Lydia Yu 15.076 HW3

Problem 1

1) **Problem 1**

```
# load map of amount of plastic in each square
p = load("plastic_map.jld2", "plastic_map");
```

```
# matrix to keep track of most amount of plastic that can be collected
# up to the current square
most_p_up_to = fill(-1000.0, 1001, 1001);
# matrix to keep track of which square you came from to get to current square
prec_nodes = Array{Tuple{Int, Int}}(undef,1001,1001);
```

```
# the first node can only the plastic that's already there
most_p_up_to[1,1] = p[1,1]

# loop through all nodes and fill in most_p_up_to and prec_node
for r = 1:1001
    for c = 1:1001
        # we already have node (1,1) filled in
        if (r==1) && (c==1)
            continue
        end

        # keep track of the indices and values from preceding nodes
        prec_max_p = Vector{Float64}()
        indices = Vector{Tuple{Int, Int}}()

        # check all potential preceding nodes within bounds?
        # keep track of max plastic from prec node and its idx
        # 1. node below
        if ((r-1) > 0) && ((r-1) < 1002)
            push!(prec_max_p, most_p_up_to[r-1, c])
            push!(indices, (r-1, c))
        end

        # 2. node from left
        if ((c-1) > 0) && ((c-1) < 1002)
            push!(prec_max_p, most_p_up_to[r, c-1])
            push!(indices, (r, c-1))
        end

        # 3. node left, down 1
        if ((r-1) > 0) && ((c-1) > 0) && ((r-1) < 1002) && ((c-1) < 1002)
            push!(prec_max_p, most_p_up_to[r-1, c-1])
            push!(indices, (r-1, c-1))
        end

        # 4. node left, down 2
        if ((r-2) > 0) && ((c-1) > 0) && ((r-2) < 1002) && ((c-1) < 1002)
            push!(prec_max_p, most_p_up_to[r-2, c-1])
            push!(indices, (r-2, c-1))
        end

        # 5. node left, up 1
        if ((r+1) > 0) && ((c-1) > 0) && ((r+1) < 1002) && ((c-1) < 1002)
            push!(prec_max_p, most_p_up_to[r+1, c-1])
            push!(indices, (r+1, c-1))
        end

        # 6. node left, up 2
        if ((r+2) > 0) && ((c-1) > 0) && ((r+2) < 1002) && ((c-1) < 1002)
            push!(prec_max_p, most_p_up_to[r+2, c-1])
            push!(indices, (r+2, c-1))
        end

        # find idx of max plastic from prec nodes
        max_idx = argmax(prec_max_p)

        # update values for current node by adding max prec plastic to p[r,c]
        most_p_up_to[r,c] = prec_max_p[max_idx] + p[r,c]
        prec_nodes[r,c] = indices[max_idx]
    end
end
```

[ ]:
```julia
# get the max plastic collected
most_p_up_to[1001, 1001]
```

[ ]:
```
154936.0784313725
```

[ ]:
```julia
# find which nodes were visited
course = Vector{Tuple{Float64, Float64}}()
r = 1001
c = 1001

while (r,c) != (1,1)
    # append this current node to nodes_visited
    # (subtract 1 to account for 1-indexing and divide by 10 to convert to miles)
    push!(course, ((r-1)/10,(c-1)/10))
    # get a tuple with indices of previous node
    node_before = prec_nodes[r,c]
    # update r,c to go to previous node
    r = node_before[1]
    c = node_before[2]
end
push!(course, (0,0));
```

[ ]:
```julia
# reverse the order of course to start at (1,1)
course = reverse(course)
```

[ ]:
```
2001-element Vector{Tuple{Float64, Float64}}:
 (0.0, 0.0)
 (0.0, 0.1)
 (0.0, 0.2)
 (0.0, 0.3)
 (0.0, 0.4)
 (0.0, 0.5)
 (0.0, 0.6)
 (0.0, 0.7)
 (0.0, 0.8)
 (0.0, 0.9)
 (0.0, 1.0)
 (0.0, 1.1)
 (0.0, 1.2)
 ⋮
 (98.9, 100.0)
 (99.0, 100.0)
 (99.1, 100.0)
 (99.2, 100.0)
 (99.3, 100.0)
 (99.4, 100.0)
 (99.5, 100.0)
 (99.6, 100.0)
 (99.7, 100.0)
 (99.8, 100.0)
 (99.9, 100.0)
 (100.0, 100.0)
```

2) If fuel is limited, the optimal course may not travel to less nodes and thus collect less total trash in order to reduce the nautical miles traveled (if the original optimal solution exceeds the fuel constraint).

