Tugas Individu 1

KECERDASAN BUATAN

FINDING SHORTEST PATH

Zakia kolbi (5025211049)

YANG KITA BAHAS!

- Penjelasan Informed Search
- Penjelasan Metode A*
- Source Code
- Output

INFORMED SEARCH

Informed Search Algorithm merupakan algoritma pencarian menggunakan pengetahuan yang spesifik kepada permasalahan yang dihadapi selain dari definisi masalahnya itu sendiri sehingga lebih hemat waktu dan biaya. Ada 2 metode Informed Search Algorithm yang terkenal yaitu, A* Search (A-Star Search) dan Greedy Best First Search.

METODE A*

Metode A* merupakan metode pencarian rute dengan menggunakan teknik heuristik. Teknik heuristik digunakan untuk meningkatkan evisiensi waktu terhadap pencarian rute.
Rute yang dicari hanyalah dua lokasi saja, yaitu lokasi awal dan lokasi akhir.

Notasi yang dipakai oleh algoritma A* adalah sebagai berikut:

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

f(n) = Nilai Heuristic / total biaya yang dibutuhkan untuk mencapai node goal dari node awal melalui node n.
g(n) = Jumlah gerakan kotak putih
h(n) = nilai heuristik dari node n ke node goal.

SOURCE CODE

```python
import heapq

def a_star(graph, start, goal, sld):
    queue = [(0, start)]
    visited = set()

    while queue:
        (cost, current_node) = heapq.heappop(queue)
        if current_node == goal:
            return cost
        if current_node not in visited:
            visited.add(current_node)
            for nbor, distance in graph[current_node].items():
                if nbor not in visited:
                    heuristic_cost = sld[nbor]
                    heapq.heappush(queue,(cost + distance + heuristic_cost, nbor))

    return float('inf')
```

SOURCE CODE

```
graph = {
    'A': {'B': 5, 'C': 3},
    'B': {'G': 1, 'E': 3, 'C': 2},
    'C': {'E': 7, 'D': 7},
    'D': {'A': 2, 'F': 6},
    'E': {'D': 2, 'F': 1},
    'F': {},
    'G': {'E': 1}
}

sld = {
    'A': 9,
    'B': 7,
    'C': 0,
    'D': 6,
    'E': 5,
    'F': 2,
    'G': 0
}

start = 'A'
goal = 'G'
result = a_star(graph, start, goal, sld)
print(f'Jarak terpendek {start} ke {goal} = {result}.')
```

OUTPUT

```
Jarak terpendek A ke G = 13.
```

Tugas Individu 2

KECERDASAN BUATAN

GENETIC ALGORITHM

Zakia kolbi (5025211049)

YANG KITA BAHAS!

- Penjelasan TSP
- Penjelasan Algoritma Genetika
- Source Code
- Output

TRAVELING SALESMAN PROBLEM

TSP atau Traveling Salesman Problem adalah masalah optimasi kombinatorial yang kompleks dimana suatu agen harus mengunjungi beberapa kota yang berbeda dan kembali ke kota asal, dengan tujuan mengunjungi semua kota dengan jarak terpendek yang memungkinkan.

GENETIC ALGORITHM

Algoritma genetika adalah teknik pencarian optimasi yang terinspirasi dari proses evolusi dalam alam. Algoritma ini digunakan untuk memecahkan masalah optimasi dengan mencari solusi terbaik dari sejumlah solusi yang mungkin. Algoritma genetika memodelkan solusi sebagai kromosom yang terdiri dari beberapa gen, di mana setiap gen mewakili nilai dalam solusi. Setiap kromosom dalam populasi diberi skor atau nilai sesuai dengan seberapa baik mereka dalam memenuhi kriteria optimalitas yang didefinisikan. Populasi kemudian dievaluasi, dan teknik seleksi digunakan untuk memilih individu terbaik dalam populasi untuk reproduksi.

