Comp140 Review!

My Code For Homework

1. https://py3.codeskulptor.org/#user307_Q9QsbQwzls_2.py
2. https://py3.codeskulptor.org/#user307_My5Y9301yl_19.py
3. https://py3.codeskulptor.org/#user307_xqFgvbWJ9u_11.py
4. https://py3.codeskulptor.org/#user307_3xvfo5sMhV_24.py
5. https://py3.codeskulptor.org/#user307_C9EYsTCZRg_23.py
6. https://py3.codeskulptor.org/#user307_iitCnJiBci_49.py
7. https://py3.codeskulptor.org/#user307_GTpdmrfthv_47.py

Recursions

https://py3.codeskulptor.org/#user307_jKA4a0P8eG_8.py

https://py3.codeskulptor.org/#user307_wo05rkBzhl_1.py

Reference diagram

Actions

reference diagram

```
list1 = [{1: 2}, {3: 4}, {5: 6}]
list2 = list1[:]
list2[1] = {7: 8}
list2[2][9] = 10
print(list1)
print(list2)
```

Can someone explain in detailed english why list2[1] = {7:8} does not change {3:4} in list1 to {7:8}? Is that because list2[1] = {7:8} create a new object but doesn't affect the old {3:4}? But why

other

1. Edit good question | 0 Updated 12 minutes ago by Anonymous Atom

   S **the students' answer,** *where students collectively construct a single answer*

   Click to start off the wiki answer

   i **the instructors' answer,** *where instructors collectively construct a single answer*

   list2 = list1[:] creates a new list which is a copy of list1, not another reference to the same existing list1. Initially, list1 contains 3 references to those 3 dictionaries, and now list2 contains 3 (independent) references to those same 3 dictionaries. Now saying list2[1] = {7: 8} only changes the [1] reference in list2 to now refer to a different dictionary. list1 is undisturbed by that change to one of the 3 references in list2.

Theoretical

Module 2

1. Modular arithmetic
   - $a \div b \mod m = a \times (b^{m-2} \mod m) \mod m$
   - $b^{-1}\%p = b^{p-2}\%p$ (inverse)
2. Projective Geometry
   - https://canvas.rice.edu/courses/51272/pages/more-on-projective-geometry-reading?module_item_id=509691
   - ax + by + cz = 0: (x,y,z) is on the line [a,b,c]
   - Zero is not a valid point!
   - a line is a "circle"
   - point: (x,y,z) not all $\emptyset$
   - line: (x,y,z) not all $\emptyset$

- If k $\neq$ 0, then (x,y,z) = (kx, ky, kz)
- (x,y,z) is on [a,b,c] iff ax + by + cz = 0

3. The `range()` function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and stops before a specified number.

   - *range(start, incre, stop)*

Module 3

1. random.randrange(a,b): returns integer [a,b)
2. Markov chain:
   - Code to generate Markov chain: https://py3.codeskulptor.org/#user307_fyfftipLS6_0.py
   - Each of the n values can take on one of the numbers 0 through 3, so there are 4^n possible states in an nth order Markov chain.
3. MSE = $\sum \sqrt{(actual_i)^2 - (expected_i)^2}$
4. Testing data should be data from the past that you did not use to create the model, but is the same type as the training data.
5. enumerate(lst,n): generate [(0,a),(1,b)...] (starts with the second input)
6. zip: put lists together into a list of tuple

Module 4

1. Function vs method: Functions are defined outside of classes, while methods are defined inside of and part of classes.
2. Set difference: Elements present on one set, but not on the other
3. Symmetric Difference: Elements from both sets, that are not present on the other
4. Abstraction: user can look at the interface to use my code, but not my original code
5. Encapsulation: package a bunch of data and method in a class
6. BFS
   - BFS explores nodes by increasing distance from the starting node
   - Parent of a node: The parent of n is a node that is one step closer (in terms of distance) to the starting node s.
   - BFS Recipe: https://canvas.rice.edu/courses/51272/pages/bfs?module_item_id=509765
7. Graphs
   - Distance between two nodes:
     - The length of the shortest path between those two nodes
     - If you run BFS starting at one of the nodes, the distance between the two nodes is 1 plus the distance to the parent of the other node.
   - Graph API: https://canvas.rice.edu/courses/51272/pages/graph-class-api?module_item_id=509766
   - Diameter of graph: **the length of the shortest path between the most distanced nodes**
8. Object & References

