PA – Redux: Boustrophedonic Lists

Due Date

- See Piazza for due date and time
  - Grading the next day
- Submit program to perforce in your student directory
  - Sub directory called:
    - /PA_Redux/...
  - Fill out your ***PA_Redux Submission Report.pdf***
    - Place it in the same directory as your solution
    - Enter the final Changelist number of your submission
    - Write up a quick discussion in the report
      - What you learned from this assignment so far

Goals

- Programming Assessment
  - Extra credit for PA1 - More practice with linked lists... (kind of)

Assignments

- Write a single function: remove()
  - Signature and structure need exactly the same
  - Place function in file named (see supplied files):
    - Boustrophedonic.cpp:
      - Function
    - Boustrophedonic.h:
      - Declaration and structure

```
struct Node
{
    Node *pNorth;
    Node *pSouth;
    Node *pEast;
    Node *pWest;
}

void remove( Node *head, int row, int col);
```

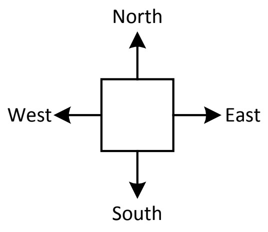
- Assumptions
  - Boustrophedonic list is complete and pristine before you remove one node
  - Only one node will be removed in the function
  - The head node will not be removed
  - The dimensions of the list are not given
  - Boustrophedonic list has an even number of columns
  - Boustrophedonic list has no restrictions on number of rows
  - On deletion, horizontal and vertical connections are preserved across the deleted node

Creating a Boustrophedonic List

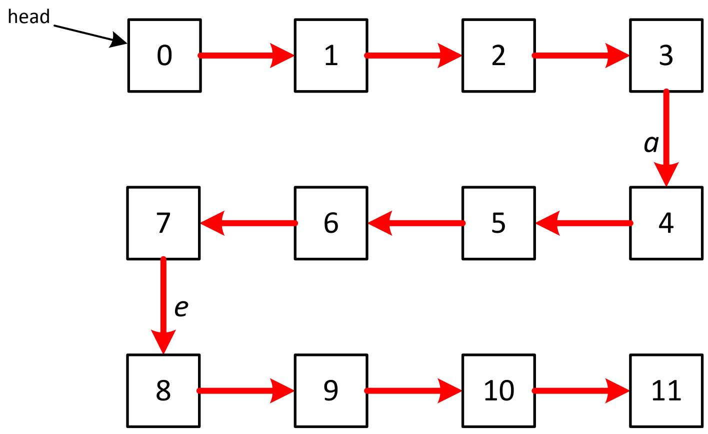
Boustrophedonic - Greek word:

- A method of writing shown in early Greek inscriptions, in which the lines run alternately from right to left and from left to right, as the furrows made in plowing a field, the plow passing alternately backward and forward.

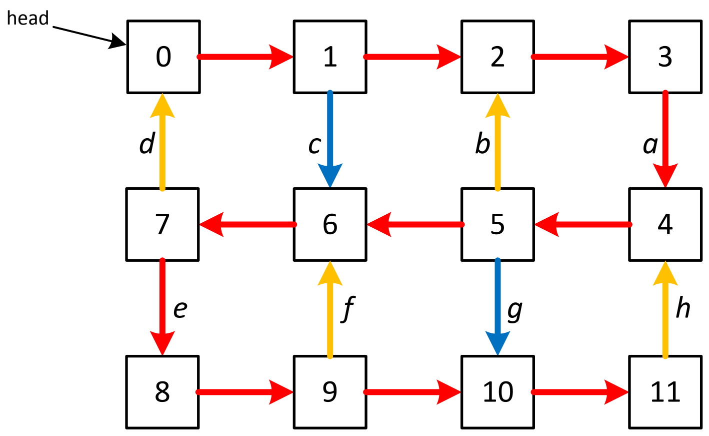
```
struct Node
{
    Node *pNorth;
    Node *pSouth;
    Node *pEast;
    Node *pWest;
}
```



- Start with the first node and continue in the Boustrophedonic path way, follow the RED LINE.
    - Start with the head, and sequentially go from 0 to 11 in the Boustrophedonic way