SOURCE CODE

```cpp
#include <bits/stdc++.h>
#include <limits.h>
using namespace std;
#define V 5
#define GENES ABCDE
#define START 0
#define POP_SIZE 10

struct individual {
    string gnome;
    int fitness;
};

int rand_num(int start, int end) {
    int r = end - start;
    int rnum = start + rand() % r;
    return rnum;
}

bool repeat(string s, char ch) {
    for (int i = 0; i < s.size(); i++) {
        if (s[i] == ch)
            return true;
    }
    return false;
}
```

```cpp
string mutatedGene(string gnome) {
    while (true) {
        int r = rand_num(1, V);
        int r1 = rand_num(1, V);
        if (r1 != r) {
            char temp = gnome[r];
            gnome[r] = gnome[r1];
            gnome[r1] = temp;
            break;
        }
    }
    return gnome;
}

string create_gnome() {
    string gnome = "0";
    while (true) {
        if (gnome.size() == V) {
            gnome += gnome[0];
            break;
        }
        int temp = rand_num(1, V);
        if (!repeat(gnome, (char)(temp + 48)))
            gnome += (char)(temp + 48);
    }
    return gnome;
}
```

SOURCE CODE

```
int cal_fitness(string gnome) {
    int map[V][V] = { { 0, 2, INT_MAX, 12, 5 },
                      { 2, 0, 4, 8, INT_MAX },
                      { INT_MAX, 4, 0, 3, 3 },
                      { 12, 8, 3, 0, 10 },
                      { 5, INT_MAX, 3, 10, 0 } };
    int f = 0;
    for (int i = 0; i < gnome.size() - 1; i++) {
        if (map[gnome[i] - 48][gnome[i + 1] - 48] == INT_MAX)
            return INT_MAX;
        f += map[gnome[i] - 48][gnome[i + 1] - 48];
    }
    return f;
}

int cooldown(int temp) {
    return (90 * temp) / 100;
}
```

```

bool lessthan(struct individual t1, struct individual t2) {
    return t1.fitness < t2.fitness;
}

void TSPUtil(int map[V][V]) {
    int gen = 1;
    int gen_thres = 5;

    vector<struct individual> population;
    struct individual temp;

    for (int i = 0; i < POP_SIZE; i++) {
        temp.gnome = create_gnome();
        temp.fitness = cal_fitness(temp.gnome);
        population.push_back(temp);
    }

    cout << "\nInitial population: " << endl << "GNOME   FITNESS VALUE\n";
    for (int i = 0; i < POP_SIZE; i++)
        cout << population[i].gnome << " " << population[i].fitness << endl;
    cout << "\n";
```

SOURCE CODE

```
    bool found = false;
    int temperature = 10000;

    while (temperature > 1000 && gen <= gen_thres) {
        sort(population.begin(), population.end(), lessthan);
        cout << "\nCurrent temp: " << temperature << "\n";
        vector<struct individual> new_population;

        for (int i = 0; i < POP_SIZE; i++) {
            struct individual p1 = population[i];

            while (true) {
                string new_g = mutatedGene(p1.gnome);
                struct individual new_gnome;
                new_gnome.gnome = new_g;
                new_gnome.fitness = cal_fitness(new_gnome.gnome);

                if (new_gnome.fitness <= population[i].fitness) {
                    new_population.push_back(new_gnome);
                    break;
                } else {
                    float prob = pow(2.7, -1 * ((float)(new_gnome.fitness - population[i].fitness) / temperature));
                    if (prob > 0.5) {
                        new_population.push_back(new_gnome);
                        break;
                    }
                }
            }
        }
```

SOURCE CODE

```
127             temperature = cooldown(temperature);
128             population = new_population;
129             cout << "Generation " << gen << " \n";
130             cout << "GNOME FITNESS VALUE\n";
131 
132             for (int i = 0; i < POP_SIZE; i++)
133                 cout << population[i].gnome << " " << population[i].fitness << endl;
134             gen++;
135         }
136     }
137 
138     int main() {
139         int map[V][V] = { { 0, 2, INT_MAX, 12, 5 },
140                           { 2, 0, 4, 8, INT_MAX },
141                           { INT_MAX, 4, 0, 3, 3 },
142                           { 12, 8, 3, 0, 10 },
143                           { 5, INT_MAX, 3, 10, 0 } };
144         TSPUtil(map);
145     }
```

