Глава 7. Дихотомия

Дихотомия, *дихотомический поиск*, *двоичный поиск*, *бинарный поиск* — это разные названия для одного и того же — способа нахождения аргумента, при котором данная величина принимает «нужное» значение. При этом сам поиск ведется в некотором смысле «подбором». Вначале из условий задачи мы определяем диапазон, внутри которого заведомо содержится искомое значение [*Left, Right*]. То есть мы должны точно знать, что искомое значение больше либо равно *Left* и меньше либо равно *Right*. На текущем шаге мы в качестве «проверяемого» значения выбираем *Middle*, равное среднему арифметическому *Left* и *Right*:

```
Middle := (Left + Right)/2
```

Далее условиями задачи определяется целевая функция *Target*. Мы вычисляем ее значение на аргументе *Middle* и сравниваем его со специальным значением *Required* («должно быть»). В зависимости от результатов сравнения мы принимаем решение, какую из границ (*Left* или *Right*) заменить значением *Middle*, и продолжаем процесс, пока не вычислим значение в *Middle*, совпадающее с *Required* с нужной точностью *Diff*, например так:

```
procedure Dichotomy;
begin
  while (abs(Right-Left)>Diff) do
    begin
      Middle:=(Left+Right)/2;
      if Target(Middle)<Required
        then Left  := Middle
        else Right := Middle;
    end;
end;
```

Далее приводятся решения нескольких олимпиадных задач методом дихотомии.

7.1. Задача «Арбузы»

Белорусская республиканская олимпиада по информатике, 1994

Арбузы

В ряд лежат N арбузов, пронумерованных от 1 до N. Нам известно, что:

— массы первого и N-го арбузов $m(1)$ и $m(N)$ соответственно;

— масса i-го арбуза $m(i)$ есть среднее арифметическое масс двух соседних арбузов, увеличенное на d $m(i) = d + (m(i - 1)+m(i + 1)) / 2$

По введенным $m(1)$, $m(N)$, d и j найти $m(j)$. Ограничение $N < 200$. Если найти $m(j)$ по введенным данным невозможно, вывести фразу «данные некорректны».

Ввод:

```
M(1)
M(N)
N
D
J
```

Вывод:

```
M(J)
```

Пример ввода	*Пример ввода*
1.0	30.0
1.0	210.0
21	10
1.0	-5.0
15	8
85.0	данные некорректны

Эта задача является характерным примером применения бинарного поиска: мы будем подбирать вес второго арбуза так, чтобы расчеты веса N-го арбуза по заданной в условиях формуле давали значение веса N-го арбуза, заданного в исходных данных.

По ходу расчетов мы последовательно будем вычислять веса всех арбузов от 3-го до N-го, и поэтому после завершения подбора методом дихотомии мы можем вывести вес любого из этих арбузов.

Прежде всего рассмотрим тело программы:

```
begin
  InputData;
  Dichotomy;
  OutputData;
end.
```

Оно выглядит предельно лаконично, в нем вызываются три процедуры:

- *InputData* — для ввода и инициализации исходных данных;
- *Dichotomy* — для нахождения решения методом дихотомии;
- *OutputData* — для вывода результатов.

Начнем с классической процедуры *Dichotomy*, осуществляющей дихотомию на множестве вещественных чисел:

```
procedure Dichotomy;
  begin
    while (abs(Right-Left)>Diff) do
      begin
        Middle:=(Left+Right)/2;
        if Target(Middle)<Required
          then Left  := Middle
          else Right := Middle;
      end;
  end;
```

Если левая и правая границы поиска отличаются более чем на *Diff* (где *Diff* — задаваемая нами точность вычислений, например 0,000001), то вычисляется *Middle* — середина отрезка [*Left*, *Right*].

Далее переопределяется значение левой или правой границы поиска в зависимости от результата сравнения значений целевой функции *Target* (*Middle*) и *Required*.

Сущность задачи отражает именно алгоритм вычисления функции *Target*:

```
  function Target(Middle:double): double;
    begin
      m[2]:=Middle;
      for i:=3 to N do
        m[i]:=2*m[i-1]-m[i-2]-2*D;
      Target:=M[N];
    end;
```

По условию задачи сказано, что веса арбузов связаны соотношением

```
m(i)=d+(m(i-1)+m(i+1))/2
```

которое описывает как вычислять вес арбуза, если известны веса его левого и правого соседей. Для вычисления веса арбуза по двум заданным весам лежащих рядом арбузов гораздо удобнее формула, легко получающаяся из предыдущей:

```
m(i+1):=2*m(i)-m(i-1)-2*D;
```

или, если вычислять вес *i*-го арбуза, а не $(i + 1)$-го, то

```
m(i):=2*m(i-1)-m(i-2)-2*D;
```

Вес первого арбуза нам известен, а вес второго арбуза мы подбираем так, чтобы получить правильное значение веса *N*-го арбуза.

Теперь перейдем к рассмотрению процедуры ввода

```
procedure InputData;
  begin
    assign(input,'input.txt'); reset(input);
    readln(m[1]);
    readln(mn);
    readln(N);
    readln(D);
    Readln(j);
    close(input);
    Left  := 0;
    Right := 1000;
    Required := mn;
  end;
```

Она вводит исходные данные, а также устанавливает начальные значения левой и правой границ дихотомического поиска (*Left* и *Right*) и величину, которую мы будем использовать как эталонную для целевой функции (*Required*):

И, наконец, процедура вывода:

```
procedure OutputData;
  begin
    assign(output,'output.txt'); rewrite(output);
    i:=1;
    while (i<=N) and (M[i]>0.001) do inc(i);
    if i>N
      then writeln(M[j]:0:3)
      else writeln('данные некорректны');
    close(output);
  end;
```

Вообще от нас требуется вывод с точностью до 0,001 веса арбуза с номером *j*, заданным в исходных данных. Однако в результате расчетов вес арбуза может оказаться нереальным (то есть меньше нуля или пренебрежимо малым — почти равным нулю). В этом случае, по задумке авторов задачи, требуется выводить ответ «данные некорректны».

Полный текст решения приводится в конце главы.

7.2. Задача «Голодание»

Гомельская областная олимпиада по информатике, 1997

Голодание

> Курс лечебного голодания длился 21 день. В результате ежедневного взвешивания пациента были получены результаты $m(1)$, ..., $m(21)$. Оказалось,

что изменение веса между i-м и $(i + 1)$-м взвешиваниями ($i = 1, ..., 20$) прямо пропорционально весу в i-й день $m(i)$.

Вводятся веса $m(1)$ и $m(21)$. Необходимо найти $m(2)$ — вес человека на второй день. Ответ вывести с 4 цифрами после запятой.