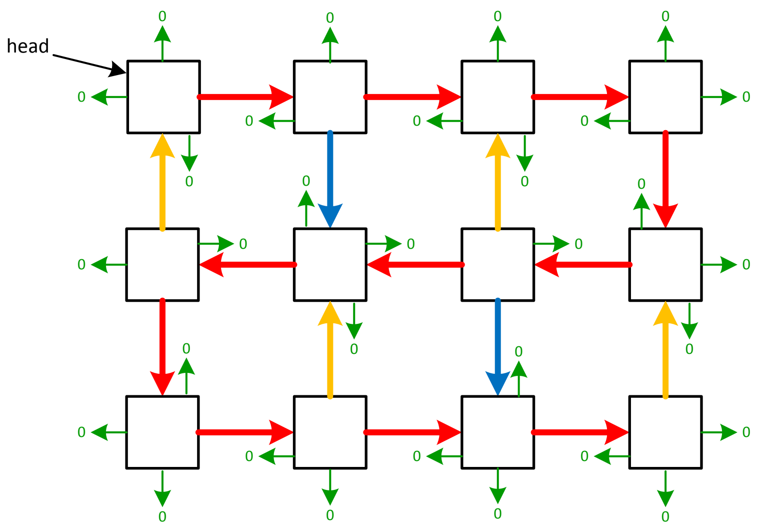


- Next we create the vertical connections.

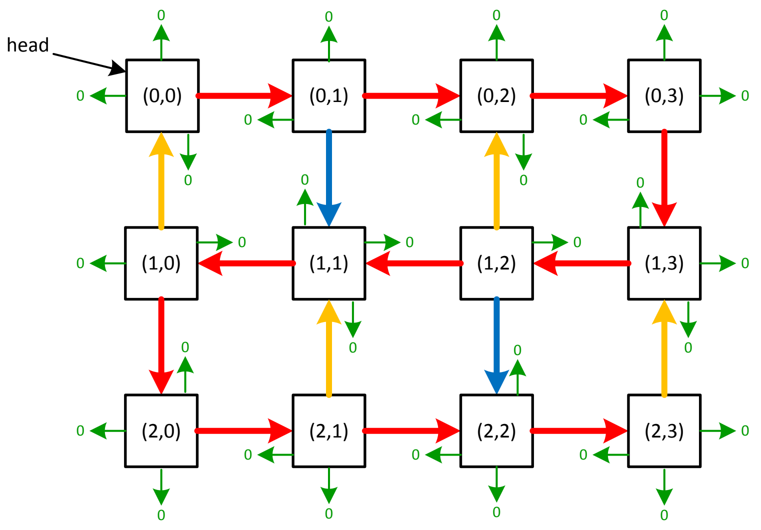


- Create the vertical connections between row 0 {0,1,2,4} and row 1 {7,6,5,4}.
    - Find arrow _a_ between 3 and 4.
        - That arrow goes down, from 3 to 4.
    - Alternate direction of _a_, (_a_ goes down in RED)
        - _b_ goes up YELLOW,
        - _c_ goes down BLUE,
        - _d_ goes up YELLOW.
- Similar for the connections between row 1 {7,6,5,4} and row 2 {8,9,10,11}
    - Alternate direction of _e_, (_e_ goes down in RED)
        - _f_ goes up YELLOW,
        - _g_ goes down BLUE,
        - _h_ goes up YELLOW.

- Drawing the list with all the links displayed.
  - Each node actually has 4 pointers, most are nulls.
    - North, South, East, West
  - Here the list is drawn with GREEN links showing the explicit null pointers.

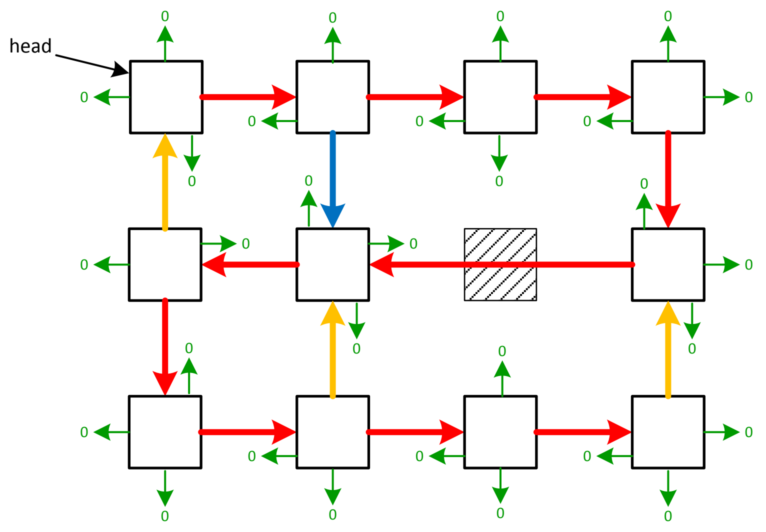


- Rows and Columns
  - Rows and columns are not provided, nor are they added in the Node structure.
  - You need to calculate these as you walk the list.
  - The lists will vary in dimensions, Row x Column (row,col)
  - There will be an even number of columns.

