Институт кибернетики

ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

АЛГОРИТМИЧЕСКИЕ СТРАТЕГИИ

Галина Ивановна Шкатова

к.т.н., доцент

Институт кибернетики

АЛГОРИТМИЧЕСКИЕ СТРАТЕГИИ

Введение

При создания компьютерной программы, предназначенной для решения какой-либо практической задачи одним из важнейших этапов процесса создания является этап разработки алгоритма.

К настоящему времени разработано множество алгоритмических стратегий, определены ключевые принципы, лежащие в основе этих стратегий, что позволяет их применять как универсальный инструментарий для широкого класса задач.

Создано множество алгоритмов, которые могут служить фундаментом современного компьютерного программирования и которые можно использовать как строительные блоки при программировании.

При изложении дальнейшего материала основной акцент делается на понимании идей представляемой алгоритмической стратегии, а не на механическом рассмотрении работы того или иного алгоритма.

Очевидно, что для решения одной и той же задачи можно создавать разные алгоритмы в зависимости от выбора алгоритмической стратегии.

Конечно же, для того чтобы успешно использовать алгоритмы, созданные в соответствии с выбранной стратегией, недостаточно просто подготовить код программы.

Необходимо знать, как различные алгоритмы ведут себя в разных ситуациях. В конечном итоге именно эта информация определяет выбор стратегии, а также позволяет критически оценивать новые алгоритмы.

Институт кибернетики

АЛГОРИТМИЧЕСКИЕ СТРАТЕГИИ

1. Методы «грубой силы».
2. Жадные алгоритмы.
3. Алгоритмы «разделяй и властвуй» (декомпозиции).
4. Эвристические алгоритмы.
5. Алгоритмы с возвратом.
6. Методы проб и ошибок.
7. Муравьиные алгоритмы.
8. Генетические алгоритмы.
9. Алгоритмы численных приближений.
10. Сравнение с образцом.

Институт кибернетики

Метод грубой силы

Из Википедии:

Метод «грубой силы» (от англ. *brute force*). именуется также методом решения «в лоб».
Метод грубой силы представляет собой прямой подход к решению задачи, обычно основанный непосредственно на формулировке задачи и определениях используемых ею концепций.
«Сила» в определении стратегии – сила компьютера, а не сила интеллекта, т.е. сила из пословицы: «Сила есть — ума не надо». Перефразировать определение данной стратегии можно проще: «Нечего думать, надо действовать!».
К методам «грубой силы» относят методы решения задачи путем перебора всех возможных вариантов. Сложность полного перебора зависит от размерности пространства всех возможных решений задачи. Если пространство решений очень велико, то полный перебор может не дать результатов в течение месяцев и даже лет.
Метод грубой силы используется для многих элементарных, но важных алгоритмических задач, таких как: вычисление суммы n чисел, поиск наибольшего элемента в списке, пузырьковая сортировка и т.д.

Пример 1. Задача поиска

Поиск является одним из наиболее часто встречаемых действий в программировании.

В общем случае под «поиском» понимается процесс нахождения конкретной информации в ранее созданном множестве данных. Обычно данные представляют собой записи, каждая из которых имеет хотя бы один ключ. *Ключ поиска* – это поле записи, по значению которого происходит поиск. Ключи используются для отличия одних записей от других. Целью поиска является нахождение всех записей (если они есть) с данным значением ключа.

Существует множество различных алгоритмов поиска, которые принципиально зависят от способа организации данных. У каждого алгоритма поиска есть свои преимущества и недостатки. Поэтому важно выбрать тот алгоритм, который лучше всего подходит для решения конкретной задачи.

Представителем методов грубой силы является метод, который носит название ***последовательного или линейного поиска.***

Идея этого метода заключается в следующем. Множество элементов просматривается последовательно в некотором порядке, гарантирующем, что будут просмотрены все элементы множества (например, слева направо). Если в ходе просмотра множества будет найден искомый элемент, просмотр прекращается с положительным результатом; если же будет просмотрено все множество, а элемент не будет найден, алгоритм должен выдать отрицательный результат.

Описание функции на C++, описывающей процесс линейного поиска может иметь вид:

```
int LinearSearch(int *x, int k, int key)
{
  // key – ключ поиска (входной параметр)
  // x –целочисленный массив(входной параметр)
  // k – количество элементов в массиве(входной параметр)
  // i – вспомогательная переменная
  int i = 0; //устанавливается начальное значение индекса массива
  for ( i = 0 ; i < k ; i++ ) // цикл по изменению индеса
    if ( x[i] == key ) // проверка на совпадение с ключом, если совпал, то
      break;  // выход из цикла ( в этом случае значение i не достигло конечного k
  return i < k ? i : -1;//вернуть найденный номер или значение -1, если нет такового
}
```

В качестве альтернативы продемонстрированному методу грубой силы можно предложить метод бинарного (двоичного, дихотомического) поиска. В нем поиск заданного элемента осуществляется на предварительно упорядоченном множестве данных. Поиск производится путем неоднократного деления этого множества на две части таким образом, что искомый элемент попадает в одну из этих частей.

Пример 2. Возведение числа в неотрицательную степень

Метод грубой силы предлагает подход, основанный непосредственно на формулировке задачи, а именно, на определении значения операции, которая носит название «возведение в степень»:

$$a^n = \underbrace{a \cdot a \cdot \ldots \cdot a}_{n \text{ раз}}$$

Из этого определения следует простой алгоритм, который сводится к одному циклу – накоплению произведения. Этот алгоритм не нуждается в дополнительном пояснении и представлен в описании функции с именем pow:

```
long int pow(int a, int n)
{
// P – вспомогательная переменная для накопления произведения
   long int P =1; //установка начального значения для произведения
   for(int i = 1; i<n; i++) //цикл по числу сомножителей
      P = P*a;//накопление произведения
   return P; // вернуть результат
}
```

В качестве примера алгоритма, в котором не используется метод грубой силы - при вычислении использовать битовое представление показателя степени n.

В соответствии с логикой этого алгоритма , например, для возведения числа A в 9 степень A^9 потребуется всего четыре операции умножения:

сначала найти: A^2, A^4, A^8 (3 умножения), а затем найти: $A \cdot A^8$ (одно умножение). А для возведения A^{87} – всего 17 операций.

Подробное описание алгоритма приведено в [1].

Институт кибернетики

Пример 3. Умножение матриц

Особенность любой задачи по обработке матрицы связана с тем, что для доступа к какому-либо элементу матрицы, следует задать две его «координаты» – номер строки и номер столбца, которые указывают, где находится данный элемент. Если посмотреть на запись элемента a[i][j], то его первая координата i задает номер строки матрицы, а j задает номер столбца. Большинство «матричных» задач связано с обработкой элементов, которая заключается в том, что для решения задачи требуется организовать просмотр/изменение всех элементов матрицы.

Порядок, в котором берутся элементы матрицы, определяется порядком изменения индексов элементов. Для изменения индексов, а их два, необходимо организовать два цикла, а точнее – двойной цикл или цикл в цикле. Один цикл будет перебирать первый индекс, то есть менять номер строки, а второй цикл – изменять второй индекс, то есть менять номер столбца. Если внешний цикл организован по номеру строки, а внутренний – по номеру столбца, то перебор элементов матрицы будет в порядке – по строчкам. Иначе – по столбикам. Так, в конструкции вида

```
for (int i=0; i<FRowCount; i++) // перебор строк
  for (int j=0; j<FColCount; j++) // перебор столбцов
     {
        // обработка a[i][j]
     }
```

реализован доступ к элементам матрицы «по строкам», так как здесь второй индекс меняется быстрее, чем первый. А в конструкции вида:

```
for (int j=0; j<FColCount; j++) // перебор столбцов
  for (int i=0; i<FRowCount; i++) // перебор строк
     {
          // обработка a[i][j]
     }
```

реализован доступ к элементам матрицы «по столбцам».

АЛГОРИТМИЧЕСКИЕ СТРАТЕГИИ

Известно, что можно перемножать матрицы только в том случае, если число столбцов в первой матрице, то есть левом сомножителе, равно числу строк во второй матрице, то есть правом сомножителе. Схематически это условие можно записать в виде:

A(m×n)·B(n×k) = C(mXk).

Здесь m - число строк в матрице A, n - число столбцов в матрице A и число строк в матрице B, k- число столбцов в матрицах B и C.