Пример ввода	*Пример вывода*
70 72	70.0987

Обозначим K коэффициент увеличения веса в день, тогда получаем:

$$m(2) = K \cdot (1)$$
$$m(3) = K \cdot (2) = K^2 \cdot m(1)$$
$$m(3) = K \cdot (2) = K^3 \cdot m(1)$$
$$...$$
$$m(21) = K \cdot (20) = K^{20} \cdot m(1)$$

Отсюда $K^{20} = m(21)/m(1)$ и $K = \sqrt[20]{m(21)/m(1)}$.

Таким образом, чтобы ответить на вопрос задачи, нам нужно вычислить корень двадцатой степени из числа. Поскольку в Паскале нет такой функции (в отличие, например, от корня второй степени — *SQRT*), мы должны реализовать соответствующие вычисления самостоятельно. Один из способов сделать это — применить дихотомию для подбора K. Рассмотрим подробнее реализацию, начиная с главной программы:

```
begin
  InputData;
  Dichotomy;
  OutputData;
end.
```

Это стандартный образец. Такой же стандартной является и процедура *Dichotomy*:

```
procedure Dichotomy;
  begin
    while (abs(Right-Left)>Diff) do
      begin
        Middle:=(Left+Right)/2;
        if Target(Middle)<Required
          then Left  := Middle
          else Right := Middle;
      end;
  end;
```

Как уже говорилось, специфика задачи заключается в функции *Target*, вызываемой из процедуры *Dichotomy*:

```
function Target(Middle:double): double;
  var
    Res : Double;
  begin
    Res:=m1;
    for i:=1 to 20 do Res:=Res*Middle;
    Target:=Res;
  end;
```

В данном случае функция *Target* в соответствии с условиями задачи вычисляет 20-ю степень числа *Middle* — в результате вычисления *Target* = *Middle*^20 с помощью цикла.

Рассмотрим теперь процедуру *InputData*:

```
procedure InputData;
  begin
    assign(input,'input.txt'); reset(input);
    readln(m1);
    readln(m21);
    close(input);
    if m21/m1 < 1
      then
        begin
          Left  := 0.0;
          Right := 1.0;
        end
      else
        begin
          Left := 1.0; Right := m21/m1;
          Middle := 1.0;
        end;
    Required := m21;
  end;
```

В ней, кроме ввода исходных данных $m1$ и $m21$, необходимо присвоить начальные значения величин *Left*, *Right* и *Required*, используемых в процедуре *Dichotomy*.

Известно, что корень любой степени из числа, меньшего 1, также меньше 1, и поэтому для случая $(m21 / m1) < 1$ устанавливаются границы *Left* и *Right*, равные 0 и 1 соответственно. Аналогично в случае $(m21 / m1) > 1$ устанавливаем *Left* = 1 и *Right* = $m21 / m1$. Кроме того, для случая $(m21/m1)=1$ устанавливаем заведомо известный ответ *Middle* = 1.0, поскольку в процедуре *Dichotomy Middle* не будет вычисляться вследствие равенства *Left* и *Right*.

И, наконец, вывод результата:

```
procedure OutputData;
  begin
    assign(output,'output.txt'); rewrite(output);
```

```
    writeln(m1*Middle:0:4);
    close(output);
  end;
```

По условию задачи требуется вывести $m2$ с точностью до четырех знаков после запятой. Понятно, что в результате дихотомии мы нашли $K = Middle$, поэтому и выводится $m2 = m1 * Middle$.

Полный текст решения приводится в конце главы.

7.3. Задача «День рождения»

Гомельская областная олимпиада по информатике, 2000.

День рождения (birthday.in / birthday.out / 5 с)

На Новый год Малыш познакомил Карлсона со своими друзьями. Оказалось, что каждый малыш один раз в год празднует день рождения, и все малыши приглашают друг друга и Карлсона на свой праздник кушать торт с вареньем. Первым угощают Карлсона. Каждый день Карлсон может съесть M килограммов варенья (если оно есть на дне рождения или в запасе у Карлсона), и еще несколько килограммов взять с собой в банку емкостью N килограммов (если банка еще не полная). На Новый год Карлсон съедает все имеющееся варенье. Карлсон пронумеровал все дни в году и составил список дней рождения всех малышей и количества варенья в каждый из этих дней.

Требуется выяснить, сколько Карлсонов потребуется, чтобы ни одному малышу не досталось ни одного килограмма варенья. На каждый день рождения может придти несколько Карлсонов, у каждого Карлсона имеется только одна банка, за один день Карлсон может побывать на нескольких днях рождения.

Ввод:

```
M N
K
A1 B1
A2 B2
...
Ak Bk
```

Здесь M — количество килограммов варенья, которое Карлсон может съесть за один день, N — емкость банки Карлсона, K — количество дней рождения, Ai — порядковый номер в году i-го дня рождения, Bi — количество килограммов варенья на i-м дне рождения.

Ограничения:

$0 < M, N, Bi < 100$, целое число;

$0 < K < 32000$,

$0 < Ai < 365$, целое число.

Вывод:

L — минимальное количество Карлсонов, которое необходимо, чтобы ни одному малышу не досталось ни одного килограмма варенья.

Пример ввода:

```
5 10
3
102 12
103 9
57 3
```

Пример вывода:

```
2
```

Применим дихотомию для решения данной задачи — будем искать количество Карлсонов дихотомическим поиском.

Главная программа выглядит традиционно:

```
begin
  InputData;
  Dichotomy;
  OutputData;
end.
```

А вот в процедуре *Dichotomy* есть некоторые изменения:

```
procedure Dichotomy;
  begin
    while (abs(Right-Left)>Diff) do
      begin
        if Right=Left+1
          then
            begin
              Middle:=Right;
              exit;
            end
          else Middle:=(Left+Right) div 2;
        Required:=Middle * N;
        if Target(Middle)>Required
          then Left  := Middle
          else Right := Middle;
      end;
    Required := Middle * N;
  end;
```

Первое из изменений связано с тем, что мы выполняем сейчас дихотомию на множестве целых чисел:

```
        if Right=Left+1
          then
```

```
        begin
          Middle:=Right;
          exit;
        end
      else Middle:=(Left+Right) div 2;
```

Поскольку мы ищем *Middle* на множестве целых чисел, мы заменили деление на операцию *div*. При использовании *div* возникает проблема «недоступности» некоторых чисел. Пусть в результате дихотомии мы добрались до границ *Left* = 6 и *Right* = 7. Теперь при вычислении *Middle* по формуле *Middle:* = (*Left* + *Right*) *div* 2 мы опять получим для *Middle* значение 6 и *НЕ МОЖЕМ* получить значение 7. А если именно оно и является ответом? Для того чтобы выйти из этой ситуации, введены операторы, дающие возможность присвоить *Middle* значение 7 в вышеописанной ситуации:

```
    if Right=Left+1
    then
      begin
        Middle:=Right;
        exit;
      end
```

Другое изменение процедуры *Dichotomy* связано со спецификой данной задачи:

```
Required:=Middle * N;
```

то есть величина *Required*, которую мы используем как эталон для сравнения, вычисляется в процедуре *Dichotomy* и зависит от значения *Middle*, выбранного на текущем шаге.