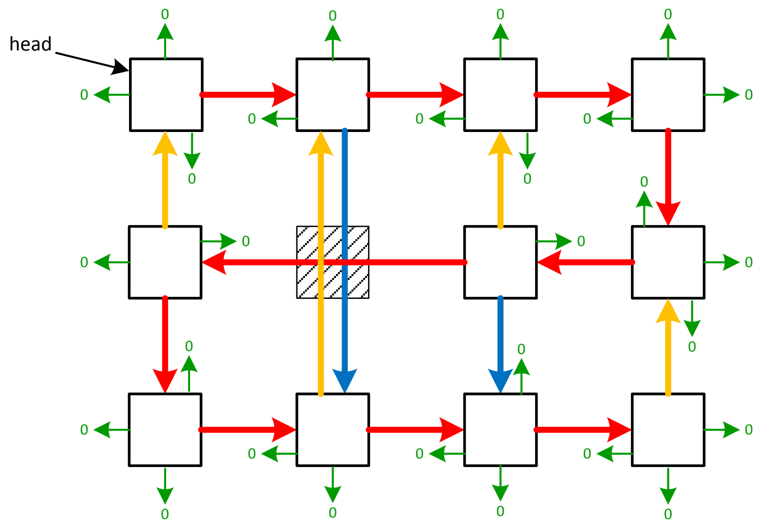


- Deletion
  - Remove only one node at a time
  - Keep horizontal connections, through the deleted node
    - Fix adjacent nodes connections
  - Keep vertical connections, through the delete node
    - Fix adjacent nodes connections
  - Many examples to follow

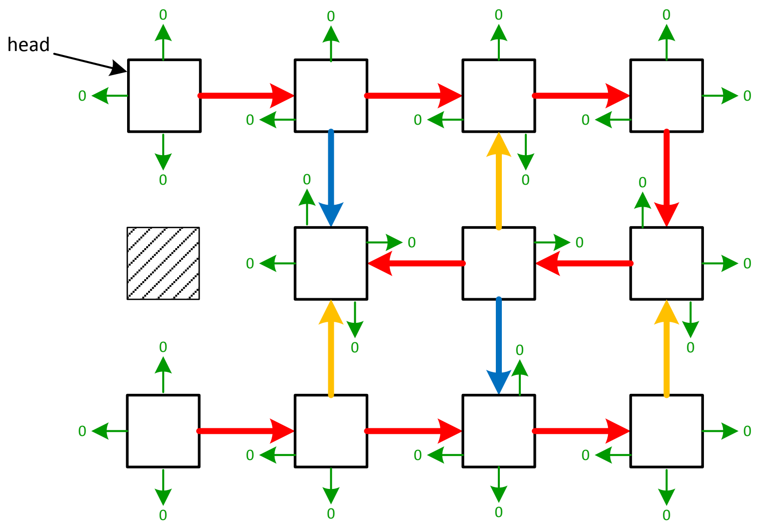
Delete Node (1,2)



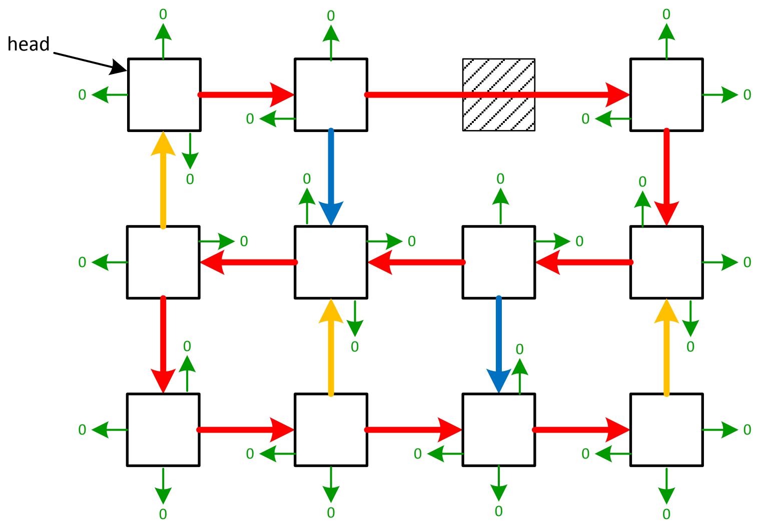
Delete Node (1,1)



