第4回課題 T322022 加藤 達也

分岐限定法

コード

```
from collections import deque
import sys

graph = []
parent = [0] * 11
g = [0] * 11
cost = {}  # Use a dictionary for cost

def gen_graph():
    graph.append([])
    graph.append([2, 3, 4])
    graph.append([5, 6])
    graph.append([2, 4, 7])
    graph.append([7, 8, 9])
    graph.append([6])
    graph.append([3, 7, 10])
    graph.append([9, 10])
    graph.append([9])
    graph.append([10])

    cost[(1, 2)] = 3
    cost[(1, 3)] = 2
    cost[(1, 4)] = 4
    cost[(2, 5)] = 2
    cost[(2, 6)] = 1
    cost[(3, 2)] = 1
    cost[(3, 4)] = 2
    cost[(3, 7)] = 5
    cost[(4, 7)] = 3
    cost[(4, 8)] = 2
    cost[(4, 9)] = 2
    cost[(5, 6)] = 3
    cost[(6, 3)] = 1
    cost[(6, 7)] = 2
    cost[(6, 10)] = 5
    cost[(7, 9)] = 2
    cost[(7, 10)] = 4
    cost[(8, 9)] = 1
    cost[(9, 10)] = 1

def c(a, b):
    if (a, b) in cost:
        return cost[(a, b)]
    return sys.maxsize

def branch_and_bound(start, goal):
    open = deque([start])
    closed = deque([])
    g[start] = 0

    while open:
        n = open.popleft()
        if n == goal:
            return
        closed.append(n)
        for m in reversed(graph[n]):
            if m not in open and m not in closed:
                g[m] = g[n] + c(n, m)
                parent[m] = n
                open.appendleft(m)
            elif m in open:
                if g[n] + c(n, m) < g[m]:
                    g[m] = g[n] + c(n, m)
                    parent[m] = n

        tmp = list(open)
        tmp = sorted(tmp, key=lambda x: g[x])
        open = deque(tmp)

def main():
    gen_graph()
    start, goal = 1, 10
    branch_and_bound(start, goal)

    n = goal
    print(n, end=' ')
    while n != start:
        n = parent[n]
        print("<-{}".format(n), end=' ')
    print("")

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

実行結果

```
10 <-9 <-4 <-1
```

山登り法

コード

```
from collections import deque
import sys

graph = []
parent = [0] * 11
g = [0] * 11
h = [0] * 11  # Initialize h as a list of appropriate length
cost = {}  # Use a dictionary for cost

def gen_graph():
    graph.append([])
    graph.append([2, 3, 4])
    graph.append([5, 6])
    graph.append([2, 4, 7])
    graph.append([7, 8, 9])
    graph.append([6])
    graph.append([3, 7, 10])
    graph.append([9, 10])
    graph.append([9])
    graph.append([10])
    graph.append([])  # Add an empty list for node 10

    cost[(1, 2)] = 3
    cost[(1, 3)] = 2
    cost[(1, 4)] = 4
    cost[(2, 5)] = 2
    cost[(2, 6)] = 1
    cost[(3, 2)] = 1
    cost[(3, 4)] = 2
    cost[(3, 7)] = 5
    cost[(4, 7)] = 3
    cost[(4, 8)] = 2
    cost[(4, 9)] = 2
    cost[(5, 6)] = 3
    cost[(6, 3)] = 1
    cost[(6, 7)] = 2
    cost[(6, 10)] = 5
    cost[(7, 9)] = 2
    cost[(7, 10)] = 4
    cost[(8, 9)] = 1
    cost[(9, 10)] = 1

    h[1] = 5
    h[2] = 1
    h[3] = 2
    h[4] = 3
    h[5] = 7
    h[6] = 4
    h[7] = 3
    h[8] = 2
    h[9] = 1
    h[10] = 0
```

```
def c(a, b):
    if (a, b) in cost:
        return cost[(a, b)]

    return sys.maxsize

def hill_climbing(start, goal):
    open = deque([start])
    closed = deque([])

    while open:
        n = open.popleft()
        if n == goal:
            return
        closed.append(n)
        neighbors = graph[n]
        best_neighbor = None
        best_h_value = sys.maxsize

        for m in neighbors:
            if h[m] < best_h_value and m not in closed:
                best_h_value = h[m]
                best_neighbor = m

        if best_neighbor is not None:
            parent[best_neighbor] = n
            open.append(best_neighbor)

def main():
    gen_graph()
    start, goal = 1, 10
    hill_climbing(start, goal)

    n = goal
    print(n, end=' ')
    while n != start:
        n = parent[n]
        print("<-{}".format(n), end=' ')
    print("")

