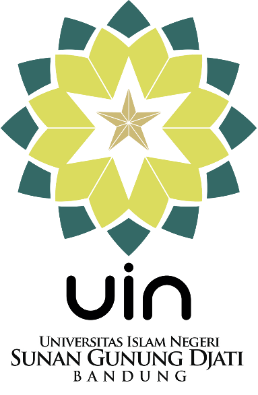
DOKUMENTASI

BEST FIRST SEARCH

Diajukan untuk memenuhi salah satu tugas mata kuliah

**Dosen Pengampu :**

**Aldy Rialdy Atmaja, MT.**

****

**Disusun Oleh:**

**Putri Mariah Ulfa (1167050123)**

**Zafiratul Amalia (1167050170)**

**TEKNIK INFORMATIKA**

**FAKULTAS SAINS DAN TEKNOLOGI**

**UNIVERSITAS ISLAM NEGERI SUNAN GUNUNG DJATI**

**BANDUNG**

**2018**

**DAFTAR ISI**

HALAMAN JUDUL…………………………………………………………..           i

KATA PENGANTAR…………………………………………………………          ii

DAFTAR ISI……………………………………………………......................          iii

BAB I PENDAHULUAN

            1.1 Latar Belakang……………………………………….....................         1

            1.2 Rumusan Masalah…………………………………………………         2

            1.3 Tujuan……………………………………………………………..         2

BAB II  PEMBAHASAN

            2.1 Metode Pencarian *Best First Search*(BFS)…………....................         3

            2.2 Algoritma *Best First Search* (BFS)…………………....................         4

 BAB II I PENUTUP

            3.1 Kesimpulan……………………………………………………....          8

            3.2 Saran…………………………………………………..................          8

DAFTAR PUSTAKA

**BAB I**

**PENDAHULUAN**

**1.1  Latar Belakang**

       Kemajuan teknologi tidak hanya menuntut kecepatan penyebaran informasi tetapi juga dalam bidang ilmu *Artificial Intelligence* untuk melakukan metode pencarian dan pelacakan yang merupakan suatu hal penting dalam suatu sistem. Karena pencarian dan pelacakan ini adalah hal yang menentukan keberhasilan sistem tersebut. sebuah algoritma pencarian dijelaskan secara luas adalah sebuah algoritma yang menerima masukan berupa sebuah masalah dan menghasilkan sebuah solusi untuk masalah tersebut, yang biasanya didapat dari evaluasi beberapa kemungkinan solusi. Sebagian besar algoritma yang dipelajari oleh ilmuwan komputer adalah algoritma pencarian. Pada dasarnya, metode pencarian dan pelacakan dibagi dua, yaitu pencarian buta (*blind search*) dan pencarian tersusun (*heuristic search)*.

Blind Searchgadalah model pencarian buta atau  pencarian yang tidak memiliki informasi awal *Blind Searching* sendiri dibagi menjadi tiga macam yaitu : *Breadth First Search, Depth first Search, Uniform Cost Search.*

Heuristic Search merupakan metode pencarian yang memperhatikan nilai heuristik (nilai perkiraan). Teknik pencarian heuristik (*heuristic searching*) merupakan suatu strategi untuk melakukan proses pencarian ruang keadaan (*state space*) suatu problema secara selektif, yang memandu  proses pencarian yang kita lakukan di sepanjang jalur yang memiliki kemungkinan sukses paling besar, dan mengesampingkan usaha yang bodoh dan memboroskan waktu. Heuristik  adalah sebuah teknik yang mengembangkan efisiensi dalam proses pencarian, namun dengan kemungkinan mengorbankan kelengkapan (*completeness*)*.*Heuristic Search  memperkirakan jarak menuju Goal (yang disebut dengan fungsi heuristik). Fungsi heuristik ini digunakan untuk mengevaluasi keadaan-keadaan problema individual dan  menentukan seberapa jauh hal tersebut dapat digunakan untuk mendapatkan solusi yang diinginkan. Jenis-jenis dari *Heuristic Searching* : *Generate and Test, Hill Climbing, Best First Search, Alpha Beta  Prunning*, *Means-End-Anlysis*, *Constraint Satisfaction.*

**1.2  Perumusan Masalah**

Berdasarkan latar belakang diatas, maka ada beberapa masalah yang akan di bahas dalam makalah ini, yaitu :