- https://canvas.rice.edu/courses/51272/pages/objects-and-references-reading?module_item_id=509757
- Variable in python: reference to an object
- **Objects never contain other objects, they can only contain references**
- **References can only refer to object**

9. Advantage of using class: You can name the attributes of your data object, making your code more readable.
10. https://canvas.rice.edu/courses/51272/pages/kevin-bacon-writeup-solutions?module_item_id=509771

Module 5

1. Error correction
    - Check by parity
        - Byte: 8 bits
        - if original byte has odd bit sum, add 1 to the end; even, add 0 to the end
        - If the new bytes have even bit sum, then no error
        - else, there is an error
        - But I cannot tell which bit flipped
        - **Odd vs Even Paraty**
            - odd: add 1 bit, should be odd (so when original is even, add 1)
            - even: add 1 bit, should be even (so when original is even, add 0)
    - Check by hamming code
        - 0 1 2 3 4 5 6; 456 are parity bits
        - P4: 0,1,2; P5: 0,2,3; P6: 1,2,3
        - Hamming difference: # of different bits
        - Only 16 code words, and between each other, the hamming distance is 3
        - So can correct single bit error
        - However if there are two bit errors, it is distance 1 from a code word that does not correspond to the correct code word we want
        - Hamming code utility: https://py3.codeskulptor.org/#user307_Z7foCSxS3F_2.py
2. Binary int conversion: https://py3.codeskulptor.org/#user307_QIxv6m6Ht7_0.py
3. z-256 API: https://canvas.rice.edu/courses/51272/pages/z256-api?module_item_id=509795
4. Polynomial calculator: https://www.emathhelp.net/calculators/algebra-1/polynomial-calculator/
5. Reed-Solomon
    a. turn original message into blocks
    b. msg polynomial
    c. generator polynomial
    d. msg % gen = remainder
    e. turn remainder into error correction bytes
    f. combine error correction bytes + msg block = RS encoded data
    g. k: # of error correction bytes that are added to each message block

h. formulas: https://canvas.rice.edu/courses/51272/pages/reed-solomon-example?module_item_id=509796

6. Programming principle
    - primary reason for not duplicating code: You only need to get the code correct once
    - why create function:
        - To keep something that might change encapsulated in one place
        - To enable reuse of code.
        - To make the code using the function easier to understand
        - To simplify a complicated predicate in a conditional.
        - To isolate a complicated expression.
    - Constants should be capitalized

Module 6

1. Read files
    - https://py3.codeskulptor.org/#user307_UcbHmim7tq_0.py read file & decode
    - https://py3.codeskulptor.org/#user307_zBmRZDCBUh_0.py count word frequency
    - https://py3.codeskulptor.org/#user307_keAtI7c0oA_0.py read float and int
    - file.read() returns byte
2. Linear Algebra
    - Matrix calculation: https://py3.codeskulptor.org/#user307_WMaqDSMWbj_0.py
    - Matrix properties: https://canvas.rice.edu/courses/51272/pages/matrix-properties?module_item_id=509822
3. filter: e.g. filter(positive,data) produce a sequence that each element contained in data returns true for the function "positive"
4. map: produce a sequence that applies a function to elements of input sequence
5. higher order function: use function as input
6. Hill descent:
    - objective of hill descent: To attempt to find the minimum value of a given mathematical function. (I don't need to calculate slope!)
    - Cyclic minimization: A process where you minimize an n-dimensional mathematical function in one dimension at a time. You "cycle" through each dimension in turn and then repeat, as necessary, until you find a minimum.
7. Lasso shooting: https://canvas.rice.edu/courses/51272/pages/lasso-shooting?module_item_id=509823

Module 7

1. DFS
    - Generate a random path
    - DFS will always find a path to connect a node to another
    - BFS recipe: change stack to queue - https://canvas.rice.edu/courses/51272/pages/bfs?module_item_id=509765
2. A*

- A* explore the nodes in the graph: by choosing the node that is furthest along a path that could potentially have the shortest distance to the target node at each step.
- A node is added into closed set: After it has been the node selected with the minimum total cost once
- g: the actual cost to get to node n from the start node.
- h: The heuristic estimate of the cost to get from node n to the end node. This estimate must be a lower bound on the actual cost
- f: The sum of the actual cost to get from the start node to node n plus the estimate to get from node n to the end nod = g + h
- https://canvas.rice.edu/courses/51272/pages/a-star?module_item_id=509847

Recipes

Module 1

https://canvas.rice.edu/courses/51272/pages/circles-writeup-solutions?module_item_id=509673

