ANSWERS

A3.4 Linked lists

Data structures · A level · OCR H446 1.4.2, AQA 7517 4.2.1.4, Eduqas A500QS 1.1 · about 25 min

BugBotLab

What this lesson is about

Nodes and pointers, the free list, and traversing, inserting and deleting: a route of waypoints.

- Nodes and pointers
- A linked list in arrays
- Traversing
- Inserting
- Deleting
- Arrays or linked lists?

Questions

6 marks in all

1. What does each node in a linked list contain? [1 mark]

- ☐ **A** Its data and a pointer to the next node
- ☐ **B** Only its data
- ☐ **C** Its data and its index in an array
- ☐ **D** A key and a hash value

> **Answer:** A. The pointer is what gives the list its order; the last node's pointer is null.

2. The linked list is stored in two arrays. What does this program print? [1 mark]

```
data = ["C", "A", "D", "B"]
nxt = [2, 3, -1, 0]
start = 1
cur = start
out = ""
while cur != -1:
    out = out + data[cur]
    cur = nxt[cur]
print(out)
```

> **Answer:**
>
> ```
> ABCD
> ```
>
> From index 1 (A) the pointers go to 3 (B), 0 (C), 2 (D) and then -1, the end.

3. Put the steps for inserting a new node straight after node P in order. [1 mark]

Number the lines 1 to 4 to put them in the right order.

___ `Set P's pointer to the new node`

___ `Set the new node's pointer to P's pointer`

___ `Take the first node from the free list`

___ `Store the data in the new node`

Answer:

```
Take the first node from the free list
Store the data in the new node
Set the new node's pointer to P's pointer
Set P's pointer to the new node
```

The new node must point at the rest of the list before P is changed, or the rest of the list is lost.

4. Which is a disadvantage of a linked list compared with an array? [1 mark]

- ☐ **A** Item number i can only be reached by following pointers from the start
- ☐ **B** Inserting an item in the middle means moving every later item
- ☐ **C** It has a fixed size set when it is created
- ☐ **D** Its items must all be stored next to each other in memory

Answer: A. There is no direct access, so reaching an item or searching is linear. The other statements describe arrays.

5. In the lesson's example, data = [17, 5, 23, empty, empty], start = 1 and free = 3. 20 is inserted in order. At which index is 20 stored? [1 mark]

Answer: 3. A new node always takes the slot at the front of the free list, which is index 3.

6. When a node is deleted from an array-based linked list, why is its index put on the free list? [1 mark]

- ☐ **A** So its slot can be reused by a later insertion
- ☐ **B** So the data in it is wiped
- ☐ **C** So the list stays sorted
- ☐ **D** So the start pointer is updated

Answer: A. Bypassing the node removes it from the list; returning it to the free list makes its space available again.

The task: the linked route

BugBot's route is a linked list stored in the arrays `names`, `xs`, `ys` and `nxt` (coordinates in cm from where the robot starts, `-1` is null), with `start` and `free` pointers. At the moment the route is A → B → E → C, but waypoint E is closed. Change only pointers and array elements; do not use the list methods `insert`, `remove`, `pop` or `del`. 1. Write `delete(name)`: unlink the node called `name` and put its slot on the **front** of the free list. 2. Write `insert_after(before, name, x, y)`: take the slot at the front of the free list, store `name`, `x` and `y` in it, link it in straight after the node called `before`, and return the slot's index. 3. Delete `"E"`, then insert `"D"` at (40, 60) after `"B"`, and print `D stored at index <index>`. 4. Traverse the list and print the names in order, separated by single spaces: `route: A B D C`. 5. Traverse again, calling `go_to(xs[i], ys[i])` for each node, so the robot drives the route without passing through E. Both functions change `start` or `free`, so declare them `global` inside the functions.

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

def go_to(x, y):
    """Drive to (x, y) cm from the start: across first, then up or down."""
    px, py = position()
    dx, dy = x - px, y - py
    if dx > 1:
        right(50, distance=dx)
    elif dx < -1:
        left(50, distance=-dx)
    if dy > 1:
        forward(50, distance=dy)
    elif dy < -1:
        backward(50, distance=-dy)

# the route as a linked list held in arrays; -1 is the null pointer
names = ["C", "B", "A", "E", "", ""]
xs    = [0,   40,  0,   20,  0,  0]
ys    = [60,  20,  20,  40,  0,  0]
nxt   = [-1,  3,   1,   0,   5,  -1]
start = 2
free = 4

def delete(name):
    pass

def insert_after(before, name, x, y):
    pass
```

The hint students can ask for: To delete, find the node before E and point it past E, then put E's slot on the front of the free list. To insert, take the slot at the front of the free list, fill it, point it at whatever B pointed to, then point B at it. Traverse by following next pointers from start until you reach -1.

A solution

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

def go_to(x, y):
    """Drive to (x, y) cm from the start: across first, then up or down."""
```

```
    px, py = position()
    dx, dy = x - px, y - py
    if dx > 1:
        right(50, distance=dx)
    elif dx < -1:
        left(50, distance=-dx)
    if dy > 1:
        forward(50, distance=dy)
    elif dy < -1:
        backward(50, distance=-dy)

# the route as a linked list held in arrays; -1 is the null pointer
names = ["C", "B", "A", "E", "", ""]
xs    = [0,   40,  0,   20,  0,  0]
ys    = [60,  20,  20,  40,  0,  0]
nxt   = [-1,  3,   1,   0,   5,  -1]
start = 2
free = 4

def find(name):
    """Index of the node called name, and the index of the node before it (-1 if it is
first)."""
    prev = -1
    cur = start
    while cur != -1 and names[cur] != name:
        prev = cur
        cur = nxt[cur]
    return cur, prev

def delete(name):
    global start, free
    cur, prev = find(name)
    if cur == -1:
        return
    if prev == -1:
        start = nxt[cur]
    else:
        nxt[prev] = nxt[cur]
    nxt[cur] = free
    free = cur

def insert_after(before, name, x, y):
    global free
    if free == -1:
        print("no space")
        return -1
    new = free
    free = nxt[free]
    names[new], xs[new], ys[new] = name, x, y
    b, _ = find(before)
    nxt[new] = nxt[b]
    nxt[b] = new
    return new

delete("E")
slot = insert_after("B", "D", 40, 60)
print("D stored at index", slot)

route = []
```

```
cur = start
while cur != -1:
    route.append(names[cur])
    cur = nxt[cur]
print("route:", " ".join(route))

cur = start
while cur != -1:
    go_to(xs[cur], ys[cur])
    cur = nxt[cur]
```

Any program that meets the task's checks is marked correct in the simulator; this is one way, not the only way.