Объектные технологии конструирования ПО

Тема: Методики оптимизации кода

Лекция №?

?.1 Логика

?.2 Циклы

?.3 Изменение типов данных

?.4 Выражения

?.5 Методы

?.6 Переписывание кода на низком уровне

В чем заключается оптимизация?

Оптимизация кода – это в первую очередь повышение быстродействия, а так же сокращение объема кода.

Инструменты оптимизации кода:

- Рефакторинг;
- Использование стратегий оптимизации.

Прекращение проверки сразу после получения ответа

Пример (выражение): if (5 < x) and (x < 10) then …

Как только вы определили, что *x* больше 5, вторую часть проверки выполнять не нужно.

Некоторые языки поддерживают "сокращенную оценку выражений", при которой компилятор генерирует код, автоматически прекращающий проверку после получения ответа.

Сокращенная оценка выполняется, например, для стандартных операторов C++ и "условных" операторов Java.

Прекращение проверки сразу после получения ответа

Если ваш язык не поддерживает сокращенную оценку, избегайте операторов *and* и *or,* используя вместо них дополнительную логику.

Пример сокращенной оценки:

```
if ( 5 < x ) then
    if ( x < 10 ) then …
```

Прекращение проверки сразу после получения ответа

Пример, в котором цикл продолжает выполняться даже после получения ответа (C++):

```
negativeInputFound = false;
for ( i = 0; i < count; i++ ) {
      if ( input[ i ] < 0 ) {
       negativeInputFound = true;
      }
 }
```

Оптимизация:

- Включите в код оператор *break* после строки *negativeInputFound = true.*
- Если язык не поддерживает оператор *break,* имитируйте его при помощи оператора *goto,* передающего управление первой команде, расположенной после цикла.

Прекращение проверки сразу после получения ответа

- Измените цикл *for* на цикл *while* и проверяйте значение *negativeInputFound* вместе с проверкой того, не превысил ли счетчик цикла значение *count.*
- Измените цикл *for* на цикл *while,* поместите сигнальное значение в первый элемент массива, расположенный после последнего исходного значения, а в условии цикла *while* просто проверяйте, не отрицательно ли значение.

Результаты использования ключевого слова break:

Язык	Время выполнения кода до оптимизации	Время выполнения оптимизированного кода	Экономия времени
C++	4,27	3,68	14%
Java	4,85	3,46	29%

Упорядочивание тестов по частоте

- Упорядочивайте тесты так, чтобы самый быстрый и чаще всего оказывающийся истинным тест выполнялся первым.

- Нормальные случаи следует обрабатывать первыми, а вероятность выполнения неэффективного кода должна быть низкой.

- Этот принцип относится к блокам *case* и цепочкам операторов *if-then-else.*

Упорядочивание тестов по частоте

Пример плохо упорядоченного логического теста (Visual Basic):

```
Select inputCharacter
    Case "+", "="
        ProcessMathSymbol( inputCharacter )
    Case "0" To "9"
        ProcessDigit( inputCharacter )
    Case ",", ".", ":", ";", "!", "?"
        ProcessPunctuation( inputCharacter )
    Case " "
        ProcessSpace( inputCharacter )
    Case "A" To "Z", "a" To "z"
        ProcessAlpha( inputCharacter )
    Case Else
        ProcessError( inputCharacter )
End Select
```

Упорядочивание тестов по частоте

Пример хорошо упорядоченного логического теста (Visual Basic):

```
Select inputCharacter
    Case "A" To "Z", "a" To "z"
        ProcessAlpha( inputCharacter )
    Case " "
        ProcessSpace( inputCharacter )
    Case ",", ".", ":", ";", "!", "?"
        ProcessPunctuation( inputCharacter )
    Case "0" To "9"
        ProcessDigit( inputCharacter )
    Case "+", "="
        ProcessMathSymbol( inputCharacter )
    Case Else
        ProcessError( inputCharacter )
End Select
```

Упорядочивание тестов по частоте

Результаты оптимизации:

Язык	Время выполнения кода до оптимизации	Время выполнения оптимизированного кода	Экономия времени
C++	0,220	0,260	-18%
Java	2,56	2,56	0%
Visual Basic	0,280	0,260	7%

Сравнения быстродействия похожих структур логики

- Такой тип тестирование можно выполнить и для блоков case, и для операторов if-then-else.
- В зависимости от среды любой из подходов может оказаться более выгодным.

