Методические рекомендации по подготовке к сдаче практической части предпрофессионального экзамена для выпускников, обучавшихся в рамках проекта «Инженерный класс в московской школе»
Информатика

Представленные на предпрофессиональном экзамене задачи можно разделить на две большие группы:

1. **Программирование + физика.** К этой группе относятся задачи, для решения которых требуется написать достаточно сложную программу. В свою очередь, для правильной работы программы требуется решить элементарную задачу по физике.
2. **Физика + программирование.** К этой группе относятся задачи, для решения которых требуется решить физическую задачу численно.

Примером задачи первого типа может служить задача 2016/2017 учебного года.

Задача 1

Чтобы уравновесить тяжелый груз, висящий на одном конце легкого стержня, требуется уравновесить его с помощью гирь. Зная массу груза, массу одинаковых гирь из набора и расстояния от точек приложения сил до точки подвеса, подобрать сочетание точек подвеса гирь, которое позволит уравновесить груз.

Формат ввода

В первой строке вводится натуральное число N – число гирь и точек подвеса. N не превышает 10.

Далее через пробел в одной строке вводятся вещественные положительные числа M, L, m_w – соответственно, масса груза, длина стержня от точки подвеса до груза, массы гирь.

Далее следует N строк, в каждой из которых содержится одно вещественное положительное число li – расстояние от точки, на которой можно закрепить гирю до точки подвеса.

Никакие числа не превышают 1000.

Груз считается уравновешенным, если моменты сил, действующих на него и на набор гирь, различаются не более чем на 5%.

На выходе программа должна выдать через запятую номера точек подвеса, или вывести 0, если уравновесить грузы невозможно.

Как можно видеть, физическая часть задачи достаточно проста и сводится к простому расчету суммы моментов сил. Предполагается решение этой задачи с использованием рекурсивного перебора.

Решение задачи 1:

```
program z14;
const
 M=10; //максимум отрезков
 eps=0.05;

type
  tmas=array[1..M] of integer;

procedure swap(var x,y:integer);
var z:integer;
begin
 z:=x;
 x:=y;
 y:=z;
end;

function check(num,mas:tmas; kolvo:integer; w1,l1,mw:real):boolean;
var
   w2:real;
   i:integer;
begin
 w2:=0;
 for i:=1 to kolvo do
  begin
   w2:=w2+mas[nums[i]];
  end;
 check:=abs(l1*w1-mw*w2)/(l1*w1)<=eps;
end;

procedure output(num:tmas;kolvo:integer);
var i:integer;
begin
 for i:=1 to kolvo-1 do
  write(num[i],',');
 writeln(num[kolvo]);
end;

procedure perest(nums,mas:tmas; cur,kolvo,n:integer;w,l1,l2:real; var
flag:boolean);
var i:integer;
begin
 if cur>kolvo then
  begin
   flag:=check(nums,mas,kolvo,w,l1,l2);
   if flag then
    output(nums,kolvo)
  end
  else
  begin
   i:=cur;
   while (i<=n) and not flag do
    begin
```

```
      swap(nums[i],nums[cur]);
      perest(nums,mas,cur+1,kolvo,n,w,l1,l2,flag);
      swap(nums[i],nums[cur]);
      i:=i+1;
     end;
  end;
end;

var mas,nums:tmas;
    j,n:integer;
    flag:boolean;
    w,l1,m_w:real;
begin
 readln(n); //чтение входных данных
 read(w);
 read(l1);
 readln(m_w);
 for j:=1 to n do
  begin
   read(mas[j]);
   nums[j]:=j;
  end;
 j:=1;
 flag:=false; //переменная хранит информацию о достижении результата
 while (j<=n) and not flag do
  begin
   perest(nums,mas,1,j,n,w,l1,m_w,flag); //проводим перестановки, ведем поиск
в ширину
   //переставляем номера, массив mas только для справки
   j:=j+1;
  end;
 if not flag then
  writeln(0);
end.
```

Пример задачи второго типа из 2016/2017 учебного года приведен ниже:

Задача 2

Тело движется вдоль оси *Ox* под воздействием силы F(x), сонаправленной с осью *Ox*. В общем виде F(x) определена как $F=a(x+1)^{1/2}+b(x+2)+cx^2+d*\ln(x+1)$, где a,b,c,d – коэффициенты, которые могут меняться на разных отрезках.

