RMG Theme Park Project

Rishabh Raniwala, Maya Ghosal, Gautham Kapoor

IDEA: Use a Greedy Algorithm

How to choose a heuristic?

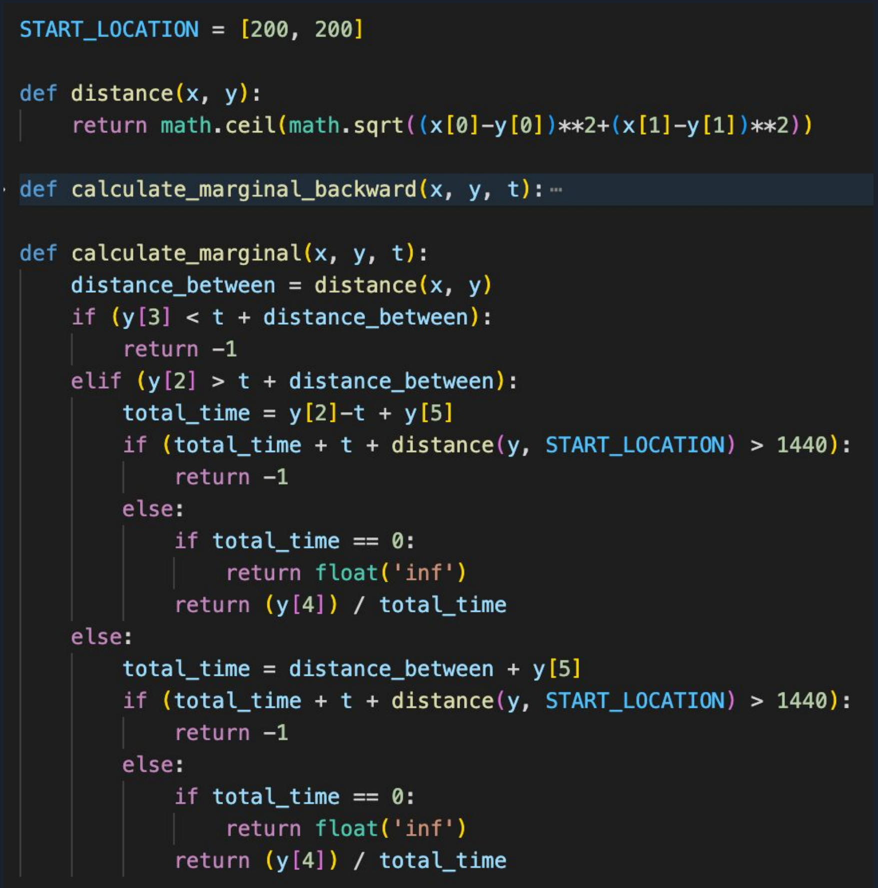
Could use marginal utility, but how should we calculate?

Teams could optimize against this, what other heuristics could we use?

