Введение в теорию графов. Задачи поиска пути в графе.

Хайрулин Сергей Сергеевич
s.khayrulin@gmail.com

Overview

- Алгоритм Флойда-Уоршелла
- Алгоритм Форда-Беллмана.
- Алгоритм Дейкстры
- Алгоритм Дейкстры для разреженных графов
- Алгоритм поиска A*.
- Волновой Алгоритм.

Литература и др. источники

- Дональд Эрвин Кнут. Искусство программирования (Том 1, 2, 3) // Вильямс 2015.
- Альфред В. Ахо, Джон Э. Хопкрофт, Джеффри Д. Ульман. Структуры данных и алгоритмы // Вильямс 2000.
- Емеличев В. А., Мельников О. И., Сарванов В. И., Тышкевич Р. И. Лекции по теории графов // М.: Наука, 1990.
- Харари Ф. Теория графов // М.: Мир, 1973.
- Косточка А. В. Дискретная математика. Часть 2 //Новосибирск: НГУ, 2001.
- Котов В. Е., Сабельфельд В. К. Теория схем программ // Наука 1991.
- http://algolist.manual.ru
- ….

Обзор алгоритмов.

Название алгоритма	**Сложность**
Флойда-Уоршела	$O(\|V\|^3)$
Форда-Беллмана	$O(\|V\|\|E\|)$
Дейкстра	$O(\|V\|^2)$
Алгоритм Дейкстра для разреженных графов	$O(\|E\|log(\|V\|))$
A*	-
Волновой Алгоритм (для невзвешанных графов)	

Алгоритм Флойда-Уоршелла

Находит расстояние от каждой вершины до каждой за количество операций порядка . Веса могут быть отрицательными, но у нас не может быть циклов с отрицательной суммой весов рёбер.

```
for k = 1 to n
  for i = 1 to n
    for j = 1 to n
      W[i][j] = min(W[i][j], W[i][k] + W[k][j])
```

Алгоритм Форда-Беллмана

- Находит расстояние от одной вершины (дадим ей номер 0) до всех остальных за количество операций порядка $O(|V||E|)$.

```
for v ∈ V
  for i = 0 to |V| − 1
    do A_vi = +∞
A_s0 = 0
for i = 1 to |V| − 1
  do for (u, v) ∈ E
    if A_vi > A_u,i−1 + w(u, v)
      then A_vi = A_u,i−1 + w(u, v)
           P_vi = u
```

Алгоритм Дейкстры

- Находит расстояние от одной вершины (дадим ей номер 0) до всех остальных за количество операций порядка $O(|V|^2)$. Все веса неотрицательны.

Алгоритм Дейкстры для разреженных графов

- Делает то же самое, что и алгоритм Дейкстры, но за количество операций порядка $|E| \cdot log(|V|)$. Следует заметить, что m может быть порядка $|V|^2$, то есть эта вариация алгоритма Дейкстры не всегда быстрее классической, а только при маленьких $|E|$

Алгоритм поиска маршрута A*

Алгоритм **A*** (англ. *A star*) — алгоритм поиска, который находит во взвешенном графе маршрут наименьшей стоимости от начальной вершины до выбранной конечной.

$f(v) = g(v) + h(v)$

$g(v)$ - наименьшая стоимость пути в v из стартовой вершины

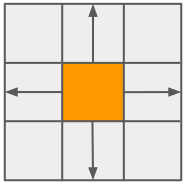
$h(v)$ - эвристическое приближение стоимости пути от v до конечной цели.

Чем меньше $f(v)$ тем минимальнее маршрут из до интересующей нас вершины (goal) из стартовой.

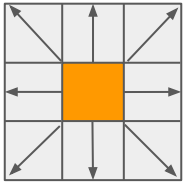
Алгоритм поиска маршрута A*

Примеры эвристик:

- Манхэттенское расстояние: $h(v) = |v.x - goal.x| + |v.y - goal.y|$



- Расстояние Чебышева: $h(v) = max(|v.x - goal.x| , |v.y - goal.y|)$



- Евклидово расстояние (не ограничено сеткой):