Зная количество отрезков, коэффициенты уравнения кривых и границы отрезков, на которых эти кривые применяются, найдите совершенную над телом работу силы f(x) с точностью $\varepsilon = 10^{-5}$.

Формат ввода

В первой строке вводится натуральное число N – число отрезков. N не превышает 5.

Далее следует N строк, в каждой из которых через пробел записаны шесть вещественных положительных чисел x_i, x_{i+1}, a,b,c,d – соответственно, границы отрезка, и коэффициенты при кривой на этом отрезке.

Гарантируется, что разрывов нет и каждый следующий отрезок начинается там, где закончился предыдущий.

Гарантируется, что каждое уравнение имеет математический смысл.

Никакие числа не превышают 1000.

Точность ε считать достигнутой, когда при вычислении интегральной суммы уменьшение отрезка x вдвое приводит к изменению суммы меньше, чем на ε

На выходе программа должна выдать вещественное число – совершенную над телом работу силы f(x). Все величины задаются в системе СИ, ответ привести в джоулях.

Решение этой задачи требует существенно большего багажа знаний по физике и сводится к численному интегрированию.

Решение задачи 2:

```
program z30;
//Задача о работе силы. Интегрирование методом прямоугольников
const
 eps = 0.00001;

function f(x,a,b,c,d:real):real;
begin
 f:=a*sqrt(x+1)+b*(x+2)+c*sqr(x)+d*ln(x+1);
end;

function work(x0,x1,a,b,c,d:real):real;
var vt,pv,x,h:real;
begin
 vt:=sqr(f((x1+x0)/2,a,b,c,d))*(x1-x0);
 h:=(x1-x0)/2;
 repeat
  pv:=vt;
  x:=x0;
  vt:=0;
  while x<x1 do
   begin
    vt:=vt+(f(x,a,b,c,d))*h;
    x:=x+h;
   end;
   h:=h/2;
 until abs(vt-pv)<eps;
 work:=vt;
end;

var w,x0,x1,a,b,c,d:real;
    i,n:integer;
begin
 w:=0;
 readln(n);
 for i:=1 to n do
  begin
   read(x0);
```

```
    read(x1);
    read(a);
    read(b);
    read(c);
    readln(d);
    w:=w+work(x0,x1,a,b,c,d);
   end;
  writeln(w);
end.
```

Рассмотрим более подробно темы, которые необходимо знать выпускникам.

1. Численное интегрирование

Задачи второго типа представляют из себя задачи по физике по темам, известные учащимся из курса средней школы. Однако в эти задачи добавлены условия, исключающие их аналитическое решение методами, которые проходят в школе. Так, например, в уже представленной задаче №30 действующая на тело сила зависит от кооординаты x, вследствие чего неприменима формула

$$A = FS \tag{1}$$

Идея решения задачи заключается в следующем: разделим очередной участок пути на отрезки, достаточно малые, чтобы считать силу F постоянной на каждом отрезке. К каждому из таких отрезков применима указанная выше формула. Фактически ученику предлагается численно проинтегрировать функцию, поскольку ее вид не позволяет проинтегрировать ее аналитически.

Наиболее простым методом численного интегрирования является метод прямоугольников. Суть метода заключается в том, что мы разбиваем отрезок на n равных частей. Обозначим длину частичного отрезка за h. В дальнейшем будем называть h шагом интегрирования. Тогда, если взять в качестве точек x_i левые концы частичных отрезков, получим соотношение

$$\int_a^b f(x)dx \approx \sum_{i=0}^{n} f(x_i)\, h \tag{2}$$

С уменьшением шага достигается все большая точность.

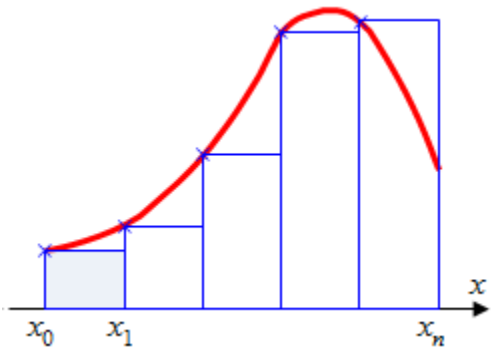

Рис. 1. Иллюстрация применения метода прямоугольников.