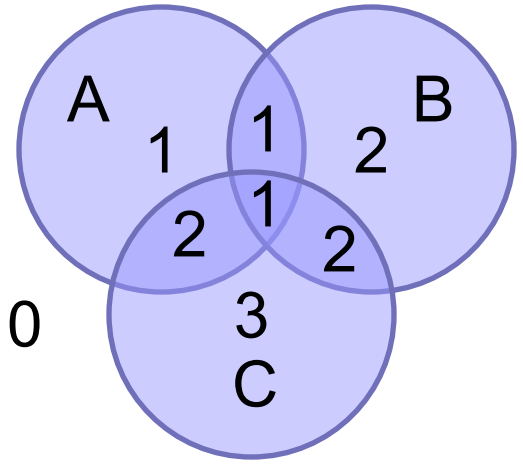
Язык	case	If-then-else	Экономия времени	Соотношение быстродействия
C#	0,260	0,330	-27%	1:1
Java	2,56	0,460	82%	6:1
Visual Basic	0,260	1,00	258%	1:4

Замена сложных выражений на обращене к таблице

Иногда просмотр таблицы может оказаться более быстрым, чем выполнение сложной логической цепи.

Суть сложной сложной цепи обычно сводится к категоризации чего-либо.

Необходимо присвоить чему-то номер категории на основе принадлежности этого чего-то к группам A, B и C:



Замена сложных выражений на обращене к таблице

Пример сложной логической цепи (C++):

```
if ( ( a && ! c ) 11 ( a && b && c ) ) {
    category = 1;
}
else if ( ( b && ! a ) 11 ( a && c && !b ) ) {
    category = 2;
}
else if ( c && !a && !b ) {
    category = 3;
}
else {
    category = 0;
}
```

Замена сложных выражений на обращене к таблице

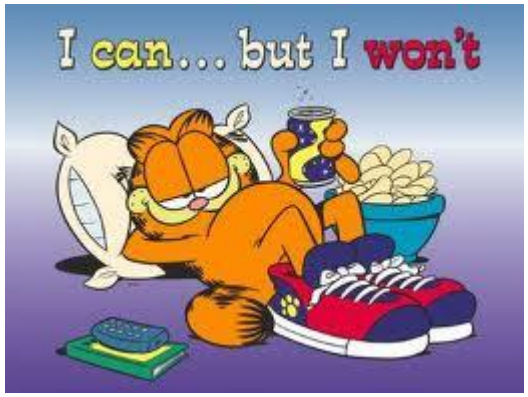
Пример использования таблицы вместо сложной логики (C++):

```
static int categoryTable[ 2 ][ 2 ][ 2 ] = {
            // !b!c   !bc   b!c   bc
            0,    3,    2,    2, // !a
            1,    2,    1,    1  // a
};
…
category = categoryTable[ a ][ b ][ c ];
```

Определение таблицы понять нелегко, поэтому используйте любые комментарии, способные помочь.

Язык	Время выполнения кода до оптимиза-ции	Время выполнения оптимизированного кода	Экономия времени	Соотношение быстродействия
C++	5,04	3,39	33%	1,5:1
Visual Basic	5,21	2,60	50%	2:1

Отложенные вычисления



- Методика отложенных вычислений основана на принципе: “Программа делает что-то, только когда это действительно нужно”.

- Отложенное вычисление похоже на стратегию решения задач «по требованию», при которой работа выполняется максимально близко к тому месту, где нужны ее результаты.

Размыкание цикла

Замыканием (switching) цикла называют принятие решения внутри цикла при каждой его итерации.

