Lab Demo 05

Friday, 05 2018

PS2 Debrief

- Common mistakes:
- TLE C/D: insert/delete in O(N) – use wrong data structure (e.g. vector/stack)
- Some used ‘advance’ method (which is O(N) for list) , should have TLE’d (but some got AC =O) e.g a[aaaaaa…<<<…
- TLE D: implemented SLL instead of DLL, unable to insert/delete in O(1) time (cannot get previous element)
- WA C/D: never keep track iterator/pointer properly
- RTE C/D: iterator out of range (e.g. never get return value of ‘it.erase()’), NULL pointer exception…

PS2 Debrief

- ‘Simulate’ a type writer, insert or delete characters at where the cursor is pointing, where special keys pressed indicate different commands, print result
- A: no ‘[‘ command, simple LIFO logic: std::stack
- B: ‘[‘ command exists, can insert in middle/front in O(N) time (e.g. using std::vector)
- C: need to insert/delete in O(1) time: std::list!

  - use an iterator to keep track of current cursor position

  - iterator always moving by 1 position in each operation (total

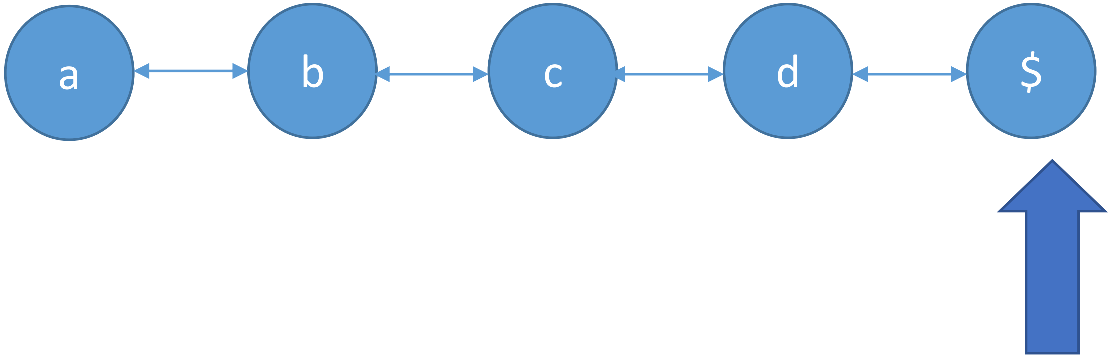
    movement O(N))

  - delete/insert at current iterator position (total insertion/deletion

    O(N))
- Can use 2 deques as well!
- D: implement own list (must be doubly linked list) or some other data structure!

PS2 Debrief

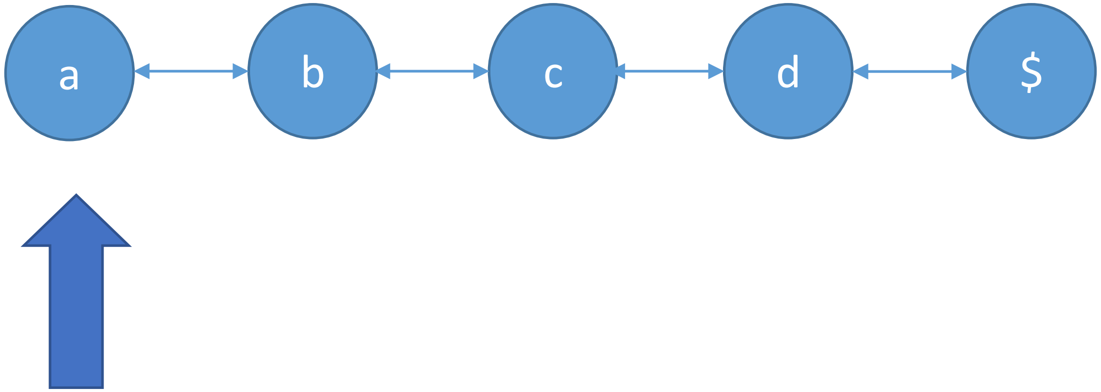
abcd [ef<]<



Iterator (list.end())