И, наконец, сравнение выглядит следующим образом:

```
if Target(Middle)>Required
```

То есть мы действуем следующим образом: для текущего значения количества Карлсонов *Middle* вычисляем величину *Required* = *Middle*N* — сколько варенья *может быть унесено* всеми Карлсонами в банках после текущего дня рождения (у всех Карлсонов все банки полны до краев). В функции *Target*(*Middle*) мы вычисляем, сколько варенья *должны унести* Карлсоны после текущего дня. Каждый из них съел свои *M* килограммов в этот день и съедал ровно по *M* (когда было что есть) во все предыдущие дни.

Функция *Target* выглядит следующим образом:

```
function Target(Middle:longint): longint;
  var
    Karlsons, Banka : longint;
  begin
    Karlsons:=Middle;
    Banka:=0;
    for i:=1 to 364 do
      begin
        Banka:=Banka + Days[i] - Karlsons*M;
        if Banka<0 then Banka:=0;
```

```
        if Banka>Karlsons*N then break;
      end;
    Target:=Banka;
  end;
```

Итак, пусть у нас *Middle* Карлсонов. В цикле по всем дням года мы вычисляем, сколько варенья останется в «обобщенной банке» всех Карлсонов:

```
Banka:=Banka + Days[i] - Karlsons*M;
```

прибавляя к тому, что было, все варенье, которое подавали к столу в этот день на всех днях рождения (*Days*[*i*]), и вычитая все варенье, которое Карлсоны были в состоянии съесть в этот день (*Karlsons*M*). Если оказывается, что подавалось меньше, чем Карлсоны могли съесть (с учетом запасов), и переменная *Banka* получает отрицательное значение, то переменная *Banka* обнуляется.

Если переменная *Banka* получает значение больше того, что по условиям задачи Карлсоны могут хранить (*Karlsons*N*), то надо прекращать цикл и увеличивать количество Карлсонов. Если же в течение всех дней обобщенная банка Карлсонов (*Banka*) так и не переполнилась, то нужно уменьшать количество Карлсонов.

Рассмотрим теперь процедуру ввода исходных данных:

```
procedure InputData;
  var
    a,b,Max : longint;
  begin
    assign(input,'birthday.in'); reset(input);
    readln(M,N);
    readln(K);
    for i:=1 to 365 do days[i]:=0;
    for i:=1 to K do
      begin
        readln(A,B);
        inc(days[A],B);
      end;
    max:=days[1];
    for i:=1 to 364 do
      if days[i]>Max then max:= days[i];
    Left   := (Max div (M+N)) -1;
    Middle := Left;
    Right  := (Max div M)+1;
  end;
```

Исходные данные в формате «номер дня, количество варенья» потенциально предполагают возможность нескольких дней рождения в один день. Мы заводим массив *days*[*i*] и, обнулив его вначале, прибавляем в нужный день *A* нужное количество варенья *B* в соответствующий элемент массива *days*[*A*].

Далее для сокращения времени поиска устанавливаются более точно левая и правая границы (минимальное и максимальное количество Карлсонов соответственно)

на основании значения *Max* — максимального количества варенья, предложенного в один день. Карлсоны в максимальном количестве съедят все варенье, «не забирая ничего с собой» даже в «самый варенный день», не говоря уже об остальных. Карлсоны в минимальном количестве должны суметь в «самый варенный день» съесть все, что помещается в них, и не переполнить остатками свои банки.

Процедура *OutputData* обеспечивает вывод переменной *Middle*, которая и содержит искомое количество Карлсонов.

```
procedure OutputData;
  begin
    assign(output,'birthday.out'); rewrite(output);
    writeln(Middle);
    close(output);
  end;
```

Полный текст решения приводится в конце главы.

7.4. Задача «Water Glass»

NEERC, северный четвертьфинал, 2000

Water Glass (input.txt / output.txt)

Стив — стеклодув. Ему дали заказ изготовить водяные часы необычной формы для одной эксцентричной леди. По причине своей эксцентричности она хочет, чтобы ее водяные часы имели форму, составленную из двух симметричных призм, имеющих в сечении вид некоторого многоугольника (рис. 7.1). Конечно, одна из сторон его лежит на поверхности стола, а через одну из вершин вода перетекает из одной призмы в другую. Как профессионал, Стив изготовил заказ. Однако он испытывает затруднения с расстановкой меток времени.

Рис. 7.1. Пример призмы для задачи «Water Glass»

Предположим, что вода течет равномерно, что в начальный момент времени верхняя призма заполнена и что вся вода перетечет из верхней призмы в нижнюю ровно за 24 часа.

Вы должны написать программу, которая по заданной форме многоугольника вычисляет уровни воды в нижней призме каждые полчаса начиная с начального момента времени.

Ввод:

Первая строка содержит N ($3 \le N \le 50$) — количество вершин в многоугольнике. Каждая из следующих N строк содержит 2 целых числа Xi и Yi ($-50 \le Xi \le 50$ и $0 \le Yi \le 50$) — координаты вершины с номером i. Обязательно выполняются следующие условия: если (Xk, Yk), ($X(k + 1)$,$Y(k + 1)$) — координаты вершин, которые формируют основание водяных часов, то $Yk = Y(k + 1)=0$, для всех $1 < i < k$ выполнено неравенство $Yi > Y(i + 1)$, а для всех $k + 1 < i < N$ — противоположное неравенство $Y(i + 1) > Yi$.

Вывод:

Вывод должен содержать 48 строк в формате, представленном в примере ниже (выводить нужно ровно три знака после десятичной точки).

