CSC 3100: Data Structures

Final Exam (close book)

Time: 08:30am - 10:30am (120 mins), May 19, 2022

2) 由 $\frac{1}{\sqrt{n}} > \frac{1}{n}$

1. [10 marks] State and prove whether the following statements are correct or not:

(1) [5 marks] $n^{1/2} + logn = O(n^{1/2})$ ✓

(2) [5 marks] $log_{10}(2^n) = \Theta(n)$ ✓

懒

2. [10 marks] Suppose the nodes of a doubly linked list structure are defined as follows:

```
Public class Node {
     public int data;
     public Node next, prev;
};
```

Design an algorithm with pseudocodes which **concatenates** two given lists (the first node of the second list will follow the last node of the first list) and returns the new list. Note that it does not create new nodes; it just rearranges the links of some existing nodes.

大概就是找到 L1 的最后一个元素，然后，next 指向 L2.head.next. L2.head.next 的.prev 指向 先要判断 L2.head 有无 next.

要从最后一位开始塞格子，然后从小到大往十位塞，再往百位

3. [10 marks] RadixSort (*this problem was selected from lecture notes*):

(1) [7 marks] Given a sequence of integer values: [170, 045, 075, 090, 002, 024, 802, 066], show how to use RadixSort to sort it in the **ascending** order.

(2) [3 marks] In RadixSort, can we start from the **most** significant digit? If yes, explain why; if no, give a counterexample.

Yes.

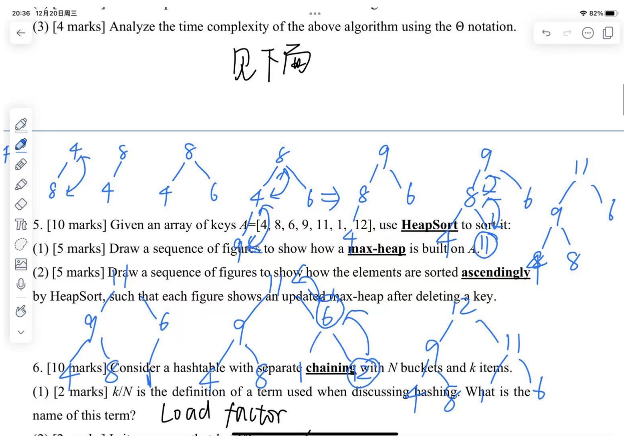
4. [10 marks] Given the **inorder sequence** X and **postorder sequence** Y of traversing a binary tree, reconstruct the binary tree such that its **inorder** and **postorder** are X and Y respectively. Here, X and Y are arrays with elements $X[1], X[2], \ldots, X[n]$ and $Y[1], Y[2], \ldots, Y[n]$ respectively, where $n \geq 1$.

(1) [2 marks] Briefly discuss the main idea of reconstruction.

(2) [4 marks] Write the pseudocodes for the reconstruction algorithm.

(3) [4 marks] Analyze the time complexity of the above algorithm using the Θ notation.

见下面



20:36 12月20日周三

(3) [4 marks] Analyze the time complexity of the above algorithm using the Θ notation.

见下面

5. [10 marks] Given an array of keys A=[4, 8, 6, 9, 11, 1, 12], use **HeapSort** to sort it:

(1) [5 marks] Draw a sequence of figures to show how a **max-heap** is built on A.

(2) [5 marks] Draw a sequence of figures to show how the elements are sorted **ascendingly** by HeapSort, such that each figure shows an updated max-heap after deleting a key.

6. [10 marks] Consider a hashtable with separate **chaining** with N buckets and k items.

(1) [2 marks] k/N is the definition of a term used when discussing hashing. What is the name of this term? Load factor

s A=[4, 8, 6, 9, 11, 1, 12], use **HeapSort** to sort it:

gures to show how a **max-heap** is built on A.

gures to show how the elements are sorted **ascendingly**

hows an updated max-heap after deleting a key.

?PT，注意是 max-heap

6. [10 marks] Consider a hashtable with separate **chaining** with N buckets and k items.