1.      Metode pencarian *Heuristic*

2.      Metode pencarian *Best First Search*

3.      Algoritma *Best First Search*

**1.3  Tujuan**

Adapun tujuan dari penulisan makalah ini, yaitu:

4.      Memahami pengertian Metode pencarian *Heuristic*

1.      Memahami metode pencarian *Best First Search*

2.      Memahami Algoritma *Best First Search*

**BAB II**

**PEMBAHASAN**

**2.1 Metode Pencarian Heuristic**

Heuristic Search merupakan metode pencarian yang memperhatikan nilai heuristic (nilai perkiraan). Teknik pencarian heuristic (*heuristic searching*) merupakan suatu strategi untuk melakukan proses pencarian ruang keadaan (*state space*) suatu problema secara selektif, yang memandu proses pencarian yang kita lakukan di sepanjang jalur yang memiliki kemungkinan sukses paling besar, dan mengesampingkan usaha yang bodoh dan memboroskan waktu. Heuristik adalah sebuah teknik yang mengembangkan efisiensi dalam proses pencarian, namun dengan kemungkinan mengorbankan kelengkapan (*completeness*)

      Heuristic Search memperkirakan jarak menuju Goal (yang disebut dengan fungsi heuristik). Fungsi heuristik ini digunakan untuk mengevaluasi keadaan-keadaan problema individual dan menentukan seberapa jauh hal tersebut dapat digunakan untuk mendapatkan solusi yang diinginkan.

**2.2 Metode Pencarian *Best First Search***

|  |  |
| --- | --- |
| |  | | --- | | **f’ = g + h’** | |

*Best First Search*(BFS) merupakan suatu cara yang menggabungkan keuntungan atau kelebihan dari pencarian *Breadth First Search* dan *Depth First Search*. Pada setiap langkah proses pencarian terbaik pertama, kita memilih node-node dengan menerapkan fungsi heuristik yang memadai pada setiap node/simpul yang kita pilih dengan menggunakan aturan-aturan tertentu untuk menghasilkan penggantinya. Fungsi Heuristik yang digunakan merupakan prakiraan (estimasi) *cost* dari *initial state* ke *goal state*, yang dinyatakan dengan :

Dimana  f’ =  prakiraan cost dari initial ke goal                                                            g  = cost dari initial state ke current state  
h’ = prakiraan cost dari *current state* ke *goal state*

*Best first search* (BFS)juga merupakan sebuah metode yang membangkit kan simpul dari simpul sebelumnya. *Best first search* memilih simpul baru yang memiliki biaya terkecil diantara  semua *leaf nodes* (simpul-simpul pada level terdalam) yang pernah dibangkitkan. Penentuan simpul terbaik dilakukan dengan menggunakan sebuah fungsi yang disebut fungsi evaluasi *f(n).* fungsi evaluasi best-first search dapat berupa biaya perkiraan dari suatu simpul menuju ke goal atau gabungan antara biaya sebenarnya dan biaya perkiraan tersebut.

 Pada setiap langkah proses pencarian terbaik pertama, kita memilih node-node dengan menerapkan fungsi heuristik yang memadai pada setiap node/simpul yang kita pilih dengan menggunakan aturan-aturan tertentu untuk menghasilkan penggantinya. Fungsi heuristic merupakan suatu strategi untuk melakukan proses pencarian ruang keadaan suatu problema secara selektif, yang memandu proses pencarian yang kita lakukan sepanjang jalur yang memiliki kemungkinan sukses paling besar.

Ada beberapa istilah yang sering digunakan pada metode *best first search,* yaitu:

1.      Start node adalah sebuah terminology untuk posisi awal sebuah pencarian

2.      Curret node adalah simpul yang sedang dijalankan dalam algoritma pencarian jalan terpendek

3.      Suksesor adalah simpul-simpul yang yang akan diperiksa setelah current node

4.      Simpul (node) merupakan representasi dari area pencarian

5.      Open list adalah tempat menyimpan data simpul yang mungkin diakses dari starting node maupun simpul yang sedang dijalankan

6.      Closed list adalah tempat menyimpan data simpul yang juga merupakan bagian dari jalur terpendek yang telah berhasil didapatkan

