6. БАЗЫ ДАННЫХ, ОСНОВАННЫЕ НА ЛОГИКЕ

В логическом программировании база данных рассматривается как набор фактов (аксиом).

Выполнение запроса выполняется как доказательство того, что некоторая формула является логической последовательностью фактов, то есть теоремой. Результатом запроса будут частные значения переменных, при которых формула является истинной.

В *Прологе* реализована реляционная модель, которая предполагает, что база данных – это описание некоторого множества отношений.

Отношение в теории баз данных имеет имя и делится на две части: заголовок *(набор атрибутов)* и тело *(набор кортежей)*. Пример отношения реляционной базы данных показан в табл. 16.

Таблица 16

Пример отношения реляционной базы данных

Кортеж	**Атрибут$_1$**	**Атрибут$_2$**	**Атрибут$_3$**
1	Bratko Ivan	Programming for Artifical intelligence	Addison-Wesley
2	Doores J.	Prolog – programming for tomorrow	Sigma Press
…	…	…	…
25	Date C. J.	An introduction to database systems	Addison-Wesley

Нетрудно видеть, что кортеж соответствует факту в *пролог*-программе, атрибут – объекту, а отношение имеет предикатную структуру.

Справедливо соответствие терминов, показанное в табл. 17.

Таблица 17

Соответствие терминов

Реляционные базы данных	**Базы данных в Прологе**
отношение	предикат базы данных
атрибут	объект
кортеж (элемент отношения)	факт (отдельное утверждение)
степень (количество атрибутов)	арность (количество объектов)
мощность (кардинальное число)	количество утверждений

6.1. Базы данных в Прологе

Пролог-программу можно рассматривать как реляционную базу данных: описание отношений частично присутствует в ней в явном виде *(факты)*, а частично – в неявном *(правила)*.

База данных *Пролога* содержит набор основных, заранее определённых отношений. Кроме того, резервируется пространство для добавления предикатов, которые будут объявлены пользователем. Встроенные предикаты дают возможность корректировать базу данных в процессе выполнения программы. Операции для работы с утверждениями, заданными пользователем, классифицируются следующим образом:

1. Добавление утверждений из диалога с пользователем и удаление существующих утверждений.
2. Добавление утверждений из файла и замена имеющихся утверждений на находящиеся в файлах утверждения для того же самого предиката.

Для предиката базы данных существуют связанные с ним факты. В любой момент времени факт содержит в точности такие значения аргументов, для которых предикат является истинным.

В табл. 18 представлены факты предиката **dBook (author, title, pub)**.

Таблица 18

Факты предиката dBook (author, title, pub)

Факты	**Объект$_1$**	**Объект$_2$**	**Объект$_3$**
	author	**title**	**pub**
dBook	Bratko Ivan	Programming for Artifical intelligence	Addison-Wesley
dBook	Doores J.	Prolog – programming for tomorrow	Sigma Press
…	…	…	…
dBook	Date C. J.	An introduction to database systems	Addison-Wesley

В программе на *Прологе* существует специальная секция, обозначаемая ключевым словом **database**, которое определяет начало последовательности объявлений описывающих базу данных предикатов:

```
database
        dBook (author, title, pub)
```

Очевидно, что доменные имена аргументов предиката базы данных должны быть объявлены в секции **domains**.

Соответствующие предикатам базы данных факты помещаются в специальную область оперативной памяти, которая называется *внутренней базой данных*. По умолчанию ей присваивается стандартное имя **dbasedom**.

Допускается наличие нескольких секций **database**. В этом случае только одна секция может быть безымянной, для остальных – следует явно указать имя. Имена секций и предикатов во внутренней базе данных должны

быть уникальными. В различных секциях запрещено использование одинаковых имён предикатов. Например,

```
database                          % имя dbasedom по умолчанию
        dBook (author, title, pub)
        library (name)
database – myWork                 % имя myWork
        dWork (job, beginning, ending)
database – myPractice             % имя myPractice
        dPractice (name, date, period)
```

Следует заметить, что факты, объявленные в секции **facts**, автоматически помещаются во внутреннюю базу данных **dbasedom**. Поэтому секция **database**, которая используется в программе вместе с секцией **facts**, обязательно должна быть именованной.

Предикаты динамической базы данных можно использовать в программе точно так же, как и другие предикаты *Пролога*.

