LAPORAN PRAKTIKUM
KECERDASAN BUATAN
“Heuristic Searching”

Dibuat untuk memenuhi tugas yang diampu oleh:

Fitri Nuraeni, S. Kom., M.Kom.

Disusun Oleh:

Endang Prayoga Hidayatulloh

NIM. 2006189

PROGRAM STUDI TEKNIK INFORMATIKA

INSTITUT TEKNOLOGI GARUT

2022

KATA PENGANTAR

Puji syukur saya panjatkan ke hadirat Allah SWT atas segala rahmat dan karunia-Nya yang telah diberikan, sehingga penyusun bisa menyelesaikan Laporan Praktikum Kecerdasan Buatan ini yang berjudul “Heuristic Searching” Adapun tujuan disusunnya laporan ini adalah sebagai syarat untuk memenuhi tugas mata kuliah Praktikum Kecerdasan Buatan.

Tersusunnya laporan ini tentu bukan karena buah kerja keras saya semata, melainkan juga atas bantuan dari berbagai pihak. Untuk itu, saya ucapkan terima kasih sebesar-besarnya kepada semua pihak yang membantu terselesaikannya laporan ini.

Saya sangat menyadari bahwa laporan ini masihlah jauh dari sempurna. Untuk itu, Saya selaku penyusun menerima dengan terbuka semua kritik dan saran yang membangun agar laporan ini bisa tersusun lebih baik lagi. Saya berharap semoga laporan ini bermanfaat untuk kita semua.

Garut, 13 Maret 2023

Endang Prayoga Hidayatulloh

DAFTAR ISI

A. Rumusan Masalah

Adapun rumusan masalah dalam laporan ini adalah:

1. Apa aitu kecerdasan buatan?
2. Apa itu metode heuristic searching didalam kecerdasan buatan?
3. Bagaimana mengaplikasikan persoalan metode searching kedalah algoritma GBFS dan A*?

B. Dasar Teori

1. Kecerdasan Buatan

   Kecerdasan buatan (Artificial Intelligence atau AI) adalah suatu cabang ilmu komputer yang bertujuan untuk menciptakan mesin atau sistem yang memiliki kemampuan untuk melakukan tugas yang biasanya memerlukan kecerdasan manusia, seperti pengenalan suara dan gambar, pemrosesan bahasa alami, pengambilan keputusan, dan belajar. Tujuan utama dari AI adalah untuk membuat mesin yang dapat berpikir seperti manusia dan melakukan tugas-tugas dengan kemampuan intelektual yang serupa.

   AI mencakup beberapa teknologi termasuk machine learning, natural language processing, computer vision, dan robotics. Teknologi ini menggunakan algoritma dan model matematis untuk mempelajari data dan membuat prediksi atau keputusan berdasarkan data yang diproses. Sebagai contoh, sistem cerdas yang menggunakan machine learning dapat diprogram untuk mengenali objek pada gambar, sedangkan sistem cerdas yang menggunakan natural language processing dapat diprogram untuk memahami bahasa manusia.

2. Metode Heuristic Searching

   Heuristic searching atau pencarian heuristik adalah metode pencarian yang menggunakan pengetahuan atau informasi tambahan untuk mempercepat proses pencarian solusi pada masalah yang kompleks. Pengetahuan ini dapat berasal dari pengalaman atau aturan-aturan yang didefinisikan sebelumnya.

   Heuristic searching atau pencarian heuristik memiliki beberapa fungsi dan kelebihan yang dapat membantu dalam menyelesaikan masalah yang kompleks dengan lebih efisien dan cepat, di antaranya:

a) Meningkatkan efisiensi pencarian: Heuristic searching dapat membantu meningkatkan efisiensi pencarian solusi dengan menggunakan pengetahuan tambahan yang diperoleh dari pengalaman atau aturan-aturan yang didefinisikan sebelumnya.
b) Menghemat waktu dan sumber daya: Dengan menggunakan pendekatan heuristik, waktu dan sumber daya yang diperlukan untuk mencari solusi dapat dikurangi secara signifikan. Algoritma heuristik memungkinkan kita untuk mencari solusi dalam waktu yang lebih singkat daripada metode pencarian lainnya.
c) Meningkatkan akurasi solusi: Heuristic searching dapat membantu meningkatkan akurasi solusi dengan menggunakan pengetahuan tambahan yang diperoleh dari pengalaman atau aturan-aturan yang didefinisikan sebelumnya. Algoritma heuristik dapat memilih solusi yang paling dekat dengan solusi optimal.
d) Mengatasi masalah kompleks: Heuristic searching dapat membantu mengatasi masalah kompleks dengan cara mempercepat pencarian solusi melalui penggunaan pengetahuan tambahan yang diperoleh dari pengalaman atau aturan-aturan yang didefinisikan sebelumnya.
e) Serbaguna: Heuristic searching dapat digunakan pada berbagai macam masalah optimasi dan pencarian, seperti perencanaan rute, permainan video, jaringan saraf, dan optimasi fungsi.

3. Algoritma Heuristic Searching

Berikut adalah definisi beberapa jenis algoritma heuristic searching:

a) Breadth-First Search (BFS)

BFS adalah algoritma pencarian heuristik yang menggunakan pendekatan sistematis dengan mengeksplorasi semua node di tingkat yang sama sebelum melanjutkan ke level berikutnya. BFS mencari solusi dengan mempertimbangkan semua kemungkinan langkah dalam level yang sama sebelum melanjutkan ke level berikutnya. Algoritma ini cocok digunakan pada grafik dengan struktur datar.

b) Greedy Best-First Search (GBFS)

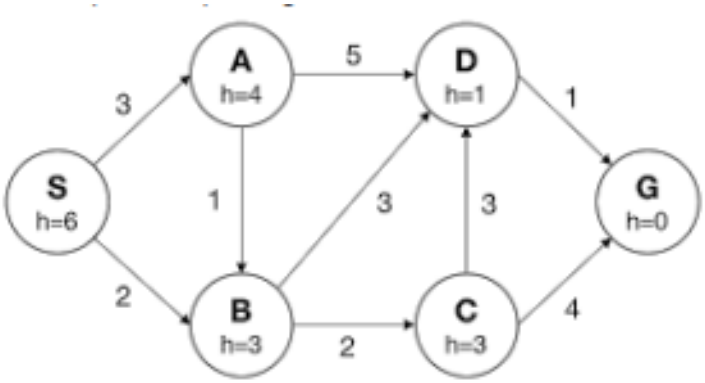
GBFS adalah algoritma pencarian heuristik yang memilih node yang paling menjanjikan untuk dieksplorasi berdasarkan nilai fungsi heuristik. Algoritma ini tidak mempertimbangkan total biaya dari awal sampai node tersebut, melainkan hanya mempertimbangkan biaya dari node saat ini ke node tujuan.

c) A* Search

A* Search adalah algoritma pencarian heuristik yang menggabungkan ide-ide dari BFS dan GBFS. A* Search menggunakan fungsi evaluasi f(n) = g(n) + h(n) untuk menentukan node yang akan dieksplorasi selanjutnya. Fungsi g(n) adalah biaya untuk mencapai node saat ini dari node awal, dan fungsi h(n) adalah estimasi biaya untuk mencapai node tujuan dari node saat ini. Algoritma ini mempertimbangkan total biaya dari awal sampai node saat ini dan estimasi biaya dari node saat ini ke node tujuan.

C. Hasil dan Pembahasan

Dalam menyelesaikan persoalan atau permasalahan untuk menyelesaikan masalah pencarian lintasan terpendek pada graf berikut.



Yang dimana titik awal dari perjalanan rute tersebut yaitu dimulai pada titik S dan berakhir pada titk G. Dalam penyelesaiannya kita dapat menggunakan algoritma GBFS dan A*. Berikut adalah Langkah dan source code dalam menyelesaikan persoalan diatas.