Module 2

https://canvas.rice.edu/courses/51272/pages/spot-it-writeup-solutions?module_item_id=509707

Recipe: generate_all_points

Inputs: a positive integer prime modulus p indicating that we are generating all points in the projective geometric space in $\mathbb{Z}_p$

1. Initialize $candidates$ to be an empty sequence
2. **For each item x_index in the sequence $0, 1, ... p-1$ do**
    A. For each item y_index in the sequence $0, 1, ... p-1$ do
        i. For each item y_index in the sequence $0, 1, ... p-1$ do
            a. if $x\_index \neq 0$ or $y\_index \neq 0$ or $z\_index \neq 0$
                1. Add the triple $(x\_index, y\_index, z\_index)$ to the end of $candidates$
3. Initialize $result$ to be an empty sequence
4. For each item $candidate$ in the sequence $candidates$, do
    A. $unique \leftarrow true$
    B. For each item $point$ in the sequence $result$, do
        i. If calling $equivalent(candidate, point, p)$ returns $true$
            a. Assign $unique$ to have the value $false$
    C. If $unique$ is equal to $true$ then
        i. add $candidate$ to the end of $result$
5. Return $result$

Recipe: create_cards

Assume that each card is a sequence of integers, where each integer corresponds to a unique image

Inputs: a positive integer prime modulus, p; a sequence of lines, $lines$, where each element in $lines$ is a line [a,b,c] in the projective geometric space in $\mathbb{Z}_p$; and a sequence of points, $points$, where each element in $points$ is a point (x,y,z) in the projective geometric space in $\mathbb{Z}_p$

1. Initialize $cards$ to be an empty sequence
2. Initialize $numpoints$ to be the length of the sequence $points$
3. For each item $line$ in the sequence $0, 1, \ldots, numpoints - 1$, do
    A. Assign $card$ to be an empty sequence
    B. If calling $incident(point, line, p)$ returns true
        a. Add $index$ to the end of the sequence $card$
    C. Add $card$ to the end of the sequence $cards$
4. Return cards.

Recipe: judge_occur

Inputs: a sequence of numbers lst; a nonnegative number $pos0$, which represents index of a number in the input sequence lst; a positive integer dis, which represents that only the number that has equal number within dis will be considered occur in the lst (i.e. if a number is at $pos0$, only if there is another number that equals to the original number from $lst_{pos0-dis}$ to $lst_{pos0+dis}$ will be considered occur)

Output: Return whether or not the number at position $pos0$ in lst occur

1. $occur \leftarrow false$;
2. $num \leftarrow lst_{pos0}$;
3. $pos \leftarrow pos0 - 1$;
4. $count \leftarrow dis$;
5. while $count$ is greater than 0 and pos is greater than 0, do
    A. if lst_{pos} is equal to num, do
        a. $occur \leftarrow true$;
        b. exit the while loop;
    B. Otherwise, do
        a. $pos \leftarrow pos - 1$;
        b. $count \leftarrow count - 1$;
6. $pos \leftarrow pos0 + 1$;
7. $count \leftarrow dis$;
8. While $count$ is greater than 0 and pos is smaller than length of lst, do
    A. if lst_{pos} is equal to num, do
        a. $occur \leftarrow true$;
        b. exit the while loop;
    B. Otherwise, do
        a. $pos \leftarrow pos + 1$;
        b. $count \leftarrow count - 1$;
9. Return $occur$;

Recipe: count_occurrence

Inputs: a sequence of numbers lst; a positive integer dis, which represents that only the number that has a equal number within dis in lst will be considered occur in the lst (i.e. if a number is at $pos0$, only if there is another number that equals to the original number from $lst_{pos0-dis}$ to $lst_{pos0+dis}$ will be considered occur)

Output: Return a mapping representing the number of occurrence of each unique number in lst

1. $dic \leftarrow$ an empty mapping;
2. $length \leftarrow$ length of lst;
3. **For each item itm in the sequence lst, do**

   A. If itm does not exist as a key in the mapping dic, do

      a. $dic_{itm} \leftarrow 0$

4. **For each item pos in the sequence $0, 1, 2, \ldots, length - 1$, do**

   A. If $judge\_occur(lst, pos, dis)$ returns $true$, do

      a. $dic_{lst_{pos}} \leftarrow dic_{lst_{pos}} + 1$;

5. Return dic;

Module 3

https://canvas.rice.edu/courses/51272/pages/stock-prediction-writeup-solutions?module_item_id=509737

Name: make_Markov

Input: An ordered sequence of data, $data$; and the order, n, of the desired Markov chain.

