知的システム論第 8 回レポート

37186305
航空宇宙工学専攻修士一年
荒居秀尚

2018 年 11 月 23 日

1　宿題 1

Julia 1.0.0 で貪欲探索と最適探索を実装した。
実装にあたって、迷路はノード間の長さを重みとして与えた隣接行列の形で表現した。

1.1　貪欲探索

```
1  mutable struct edge
2      from::Int64
3      to::Int64
4      cost::Int64
5  end
6
7  function greedy(m)
8      openlist = [edge(1, 1, 0)]
9      closedlist = []
10     totalcost = 0
11     while length(openlist) > 0
12         s = pop!(openlist)
13         push!(closedlist, s)
14         totalcost += s.cost
15         if size(m)[1] == s.to
16             break
17         end
18         candidate = [i for i in 1:length(m[s.to, :]) if m[s.to, i] != 0]
19         next = reverse(sort(candidate, by=x -> m[s.to, x]))
20         closed_from = [n.from for n in closedlist]
21         for n in next
22             if n in closed_from
23                 continue
24             end
25             push!(openlist, edge(s.to, n, m[s.to, n]))
26         end
```

```
    end
    closedlist, totalcost
end

m = [0 1 0 2 0 0 0 0 0 0;
     1 0 2 0 0 0 0 0 0 0;
     0 2 0 0 0 0 1 0 0 0;
     2 0 0 0 1 0 0 0 0 0;
     0 0 0 1 0 2 0 0 0 0;
     0 0 0 0 2 0 1 0 0 0;
     0 0 1 0 0 1 0 1 0 0;
     0 0 0 0 0 0 1 0 1 2;
     0 0 0 0 0 0 0 1 0 0;
     0 0 0 0 0 0 0 2 0 0]

greedy(m)
```

これを実行すると、通る経路としては

$$A \to B \to C \to G \to F \to E \to D \to H \to I \to J$$

となる。ただし分岐に関しては、遡りは行わないこととした。この時ゴールまでのコストは 12 であった。

1.2 最適探索

```
import Base: sort


mutable struct node
    id::Int64
    min_cost::Int64
    from::Int64
end

function sort(a::Array{node, 1})
    costs = [n.min_cost for n in a]
    idx = sortperm(costs)
    a = a[idx]
end

function dijkstra(m)
    initnode = node(1, 0, 1)
    pendinglist = [initnode]
    allnodes = []
    closedlist = []
    while true
        pending_id = [n.id for n in pendinglist]
        if size(m)[1] in pending_id
            break
```

```
        end

        pendinglist = sort(pendinglist)
        s = popfirst!(pendinglist)
        all_id = [n.id for n in allnodes]
        if !(s.id in all_id)
            push!(allnodes, s)
        end
        idx = s.id
        root = m[idx, :]
        candidate_idx = [i for i in 1:length(root) if root[i] != 0]
        pending_id = [n.id for n in pendinglist]
        for i in candidate_idx
            cost = s.min_cost + root[i]
            if i in pending_id
                pending_idx = indexin(i, pending_id)[1]
                if cost < (pendinglist[pending_idx].min_cost)
                    pendinglist[pending_idx].min_cost = cost
                    pendinglist[pending_idx].from = idx
                end
                continue
            end
            push!(pendinglist, node(i, cost, idx))
        end
    end

    pending_id = [n.id for n in pendinglist]
    all_id = [n.id for n in allnodes]
    poslast = indexin(size(m)[1], pending_id)[1]
    current = pendinglist[poslast]
    push!(closedlist, current)
    while true
        if current.from == 1
            push!(closedlist, initnode)
            break
        end

        pos = indexin(current.from, all_id)[1]
        push!(closedlist, allnodes[pos])
        current = allnodes[pos]
    end
    sort(closedlist)
end

m = [0 1 0 2 0 0 0 0 0 0;
     1 0 2 0 0 0 0 0 0 0;
     0 2 0 0 0 0 1 0 0 0;
     2 0 0 0 1 0 0 0 0 0;
     0 0 0 1 0 2 0 0 0 0;
```

```
      0 0 0 0 2 0 1 0 0 0;
      0 0 1 0 0 1 0 1 0 0;
      0 0 0 0 0 0 1 0 1 2;
      0 0 0 0 0 0 0 1 0 0;
      0 0 0 0 0 0 0 2 0 0]

dijkstra(m)
```