Пример ввода:

```
3
10 20
0 0
20 0
```

Пример вывода

```
00:00 0.000
00:30 0.209
01:00 0.421
01:30 0.635
02:00 0.851
02:30 1.070
03:00 1.292
03:30 1.516
04:00 1.743
04:30 1.972
05:00 2.205
05:30 2.441
06:00 2.679
06:30 2.922
07:00 3.167
07:30 3.417
08:00 3.670
08:30 3.927
09:00 4.189
09:30 4.454
10:00 4.725
10:30 5.000
11:00 5.280
11:30 5.566
12:00 5.858
12:30 6.156
13:00 6.460
13:30 6.771
```

Пример вывода (*продолжение*):

```
14:00 7.090
14:30 7.417
15:00 7.753
15:30 8.098
16:00 8.453
16:30 8.820
17:00 9.199
17:30 9.592
18:00 10.000
18:30 10.426
19:00 10.871
19:30 11.340
20:00 11.835
20:30 12.362
21:00 12.929
21:30 13.545
22:00 14.226
22:30 15.000
23:00 15.918
23:30 17.113
```

Эта задача взята с четвертьфинала студенческого чемпионата мира по программированию 2000 года, который проходил в Санкт-Петербурге. Математическая модель поставленной задачи: дан многоугольник специального вида, одна сторона которого лежит на оси *X*, а все остальные вершины лежат выше оси *X*. Требуется разбить его на 48 равновеликих (равных по площади) частей прямыми, параллельными оси *X*. Нужно вывести значения высот прямых, параллельных оси *X*, в порядке возрастания времени суток от 00:00 до 23:30.

Семантика задачи связана с необходимостью нанести метки времени на часы типа песочных. Для решения этой задачи применим дихотомию 47 раз для нахождения 47 чисел — высот, соответствующих 47 прямым, которые разделят наш многоугольник на 48 равных по площади частей:

```
begin
  InputData;
  H[1]:=0.0;
  for i:=2 to 48 do
    begin
      Left    := H[i-1];
      Right := MaxY;
      Required := S*(i-1)/48;
      Dichotomy;
      H[i]    := Middle;
    end;
  OutputData;
end.
```

В массив *H* мы будем сохранять вычисленные высоты. Левой границей всегда будет найденная на предыдущем этапе, высота $H[i-1]$. Перед поиском $H[2]$ мы устанавливаем $H[1] = 0$. Правая граница устанавливается на максимальное значение *Y* из всех введенных координат вершин.

Required — требуемая часть вычисленной заранее (в процедуре *InputData*) площади заданного многоугольника *S*.

Процедура *Dichotomy* выглядит стандартно:

```
procedure Dichotomy;
  begin
    while (abs(Right-Left)>Diff) do
      begin
        Middle:=(Left+Right)/2;
        if Target(Middle)<Required
          then Left  := Middle
          else Right := Middle;
      end;
  end;
```

Семантику задачи, как обычно, отображает функция *Target*:

```
  function Target(Middle:extended): extended;
    var
      i,j : integer;
    begin
      for i:=B downto 2 do
        if Intersected(i,i-1,Middle,Px1,Py1)
          then begin Bn:=i; break; end;
      for i:=E to N do
        if Intersected(i,i+1,Middle,Px2,Py2)
          then begin En:=i; break; end;
      Target:=Square;
    end;
```

В функции *Target* мы находим точку (*Px*1, *Py*1) пересечения текущей прямой $y = Middle$ с одним из отрезков i, $i-1$ с одной стороны от базы заданного многоугольника (от вершины с номером *B* к вершине с номером *E*, которая лежит на оси *X*) и точку (*Px*2, *Py*2) пересечения прямой $y = Middle$ с одним из отрезков i, $i + 1$ с другой стороны от базы многоугольника.

Для поиска используется функция *Intersected*, принимающая значение *TRUE*, если точка найдена. Затем для вычисления площади фигуры, ограниченной сверху отрезком между найденными точками (*Px*1, *Py*1) и (*Px*2, *Py*2) используется функция *Square*. Рассмотрим их подробнее.

```
function Intersected (i,j:integer; YH : extended;
                      var Px,Py:extended)         : boolean;
```

```
                    {Пересекается ли отрезок от точки i до точки j с прямой
                     Y=YH, если да – Px,Py – координаты точки пересечения.}
  var
    a,b,c : extended;
    Yes   : boolean;
  begin
    Yes := (YH<=max(y[i],y[j])) and (YH>=min(y[i],y[j]));
    Intersected := Yes;
    if not Yes then exit;
    Py:=YH;
    a:=y[j]-y[i];
    b:=x[i]-x[j];
    c:=y[i]*(x[j]-x[i])-x[i]*(y[j]-y[i]);
    if a<>0
      then px:=(-b*YH-c)/a
      else px:=x[i]
  end;
```

Прежде всего устанавливается сам факт — пересекутся или нет текущий отрезок и заданная прямая $y = YH$. Пересечение происходит в том и только в том случае, если *YH* имеет значение между минимальным и максимальным значением координат $Y[i]$ и $Y[j]$. В случае установления факта пересечения используется известная формула построения прямой по координатам двух точек, и затем точки пересечения заданной и построенной прямых (находим *X* по заданному *Y*). Особый случай — когда построенная прямая параллельна оси *X*, в качестве *px* в этом случае можно выбрать любую из точек $x[i]$ или $x[j]$.

Функция *Square* использует стандартную формулу вычисления площади многоугольника, заданного координатами своих вершин. Предварительно формируется специальный массив *CX*, в котором содержатся координаты вершин нижней части многоугольника, отсеченной от исходного прямой $y = Middle$, а также две вновь построенные вершины — точки пересечения.

```
function Square:extended;
  var
    i : integer;
    S : extended;
  begin
    CX:=X; CY:=Y;
    CX[Bn-1]:=Px1; CY[Bn-1]:=Py1;
    CX[En+1]:=Px2; CY[En+1]:=Py2;
    S:=0; CX[En+2]:=Px1; CY[En+2]:=Py1;
    for i:=Bn-1 to En+1 do
      S:=S+(cx[i+1]-cx[i])*(cy[i+1]+cy[i]);
    Square := Abs(S)/2;
  end;
```

Теперь рассмотрим процедуру *Inputdata* для ввода и инициализации исходных данных:

```
procedure InputData;
  var i : integer;
    begin
      assign(input,'input.txt'); reset(input);
      readln(N);
      for i:=1 to N do readln(x[i],y[i]);
      close(input);

      MaxY := y[1];
      for i:=1 to N do
        if y[i]>MaxY then MaxY:=y[i];

      S:=0; x[N+1]:=x[1]; y[N+1]:=y[1];
      for i:=1 to N do
        S:=S+(x[i+1]-x[i])*(y[i+1]+y[i]);
      S:=abs(S)/2;
      for i:=1 to N do
        if (y[i]=0) and (y[i+1]=0) then k:=i;                    {Дно сосуда}
      B:=K;              {Указатель поиска пересекающихся отрезков с начала}
      E:=K+1;            {Указатель поиска пересекающихся  отрезков с конца}
    end;
```

Здесь после ввода координат вершин многоугольника ($x[i]$, $y[i]$) сначала находится максимальная из введенных ордината *MaxY*. Затем вычисляется S — площадь исходного многоугольника. И, наконец, находятся B и E — номера вершин многоугольника, образующих «базу» — сторону многоугольника, которая лежит на оси X.