7.      Goal node yaitu simpul tujuan

8.      Parent adalah curret node dari suksesor.

**2.3 Algoritma *Best First Search***

      Algoritma best first search ini merupakan kombinasi dari algoritma depth first search  dengan algoritma breadth first search  dengan mengambil kelebihan dari kedua  algoritma tersebut. Apabila pada pencarian dengan algoritma  hill climbing tidak  diperbolehkan untuk kembali ke node pada level yang lebih rendah meskipun node di  level yang lebih rendah tersebut memiliki nilai heuristik yang lebih baik, lain halnya pada algoritma best first search, pencarian diperbolehkan mengunjungi node yang ada di level yang lebih rendah, jika ternyata node di level yang lebih tinggi memiliki nilai heuristik yang lebih buruk.

       Algoritma best first search merupakan salah satu bagian dari tipe informed search. Algoritma ini menggunakan nilai-nilai heuristik tiap simpul yang dibuka. Simpul dengan nilai heuristik terbaik akan dibuka lebih dahulu. Bila goal state belum ditemukan, akan dilakukan pemeriksaan pada simpul berikutnya dengan nilai heuristik terbaik pada kedalaman yang sama. Simpul tersebut kemudian dibuka dan diperiksa apakah terdapat goal state pada cabang-cabangnya. Bila goal state belum ditemukan,  akan dilakukan proses yang sama pada simpul berikutnya.

       Merupakan metode yang membangkitkan suksesor dengan mempertim- bangkan harga (didapat dari fungsi heuristik tertentu) dari setiap node, bukan dari aturan baku seperti DFS maupun BFS. Gambar 3.4 mengilustrasikan langkah-langkah yang dilakukan oleh algoritma *Best First Search*. Pertama kali, dibangkitkan node A. Kemudian semua suksesor A dibangkitan, dan dicari harga paling minimal. Pada langkah 2, node D terpilih karena harganya paling rendah, yakni 1. Langkah 3, semua suksesor D dibangkitkan, kemudian harganya akan dibandingkan dengan harga node B dan C. Ternyata harga node B paling kecil dibandingkan harga node C, E, dan F. Sehingga B terpilih dan selanjutnya akan dibangkitkan semua suksesor B. Demikian seterusnya sampai ditemukan node Tujuan.

**Gambar 2.1**Langkah-langkah yang dilakukan oleh algoritma *Best First Search*.

Untuk mengimplementasikan algoritma pencarian ini, diperlukan dua buah senarai, yaitu: OPEN untuk mengelola node-node yang pernah dibangkitkan tetapi belum dievaluasi dan CLOSE untuk mengelola node-node yang pernah dibangkitkan dan sudah dievaluasi. Algoritma selengkapnya adalah sebagai berikut.

1.      OPEN berisi initial state dan CLOSED masih kosong.

2.      Ulangi sampai goal ditemukan atau sampai tidak ada di dalam OPEN.  
              a.  Ambil simpul terbaik yang ada di OPEN.  
              b.   Jika simpul tersebut sama dengan goal, maka sukses  
              c.   Jika tidak, masukkan simpul tersebut ke dalam CLOSED  
              d.  Bangkitkan semua aksesor dari simpul tersebut  
              e.   Untuk setiap suksesor kerjakan.

Algoritma yang menggunakan metode best-first search, yaitu:

**a.      Greedy Best-First**

      Greedy Best-First adalah algoritma best-first yang paling sederhana dengan hanya memperhitungkan biaya perkiraan (estimated cost) saja, yakni *f(n) = h(n).* Biaya yang sebenarnya (actual cost) tidak diperhitungkan. Dengan hanya memperhitungkan biaya perkiraan yang belum tentu kebenarannya, maka algoritma ini menjadi tidak optimal.

**b.      A\***

      Algoritma A\* merupakan algoritma best first search dengan modifikasian fungsi heuristik, yang akan meminimumkan total biaya lintasan, dan pada kondisi yang tepat  akan memberikan solusi yang terbaik dalam waktu yang optimal.

