Тестирование программ

Спецкурс

Преподаватели: Андрей Сергеевич Лопатин, Сергей Владимирович Копелиович, Юрий Сергеевич Петров

Этап 1: Придумывание идеи

Тестирование идей

- Идеи должны быть проверены – как минимум, на тесте из условия
- Методы тестирования:
  - Обсудить с товарищем по команде
  - Проверить на крайних, простых тестах
  - Если идея реализуется быстро – можно написать программу и проверить её

Этап 2: Проектирование кода

Проектирование кода

- Когда проектируете код – имейте в виду, что его наверняка придётся отлаживать
- На этапе проектирования необходимо аккуратно и правильно разбить код на процедуры – это облегчает последующее тестирование
- Полезно выписать ключевые места решения и инварианты

Этап 3: подготовка к написанию

Подготовка к написанию

- Важна правильная организация работы
- Если это олимпиада, то каждую задачу полезно класть в свою папку
- Стоит заранее предусмотреть тестовые фичи, которые вы будете использовать (например, мультитест, стресс-тест)

Мультитест

- Никогда не стоит стирать тесты (информация не должна теряться) – от лишних тестов хуже не будет
- Для того, чтобы было удобнее тестировать на большом количестве тестов, в своей программе важно делать поддержку мультитеста – потом можно будет легко сравнить ответы сразу на всех тестах

Стресс-тест

- Очень мощная технология, позволяющая за короткое время протестировать свою программу на огромном числе тестов (миллионы)
- Тесты – либо случайные, либо упорядоченная генерация маленьких тестов
- Хорошо скрещивается с мультитестом

Реализация стресс-тестирования

- Стресс можно пускать как из самой программы, так и из операционной системы
- Из операционной системы используются .bat (.cmd) и .sh

Пример .bat-файла

```
@echo off
:loop
gen >input.txt
if errorlevel 1 goto exit
del output.txt
sol1
if errorlevel 1 goto exit
move output.txt output.ans
sol2
if errorlevel 1 goto exit
fc output.txt output.ans
if errorlevel 1 goto exit
goto loop
:exit
```

Пример .sh-файла

```
#!/bin/sh
while true; do
    ./gen >input.txt || break
    rm output.txt
    ./sol1 || break
    mv output.{txt,ans}
    ./sol2 || break
    diff output.{txt,ans} || break
done
```

С чем стресс-тестировать?

- Медленное решение – дословный перевод условия на язык программирования
- Медленное решение не должно быть слишком умным
- Если в решении просят найти оптимальный объект из какого-то множества, поможет абсолютно полный перебор всего
- Можно стресс-тестировать решение и с самим собой (на runtime error или time limit)

Этап 4. Написание кода

Assert

- Самый важный оператор в программе на любом языке – assert
- С помощью него проверяются наиболее сложные и нетривиальные моменты программы, например, связность получившегося графа, корректность получившегося результата, etc.

Тестирование по частям

- Восходящее и нисходящее кодирование
- Можно тестировать какую-то процедуру по отдельности или всю программу без какой-то процедуры
- Можно даже искать ошибки “двоичным поиском”

Комментарии с инвариантом

```
{ a[l] <= k < a[r] }
l := 0; r := n + 1;
while r – l > 1 do begin
    m := (l + r) div 2;
    if k < a[m] then r := m else l := m;
end;
```

- Кстати, о двоичном поиске… аккуратней с вещественными числами!
- Кроме того, на инвариант можно ставить assert

Этап 5: отладка

Чтение кода

- При чтении кода очень полезно иметь представление о том, какие обычно бывают баги и в каких местах
- Например, перепутанные x и y, забытое запоминание в dfs, …

Отладочный вывод

- Полезно отслеживать вход/выход из процедуры и её параметры
- При отладке рекурсии можно варьировать размер отступа в зависимости от глубины рекурсии
- Иногда важные массивы и/или объекты следует выводить (и проверять на корректность заодно)
- Полезно разбить программу на несколько существенных частей и выводить признак завершения какой-то из частей.

Этап 6: Генерация тестов

Случайная перестановка

```
for i := 1 to n do begin
    t := random (i) + 1;
    a[i] := a[t];
    a[t] := i;
end;
```

- shuffle

Произвольный комбинаторный объект

- Пусть $S(n) = \{a_1 S(n_1), a_2 S(n_2), \ldots, a_k S(n_k)\}$. Тогда, зная количества $|S(n_1)| = c_1, \ldots, |S(n_k)| = c_k$, обозначим $C = c_1 + \ldots + ck$, можно пойти по i-й ветке с вероятностью c_i/C:

```
Procedure rec (n : tstate);
var i, c  : integer;
Begin
    C := 0;
     for i := 1 to k do inc (C, c[i]);
     X := random (C);
     for i := 1 to k do if x < c[i] then begin
              write (a[n, i]);
              rec (n_i);
     end else dec (x, c[i]);
end;
```

Деревья

- Скобочная последовательность
- Код Прюфера
- Генерация предка:
  - P[i] = random (random (random (i + 1))) + 1;