Формулу для получения элемента, расположенного в *i*-строке и *j*-столбце матрицы-произведения можно записать в виде:

$$C_{i,j} = \sum_{k=1}^{n} A_{i,k} \cdot B_{k,j} .$$

Для программирования всего процесса вычисления всех элементов результирующей матрицы необходимо использовать три цикла: два из них перебирают индексы элементов матрицы-произведения, а третий – реализует вычисление формулы:

```
for i=1 to m do  //цикл по номеру строки
   for j=1 to k do //цикл по номеру столбца
     begin
       C[i,j]=0; //подготовка ячейки для суммирования
       for k=1 to n do // цикл по числу слагаемых
         C[i,j]=C[i,j]+A[i,k]*B[k,j]  // формула
     end
```

Процесс перемножения матриц, реализованное с использованием представленной формулы, которая следует из определения операции умножения, является представителем стратегии «грубой силы».

На первый взгляд это минимальный объем работы, необходимый для перемножения двух матриц. Однако исследователям не удалось доказать минимальность, и в результате они обнаружили другие алгоритмы, умножающие матрицы более эффективно, например: алгоритм Винограда, алгоритм Штрассена [2].

Институт кибернетики

АЛГОРИТМИЧЕСКИЕ СТРАТЕГИИ

Хотя методы грубой силы редко дают искусные или эффективные алгоритмы, его рассмотрение нельзя опустить, поскольку данный метод представляет собой важную стратегию разработки алгоритмов.

1) Во-первых, в отличие от других стратегий, метод грубой силы применим к очень широкому диапазону задач. Похоже, это единственный подход, для которого существенно сложнее указать задачу, для решения которой он неприемлем.

2) Во-вторых, для некоторых важных задач метод грубой силы дает вполне рациональные алгоритмы.

3) В-третьих, стоимость разработки более эффективного алгоритма может оказаться неприемлемой, если требуется решить всего несколько экземпляров задачи, а алгоритм, основанный на грубой силе, позволяет их решать за приемлемое время.

4) В-четвертых, даже будучи неэффективным в общем случае, метод грубой силы может оказаться полезен для решения небольших по размерам экземпляров задач.

5) Наконец, алгоритм, основанный на грубой силе, может служить для важных теоретических и дидактических целей, например, мерилом для определения эффективности других алгоритмов для решения данной задачи.

Жадные алгоритмы

Жадный алгоритм (*Greedy algorithm*) — алгоритм, заключающийся в принятии локально оптимального решения на каждом его этапе, допуская, что конечное решение также окажется оптимальным.

Рассмотрим небольшую "детскую" задачу. Допустим, что у нас есть монеты достоинством 50, 10, 5 копеек и 1 копейка и нужно вернуть сдачу в 63 копейки наименьшим количеством монет.

Почти не раздумывая, мы преобразуем эту величину в одну монету по 50 копеек, одну монету в 10 копеек и три монеты по одной копейке. Нам не только удалось быстро определить перечень монет нужного достоинства, но и, по сути, мы составили самый короткий список монет требуемого достоинства.

Алгоритм заключался в выборе монеты самого большого достоинства (50 копеек), но не больше 63 копеек, добавлению ее в список сдачи и вычитанию ее стоимости из 63 (получается 13 копеек). Затем, снова выбираем монету самого большого достоинства, но не больше остатка (13 копеек): этой монетой опять оказывается монета в 10 копеек. Эту монету мы опять добавляем в список сдачи, вычитаем ее стоимость из остатка и т.д.

Обратите внимание, что алгоритм для определения сдачи обеспечивает в целом оптимальное решение лишь вследствие особых свойств монет. Если бы у нас были монеты достоинством 1 копейка, 5 и 11 копеек и нужно было бы дать сдачу 15 копеек, то "жадный" алгоритм выбрал бы сначала монету достоинством 11 копеек, а затем четыре монеты по одной копейке, т.е. всего пять монет. Однако, в данном случае можно было бы обойтись тремя монетами по 5 копеек.

Из примера следует, что не каждый "жадный" алгоритм позволяет получить оптимальный результат в целом. Как нередко бывает в жизни, "жадная стратегия" подчас обеспечивает лишь сиюминутную выгоду, в то время как в целом результат может оказаться неблагоприятным.

АЛГОРИТМИЧЕСКИЕ СТРАТЕГИИ

Функция, предназначенная для возвращения сдачи из S копеек минимальным количеством монет в набора из m штук: x[0], x[1], …, x[m-1] «жадным» методом (предполагается, что монеты располагаются в массиве в порядке убывания достоинства) может быть представлена в виде:

```
void Money(int S, int*x, int*m, int n)// реализуется жадный алгоритм
{
//   S -  величина сдачи - входной параметр
//   x -   указатель на массив, в котором хранятся достоинства
//  разменных монет  - входной параметр
 //   m -  указатель на массив, в котором будут  храниться
//        количества монет каждого достоинства – выходной параметр
//   n -  количество монет-номиналов – входной параметр
     int i=0;  //начальное значение номера монеты
     while (i<n) // пока монеты не будут исчерпаны
       {
    m[i] = S/x[i];   //вычислить кол-во монет x[i] –
                     // это целочисленная операция!!!
    S = S-x[i]*m[i];   //оставшаяся сдача
    i++;          //изменить номер
       }
}
```

Фрагмент кода с вызовом функции может иметь вид:

```
int a[4] = {50,10,5,1};
int k[4] = {0};
Money(63,a,k,4);
```

Как альтернативу «жадному» алгоритму (т.к. есть пример, что оптимальное решение достигается в частных случаях), можно предложить метод «грубой силы» и свести решение к перебору вариантов сдачи с проверкой на оптимум. Но это решение будет значительно более трудоемким.

Институт кибернетики

Алгоритмы «разделяй и властвуй» (декомпозиции)

Возможно, самым важным и наиболее широко применимым методом проектирования эффективных алгоритмов, является метод, называемый методом "разделяй и властвуй" (или методом декомпозиции, или методом разбиения).

Этот метод предполагает такую декомпозицию (разбиение) задачи размера n на более мелкие задачи, что на основе решений этих более мелких задач, можно легко получить решение исходной задачи и работают по следующей схеме[3]:

- Экземпляр задачи разбивается на несколько меньших экземпляров той же задачи, в идеале одинакового размера (но необязательно и не всегда).
- Решаются меньшие экземпляры задачи (обычно рекурсивно, хотя иногда для небольших экземпляров применяется какой-либо другой алгоритм).
- При необходимости решение исходной задачи находится путем комбинации решений меньших экземпляров.

На основе этого метода построены многие эффективные алгоритмы, хотя к некоторым задачам он может быть неприменим, а в других случаях давать худшие результаты, чем иное более простое алгоритмическое решение.

Если говорить о быстроте разработки, то ему вообще нет равных. Именно легкость разработки алгоритмов по методу декомпозиции обусловила огромную популярность этого метода. Метод декомпозиции дал кибернетике ряд очень важных и эффективных алгоритмов.

АЛГОРИТМИЧЕСКИЕ СТРАТЕГИИ

Пример: Пусть требуется вычислить сумму чисел: $S = a_0 + a_1 + \ldots + a_{n-1}$

Решение методом «грубой силы»:

```
int Sum(int*a, int n)
{
  int S=0;
  for(int i=0; i<n; i++)
     S = S+a[i];
  return S;
}
```

При решении методом декомпозиции исходная задача – «Сумма n элементов» разбивается на две суммы с числом элементов n/2: $a_0 + a_1 + \ldots + a_{n-1} = (a_0 + \ldots + a_{n/2-1}) + (a_{n/2} + \ldots + a_{n-1})$

Каждая из сумм по той же схеме разбивается на 2 суммы и т.д., пока в очередной сумме не останется один элемент. Например:

```
4+5+1+9+13+11+7 =(4+5+1)+(9+13+11+7) = [(4)+(5+1)]+[(9+13)+(11+7)]
=[(4)+(5)+(1)]+[(9)+(13)]+[(11)+(7)] = 50
```

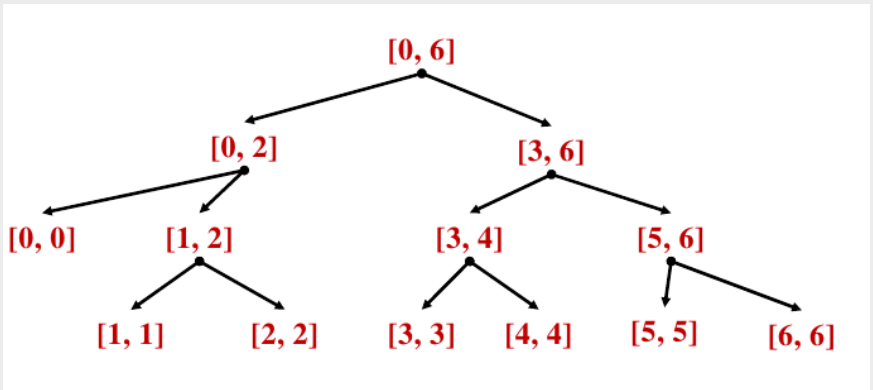
Сумма сводится к сумме, но с другим числом элементов – элемент рекурсии.

```
int Sum(int*a, int ltft, int right)
{
  // left -   левая граница индексов,right – правая граница
индексов
  if(left==right) return a[left];//если один элемент
  int k = (r-left+1)/2;//значение среднего индекса
  return  Sum(a,left, left+k-1) + Sum(a, left+k, right) ;
}
```

Институт кибернетики

Структура рекурсивных вызовов для примера сложения:

$$S = a_0 + a_1 + \ldots + a_6 = 4 + 5 + 1 + 9 + 13 + 11 + 7$$



Здесь в квадратных скобках указаны значения левых и правых индексов суммируемых элементов

АЛГОРИТМИЧЕСКИЕ СТРАТЕГИИ

Примеры алгоритмов, основанных на методе «разделяй и властвуй» – декомпозиции:

- Сортировка слиянием.
- Быстрая сортировка.
- Бинарный поиск.
- Обход двоичного дерева.
- Решение задачи о поиске выпуклой оболочки.
- Умножение матриц методом Штрассена.
- И др.