- Если во время выполнения цикла решение не изменяется, вы можете **разомкнуть (unswitch)** цикл, приняв решение вне цикла.
- Для размыкания нужно вывернуть цикл наизнанку, т. е. поместить циклы в условный оператор, а не условный оператор внутрь цикла.

Размыкание цикла

Пример замкнутого цикла (C++):

```
for ( i = 0; i < count; i++ ) {
      if ( sumType == SUMTYPE_NET ) {
            netSum = netSum + amount[ i ];
      }
      else {
          grossSum = grossSum + amount[ i ];
     }
}
```

Пример разомкнутого цикла (C++):

```
if (sumType == SUMTYPE_NET ) {
      for ( i = 0; i < count; i++ ) {
        netSum = netSum + amount[ i ];
      }
 } else {
      for ( i = 0; i < count; i++ ) {
        grossSum = grossSum + amount[ i ];
      }
}
```

Язык	Время выполнения кода до оптимизации	Время выполнения оптимизированного кода	Экономия времени
C++	2,81	2,27	19%
Java	3,97	3,12	21%
Visual Basic	2,78	2,77	< 1%
Python	8,14	5,87	28%

Объединение циклов

- Если два цикла работают с одним набором элементов, можно выполнить их объединение (jamming).
- Выгода здесь объясняется устранением затрат, связанных с выполнением дополнительного цикла.

Пример отдельных циклов, которые можно объединить (Visual Basic):

```
For i = 0 to employeeCount - 1
    employeeName( i ) = ""
Next
…
For i = 0 to employeeCount - 1
    employeeEarnings( i ) = 0
Next
```

Объединение циклов

Пример объединенного цикла (Visual Basic):

```
For i = 0 to employeeCount - 1
    employeeName( i ) = ""
    employeeEarnings( i ) = 0
Next
```

Язык	Время выполнения кода до оптимизации	Время выполнения оптимизированного кода	Экономия времени
C++	3,68	2,65	28%
PHP	3,97	2,42	32%
Visual Basic	3,75	3,56	4%

Развертывание цикла

Пример однократного развертывания цикла (Java):

```
i = 0;
while ( i < count - 1 ) {
      a[ i ] = i;
      a[ i + 1 ] = i + 1;
       i = i + 2;
 }
```

//Эти строки обрабатывают случай, который может быть упущен из-за увеличении счетчика цикла на 2, а не на 1.

```
if ( i == count ) {
      a[ count - 1 ] = count - 1;
 }
```

Язык	Время выполнения кода до оптимизации	Время выполнения оптимизированного кода	Экономия времени
C++	1,75	1,15	34%
Java	1,01	0,581	43%
PHP	5,33	4,49	16%
Python	2,51	3,21	-27%

Развертывание цикла

- Целью развертывания (unrolling) цикла является сокращение затрат, связанных с его выполнением.
- Полное развертывание цикла — быстрое решение, эффективное при малом числе элементов, но оно непрактично, если элементов много или вы не знаете заранее, с каким числом элементов вы будете иметь дело.

Пример цикла, допускающего развертывание (Java):

```
i = 0;
while ( i < count ) {
    a[ i ] = i;
    i = i + 1;
 }
```

Минимизация объема работы, выполняемой внутри цикла

- Одной из методик повышения эффективности циклов является минимизация объема работы, выполняемой внутри цикла.
- Если вы можете вычислить выражение или его часть вне цикла и использовать внутри цикла результат вычисления, сделайте это.
- Это хорошая методика программирования, которая иногда улучшает читабельность кода.

Минимизация объема работы, выполняемой внутри цикла

Пример цикла, включающего сложное выражение с указателями (C++):

```
for ( i = 0; i < rateCount; i++ ) {
    netRate[ i ] = baseRate[ i ] * rates->discounts->factors->net;
 }
```

Пример упрощения сложного выражения с указателями (C++):

```
quantityDiscount = rates->discounts->factors->net;
for ( i = 0; i < rateCount; i++ ) {
    netRate[ i ] = baseRate[ i ] * quantityDiscount;
}
```

Язык	Время выполнения кода до оптимизации	Время выполнения оптимизированного кода	Экономия времени
C++	3,69	2,97	19%
C#	2,27	1,97	13%
Java	4,13	2,35	43%

Сигнальные значения

- Если цикл включает проверку сложного условия, время его выполнения часто можно сократить, упростив проверку.

