Project 1

Algorithm 2: Greedy Approach to Hamiltonian Problem

Submitted By:

Thomas Ortiz

thomasortiz@csu.fullerton.edu

Alexander Sanchez

Alexandersanchez@csu.fullerton.edu

Ansh Tomar

atomar1@csu.fullerton.edu

Edgar Cardenas

cedgar@csu.fullerton.edu

Pseudocode:

Problem: Find the preferred starting city for a given circular road of cities that runs clockwise, where a car can visit each city and return to the starting point with fuel left over.

Input:

- *array [integers]* - holds the distances between neighboring cities on a circular path.
    - *city_distances*
- *array [integers]* - holds the available gas at each city, in gallons.
    - *fuel*
- *integer* - represents miles per gallon of the traveling car.
    - *Mpg*

Output:

- *integer* - the index of the preferred starting city.
    - *starting_City*

Constraints and Assumptions:

- *city_distances* only contains positive integers. The last index will connect the last city back to the starting city.
- *fuel* only contains positive integers.
- *mpg* is a positive integer.
- There is always exactly one valid starting city.
- Both input arrays are non-empty and of the same length.
- There are always at least two cities.
- The amount of fuel available is guaranteed to be enough to travel to each city back to the starting point.
- The car must return to the starting city with a level of fuel ≥ 0.

```
function greedyHamiltonian(city_distances[], fuel[], mpg)
    starting_City = int          // holds starting point tp return
    currentCity = int            // used to hold current city being visited
    nextCity = int               // used to get next city to visit
    tankFuelLevel = int          // holds the level of fuel in car
    visitedCount = int           // keeps track of how many cities have been visited
    fuelLength = int             // holds the length of the fuel array

    for index from 0 to length of fuel[] - 1:
        Reset tankFuelLevel to 0
        Set currentCity to index
        Reset visitedCount to 0

        While visitedCount is less than length of fuel[]:
            Add fuel from currentCity to tankFuelLevel

            if tankFuelLevel is sufficient to travel to next city:
                Subtract distance to next city from tankFuelLevel
                Move to the next city
                Increment visitedCount

            else:
                Break the while loop if tankFuelLevel is not enough to travel

        if visitedCount equals length of fuel[]:
            Set startingCity to index
            Break the for loop

    Return starting_City
```

Mathematical Analysis and Efficiency Proof:

Time Complexity:

```
function greedyHamiltonian(city_distances[], fuel[], mpg)
    starting_City = int          //O(1)
    currentCity = int            //O(1)
    nextCity = int               //O(1)
    tankFuelLevel = int          //O(1)
    visitedCount = int           //O(1)
    fuelLength = int             //O(1)

    for index from 0 to length of fuel[] - 1:    // Outer for loop is O(n)
        Reset tankFuelLevel to 0                 // O(1)
        Set currentCity to index                 // O(1)
        Reset visitedCount to 0                  // O(1)

        While visitedCount is less than length of fuel[]:    // Worst Case: O(n)
            Add fuel from currentCity to tankFuelLevel       //O(1)

            if tankFuelLevel is sufficient to travel to next city:    //O(1)
                Subtract distance to next city from tankFuelLevel     //O(1)
                Move to the next city                                 //O(1)
                Increment visitedCount                                //O(1)

            else:
                Break the while loop if tankFuelLevel is not enough to travel    //O(1)

        if visitedCount equals length of fuel[]:     //O(1)
            Set startingCity to index                //O(1)
            Break the for loop                       //O(1)

    Return starting_City                             //O(1)
```

- Outer ***for*** loop will iterate over each city to test for a potential starting point.
    - It will run ***n*** times for ***n*** amount of cities.
    - Everything inside the for loop will perform at an ***O*(1)** constant.
- Inner ***While*** loop will iterate up to ***n*** times to check each city as a potential starting point. ***O*(n)** is the worst case.

- Time Complexity = ***O*(1) * *O*(n) * *O*(n)**
  ***O*(n) * *O*(n)**
  ***O*(n^2)**

Proof by Induction:

1. $T(n) = n^2 + k$
2. Find c such that

$$T(n) \leq c \cdot f(n)$$

Assume $T(n) = n^2 + k$ and $f(n) = n^2$

Then, $T(n) \leq c \cdot f(n)$

$$n^2 + k \leq c \cdot n^2$$

Solve for c:

$$c \geq \frac{n^2 + k}{n^2} = 1 + \frac{k}{n^2}$$

Since $n \geq 1$, we assume $c = 1 + k$ and $n_o = 1$

3. When $n = n_o = 1$, $T(n_o) \leq c \cdot f(n_o)$

$$T(1) = 1^2 + k = 1 + k$$

$$c \cdot f(1) = c \cdot 1^2 = c$$

Since $c = 1 + k$:

$$T(1) = 1 + k \leq c = 1 + k$$

$$1 + k \leq 1 + k$$

Therefore, the base case holds.

4. If $n > n_o$ and $t(n) \leq c \cdot f(n)$, then $T(n+1) \leq c \cdot f(n+1)$

$$T(n) \leq c \cdot f(n)$$
$$n^2 + k \leq c \cdot n^2$$

Substitute (n) with (n+1)

$$T(n+1) = (n+1)^2 + k$$
$$n^2 + 2n + 1 + k$$

Expand and solve

$$c \cdot f(n+1) = c \cdot (n+1)^2$$
$$c \cdot (n^2 + 2n + 1)$$
$$c \cdot n^2 + 2c \cdot n + c$$

$$n^2 + 2n + 1 + k \leq c \cdot n^2 + 2c \cdot n + c$$

Therefore,

$$T(n+1) \leq c \cdot f(n+1)$$

5. By definition of *O,*

$$n^2 + k \in O(n^2)$$