      Algoritma A juga membutuhkan dua antrian, yaitu OPEN dan CLOSED. Selain itu, ada juga fungsi heuristik yang memprediksi keuntungan tiap node yang di buat. Yang akan memungkinkan algoritma untuk melakukan pencarian-pencarian lintasan yang lebih di harapkan. Fungsi ini di sebut f’(n) sebagai pendekatan dari fungsi f(n) yang merupakan fungsi evaluasi yang sebenarnya terhadap node n. dalam banyak penarapan, akan lebih baik jika fungsi di definisikan sebagai kombinasi atau jumlah dua komponen yaitu g(n) dan h(n). Fungsi g(n) merupakan ukuran biaya yang di keluarkan dari keadaan awal sampai ke node n. Nilai yang didapat g(n) merupakan jumlahan biaya penerapan setiap aturan yang dilakukan pada sepanjang lintasan trbaik menuju suatu simpul dan bukan merupakan hasil estimasi.

      Fungsi h(n) merupakan pengukur biaya tambahan yang harus dikeluarkan dari node n sampai mendapatkan tujuan. Perlu diketahui bahwa g(n), tidak negatif karena bila negatif maka lintasan yang membalik siklus pada graf akan tampak lebih baik dengan semakin panjangnya lintasan.

Secara matematis,fungsi F sebagai estimasi fungsi evaluasi terhadap  node ndapat di tuliskan :

                                                              f’(n) = g(n) + h’(n)

Dengan   f’(n) = fungsi evaluasi

                            g(n) = biaya yang sudah di keluarkan dari keadaan awal sampai

                                       keadaan n

                     h’(n) = estimasi biaya untuk sampai pada suatu tujuan mulai dari n

                  dari fungsi di atas maka ada beberapa kondisi yang perlu diperhatikan, yaitu:

                     Jika h = h’, berarti proses pencarian telah sampai ke tujuan ( goal ).

                     Jika g = h’ = 0 maka f’ random, artinya system tidak dapat di kendalikan.

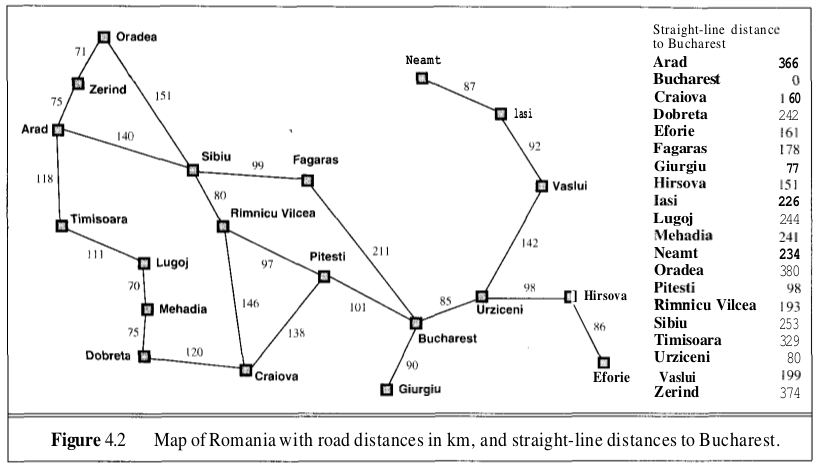
                     Jika g = k, k adalah konstanta dan biasanya bernilai 1, h’ = 0, artinya system

                     menggunakan breadth first search.

**Greedy BFS Algorithm**

Heuristic functions are clearly problem-specific.

Let us understand this better through an example.

[](https://user-images.githubusercontent.com/22571531/27821959-ac69755a-60c1-11e7-8286-951cd1c0437f.png)

Source: Artificial Intelligence A Modern Approach by Stuart J. Russell and Peter Norvig

Given here is the map of Romania with cities and distance between them. We need to find the shortest route from Arad to Bucharest.

The heurestics that we are using here is the straight-line distance from the city to the goal(Here, Bucharest). Note that, this straight line distance is obtained only by knowing the map coordinates of the 2 cities.

