DSA Assignment (Task-3)

Name: Muppa Prabhas Reddy

Roll_no.: CS21B1039

Date of submission: 25-06-22

Problem Title: Cargo shipping System

Implementation of Priority Queue using Linked List

Input: In this program, the details of the person like name, age who is registering for the transportation of their cargo is taken as input. Then the type of cargo either fragile or tough, weight of the cargo and the station code of destination is taken as input from the user. However, the source station is automatically read based on the location.

Sample input: Consider that the current station is 2. Then the input could be:

Name: Prabhas
Type of good: Fragile (F)
weight: 200 kg
destination: 5

} given by user.

source: 2 (automatically taken)

Note: All inputs are in the permissible range of program.

Output: The cargo is dispatched whenever the destination is reached. The program would display the details of all the cargo / goods, including the amount to be paid (amount varies with type of cargo and the journey distance), whose destination is reached. And at the same time, the total number of goods present, available capacity and occupied capacity are displayed at each station. Displaying list is also possible

sample output: Assume that station-5 has been reached. Then the cargo enqueued previously in the sample input would be dispatched.

Before:

Total no. of goods = 1 | Occupied capacity = 200 kg | Available capacity = MAX_LOAD - 200 kg

Details of the dispatched cargo ----

Name : Prabhas

Type of good : Fragile (F)

weight : 200 kg

source : 2

destination : 5

Amout : 1000 (say)

After:

Total no. of goods = 0 | Occupied capacity = 0 kg | Available = MAX_LOAD.

Algorithm for the implementation of priority queue using linked list for the problem:

1. Start
2. Declare the structure of node (Typical node structure)

```
struct queue
{
    char name[];
    int dispatch_no;
    int amount;
    char type;
    int source, destination, gross_wt;
    struct queue* next; // link to next node
};
```

3. Initialize head, front and rear to NULL, after declaring them globally.

```
queue* head = NULL, *front = NULL, * rear = NULL;
```

4. Create header node to store information about the list.
5. Initialize all the int variables in the header node to '0' and string to '\0'.
6. Use head → gross_wt to store occupied capacity.

head → amount = Net amount received.

head → dispatch_no = total no. of goods.

head → next = NULL;

7. Declare local variables name[20] , amount , end, wt, type.

8. Repeat the steps 9 to 18 while station < MAX_STATION. starting from station-1.   // constant time because MAX_STATION is fixed

9. Display station number, total number of goods, occupied and available capacity.   // Constant time

10. Read the choice of user (1-Enqueue / 2-display) 0-exit)

11. if choice = 1,   // constant time
Then Go to step-12.

12. Algorithm to enqueue maintaining priority: (passing from main function)

12.1. if current_station >= MAX_STATION
print enqueue not possible and exit.   Constant time

12.2: Read name , type of cargo , weight (/~~destination~~) from user

12.3. if weight > available_weight
print : overloaded and exit   Constant time
else
read : destination / end

12.4. if end > current_station and end <= MAX_STATION
enqueue ( ) → Go to step 13.   Constant time
else
print : Invalid destination.

13. Enqueue a new node to the queue :

13.1. if (available capacity > MAX_LOAD)

print : Overloaded and exit. // constant time

13.2. Create new node - NEW & assign ptr = head → next

13.3. Allocate memory to NEW and Nullcheck.

13.4. NEW → INFO = DATA (passed from main())

NEW → next = NULL;

head → gross-wt = head → gross-wt + NEW → wt;

constant time

13.5. if front = NULL OR rear = NULL

head → next = temp;

front = rear = temp;

constant time

else

Go to step 13.6

13.6. if (ptr → destination > NEW → destination) then insert as first node.

front = ~~temp~~ NEW;

constant time

else search for location of node whose destination is less than NEW → destination. let ptr be that location.

while (ptr → next != NULL && ptr → next → destination <= NEW).

13.7. if ptr → next = NULL (insert as last node)

ptr → next = temp/NEW;

rear = rear → next; (change rear)

'n' time

13.8 else

NEW → next = ptr → next;

ptr → next = NEW (no need to change rear).

