вариативной части обучения, в котором [5]:

W — общее количество часов, выделенных на вариативную часть учебной программы;

w — количество часов, требуемых для изучения каждой конкретной дисциплины;

P — рейтинг каждой дисциплины, который варьируется от 1 до 3;

B — искомый оптимальный набор дисциплин, который будет выбран на основе данного алгоритма.

Способ реализации

Для решения данной задачи разработано графическое приложение WinForms на языке C++.

Это приложение включает в себя следующие компоненты: модуль для выхода, модуль для загрузки базы данных в приложение, модуль для формирования выборки дисциплин, исходную базу данных, конечную базу данных. Доступ к модулям осуществляется через интерфейс приложения с помощью соответствующих кнопок (рис. 1).

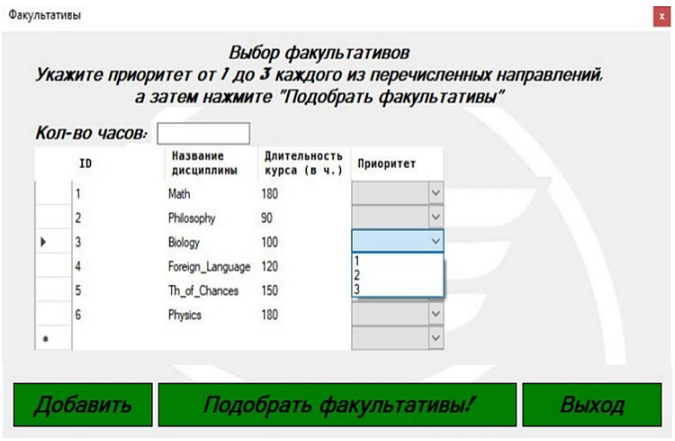

Рис. 1. Интерфейс приложения

Кнопка выхода является вспомогательной и используется для выхода из программы. Кнопка загрузки базы данных («Добавить») после активации обращается к файлу базы данных Access, содержащей информацию о студентах и результатах их выбора, и подгружает данные из нее в объект DataGridView, который реализует функционал базы данных в приложениях WinForms. Пример программного кода:

```
System::Void Facultatives::MyForm::btn_add_db_Click(System::Object^ sender, System::EventArgs^ e) {
        database->Rows->Clear();
        // Подключение базы данных
        String^ connectionString = "provider = Microsoft.ACE.OLEDB.12.0; Data Source=Facultatives.mdb";
        OleDbConnection^ dbConnection = gcnew OleDbConnection(connectionString);

        dbConnection->Open();
        String^ query = "SELECT * FROM [Facultatives]";
        OleDbCommand^ dbComand = gcnew OleDbCommand(query, dbConnection);  //
        OleDbDataReader^ dbReader = dbComand->ExecuteReader();

        if (dbReader->HasRows == false) {
                MessageBox::Show("Ошибка считывания данных!", "Ошибка!");
        }
        else {
                while (dbReader->Read()) {
                        database->Rows->Add(dbReader["id"], dbReader["name"], dbReader["weight"], dbReader["price"]);
                        wl++; // В переменную wl записывается кол-во строк - дисциплин
                }
        }
        dbReader->Close();
        dbConnection->Close();
        return System::Void();
}
```

Во время последовательной загрузки данных из базы, переменная *wl* аккумулирует количество строк, что соответствует доступным дисциплинам для выбора. Это вспомогательная переменная, которая будет использована в процессе создания выборки.

Параллельно загрузке базы данных построчно, в переменную *wl* записывается количество строк, то есть количество дисциплин на выбор. Данная переменная является вспомогательной и будет использована при построении выборки.

После загрузки базы данных необходимо заполнить колонку «Приоритет» (3 — наивысший приоритет). После заполнения указанной колонки необходимо активировать кнопку подбора дисциплин. Обработка данных начинается с проверки ошибок при заполнении полей, а также с обнуления счетчика таблицы выборки.