**Input**

Input is taken from the file

input.txt

Each line in the input is of the form

city1 city2 dist

It denotes each element of the adjacency list. That is, dist is the distance between city1 and city2. An undireced graph is drawn depicting the map of Romania. Starting city: Arad Goal city: Bucharest

Heuristics is loaded from the file

heuristics.txt

Each line is of the form

city h

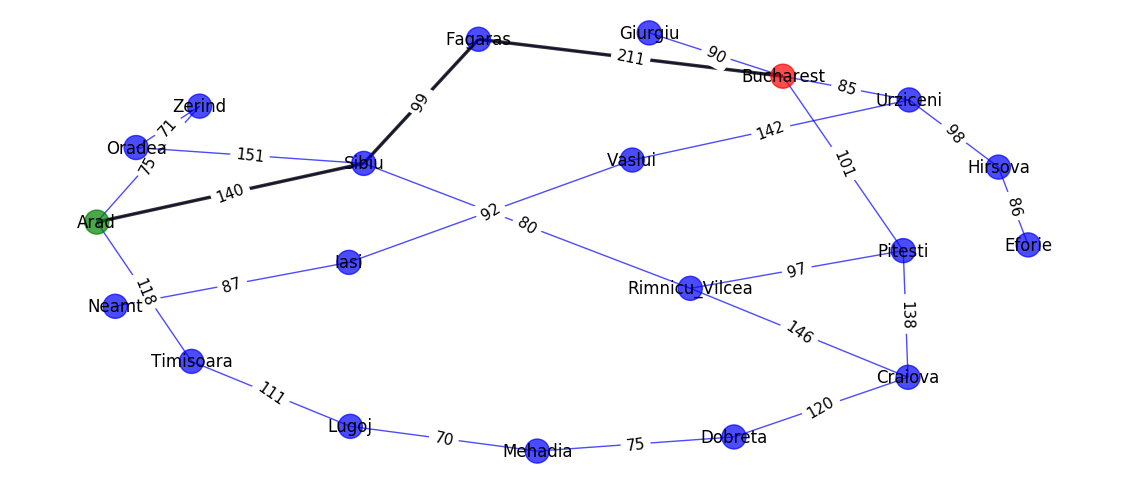
'h' stands for the heuristics value(here, the straight line distnace) from the city 'city' to goal city(here, Bucharest)

**Working:**

Starting from source city, we choose the next city which is the closest to the Goal city amongst all it's neighbours(based on the heuristics function). We terminate, once we've reached the goal city.

Here, Arad is the starting city. The first node to be expanded from Arad will be Sibiu, because it is closer to Bucharest than either Zerind or Timisoara. The next node to be expanded will be Fagaras, because it is closest. Fagaras in turn generates Bucharest, which is the goal.

Here, is the resultant graph.  
Green coloured node denotes the starting city(Here, Arad).  
Red coloured node denotes the goal city(Here, Bucharest).  
Edges marked with black is the route from Arad to Bucharest generated by the greedy bfs algorithm.

[](https://user-images.githubusercontent.com/22571531/27821965-b12030d4-60c1-11e7-9e37-9388e4dd8751.png)

**Complexity**

Time: O(b^m)  
b - Branching factor, the average number of successors per state  
m - maximum depth of the search

