***1.10.* Найти случайное событие X из равенства Х + А + Х + А = В**

**Изображение выглядит как зарисовка, рисунок, графическая вставка, Штриховая графика

Автоматически созданное описание**

Х + А + Х + А = В

X + A = B

X = B / A + X \* A

***2.5.* Черный и белый короли находятся соответственно на первой и третьей горизонталях шахматной доски. На одно из незанятых полей первой или второй горизонтали наудачу ставится ферзь. Определить вероятность того, что образовавшаяся позиция матовая для черного короля, если положения королей равновозможны на любых полях указанных горизонталей.**

Клеток доступных для белого короля – 8, клеток доступных для чёрного короля – 8, клеток доступных для ферзя – 15.

Рассмотрим случаи, когда такое возможно:

Случай 1:

Изображение выглядит как снимок экрана, прямоугольный, шахматы, настольная игра

Автоматически созданное описание

Чёрный король находится в левом углу, ферзь в любой клетке первой горизонтали, кроме соседствующей с чёрным королём, либо в одной из двух клеток над ЧК, белый король – в первых двух клетках. Стоит учесть, что подобная ситуация аналогична и для правого конца диагонали, т.е. вероятность подобного исхода стоит умножить на 2.

БК – 2/8

ЧК - 1/8

Ф - 8/15

I = 2/8 \* 1/8 \* 8/15 \* 2 = 1/30

Случай 2:

Изображение выглядит как прямоугольный, снимок экрана, шахматы, шахматная фигура

Автоматически созданное описание

ЧК находится в любой из клеток, кроме двух крайних. БК занимает место ровно над ЧК. Ферзь находится на первой диагонали, на любой клетке, кроме соседствующих с ЧК, либо между королями.

БК = 1/8

ЧК = 6/8

Ф = 6/15

II = 1/8 \* 6/8 \* 6/15 = 3/80

Случай 3:

ЧК на a1, БК на c3, Ф на b2, плюс зеркальная

III = 2/8 \* 1/8 \* 1/15 = 2/960

Случай 4:

Координата БК отличается от координаты ЧК на 1, Ф должен стоять над ЧК

IV = 6/8 \* 2/8 \* 1/15 = 1/80

Итого 1/30 + 3/80 + 2/960 + 1/80 = 41/480 = 0.08541

Проверим, составив программную модель на языке Python:

from random import randint  
  
tr = 10000000 # количество тестов  
pos = 0 # счётчик положительных результатов  
  
def occupied(x, y):  
 if (y == 2) and (abs(x - wPos) < 2): # под ударом БК  
 return True  
 if (y == qPosY) and (x != qPosX): # под ударом Ф  
 return True  
 if (y != qPosY) and (abs(x - qPosX) <= 1): # на одной диагонали с Ф  
 return True  
 return False  
  
def check(x):  
 for i in range (-1, 2):   
 for j in range (1, 3): # клетки первой и второй диагонали  
 curX = x + i # клетки слева, над, справа от ЧК  
 if 0 < curX < 9:   
 if not occupied(curX, j): return False  
 return True  
  
  
for i in range(tr):  
 if i / tr \* 100 % 1 == 0: print(i / tr \* 100)  
 fl = False  
 wPos = randint(1, 8)  
 bPos = randint(1, 8)  
 qPosY = randint(1, 2)  
 qPosX = randint(1, 8)  
 while (qPosY == 1 & qPosX == bPos): # переопределяем коорд ферзя, если совпадают с ЧК  
 qPosY = randint(1, 2)  
 qPosX = randint(1, 8)  
  
 if check(bPos): pos += 1  
  
  
print("---------------------------")  
print(pos/tr)

Результат работы программы:



***3.11.*** **На отрезке длиной l независимо одна от другой поставлены две точки, положение каждой из которых равновозможно на этом отрезке. Определить вероятность того, что из трех получившихся частей одного отрезка можно построить треугольник.**

Для того, чтобы из трех отрезков можно было построить треугольник, необходимо выполнение неравенства треугольника: сумма длин любых двух отрезков должна быть больше длины третьего отрезка.

Рассмотрим отрезок длиной l и две независимо поставленные на нем точки A и B. Будем считать, что A левее B.

Обозначим длины полученных отрезков как a = |AB|, b = |O1A|, c = |O2B|, где O1 - левый конец отрезка, O2 - правый.

Изображение выглядит как зарисовка, линия, диаграмма, белый

Автоматически созданное описание

Тогда неравенство треугольника запишется как:

a + b > c

a + c > b

b + c > a

Из первого неравенства следует, что точка B не может лежать правее середины отрезка l, т.е. c <= l/2.

Из второго неравенства следует, что точка A не может лежать левее середины отрезка l, т.е. b >= l/2.

Из третьего неравенства следует, что a < l/2, что выполняется автоматически, так как a = c - b, c <= l/2, b >= l/2.