АЛГОРИТМИЧЕСКИЕ СТРАТЕГИИ

Эвристические алгоритмы

Из Википедии:

Эври́стика (от др. греч. εὑρίσκω (heuristiko), лат.. Evrica — «отыскиваю», «открываю») — отрасль знания, изучающая творческое, неосознанное мышление человека.

Эвристика связана с психологией, физиологией высшей нервной деятельности, кибернетикой и другими науками, но сама как наука ещё полностью не сформировалась.

Эвристический алгоритм —алгоритм решения задачи, не имеющий строгого обоснования, но, тем не менее, дающий приемлемое решение задачи в большинстве практически значимых случаев.

Эвристический алгоритм — это алгоритм решения задачи, правильность которого для всех возможных случаев не доказана, но про который известно, что он даёт достаточно хорошее решение в большинстве случаев. В действительности может быть даже известно (то есть доказано) то, что эвристический алгоритм формально неверен. Его всё равно можно применять, если при этом он даёт неверный результат только в отдельных, достаточно редких и хорошо выделяемых случаях, или же даёт неточный, но всё же приемлемый результат.

Проще говоря, эвристика — это не полностью математически обоснованный (или даже «не совсем корректный»), но при этом практически полезный алгоритм.

Важно понимать, что эвристика, в отличие от корректного алгоритма решения задачи, обладает следующими особенностями:

- Она не гарантирует нахождение лучшего решения.
- Она не гарантирует нахождение решения, даже если оно заведомо существует (возможен «пропуск цели»).
- Она может дать неверное решение в некоторых случаях.

Институт кибернетики

Применение эвристических алгоритмов

Эвристические алгоритмы широко применяются:

- В практике принятия управленческих решений.
- Для решения задач высокой вычислительной сложности. В таких случаях, например, вместо алгоритма, в котором реализуется полный перебор вариантов, занимающий существенное время, а иногда технически невозможный, применяется значительно более быстрый, но недостаточно теоретически обоснованный алгоритм.
- В областях искусственного интеллекта.
- В задачах распознавания образов.
- По причине отсутствия общего решения поставленной задачи.
- Применяются в антивирусных программах, компьютерных играх и т. д.

По большому счету многие из разрабатываемых программ носят эвристический характер – не во всех случаях программисты обеспокоены проблемами, связанными с доказательством правильности их программ. Потому остановимся на примерах эвристических методов, которые используются в практике принятия управленческих решений:

1. Метод «Мозгового штурма».
2. Метод ключевых вопросов.
3. Метод ассоциаций.
4. Метод аналогий (эмпатии).
5. Метод инверсии.
6. И др.

Следует отметить, что при применении перечисленных методов важное место занимают задачи формализации данных и разработка информационных моделей.

АЛГОРИТМИЧЕСКИЕ СТРАТЕГИИ

1. **Метод «Мозгового штурма»** реализуется в двух стратегиях:

- Прямая "мозговая атака" - является методом коллективного генерирования идей решения творческой задачи. Цель этого метода заключается в сборе как можно большего количества идей, освобождении от инерции мышления, преодолении привычного хода мысли в решении творческой задачи.
- Обратная "мозговая атака" - предполагает не генерацию новых идей, а критику уже имеющихся.

Обратная "мозговая атака" может быть проведена сразу после прямой, когда после коллективного генерирования идей формируются контр идеи.

2. **Метод ключевых вопросов** целесообразно применять для сбора дополнительной информации в условиях проблемной ситуации или упорядочения уже имеющейся при решении проблемы. Известно, что еще в Древнем Риме политикам рекомендовалось для сбора более полной информации о событии ставить перед собой ряд вопросов и отвечать на них: кто? что? зачем? где? чем? как? когда?

Принципиальные требования к использованию метода:

- Проблемность и оптимальность. Искусно поставленными вопросами необходимо снижать «Проблемность» задачи до оптимального уровня или снижать неопределенность проблемы.
- Дробление информации. С помощью вопросов постараться разделить проблему на подпроблемы.
- Целеполагание. Каждый новый вопрос должен формировать стратегию, модель решения проблемы.

3. **Метод ассоциаций**

Основан на использовании в творческом процессе ассоциаций, метафор и случайно выбранных понятий. Между любыми двумя разными понятиями можно осуществить логическую связь, т.е. переход в 4-5 этапов:

- «Древесина» – «лес»,
- «лес» - «поле»,
- «поле» – «футбольное»,
- «футбольный» – «мяч».

В результате процесса зарождения новых ассоциативных связей и возникают творческие идеи решения проблемы.

Институт кибернетики

АЛГОРИТМИЧЕСКИЕ СТРАТЕГИИ

4. Метод аналогий (эмпатии)

Из Википедии:

Анало́гия (др.греч. ἀναλογία — пропорция, соответствие, соразмерность) — подобие, равенство отношений; сходство предметов, явлений, процессов, величин и т. п. в каких-либо свойствах, а также познание путём сравнения.

В решении творческих задач используют различные аналогии: конкретные и абстрактные; ведутся поиски аналогии живой природы с неживой, например в области техники. Виды аналогий:

- Прямая – находится в биологических системах (Георг де Местраль – изобрел застежку липучку – аналог репейника, вертолет – аналог стрекозы).
- Символическая (абстрактная) аналогия – это поэтические метафоры и сравнения, в которых характеристики одного предмета отождествляются с характеристиками другого. Необходимо отрешиться от словесного описания задачи и создать ее зрительный образ.
- Фантастическая аналогия – позволяет представить вещи в новой ипостаси, которую они не имеют или не получают (дельфины для охраны ВМФ баз, голуби для точного наведения ракет).
- Личная аналогия – отождествляем себя с чем-либо или кем-либо. ЛПР должен поставить себя на место изучаемой проблемы.

5. Метод инверсии

Инверсия - изменение процедур деятельности на противоположные, обращение функций, взгляд на систему с противоположной точки зрения, нежели общепринятая, замена динамики статикой и наоборот. Классический пример инверсии – изобретение ракеты. К. Циолковский думал, что придумал пушку, но летающую.

Институт кибернетики

Поиск с возвратом

Из Википедии:

Поиск с возвратом (*backtracking*) — общий метод нахождения решений задачи, в которой требуется полный перебор всех возможных вариантов в некотором множестве М. Как правило, позволяет решать задачи, в которых ставятся вопросы типа: «Перечислите все возможные варианты …», «Сколько существует способов …», «Есть ли способ …», «Существует ли объект…» и т. п.

Термин *backtracking* был введен в 1950 году американским математиком Дерриком Генри Лемером .

Незначительные модификации метода поиска с возвратом, связанные с представлением данных или особенностями реализации, имеют и иные названия: метод проб и ошибок, метод ветвей и границ, поиск в глубину, и т. д.

Решение задачи методом поиска с возвратом сводится к последовательному расширению частичного решения. Если на очередном шаге такое расширение провести не удается, то возвращаются к более короткому частичному решению и продолжают поиск дальше. Данный алгоритм позволяет найти все решения поставленной задачи, если они существуют. Для ускорения метода стараются вычисления организовать таким образом, чтобы как можно раньше выявлять заведомо неподходящие варианты. Зачастую это позволяет значительно уменьшить время нахождения решения.

Можно сказать, что метод поиска с возвратом является универсальным.

Метод поиска с возвратом позволяет решать множество переборных задач. Например, с помощью него можно получить все подмножества, размещения, перестановки, сочетания данного множества M.