1. Mengimport module yang dibutuhkan, module yang dibutuhkan yaitu:
   - Queue
   - Matplotlib

```
import queue
import matplotlib.pyplot as plt
```

2. Membuat fungsi nilai heuristic untuk setiap lokasi titiknya.

```
def getHeuristics():
     heuristics = {
         'A': 3,
```

```
        'B': 3,
        'C': 3,
        'D': 1,
        'G': 0,
        'S': 6}
  return heuristics
```

3. Membuat fungsi nilai dari koordinat titik lokasi.

```
def getCity():
     city = {
         'S': [15, 30],
         'A': [35, 50],
         'B': [35, 10],
         'C': [55, 10],
         'D': [55, 50],
         'G': [75, 30]}
     citiesCode = {
         1: 'S',
         2: 'A',
         3: 'B',
         4: 'C',
         5: 'D',
         6: 'G'}

  return city, citiesCode
```

4. Membuat fungsi graph untuk menentukan relasi dan dari satu lokasi ke titik lokasi lain.

```
def createGraph():
    graph = {
        'S': [['A', '3'], ['B', '2']],
        'A': [['S', '3'], ['D', '5'], ['B', '6']],
        'B': [['S', '2'], ['D', '3'], ['A', '6'], ['C', '6']],
        'D': [['A', '5'], ['B', '3'], ['C', '6'], ['G', '1']],
        'C': [['B', '6'], ['D', '6'], ['G', '6']],
        'G': [['C', '6'], ['D', '1']]}
```

```
    return graph
print(createGraph())
```

5. Membuat fungsi untuk menentukan jarak terdekat dengan tujuan titik dengan menggunakan algoritma GBFS.

```
 def GBFS(startNode, heuristics, graph, goalNode="G"):
    priorityQueue = queue.PriorityQueue()
    path = []

    priorityQueue.put((heuristics[startNode], startNode))

    while priorityQueue.empty() == False:
        current = priorityQueue.get()[1]
        path.append(current)

        if current == goalNode:
            break

        for i in graph[current]:
            if i[0] not in path:
                priorityQueue.put((heuristics[i[0]], i[0]))

    return path
```

6. Membuat fungsi untuk menentukan jarak terdekat dengan tujuan titik dengan menggunakan algoritma A*.

```
  def Astar(startNode, heuristics, graph, goalNode="G"):
    priorityQueue = queue.PriorityQueue()
    distance = 0
    path = []

    priorityQueue.put((heuristics[startNode]   +   distance,
[startNode, 0]))

    while priorityQueue.empty() == False:
        current = priorityQueue.get()[1]
```

```
        path.append(current[0])
        distance += int(current[1])

        if current[0] == goalNode:
            break

        priorityQueue = queue.PriorityQueue()

        for i in graph[current[0]]:
            if i[0] not in path:
                priorityQueue.put((heuristics[i[0]]        +
int(i[1]) + distance, i))

    return path
```

7. Membuat fungsi untuk membuat visualisasi map dan prtunjuk arah terdekat dengan struktur data serta algoritma yang sudah dibuat sebelumnya.

```
def drawMap(city, gbfs, astar, graph):
    for i, j in city.items():
        plt.plot(j[0], j[1], "ro")
        plt.annotate(i, (j[0] + 5, j[1]))

        for k in graph[i]:
            n = city[k[0]]
            plt.plot([j[0], n[0]], [j[1], n[1]], "gray")

    for i in range(len(gbfs)):
        try:
            first = city[gbfs[i]]
            secend = city[gbfs[i + 1]]

            plt.plot([first[0],    secend[0]],    [first[1],
secend[1]], "green")
        except:
            continue
```