1. $chain \leftarrow$ an empty mapping;
2. $length \leftarrow$ the number of elements in $data$;
3. **For each number, idx, from 0 to $length - n - 1$ do**

   A. $current \leftarrow$the sequence of values $data_{idx}$ through $data_{idx+n-1}$;

   B. $next \leftarrow$ the mapping for $chain_{current}$, or a new empty mapping if one does not exist;

   C. If $data_{idx+n}$ exists as a key in the mapping $next$, increment $next_{data_{idx+n}}$ by 1. otherwise, set $next_{data_{idx+n}}$ to be 1;

   D. $chain_{current} \leftarrow$ the mapping $next$;

4. For every key, $state$, in $chain$, normalize the values within the mapping $chain_{state}$ so that they sum to 1, by dividing each count in this mapping by the total counts in the mapping, thus converting the counts in the mappings into probabilities;
5. Return $chain$

Name: weighted_choice

Input: A mapping, choices, that maps possible choices to the probability of selecting that choice

1. $rnd \leftarrow$ a uniform distribution random number in [0,1);
2. $total \leftarrow 0$;
3. For each key, $choice$, in the $choices$ mapping do
   A. $total \leftarrow total + choices_{choices}$;
   B. If $rnd < total$, then return $choice$;

Name: predict

Input: An n^{th} order Markov chain, $model$; the last n values, $last$, that have occurred; and the number of predictions to make, num.

1. $choices \leftarrow$ an empty sequence;
2. For each number, $trial$, from 0 to $num - 1$ do
   A. If $last$ exists as a key in the mapping $model$ do
      i. $next \leftarrow model_{last}$;
      ii. $choices_{trial} \leftarrow weighted\_choice(next)$;
   B. Otherwise (i.e., $last$ does not exist as a key in $model$) do
      i. $choices_{trial} \leftarrow$ a uniform distribution random integer between 0 and 3;
   C. $last \leftarrow$ a new sequence containing $last_1, \ldots, last_{n-1}, choices_{trial}$;
3. Return $choices$

Module 4

https://canvas.rice.edu/courses/51272/pages/kevin-bacon-writeup-solutions

Recipe: Find Path

Input: Undirected graph $graph$, start node $start\_person$ in $graph$, end node end_person in $graph$, and mapping $parents$ that associates each node in$graph$ with its parent in the traversal of $graph$

Output: $path$, a sequence of "steps" leading from $start\_person$ to end_person in the $graph$. The step at each index i in $path$, $0 \leq i < k - 1$, where $k =$length of $path$, will be represented as a tuple of the form $(node_i$, attributes of edge $(node_i, node_{i+1})$ in $graph)$; the final node in $path$, $node_{k-1}$, will be end_person and the final step in $path$ will be represented as the tuple $(end\_person$, empty set), since there is no succeeding node in the path after end_person. The node in the first step in $path$, $node_0$, will be $start\_person$.

1. $path \leftarrow$a sequence containing only the single tuple $(end\_person, \text{empty set})$;
2. $current \leftarrow end\_person$;
3. while $current \neq start\_person$ do
   A. $previous \leftarrow parents_{current}$;
   B. if $previous = null$ then
      I. return an empty sequence;
   C. $edge\_attrs \leftarrow$ the set of attributes on the edge $(previous, current)$ in graph;
   D. Insert the tuple $(previous, current)$ before the first element in the sequence $path$;

E. $current \leftarrow previous$;

4. return $path$;

Module 5

https://canvas.rice.edu/courses/51272/pages/qr-code-writeup-solutions

Recipe: $multiply\_by\_term$ (using an abstract mathematical approach)

Input: $poly$, a polynomial; c, a coefficient in$\mathbb{Z}_{256}$; and p, an exponent in $\mathbb{Z}$

Output: A polynomial that is the produce $poly \times cx^p$

1. $result \leftarrow 0$;
2. foreach term $a_i x^i$ in the polynomial $poly$ do
   A. $b \leftarrow a_i \times x$, using $\mathbb{Z}_{256}$ math;
   B. $j \leftarrow p + i$, using standard integer math;
   C. $result \leftarrow result + bx^j$, using $\mathbb{Z}_{256}$ math for coefficient addition;
3. return $result$;

Recipe: $multiply\_by\_term$ (using a Polynomial class-based approach)