Пример аппроксимации функции прямоугольникам показан на рис. 1. Ниже приведен фрагмент программы, рассчитывающей интеграл методом прямоугольников.

```
{a,b - границы интегрирования, eps – точность, h - шаг, f(x) - целевая
функция, psum - сумма на предыдущей итерации, sum - сумма на текущей
итерации}
 sum:=f(a)*(b-a);
 h:=(x1-x0)/10;
 repeat
  psum:=sum;
  x:=a;
  while x<b do
   begin
    sum:=sum+f(x)*h;
    x:=x+h;
   end;
   h:=h/2;
 until abs(sum-psum)<eps;
```

Возможны дальнейшие улучшения метода, в частности, вынос шага за скобки и взятие значения не с края, а из середины шагового отрезка, как показано на рис. 2.

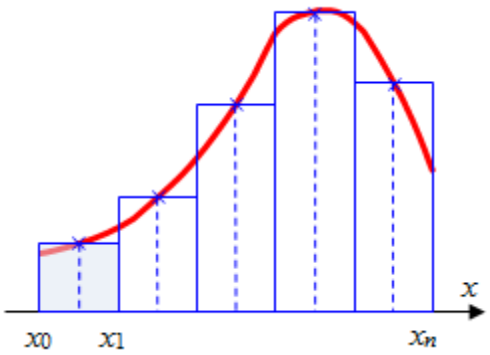

Рис. 2. Иллюстрация метода средних прямоугольников.

Метод средних прямоугольников позволяет достичь большей точности. Более продвинутые методы численного интегрирования, такие, как метод трапеций или метод Симпсона, желательны, но не обязательны.

2. Рекурсивный перебор

В узком смысле слова рекурсия представляет собой вызов процедуры или функции внутри самой себя. Применительно к ряду задач, в число которых входит задача перебора элементов множества, рекурсия представляет собой решение задачи через самоподобие, и сведение ее с помощью самоподобия к некоторому элементарному случаю.

Рассмотрим задачу получения всех перестановок элементов какого-то множества. Элементарным случаем в этой задаче является случай одного

элемента: существует одна-единственная перестановка. Для двух элементов *a*,*b* будем действовать так: сначала поставим на первое место *a* и рассмотрим все перестановки оставшегося элемента *b*. Потом поставим на первое место *b* и рассмотрим все перестановки оставшегося элемента *a*. Получим *ab* и *ba*. Для трех элементов мы сначала поставим на первое место *a* и рассмотрим все перестановки *b* и *c*. Проиллюстрируем все возможные варианты в виде таблицы.

Таблица 1. Все возможные перестановки *a*, *b*, *c*.

Исходные данные	Первый уровень	Второй уровень
a b c	a | b c	a b | c
		a c | b
	b | a c	b a | c
		b c | a
	c | a b	c a | b
		c b | a

Ниже приведены две программы, получающие все перестановки в строке:

```
program shakestr;

procedure perest(head,tail:string);
var newhead,newtail:string;
    i:integer;
begin
 if length(tail)=0 then //если все данные взяты, то
  writeln(head) //вывести ответ
  else //иначе
  begin
   for i:=1 to length(tail) do //перебираем доступные нам символы
    begin
     newhead:=head+copy(tail,i,1); //добавляем в ответ очередной символ
     newtail:=tail;
     delete(newtail,i,1); //обновляем доступные символы
     perest(newhead,newtail);//уходим в рекурсию. Важно: newtail и newhead,
чтобы не терять данные на возврате на этот уровень рекурсии
    end;
  end;
end;

var
 tail:string;
begin
 readln(tail); //берем исходные данные
 perest('',tail); //запускаем перестановки: доступны все данные, ответа еще
нет.
end.
```

