

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

Факультет Информатика и вычислительная техника

Кафедра Кибербезопасность информационных систем

**ЛАБОРАТОРНАЯ РАБОТА № 4**

на тему «Методы Монте-Карло»

Выполнил обучающийся гр. ВКБ32

Беседин Данил Евгениевич

Проверил

Доцент, Савельев Василий Александрович

Ростов-на-Дону

2022

Вероятностное программирование включает в себя широкий выбор задач, моделей и методов, опирающихся на статистические закономерности. При решении задач может возникать необходимость использовать случайные (псевдослучайные) числа. Практически все современные системы программирования содержат генераторы последовательности псевдослучайных чисел. Однако качество таких генераторов может очень сильно отличаться. Поэтому при использовании генераторы следует тестировать на качество последовательности с помощью наборов тестов dieharder или тестов NIST SP800-22. Если встроенный генератор не проходит тесты – следует найти или реализовать качественный PRNG (pseudorandom numbers generator). Использование библиотечных или опубликованных PRNG не возбраняется.

|  |
| --- |
| from sys import argv  import graphlib as gr  if len(argv)>1:      if argv[1]!='/?':          filename=argv[1]      else:          print('Бросаем точки')          exit()      if len(argv)>4:          if argv[2]=='/b':              is\_bin=True          else:              is\_bin=False          start=int(argv[3])          fin=int(argv[4])      else:          is\_bin=False          start=0          fin=3  else:      is\_bin=False      start=0      fin=3      filename='input.txt'  def prima(cpoint,tpoint,rebrs,length=0,dellst=[],lens=[],path=''):      if length==0:          lens=[]      path+=str(cpoint)+'-'      if cpoint==tpoint:          lens.append(length)          #print(Брошено точек',path[:-1],':',length)          return None      for num in dellst:          rebrs.pop(num)      dellst=[]      for c,d in enumerate(rebrs):          if (cpoint in rebrs[c]):              dellst.append(c)      dellst.reverse()      for num in dellst:          if rebrs[num][0]!=cpoint:              nextp=rebrs[num][0]          else:              nextp=rebrs[num][1]          sqr\_count(nextp,tpoint,rebrs[:],length+rebrs[num][2],dellst,lens,path)      if not lens:          otv='{} {} -1'.format(cpoint,tpoint)      else:          otv=min(lens)      lens=''      return otv  graph=prima(filename,is\_bin)  length=sqr\_count(start,fin,graph)  print(length) |

Тест

|  |
| --- |
| class Test3(unittest.TestCase):      def test\_1(self):          self.assertEqual(sqr\_count(0,3,graph), 20)      def test\_2(self):          self.assertEqual(sqr\_count(2,3,graph), 11)      def test\_3(self):          self.assertEqual(sqr\_count(3,0,graph), 20)      def test\_4(self):          self.assertEqual(sqr\_count(3,2,graph), 11)      def test\_5(self):          self.assertEqual(sqr\_count(2,4,graph), '2 4 -1')      def test\_6(self):          self.assertEqual(sqr\_count(4,2,graph), '4 2 -1')      def test\_7(self):          self.assertEqual(sqr\_count(5,6,graph), '5 6 -1')      def test\_8(self):          self.assertEqual(sqr\_count(6,5,graph), '6 5 -1')      def test\_9(self):          self.assertEqual(sqr\_count(1,3,graph), 15)      def test\_10(self):          self.assertEqual(sqr\_count(3,1,graph), 15)  if \_\_name\_\_ == "\_\_main\_\_":      graph=gr.load\_graph(filename,is\_bin)      unittest.main() |



**Задание 2 «Приближенное вычисление площади фигуры»**

Методы Монте-Карло или методы статистических испытаний – это группа численных методов, основанных на воспроизведении большого числа реализаций случайного процесса. Таким образом, суть метода заключается в статистическом моделировании случайных процессов, численном моделировании реализаций случайных процессов и оценивании параметров по реализациям случайных процессов методами математической статистики.

Под численным статистическим моделированием обычно понимают реализацию с помощью компьютера вероятностной модели некоторого объекта с целью оценивания изучаемых интегральных характеристик на основе закона больших чисел.