- В случае циклов поиска используется сигнальное значение (sentinel value) — значение, которое располагается сразу после окончания диапазона поиска и непременно завершает поиск.

- Сигнальное значение можно использовать почти в любой ситуации, требующей выполнения линейного поиска, причем не только в массивах, но и в связных списках.

Сигнальные значения

Пример проверки сложного условия цикла (C#):

```
found = FALSE;
i = 0;
//Проверка сложного условия.
while ( ( !found ) && ( i < count ) ) {
    if ( item[ i ] == testValue ) {
     found = TRUE;
    }
    else {
     i++;
    }
 }
if ( found ) { … }
```

Сигнальные значения

Пример использования сигнального значения для ускорения цикла (C#):

```
// Установка сигнального значения с сохранением начальных значений.
initialValue = item[ count ];
//Не забудьте зарезервировать в конце массива место для сигнального значения.
item[ count ] = testValue;
i = 0;
while ( item[ i ] != testValue ) {
        i++;
}
if ( i < count ) { … }
```

Язык	Время выполнения кода до оптимизации	Время выполнения оптимизированного кода	Экономия времени	Соотношение быстродействия
C#	0,771	0,590	23%	1,3:1
Java	1,63	0,912	44%	2:1
Visual Basic	1,34	0,470	65%	3:1

Вложение более ресурсоемкого цикла в менее ресурсоемкий

- Если вы имеете дело с вложенными циклами, подумайте, какой из них должен быть внешним, а какой внутренним.
- Ключ к улучшению цикла в том, что внешний цикл состоит из гораздо большего числа итераций, чем внутренний.

Вложение более ресурсоемкого цикла в менее ресурсоемкий

Пример вложенного цикла, который можно улучшить (Java):

```
for ( column = 0; column < 100; column++ ) {
      for ( row = 0; row < 5; row++ ) {
            sum = sum + tablet row ][ column ];
      }
}
```

Язык	Время выполнения кода до оптимизации	Время выполнения оптимизированного кода	Экономия времени
C++	4,75	3,19	33%
Java	5,39	3,56	34%
PHP	4,16	3,65	12%
Python	3,48	3,33	4%

Снижение стоимости оператора

- Под снижением стоимости (strength reduction) понимают замену дорогой операции на более дешевую, например, умножения на сложение.

- Сложение обычно выполняется быстрее, чем умножение, и, если вы можете вычислить то же число заменив умножение на прибавление значения при каждой итерации цикла, это скорее всего приведет к ускорению выполнения кода.

Снижение стоимости оператора

Пример умножения с использованием индекса цикла (Visual Basic):

```
For i = 0 to saleCount - 1
    commission( i ) = (i + 1) * revenue * baseCommission * discount
Next
```

Пример замены умножения на сложение (Visual Basic):

```
incrementalCommission = revenue * baseCommission * discount
cumulativeCommission = incrementalCommission
For i = 0 to saleCount - 1
    commission( i ) = cumulativeCommission
    cumulativeCommission = cumulativeCommission +
incrementalCommission
Next
```

Язык	Время выполнения кода до оптимизации	Время выполнения оптимизированного кода	Экономия времени
C++	4,33	3,80	12%
Visual Basic	3,54	1,80	49%

Использование целых чисел вместо чисел с плавающей точкой

Сложение и умножение целых чисел, как правило, выполняются быстрее, чем аналогичные операции над числами с плавающей запятой.