PS2 Debrief

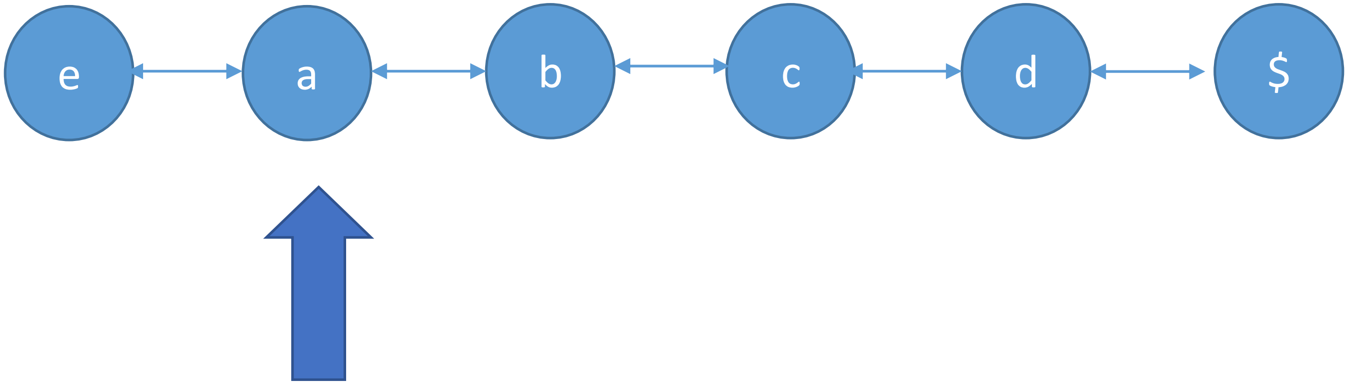
abcd**[** ef<]<



Iterator (list.begin())

PS2 Debrief

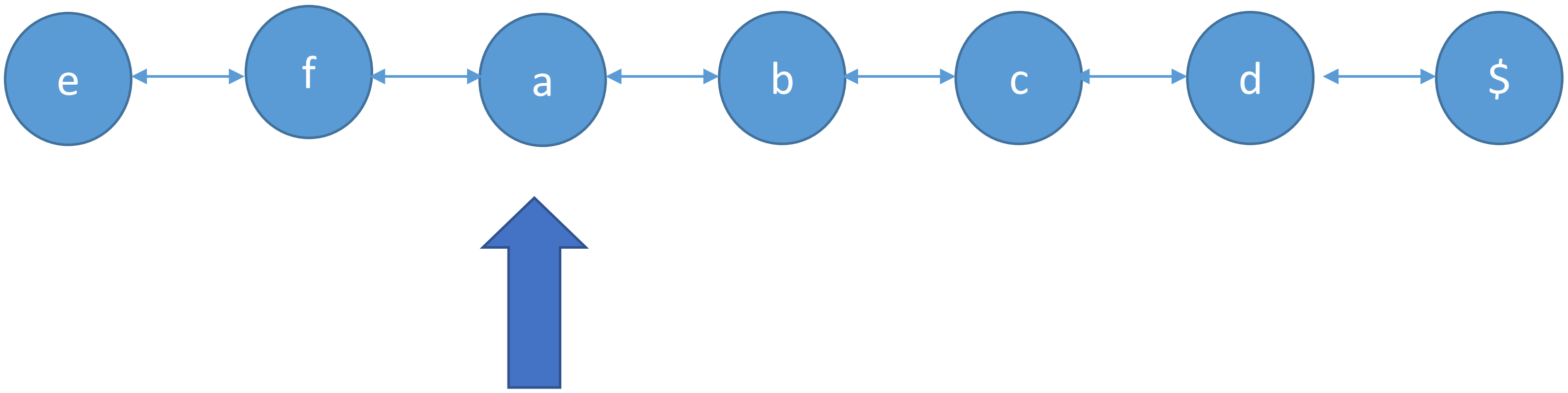
abcd[e f<]<



Iterator (still pointing to same element)!