  $h(v) = \sqrt{(v.x - goal.x)^2 + (v.y - goal.y)^2}$

Алгоритм поиска маршрута A*

Q - множество вершин, которые нужно рассмотреть

U - множество закрытых вершин

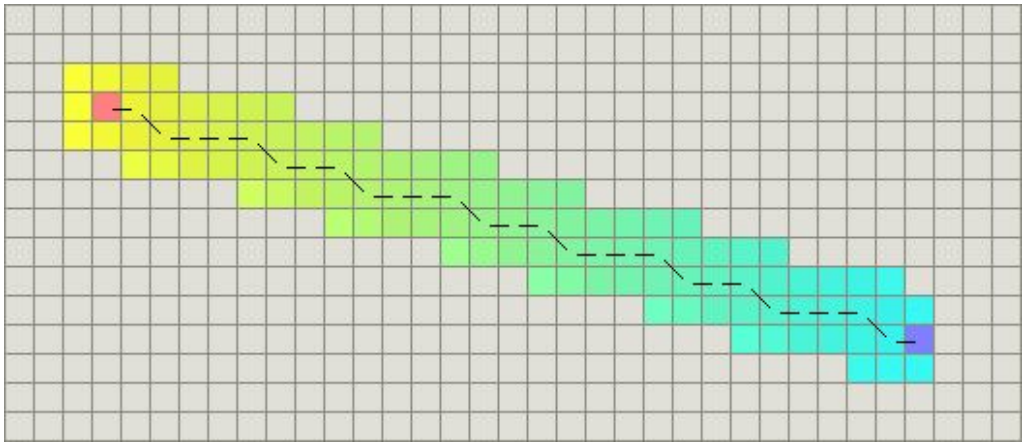
f(v) - эвристическая функция

g(v) - стоимость пути от начальной вершины до v

h(v) - эвристическая оценка расстояния от вершины v до целевой вершины (goal)

Алгоритм поиска маршрута A*

На каждой итерации алгоритма из множества Q выбирается вершина с наименьшим значением f(v) и просматриваются ее соседи. Для каждого соседа обновляются расстояния, значение эвристической функции и он добавляется в множество Q.



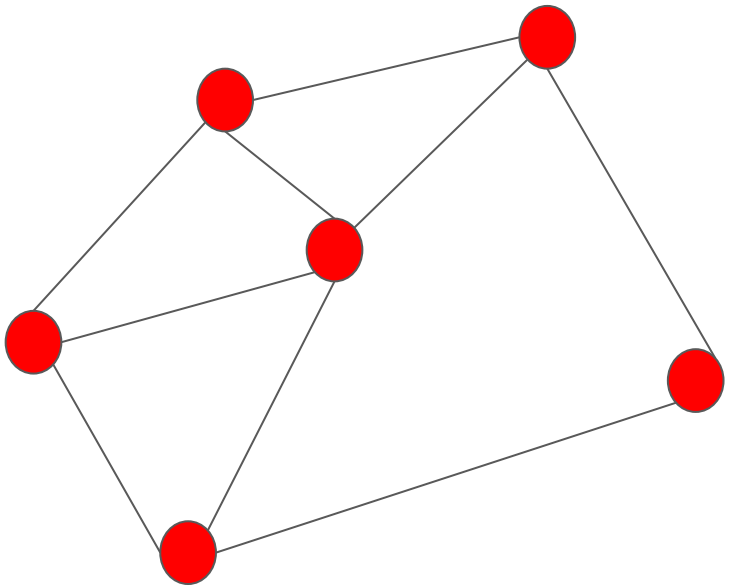
Алгоритм поиска маршрута A*

```
bool A*(start, goal):
    U = ∅
    Q = ∅
    Q.push(start)
    g[start] = 0
    f[start] = g[start] + h(start)
    while Q.size() != 0
        current = вершина из Q с минимальным значением f
        if current == goal
            return true                          // нашли путь до нужной вершины
        Q.remove(current)
        U.push(current)
        for v : смежные с current вершины
            tentativeScore = g[current] + d(current, v)
            if v∈U and tentativeScore ≥ g[v]
                continue
            if v ∉ U or tentativeScore < g[v]
                g[v] = tentativeScore
                f[v] = g[v] + h(v)
                if v ∉ Q
                    Q.push(v)
    return false
```

Волновой алгоритм. Поиск пути в невзвешенном графе.

Требуется найти путь между вершинами s и t графа (s не совпадает с t), содержащий минимальное количество промежуточных вершин (ребер). Прекрасно подойдет, если все пути из вершины в соседнюю равны по длине (цене, весу) Время O(n).

