WORKSHEET

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

Name _____ Class _____ Date _____

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

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)
```

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`

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

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]

---

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

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
```

Plan your program here, then type it in and press Run.

Do it on the robot

www.bugbotlab.com/learn/a3-4-linked-lists/

The simulator checks it and tells you when it passes. Nothing to install, no account.

Challenges

1. Draw the four arrays and both pointers after your program has run. Check them by printing the arrays.
2. A **doubly linked list** also stores a pointer to the previous node. What does that make easier, and what does it cost?
3. What should `insert_after` do if the free list is empty? And if there is no node called `before`?