Таким образом, вероятность того, что из трех получившихся частей отрезка можно построить треугольник, равна вероятности того, что точка B лежит правее середины отрезка, а точка A левее середины отрезка. Эта вероятность равна произведению вероятностей каждого из описанных событий, т.е. 0.5 \* 0.5.

Ответ: вероятность равна ¼ = 0.25.

Проверим результат, смоделировав данную ситуацию при помощи программы на языке Python.

from random import randint  
  
len = 10000000000000 # количество дискретных точек на прямой  
tr = 100000000 # количество тестов  
pos = 0 # счётчик положительных результатов  
  
for i in range(tr):  
 if (i / tr \* 100 % 1 == 0): print(i / tr \* 100)  
 p1 = randint(0, len+1) # определим точки при помощи RNG  
 p2 = randint(0, len+1)  
 l1 = min(p1, p2) # длина первого отрезка равна координате левой из двух точек  
 l2 = max(p1, p2) - min(p1, p2) # длина среднего отрезка равна разности координат точек  
 l3 = len - max(p1, p2) # длина последнего отрезка равна разности координаты конца отрезка и коор. правой точки  
  
 if max(l1, l2, l3) < l1 + l2 + l3 - max(l1, l2, l3): # для определения положительного результата имеет смысл сравнивать лишь наибольший отрезок с суммой остальных  
 pos += 1   
print("-------------------------------------")  
print(pos/tr)

Результат работы программы:



Очевидно, результат моделирования совпадает с теоретическим

***4.32.* В урне имеются два шара —белый и черный. Производятся извлечения по одному шару до тех пор, пока не появится черный, причем при извлечении белого шара в урну возвращается этот шар и добавляется еще два белых шара. Определить вероятность того, что при первых пятидесяти опытах черный шар не будет извлечен.**

При первом извлечении вероятность равна 1/2, при втором 3/4, третьем – 5/6.

Нетрудно заметить, что вероятность P для n-ого извлечения будет равна .

Тогда вероятность того, что черный шар не будет извлечен при первых пятидесяти попытках будет равна . Получим очень длинную дробь, в числителе которой находится произведение всех нечётных чисел от 1 до 99, а в знаменателе - произведение всех чётных чисел от 2 до 100. Умножим числитель и знаменатель на знаменатель, чтобы получить 100! в числителе. Рассмотрим знаменатель: там находится квадрат произведение всех чётных чисел от 2 до 100. Если разделить каждый элемент на 2, получится 50!. Вынося из скобок коэффициенты, не забудем про квадрат. В итоге имеем . Согласно калькулятору Wolfram Alpha, это примерно равно 0.0795892.

Проверим результат, смоделировав данную ситуацию при помощи программы на языке Python.

from random import randint  
  
tr = 10000000 # количество тестов  
pos = 0 # счётчик положительных результатов  
  
for i in range(tr):  
 if i / tr \* 100 % 1 == 0: print(i / tr \* 100)  
 counter = 0  
 ar = ["w", "b"]  
 while counter < 50:  
 if ar[randint(0, len(ar) - 1)] == "w":  
 ar.append("w")  
 ar.append("w")  
 counter += 1  
 else: break  
 if counter >= 50: pos += 1  
  
print(pos / tr)

Результат работы программы:



***5.34*. В электропоезд, состоящий из η вагонов, входят к(к >= η ) пассажиров, которые выбирают вагоны наудачу. Определить вероятность того, что в каждый вагон войдет хотя бы один пассажир.**

Попробуем решить задачу от обратного. Пусть A – событие, при котором ни в одном вагоне нет ни одного пассажира, Bi событие, при котором в i-м вагоне нет пассажиров. Тогда искомая вероятность

. События Bi независимые. Пусть B`i – шанс, что в некоторых i вагонах пусто.

Вероятность такого события равна произведению вероятностей того, что отдельный пассажир выберет любой выгон из множества длиной n – I, т. е. .

Используем вероятность того, что перестановка длиной i пустует для нахождения P(C).

.

Для простоты вычислений разложим ряд в виде явной суммы.

Проверим результат, смоделировав данную ситуацию при помощи программы на языке Python.

from random import randint  
  
tr = 10000 # количество тестов  
pos = 0 # счётчик положительных результатов  
n =   
k =   
  
for i in range(tr):  
 if i / tr \* 100 % 1 == 0: print(i / tr \* 100)  
 a = [0] \* n  
  
 for j in range(k):  
 a[randint(0, len - 1)] += 1  
  
 if not 0 in a: pos += 1  
  
print(pos/tr)

Переменным n и k присвоим значения 6 и 9 соответственно.

Проверим формулу при помощи программы Wolfram Alpha, там она имеет вид

1 - Sum[(-1)^(i-1)\*(6!/i!(6-i)!)\*((1-i/6)^9),{i,1,6}]

Результат равен 0.189043

Результат работы модели на языке Python:



Также проверим для n и k равным 2 и 100; 12 и 13

2, 100:

Wolfram Alpha – ~1

Python – 1

12, 13:

Wolfram Alpha – 0.000349

Python - 0.00033