13.9. After insertion into the list create dispatch numbers. Go to step 14, and then stop.

14. Algorithm to generate dispatch numbers :

14.1. Initialize i = 1.

14.2 Set temp = front.

constant time

14.3. Repeat steps 14.4 to 14.6 until temp ≠ NULL;

14.4. temp → dispatch_no = i

kn time

14.5. increment of i (i++)

14.6. Move temp to next node i.e. temp = temp → next.

14.7. stop.

15. if choice = 2

display list → Go to step 16.

16. Algorithm to display the list:

16.1. Set ptr = front.

16.2. if ptr = NULL

print : Empty list and exit.

16.3. Repeat step 16.4 to 16.5 while ptr ≠ NULL.

16.4. print : ptr → INFO

16.5. ptr = ptr → next

16.6. stop.

17. if choice = 0, go to step-18
 else
 ask user to continue or exit (if exit → go to step-18 else → go to step-9.

18. **Algorithm to dispatch cargo on reaching destination:**

18.1. if front → destination = current_station
 Follow step-18.2
 else
 print: No cargo to dispatch at current_station.

Constant time

18.2. Dequeue while front ≠ NULL & front → destination = current_station
 To dequeue, go to step-19. } Average case = 'kn' time

19 **Algorithm to dequeue and print details:**

19.1 if front = NULL
 print: Underflow and exit.

19.2. Set temp = front.

19.3 front = front → next.

19.4 Update header node values.

constant time

19.5 print : temp → INFO // constant time

19.6 Go to step-14, to regenarate dispatch numbers. // 'kn' time

19.7 free temp // constant time

19.8 stop.

List of functions used and their processes:

1. enqueue() – takes the details of user and puts them in the queue maintaing priority based on destination.

2. Create_dispatch() – Generates the dispatch numbers sequentially for a given queue at a particular time.

3. displaylist() – prints the details of persons in the queue.

4. dequeue() – removes particular cargo whose destination is reached and prints it's details.

Dry run for sample test case:

Let us consider that MAX_LOAD = 1000 kg and MAX_STATION = 5.

T- Tough & F- Fragile

1. station-1. (1<5) → enqueue possible

No. of goods = 0, Available = 1000 kg, Occupied = 0 kg

1.1 Enqueue (Prabhas, T, 200, 4) – A

1.2. Enqueue (Mahesh, F, 100, 3) – B

Before 1.2.

No. of goods = 1, Available = 800 kg, Occupied = 200 kg

DISPATCH	NAME	TYPE	WEIGHT	SOURCE	DESTINATION
1	Prabhas	T	200	1	4

After 1.2.

No. of goods = 2, Available = 700 kg. Occupied = 300 kg

Since destination of second insertion is less than first insertion it comes first in queue.

A → destination > B → destination.

DISPATCH	NAME	TYPE	WEIGHT	SOURCE	DESTINATION.
1	Mahesh	F	100	1	3
2	Prabhas	T	200	1	4.

2. station-2. (2<5) → enqueue possible

2.1. No enqueues at station -2 (let).

2.2. So there is no cargo to dispatched at station-2.

2.3. Go to next station.

3. station-3. (3<5) → enqueue possible.

3.1. Enqueue (NTR,T,250, 4) → C

C → destination > B → destination

After 3.1.

No. of goods = 3, Occupied = 550 kg, Available = 450 kg

DISPATCH	NAME	TYPE	WEIGHT	SOURCE	DESTINATION
1	Mahesh	F	100	1	3
2	Prabhas	T	200	1	4
3	NTR	T	250	3	4.

3.2. No more enqueues at station-3.

3.3. B→ destination = current-station = 3.

3.4. dequeue (B).

After 3.4:

No. of goods = 2, Occupied = 450 kg , Available = 550 kg

details of dispatched cargo.

NAME	TYPE	WEIGHT	SOURCE	DESTINATION	AMOUNT
Mahesh	F	100	1	3	1000.

4. station-4. (4<5) → enqueue possible.

4.1. Assume no enqueues at station- 4.

4.2. A→ destination = C→ destination = 4.

4.3. Dequeue (A) and (C).

After 4.3:

No. of goods = 0, Occupied = 0 , Available = 1000.

details of dispatched cargo.

NAME	TYPE	WEIGHT	SOURCE	DESTINATION	AMOUNT
Prabhas	T	200	1	4	2000
NTR	T	250	3	4	500.