Пример кода программы:

```
// Вывод выборки на экран
        res_table->Rows->Clear(); // Обнуляем таблицу результата
        j = 0; // И вспомогательные переменные
        counter_values = 0;
        if (wl == 0) { // Если дисциплины не загружены - ошибка
        MessageBox::Show("Таблица не загружена!", "Ошибка!");
                return System::Void();
        }
        else if (max_capacity_label->Text == "") { // Если поле часов пустое - ошибка
        MessageBox::Show("Не введено кол-во часов!", "Ошибка!");
                return System::Void();
        }
        else { // Иначе в max_capacity количество часов для ВЧ
                max_capacity = Convert::ToInt32(max_capacity_label->Text);
        }
        for (i = 0; i < wl; i++) { // Если есть пустоты в столбце приоритетов - ошибка
if (database->Rows[i]->Cells[3]->Value->ToString() == "") {
        MessageBox::Show("Не все приоритеты заполнены!", "Ошибка!");
                return System::Void();
        }
                else { // Если пустот нет - формируем массивы весов и стоимостей
        string_buffer = database->Rows[i]->Cells[2]->Value->ToString();
        weights[i] = Convert::ToInt32(string_buffer);
        string_buffer = database->Rows[i]->Cells[3]->Value->ToString();
        values[i] = Convert::ToInt32(string_buffer);
                }
```

```
        }
        for (i = 1; i < wl; i++) if (values[i] == values[0]) counter_values++; // Сравниваем приоритет первой строки с остальными
        // При каждом совпадении инкрементируем. Если counter_values = кол-ву дисциплин - все приоритеты одинаковые - ошибка
if (counter_values == wl - 1) {
        MessageBox::Show("Введены одинаковые приоритеты!", "Ошибка!");
                return System::Void();
        } // Иначе - вызов функции построения матрицы весов
```

В рамках данного алгоритма, если нет ошибок, данные подвергаются преобразованию из колонок базы данных DataGridView, а также осуществляется заполнение массивов для дальнейшей обработки в числовом формате. Преобразование данных позволяет работать с ними как с числами в дальнейшем. При этом производится вызов функции *one_item*:

```
else Facultatives::MyForm::one_item(wl, max_capacity, weights, values);
```

Гибкость и расположение проверки условия определяют тип цикла, выбираемый в качестве управляющей структуры [6].

Функция *one_item* принимает в качестве входных аргументов количество дисциплин, количество часов, отведенное на эти дисциплины в текущем семестре, а также массивы, содержащие информацию о каждой дисциплине и их приоритете. Далее реализуется логика, описанная в методологии решения:

```
System::Void Facultatives::MyForm::one_item(int wl, int max_capacity, int weights[], int values[]) {
        int** arr; // Матрица весов
        arr = new int* [wl + 1]; // arr[n+1][m+1], где n - кол-во дисциплин, m - кол-во часов на ВЧ
        for (int i = 0; i < wl + 1; i++)
                arr[i] = new int[max_capacity + 1];
        for (int i = 0; i <= wl; i++) {
                for (int j = 0; j <= max_capacity; j++) {
                        if (i == 0 || j == 0) arr[i][j] = 0; // Нулевые строка и столбец заполняются нулями
                        else {
                                if (weights[i - 1] > j) arr[i][j] = arr[i - 1][j]; // если вес предмета больше текущей вместимости - вписываем
                                предыдущее значение в столбце
                        else arr[i][j] = max(arr[i - 1][j], values[i - 1] + arr[i - 1][j - weights[i - 1]]); // Иначе записываем значение по формуле:
максимум из предыдущей ячейки в столбце и суммы стоимости предмета с макс. стоимостью в предыдущей строке, которая помещается в
остаток места в рюкзаке
                        }
                }
        }
        //int answer = arr[wl][max_capacity]; // Макс. стоимость
        find_ans(wl, max_capacity, weights, arr); // Вызов функции формирования искомого массива индексов дисциплин
        for (int i = 0; i < wl + 1; i++) // Очистка памяти
                delete[]arr[i];
        delete[]arr;
        return System::Void();
}
```

Внутри функции *one_item* после формирования матрицы стоимостей происходит вызов функции *find_ans*, формирующей выходную выборку:

```
System::Void Facultatives::MyForm::find_ans(int wl, int max_capacity, int weights[], int** arr) {
        if (arr[wl][max_capacity] == 0) return; // Рекурсивная функция построения искомой выборки. Если значение ячейки = 0 - алгоритм "уперся
в стенку" и заканчивает обход матрицы
        if (arr[wl - 1][max_capacity] == arr[wl][max_capacity]) find_ans(wl - 1, max_capacity, weights, arr); // Если стоимости предметов равны,
вызываем ф-цию для wl-1
        else { // Если не равны, значит там однозначно меньшая цена. Записываем текущий предмет в выборку и вызываем функцию для
предыдущего предметы и цены предметов до предыдущего
                ans[j] = wl-1; // Искомая выборка
                j++; // Счетчик выборки
                find_ans(wl - 1, max_capacity - weights[wl - 1], weights, arr);
        }
        return System::Void();
}
```