Input: $poly$, a instance of the class Polynomial; c, a coefficient in$\mathbb{Z}_{256}$; and p, an exponent in $\mathbb{Z}$

Output: An instance of the class Polynomial that represents the produce $poly \times cx^p$

1. $result \leftarrow$ a new instance of the class Polynomial representing 0;
2. $terms \leftarrow get\_terms(poly)$:
3. **foreach key i in the map $terms$ do**
   A. $a_i \leftarrow terms_i$;
   A. $b \leftarrow a_i \times c$, using $\mathbb{Z}_{256}$ math;
   B. $j \leftarrow p + i$, using standard integer math;
   C. $result \leftarrow add\_term(result, b, j)$;
3. return $result$;

Recipe: add_polynomial

Input: $poly_1$ and $poly_2$, both polynomials

Output: A polynomial that is the sum $poly_1 + poly_2$

1. $result \leftarrow poly_1$;
2. foreach term $a_i x^i$ in $poly_2$ do
   A. $result \leftarrow add\_term(result, a_i, i)$;
3. return $result$;

Recipe: multiply_by_polynomial

Input: $poly_1$ and $poly_2$, both polynomials

Output: A polynomial that is the sum $poly_1 \times poly_2$

1. $result \leftarrow 0$;
2. foreach term $a_i x^i$ in $poly_2$ do
   A. $termprod \leftarrow multiplu\_by\_term(poly_1, a_i, i)$;
   B. $result \leftarrow add\_polynomial(result, termprod)$;
3. return $result$;

Recipe: remainder

Input: $numerator$, a polynomial of the form $a_n x^n + \cdots + a_1 x^1 + a_0$; and $denominator$, a polynomial of the form $b_m x^m + \cdots + b_1 x^1 + b_0$

Output: A polynomial that is the remainder after dividing $numerator$ by $denominator$

1. $remaining \leftarrow$ a polynomial equal to $numerator$, of the form $c_k x^k + \cdots + c_1 x^1 + c_0$
2. $m \leftarrow$ the degree of the polynomial $denominator$;
3. $k \leftarrow$ the degree of the polynomial $remaining$;
4. while $k \geq m$ and $remaining \neq 0$ do
   A. $factor \leftarrow divide\_terms(c_k, k, b_m, m)$;
   B. $subtrachend \leftarrow multiply\_by\_polynomial(denominator, factor)$;
   C. $remaining \leftarrow subtract\_polynomial(remaining, subtrahend)$;
   D. $k \leftarrow$ the degree of the polynomial $remaining$;
5. return $remaining$;

Module 6

https://canvas.rice.edu/courses/51272/pages/sports-analytics-writeup-solutions

Recipe: $ReadMatrix$

Input: $lines$, a multi-line string where each line consists of a series of comma-separated substrings (i.e., each substring is separated from the next by ","), with each substring representing a decimal value

Output: a matrix whose (i,j) entry is the j-th value on the i-th line in $lines$.

1. $i \leftarrow 0$;
2. foreach $line$ in $lines$ do
   A. $row_i \leftarrow$ an empty sequence;
   B. foreach val_str comma-separated substring in $line$ do
      I. $val \leftarrow$ the numeric value represented by the string val_str when interpreted in decimal notation;

II. append val to the end of the sequence row_i;

3. $M \leftarrow$ the matrix whose i-th row is given by the sequence row_i;
4. return M;

Recipe: $GeneratePredictions$

Input: w, an $m \times 1$ matrix giving the weights of a linear model; and X, an $n \times m$ matrix of explanatory variables

Output: An $n \times 1$ matrix that is the vector of wins predicted by the model given by w when provided with the data X

1. return Xw;

Recipe: $PredictionError$

Input: w, an $m \times 1$ matrix giving the weights of a linear model; X, an $n \times m$ matrix of explanatory variables; and y, an $n \times 1$ matrix of actual values for the measured variable

Output: The Mean-Squared Error between the values predicted by the model given by w and the actual measured values

1. $p \leftarrow$ the $n \times 1$ matrix returned by $GeneratePredictions(w, X)$;
2. $actual \leftarrow$ the sequence of values given by the entries of the matrix y;
3. $predict \leftarrow$ the sequence of values given by the entries of the matrix p;
4. return $MSE(actual, predict)$;

Module 7

Map search answers: https://canvas.rice.edu/courses/51272/pages/map-search-writeup-solutions