|  |
| --- |
|  |
| import networkx as nx |
|  | import matplotlib.pyplot as plt |
|  | import Queue as Q |
|  |  |
|  |  |
|  |  |
|  | def getPriorityQueue(list): |
|  | q = Q.PriorityQueue() |
|  | for node in list: |
|  | q.put(Ordered\_Node(heuristics[node],node)) |
|  | return q,len(list) |
|  |  |
|  |  |
|  |  |
|  | def greedyBFSUtil(G, v, visited, final\_path, dest, goal): |
|  | if goal == 1: |
|  | return goal |
|  | visited[v] = True |
|  | final\_path.append(v) |
|  | if v == dest: |
|  | goal = 1 |
|  | else: |
|  | pq\_list = [] |
|  | pq,size = getPriorityQueue(G[v]) |
|  | for i in range(size): |
|  | pq\_list.append(pq.get().description) |
|  | for i in pq\_list: |
|  | if goal != 1: |
|  | #print "current city:", i |
|  | if visited[i] == False : |
|  | goal = greedyBFSUtil(G, i, visited, final\_path, dest, goal) |
|  | return goal |
|  |  |
|  |  |
|  |  |
|  | def greedyBFS(G, source, dest, heuristics, pos): |
|  | visited = {} |
|  | for node in G.nodes(): |
|  | visited[node] = False |
|  | final\_path = [] |
|  | goal = greedyBFSUtil(G, source, visited, final\_path, dest, 0) |
|  | prev = -1 |
|  | for var in final\_path: |
|  | if prev != -1: |
|  | curr = var |
|  | nx.draw\_networkx\_edges(G, pos, edgelist = [(prev,curr)], width = 2.5, alpha = 0.8, edge\_color = 'black') |
|  | prev = curr |
|  | else: |
|  | prev = var |
|  | Return |
|  |  |
|  |  |
|  |  |
|  | class Ordered\_Node(object): |
|  | def \_\_init\_\_(self, priority, description): |
|  | self.priority = priority |
|  | self.description = description |
|  | Return |
|  | def \_\_cmp\_\_(self, other): |
|  | return cmp(self.priority, other.priority) |
|  |  |
|  | def getHeuristics(G): |
|  | heuristics = {} |
|  | f = open('heuristics.txt') |
|  | for i in G.nodes(): |
|  | node\_heuristic\_val = f.readline().split() |
|  | heuristics[node\_heuristic\_val[0]] = node\_heuristic\_val[1] |
|  | return heuristics |
|  |  |
|  |  |
|  |  |
|  | #takes input from the file and creates a weighted graph |
|  | def CreateGraph(): |
|  | G = nx.Graph() |
|  | f = open('input.txt') |
|  | n = int(f.readline()) |
|  | for i in range(n): |
|  | graph\_edge\_list = f.readline().split() |
|  | G.add\_edge(graph\_edge\_list[0], graph\_edge\_list[1], length = graph\_edge\_list[2]) |
|  | source, dest= f.read().splitlines() |
|  | return G, source, dest |
|  |  |
|  |  |
|  |  |
|  | def DrawPath(G, source, dest): |
|  | pos = nx.spring\_layout(G) |
|  | val\_map = {} |
|  | val\_map[source] = 'green' |
|  | val\_map[dest] = 'red' |
|  | values = [val\_map.get(node, 'blue') for node in G.nodes()] |
|  | nx.draw(G, pos, with\_labels = True, node\_color = values, edge\_color = 'b' ,width = 1, alpha = 0.7) #with\_labels=true is to show the node number in the output graph |
|  | edge\_labels = dict([((u, v,), d['length']) for u, v, d in G.edges(data = True)]) |
|  | nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels = edge\_labels, label\_pos = 0.5, font\_size = 11) #prints weight on all the edges |
|  | return pos |
|  |  |
|  |  |
|  |  |
|  | #main function |
|  | if \_\_name\_\_ == "\_\_main\_\_": |
|  | G, source,dest = CreateGraph() |
|  | heuristics = getHeuristics(G) |
|  | pos = DrawPath(G, source, dest) |
|  | greedyBFS(G, source, dest, heuristics, pos) |
|  | plt.show() |
|  |  |

**BAB III**

**PENUTUP**

**3.1 Kesimpulan**

Berdasarlan uraian pembahasan pada bab sebelumnya dapat ditarik kesimulan

      Sebagai berikut :

1.      Metode pencarian heuristik (*heuristic searching*) merupakan suatu strategi untuk melakukan proses pencarian ruang keadaan (*state space*) suatu problema secara selektif.

2.      Best *First Search* merupakan sebuah metode yang membangkitkan simpul dari simpul sebelumnya. Best first search memilih simpul baru yang memiliki biaya terkecil diantara  semua *leaf nodes* (simpul-simpul pada level terdalam) yang pernah dibangkitkan.

3.      Algoritma *best first search* ini merupakan kombinasi dari algoritma *depth first search* dengan algoritma *breadth first search*  dengan mengambil kelebihan dari kedua  algoritma tersebut.

**3.2 Saran**

     dalam bidang Artificial Intelligence dikenal metode pencarian dan pelacakan. dan yang dibahas dalam makalah ini yaitu metode pencarian best first search yang merupakan bagian dari pencarian heuristic