これを実行すると、最終的な経路としては

$$A \to B \to C \to G \to H \to J$$

のコスト 7 の経路を発見することができることが確かめられた。また、この dijkstra(m) 内部で最終的な allnodes 配列を出力してみると、

$$A \to D \to E \to F$$

の経路も $G \to H$ を計算する前に検証していることがわかる。

2 宿題 2

Julia 1.0.0 で実装した。

```
import Base: sort, indexin


function options(p)
    idx = indexin(0, p)[1]
    indices = [CartesianIndex(i, j) for (i, j) in [
                (idx[1]-1, idx[2]), (idx[1]+1, idx[2]),
                (idx[1], idx[2]-1), (idx[1], idx[2]+1)
                ]]
    indices = [i for i in indices if 0 < i[1] <= size(p)[1] && 0 < i[2] <=
        size(p)[2]]
end


function flip(p, idx)
    zeropos = indexin(0, p)[1]
    p[zeropos], p[idx] = p[idx], p[zeropos]
    p
end


function number_of_misplace(p, o)
    sum(sign.(abs.(p - o)))
end


function manhattan(p, o)
```

```
    distance = 0
    for i in 1:maximum(p)
        idxp = indexin(i, p)[1]
        idxo = indexin(i, o)[1]
        cartesian_diff = idxp - idxo
        distance += abs(cartesian_diff[1]) + abs(cartesian_diff[2])
    end
    distance
end


mutable struct State
    state::Array{Int64, 2}
    before::Array{Int64, 2}
    min_cost::Int64
    heuristic::Int64
end


function sort(a::Array{State, 1})
    costs = [s.min_cost + s.heuristic for s in a]
    a[sortperm(costs)]
end


function indexin(elem::Array{Int64, 2}, arr::Array{Array{Int64, 2}, 1})
    [i for i in 1:length(arr) if arr[i] == elem][1]
end


function astar(p, o, f)
    pendinglist = [State(copy(p), copy(p), 0, f(p, o))]
    visitedlist = []
    closedlist = []
    while true
        pendinglist = sort(pendinglist)
        next = popfirst!(pendinglist)
        visited_state = [s.state for s in visitedlist]
        if !(next.state in visited_state)
            push!(visitedlist, next)
        end
        next_option = options(next.state)
        next_p = [flip(copy(next.state), i) for i in next_option]
        next_state = [State(np, copy(next.state), next.min_cost+1, f(np, o))
            for np in next_p]
        if o in next_p
            push!(visitedlist, State(o, copy(next.state), next.min_cost+1, f(
                o, o)))
            break
```

```
        end

        pending_state = [s.state for s in pendinglist]
        for ns in next_state
            if !(ns.state in pending_state)
                push!(pendinglist, ns)
            end
        end
    end
    visited_state = [s.state for s in visitedlist]
    current = o
    while true
        idxcurrent = indexin(current, visited_state)
        push!(closedlist, visitedlist[idxcurrent])
        if current == p
            break
        end
        before = visitedlist[idxcurrent].before
        idxbefore = indexin(before, visited_state)
        current = visitedlist[idxbefore].state
    end
    reverse(closedlist)
end


puzzle = [4 3 5;
          2 1 0]

optimal = [1 2 3;
           4 5 0]

astar(puzzle, optimal, number_of_misplace)
```

これを実行すると、最終的な経路としては、

$$\begin{bmatrix}4&3&5\\2&1&0\end{bmatrix}\to\begin{bmatrix}4&3&0\\2&1&5\end{bmatrix}\to\begin{bmatrix}4&0&3\\2&1&5\end{bmatrix}\to\begin{bmatrix}4&1&3\\2&0&5\end{bmatrix}\to\begin{bmatrix}4&1&3\\0&2&5\end{bmatrix}\to\begin{bmatrix}0&1&3\\4&2&5\end{bmatrix}\to\begin{bmatrix}1&0&3\\4&2&5\end{bmatrix}\to\begin{bmatrix}1&2&3\\4&0&5\end{bmatrix}\to\begin{bmatrix}1&2&3\\4&5&0\end{bmatrix}$$

が出力される。なお、上のプログラム例では誤りタイルの数をヒューリスティック評価値としているが、これをマンハッタン距離に変えても全く同じ結果が得られる。