BFS_DFS

Input: $graph$, a graph; RAC, a restricted access container class; $start$, the start node in $graph$;, and end, the end node in $graph$

Output: $parent$, a mapping, in which for each node $node$ in $graph$, $parent_{node}$ gives the parent of $node$ in the exploration of $graph$ starting from start

1. $rac \leftarrow$ a new instance of the RAC class;
2. foreach node, $node$, in $graph$ do
   A. $parent_{node} \leftarrow null$;
3. push $start$ onto rac;
4. while rac is not empty do
   A. if $parent_{nbr} = null$ and $nbr \neq start$ then
      i. $parent_{nbr} \leftarrow node$;
      ii. push nbr onto rac;
      iii. if $nbr = end$ then

a. return $parent$;

5. return $parent$;

Recursive_DFS

Input: $graph$, a graph; $start$, the start node in $graph$; end, the end node in $graph$; and $parent$, a mapping, in which each $parent_{node}$ gives the parent of $node$ in the exploration of graph starting from $start$ (on the initial call, i.e., from outside this function, $parent$ should be initialized to contain only the single correspondence $start \mapsto null$, for the original start node)

Output: $true$ if end has been found, or $false$ otherwise (also modifies $parent$)

1. if $start = end$ then
   A. return $true$;
2. foreach neighbor, nbr, of $start$ in $graph$ do
   A. if nbr is not a key in $parent$ then
      i. $parent_{nbr} \leftarrow start$;
      ii. if $Recursive\_DFS(graph, nbr, end, parent) = true$ then
         a. return $true$;
3. return $false$;

A*

Input: $graph$, a graph; $start$, the start node in $graph$; end, the end node in $graph$; $edgedist(u, v)$, a function that gives the distance of edge (u, v) in $graph$; and $heurdist(u)$, a function that gives the heuristic distance from node u to node end

Output: $parent$, a mapping, in which each for each node $node$ in $graph$, $parent_{node}$ gives the parent of $node$ in the exploration of of $graph$ starting from $start$

1. foreach node, $node$, in $graph$ do
   A. $parent_{node} \leftarrow null$;
2. $g\_cost_{start} \leftarrow 0$;
3. $h\_cost_{start} \leftarrow heurdist(start)$;
4. $f\_cost_{start} \leftarrow g\_cost_{start} + h\_cost_{start}$;
5. $openset \leftarrow \{start\}$;
6. $closedset \leftarrow \emptyset$;
7. while $openset \neq \emptyset$ do
   A. $curnode \leftarrow null$;
   B. $f\_cost \leftarrow \infty$;
   C. foreach node, $node$, in $openset$ do
      I. if $f\_cost \leq f\_low$ then
         i.$curnode \leftarrow node$;

ii. $f\_low \leftarrow f\_cost_{node}$;

D. if $curnode = end$ then

I. return $parent$;

E. Remove $curnode$ from $openset$;

F. Add $curnode$ to $closedset$;

G. foreach neighbor, nbr, of $curnode$ in $graph$ do

I. if $nbr \notin closedset$ then

i. $new\_g\_cost \leftarrow g\_cost_{curnode} + edgedist(curnode, nbr)$;

ii. if $nbr \notin openset$ then

a. $g\_cost_{nbr} \leftarrow new\_g\_cost$;

b. $h\_cost_{nbr} \leftarrow heurdist(nbr)$;

c. $f\_cost_{nbr} \leftarrow g\_cost_{nbr} + h\_cost_{nbr}$

d. $parent_{nbr} \leftarrow curnode$;

e. Add nbr to $openset$;

iii. else if $new\_g\_cost < g\_cost_{nbr}$ then

a. $g\_cost_{nbr} \leftarrow new\_g\_cost$;

b. $f\_cost_{nbr} \leftarrow g\_cost_{nbr} + h\_cost_{nbr}$;

c. $parent_{nbr} \leftarrow curnode$;

8. return $parent$;

1. https://py3.codeskulptor.org/#user307_Q9QsbQwzls_2.py
2. https://py3.codeskulptor.org/#user307_My5Y9301yI_19.py
3. https://py3.codeskulptor.org/#user307_xqFgvbWJ9u_11.py
4. https://py3.codeskulptor.org/#user307_3xvfo5sMhV_24.py
5. https://py3.codeskulptor.org/#user307_C9EYsTCZRg_23.py
6. https://py3.codeskulptor.org/#user307_iitCnJiBci_49.py
7. https://py3.codeskulptor.org/#user307_GTpdmrfthv_47.py