SOURCE CODE :

|  |
| --- |
|  |
| import networkx as nx |
|  | import matplotlib.pyplot as plt |
|  | import Queue as Q |
|  |  |
|  |  |
|  |  |
|  | def getPriorityQueue(list): |
|  | q = Q.PriorityQueue() |
|  | for node in list: |
|  | q.put(Ordered\_Node(heuristics[node],node)) |
|  | return q,len(list) |
|  |  |
|  |  |
|  |  |
|  | def greedyBFSUtil(G, v, visited, final\_path, dest, goal): |
|  | if goal == 1: |
|  | return goal |
|  | visited[v] = True |
|  | final\_path.append(v) |
|  | if v == dest: |
|  | goal = 1 |
|  | else: |
|  | pq\_list = [] |
|  | pq,size = getPriorityQueue(G[v]) |
|  | for i in range(size): |
|  | pq\_list.append(pq.get().description) |
|  | for i in pq\_list: |
|  | if goal != 1: |
|  | #print "current city:", i |
|  | if visited[i] == False : |
|  | goal = greedyBFSUtil(G, i, visited, final\_path, dest, goal) |
|  | return goal |
|  |  |
|  |  |
|  |  |
|  | def greedyBFS(G, source, dest, heuristics, pos): |
|  | visited = {} |
|  | for node in G.nodes(): |
|  | visited[node] = False |
|  | final\_path = [] |
|  | goal = greedyBFSUtil(G, source, visited, final\_path, dest, 0) |
|  | prev = -1 |
|  | for var in final\_path: |
|  | if prev != -1: |
|  | curr = var |
|  | nx.draw\_networkx\_edges(G, pos, edgelist = [(prev,curr)], width = 2.5, alpha = 0.8, edge\_color = 'black') |
|  | prev = curr |
|  | else: |
|  | prev = var |
|  | Return |
|  |  |
|  |  |
|  |  |
|  | class Ordered\_Node(object): |
|  | def \_\_init\_\_(self, priority, description): |
|  | self.priority = priority |
|  | self.description = description |
|  | Return |
|  | def \_\_cmp\_\_(self, other): |
|  | return cmp(self.priority, other.priority) |
|  |  |
|  | def getHeuristics(G): |
|  | heuristics = {} |
|  | f = open('heuristics.txt') |
|  | for i in G.nodes(): |
|  | node\_heuristic\_val = f.readline().split() |
|  | heuristics[node\_heuristic\_val[0]] = node\_heuristic\_val[1] |
|  | return heuristics |
|  |  |
|  |  |
|  |  |
|  | #takes input from the file and creates a weighted graph |
|  | def CreateGraph(): |
|  | G = nx.Graph() |
|  | f = open('input.txt') |
|  | n = int(f.readline()) |
|  | for i in range(n): |
|  | graph\_edge\_list = f.readline().split() |
|  | G.add\_edge(graph\_edge\_list[0], graph\_edge\_list[1], length = graph\_edge\_list[2]) |
|  | source, dest= f.read().splitlines() |
|  | return G, source, dest |
|  |  |
|  |  |
|  |  |
|  | def DrawPath(G, source, dest): |
|  | pos = nx.spring\_layout(G) |
|  | val\_map = {} |
|  | val\_map[source] = 'green' |
|  | val\_map[dest] = 'red' |
|  | values = [val\_map.get(node, 'blue') for node in G.nodes()] |
|  | nx.draw(G, pos, with\_labels = True, node\_color = values, edge\_color = 'b' ,width = 1, alpha = 0.7) #with\_labels=true is to show the node number in the output graph |
|  | edge\_labels = dict([((u, v,), d['length']) for u, v, d in G.edges(data = True)]) |
|  | nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels = edge\_labels, label\_pos = 0.5, font\_size = 11) #prints weight on all the edges |
|  | return pos |
|  |  |
|  |  |
|  |  |
|  | #main function |
|  | if \_\_name\_\_ == "\_\_main\_\_": |
|  | G, source,dest = CreateGraph() |
|  | heuristics = getHeuristics(G) |
|  | pos = DrawPath(G, source, dest) |
|  | greedyBFS(G, source, dest, heuristics, pos) |
|  | plt.show() |
|  |  |

**DAFTAR PUSTAKA**

<http://blognyacomel.blogspot.com/2013/11/best-first-search.html>

          diakses 9 Oktober 2018 pukul 23.23

<http://buatugasai.blogspot.com/2013/04/metode-pencarian-dan-pelacakan_4.html>

          diakses 10 Oktober 2018 pukul 01.14

[http://najibzot.blogspot.com/p/teknik-searching-kecerdasan-buatan-di.html](view-source:http://najibzot.blogspot.com/p/teknik-searching-kecerdasan-buatan-di.html)

          diakses 10 Oktober 2018 pukul 01.32