(1) [2 marks] k/N is the definition of a term used when discussing hashing. What is the name of this term? Load factor

(2) [2 marks] Is it necessary that $k < N$? X

(3) [3 marks] In the worst-case, how many items could be in a single bucket? k

(4) [3 marks] If we resize the table to a table of size $2N$, what is the asymptotic running time in terms of k and N to put all the items in the new table?

N+k

7. [10 marks] Given a binary search tree with root node T, find which value is the **median** value, and delete that value. Assume that for each node p, we can access its left child and right child by *p.leftChild* and *p.rightChild* respectively, and access its key value by *p.key*.

(a) [5 marks] Design an algorithm to solve the above problem and show its main steps.

(b) [5 marks] Use O notation to analyze the time complexity of your algorithm. Your bound should be as tight as possible (as if you are using Θ notation).

(a) 中序遍历，找到．再遍历一次 执行删除．

O(n) O(n) O(n) O(n)

8. [10 marks] Consider an undirected graph with n nodes and m edges ($m \geq n$), and its adjacent list, where for each node v, its adjacent list is denoted by *Adj(v)*.

(1) [3 marks] What is the time complexity to count the **total number of edges**? Use O notation and the bound should be very tight (as if you are using Θ notation).

(2) [4 marks] Assume that the size of the adjacent list of each node v, denoted by *d(v)*, is known in advance, can we **sort all the nodes according to their degrees** in an ascending order in *O(n)* time cost? If yes, briefly show the idea; if no, please explain why.

(1) n 求和

Θ(1)

n 求和．BFS | DFS Θ(n)

(2) count sort

(3) [3 marks] What is the big-O time cost of **removing a node *v* from the graph**? Notice that removing a node means that all the edges linked to this node will be removed as well, and the bound should be very tight (as if you are using Θ notation).

① 找到 node 的 adj. list (n)

② For each adj. 去到他的 adj. 并删去他们的 node $n(n^2)$

③ 返回 node 删去 adj list $O(n^2)$

9. [10 marks] Consider an undirected graph G_1, which is depicted as below:

(1) [6 marks] Show the sequences of nodes of **BFS traversal** and **DFS traversal** respectively, by assuming that the starting node is set to v_0.

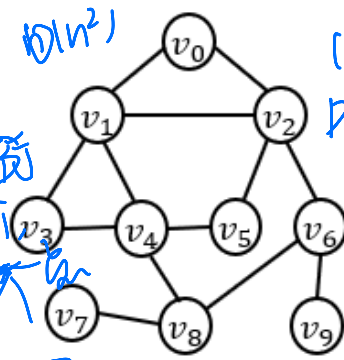
(2) [4 marks] Compare BFS with DFS in terms of time cost and extra space cost, if we use the **adjacent matrix** to represent the graph.

$O(n^2)$?

BFS. 每行遍历 $O(n^2)$

DFS. 跳到 j 行 然后找一个新

$O(V^2)$?



Graph G_1

(1) $V_0\ V_1\ V_3\ V_4\ 5\ 2\ 6$ DFS $8\ 7\ 9$

BFS 0 1 2 3 4 5 6 8 9 7

(2)

10. [10 marks] You are given an edge-weighted undirected graph, using the adjacency list representation, together with the list of edges in its minimum spanning tree (MST). Describe an efficient algorithm for updating the MST, when each of the following operations is performed on the graph. Assume that common graph operations (e.g., DFS, BFS, finding a cycle, etc.) are available to you, and don't describe how to re-implement them.

(1) [5 marks] Update the MST when the weight of an edge that **was** part of the MST is **increased**. Show the main ideas of your algorithm and give the order-of-growth running time of your algorithm as a function of *V* and/or *E*.

(2) [5 marks] Update the MST when the weight of an edge that **was not** part of the MST is **decreased**. Show the main ideas of your algorithm and give the order-of-growth running time of your algorithm as a function of *V* and/or *E*.

→ $O(E)$

→ $O(n)$

(1) 删去所有增加的边.

(2)

再用 DFS 对不同组的点进行分组, 再用 ku 算法. → $O(E\log E)$

4

Inorder

In: 4 2 5 1 6 3 7

post: 4 5 2 6 7 3 1

对于每个 tree / subtree.
root 总在 postorder 的最后一个
在 IO 找到他，左边的即左子树
右边即是右子树.