Total utility

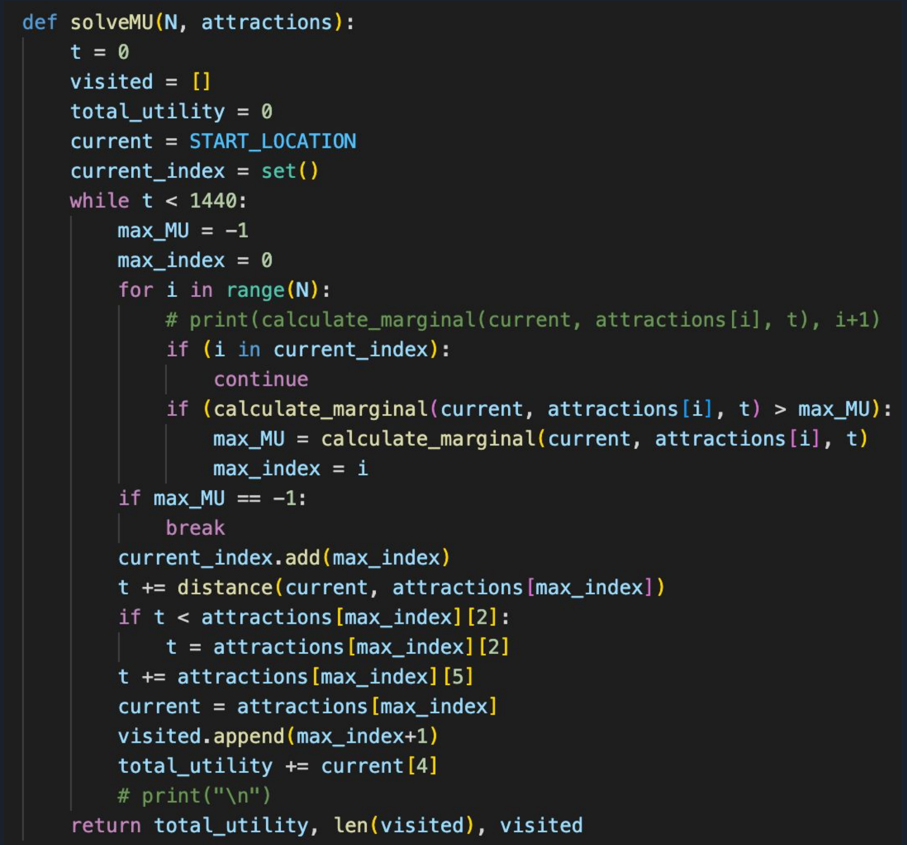
Total time

Calculating Marginal Utility



```python
START_LOCATION = [200, 200]

def distance(x, y):
    return math.ceil(math.sqrt((x[0]-y[0])**2+(x[1]-y[1])**2))

def calculate_marginal_backward(x, y, t):⋯

def calculate_marginal(x, y, t):
    distance_between = distance(x, y)
    if (y[3] < t + distance_between):
        return -1
    elif (y[2] > t + distance_between):
        total_time = y[2]-t + y[5]
        if (total_time + t + distance(y, START_LOCATION) > 1440):
            return -1
        else:
            if total_time == 0:
                return float('inf')
            return (y[4]) / total_time
    else:
        total_time = distance_between + y[5]
        if (total_time + t + distance(y, START_LOCATION) > 1440):
            return -1
        else:
            if total_time == 0:
                return float('inf')
            return (y[4]) / total_time
```

Using Marginal Utility to Output Ideal Attractions



```
def solveMU(N, attractions):
    t = 0
    visited = []
    total_utility = 0
    current = START_LOCATION
    current_index = set()
    while t < 1440:
        max_MU = -1
        max_index = 0
        for i in range(N):
            # print(calculate_marginal(current, attractions[i], t), i+1)
            if (i in current_index):
                continue
            if (calculate_marginal(current, attractions[i], t) > max_MU):
                max_MU = calculate_marginal(current, attractions[i], t)
                max_index = i
        if max_MU == -1:
            break
        current_index.add(max_index)
        t += distance(current, attractions[max_index])
        if t < attractions[max_index][2]:
            t = attractions[max_index][2]
        t += attractions[max_index][5]
        current = attractions[max_index]
        visited.append(max_index+1)
        total_utility += current[4]
        # print("\n")
    return total_utility, len(visited), visited
```

As An Aside

Inputs calculated using this heuristic!

First idea - make a cluster that seems to have higher marginal utility but ends up performing worse than a farther cluster

Second idea - create two clusters in the corners of the board along with two clusters closer to the starting location. A greedy algorithm would take a far cluster and a close cluster while the ideal solution traverses both far clusters.

Took the lower bound on marginal utility rather than the upper bound to be nice to our classmates ;)