Пример неэффективного цикла с индексом с плавающей запятой (Visual Basic):

```
Dim x As Single
For x = 0 to 99
    a( x ) = 0
Next
```

Пример эффективного цикла с целочисленным индексом (Visual Basic):

```
Dim i As Integer
For i = 0 to 99
    a( i ) = 0
Next
```

Язык	Время выполнения кода до оптимизации	Время выполнения оптимизированного кода	Экономия времени
C++	2,80	0,801	71%
Visual Basic	6,84	0,280	96%

Использование массивов с минимальным числом измерений

- Использовать массивы, имеющие несколько измерений, накладно.

- Если вы сможете структурировать данные так, чтобы их можно было хранить в одномерном, а не двумерном или трехмерном массиве, то ускорите выполнение программы.
- При инициализации массива из 50 строк и 20 столбцов этот код выполняется вдвое дольше, чем код инициализации аналогичного одномерного массива, сгенерированный компилятором Java.

Использование массивов с минимальным числом измерений

Пример стандартной инициализации двумерного массива (Java):

```
for ( row = 0; row < numRows; row++ ) {
      for ( column = 0; column < numColumns; column++ ) {
matrix[ row ][ column ] = 0;
      }
}
```

Пример одномерного представления массива (Java):

```
for ( entry = 0; entry < numRows * numColumns; entry++ ) {
        matrix[ entry ] = 0;
}
```

Язык	Время выполнения кода до оптимизации	Время выполнения оптимизированного кода	Экономия времени	Соотношение быстродействия
C++	8,75	7,82	11%	1:1
C#	3,28	2,99	9%	1:1
Java	7,78	4,14	47%	2:1
PHP	6,24	4,10	34%	1,5:1

Минимизация числа обращений к массивам

- Кроме минимизации числа обращений к двумерным или трехмерным массивам часто выгодно минимизировать число обращений к массивам вообще.
- Подходящий кандидат для применения этой методики – цикл, в котором повторно используется один и тот же элемент массива.

Пример необязательного обращения к массиву внутри цикла (C++):

```
for ( discountType = 0; discountType < typeCount; discountType++ ) {
      for ( discountLevel = 0; discountLevel < levelCount; discountLevel++ ) {
            rate[ discountLevel ] = rate[ discountLevel ] * discount[ discountType ];
      }
}
```

Минимизация числа обращений к массивам

Пример вынесения обращения к массиву за пределы цикла (C++):

```
for ( discountType = 0; discountType < typeCount; discountType++ ) {
    this.Discount = discount[ discountType ];
     for ( discountLevel = 0; discountLevel < levelCount; discountLevel++ ) {
       rate[ discountLevel ] = rate[ discountLevel ] * thisDiscount;
     }
}
```

Язык	Время выполнения кода до оптимизации	Время выполнения оптимизированного кода	Экономия времени
C++	32,1	34,5	-7%
C#	18,3	17,0	7%
Visual Basic	23,2	18,4	20%

Использование дополнительных индексов

- Использование дополнительного индекса предполагает добавление данных, связанных с основным типом данных и повышающих эффективность обращений к нему. Связанные данные можно добавить к основному типу или хранить в параллельной структуре.
- Пример использования индексов:

  В языке С строки заканчиваются нулевым байтом, а в Visual Basic их длина хранится в начальном байте.

  Чтобы определить длину строки в С нужно начать с начала строки и продвигаться по ней, а в Visual Basic, нужно просто прочитать байт.

Независимая параллельная структура индексации

- Иногда выгоднее работать с индексом типа данных, а не с самим типом данных.
- Если элементы типа данных велики или их накладно перемещать, сортировка и поиск по индексам будут выполняться быстрее, чем непосредственные операции над данными.
- Если каждый элемент данных велик, создавайте вспомогательную структуру, состоящую из ключевых значений и указателей на подробную информацию.
- Если различие размеров элемента структуры данных и элемента вспомогательной структуры велико, элемент-ключ храните в памяти, а сами данные – на внешнем носителе.

Кэширование

- Кэширование – это такой способ хранения нескольких значений, при котором значения, используемые чаще всего, получить легче, чем значения, используемые реже.
- Если программа случайным образом читает записи с диска, метод может хранить в кэше записи, считываемые наиболее часто.
- Возможно кэшировать результаты ресурсоемких вычислений, особенно если их параметры просты.