Существование функций делает возможным использование такого средства программирования, как рекурсия. Рекурсия позволяет функции вызывать на выполнение саму себя. Это может показаться несколько неправдоподобным, и на самом деле зачастую обращение функции к самой себе вызывает ошибку, однако при правильном использовании механизм рекурсии может оказаться очень мощным инструментом в руках программиста [7].

Объект называется рекурсивным, если его части определены через него самого. Рекурсия встречается не только в математике, но и в обычной жизни [8].

Функция реализует рекурсивный поиск индексов в матрице способом, также описанным в методологии решения. Индексы записываются в отдельный массив ans. После обхода всей матрицы, функция завершается и возвращает управление в функцию *one_item*, которая также завершается и возвращает управление функции кнопки. Эта функция в свою очередь выводит во вторую базу данных DataGridView полученную выборку, перебирая массив индексов ans и изменяя тип данных для вставки в таблицу:

```
j--; // j в конце рекурсии получился на единицу больше, поэтому сперва уменьшаем его на 1
        while (j >= 0) { // В рез. таблицу записываем индекс, название и длительность дисциплины, индекс которой равен ans[j]
                res_table->Rows->Add(database->Rows[ans[j]]->Cells[0]->Value, database->Rows[ans[j]]->Cells[1]->Value, database->Rows[ans[j]]->Cells[2]->Value);
                j--; // Обходим массив до нулевого элемента включительно
        }
        return System::Void();
}
```

По завершении выполнения функции управление возвращается в функцию *main*. Искомый набор дисциплин найден. Оптимальный набор дисциплин выводится на экран (рис. 2).

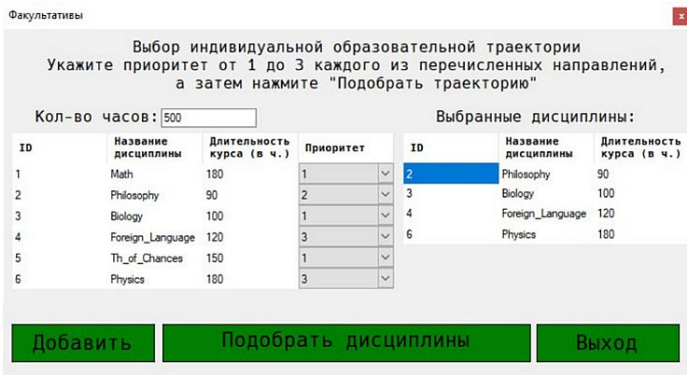

Рис. 2. Вывод результата на экран

Заключение

Построение вариативной части учебного плана является автоматизируемым процессом. При большом наборе входных данных эта деятельность может оказаться трудоемкой для ручного выполнения. В работе было продемонстрировано, что автоматизация процесса построения индивидуальных образовательных траекторий студентов в вариативной части учебного плана университета является не только возможной, но и весьма перспективной. Эта автоматизация позволит сократить трудозатраты и вероятность ошибок, что может существенно облегчить работу учебных заведений.

Разработанная система имеет потенциал для дальнейшего усовершенствования, что включает в себя расширение возможностей, таких как чтение данных из таблиц Excel и интеграция в системы дистанционного обучения университета. Это позволит сделать ее более функциональной и удобной для использования в реальном образовательном процессе.

Литература

1. Бхаргава, А. Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих = Grokking Algorithms: An illustrated guide for programmers and other curious people / Пер. с англ. Е. Матвеева. — Санкт-Петербург: Питер, 2017. — 288 с. — (Библиотека программиста).

2. Рафгарден, Т. Совершенный алгоритм. Алгоритмы для NP-трудных задач = Algorithms Illuminated. Part 4: Algorithms for NP-Hard Problems / Пер. с англ. А. Логунова. — Санкт-Петербург: Питер, 2021. — 304 с. — (Библиотека программиста).