Все различные утверждения внутренней базы данных составляют *динамическую базу данных (ДБД)*.

Можно выделить две особенности динамической базы данных *Пролога*:

1. Во время работы программы можно удалять любые содержащиеся ДБД утверждения, а также добавлять новые.
2. ДБД может быть записана на диск и считана с диска в оперативную память.

Названные особенности позволяют различать динамическую и *статическую базу данных*, в которой утверждения представлены фактами и являются частью кода программы. Если факты определены в секции **facts**, то к ним можно применять предикаты для работы с базой данных. Для фактов, которые объявлены в секции **predicates**, эти предикаты недоступны.

Для достижения максимальной скорости работы программы факты, объявленные как недетерминированные предикаты, компилируются в двоичный код, но они **не могут быть изменены** во время работы программы. Если существует необходимость их добавления в ДБД, то предикаты ДБД и фактов из секции **predicates** обязательно должны иметь одинаковый формат представления объектов. Как правило, добавление в ДБД происходит сразу же после активизации программы (при помощи специально написанной процедуры добавления).

При работе с предикатами ДБД следует помнить следующее:

1. Добавлять можно **только факты**, но не правила.
2. Утверждения ДБД **не могут** содержать свободных переменных.
3. Новые факты ДБД могут быть получены из других фактов программы, а также путём логического вывода из имеющихся правил.

Пример 6-1. Объявление предикатов баз данных.

```
database        dBook (author, title, pub)
facts – db      library (name)
predicates      nondeterm book (author, title, pub)
```

dBook – предикат, который описывает динамическую базу данных с именем **dbasedom**.

book – недетерминированный предикат, который описывает факты и имеет одинаковый с предикатом ДБД формат, благодаря чему утверждения **book** могут быть записаны в динамическую базу данных **dbasedom**.

library – предикат, который описывает факты статической базы данных с именем **db**. Имя для секции **facts** необходимо указать, поскольку пространство **dbasedom** занято утверждениями **dBook**.

6.2. Предикаты для работы с динамической базой данных

Встроенные предикаты для работы с динамической базой данных применимы только в отношении фактов. Поток данных для этих предикатов – **(i)** или **(i, i)**.

6.2.1. Добавление

Для добавления в базу данных существует два способа:

1. Во время выполнения программы с помощью предикатов **asserta** и **assertz**.
2. Путём загрузки файла с фактами базы данных с помощью предиката **consult**.

Форматы предикатов добавления *в начало* базы данных:

```
asserta (fact)
asserta (fact, dbaseName)
```

, где **fact** – добавляемый факт,
dbaseName – имя базы данных.

Новый факт добавляется в ДБД перед имеющимися фактами для заданного предиката.

Форматы предикатов добавления *в конец базы* данных:

```
assertz (fact)
assertz (fact, dbaseName)
```

, где **fact** – добавляемый факт,
dbaseName – имя базы данных.

Новый факт добавляется после имеющихся в ДБД фактов для заданного предиката базы данных.

Формат предиката загрузки фактов из файла:

consult (fileName)
consult (fileName, dbaseName)

, где **fileName** – имя файла с утверждениями базы данных,
dbaseName – имя базы данных.

Формат файла должен соответствовать формату предиката **save** (раздел 6.2.3). Файл с утверждениями базы данных при необходимости может быть подготовлен в любом текстовом редакторе, но он не должен содержать:

- пустые строки;
- пробелы (исключая строковые константы);
- комментарии;
- прописные символы.

Предикат неуспешен, если указанный файл отсутствует.

Первый формат указанных выше предикатов используется для добавления во внутреннюю базу данных с именем **dbasedom**. Второй – для именованных секций **database**.

Пример 6-2. Добавление в ДБД из фактов программы.

```
domains
        brand = symbol
        year, price = integer
database
        dCar (brand, year, price)

predicates
        nondeterm car (brand, year, price)
        assertDB
clauses
        % процедура добавления в ДБД
        assertDB :-                                     % 1
                car (Brand, Year, Price),
                assertz (dCar (Brand, Year, Price)),
                fail.
        assertDB :- !.                                  % 2
        % факты
        car (chevrolet, 2009, 2300).
        car (pontiac, 2008, 8550).
        car (toyota, 2010, 11000).
        % ...
goal    assertDB.
```

Чтобы добавить все факты в ДБД, правило %1 использует предикат принудительного возврата **fail**. Правило %2 обеспечит успешное завершение процедуры **assertDB** после последней неудачи согласования правила %1.