Процедура *OutputData* выводит найденные значения $H[i]$ в определяемом условиями задачи формате:

```
procedure OutputData;
  var
    c30 : array [0..1] of string[2];
    s2  : string[2];
  begin
    c30[0]:='30'; c30[1]:='00';
    assign(output,'output.txt'); rewrite(output);
    for i:=1 to 48 do
      begin
        Str(((i-1) div 2):2, S2);
        if s2[1]=' ' then s2[1]:='0';
        writeln (s2, ':', c30[i mod 2],' ',H[i]:0:3);
      end;
```

```
    close(output);
  end;
```

Полное решение задачи приводится в конце главы.

7.5. Задача «Последовательность Фибоначчи»

NEERC, центральный четвертьфинал, 2001

Последовательность Фибоначчи (input.txt / output.txt)

Fi — бесконечная последовательность натуральных чисел, удовлетворяющая условию Фибоначчи: $F(i + 2) = F(i + 1) + F(i)$ (для любого целого i).

Напишите программу, которая вычисляет значение *Fn* по данным *i*, *Fi*, *j*, *Fj*, *n*. Здесь *Fi* и *Fj* — два члена последовательности, $i <> j$.

Пример:

```
F(3)=5, F(-1)=4
```

Найти $F(5)$. Ответ: $F(5) = 12$

Ввод и вывод:

Входной файл содержит 5 целых чисел: *i*, *Fi*, *j*, *Fj*, *n*.

Выходной файл должен содержать значение *Fn*.

Ограничения: $-1000 \le i, j, n \le 1000$, $-200\,000\,000 \le Fk \le 2\,000\,000\,000$ ($k = min(i, j, n)$, ..., $max(i, j, n)$)

Пример ввода и вывода

```
3 5 -1 4 5              12
```

Описание решения:

Пусть для определенности $i < j$ (иначе мы просто меняем их местами). Применим дихотомию для поиска $F(i + 1)$. Зная $F(i + 1)$ и применяя соотношение Фибоначчи, можем найти любое число этой последовательности.

Главное тело программы выглядит как обычно:

```
begin
  InputData;
  Dichotomy;
  OutputData;
end.
```

Рассмотрим процедуру *Dichotomy*:

```
procedure Dichotomy;
  begin
    while (abs(Right-Left)>Diff) do
```

```
      begin
        if Right=Left+1
          then
            begin
              Middle:=Right; exit;
            end
          else
            begin
              Middle:= (Left div 2) + (Right div 2);
              if ((Left  mod 2)<>0) and ((Right mod 2)<>0)
                then
                  begin
                    if (Left>0) and (Right>0) then Inc(Middle);
                    if (Left<0) and (Right<0) then Dec(Middle);
                  end;
            end;
        if Target(Middle)<Required
          then Left  := Middle
          else Right := Middle;
      end;
  end;
```

Она выглядит несколько сложнее обычного. Чем это вызвано? Во-первых, в данном случае мы вновь, как и в задаче из пункта 3, имеем дихотомию на множестве целых чисел, и потому вводится текст для достижения всех целых чисел:

```
if Right=Left+1
  then
    begin
      Middle:=Right; exit;
    end
```

Кроме того, данная задача требует нахождения значений близких к границам представления целых чисел типом *Longint* (–2*e*9 ... 2*e*9).

Но вычисления типа *Middle*: = (*Left* + *Rigth*) *div* 2 проводятся некорректно, когда *Left* и *Right* близки к одной из границ диапазона (точнее, когда их сумма выходит за границы). Эту проблему и решает математически эквивалентный, но несколько более громоздкий код, который *СНАЧАЛА* делит нацело на 2 каждое из чисел *Left* и *Right*, а затем складывает полученные частные. Затем итоговый результат модифицируется в том случае, если *ОБА* числа были нечетными и одного знака. При этом если оба числа были положительными, то и результат увеличивается на 1, а если оба числа были отрицательными, то и результат уменьшается на 1.

```
  Middle:= (Left div 2) + (Right div 2);
  if ((Left mod 2)<>0) and ((Right mod 2)<>0)
    then
```

```
      begin
        if (Left>0) and (Right>0) then Inc(Middle);
        if (Left<0) and (Right<0) then Dec(Middle);
      end;
```

Теперь обратимся к функции *Target*:

```
function Target(Middle:longint): extended;
  var
    a,b,c : extended;
    k     : longint;
  begin
    a:=Fi;
    b:=Middle;
    for k:=i+2 to j do
      begin
        c:=a+b;
        a:=b;
        b:=c;
      end;
    Target:=c;
  end;
```

Как уже говорилось ранее, именно эта функция отражает основную специфику задачи. В данном случае по введенному числу *Fi* и подбираемому *Middle* с помощью соотношения Фибоначчи вычисляет *Target = Fj*.

Теперь рассмотрим подробнее процедуру ввода и инициализации исходных данных:

```
procedure InputData;
  var
    ti,tFi : longint;
  begin
    assign(input,'input.txt'); reset(input);
    readln(i,Fi,j,Fj,N);
    close(input);
    if j<i
        then
          begin
            ti:=i;  tFi:=Fi;
            i:=j;   Fi:=Fj;
            j:=ti;  Fj:=tFi;
          end;
       if j=i+1
         then
           begin
             inc(j);
```

```
            Fj:=Fi+Fj;
          end;
    Left := -2000000000; Right := 2000000000; Required := Fj;
  end;
```

Вводятся *i*, *Fi*, *j*, *Fj*, *n*. Если $i > j$, то пары (*i, Fi*) и (*j, Fj*) обмениваются значениями. Устанавливаются границы поиска: $Left = -2e9$, $Right = +2e9$. Устанавливается значение требуемой для поиска величины $Required = Fj$. Теперь рассмотрим процедуру вывода результатов:

```
procedure OutputData;
  var
    a,b,c,Fn : extended;
    k        : longint;
  begin
    assign(output,'output.txt'); rewrite(output);
    a:=Fi;
    b:=Middle;
    if n=i   then c:=Fi;
    if n=i+1 then c:=Middle;
    if n>i+1
      then
        for k:=i+2 to n do
          begin
            c:=a+b;
            a:=b;
            b:=c;
          end;
        if n<i
          then
            for k:=i-1 downto n do
              begin
                a:=c-b;
                c:=b;
                b:=a;
              end;
    Fn:=c; writeln(Fn:0:0); close(output);
  end;
```

При выводе отдельно рассматриваются случаи разных *n*:

- $n = i$, тогда $Fn = Fi$ (из исходных данных);
- $n = i + 1$, тогда $Fn = Middle$ (которое мы вычислили в результате дихотомии);
- $n > i + 1$, тогда ищем *Fn* по прямому соотношению Фибоначчи $F(k) = F(k - 1) + F(k - 2)$;
- $n < i$, тогда ищем *Fn* по обратному соотношению Фибоначчи $F(k - 2) = F(k) - F(k - 1)$.