Недостатки

Достаточно легко проектировать и программировать алгоритмы решения задач с использованием этого метода. Однако время нахождения решения может быть очень велико даже при небольших размерностях задачи (количестве исходных данных), причём настолько велико (может составлять годы или даже века), что о практическом применении не может быть и речи. Поэтому при проектировании таких алгоритмов, обязательно нужно теоретически оценивать время их работы на конкретных данных. Существуют также задачи выбора, для решения которых можно построить уникальные, «быстрые» алгоритмы, позволяющие быстро получить решение даже при больших размерностях задачи. Метод поиска с возвратом в таких задачах применять неэффективно.

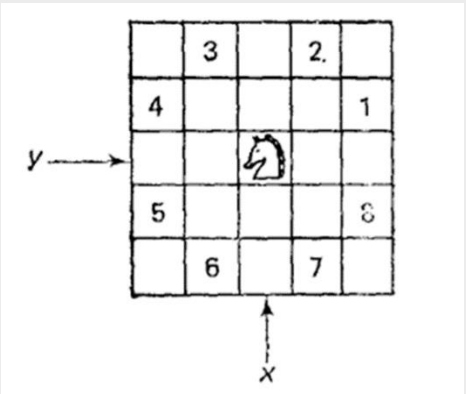
АЛГОРИТМИЧЕСКИЕ СТРАТЕГИИ

Продемонстрируем основные принципы алгоритма с возвратом на хорошо известном примере – задаче о ходе коня[4]:

Пример. Дана доска $n \times n$, содержащая n^2 полей. Конь, который ходит согласно шахматным правилам, помещается на поле с начальными координатами x0,y0. Нужно покрыть всю доску ходами коня, т.е. вычислить обход доски, если он существует, из $n^2 - 1$ ходов, такой, что каждое поле посещается ровно один раз.

Решение.

Очевидно, что задачу покрытия n^2 полей можно свести к более простой: или вычислить очередной ход, или установить, что он невозможен. Алгоритм на каждом шаге предполагает проверку возможности сделать ход в каждом из 8 допустимых направлений, Как это показано на рисунке.



Институт кибернетики

АЛГОРИТМИЧЕСКИЕ СТРАТЕГИИ

Схематически процедура выполнения очередного средствами языка Pascal хода может быть представлена следующим образом [4]:

```
procedure попытка следующего хода;
begin инициация выборки ходов;
   repeat выбор следующего возможного хода из списка очередных ходов;
        if он приемлем then
           begin запись хода;
             if доска не заполнена then
               begin попытка следующего хода
                  if неудача then стирание предыдущего хода – возврат назад
               end
           end
     until (ход был удачным)∨(нет других возможных ходов)
end
```

Институт кибернетики

АЛГОРИТМИЧЕСКИЕ СТРАТЕГИИ

Если мы хотим более точно описать алгоритм, то должны выбрать некоторое представления для данных. Очевидно, что доску можно представить в виде целочисленной матрицы, например: h(8×8).

Целочисленной потому, что каждую клетку доски (элемент матрицы h) мы будем представлять целым числом, отражающим номер хода, например:

Очевидно, можно остановиться на таких соглашениях: $h_{x,y} = 0$, если поле x,y не посещалось. И $h_{x,y} = i$, если поле x,y посещалось на i-ом ходу ($1 \leqslant i \leqslant n^2$).

Получить следующий ход u,v из x,y можно просто добавлением разности координат. Эти разности можно поместить, например в два массива:

a[8] = (2, 1, -1, -2, -2, -1, 1, 2); b[8] = (1, 2, 2, 1, -1, -2, -2, -1);

Подробно с описанием алгоритма можно познакомиться в работе Н. Вирта[4]. Пример обхода доски:

1	6	15	10	21
14	9	20	5	16
19	2	7	22	11
8	13	24	17	4
25	18	3	12	23

Институт кибернетики

АЛГОРИТМИЧЕСКИЕ СТРАТЕГИИ

Алгоритм обхода доски реализуется выполнением последовательности шагов. Каждый шаг характеризуется состоянием - текущим положением коня. Алгоритм на каждом шаге предполагает проверку возможности сделать следующий ход в одном из 8 допустимых направлений.

Описание функции, реализующей покрытие доски на языке С++ может иметь следующий вид:

```
// R – размер матрицы, представлящей шахматную доску глобальная переменная
void Step(int i, int x, int y, Boolean &q)// попытка следующего хода
{
// i- номер хода – входной параметр
// q – признак того, удачен ли ход – выходной параметр
// x,y – координаты текущего положения коня – входные параметры
  Boolean q1 = false; // вспомогательная переменная
  for (int k=0; k<8 && !q1; k++)//пока есть возможность выбора хода в одном из 8
направлений
    { // k – номер шага из 8 возможных;   u,v – координаты следующего хода
      int u = x+a[k]; //вычислить новую координату x
      int v = y+b[k];  //вычислить новую координату y
      if ((u>=0&&u< R) && (v>=0&&v< R))// если шаг возможен
         if(h[u][v] == 0) // если клетка не занята ходом
           {
              h[u][v] = i;//выставить ход
              if (i<R*R)// если шаги остались
                {
                   Step(i+1,u,v,q1);//сделать следующий ход
                   if (!q1) h[u][v] = 0;//отменить ход
                }
              else q1=true;// если шагов не осталось

           }

    }
  q=q1;// для возврата результата
}
```

Институт кибернетики

АЛГОРИТМИЧЕСКИЕ СТРАТЕГИИ

Метод проб и ошибок

Метод проб и ошибок является модификацией метода поиска с возвратом, отличие от которого в том, что он основан на несистематическом, ненаправленном поиске и переборе вариантов решения. Поэтому в «просторечии» его иногда называют методом «тыка» (научного).

В 1898 г. метод проб и ошибок был описан Э. Торндайком как «форма научения, основанная на закреплении случайно совершённых двигательных и мыслительных актов, за счёт которых была решена значимая для животного задача». В следующих пробах время, которое затрачивается животным на решение аналогичных задач в аналогичных условиях, постепенно, хотя и не линейно, уменьшается, до тех пор, пока не приобретает форму мгновенного решения.

Последующий анализ метода проб и ошибок показал, что он не является полностью хаотическим и нецелесообразным, а интегрирует в себе прошлый опыт и новые условия для решения задачи.

Если рассматривать абсолютно случайный перебор вариантов, то можно сделать следующие выводы:

Достоинства метода:

- Этому методу не надо учиться.
- Методическая простота решения.
- Удовлетворительно решаются простые задачи (не более 10 проб и ошибок).

Недостатки метода:

- Плохо решаются задачи средней сложности (более 20—30 проб и ошибок) и практически не решаются сложные задачи (более 1000 проб и ошибок).
- Нет приёмов решения.
- Нет алгоритма мышления, мы не управляем процессом думанья. Идет почти хаотичный перебор вариантов.
- Неизвестно, когда будет решение и будет ли вообще.
- Отсутствуют критерии оценки силы решения, поэтому неясно, когда прекращать думать. А вдруг в следующее мгновение придет гениальное решение?
- Требуются большие затраты времени и волевых усилий при решении трудных задач.
- Иногда ошибаться нельзя ИЛИ этот метод не подходит (не будет человек резать на бомбе провода наугад).

Институт кибернетики

АЛГОРИТМИЧЕСКИЕ СТРАТЕГИИ

Считается, что для метода проб и ошибок выполняется правило — «первое пришедшее в голову решение — слабое». Объясняют этот феномен тем, что человек старается поскорее освободиться от неприятной неопределённости и делает то, что пришло в голову первым.

Метод проб и ошибок лежит в основе технологий творчества и изобретательства.

Неизменными атрибутами творчества привыкли до сих пор считать озарение, интуицию, прирожденные способности, счастливый случай.

Вообще, решение изобретательских задач – один из древнейших видов человеческой деятельности. И поразительно консервативный. И в наши дни в направлении решения задачи последовательно выдвигаются пришедшие в голову идеи. При этом всякий раз неудачная идея отбрасывается, а вместо нее выдвигается новая. Правил поиска нет: ключом к решению может оказаться любая идея, даже самая «дикая».

Нет и определенных правил первоначальной оценки идей: пригодна или непригодна идея, заслуживает она проверки или нет – об этом приходится судить субъективно.

Когда-то варианты решения задачи перебирали наугад – в случайном порядке. Современный изобретатель фильтрует их, отбрасывая неудачные. Увеличение степени фильтрации – главная тенденция исторического развития метода «проб и ошибок».

Наиболее широко применяемыми подходами к построению алгоритмов оптимизации (поиска наилучшего решения), в основе которых лежит метод «проб и ошибок» являются следующие:

- Алгоритмы случайного поиска.
- Муравьиные алгоритмы.
- Генетические и эволюционные алгоритмы.