Свое экзотическое название метод получил от города Монте-Карло (княжество Монако), который известен благодаря своему казино, поскольку именно рулетка является одним из самых широко известных генераторов случайных чисел.

Применим метод статических испытаний или метод Монте-Карло к задаче вычисления площади геометрической фигуры на плоскости.

Метод заключается в следующем. Поместим данную фигуру в квадрат и будем наугад бросать точки в этот квадрат. Будем исходить из того, что чем больше площадь фигуры, тем чаще в нее будут попадать точки. Таким образом, при большом числе N точек, наугад выбранных внутри квадрата, доля точек, содержащихся в данной фигуре k, приближенно равна отношению площади этой фигуры и площади квадрата.

|  |
| --- |
| vector<int> topologySortInit(graphNotWeighted &G) {      int n = G.size();      vector<int> resultOrder;      vector<char> color(n, 'w');      for (int v = 0; v < n; v++)          if (color[v] == 'w')              topologySortProc(G, v, color, resultOrder);      reverse(resultOrder.begin(), resultOrder.end());      return resultOrder;  }  void topologySortProc(graphNotWeighted &G, int start, vector<char> &color, vector<int> &order) {      stack<pair<int, int> > S;      S.push(make\_pair(start, -1));      color[start] = 'g';      while (S.size()) {          int v = S.top().first;          int u = S.top().second;          int w = -1;          for (int i = 0; i < G[v].size(); i++)              if (G[v][i] != u && color[G[v][i]] == 'w') {                  w = G[v][i];                  break;              }          if (w == -1) {              S.pop();              color[v] = 'b';              order.push\_back(v);          }          else {              S.top().second = w;              if (color[w] == 'w') {                  S.push(make\_pair(w, -1));                  color[w] = 'g';              }          }      }      return;  } |

**Задание 3**

Модифицировать алгоритм топологической сортировки так, чтобы он отыскивал один из циклов, если таковые имеются, и производил топологическую сортировку, если циклов нет. Написать тесты и отладить программу.

|  |
| --- |
| graphNotWeighted transposingGraph(graphNotWeighted &G) {      int n = G.size();      graphNotWeighted GT(n);      vector<int> count(n);      for (int i = 0; i < n; i++)          for (int j = 0; j < G[i].size(); j++)              count[G[i][j]]++;      for (int i = 0; i < n; i++)          GT[i].reserve(count[i]);      for (int i = 0; i < n; i++)          for (int j = 0; j < G[i].size(); j++)              GT[G[i][j]].push\_back(i);      return GT;  }  graphNotWeighted getComponent(graphNotWeighted &G, int start, vector<char> &color) {      stack<pair<int, int> > S;      graphNotWeighted component(G.size());      bool isEmpty = false;      for(int p = 0; p < G[start].size(); p++)          if(color[G[start][p]] == 'w')              isEmpty = true;      if(isEmpty == false) {          component[start].push\_back(start);          color[start] = 'b';          return component;      }      S.push(make\_pair(start, -1));      while (S.size()) {          int v = S.top().first;          int u = S.top().second;          int w = -1;          for (int i = 0; i < G[v].size(); i++)              if (G[v][i] != u && color[G[v][i]] == 'w') {                  w = G[v][i];                  break;              }          if (w == -1) {              S.pop();              color[v] = 'b';          }          else {              S.top().second = w;              if (color[w] == 'w') {                  component[w].push\_back(v);                  S.push(make\_pair(w, -1));                  color[w] = 'g';              }          }      }      return component;  }  vector<graphNotWeighted> strongConnectedComponents(graphNotWeighted &G) {      int n = G.size();      vector<int> order = topologySortInit(G);      graphNotWeighted GT = transposingGraph(G);      vector<graphNotWeighted> strongConnectedComponents;      vector<char> color(n, 'w');      for (int i = 0; i < n; i++)          if (color[order[i]] == 'w')              strongConnectedComponents.push\_back(getComponent(GT, order[i], color));      return strongConnectedComponents;  } |