Полный текст решения приводится в конце главы.

7.6. Задача «Equipment Box»

Central European Programming Contest, 1999

Equipment Box

> В египетской пирамиде есть большая комната, которая называется «Комната, из которой никто не возвращается». Ее пол покрыт прямоугольными плитками одного размера. В ней очень много ловушек и механизмов. Специальная группа, сформированная ACM, потратила несколько лет на изучение секретного плана этой комнаты. Она сформировала умный план, позволяющий избежать все ловушки. Специально обученный инженер-механик был послан обезвредить самую страшную ловушку «Дребезжащие Кости». После обезвреживания инженер должен выйти из комнаты. Очень важно становиться только внутрь каждой прямоугольной плитки, покрывающей пол. Человек не должен наступать на стороны прямоугольника плитки. Однако после операции инженер понял, что план ACM не учел его ящик с инструментами. Ящик нужно размещать на полу, поскольку механику нужно иметь свободными обе руки, чтобы избежать контакта с другими ловушками. Но когда ящик лежит на полу, он может касаться линии — границы прямоугольной плитки. И это и есть задача, которую предстоит решить вам.

Ввод:

Ввод состоит из T тестовых случаев. Их количество (T) задается в первой строке входного файла. Каждый тест состоит из одной строки. Строка содержит ровно четыре целых числа, разделенных одиночными пробелами: A, B, X и Y. A и B задают размеры прямоугольной плитки, X и Y — размеры ящика с инструментами ($1 \le A, B, X, Y \le 50\,000$).

Вывод:

Ваша задача — определить, можно ли положить ящик с инструментами на прямоугольную плитку так, чтобы он целиком лежал внутри плитки и не касался ни одной из границ. Если это возможно — выводите «возможно», иначе выводите «невозможно».

Пример ввода:

```
2
10 10 8 8
8 8 10 10
```

Пример вывода:

```
Escape is possible.
Box cannot be dropped.
```

Идея решения такова: вначале «нормализуем» длины сторон, так что бы первая у обоих прямоугольников была меньше второй, то есть, $A1 < B1$ и $A2 < B2$.

Далее, если каждая из сторон первого прямоугольника строго больше соответствующей стороны второго прямоугольника ($A1 > A2$ и $B1 > B2$), то ответ — «возможно»;

в другом случае дихотомией пытаемся подобрать угол наклона второго прямоугольника, при котором он поместится в первый прямоугольник. Если угол подобрался, то ответ «возможно», если нет — «невозможно».

Рассмотрим подробнее реализацию решения.

Тело главной программы выглядит следующим образом:

```
begin
  assign(output,'output.txt'); rewrite(output);
  assign(input,'input.txt'); reset(input);
  readln(T);
  for i:=1 to T do
    begin
      InputData;
      Exist:= (A1>A2) and (B1>B2);
      if not Exist
        then
          begin
            Required := 1;
            Diff := 1E-3;
            Left := 0;
            Right := Pi/2;
            Dichotomy;
          end;
     if Exist
       then writeln('Escape is possible.')
       else writeln('Box cannot be dropped.');
    end;
  close(input); close(output);
end.
```

Прежде всего вызывается процедура *InputData* для ввода данных очередного теста:

```
procedure InputData;
  begin
    readln(A1,B1,A2,B2);
    if A1>B1 then Swaps(A1,B1);
    if A2>B2 then Swaps(A2,B2);
  end;
```

Эта процедура пользуется процедурой *Swaps*:

```
procedure Swaps(var A,B:longint);
  var
    t : longint;
  begin
    t:=A; A:=B; B:=t;
  end;
```

Далее в случае необходимости вызывается процедура *Dichotomy*. Угол мы подбираем в диапазоне от 0 до *Pi*/2. Экспериментально выяснилось, что для данной задачи достаточно иметь *Diff* = 1*E* – 3.

Процедура *Dichotomy* выглядит как обычно:

```
procedure Dichotomy;
  begin
    while (abs(Right-Left)>Diff) do
      begin
        Middle:=(Left+Right)/2;
        if Target(Middle)<Required
          then Left  := Middle
          else Right := Middle;
      end;
  end;
```

Рассмотрим, наконец, функцию *Target*, в которой собственно и содержится вся специфика данной задачи:

```
function Target(Middle:extended): extended;
  begin
    if ( A1 > (B2*sin(Middle)+A2*cos(Middle) ))  and
       ( B1 > (B2*cos(Middle)+A2*sin(Middle) ))
      then
        begin
          Exist:=true;
          Diff:=1e+10;
          Target:=0;
          exit;
        end;
    if ( A1 <= (B2*sin(Middle)+A2*cos(Middle) )) then Target:=2;
    if ( B1 <= (B2*cos(Middle)+A2*sin(Middle) )) then Target:=0;
  end;
```

Здесь *Middle* — это текущее значение угла поворота в радианах. (*A*1 > (*B*2*sin(*Middle*) + *A*2*cos(*Middle*))) и (*B*1 > (*B*2*cos(*Middle*) + *A*2*sin(*Middle*))) — это условия того, что при заданном угле поворота *Middle* второй прямоугольник помещается в первом.

Если это так, то переменная *Exist* получает значение *TRUE*, а для досрочного выхода из поиска мы меняем величину *Diff*.

Если условия «возможности» не выполнены, то при нарушении одного ограничения мы увеличиваем угол, а при нарушении другого — уменьшаем его. Это делается с помощью переменных: *Required* которой навсегда присвоено значение 1), и *Target*, которой присваивается значение 0 или значение 2, в зависимости от того, как мы хотим изменить угол поворота *Middle*.

Полный текст решения задачи представлен в конце главы.