6.2.2. Удаление

Для удаления конкретного факта используются предикаты, имеющие следующие форматы:

retract (fact)
retract (fact, dbaseName)

, где **fact** – удаляемый факт,
dbaseName – имя базы данных.

Первый формат используется для удаления из внутренней базы данных с именем **dbasedom**. Второй – для именованных секций **database**.

Пример 6-3. Удаление фактов из ДБД.

```
domains         list = integer*
facts - dba1    fact1 (integer, string, list)
facts - dba2    fact2 (integer, string)
clauses         fact1 (1, "fact1", [1,2,3]).
                fact1 (2, "fact2", [1,3]).
                fact1 (3, "fact2", [3,2,1]).
                fact2 (1, "one").
                fact2 (1, "one once more").
                fact2 (2, "two").
```

В табл. 19 показаны разные цели и решения для них.

Таблица 19

Цели и решения

Цель	Решение
goal fact1 (X, Y, Z).	**X = 1, Y = fact1, Z = [1,2,3]** **X = 2, Y = fact2, Z = [1,3]** **X = 3, Y = fact2, Z = [3,2,1]** **3 Solutions**
goal retract (fact1 (X, Y, [_, 2 | Z])).	**X = 1, Y = fact1, Z = [3]** **X = 3, Y = fact2, Z = [1]** **2 Solutions**
goal retract (fact1 (2,Y,Z)).	**Y = fact2, Z = [1,3]** **1 Solution**
goal retract (fact1 (X, Y, [_, 2 | Z])), **retract (fact1 (2,Y,Z)),** **fact1 (X,Y,Z).**	**No Solution**
goal retract (fact2 (1,X)).	**X = one** **X = one once more** **2 Solutions**

Для удаления из ДБД всех фактов, совпадающих с образцом, используется предикат **retractall**:

retractall (fact)
retractall (fact, dbaseName)

, где **fact** – образец удаляемых фактов,
dbaseName – имя базы данных.

Первый формат используется для удаления из внутренней базы данных с именем **dbasedom**. Второй – для именованных секций **database**.

Пример 6-4. Удаление фактов из ДБД по образцу.

```
domains                 имя, позиция, команда = string
database – составы      игрок (имя, позиция, команда)
facts                   кандидат (имя)
predicates              просмотрИгроков
                        просмотрКандидатов
                        удалитьПоОбразцу
                        удалитьВсех
clauses
        игрок ("Николас Ломбертс","защитник", "Зенит").
        игрок ("Фатих Текке", "нападающий", "Зенит").
        игрок ("Александр Рыбка",  "вратарь", "Динамо-Киев").
        игрок ("Вячеслав Малафеев", "вратарь", "Зенит").
        игрок ("Томаш Губочан", "защитник", "Зенит").
        игрок ("Сантос Моцарт", "полузащитник", "Спартак-Москва").
        % ...
        кандидат ("Рони Зайринг").
        кандидат ("Марк Плотников").
        кандидат ("Хайнц Георг Накке").
        % ...
        просмотрИгроков :-
                игрок (И, П, К), write (И, '\t', П, '\t', К), nl, fail.
        просмотрИгроков :- !.
        просмотрКандидатов :-
                кандидат (И), write (И), nl, fail.
        просмотрКандидатов :- !.
        удалитьПоОбразцу :-
                write ("Игроки\n"), просмотрИгроков,
                retractall (игрок (_, _, "Зенит")),
                write ("\nИгроки без 'Зенита'\n"), просмотрИгроков.
        удалитьВсех :-
                retractall (_, составы),
                write ("\nИгроки после удаления всех\n"),
                просмотрИгроков,
                write ("\nКандидаты\n"), просмотрКандидатов,
                retractall (_),
```

```
        write ("\nКандидаты после удаления всех\n\n"),
        просмотрКандидатов.
goal    удалитьПоОбразцу, удалитьВсех, exit.
```