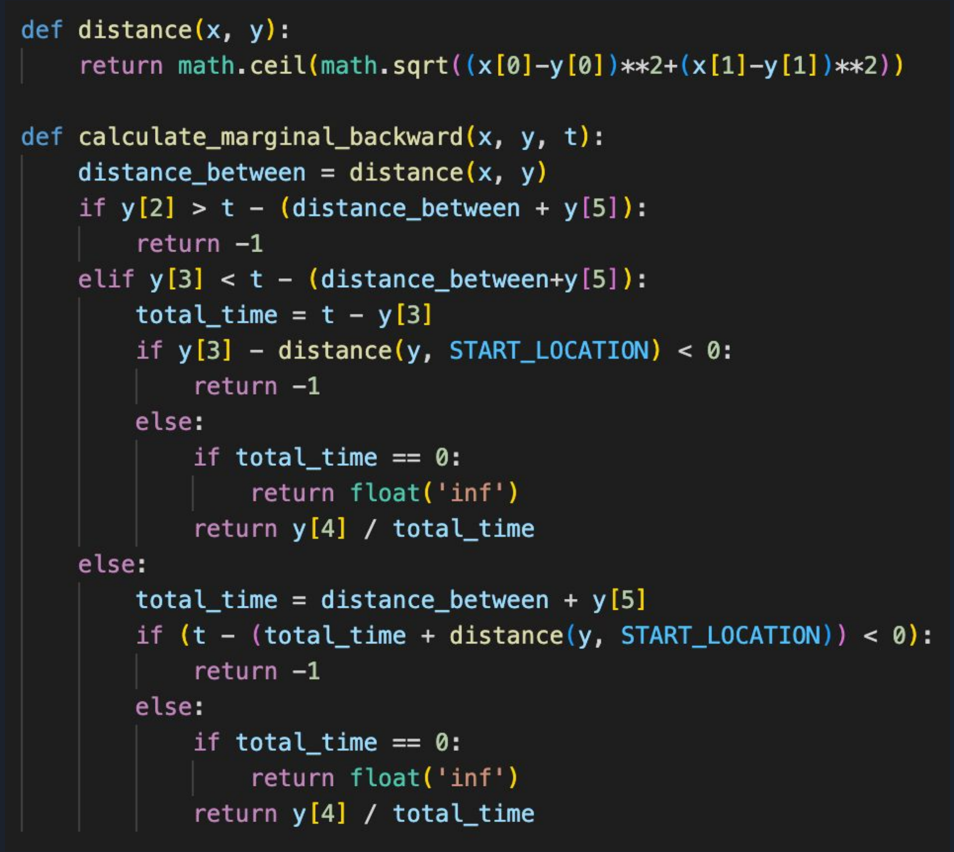
Still Falling Behind the Baseline!

How to catch up?

Currently calculating marginal utility from 0 to 1440, but as the problem is symmetrical, we can also calculate travelling backward from 1440 to 0!

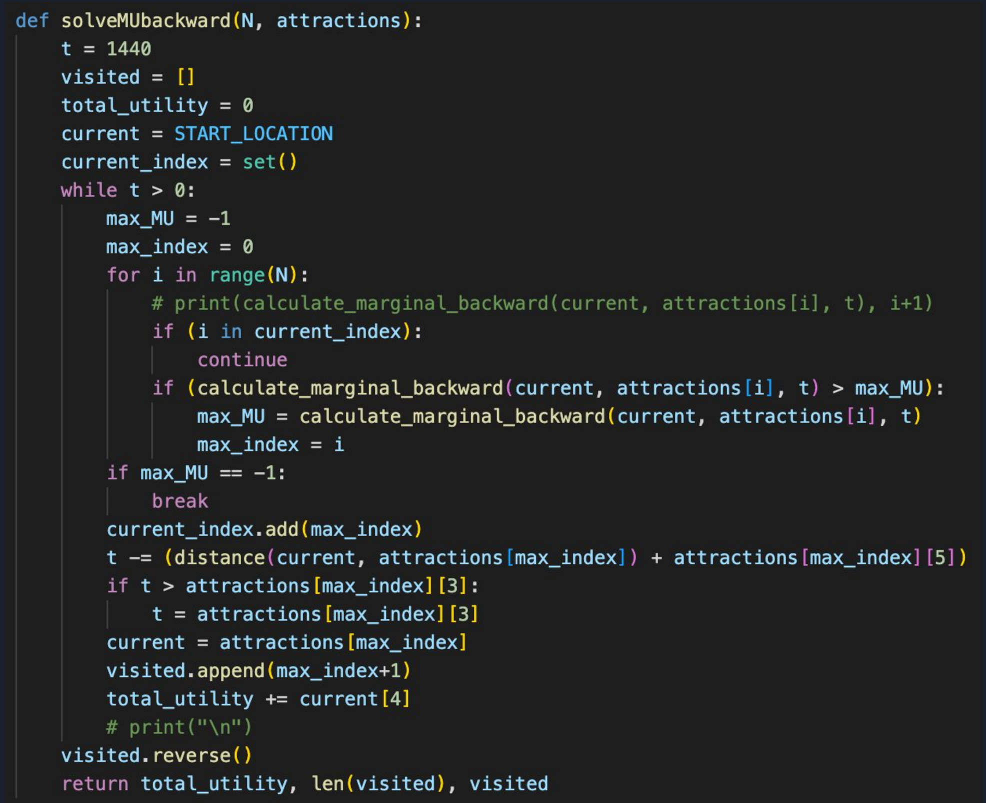
Could also apply this to total time, would not work for total utility as this does not change depending on the direction we travel