Пример метода, напрашивающегося на кэширование (Java):

```
double Hypotenuse( double sideA, double sideB) {
        return Math.sqrt( ( sideA * sideA ) + ( sideB * sideB ) );
}
```

Аглебраические тождества

Алгебраические тождества иногда позволяют заме- нить дорогие операции на более дешевые.

Пример:

not a and not b

not (a or b)

Выбрав второе выражение вместо первого, вы сэкономите одну операцию *not.*

Язык	Время выполнения кода до оптимизации	Время выполнения опти-мизированного кода	Экономия времени	Соотношениеб ыстродейст-вия
C++	7,43	0,010	99,9%	750:1
Visual Basic	4,59	0,220	95%	20:1
Python	4,21	0,401	90%	10:1

Кэширование

Пример кэширования для предотвращения дорогих вычислений (Java):

```
private double cachedHypotenuse = 0; private double cachedSideA = 0;
private double cachedSideB = 0;
public double Hypotenuse( double sideA, double sideB ) {
    if ( ( sideA == cachedSideA ) && ( sideB == cachedSideB ) ) {
      return cachedHypotenuse;
    }
    // Вычисление новой гипотенузы и ее кэширование.
    cachedHypotenuse = Math.sqrt( ( sideA * sideA ) + ( sideB * sideB ) );
    cachedSideA = sideA;
    cachedSideB = sideB;
    return cachedHypotenuse;
}
```

Язык	Время выполнения кода до оптимизации	Время выполнения оптимизированного кода	Экономия времени	Соотношениеб ыстродейст-вия
C++	4,06	1,05	74%	4:1
Java	2,54	1,40	45%	2:1
Python	8,16	4,17	49%	2:1

Снижение стоимости операций

Подразумевает замену дорогой операции более дешевой.

- замена умножения сложением;
- замена возведения в степень умножением;
- замена тригонометрических функций их эквивалентами;
- замена типа *longlong* на *long* или *int* (следите при этом за аспектами производительности, связанными с применением целых чисел естественной и неестественной длины);
- замена чисел с плавающей запятой числами с фиксированной точкой или целые числа;
- замена чисел с плавающей запятой с удвоенной точностью числами с одинарной ТОЧНОСТЬЮ;
- замена умножения и деления целых чисел на два операциями сдвига.

Снижение стоимости операций

Пример вычисления многочлена (Visual Basic):

```
value = coefficient(0) For power = 1 To order
value = value + coefficient(power) * x*power Next
```

Пример снижения стоимости вычисления многочлена (VisualBasic):

```
value = coefficient( 0 )
powerOfX = x
For power = 1 to order
value = value + coefficient( power ) * powerOfX
powerOfX = powerOfX * x Next
```

Язык	Время выполнения кода до оптимизации	Время выполнения оптимизированного кода	Экономия времени	Соотношениеб ыстродейст-вия
Python	3,24	2,60	20%	1:1
Visual Basic	6,26	0,160	97%	40:1

Инициализация во время компиляции

Если вы вызываете метод, передавая ему в ка- честве единственного аргумента константу попробуйте сначала вычислить нужное значение, присвоить его константе и избежать вызова метода.

Это также справедливо для умножения, деления, сложения и других операций.

Пример метода, вычисляющего двоичный логарифм с использованием системных методов (C++):

```
unsigned int Log2( unsigned int x ) {
    return (unsigned int) ( log( x ) / log( 2 ) );
}
```

Недостатки системных методов

Системные методы дороги и часто обеспечивают избыточную точность. Если не требуется такая точность, нет смысла тратить на нее время.