5. station -5 (5 ≮ 5) → enqueue not possible

5.1. No cargo to be dispatched at station - 5.

Time and Space Complexities:

① Create_dispatch( ).

This function is basically a traversal of the linked list. Because of the traversal the order of time complexity would be $O(n)$.

Let $f(n) = c_1 n + c_2$ (c_1 & c_2 are constants)

Consider $g(n) = n$

$c_1 n + c_2 \leq c_3 n$. Then there exists n_0 such that this inequality holds for all $n \geq n_0$.

$$c_1 n_0 + c_2 \leq c_3 n_0$$

$$n_0 \geq \frac{c_2}{c_3 - c_1}$$

$\therefore$ $c_1 n + c_2 \leq c_3 n$ for all $n \geq \frac{c_2}{c_3 - c_1}$. So By Big-O notation

Time complexity $= O(g(n)) = O(n)$.

1 Best case : $O(n)$ – Unlike searching, all nodes should be traversed

2. Average case: $O(n)$ – Need to traverse all the nodes.

3. worst case: $O(n)$ – We need to traverse all the nodes

Space complexity: This algorithm requires only a fixed amount of extra space for variables i and temp. It is not called recursively. So space complexity $= O(1)$.

② enqueue( ).

In the enqueue function, we need to traverse the list until finding a node N whose priority is less than NEW.

Here $f(n) = c_1 n + c_2$

$\Rightarrow c_1 n + c_2 \leq c_3 n$. This inequality holds for all $n \geq \frac{c_2}{c_3 - c_1}$

Here $c = c_3$ and $n_0 = \frac{c_2}{c_3 - c_1}$

$\therefore$ Time complexity $= O(g(n)) = O(n)$.

1. Best case : The best case time complexity would be insertion of the node at the front which occurs in constant time.

   $\therefore$ Time complexity (best) $= O(1)$.

2. Average case : $O(n)$ – need to traverse the linked list to find the position of insertion

3. Worst case : $O(n)$ – It is possible that the node we are inserting have the least priority. Then we need to insert at rear end.

Space complexity : This algorithm requires only a constant amount of extra space for variables.– ptr.

$\therefore$ Space complexity $= O(1)$.

③ display_list ( ):

The display_list function involves printing the values of the nodes of linked list, which is basically traversing the list.

$f(n) = c_1 n + c_2 = O(n)$ with the time complexity of traversal function

$\therefore$ Time complexity $= O(n)$.

1. Best case : $O(n)$ – Need to traverse the complete list containing 'n' number of nodes. whatever may be the value of 'n' all nodes should be traversed.

2. Average case : $O(n)$ – Need to traverse all the nodes.

3. Worst case : $O(n)$ – Need to traverse entire list.

Space complexity : This algorithm requires only a constant amount of extra space for the variable –ptr. No recursive calls are made.

$\therefore$ Space complexity $= O(1)$.

④ dequeue ( ):

The pop operation in a priority queue (linked list implementation) requires order of constant time. Because no traversals are made. Only the front node was popped out each time it is called.

$f(n) = c_1 \le c_1(1)$ for all $n \ge 1$.

$\therefore$ $c = O(1)$ with $c = c_1$ and $n_0 = 1$

$\therefore$ The tight upper bound for $f(n) = c$, is $g(n) = 1$

Time complexity $= O(g(n)) = O(1)$.

1 Best case: $O(1)$ – Direct access to the front element

2 Average case: $O(1)$ – No traversal is required.

3. Worst case: $O(1)$ – No traversal required. Direct access to front of queue

Note: Here basically all the 3 cases (best, average and worst) are equivalent.

Space complexity: This algorithm requires only a fixed amount of extra space. No recursive calls are made.

$\therefore$ space complexity $= O(1)$.

Time and Space complexities for entire algorithm:

1. We analyze algorithms only at larger values of 'n'. i.e. Asymptotic Time complexity. What this means is, below n_0 we do not care for rate of growth. This program is a combination of time complexities – $O(1)$ and $O(n)$. So the tight upper bound would be $O(n)$. (rate of growth in increasing order: $O(n) > O(1)$).

$$\boxed{\text{Time Complexity} = O(n)}$$

2. This program requires constant extra space. No recursion call is made which could use extra space.

$$\boxed{\text{Space Complexity} = O(1)}$$