и в массиве:

```
program shakearray;
const
 N=5;
type tmas = array [1..N] of integer;

procedure init(var mas:tmas);
var i:integer;
begin
 for i:=1 to N do
  mas[i]:=i;
end;

procedure change(var x,y:integer);
var z:integer;
begin
 z:=x;
 x:=y;
 y:=z;
end;

procedure perest(mas:tmas; cur:integer);
var i:integer;
begin
 if cur>N then
  begin
   writeln(mas);
  end
  else
  begin
   for i:=cur to N do
    begin
     change(mas[i],mas[cur]);
     perest(mas,cur+1);
     change(mas[i],mas[cur]);
    end;
  end;
end;

var mas:tmas;
begin
 init(mas);
 perest(mas,1);
end.
```

Коснемся более подробно принципа работы программы, осуществляющей перестановку элементов в массиве. Переменная cur отделяет ту часть массива, которая уже добавлена в ответ от той части, из которой еще можно брать элементы для добавления. Проще говоря, от начала до cur идет ответ, от cur включительно и до конца идет набор данных для перебора, cur – ячейка, которую прямо сейчас добавляют в ответ. Повторный обмен после рекурсивного вызова процедуры perest предназначен для избежания искажений данных при перестановках. Читателю предлагается убрать повторный обмен и посмотреть на изменение результатов перестановок.

Задача поиска сочетаний C_n^k технически отличается от задачи перебора только условием выхода из рекурсии: прекращать работу программы надо в момент, когда в ответе образуется k элементов.

Для самостоятельного упражнения читателю рекомендуется решить с помощью рекурсивного перебора задачу о рюкзаке, задачу коммивояжера и задачу о трубе.

3. Динамическое программирование

Низкая скорость работы рекурсии связана с многочисленными проходами по одним и тем же промежуточным состояниям системы. В связи с этим существует класс задач и объемы данных, для которого рекурсия неэффективна. Метод, основанный на однократном расчете промежуточных состояний системы, называется динамическим программированием.

Лучше всего проиллюстрировать этот подход на достаточно старой задаче о свинье-копилке.

Задача:

В свинью-копилку помещается 1000 г монет. В наличии до 5 типов монет, известны их достоинство и масса, оба числа натуральные. Считая количество монет каждого типа неограниченным, рассчитать, на какую максимальную сумму можно полностью наполнить копилку.

Решение:

Очевидно, что задача теоретически может быть решена рекурсивным перебором. Однако скорость такого решения не укладывается в любые разумные сроки. Попробуем решить задачу иначе.

Предположим, что у нас есть куча монет массой *M*. Она была получена из кучи меньшей массы *M'* путем добавления одной монеты. Зная виды монет, мы можем предположить, из каких меньших куч могла быть получена куча массой *M* и выбрать наиболее выгодный вариант. Проиллюстрируем на примере. Пусть у нас есть монеты массой 2 г и ценой 4 коп. и монеты массой 3 г и ценой 5 коп.

Масса	0	1	2	3	4	5	6
Цена	0	н	4	5	8	9	12

Начальная куча массой 0 г имеет цену 0 коп.

Куча массой 1 г с такими монетами получена быть не может.

Куча массой 2 г может быть получена путем добавления монеты массой 2 г в начальную кучу. Цена этой кучи – 4 коп.

Куча массой 3 г может быть получена путем добавления монеты массой 3 г в начальную кучу. Цена этой кучи – 5 коп.

Куча массой 4 г может быть получена только путем добавления монеты массой 2 г в кучу массой 2 г. Куча массой 1 г получена быть не может, поэтому в нее нельзя добавить монету массой 3 г. Цена этой кучи – 8 коп.

Куча массой 5 г может быть получена или добавлением монеты массой 2 г в кучу массой 3 г, или добавлением монеты массой 3 г в кучу массой 2 г. Цена этой кучи 9 коп.

Куча массой 6 г может быть получена или добавлением монеты массой 2 г в кучу массой 4 г, или добавлением монеты массой 3 г в кучу массой 3 г. В первом случае цена кучи 12 коп., во втором – 10 коп. Выбираем первый вариант.

Нетрудно видеть, что задача перекликается с задачей №22 ЕГЭ по информатике. Технически ее можно решить с помощью массива целых чисел, проиндексированного от 0 до 1000, где индекс – масса кучи монет, а значение ячейки массива – цена этой кучи.

Аналогичные рассуждения и подход применялись при решении задачи демонстрационного варианта.