Calculating Backward Marginal Utility



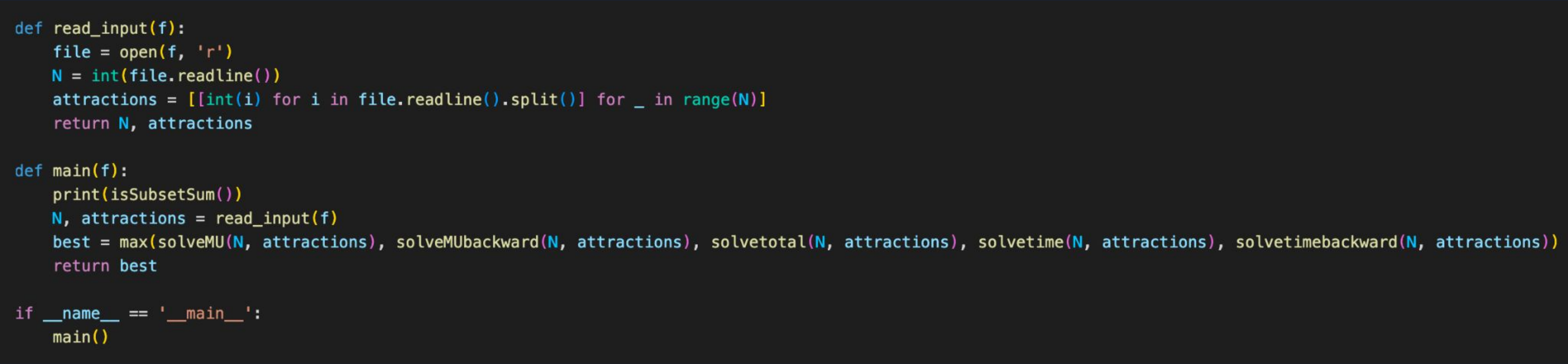
```python
def distance(x, y):
    return math.ceil(math.sqrt((x[0]-y[0])**2+(x[1]-y[1])**2))

def calculate_marginal_backward(x, y, t):
    distance_between = distance(x, y)
    if y[2] > t - (distance_between + y[5]):
        return -1
    elif y[3] < t - (distance_between+y[5]):
        total_time = t - y[3]
        if y[3] - distance(y, START_LOCATION) < 0:
            return -1
        else:
            if total_time == 0:
                return float('inf')
            return y[4] / total_time
    else:
        total_time = distance_between + y[5]
        if (t - (total_time + distance(y, START_LOCATION)) < 0):
            return -1
        else:
            if total_time == 0:
                return float('inf')
            return y[4] / total_time
```