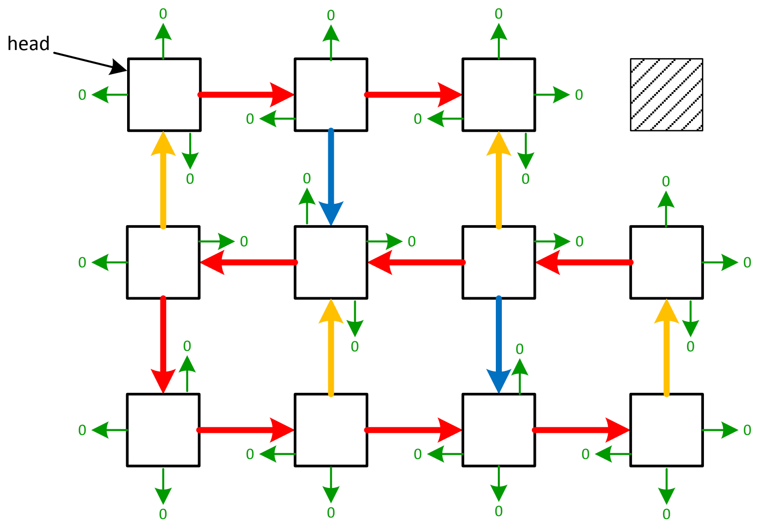
Delete Node (1,0)



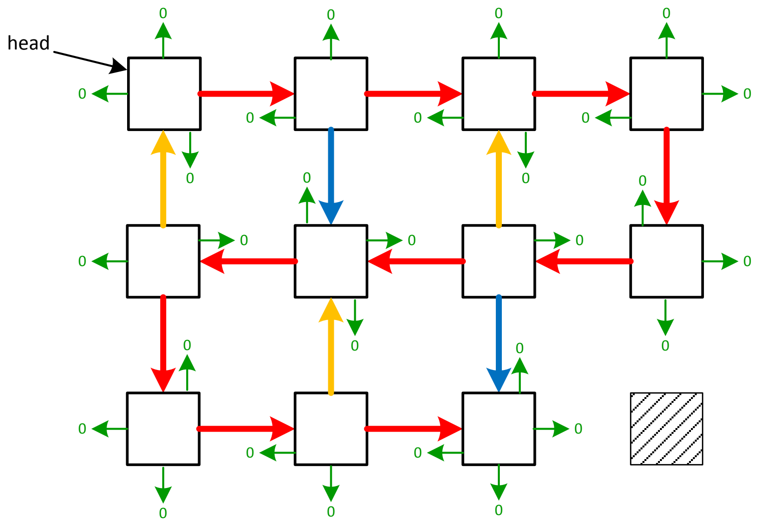
Delete Node (0,2)



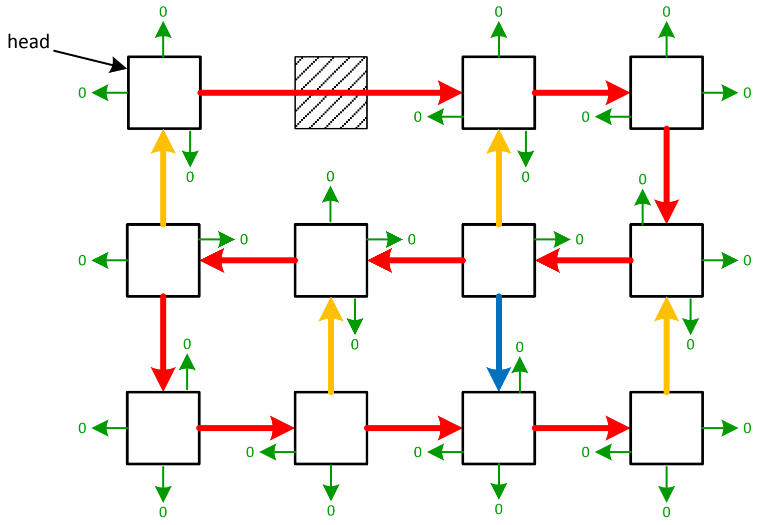
Delete Node (0,3)