Генетические и муравьиные алгоритмы в данной работе выделены в отдельные алгоритмические стратегии и будут рассмотрены нами отдельно. Эти алгоритмы разрабатываются в рамках научного направления, которое можно назвать «природные вычисления».

Институт кибернетики

АЛГОРИТМИЧЕСКИЕ СТРАТЕГИИ

Муравьиные алгоритмы

Практическая необходимость решения ряда задач в оптимизационной постановке при проектировании и исследовании сложных систем привела разработчиков алгоритмического обеспечения к использованию биологических механизмов поиска наилучших решений.

Из Википедии.

Муравьи – социальные насекомые, которые живут в коллективе – колонии. Число муравьёв в одной колонии может достигать нескольких миллионов, а сама колония может быть распределена по территории в десятки и сотни километров. В то же время, колония не имеет централизованного управления, а обмен информацией между особями осуществляется при помощи непрямого обмена, благодаря феромону, который муравьи оставляют в процессе своего передвижения в пространстве.

Феромо́ны (др.-греч. φέρω — несу + ὁρμάω — возбуждаю, побуждаю) — собирательное название веществ — продуктов внешней секреции, выделяемых некоторыми видами животных и обеспечивающих химическую коммуникацию между особями одного вида.

Муравьиный алгоритм (алгоритм оптимизации подражанием муравьиной колонии, ant colony optimization, ACO) — один из эффективных полиномиальных алгоритмов для нахождения приближённых решений задачи коммивояжера, а также решения аналогичных задач поиска маршрутов на графах.

Коммивояжёр – это муравей, которому необходимо по кратчайшему маршруту посетить все пункты, ни разу не вернувшись в тот, где он уже был.

Институт кибернетики

АЛГОРИТМИЧЕСКИЕ СТРАТЕГИИ

Оригинальная идея исходит от наблюдения за муравьями в процессе поиска кратчайшего пути от колонии до источника питания.

Среди экспериментов по выбору между двумя путями неравной длины, ведущих от колонии к источнику питания, биологи заметили, что, как правило, муравьи используют кратчайший маршрут. Модель такого поведения заключается в следующем:

- Муравей проходит от колонии по пути, выбранному случайным образом.
- Если он находит источник пищи, то возвращается в гнездо, оставляя за собой след из феромона.
- Эти феромоны привлекают других муравьёв, находящихся вблизи, которые вероятнее всего пойдут по этому маршруту.
- Вернувшись в гнездо, они «укрепят» феромонную тропу.
- Если существует 2 маршрута, то по более короткому за то же время успеют пройти больше муравьёв, чем по длинному. Короткий маршрут станет более привлекательным.
- Длинные пути, в конечном итоге, исчезнут из-за испарения феромонов.

Описанная система перемещения муравьев базируется на положительной (другие муравьи укрепляют феромонную тропу) и отрицательной (испарение феромонной тропы) обратной связей.

Первая версия «муравьиного» алгоритма, предложенная доктором наук Марко Дориго в 1992 году, была направлена на поиск оптимального пути в графе.

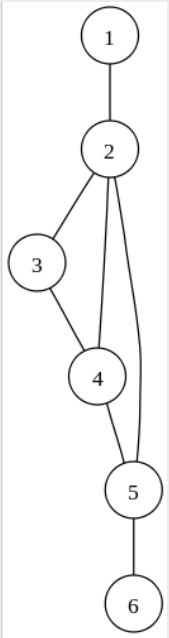
Институт кибернетики

АЛГОРИТМИЧЕСКИЕ СТРАТЕГИИ

Граф (англ. *graph*) — основной объект изучения математической теории графов, представляющий собой совокупность непустого множества вершин и наборов пар вершин (связей между вершинами). Граф в абстрактной и наглядной форме дает представление о связях между множеством объектов, что облегчает понимание задачи. Объекты в графах представляются как вершины, или узлы графа, а связи — как дуги, или рёбра. Для разных областей применения виды графов могут различаться направленностью, ограничениями на количество связей и дополнительными данными о вершинах или рёбрах. Многие структуры, представляющие практический интерес в математике и информатике, могут быть представлены графами.

Самый первый пример, который приходит в голову - это социальная сеть.

Вершинами графа являются люди, а ребрами отношения (дружба). Граф может быть неориентированным, то есть я могу дружить только с теми, кто дружит со мной.
Либо ориентированным, где можно добавить человека в друзья, без того чтобы он добавлял вас.
Особенно широкое применение графы нашли, в том числе, при решении задач на «системы дорог».



Неориентированный граф с шестью вершинами и семью рёбрами

Институт кибернетики

АЛГОРИТМИЧЕСКИЕ СТРАТЕГИИ

Алгоритм, предложенный Марко Дориго, основывается на обработке графа, в узлы которого помещаются источники питания для муравьев (своего рода муравейники), а ребра определяют маршруты, ведущие к источникам питания[3]. Разработаны различные способы задания или описания графов. Например, с помощью матрицы инцидентности.

Сам метод и реализация алгоритма используется автором для решения задачи коммивояжёра.

С учётом особенностей задачи коммивояжёра, мы можем описать локальные правила поведения муравьёв при выборе пути:

1. Муравьи имеют собственную «память». Поскольку каждый город может быть посещён только один раз, то у каждого муравья есть список уже посещённых пунктов – список запретов. Обозначим через $J_{i,k}$ - список пунктов, которые необходимо посетить муравью k, находящемуся в пункте i.

2. Муравьи обладают «зрением». Видимость есть эвристическое желание посетить пункт j, если муравей находится в пункте i. Будем считать, что видимость обратно пропорциональна расстоянию между пунктами $\eta_{i,j} = 1/ D_{i,j}$.

3. Муравьи обладают «обонянием» – они могут улавливать след феромона, подтверждающий желание посетить пункт j из пункта i на основании опыта других муравьёв. Количество феромона на ребре (i,j) в момент времени t обозначим через $\tau_{i,j}(t)$.

4. Пройдя ребро (i,j), муравей откладывает на нём некоторое количество феромона, которое должно быть связано с оптимальностью сделанного выбора.

Институт кибернетики

АЛГОРИТМИЧЕСКИЕ СТРАТЕГИИ

Вычислительные формулы:

$$P_{i,j} = 100 \cdot \frac{\eta_{i,j}^{\beta} \cdot \tau_{i,j}^{\alpha}}{\sum_{k=0}^{N} \eta_{i,k}^{\beta} \cdot \tau_{i,k}^{\alpha}}$$

N – число пунктов;

$P_{i,j}$ – вероятность перехода из i в j;

$\eta_{i,j}$ – величина, обратная длине перехода из i в j;

$\tau_{i,j}$ – количество феромона при переход из i в j;

β – величина, определяющая «жадность» алгоритма;

α - величина, определяющая «стадность» алгоритма.

Здесь константа 100 используется для преобразования вероятностной оценки к процентному виду. α– определяет степень влияния феромона. β– определяет степень влияния расстояния. Например, при α = 0 алгоритм вырождается до жадного алгоритма (будет выбран ближайший город).

$$\Delta\tau_{(i,j),k} = \begin{cases} \dfrac{Q}{L_k(t)}, & (i,j) \in T_k(t) \\ 0, & (i,j) \notin T_k(t) \end{cases}$$

$\Delta\tau_{(i,j),k}$ – откладываемое количество феромона k-ым муравьем на пути от i до j ;
$T_k(t)$ – маршрут, пройденный муравьём k к моменту времени t; $L_k(t)$ – длина этого маршрута; Q- параметр, имеющий значение порядка длины оптимального пути.

Правило испарения феромонов, где $p \in [0,1]$ – коэффициент испарения, m – количество муравьёв в колонии можно записать в виде:

$$\tau_{i,j}(t+1) = (1-p) \cdot \tau_{i,j}(t) + \Delta\tau_{i,j}(t);$$

$$\Delta\tau_{i,j}(t) = \sum_{k=1}^{m} \Delta\tau_{(i,j),k}(t)$$

Институт кибернетики

АЛГОРИТМИЧЕСКИЕ СТРАТЕГИИ

Муравьиный алгоритм для задачи коммивояжера может быть представлен в виде:

1. Ввод исходных данных и инициализация параметров настройки и переменных алгоритма. Исходными данными являются: количество муравейников-пунктов; количество муравьёв, приходящихся на один муравейник ; матрица с расстояниями между пунктами.
2. Определение вероятностей перехода из одного пункта в другой
3. Генерация случайного маршрута движения каждого муравья из последнего местонахождения с учётом рассчитанных вероятностей перехода; определение результирующей длины каждого маршрута (критерий оптимизации); сохранение в памяти вариантов маршрутов с наилучшим значением критерия.
4. Расчёт изменения концентрации феромона между двумя муравейниками и нового значения концентрации феромона.
5. Повторение процедуры с п. 2 до выполнения одного или нескольких выбранных условий окончания.
6. Вывод вариантов маршрутов, обеспечивающих наилучшее значение критерия оптимальности.

