Fmlajfm.sd fmlskdjfksd fkl

ALGORITHMS USED IN EMERGENCY EVACUATION ROUTE SIMULATOR IN AMS BULDING-UNC

**Constructing the Representation of the Building**

1. Declare *wall, path, stairs, space* and *evacuation area* as integer, set *wall* as 3, *path* as 12, *stairs* as 11, *space* as 2 and *evacuation* *area* as 10.
2. Declare an *array* that can represent the desired building. 2D array represents a single floor, while 3D array can represent 2 or more floors.
3. Plot the elements of the building by floor using the elements stated on number 1. Remember that all *path* must be connected horizontally, vertically and slanting manner only. *One* path will represent an actual path; *path* represents a two-way direction so no need to declare two virtual path as a double way. Remember not to have a hanging *path* because it will never be visited. *Stairs* should always beside a *path*, *stairs* are also a two-way direction, so no need to put connected *stairs*. *Evacuation* *area* should also beside *path*. *Stairs* and *evacuation* *area* should have *path* in between. *Space* represents the area that the building wasn’t consuming. Always put *space* value around the array so that searching won’t be out of bounds.

**Evacuation Route finding: applying the Concept of Dijkstra’s Algorithm**

1. Create a linked list named *route* with integer values; *value, floor, row, col, order* then the *link*.(LIFO manner)
2. Declare different *route* linked list; *possible* for all multiple routes encountered, usually all the possible route information that wasn’t the shortest was temporarily stored here. *Current* which will be used during simulation, it will hold the value of all the path that the algorithm is currently taking. *Shortest* holds the shortest route, shortest will be updated when the current route who reached the evacuation area is shorter than the recent shortest. *AllPossibleRoute* stores all the current route when it reached the evacuation area for the use of reference and give an alternative route.
3. Declare integers: *initial location* as 13, *traced value* as 14, *shortest* and *temporaryShort* as 0, *order* as 1, *possiblePath* as 0 and *flag* as 0. *initial location represents* as the first path in the *route* linked list *current*, basically the desired location to start the simulation. Traced value is increments when multiple paths are encountered it served as an ID in every possible route. Shortest represents the number of nodes in the shortest linked list and temporaryshort represents the count of node in current. Order increments to group the possible path encounter. Relative possible route has the same order value and pushed in the linked list. Possiblepath is a counter during the searching around the current location and flag represents that the evacuation is reached and around the current location.
4. Get the desired location, required values are for the array, 2D requires for the x and y axis and 3D requires 3 inputs. Desired location is now the current location.
5. Search around the current location (8 elements around the current path). Searching manner is clock-wise, starting upper-left, upper element, upper-right, right element, lower-right, lower element, lower-left and the left most element. Every element checks if the current element around the current location has a value greater than or equal to evacuation area. Horizontal and vertical element around the current location has the additional condition of if the currently tested element is not equal to the current location. If the element has slanting position around the current location the additional conditions are: it should not equal to the current location and the nearest element around the current location is not equal to path or block (has an integer value of 4). Every passed element on that condition is entitled to be Pushed in the possible linked list, increment in Possiblepath and increment tempshort.
6. if flag is 1, means next to current is evacuation center; if shortest linked list is null, automatically set the current linked list to shortest then shortestpath is equal to tempshort, else compare current and shortest if current is shorter then replace it with the value in current then save to allpossbleroute else just push current to allpossibleroute. Set again the flag to zero. Then transfer the latest value in possible linked list then return to step number 5 by passing the new value in current location as the lastest value in current linked list.

Hence, if possibepath is 1 and flag is one, the possiblepath is the evacuation area.

Pop the value of the current while current linked list’s order is not equal to possible linked list’s order. Then transfer the latest value of possible to current, then return to step 5 with the latest value of current linked list.

Else if possible linked list is now empty, the simulation would be done.

1. If possible is equal to zero, it means that the current location has no path to select or at a dead end. Pop all values with order greater than or equal on the top most of the possible linked list, then push the top of possible to current then pop possible to remove the value passed to current. If possible linked list is empty and shortest linked list has value, then the simulation is done.

If possible is equal to 1 and flag is zero, it means it is in a straight or single line of path, get the value pushed to possible linked list then push to current linked list then pop possible linked list, after transferring repeat step 5 but the current location is now the value from possible linked list.

Is possible is greater than 1, increment order and replace all recently collected possible path’s order to latest value of order. Then transfer the latest value of possible to current then repeat step 5 with current location value of the latest value of current.

If possible linked list is empty means all path is visited and the simulation is done.