You can't use the Dike Height optimization algorithm because there is no way to integrate the fuel constraint.

Variables: $X_{ij} = \begin{cases} 1 & \text{if pass through square } i,j \\ 0 & \text{ow} \end{cases}$

Parameter: p_{ij} = amount of plastic in square i,j

Constraints: need to pass through 1st and last square:

$$x_{1,1} = 1$$

$$x_{1001,1001} = 1$$

If I pass through i,j, I need to have passed through one of its previous neighbors:

$$(X_{i-1,j} + X_{i,j-1} + X_{i+1,j-1} + X_{i+2,j-1} + X_{i-1,j-1} + X_{i-2,j-1}) = 1$$

need to stay within the fuel limit

$$\sum_{i,j} 150\, x_{ij} \leq 45{,}000$$

$x_{ij} \in \{0,1\}$ binary var.

Objective: $\max\limits_{X_{ij}} \sum_{i=1}^{1001} \sum_{j=1}^{1001} p_{ij} x_{ij}$ (max. total plastic collected)

Problem 2

1) variables: $y_i = \begin{cases} 1 \text{ if distribution center is opened at hospital location } i \\ 0 \text{ otherwise} \end{cases}$

$x_{ij} = \begin{cases} 1 \text{ if hospital } i \text{ is assigned to blood distribution site } j \\ 0 \text{ otherwise} \end{cases}$

parameters: H = set of all hospital locations

d_{ij} = drive time between location i and j

objective: $\min\limits_{y_i, x_{ij}} z$

constraints: $\sum\limits_{i=1}^{H} y_i \leq 10$ (10 distribution sites must be opened)

$\sum\limits_{j=1}^{H} x_{ij} = 1 \quad \forall i$ (each hospital i is assigned to 1 distribution center)

$x_{ij} \leq y_i \quad \forall i,j \in H$ (hospital i can only be served by distribution center j if j is open)

$x_{ij}, y_i \in \{0,1\}$

$z \geq x_{ij} \cdot d_{ij} \quad \forall i,j \in H$ (min-max problem – minimize the max. distance that any hospital has to drive)

$z \geq 0$

2)

Problem 2

```
hospital = CSV.read("ziekenhuizen.csv", DataFrame);
```

```
drive = CSV.read("reistijden.csv", DataFrame);
# convert everything to strings
# (some columns are Time types and I couldn't get strings to convert Time)
drive[!, 2:ncol(drive)] = string.(drive[!, 2:ncol(drive)]);
```

```
# function to get min of max drive times to centers in solution
function objective(solution)
    # for each center in solution, find the max drive time any hospital needs to get to it
    max_drive_times = [maximum(drive[:, x]) for x in solution]
    # the minimum of the max drive times to each center in the solution is the objective value
    obj_val = minimum(max_drive_times)
    return obj_val
end
```

```
objective (generic function with 1 method)
```

```
# initialize greedy solution with first 10 hospitals
solution = hospital[!,"name"][1:10];
obj = objective(solution)
```

```
"02:44:52"
```

```
# algorithm converges when new solution produces same obj value as current solution
no_improvement = false
while !no_improvement
    # loop through all current centers in solution and do local search
    for center in solution
        # loop through all other hospitals to potentially replace center with
        for new in hospital[!,"name"]
            # do not check if new is already a part of solution
            if new ∉ solution
                # generate new solution by replacing center with new
                new_sol = deepcopy(solution)
                # remove center from new_sol
                deleteat!(new_sol, new_sol.==center)
                # add new to new_sol
                push!(new_sol, new)

                # compare new_sol to current solution
                new_obj = objective(new_sol)
                if new_obj < obj
                    solution = new_sol
                    break
                end
                # converge to solution if new is same as previous
                if new_sol == solution
                    no_improvement = true
                    break
                end

            end
        end
        if no_improvement
            break
        end
    end
end
```

Too many midterms and other work this week to figure out why this gets stuck in the while loop; the marginal hour spent debugging became more costly than the value of earning more points on this HW :(