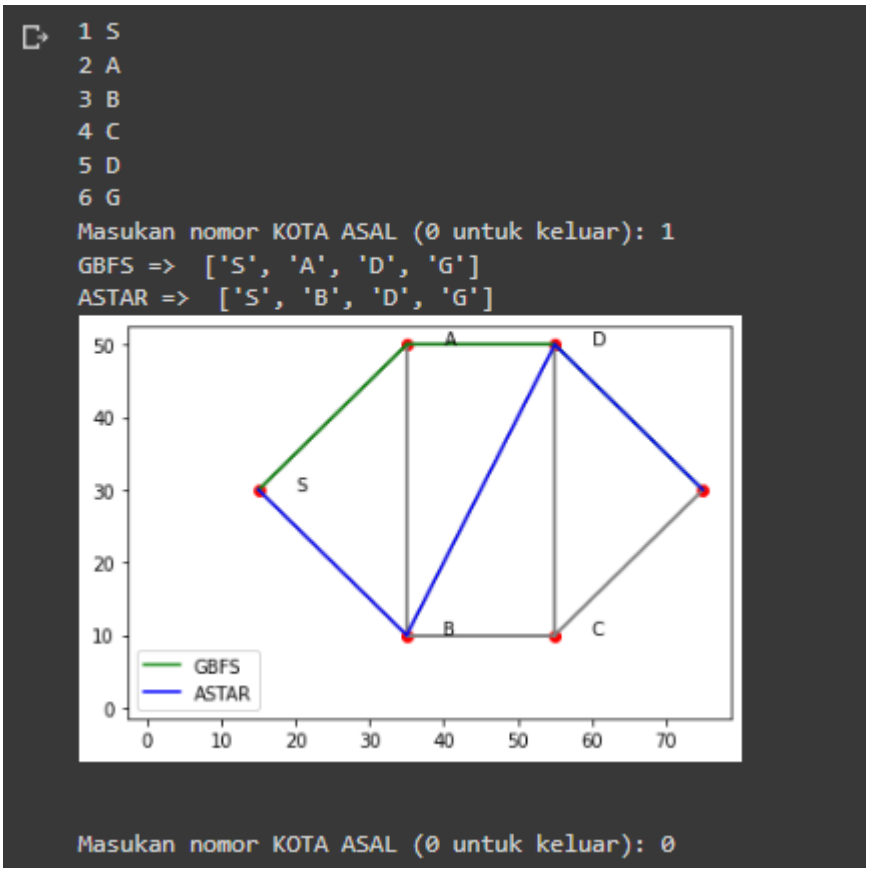
```
    for i in range(len(astar)):
        try:
            first = city[astar[i]]
            secend = city[astar[i + 1]]

            plt.plot([first[0],    secend[0]],    [first[1],
secend[1]], "blue")
        except:
            continue

    plt.errorbar(1, 1, label="GBFS", color="green")
    plt.errorbar(1, 1, label="ASTAR", color="blue")
    plt.legend(loc="lower left")

    plt.show()
```

8. Membuat fungsi utama program.

```
def main():
    heuristic = getHeuristics()
    graph = createGraph()
    city, citiesCode = getCity()

    for i, j in citiesCode.items():
        print(i, j)

    while True:
        inputCode  =  int(input("Masukan  nomor  KOTA  ASAL  (0
untuk keluar): "))

        if inputCode == 0:
            break

        cityName = citiesCode[inputCode]

        gbfs = GBFS(cityName, heuristic, graph)
        astar = Astar(cityName, heuristic, graph)
        print("GBFS => ", gbfs)
```

```
        print("ASTAR => ", astar)

        drawMap(city, gbfs, astar, graph)
        print("\n")

main()
```

Hasil output program yang menunjukan titik awal dari point S:



Hasil dari penggunaan algoritma GBFS dan A* dalam menunjukan arah terdekat dalam mencapai tujuan yaitu:

- Dengan menggunakan algoritma GBFS ditunjukan bahwa jalur terdekat dari titik S sampai ke titik G yaitu dengan alur jalur: ['S', 'A', 'D', 'G'].
- Sedangkan dengan menggunakan algoritma GBFS ditunjukan bahwa jalur terdekat dari titik S sampai ke titik G yaitu dengan alur jalur: ['S', 'B', 'D', 'G'].

D. Kesimpulan

Algoritma BFS, GBFS, dan A* Search sangat populer dan sering digunakan dalam pengembangan game, robotika, dan aplikasi lainnya yang memerlukan pencarian solusi dalam masalah yang kompleks.

DAFTAR PUSTAKA