7.7. Решения задач

Листинг 7.1. Текст программы к задаче «Арбузы»

```
{$N+,E+}
program by94d1t2;
const
  MaxN = 200;
  Diff = 0.000001;
var
  m     : array [1..MaxN] of double;
  N,j,i : integer;
  D,mn  : double;

  Right, Left, Middle, Required  :  double;

  function Target(Middle:double): double;
    begin
      m[2]:=Middle;
      for i:=3 to N do
        m[i]:=2*m[i-1]-m[i-2]-2*D;
      Target:=M[N];
    end;

  procedure Dichotomy;
    begin
      while (abs(Right-Left)>Diff) do
        begin
          Middle:=(Left+Right)/2;
          if Target(Middle)<Required
            then Left  := Middle
            else Right := Middle;
        end;
    end;

  procedure OutputData;
    begin
      assign(output,'output.txt'); rewrite(output);
      i:=1;
      while (i<=N) and (M[i]>0.001) do inc(i);
      if i>N
        then writeln(M[j]:0:3)
        else writeln('данные некорректны');
      close(output);
    end;
```

```
  procedure InputData;
    begin
      assign(input,'input.txt'); reset(input);
      readln(m[1]);
      readln(mn);
      readln(N);
      readln(D);
      Readln(j);
      close(input);
      Left  := 0;
      Right := 1000;
      Required := mn;
    end;

begin
  InputData;
  Dichotomy;
  OutputData;
end.
```

Листинг 7.2. Текст программы к задаче «Голодание»

```
{$N+,E+}
program go97d1t4;
const
  Diff = 0.000000001;
var
  m1,m21 : double;
  i         : integer;
  Right, Left, Middle, Required  :  double;

  function Target(Middle:double): double;
    var
      Res : Double;
    begin
      Res:=m1;
      for i:=1 to 20 do Res:=Res*Middle;
      Target:=Res;
    end;

  procedure Dichotomy;
    begin
    while (abs(Right-Left)>Diff) do
      begin
        Middle:=(Left+Right)/2;
```

продолжение ⇨

Листинг 7.2 (*продолжение*)

```
        if Target(Middle)<Required
          then Left  := Middle
          else Right := Middle;
      end;
  end;

  procedure OutputData;
    begin
      assign(output,'output.txt'); rewrite(output);
      writeln(m1*Middle:0:4);
      close(output);
    end;

  procedure InputData;
    begin
      assign(input,'input.txt'); reset(input);
      readln(m1);
      readln(m21);
      close(input);
      if m21/m1 < 1
        then
          begin
            Left  := 0.0;
            Right := 1.0;
          end
        else
          begin
            Left := 1.0; Right := m21/m1;
            Middle := 1.0;
          end;
        Required := m21;
    end;
  begin
    InputData;
    Dichotomy;
    OutputData;
  end.
```

Листинг 7.3. Текст программы к задаче «День рождения»

```
program go00d2t3;
const
  Diff = 0.5;
```

```
var
  days    : array [1..365] of longint;
  i,M,N,K : longint;
  Right, Left, Middle, Required  :  longint;

  function Target(Middle:longint): longint;
    var
      Karlsons, Banka : longint;
        begin
          Karlsons:=Middle;
          Banka:=0;
          for i:=1 to 364 do
            begin
              Banka:=Banka + Days[i] - Karlsons*M;
              if Banka<0 then Banka:=0;
              if Banka>Karlsons*N then break;
            end;
          Target:=Banka;
        end;

  procedure Dichotomy;
    begin
      while (abs(Right-Left)>Diff) do
        begin
          if Right=Left+1
            then
              begin
                Middle:=Right;
                exit;
              end
            else Middle:=(Left+Right) div 2; Required:=Middle * N;
          if Target(Middle)>Required
            then Left  := Middle
            else Right := Middle;
        end;
      Required := Middle * N;
    end;

  procedure OutputData;
    begin
      assign(output,'birthday.out'); rewrite(output);
      writeln(Middle);
      close(output);
    end;
```

продолжение ⇩

Листинг 7.3 (*продолжение*)

```
  procedure InputData;
    var
      a,b,Max : longint;
    begin
      assign(input,'birthday.in'); reset(input);
      readln(M,N);
      readln(K);
      for i:=1 to 365 do days[i]:=0;
      for i:=1 to K do
        begin
          readln(A,B);
          inc(days[A],B);
        end;
      max:=days[1];
      for i:=1 to 364 do
        if days[i]>Max then max:= days[i];
      Left   := (Max div (M+N)) -1 ;
      Middle := Left;
      Right := (Max div M)+1;
    end;

  begin
    InputData;
    Dichotomy;
    OutputData;
  end.
```

Листинг 7.4. Текст программы к задаче «Water Glass»

```
{$N+,E+}
program qn00d1t2;
const
  MaxN = 50;
  Diff = 1E-18;
var
  X,Y,CX,CY            : array [1..MaxN+1] of extended;
  H                 : array [1..48] of extended;
  N,j,i,B,E,K,Bn,En : integer;
  S,MaxY            : extended;
  Px1,Py1,Px2,Py2   : extended;
  Right, Left, Middle, Required  :  extended;

  function min(a,b:extended):extended;
```

```
  begin
    if a<b then min:=a else min:=b;
  end;
function max(a,b:extended):extended;
  begin
    if a>b then max:=a else max:=b;
  end;

function Intersected (i,j:integer; YH:extended;
                      var Px,Py:extended) : boolean;
               {Пересекается ли отрезок от точки i до точки j с прямой
                Y=YH, если да - Px,Py - координаты точки пересечения}
  var
    a,b,c  : extended;
    Yes    : boolean;
  begin
    Yes    := (YH<=max(y[i],y[j])) and (YH>=min(y[i],y[j]));
    Intersected := Yes;
    if not Yes then exit;
    Py:=YH;
    a:=y[j]-y[i];
    b:=x[i]-x[j];
    c:=y[i]*(x[j]-x[i])-x[i]*(y[j]-y[i]);
    if a<>0
      then px:=(-b*YH-c)/a
      else px:=max(x[i],x[j])
  end;

function Square:extended;
  var
    i : integer;
    S : extended;
  begin
    CX:=X; CY:=Y;
    CX[Bn-1]:=Px1; CY[Bn-1]:=Py1;
    CX[En+1]:=Px2; CY[En+1]:=Py2;
    S:=0; CX[En+2]:=Px1; CY[En+2]:=Py1;
    for i:=Bn-1 to En+1 do
      S:=S+(cx[i+1]-cx[i])*(cy[i+1]+cy[i]);
    Square := Abs(S)/2;
  end;

function Target(Middle:extended): extended;
```