Графы

- Маленькие случайные (см. перебор объектов)
- Заданное разбиение на компоненты

Невыпуклый многоугольник

- Задаётся случайный набор точек и соединяется в случайном порядке
- Потом последовательно уничтожаются пересечения

Стандартные красивые тесты

- Строки: abacabadabacaba…
- Графы: цепочка, кольцо, четыре треугольника, дерево, кактус
- Многоугольники: труба, спираль, звезда

Этап 7: анализ времени выполнения

Методы анализа времени выполнения

- Счётчик числа итераций
- Счётчик тактов (rdtsc)
- Встроенный таймер (GetTickCount (), clock (), gettimeofday)
- Профайлер

Профайлер

- Во многих средах и системах разработки бывает полезная утилита – профайлер, которая собирает статистику по времени работы и числу итераций
- Например, в gcc - gprof

Использование gprof

- Сначала собрать программу с ключом –pg (желательно – без оптимизаций)
- Запустить программу, дождаться её завершения
- Запустить gprof с параметрами “-a имя программы”

Пример вывода Gprof

- index % time self children called name
- 0.03 0.13 1/1 main [2]
- [1] 100.0 0.03 0.13 1 IsolatedPrimes::findPrime(int, int, int) [1]
- 0.07 0.03 110/110 NextWindow(int) [3]
- 0.03 0.00 10813177/10813177 check(char*, int) [4]
- 0.00 0.00 45605/7858681 set(char*, int) [5]
- 0.00 0.00 10762428/10762428 advance(int&, int&) [9]
- 0.00 0.00 1/111 FillWindow(int) [10]
- 0.00 0.00 1/1 answer(unsigned long long) [11]
- ---
- <spontaneous>
- [2] 100.0 0.00 0.16 main [2]
- 0.03 0.13 1/1 IsolatedPrimes::findPrime(int, int, int) [1]
- 0.00 0.00 1/1 logopen() [12]

Этап 8: неожиданные баги

На ноль делить нельзя

```
var a, b : integer;
begin
  read (a, b);
  writeln (a div b);
end.
```

При каких значениях a и b будет ошибка “деление на ноль”?

На ноль делить нельзя

1. Только b = 0
2. b = 0 и ещё одно значение b
3. b = 0 и ещё одна пара (a, b)
4. Ни одно из вышеперечисленных

На ноль делить нельзя

- Правильный ответ: b = 0 и (a = -2147483648, b = -1)

Степени двойки

```
#include <cstdio>
#include <cassert>

int main () {
  int s = 0, t;

  scanf ("%d", &t);
  assert (! (t & (t - 1)));
  while (t) {
    t >>= 1;
    s += t;
  }
  printf ("%d\n", s);
  return 0;
}
```

Степени двойки

Зависнет ли программа?

1. Всегда виснет
2. Виснет при одном значении t
3. Всегда завершается
4. Ни одно из вышеперечисленных

Степени двойки

- Правильный ответ: виснет при t = -2^31 (-2147483648)
- В java для борьбы с этим есть логический сдвиг (>>>), в C – беззнаковые типы.

Приведение типов в языке java

Программа 1:

```
import java.util.*;

public class t1 {
  public static void main (String args [])
  throws Exception {
    Scanner in = new Scanner (System.in);
    int a = in.nextInt ();
    long b = in.nextLong ();
    a = a + b;
    System.out.println (a);
  }
}
```

Приведение типов в языке java

Программа 2:

```
import java.util.*;

public class t2 {
  public static void main (String args [])
  throws Exception {
    Scanner in = new Scanner (System.in);
    int a = in.nextInt ();
    long b = in.nextLong ();
    a += b;
    System.out.println (a);
  }
}
```

Приведение типов в языке java

Компилируются ли эти две программы?

1. Обе компилируются
2. Обе не компилируются
3. Первая компилируется, вторая – нет
4. Вторая компилируется, первая – нет

Приведение типов в языке java

Правильный ответ:

В java строгое приведение типов. Поэтому первая программа не компилируется (нельзя в int записать long). Но вторая – компилируется!

Сумма двух чисел

```
#include <cstdio>

int main () {
  printf ("%d\n", 12345 + 54321);
  return 0;
}
```

Сумма двух чисел

Что выведет программа?

1. 66666
2. 1130
3. Завершится с ошибкой
4. Ни одно из вышеперечисленного

Сумма двух чисел

Правильный ответ – 17777

Использование функций

```
#include <cstdio>
#include <cassert>

int nextInt () {
  int tmp;
  assert (scanf ("%d", &tmp) == 1);
  return tmp;
}

void display (int a, int b) {
  printf ("%d %d\n", a, b);
}

int main () {
  display (nextInt (), nextInt ());
  return 0;
}
```

Использование функций

Что выведет программа на вводе “2 3”

1. 2 2
2. 3 2
3. 2 3
4. Как повезёт…

Использование функций

Правильный ответ – как повезёт!

Порядок вычисления параметров в C/C++ строго не регламентирован…

Спасибо за внимание!

Контакты:

http://vkontakte.ru/kotehok