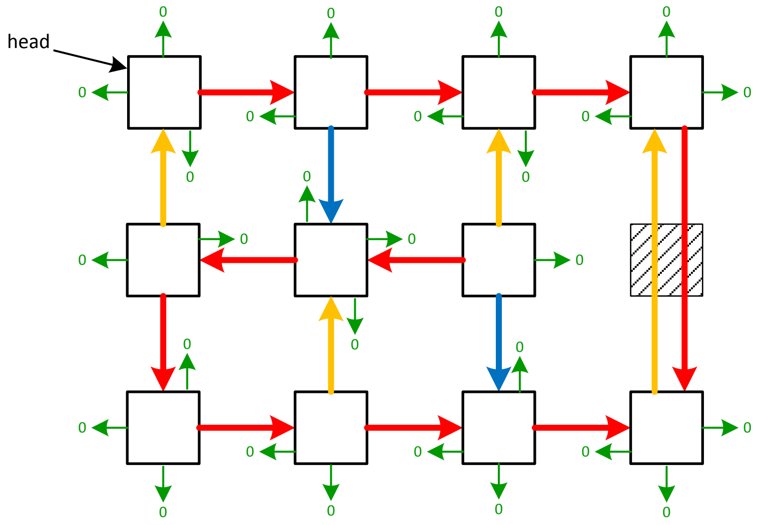
Delete Node (2,3)



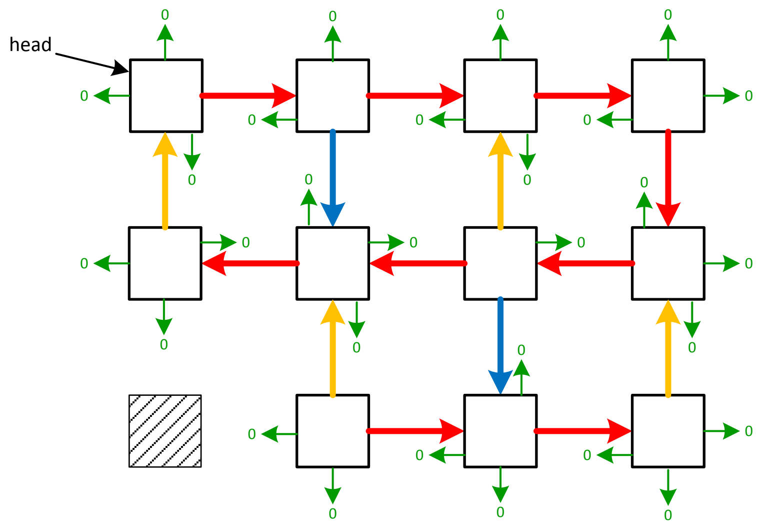
Delete Node (0,1)



Delete Node (1,3)



Delete Node (2,0)



General:

- Write all programs in cross-platform C or C++.
    - Optimize for execution speed and robustness.
- Create a programming file for each problem, for example
    - Student directory
        - /PA_Redux/…
    - Make sure that each problem can be compiled and run through the checked in solution
- Do all your work by yourself
    - Feel free to talk with others about setup, version control, ideas
    - But do not copy your friend's code.
        - Please don't - I can tell with my difference tools
    - Feel free to share ideas
- Check in the problems multiple times, at least 3 times per problem
    - Have reasonable check-in comments
    - Seriously, I'm checking
- Make sure that your program compiles and runs
    - Warning level 4, some times that is not possible due to MS headers…
    - Your code should be squeaky clean.
- We are using Perforce
    - You should have received the document describing how to login.
        - Please look at the documentation and videos under the reference directory
    - Submit program to perforce in your student directory
        - Sub directory called: /PA_Redux/…
            - As described above
    - All your code must compile from perforce with no modifications.
        - Otherwise it's a 0, no exceptions
    - Only Visual Studio 2013 allowed

Validation

Simple check list to make sure that everything is checked in correctly

- Do they compile and run without any errors?
- Warning level 4 free?
- Submitted it into /PA_Redux directory?
- Can you delete you local drive, regrab the PA_Redux directory?
    - Is all the code there?

Hints

Most assignments will have hints in a section like this.

- Do many little check-ins
    - Iteration is easy and it helps.
    - Perforce is good at it.
- Think about performance, did you read the Code Complete chapters?
    - A lot of good ideas in there.