Пример метода, возвращающего значение двоичного логарифма (C++):

```
unsigned int Log2( unsigned int x ) {
    if (x< 2 ) return 0 ;
    if(x< 4 ) return 1 ;
    if(x< 8 ) return 2 ;
    if(x< 16 ) return 3 ;
    if(x< 32 ) return 4 ;
    if(x< 64 ) return 5 ;
    ….
    if ( x < 2147483648 ) return 30;
    return 31 ;
}
```

Инициализация во время компиляции

Пример метода, вычисляющего двоичный логарифм с использованием системного метода и константы (C++):

```
const double L0G2 = 0.69314718;
unsigned int Log2( unsigned int x ) {
        return (unsigned int) ( log( x ) / L0G2 );
}
```

Язык	Время выполнения кода до оптимизации	Время выполнения оптимизированного кода	Экономия времени
C++	9,66	5,97	38%
Java	17,0	12,3	28%
PHP	2,45	1,50	39%

Недостатки системных методов

Язык	Время выполнения кода до оптимизации	Время выполнения оптимизированного кода	Экономия времени	Соотношениеб ыстродейст-вия
C++	9,66	0,662	93%	15:1
Java	17,0	0,882	95%	20:1
PHP	2,45	3,45	-41%	2:3

Пример метода, определяющего примерное значение двоичного логарифма с использованием оператора сдвига вправо (C++):

```
unsigned int Log2( unsigned int x ) {
    unsigned int i = 0;
    while ((x=(x"1))!=0){
        i++;
    }
    return i ;
}
```

Использование констант корректного типа

- Используйте именованные константы и литералы, имеющие тот же тип, что и переменные, которым они присваиваются.
- Если константа и соответствующая ей переменная имеют разные типы, перед присвоением константы переменной компилятор должен будет выполнить преобразование типа.
- Хорошие компиляторы преобразуют типы во время компиляции, чтобы не снижалась производительность в период выполнения программы.
- Менее эффективные компиляторы или интерпретаторы генерируют код, преобразующий типы в период выполнения.

Предварительное вычисление результатов

- При низкоуровневом проектировании часто приходится решать, вычислять ли результаты по ходу дела или лучше вычислить их один раз, сохранить и извлекать по мере надобности. Если результаты используются много раз, используйте второй вариант.
- Этот выбор проявляется несколькими способами. На самом простом уровне вы можете вычислить часть выражения вне цикла вместо вычисления его внутри.
- На более сложном уровне вы можете вычислить табличные данные один раз при запуске программы и использовать их позднее; вы также можете сохранить результаты в файле данных или встроить их в программу

Предварительное вычисление результатов

Пример сложного выражения, которое можно вычислить предварительно (Java):

```
double ComputePayment( long loanAmount, int months, double interestRate) {
    return loanAmount /
        ( ( 1.0 - Math.pow( ( 1.0 + ( interestRate /12.0)), -months ) ) / ( interestRate / 12.0 ) );
 }
```

Предварительное вычисление результатов

Пример предварительного вычисления сложного выражения (Java):

```
double ComputePayment( long loanAmount, int months, double interestRate ) {
  //Новая переменная interestIndex используется как индекс массива
loanDivisor.
      int interestIndex = Math.round( ( interestRate - LOWEST_RATE ) *
GRANULARITY * 100.00 );
      return loanAmount / loanDivisor[ interestIndex ][ months ];
}
```

Язык	Время выполнения кода до оптимизации	Время выполнения оптимизированного кода	Экономия времени	Соотношениебыстродействия
Java	2,97	0,251	92%	10:1
Python	3,86	4,63	-20%	1:1

Предварительное вычисление результатов

- вычисление результатов до выполнения программы и связывание их с константами во время компиляции;
- вычисление результатов до выполнения программы и присвоение их переменным, используемым в период выполнения;
- вычисление результатов до выполнения программы и сохранение их в файле, загружаемом в период выполнения;

Предварительное вычисление результатов

- однократное вычисление результатов при запуске программы и их использование во всех последующих случаях;
- вычисление как можно большего числа значений до начала цикла, позволяющее свести к минимуму объем работы, выполняемой внутри цикла;
- вычисление результатов при возникновении первой потребности в них и их сохранение, позволяющее получить результаты, когда они понадобятся снова.