Using Backward Marginal Utility to Output Ideal Attractions



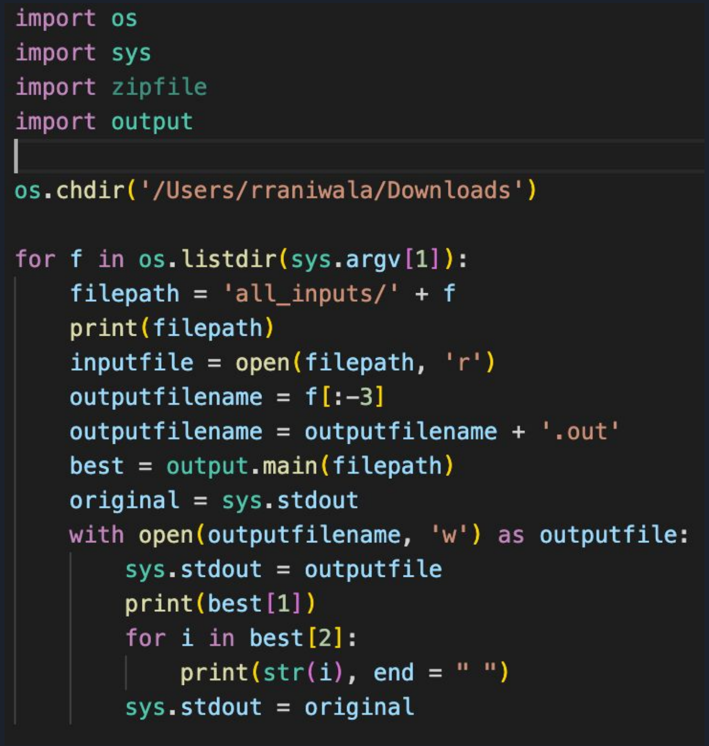
```python
def solveMUbackward(N, attractions):
    t = 1440
    visited = []
    total_utility = 0
    current = START_LOCATION
    current_index = set()
    while t > 0:
        max_MU = -1
        max_index = 0
        for i in range(N):
            # print(calculate_marginal_backward(current, attractions[i], t), i+1)
            if (i in current_index):
                continue
            if (calculate_marginal_backward(current, attractions[i], t) > max_MU):
                max_MU = calculate_marginal_backward(current, attractions[i], t)
                max_index = i
        if max_MU == -1:
            break
        current_index.add(max_index)
        t -= (distance(current, attractions[max_index]) + attractions[max_index][5])
        if t > attractions[max_index][3]:
            t = attractions[max_index][3]
        current = attractions[max_index]
        visited.append(max_index+1)
        total_utility += current[4]
        # print("\n")
    visited.reverse()
    return total_utility, len(visited), visited
```

Reading Input and Taking Best Heuristic



```
def read_input(f):
    file = open(f, 'r')
    N = int(file.readline())
    attractions = [[int(i) for i in file.readline().split()] for _ in range(N)]
    return N, attractions

def main(f):
    print(isSubsetSum())
    N, attractions = read_input(f)
    best = max(solveMU(N, attractions), solveMUbackward(N, attractions), solvetotal(N, attractions), solvetime(N, attractions), solvetimebackward(N, attractions))
    return best

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

Creating Output Files Efficiently



```python
import os
import sys
import zipfile
import output

os.chdir('/Users/rraniwala/Downloads')

for f in os.listdir(sys.argv[1]):
    filepath = 'all_inputs/' + f
    print(filepath)
    inputfile = open(filepath, 'r')
    outputfilename = f[:-3]
    outputfilename = outputfilename + '.out'
    best = output.main(filepath)
    original = sys.stdout
    with open(outputfilename, 'w') as outputfile:
        sys.stdout = outputfile
        print(best[1])
        for i in best[2]:
            print(str(i), end = " ")
        sys.stdout = original
```

With More Time...

Attempt to brute force small sized solutions

Use SubsetSum to brute force solutions

Use clusters of size k x k, calculate marginal utility of each cluster, traverse clusters with the largest marginal utility

With the above approach, we can also take a subset of vertices in a cluster that provides the best marginal utility

Brute force a certain number of k vertices and perform greedy starting on last vertex for all permutations of k vertices

What We Learned

Algorithmic design is hard! It requires a lot of brainstorming before starting to code, a lot of trial and error, and a lot of patience

Defense in algorithmic design can be as crucial to succeeding in an objective as offense

How to use Python scripting efficiently to accomplish a string of unrelated tasks

Thanks for a challenging but very fun project!

Questions?