Условием окончания может быть определенное время жизни колонии или отсутствие улучшения целевой функции на заданном промежутке времени.

Сложность данного алгоритма зависит от времени жизни колонии (t_{max}), количества городов (n) и количества муравьёв в колонии (m).

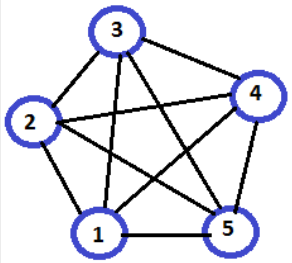
Институт кибернетики

АЛГОРИТМИЧЕСКИЕ СТРАТЕГИИ

Продемонстрируем вычисление маршрута отдельного муравья при условии, что он начинает свое движение из пункта с номером 1, при значении параметров:

$\alpha = 1$, $\beta = 1$;

Для примера [из 5], предположим, что имеется 5 мест, хранящих источники питания, назовем их пунктами. Пункты пронумерованы от 1,…,5. Известны расстояния между пунктами, а также условная доза феромонов, оставленная муравьями в процессе движения по соответствующему маршруту, как это показано на рис.1.



Связи (i-j)	Расстояние между i и j ($L_{i,j}$)	Оценка феромонов ($\tau_{i,j}$)
1-2	38	3
1-3	74	2
1-4	59	2
1-5	45	2
2-3	46	1
2-4	61	1
2-5	72	1
3-4	49	2
3-5	85	2
4-5	42	1

Рис.1. Граф, содержащий муравьиные пункты и связи между ними

Передвижение отдельного муравья начинается с размещения его случайным образом в вершину графа, затем начинается движение муравья в направлении, определенному вероятностным методом. Предположим, что это вершина с номером 1.

Институт кибернетики

АЛГОРИТМИЧЕСКИЕ СТРАТЕГИИ

Рассчитаем вероятности перехода из вершины 1 в остальные вершины:

$$P_{1,2} = 100\frac{\frac{1}{38}\cdot 3}{\frac{1}{38}\cdot 3+\frac{1}{74}\cdot 2+\frac{1}{59}\cdot 2+\frac{1}{45}\cdot 2} = \frac{7.9}{0.18} = 42.83$$

$$P_{1,3} = 100\frac{\frac{1}{74}\cdot 2}{\frac{1}{38}\cdot 3+\frac{1}{74}\cdot 2+\frac{1}{59}\cdot 2+\frac{1}{45}\cdot 2} = 14.66$$

$$P_{1,4} = 100\frac{\frac{1}{59}\cdot 2}{\frac{1}{38}\cdot 3+\frac{1}{74}\cdot 2+\frac{1}{59}\cdot 2+\frac{1}{45}\cdot 2} = 18.4$$

$$P_{1,5} = 100\frac{\frac{1}{45}\cdot 2}{\frac{1}{38}\cdot 3+\frac{1}{74}\cdot 2+\frac{1}{59}\cdot 2+\frac{1}{45}\cdot 2} = 24.11$$

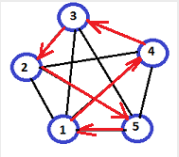
Разыгрывается ДСЧ – куда пойдем? Предположим – это пункт 4. Вычеркнем 1 пункт – по условию задачи коммивояжёра, пункт можно посетить только 1 раз. Вычисляем $P_{4,2}$, $P_{4,3}$,$P_{4,5}$ по тем же правилам, например:

$$P_{4,2} = 100\frac{\frac{1}{61}\cdot 1}{\frac{1}{61}\cdot 1+\frac{1}{49}\cdot 2+\frac{1}{42}\cdot 1} = 20.23$$

Снова разыгрывается ДСЧ – куда пойдем? Предположим – это пункт 3. Вычеркиваем город 4 и т.д.

Институт кибернетики

Пусть обход вершин графа будет таким, как это показано на следующем рисунке:



Расстояние, пройденное муравьем, получается простым суммированием

(59+49+46+72+45).

В общем случае, стартовая точка, куда помещается муравей, зависит от ограничений, накладываемых условиями задачи. Для каждой задачи способ размещения муравьёв является определяющим: либо все они помещаются в одну точку, либо в разные с повторениями, либо без повторений.

Заметим, что вычисленные вероятности определяют только ширину зоны пункта *j* в общей зоне всех пунктов. А выбор самого пункта муравьем определяется случайным числом. Правило для определения вероятностей не изменяется в ходе алгоритма, но у двух разных муравьев значение вероятности перехода будут отличаться, т.к. они имеют разный список разрешенных городов.

К недостаткам метода можно отнести:

1. Теоретический анализ затруднён.
2. Сходимость гарантируется, но время сходимости не определено.
3. Сильно зависят от настроечных параметров, которые подбираются только исходя из экспериментов.

Генетические алгоритмы

Генетические алгоритмы изначально разрабатывались для решения задач оптимизации. В основе генетического алгоритма лежит метод случайного поиска. Из теории оптимизации известно, что основным недостатком случайного поиска является то, что нам неизвестно, сколько времени понадобится для решения задачи.

Для того, чтобы избежать таких расходов времени при решении задачи, применяются методы, проявившиеся в биологии. Эти методы открыты при изучении эволюции и происхождения видов.

Исследователи обращаются к природным механизмам, которые миллионы лет обеспечивают адаптацию биоценозов к окружающей среде. В живой природе особи конкурируют друг с другом за различные ресурсы. Более приспособленные будут иметь больше шансов выжить. Одним из таких механизмов, имеющих фундаментальный характер, является механизм наследственности. Его использование для решения задач оптимизации привело к появлению генетических алгоритмов.

Впервые подобный алгоритм был предложен в 1975 году Джоном Холландом (John Holland) в Мичиганском университете. Он получил название "репродуктивный план Холланда" и лег в основу практически всех вариантов генетических алгоритмов.

АЛГОРИТМИЧЕСКИЕ СТРАТЕГИИ

Генетический алгоритм является самым известным на данный момент представителем эволюционных алгоритмов и по сути является алгоритмом для нахождения глобального экстремума многоэкстремальной функции многих переменных, такой как $f(x_1,x_2,...,x_n)$. Ее называют целевой функцией.

Например, целевая функция может иметь вид: $f(x,y)=\frac{x}{x^2+2y^2+1}$

Функция от двух переменных - двух генов.

Для того, чтобы заработал генетический алгоритм, необходимо независимые переменные (гены) $x_1,x_2,...,x_n$ представить в виде хромосом – цепочек символов, с которыми и работает алгоритм.

Как создать хромосомы?

Первый шаг – выполнить преобразование независимых переменных в цепочки бит, которые будут содержать всю необходимую информацию о каждой особи.

Имеется два варианта кодирования параметров: в двоичном формате; в формате с плавающей запятой. В случае, если мы используем двоичное кодирование, мы используем N бит для каждого параметра, причем, N может быть различным для каждого параметра. Если параметр может изменять свои значения от MAX до MIN, можно использовать следующие формулы преобразования:

$$r = q \cdot (MAX-MIN)/(2^N-1)+MIN,$$

$$q = (r-MIN)/(MAX-MIN) \cdot (2^N-1),$$

где r – целочисленные двоичные гены, а q– эквивалент в форме с плавающей точкой. Хромосомы с плавающей точкой создаются при помощи размещения закодированных параметров один за другим.

Институт кибернетики

АЛГОРИТМИЧЕСКИЕ СТРАТЕГИИ

Как известно в теории эволюции важную роль играет то, каким образом признаки родителей передаются потомкам. В генетических алгоритмах [7] за передачу признаков родителей потомкам отвечает оператор, который называется *скрещивание* (его также называют кроссовер или кроссинговер). Этот оператор определяет передачу признаков родителей потомкам. Действует он следующим образом:

- из популяции выбираются две особи, которые будут родителями;
- определяется (обычно случайным образом) точка разрыва;
- потомок определяется как конкатенация части первого и второго родителя.

Рассмотрим функционирование этого оператора. Пусть две особи имеют следующий хромосомный набор:

Хромосома_1: 0000000000
Хромосома_2: 1111111111

Допустим, что разрыв происходит после 3-го бита хромосомы. Тогда:

Хромосома_1: 0000000000 >> 000 1111111 Результирующая_хромосома_1
Хромосома_2: 1111111111 >> 111 0000000 Результирующая_хромосома_2

Результирующая хромосома в качестве потомка определяется случайным выбором из этих двух.