PS2 Debrief

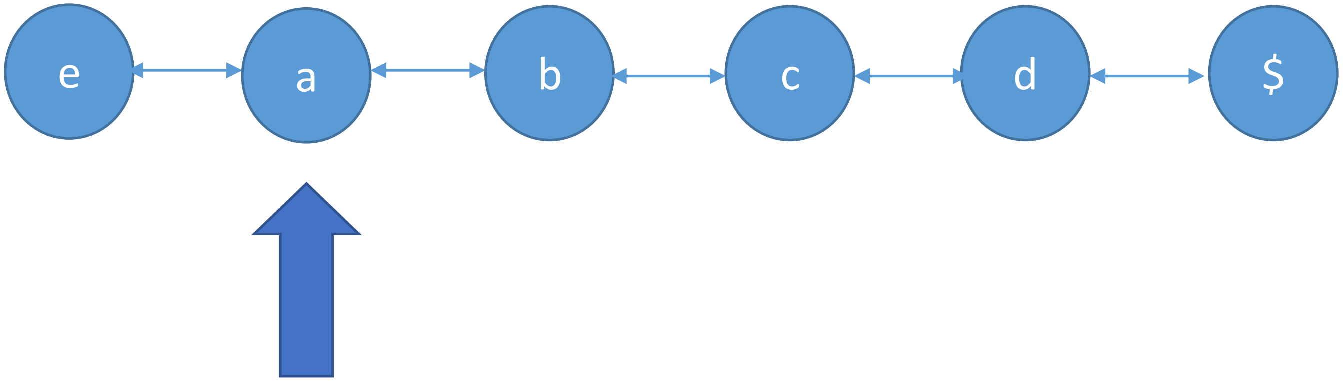
abcd[**ef** <]<



Iterator (still pointing to same element)!

PS2 Debrief

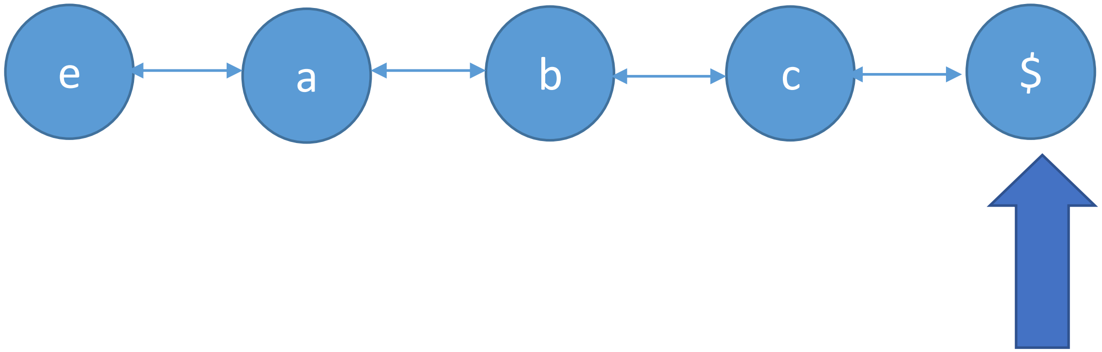
abcd[ef**<**]<



Iterator (still pointing to same element)!

PS2 Debrief

abcd[ef<**]<**



Iterator (still pointing to “list.end()” after deletion)

C++ STL priority_queue demo

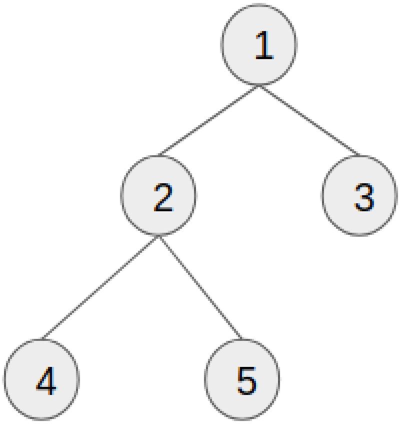
- constructor
- top, push, pop
- http://en.cppreference.com/w/cpp/container/priority_queue
- It is a MAX priority queue by default