OUTPUT

```
Initial population:
GNOME    FITNESS VALUE
024310 2147483647
021340 2147483647
024310 2147483647
041320 2147483647
013240 21
043210 24
043210 24
024130 2147483647
024310 2147483647
043120 2147483647
```

```
Current temp: 10000
Generation 1
GNOME FITNESS VALUE
012340 24
042310 21
034210 31
021340 2147483647
031240 32
034210 31
021340 2147483647
034120 2147483647
024130 2147483647
041320 2147483647
```

```
Current temp: 9000
Generation 2
GNOME FITNESS VALUE
042130 32
013240 21
043210 24
043210 24
013240 21
024310 2147483647
024310 2147483647
032140 2147483647
021430 2147483647
014320 2147483647
```

```
Current temp: 8100
Generation 3
GNOME FITNESS VALUE
031240 32
043210 24
034210 31
013240 21
012430 31
042310 21
014320 2147483647
031240 32
023410 2147483647
041320 2147483647
```

```
Current temp: 7290
Generation 4
GNOME FITNESS VALUE
043210 24
043210 24
042310 21
043210 24
012340 24
013240 21
034210 31
012340 24
023140 2147483647
043120 2147483647
```

```
Current temp: 6561
Generation 5
GNOME FITNESS VALUE
043210 24
012340 24
013240 21
013240 21
013240 21
013240 21
013240 21
031240 32
032140 2147483647
034120 2147483647
```

Tugas Individu 3

KECERDASAN BUATAN

MAP COLORING

Zakia kolbi (5025211049)

YANG KITA BAHAS!

- Penjelasan CSP
- Penjelasan Map Coloring
- Source Code
- Output

CONSTRAINT SATISFACTION PROBLEM

CSP atau Constraint Satisfaction Problem adalah permasalahan yang tujuannya adalah mendapatkan suatu kombinasi variabel-variabel tertentu yang memenuhi aturan-aturan (constraints) tertentu.

MAP COLORING

Map Coloring merupakan masalah matematika di mana kita harus menentukan jumlah minimum warna untuk memberikan warna yang berbeda pada setiap daerah sehingga tidak ada dua daerah yang bersebelahan memiliki warna yang sama.

SOURCE CODE

```python
from typing import Dict, List

def constraints(node: str, color: str, assignment: Dict[str, str], graph: Dict[str, List[str]]) -> bool:
    for neighbor in graph[node]:
        if neighbor in assignment and assignment[neighbor] == color:
            return False
    return True

def backtrack(assignment: Dict[str, str], graph: Dict[str, List[str]],
              domain: Dict[str, List[str]]) -> Dict[str, str]:
    if len(assignment) == len(graph):
        return assignment
    node = None
    for n in graph:
        if n not in assignment:
            node = n
            break
```

SOURCE CODE

```
    for value in domain[node]:
        if constraints(node, value, assignment, graph):
            assignment[node] = value
            result = backtrack(assignment, graph, domain)
            if result is not None:
                return result
            del assignment[node]
    return None


def map_coloring(graph: Dict[str, List[str]], colors: List[str]) -> Dict[str, str]:
    domain = {node: colors for node in graph}
    return backtrack({}, graph, domain)
```

SOURCE CODE

```
graph = {
    'WA':  ['NT', 'SA'],
    'NT':  ['WA', 'SA', 'Q'],
    'SA':  ['WA', 'NT', 'Q', 'NSW', 'V'],
    'Q':   ['NT', 'SA', 'NSW'],
    'NSW': ['Q', 'SA', 'V'],
    'V':   ['SA', 'NSW'],
    'T':   ['V']
}
colors = ['red', 'green', 'blue']
solution = map_coloring(graph, colors)
print(solution)
```

OUTPUT

```
{'WA': 'red', 'NT': 'green', 'SA': 'blue', 'Q': 'red', 'NSW': 'green', 'V': 'red', 'T': 'green'}
PS C:\Users\USER>
```

TERIMA KASIH