Решение:

```
Игроки
Николас Ломбертс    защитник        Зенит
Фатих Текке         нападающий      Зенит
Александр Рыбка     вратарь         Динамо-Киев
Вячеслав Малафеев   вратарь         Зенит
Томаш Губочан       защитник        Зенит
Сантос Моцарт       полузащитник    Спартак-Москва

Игроки без 'Зенита'
Александр Рыбка     вратарь         Динамо-Киев
Сантос Моцарт       полузащитник    Спартак-Москва

Игроки после удаления всех

Кандидаты
Рони Зайринг
Марк Плотников
Хайнц Георг Накке

Кандидаты после удаления всех
```

6.2.3. Сохранение

Для сохранения служит предикат, имеющий следующие форматы:

save (fileName)
save (fileName, dbaseName)

, где **fileName** – произвольная допустимая спецификация файла с утверждениями базы данных (принятый тип для *Пролога* **.dba**);
dbaseName – имя базы данных.

Этот предикат сохраняет в текстовом файле с заданным именем базу данных, находящуюся в оперативной памяти (либо **dbasedom** согласно первому формату, либо именованную – согласно второму). Утверждения записываются построчно в соответствии с объявленными предикатами базы данных. Если в секции объявлено несколько предикатов, то все имеющиеся на момент сохранения утверждения будут выгружены в файл. Утверждения группируются по именам предикатов базы данных.

Пример 6-5. Загрузка и сохранение утверждений ДБД (и не только).

```
domains
        % множественное объявление объектов-структур
        инфо  =  столица (string, string) ;
                 площадь (string, ulong) ;
                 численность (string, ulong)
database
        % объявление предикатов базы данных dbasedom
        страна (инфо)
        оСтране (string, string, ulong, ulong)
predicates
        % объявление предикатов
        показать        создать
clauses
        % факты
        страна (столица ("Дания", "Копенгаген")).
        страна (площадь ("Дания", 43000)).
        страна (численность ("Дания", 5100000)).
        страна (столица ("Сингапур", "Сингапур")).
        страна (площадь ("Сингапур", 600)).
        страна (численность ("Сингапур", 2400000)).

        % добавление в dbasedom утверждений оСтране (...)
        создать :-
                страна (столица (Страна, С)),
                страна (площадь (Страна, П)),
                страна (численность (Страна, Ч)),
                % проверка отсутствия утверждения оСтране (...)
                not (оСтране (Страна, С, П, Ч)),
                % добавление утверждения в конец ДБД
                assertz (оСтране (Страна, С, П, Ч)),
                fail.
        создать :- !.

        % просмотр утверждений оСтране (...)
        показать :-
                оСтране (Страна, С, П, Ч),
                write (Страна, '\t', С, '\t', П, '\t', Ч, '\n'),
                fail.
        показать :- write ("Конец просмотра\n"), !.
goal
        % создать и показать утверждения оСтране (...)
        создать, показать,
        % сохранить утверждения dbasedom в файле
        save ("E:страны.dba"),
        % удалить из dbasedom все утверждения оСтране (...)
        retractall (оСтране(_,_,_,_)),
```

% удалить из dbasedom все остальные утверждения и показать состояние
retractall (_), показать,
% загрузить утверждения из файла
consult ("E:страны.dba"),
% добавить факты в dbasedom
assertz (страна (столица ("Болгария", "София"))),
assertz (страна (площадь ("Болгария", 111000))),
assertz (страна (численность ("Болгария", 9000000))),
% создать и показать утверждения оСтране (...)
создать, показать,
% сохранить утверждения dbasedom в новом файле
save ("E:страныНовые.dba"),
exit.

Решение:

Дания Копенгаген 43000 5100000
Сингапур Сингапур 600 2400000
Конец просмотра
Конец просмотра
Дания Копенгаген 43000 5100000
Сингапур Сингапур 600 2400000
Болгария София 111000 9000000
Конец просмотра

Распечатка файла **страны.dba**

```
страна(столица("Дания","Копенгаген"))
страна(площадь("Дания",43000))
страна(численность("Дания",5100000))
страна(столица("Сингапур","Сингапур"))
страна(площадь("Сингапур",600))
страна(численность("Сингапур",2400000))
остране("Дания","Копенгаген",43000,5100000)
остране("Сингапур","Сингапур",600,2400000)
```