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

実行結果

```
10 <-6 <-2 <-1
```

最良優先探索

コード

```
from collections import deque
import sys

graph = []
parent = [0] * 11
g = [0] * 11
h = [0] * 11  # Initialize h as a list of appropriate length
cost = {}  # Use a dictionary for cost

def gen_graph():
    graph.append([])
    graph.append([2, 3, 4])
    graph.append([5, 6])
    graph.append([2, 4, 7])
    graph.append([7, 8, 9])
    graph.append([6])
    graph.append([3, 7, 10])
    graph.append([9, 10])
    graph.append([9])
    graph.append([10])
    graph.append([])  # Add an empty list for node 10

    cost[(1, 2)] = 3
    cost[(1, 3)] = 2
    cost[(1, 4)] = 4
    cost[(2, 5)] = 2
    cost[(2, 6)] = 1
    cost[(3, 2)] = 1
    cost[(3, 4)] = 2
    cost[(3, 7)] = 5
    cost[(4, 7)] = 3
    cost[(4, 8)] = 2
    cost[(4, 9)] = 2
    cost[(5, 6)] = 3
    cost[(6, 3)] = 1
    cost[(6, 7)] = 2
    cost[(6, 10)] = 5
    cost[(7, 9)] = 2
    cost[(7, 10)] = 4
    cost[(8, 9)] = 1
    cost[(9, 10)] = 1

    h[1] = 5
    h[2] = 1
    h[3] = 2
    h[4] = 3
    h[5] = 7
    h[6] = 4
    h[7] = 3
    h[8] = 2
    h[9] = 1
    h[10] = 0

def c(a, b):
    if (a, b) in cost:
        return cost[(a, b)]

    return sys.maxsize

def best_first_search(start, goal):
    open = deque([start])
    closed = deque([])

    while open:
        n = open.popleft()
        if n == goal:
            return
        closed.append(n)
        for m in reversed(graph[n]):
            if m not in open and m not in closed:
                parent[m] = n
                open.appendleft(m)
        tmp = list(open)
        tmp = sorted(tmp, key = lambda x: h[x])
        open = deque(tmp)
def main():
    gen_graph()
    start, goal = 1, 10
    best_first_search(start, goal)

    n = goal
    print(n, end=' ')
    while n != start:
        n = parent[n]
        print("<-{}".format(n), end=' ')
    print("")

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

実行結果

```
10 <-7 <-3 <-1
```

A*探索

コード

```
from collections import deque
import sys

graph = []
parent = [0] * 11
g = [0] * 11
h = [0] * 11
f = [0] * 11
cost = {}

def gen_graph():
    graph.append([])
    graph.append([2, 3, 4])
    graph.append([5, 6])
    graph.append([2, 4, 7])
    graph.append([7, 8, 9])
    graph.append([6])
    graph.append([3, 7, 10])
    graph.append([9, 10])
    graph.append([9])
    graph.append([10])
    graph.append([])

    cost[(1, 2)] = 3
    cost[(1, 3)] = 2
    cost[(1, 4)] = 4
    cost[(2, 5)] = 2
    cost[(2, 6)] = 1
    cost[(3, 2)] = 1
    cost[(3, 4)] = 2
    cost[(3, 7)] = 5
    cost[(4, 7)] = 3
    cost[(4, 8)] = 2
    cost[(4, 9)] = 2
    cost[(5, 6)] = 3
    cost[(6, 3)] = 1
    cost[(6, 7)] = 2
    cost[(6, 10)] = 5
    cost[(7, 9)] = 2
    cost[(7, 10)] = 4
    cost[(8, 9)] = 1
    cost[(9, 10)] = 1

    h[1] = 5
    h[2] = 1
    h[3] = 2
    h[4] = 3
    h[5] = 7
    h[6] = 4
    h[7] = 3
    h[8] = 2
    h[9] = 1
    h[10] = 0
```

```
def c(a, b):
    if (a, b) in cost:
        return cost[(a, b)]

    return sys.maxsize

def a_star_search(start,goal):
    open = deque([start])
    closed = deque([])
    g[start] = 0

    while open != []:
        n = open.popleft()
        if n == goal: return

        closed.append(n)
        for m in reversed(graph[n]):
            if m not in open and m not in closed:
                g[m] = g[n] + c(n,m)
                f[m] = g[m] + h[m]
                parent[m] = n
                open.appendleft(m)
            elif m in open:
                if g[n] + c(n, m) + h[m] < f[m]:
                    g[m] = g[n] + c(n,m)
                    f[m] = g[m] + h[m]
                    parent[m] = n
            elif m in closed:
                if g[n] + c(n,m) + h[m] < f[m]:
                    g[m] = g[n] + c(n,m)
                    f[m] = g[m] + h[m]
                    parent[m] = n
                    open.appendleft(m)
                    closed.remove(m)

        tmp = list(open)
        tmp = sorted(tmp, key = lambda x: f[x])
        open = deque(tmp)


def main():
    gen_graph()
    start, goal = 1, 10
    a_star_search(start, goal)

    n = goal
    print(n, end=' ')
    while n != start:
        n = parent[n]
        print("<-{}".format(n), end=' ')
    print("")

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

実行結果

```
10 <-9 <-4 <-1
```