Следующий генетический оператор предназначен для того, чтобы поддерживать разнообразие особей с популяции. Он называется оператором *мутации*. При использовании данного оператора некоторые биты в хромосоме с определенной вероятностью инвертируется.

Институт кибернетики

АЛГОРИТМИЧЕСКИЕ СТРАТЕГИИ

Кроме того, используется еще и так называемый оператор *инверсии*, который заключается в том, что хромосома делится на две части, и затем они меняются местами. Схематически это можно представить следующим образом:

000 1111111 >> 1111111 000

В принципе для функционирования генетического алгоритма достаточно этих двух генетических операторов, но на практике применяют еще и некоторые дополнительные операторы или модификации этих двух операторов. Например, кроссовер может быть не одноточечный (как было описано выше), а многоточечный, когда формируется несколько точек разрыва (чаще всего две).

Кроме того, в некоторых реализациях алгоритма оператор мутации представляет собой инверсию только одного случайно выбранного бита хромосомы.

На начальном этапе создания алгоритма нужно случайным образом создать начальную популяцию; даже если она окажется совершенно неконкурентоспособной, вероятно, что генетический алгоритм всё равно достаточно быстро переведёт её в жизнеспособную популяцию. Таким образом, на первом шаге можно особенно не стараться сделать слишком уж приспособленных особей, достаточно, чтобы они соответствовали формату особей популяции, и на них можно было подсчитать функцию приспособленности (Fitness). Итогом первого шага является популяция H, состоящая из N особей.

АЛГОРИТМИЧЕСКИЕ СТРАТЕГИИ

Генетический алгоритм можно описать следующими шагами:

1. Задать целевую функцию (приспособленности) для особей популяции.
2. Создать начальную популяцию.
3. Цикл по поколениям, пока не выполнено условие останова. В цикле:
   //для поколения
   1) Вычислить значение целевой функции для особей.
   2) Оценить приспособленность каждой особи.
   3) Выполнить селекцию по приспособленности (не все выживают).
   4) Случайным образом разбить популяцию на две группы пар.
   5) Выполнить скрещивание - кроссовер для пар популяции и заменить родителей на потомков.
   6) Произвести вероятностную мутацию.
   7) Объявить потомков новым поколением
4. Конец цикла по поколениям

На этапе селекции нужно из всей популяции выбрать определённую её долю, которая останется «в живых» на этом этапе эволюции. Есть разные способы проводить отбор. Вероятность выживания особи h должна зависеть от значения целевой функции. Сама доля выживших s обычно является параметром генетического алгоритма, и её просто задают заранее. По итогам отбора из N особей популяции H должны остаться sN особей, которые войдут в итоговую популяцию H'. Остальные особи погибают.

Институт кибернетики

АЛГОРИТМИЧЕСКИЕ СТРАТЕГИИ

Отметим, что для размножения обычно выбираются особи из всей популяции H (известны случаи, когда гениальные дети рождаются у «весьма неодаренных» родителей), а не из выживших на первом шаге элементов H_0 (хотя последний вариант тоже имеет право на существование)? Дело в том, что главный бич многих генетических алгоритмов — недостаток разнообразия (diversity) в особях. Есть разные способы борьбы с таким нежелательным эффектом: помимо мутации один из них — выбор для размножения не самых приспособленных, но вообще всех особей.

Другой важный момент – определение критериев останова. Обычно в качестве них применяются :

- Достижение определенного числа поколений.
- Истечение времени, отпущенного на эволюцию.
- Схождение популяции (путем сравнивания приспособленности популяции на нескольких поколениях и остановки при стабилизации этого параметра).

Под схождением понимается такое состояние популяции, когда ни операция кроссовера, ни операция мутации не вносят изменения в генетическое разнообразие популяции в течение нескольких поколений.

Генетический алгоритм не является волшебной палочкой при решении всех проблем минимизации/максимизации. Во множестве случаев другие алгоритмы гораздо быстрее и намного практичнее. Однако для задач с огромным количеством параметров и там где проблема сама по себе может быть легко определена, генетический алгоритм может быть подходящим выбором.

Институт кибернетики

Численные алгоритмы

Вычислительные (численные) методы — это методы решения математических задач, в которых предполагается как представление исходных данных, так и результатов решения в виде чисел.

Примерами применения численных методов являются:

1. Решение систем линейных и нелинейных уравнений.
2. Решение задач интерполирования.
3. Приближенное вычисление функций.
4. Численное решение дифференциальных уравнений.
5. Решение задач оптимизации.
6. И др.

Численные методы являются одним из мощных математических средств решения задач, и занимают особое место в курсе «Математика».

По большому счету численные вычисления лежат в основе большого числа разнообразных программ. Так большого объема вычислений с обработкой многочленов и матриц требует компьютерная графика. Эти вычисления обычно делаются для каждой точки экрана, поэтому даже незначительные улучшения алгоритмов могут привести к заметному ускорению.

Помимо приведенных примеров применения численных методов следует отметить еще одно - вычисление тригонометрических функций. Чего не осознают многие программисты — это то, что стандартные процедуры вычисления значений тригонометрических функций типа синуса или косинуса используют их представление степенными рядами, а начальные отрезки таких рядов представляют собой многочлены и вычисление значений тригонометрических функций сводится к вычислению значений многочленов.

Приведем примеры численных вычислений:

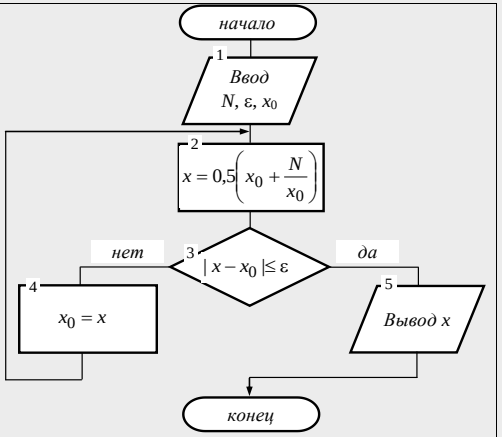
АЛГОРИТМИЧЕСКИЕ СТРАТЕГИИ

Пример 1. Разработать алгоритм для вычисления приближенного значения квадратного корня из числа *N* по формуле Ньютона: $x_{i+1} = 0{,}5\left(x_i + \frac{N}{x_i}\right)$, где $(i = 0, 1, 2, \ldots)$

Начальное приближение корня задано и равно x_0.

Так как речь идет о приближенном значении величины $\sqrt{N}$, то необходимо указать величину ε, характеризующую требуемую точность вычислений. Определение корня состоит в многократном вычислении (пересчете) по формуле, в правой части которой используется предыдущее значение переменной *x*. Вычисления следует прекратить при достижении требуемой точности, т.е. при выполнении условия $|x_{i+1} - x_i| \le \varepsilon$. Блок-схема вычисления представлена на рисунке справа.

Здесь вместо элементов массива используется переменная *x*, которая последовательно принимает значения x_1, x_2, … За результат счета будет взято то значение *x*, при котором выполняется $|x - x_0| \le \varepsilon$.



Институт кибернетики

Пример 2. Составить алгоритм вычисления суммы членов сходящегося ряда:

$$S = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots + \frac{(-1)^n x^{2n}}{(2n)!} = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!}$$

с точностью ε.

Ряд сходится при любых значениях x. Достаточным условием обеспечения заданной точности является достижение очередным членом ряда $a_n = \frac{(-1)^n x^{2n}}{(2n)!}$ величины $|a_n| \le \varepsilon$.

Общий алгоритм прост: задать начальное значение суммы ряда, а затем многократно вычислять очередной член ряда и добавлять его к ранее найденной сумме, пока абсолютная величина очередного члена ряда не станет меньше заданной точности. Предсказать заранее, сколько членов ряда потребуется просуммировать, невозможно. Прямое вычисление члена ряда по приведенной формуле, когда x возводится в степень и вычисляется факториал, трудоемко, и может произойти потеря точности. Поэтому следует воспользоваться рекуррентной формулой получения последующего члена ряда через предыдущий. Запишем выражения для n-го и $(n+1)$-го членов ряда:

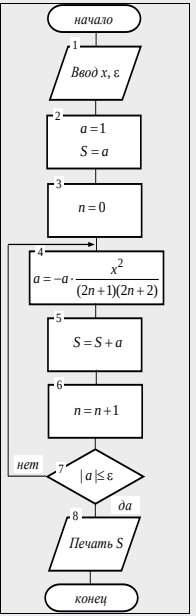
$$a_n = \frac{(-1)^n x^{2n}}{(2n)!}; \qquad a_{n+1} = \frac{(-1)^{n+1} x^{2(n+1)}}{(2(n+1))!}.$$

Найдем их отношение: $\frac{a_{n+1}}{a_n} = -\frac{x^2}{(2n+1)(2n+2)}$,

отсюда получим формулу:

$a_{n+1} = -a_n \cdot \frac{x^2}{(2n+1)(2n+2)}$, где $n = 0, 1, 2, 3, \ldots$.