Распечатка файла **страныНовые.dba**

```
страна(столица("Дания","Копенгаген"))
страна(площадь("Дания",43000))
страна(численность("Дания",5100000))
страна(столица("Сингапур","Сингапур"))
страна(площадь("Сингапур",600))
страна(численность("Сингапур",2400000))
страна(столица("Болгария","София"))
страна(площадь("Болгария",111000))
страна(численность("Болгария",9000000))
остране("Дания","Копенгаген",43000,5100000)
остране("Сингапур","Сингапур",600,2400000)
остране("Болгария","София",111000,9000000)
```

6.3. Повышение эффективности за счёт базы данных

В процессе вычислений иногда возникает ситуация, когда одну и ту же цель приходится достигать снова и снова. Поскольку в *Прологе* отсутствует специальный механизм выявления подобной ситуации, соответствующая цепочка вычислений каждый раз повторяется заново.

Повторений легко избежать, если запоминать каждое вновь вычисленное значение в ДБД. Для этого применяется встроенный предикат **asserta** для добавления промежуточных результатов в базу данных в виде фактов. Эти факты должны предшествовать другим предложениям, чтобы предотвратить применение общего правила в случаях, для которых результат уже известен. Запоминание промежуточных результатов – стандартный метод, позволяющий избегать повторных вычислений.

Пролог-программа при попытке достижения какой-либо цели сначала будет проверять накопленные об этом отношении факты и только после этого применять общее правило.

В ряде случаев повторных вычислений можно избежать применением другого алгоритма, а не только запоминанием промежуточных результатов.

Пример 6-6. Вычисление N-го числа Фибоначчи.

Числа $\mathbf{F_N}$, образующие последовательность 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, ... называются *числами Фибоначчи*.

При всех $\mathbf{N > 2}$ число Фибоначчи вычисляется по следующей формуле:

$$\mathbf{F_N = F_{N-1} + F_{N-2}}.$$

Декларативный смысл. Каждое число последовательности, за исключением первых двух, представляет собой сумму предыдущих двух чисел.

Программа без использования ДБД:

```
domains
        x = integer         res = real
predicates
        fib (x, res)
clauses
        fib (1, 1) :- !.
        fib (2, 1) :- !.
        fib (N, F) :-
                % вычислить (N-1)-ое число Фибоначчи
                N1 = N - 1, fib (N1, F1),
                % вычислить (N-2)-ое число Фибоначчи
                N2 = N - 2, fib (N2, F2),
                % вычислить N-ое число Фибоначчи
                F = F1 + F2.
goal   fib (40, Res), write (Res), exit.
```

Результат **Res = 102334155** будет получен приблизительно через 6 сек. Для чисел **N > 43** значение вычислить проблематично из-за необходимости хранения промежуточных результатов в стеке. На рис. 5 показана последовательность вычислений шестого числа Фибоначчи при помощи процедуры **fib**, которая имеет явную тенденцию к одним и тем же вычислениям. Так, число Фибоначчи f_3 понадобилось в трёх местах, и три раза были повторены одни и те же вычисления.