Устранение часто используемых подвыражений

Если какое-то выражение повторяется в коде несколько раз, присвойте его значение переменной и используйте переменную вместо вычисления выражения в нескольких местах.

Пример часто используемого подвыражения (Java):

```
payment = loanAmount / ((1.0 - Math.pow( 1.0 + ( interestRate /12.0), - months ) )
        / ( interestRate / 12.0 ));
```

Пример устранения часто используемого подвыражения (Java):

```
monthlyInterest = interestRate / 12.0;
payment = loanAmount / (( 1.0 - Math.pow( 1.0 + monthlyInterest, - months ) )
        / monthlyInterest );
```

Язык	Время выполнения кода до оптимизации	Время выполнения оптимизированного кода	Экономия времени
Java	2,94	2,83	4%
Python	3,91	3,94	-1%

О методах

- Одним из самых эффективных способов оптимизации кода является грамотная декомпозиция программы на методы.
- Небольшие, хорошо определенные методы делают программу компактнее, устраняя повторяющиеся фрагменты кода.
- Методы упрощают оптимизацию, потому что рефакторинг одного метода улучшает все методы, которые его вызывают.
- Небольшие методы относительно легко переписать на низкоуровневом языке.
- Объемные методы понять сложно, а после переписывания их на низкоуровневом языке вроде ассемблера это вообще невыполнимо.

Встраивание методов

- В некоторых случаях можно сэкономить несколько нано секунд, встроив код метода в программу, используя ключевое слово *inline* языка C++ или аналогичную возможность.
- Если ваш язык не поддерживает *inline*, но имеет препро цессор макросов, вы можете встраивать код при помощи макроса, включая и выключая встраивание по требованию.
- Встроив код, вы можете как улучшить производитель-ность, так и ухудшить ее.

Язык	Время выполнения кода до оптимизации	Время выполнения опти-мизированного кода	Экономия времени
C++	0,471	0,431	8%
Java	13,1	14,4	-10%

Переписывание кода на низкоуровневом языке

1. Напишите все приложение на высокоуровневом языке.

2. Выполните полное тестирование приложения и проверьте его корректность.

3. Если производительность недостаточна, выполните профилирование приложения с целью выявления горячих точек. Так как около 50% времени выполнения программы обычно приходится примерно на *5%* кода, горячими точками обычно будут небольшие фрагменты программы.

4. Перепишите несколько небольших фрагментов на низкоуровневом языке для повышения общей производительности программы.

В заключении…

- Оптимизация кода подразумевает нахождение компромисса между сложностью, удобочитаемостью, простотой и удобством сопровождения программы, с одной стороны, и желанием повы- сить производительность – с другой.

- Необходимое при оптимизации перепрофилирование кода приводит к значительному росту затрат на сопровождение программы.
- Хороший способ против преждевременной оптимизации и для созданию ясного кода: требуйте, чтобы оптимизация приводила к *измеримому улучшению.* Если оптимизация оправдывает профилирование кода и оценку результатов, ее следует выполнить, если будет показано, что она работает.

В заключении…

- Но если оптимизация не оправдывает проведения профилирования, она не может оправдать ухудшения удобочитаемости, удобства сопровождения и других характеристик кода.
- Влияние неоцененной оптимизации кода на производительность в лучшем случае может быть только теоретическим, тогда как ее влияние на удобочитаемость столь же определенно, сколь пагубно.

Выводы

- Результаты конкретных видов оптимизации во многом зависят от языка, компилятора и среды. Не оценив резуль татов оптимизации, вы не сможете сказать, помогает она программе или вредит.
- Первый вид оптимизации часто далеко не самый лучший. Обнаружив эффективный вид оптимизации, продолжайте пробовать и, возможно, найдете еще более эффективный.
- Одни считают, что оптимизация настолько ухудшает надежность и удобство сопровождения программы, что ее вообще выполнять не следует. Другие думают, что при соблюдении должной предосторожности она прино- сит пользу. Если вы решите использовать методики, описанные в этой главе, будьте внимательны и осторожны.

Ссылки

Спасибо за внимание!