Conversion to Min Priority Queue

- C++ <priority_queue> which is a Max PQ can be converted into a Min PQ in two ways
  - Number-only technique
  - Changing the comparison function

Tree

- Create a complete binary tree in C++
- Insert a set of integers in the tree from left to right
- Input: 1 2 3 4 5
- Tree:



Tree

- Print tree elements in pre-order, in-order and post-order fashion
- Find the height of the tree, size of the tree
- Find the number of leaf nodes in the tree

PS3 Status (as of today)

Name	A	B	C	D (ignored)
Group A: 1 + 2	AC	AC	AC	Ignored
Group B: 4 + 2	AC	AC		
Group C: 1 + 0	AC			
The rest of you… Group D: 13 + 20 :O				

~8 more days to complete PS3

Hacking Solution for PS3 Subtask I

UpdateDilation() and **GiveBirth()** can make things difficult

- But in Subtask I, 1 ≤ **n** ≤ 15
- You can just use an array of size 15 and keep **re-sorting** the positions of up to 15 women for every **ArriveAtHospital()**, **UpdateDilation(),** and **GiveBirth()** operations that can change the ordering
- This way, if done correctly, can give you “free” 1 point
- This is not a “proper PQ” solution though and only uses sorting knowledge that we have learned in the first half of CS2040C :O…
- But this is a solution that you should write if you have nothing else for the harder subtasks, e.g. during individual tests… ☹
  - At least non zero

Easiest Solution for PS3 Subtask II

It is a classic PQ example! Read the wording carefully!

Easiest solution: Just use C++ STL priority_queue!

- Implement a “woman object”
  - Important note: Real life woman is NOT an object!
    - PS: Some senior students name this variable to “mommy” ☺
- Or, we can just use pair or tuple from earlier
  - pair<int, int> woman, first field is dilation, second field is arrival index
    - We can negate the second field :O
- DONE, ArriveAtHospital==push, GiveBirth==pop, Query==top

PS: Other solutions exist, like the one in Tut07 later!

Why PS3 Subtask III is Harder?

Why it is not easy for C++ priority_queue to handle **UpdateDilation()** operation efficiently, i.e. **faster than O(n)**?

- This requires ability to modify a key inside the Priority Queue (likely Binary Heap) where this key can be anywhere in the Binary Heap (not necessarily in the root – the easiest place)
- This operation is sometimes called as **heapUpdateKey(i, v)**
- To do this efficiently, we need something that is hidden in VisuAlgo

Note, the **GiveBirth()** operation is also more complex now

- It may involve deletion of a key that is not necessarily the current maximum of the Binary Heap :O

heapUpdateKey(i, v)

To update the value of a key **i** to a new value **v** (where **i** is not necessarily the root---index **1**), we need:

1. A way to fix (Max) Heap property as changing the previous value to a new value **v** may cause violation of (Max) Heap property

   Hint: Anything to consider?

2. A way to quickly identify this index **i**

   Hint: Something that you learn yesterday? (Thu of Week 07), see next slide

C++ STL unordered_map **SHORT**demo

unordered_set is similar

- constructor
- insert, operator [], find, count
- range-based for loop to access the keys (in unordered fashion)
- erase
- empty, size
- http://en.cppreference.com/w/cpp/container/unordered_map

GiveBirth(i)

To delete key **i** (where **i** is not necessarily the root---index **1**), we just need:

```
heapUpdateKey(i, INF) // i will be at the root now
ExtractMax() // then ☺
```

Of course you still need a fast way to map a woman name to her index **i** quickly, the same thing that we discussed earlier

PS: Other ways exists

PS3…

All the best in clearing PS3, if you have not done so

- Subtask IV requires you to avoid using STL :O…
  - AC-ing it shows Steven that you really understand Binary Heap and Hash Table concepts (or some other concept :O)…
  - 0 point, totally optional this time

Remember: If you keep delaying **your first attempt** for PS3, you may run out of time even though you have ~8 days working time for PS3