продолжение ⇨

Листинг 7.4 (*продолжение*)

```
    var
      i,j : integer;
    begin
      for i:=B downto 2 do
        if Intersected(i,i-1,Middle,Px1,Py1)
          then begin Bn:=i; break; end;
      for i:=E to N do
        if Intersected(i,i+1,Middle,Px2,Py2)
          then begin En:=i; break; end;
      Target:=Square;
    end;

  procedure Dichotomy;
    begin
      while (abs(Right-Left)>Diff) do
        begin
          Middle:=(Left+Right)/2;
          if Target(Middle)<Required
            then Left  := Middle
            else Right := Middle;
        end;
    end;

  procedure OutputData;
    var
      c30 : array [0..1] of string[2];
      s2  : string[2];
    begin
      c30[0]:='30'; c30[1]:='00';
      assign(output,'output.txt'); rewrite(output);
      for i:=1 to 48 do
        begin
          Str(((i-1) div 2):2, S2);
          if s2[1]=' ' then s2[1]:='0';
          writeln (s2, ':', c30[i mod 2],' ',H[i]:0:3);
        end;
    close(output);
    end;

  procedure InputData;
    var
      i: integer;
    begin
      assign(input,'input.txt'); reset(input);
```

```
      readln(N);
      for i:=1 to N do readln(x[i],y[i]);
      close(input);

      MaxY := y[1];
      for i:=1 to N do
        if y[i]>MaxY then MaxY:=y[i];

      S:=0; x[N+1]:=x[1]; y[N+1]:=y[1];
      for i:=1 to N do
        S:=S+(x[i+1]-x[i])*(y[i+1]+y[i]);
      S:=abs(S)/2;

      for i:=1 to N do
        if (y[i]=0) and (y[i+1]=0) then k:=i;      { Дно сосуда}

      B:=K;    { Указатель поиска пересекающихся отрезков с начала}
      E:=K+1;  { Указатель поиска пересекающихся отрезков с конца }
    end;

  begin
    InputData;
    H[1]:=0.0;
    for i:=2 to 48 do
      begin
        Left     := H[i-1];
        Right    := MaxY;
        Required := S*(i-1)/48;
        Dichotomy;
        H[i]     := Middle;
      end;
   OutputData;
  end.
```

Листинг 7.5. Текст программы к задаче «Последовательность Фибоначчи»

```
{$N+,E+}
program qc01d1t1;
const
  MaxN = 1000;
  Diff = 0.000001;
var
  N,Fi,Fj,Fn                    : longint;
  j,i                           : longint;
  Right, Left, Middle, Required  : longint;
```

продолжение ⇩

Листинг 7.5 (*продолжение*)

```
  function Target(Middle:longint): extended;
    var
      a,b,c : extended;
      k     : longint;
    begin
      a:=Fi;
      b:=Middle;
      for k:=i+2 to j do
        begin
          c:=a+b;
          a:=b;
          b:=c;
        end;
      Target:=c;
    end;

  procedure Dichotomy;
    begin
      while (abs(Right-Left)>Diff) do
        begin
          if Right=Left+1
            then
              begin
                Middle:=Right; exit;
              end
            else
              begin
                Middle:= (Left div 2) + (Right div 2);
                if ((Left  mod 2)<>0) and ((Right mod 2)<>0)
                  then
                    begin
                      if (Left>0) and (Right>0) then Inc(Middle);
                      if (Left<0) and (Right<0) then Dec(Middle);
                    end;
              end;
          if Target(Middle)<Required
            then Left  := Middle
            else Right := Middle;
        end;
  end;

procedure OutputData;
```

```
var
  a,b,c,Fn : extended;
  k        : longint;
begin
  assign(output,'output.txt'); rewrite(output);
  a:=Fi;
  b:=Middle;
  if n=i   then c:=Fi;
  if n=i+1 then c:=Middle;
  if n>i+1
    then
      for k:=i+2 to n do
        begin
          c:=a+b;
          a:=b;
          b:=c;
        end;
  if n<i
    then
      for k:=i-1 downto n do
        begin
          a:=c-b;
          c:=b;
          b:=a;
        end;
  Fn:=c; writeln(Fn:0:0); close(output);
end;

procedure InputData;
  var
    ti,tFi : longint;
  begin
    assign(input,'input.txt'); reset(input);
    readln(i,Fi,j,Fj,N);
    close(input);
    if j<i
      then
        begin
          ti:=i;  tFi:=Fi;
          i:=j;   Fi:=Fj;
          j:=ti;  Fj:=tFi;
        end;
    if j=i+1
```

продолжение ⇨

Листинг 7.5 (*продолжение*)

```
        then
          begin
            inc(j);
            Fj:=Fi+Fj;
          end;
      Left := -2000000000; Right := 2000000000; Required := Fj;
    end;
  begin
    InputData;
    Dichotomy;
    OutputData;
  end.
```

Листинг 7.6. Текст программы к задаче «Equipment Box»

```
{$N+,E+}
program ce99t7;
const
  Diff : extended = 1E-3;
var
  A1,A2,B1,B2,i,T                 : longint;
  Right, Left, Middle, Required  : extended;
  Exist                            : boolean;

function Target(Middle:extended): extended;
  begin
    if ( A1 > (B2*sin(Middle)+A2*cos(Middle) ))  and
       ( B1 > (B2*cos(Middle)+A2*sin(Middle) ))
      then
        begin
          Exist:=true; Diff:=1e+10; Target:=0; exit;
        end;
    if ( A1 <= (B2*sin(Middle)+A2*cos(Middle)) ) then Target:=2;
    if ( B1 <= (B2*cos(Middle)+A2*sin(Middle)) ) then Target:=0;
  end;

  procedure Dichotomy;
    begin
      while (abs(Right-Left)>Diff) do
        begin
          Middle:=(Left+Right)/2;
          if Target(Middle)<Required
            then Left  := Middle
```

```
          else Right := Middle;
      end;
  end;

procedure Swaps(var A,B:longint);
  var
    t : longint;
  begin
    t:=A; A:=B; B:=t;
  end;

procedure InputData;
  begin
    readln(A1,B1,A2,B2);
    if A1>B1 then Swaps(A1,B1);
    if A2>B2 then Swaps(A2,B2);
  end;
begin
  assign(output,'output.txt'); rewrite(output);
  assign(input,'input.txt'); reset(input);
  readln(T);
  for i:=1 to T do
    begin
      InputData;
      Exist:= (A1>A2) and (B1>B2);
      if not Exist
        then
          begin
            Required := 1;
            Diff := 1E-3;
            Left := 0;
            Right := Pi/2;
            Dichotomy;
          end;
      if Exist
        then writeln('Escape is possible.')
        else writeln('Box cannot be dropped.');
    end;
  close(input); close(output);
end.
```