Значение $a_0 = 1$.



Пример 3. Составить блок-схему вычисления значения многочлена степени n.

Для вычисления значения многочлена степени n вида:

$$y = a_1 x^n + a_2 x^{n-1} + \ldots + a_n x + a_{n+1}$$

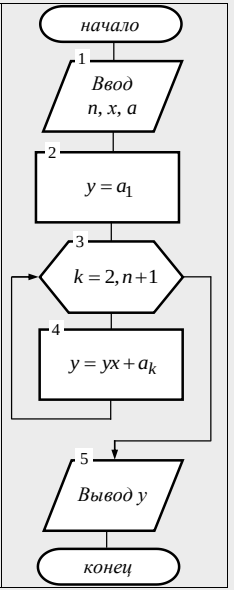
удобна формула Горнера:

$$y = (\ldots((a_1 x + a_2)x + a_3)x + \ldots + a_n)x + a_{n+1},$$

обеспечивающая минимальное количество операций, так как для возведения переменной x в степень n используют рекуррентную формулу:

$$y = yx + a_k,$$

где $k = 2, 3, \ldots, n+1$. Перед циклом необходимо задать начальное значение полинома, равное коэффициенту в наибольшей степени x, а внутри цикла значение полинома вычислять по вышеприведенной формуле.



Сравнение с образцом

Постановка задачи «Сравнение с образцом» может быть сформулирована на примере обработки строк следующим образом:

Даны образец (строка) и строка . Требуется определить индекс, начиная с которого образец содержится в строке . Если образец не содержится — вернуть индекс, который не может быть интерпретирован как позиция в строке (например, отрицательное число).

Поиск подстроки [2] в длинном куске текста — важный элемент текстовых редакторов. Однако ту же самую технику можно использовать для поиска битовых или байтовых строк в двоичном файле. Так, например, осуществляется поиск вирусов в памяти компьютера. В программах обработки текстов обычно имеется функция проверки синтаксиса, которая не только обнаруживает неправильно написанные слова, но и предлагает варианты их правильного написания. Один из подходов к проверке состоит в составлении отсортированного списка слов документа. Затем этот список сравнивается со словами, записанными в системном словаре и словаре пользователя; слова, отсутствующие в словарях, помечаются как возможно неверно написанные.

По условию задачи требуется найти только первое вхождение некоторой подстроки в длинном тексте. Поиск последующих вхождений основан на том же подходе (имеет смысл завести дополнительную функцию, вызываемую при каждом обнаружении образца).

Стандартный алгоритм начинает со сравнения первого символа текста с первым символом подстроки. Если они совпадают, то происходит переход ко второму символу текста и подстроки. При совпадении сравниваются следующие символы. Так продолжается до тех пор, пока не окажется, что подстрока целиком совпала с отрезком текста, или пока не встретятся несовпадающие символы. В первом случае задача решена, во втором мы сдвигаем указатель текущего положения в тексте на один символ и заново начинаем сравнение с подстрокой.

АЛГОРИТМИЧЕСКИЕ СТРАТЕГИИ

Процесс использования стандартного алгоритма на базе поиска образца “they” в тексте “there they are” приведен в следующей таблице:

1 проход	THERE THEY ARE THEY	Три первых символа совпадают, а четвертый - нет
2 проход	THERE THEY ARE THEY	Сдвиг указателя в тексте Указатель подстроки – в начале
3 проход	THERE THEY ARE THEY	Сдвиг указателя в тексте Указатель подстроки – в начале
4 проход	THERE THEY ARE THEY	Сдвиг указателя в тексте Указатель подстроки – в начале
5 проход	THERE THEY ARE THEY	Сдвиг указателя в тексте Указатель подстроки – в начале
6 проход	THERE THEY ARE THEY	Сдвиг указателя в тексте Указатель подстроки – в начале
7 проход	THERE THEY ARE THEY	Сдвиг указателя в тексте и сдвиг в тексте до полного совпадения

При первом проходе три первых символа подстроки совпадают с символами текста. Однако только седьмой проход дает полное совпадение. Из алгоритма ясно, что основная операция — сравнение символов, и именно число сравнений и следует подсчитывать. Если подсчитать количество сравнений, то их будет 13. В наихудшем случае при каждом проходе совпадают все символы за исключением последнего.

Институт кибернетики

АЛГОРИТМИЧЕСКИЕ СТРАТЕГИИ

Описание алгоритма можно представить в виде:

Обозначение данных:

```
text  - исходная строка
substring – заданная подстрока
textLoc - указатель текущего сравниваемого символа в тексте
subLoc указатель текущ. сравниваемого символа в подстроке
textStart указатель на начало сравнения в тексте
```

Следующий алгоритм осуществляет стандартное сравнение строк:

```
subLoc=1 //указатель текущ. сравниваемого символа в подстроке
textLoc=1 //указатель текущего сравниваемого символа в тексте
textStart=1 //указатель на начало сравнения в тексте
        //пока текст не закончен и не закончены символы в подстроке
while TextLoc<=length(text) and subLoc<=length(substring) do
   if text[textLoc]=substring[subLoc] then //если совпадение символов
        begin
          textLoc=textLoc+1 //указатель в тексте увеличивают
          subLoc=subLoc+1 //указатель в подстроке увеличивают
        end
   else // начать сравнение заново со следующего символа
      begin
        textStart=textStart+1 // указатель на начало сравнения в
                              //тексте  увеличивают
        textLoc= textLoc+1//указатель в тексте увеличивают
        subLoc= 1// указатель в подстроке устанавливают в начало
      end
   end if
end while
if (subLoc > length(substring)) then
   return textStart // совпадение найдено
else
   return 0 // совпадение не найдено
end if
```

Институт кибернетики

Идея алгоритма Бойера-Мура

Проблема стандартного алгоритма заключается в том, что он затрачивает много усилий (много сравнений) впустую. Уменьшить этот недостаток помогает алгоритм Бойера-Мура. Он осуществляет сравнение с образцом справа налево, а не слева направо. Исследуя искомый образец, можно осуществлять более эффективные прыжки в тексте при обнаружении несовпадения.

1 проход	THERE THEY ARE THEY
2 проход	THERE THEY ARE THEY
3 проход	THERE THEY ARE THEY

В примере мы сначала сравниваем у с г и обнаруживаем несовпадение. Поскольку мы знаем, что буква г вообще не входит в образец, мы можем сдвинуться в тексте на целых четыре буквы (т. е. на длину образца) вправо. Затем мы сравниваем букву у с h и вновь обнаруживаем несовпадение. Однако поскольку на этот раз h входит в образец, мы можем сдвинуться вправо только на две буквы так, чтобы буквы h совпали. Затем мы начинаем сравнение справа и обнаруживаем полное совпадение кусочка текста с образцом. В алгоритме Бойера—Мура делается 6 сравнений вместо 13 сравнений в исходном простом алгоритме.

Помимо описанных алгоритмов распространение получили такие, как: алгоритм Кнута-Морриса-Пратта; статистические методы. Подробно с алгоритмами сравнения с образцом можно познакомиться в работе [2].

АЛГОРИТМИЧЕСКИЕ СТРАТЕГИИ

Список литературы:

1. Рыбалка С.А., Шкатова Г.И. C++ Builder. Задачи и решения. Учебное пособие. – Томск: Изд-во ТПУ, 2010. – 486 с.
2. Дж. Макконнелл Основы современных алгоритмов, Москва: Техносфера, 2004. - 368с.
3. Ананий В.Левитин Алгоритмы: введение в разработку и анализ Москва, изд.дом «Вильямс», 2009г.
4. Вирт Н. Алгоритмы + структуры данных = программы: пер. с англ / Н. Вирт. – М.: Мир, 1985. – 406 с.
5. www.youtube.com/watch?v=EwDP_bAb-OI
6. http://rain.ifmo.ru/cat/data/theory/unsorted/ant-algo-2006/article.pdf
7. Гладков Л.А. Генетические алгоритмы: Учебное пособие / Л.А. Гладков, В.В. Курейчик,– Изд. 2-е. – М.: Физматлит, 2006. – 320 с.

Институт кибернетики

Спасибо за внимание!