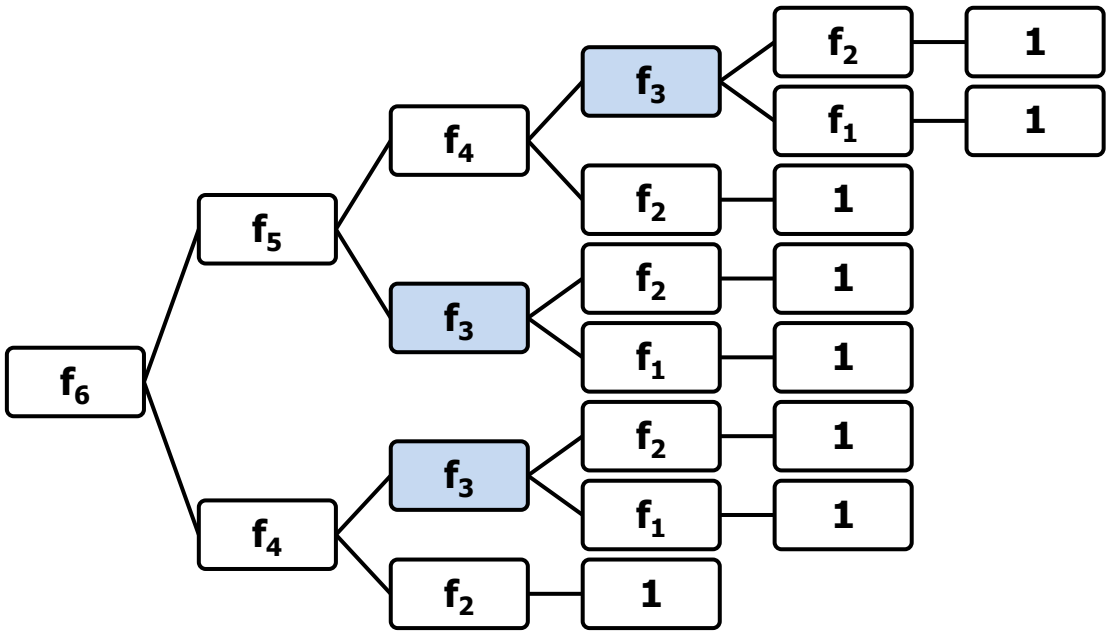

Рис. 5. Вычисление шестого числа Фибоначчи процедурой fib

Программа с использованием ДБД:

```
database
        % объявление предиката ДБД
        fibDBA (integer, real)
predicates
        % объявление процедурного предиката
        fibBEST(integer, real)
clauses
        fibBEST (1, 1) :- !.
        fibBEST (2, 1) :- !.
        fibBEST (N, F) :-
                % попытаться получить число Фибоначчи из ДБД
                fibDBA (N, F), !.
        % общее правило формирования числа Фибоначчи
        fibBEST (N, F) :-
                % вычислить (N-1)-ое число Фибоначчи
                N1 = N - 1, fibBEST (N1, F1),
                % вычислить (N-2)-ое число Фибоначчи
                N2 = N - 2, fibBEST (N2, F2),
```

```
        % вычислить N-ое число Фибоначчи
        F = F1 + F2,
        % запомнить N-ое число Фибоначчи в ДБД
        asserta (fibDBA (N, F)).
goal   fibBEST (1400, X), write (X), exit.
```

Рис. 6 показывает вычислительный процесс для определения шестого числа Фибоначчи при помощи процедуры **fibBEST**, которая сохраняет промежуточные результаты в ДБД.

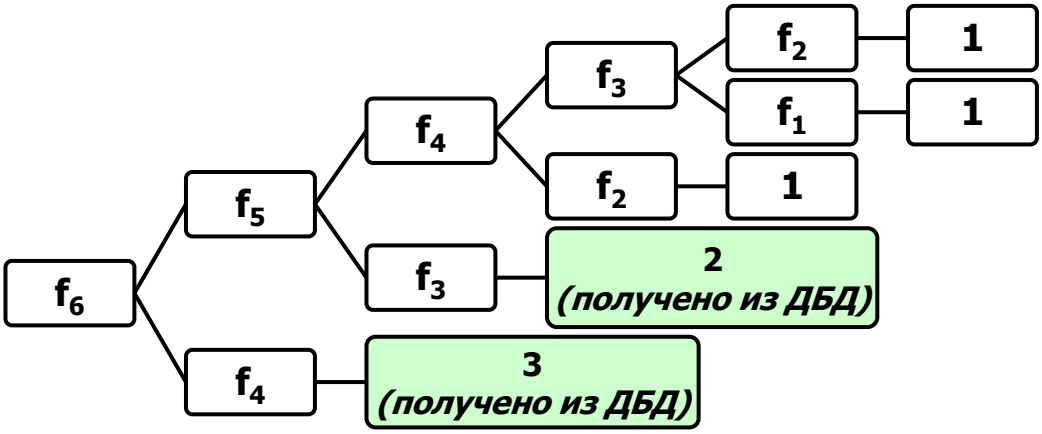

Рис. 6. Вычисление шестого числа Фибоначчи процедурой fibBEST

Процедура **fibBEST** сначала обрабатывает первые два числа Фибоначчи как два особых случая, затем пытается найти в ДБД вычисленное значение **F** для очередного числа **N**. Только после этого процедура переходит к общему правилу формирования очередного числа и сохранению его в ДБД.

Результат **X = 1.71084769E+292** получается за доли секунды.

Сравнение рис. 6 и рис. 5 показывает, что количество вычислений стало меньше. Для больших значений **N** это уменьшение вычислительной сложности более ощутимо.