3. Баушев, А. Н. Структуры и алгоритмы обработки данных: Учебное пособие. Часть I. — Санкт-Петербург: ПГУПС, 2019. —54 с.

4. Баушев, А. Н. Структуры и алгоритмы обработки данных: Учебное пособие. Часть II / А. Н. Баушев, А. В. Тырва. — Санкт-Петербург: ПГУПС, 2020. — 44 с.

5. Задача о рюкзаке // Викиконспекты. — Обновлено 04.09.2022.
URL: http://neerc.ifmo.ru/wiki/index.php?title=Задача_о_рюкзаке (дата обращения 16.04.2023).

6. Макконнелл, С. Совершенный код. Мастер-класс. Второе издание = Code Complete. Second Edition / Пер. с англ. под общей ред. В. Г. Вшивцева. — Москва: Русская редакция, 2010. — 896 с.

7. Лафоре, Р. Объектно-ориентированное программирование в C++. Четвертое издание / Пер. с англ. А. Кузнецова, [и др.]. — Санкт-Петербург: Питер, 2016. — 928 с. — (Классика Computer Science).

8. Вирт, Н. Алгоритмы и структуры данных. Новая версия для Оберона / Пер. с англ. под ред. Ф. В. Ткачева.— Москва: ДМК Пресс, 2010. — 272 с. — (Классика программирования).

DOI: 10.24412/2413-2527-2023-335-68-73

Optimization of the Educational Process: Construction of Individual Educational Trajectories Using the Variable Part of the Curriculum Using the Example of the Knapsack Problem

D. Yu. Fedorov, PhD A. V. Zabrodin
Emperor Alexander I St. Petersburg State Transport University
Saint Petersburg, Russia
elfbrus@gmail.com, teach-case@yandex.ru

Abstract. **The article examines the use of combinatorial optimization problems for the formation of a variable part of the curriculum in order to build individual educational learning trajectories. A specific knapsack problem is considered in the context of learning optimization. A method for forming individual curricula is presented based on an analysis of the needs and abilities of students. Particular attention is paid to the process of adapting curricula to the specific needs of students, which contributes to more effective and personalized learning. The results of the study may be useful for educational institutions and teachers seeking to improve the quality of education and support the individual educational experience of each student.**

Keywords: **curriculum, discipline, algorithm, knapsack problem, trajectory, optimization, statistics, sorting.**

REFERENCES

1. Bhargava A. Grokking Algorithms: An illustrated guide for programmers and other curious people [Grokaem algoritmy. Illyustrirovannoe posobie dlya programmistov i lyubopytstvuyushchikh]. Saint Petersburg, Piter Publishing House, 2017, 288 p.

2. Roughgarden, T. Algorithms Illuminated. Part 4: Algorithms for NP-Hard Problems [Sovershennyy algoritm. Algoritmy dlya NP-trudnykh zadach]. Saint Petersburg, Piter Publishing House, 2021, 304 p.

3. Baushev A. N. Data processing structures and algorithms: Study guide. Part I [Struktury i algoritmy obrabotki dannykh: Uchebnoe posobie. Chast I]. Saint Petersburg, Petersburg State Transport University, 2019, 54 p.

4. Baushev A. N., Tyrva A. V. Data processing structures and algorithms: Study guide. Part II [Struktury i algoritmy obrabotki dannykh: Uchebnoe posobie. Chast II]. Saint Petersburg, Petersburg State Transport University, 2020, 44 p.

5. The Backpack Problem [Zadacha o ryukzake], *Wikinotes [Vikikonspekty]*. Last update at September 04, 2022. Available at: http://neerc.ifmo.ru/wiki/index.php?title=Задача_о_рюкзаке (accessed 16 Apr 2023).

6. McConnell S. Code Complete. Second Edition [Sovershennyy kod. Master-klass. Vtoroe izdanie]. Moscow, Russian Edition Publishers, 2010, 896 p.

7. Lafore R. Object-oriented programming in C++. Fourth Edition [Obektno-orientirovannoe programmirovanie v C++. Chetvertoe izdanie]. Saint Petersburg, Piter Publishing House, 2016, 928 p.

8. Wirth N. Algorithms and data structure. New version for Oberon [Algoritmy i struktury dannykh. Novaya versiya dlya Oberona]. Moscow, DMK Press Publishing House, 2010, 272 p.