Волновой алгоритм. Поиск пути в невзвешенном графе.



Волновой алгоритм. Поиск пути в невзвешенном графе.

```
T = [|V|] #ищем путь из вершины s --> t
for v in V:
  T[v] = -1
T[s] = 0
OldFront = [s], NewFront = []
time = 0
while True
  for u in OldFront
    for v : все смежные вершины вершины u
      if T[v] == -1
        T[v] = time + 1
        NewFront.push(v)
  if NewFront is empty
    return False # нет решений
  if t in NewFront
    T[t] = time + 1 # наименьшее количество вершин до достижения вершины t
    return True
  OldFront += NewFront
  NewFront = []
  time += 1
```

Указания для задач

Для замера работы функции нужно использовать метод now() класса datetime модуля datetime

Указания для задач

```
array = [0] * N
array.insert(N,0)
```

```
import datetime


def main():
    t1 = datetime.datetime.now()
    #You'r code here
    ...
    print(datetime.datetime.now() - t1)

if __name__ == '__main__':
    main()
```

Указания для задач

```
import numpy as np


...
# Generate numpy Array with N random numbers
array = np.random.rand(N)
#Sort Array by quick sort
sorted(array)
...
```

Указания для задач

jupyter MyTests Last Checkpoint: a minute ago (unsaved changes)

File Edit View Insert Cell Kernel Help Python 2

Code CellToolbar

In [6]:
```
import datetime
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
```

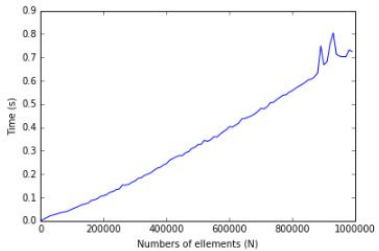
In [7]:
```
times = []
for N in xrange(1000, 1000000, 10000):
    array = np.random.rand(N)
    t1 = datetime.datetime.now()
    sorted(array)
    t_res = datetime.datetime.now() - t1
    times.append([N,t_res])
y = [t[1].total_seconds() for t in times]
x = [t[0] for t in times]
```

In [8]:
```
%matplotlib inline
```

In [9]:
```
plt.plot(x, y)

plt.xlabel('Numbers of ellements (N)')
plt.ylabel('Time (s)')

plt.savefig("test.png")
plt.show()
```



Задачи

- Реализовать алгоритм перемножения квадратных матриц. Матрицы могут задаваться как список списков. Считывать можно из файла потока ввода, или задавать случайным образом (используя функцию np.random.rand(N)). Оценить временную и ассимптотическую сложность алгоритма, построить график.
- Найти все пифагоровы тройки ($c^2 = a^2 + b^2$) для заданного интервала. Интервал задается парой чисел через пробел считанных из входного потока (например: 10 100) помните, что верхняя грань отрезка должна быть больше нижней. Если задано одно число, то считаем, что ограничение снизу равно по умолчанию 1. Оценить временную и ассимптотическую сложность алгоритма, построить график.
- Реализовать алгоритм факторизации числа (разложение числа как произведение двух других чисел). Оценить временную и ассимптотическую сложность алгоритма, построить график.
- Реализовать алгоритм рассчитывающий сочетания и размещения.
- Факториал довольно емкостная функция, при расчете которого для больших значений может случится переполнение (т.е. полученное число будет больше чем максимально возможное число в вашей системе). Подумайте как преодолеть эту проблему.

Задачи

Написать оболочку для работы с графами:

- создавать графы
- Выводить граф (в виде таблицы смежности)
- Удалять ребра
- Ищет путь в графе для заданных вершин
  - Флойда-Уоршела
  - Форда-Беллмана
  - Дейкстра

Задачи

1. Скачать файл https://goo.gl/z7H7DU
2. Файл содержит карту препятствия обозначены символом ‘%’ клетки, по которым можно передвигаться обозначены ‘-’, при этом каждая клетка по которой можно двигаться имеет вес 1.
3. Робот начинает движение в клетке обозначенной буквой ‘P’ и движется в клетку обозначенной буквой ‘T’.
4. Нужно рассчитать оптимальную траекторию пути робота с помощью алгоритма A*.
5. Выведите траекторию в отдельный файл.

Спасибо за внимание!