RC(x, y, ex)

If len(x) = 1: $\Theta(1)$
  return $\Theta(1)$

看不了看是最后一个值 y[-1] 吗? 递归，

else:
  root = y[len(y)] $\Theta(1)$
  l_tree_len = x.find(root) $\Theta(n)$
  r_tree_len = len(x) − l_tree_len − 1 $\Theta(1)$
  root.left = ~~x~~[~~find~~ l_treen_len] $\Theta(1)$
  root.right = ~~x~~[n−1] $\Theta(1)$
  RC(X[1, l_tree_len], y[1, l_tree_len]) T(l_tree_len)
  RC(X[len_tree_left+1, −1], y[len_tree_left, −2]) T(len_tree_right)

In: 4 2 6 5 7 1

$T(n) = O(n) + T(n-m) + T(m)$

Worst case. m=1 forever
$T(n) = O(n^2)$

Avg. case = $O(n) + 2T(\frac{1}{2}n)$
$= O(n \log n)$

Avg = $O(n \log(n))$

d=1 b=1

$T(n) = aT(\frac{n}{b}) + O(n^d)$

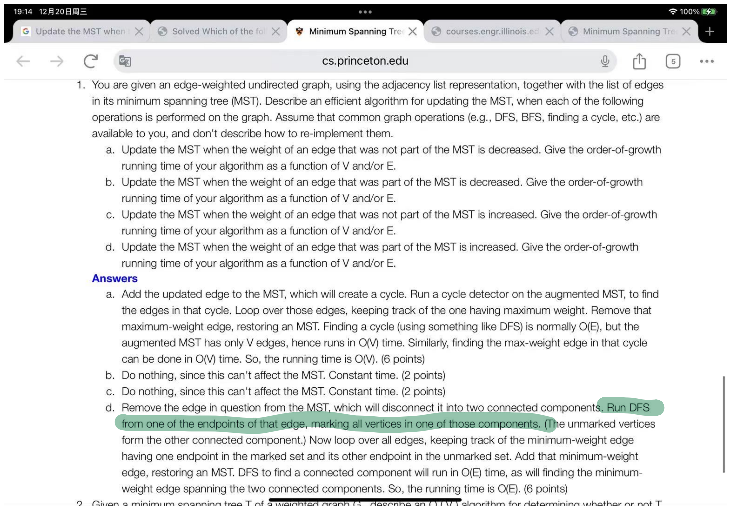
1° $a > b^d \rightarrow \log_b a$

2° $a = b^d \rightarrow n^d \log n$

$3^{\circ}\quad a<b^{d} \rightarrow n^{d}$

1. 你得到一个边权重的无向图，使用邻接表表示法，以及边的列表形式表示其最小生成树（MST）。描述一个高效的算法来更新MST，当下列任一操作在图上执行时。假设常见的图操作（如DFS, BFS, 寻找循环等）对你来说是可用的，并且不用描述如何重新实现它们。
   a. 当MST中不包含的边的权重降低时，更新MST。给出你的算法的增长阶的运行时间，作为V和/或E的函数。
   b. 当MST中包含的边的权重降低时，更新MST。给出你的算法的增长阶的运行时间，作为V和/或E的函数。
   c. 当MST中不包含的边的权重增加时，更新MST。给出你的算法的增长阶的运行时间，作为V和/或E的函数。
   d. 当MST中包含的边的权重增加时，更新MST。给出你的算法的增长阶的运行时间，作为V和/或E的函数。

a. 将更新的边加入MST，这将会创建一个循环。运行一个循环检测器在扩充后的MST上，找出那个循环。在那些边上循环，追踪最大权重的边。移除那个最大权重边，恢复一个MST。寻找循环（使用类似DFS的东西）通常是O(E)，但是扩充的MST只有V条边，因此运行在O(V)时间。同样地，找到那个循环中最大权重的边可以在O(V)时间内完成。所以，运行时间是O(V)。（2分）
b. 什么都不做，因为这不会影响MST。常数时间。（2分）
c. 什么都不做，因为这不会影响MST。常数时间。（2分）
d. 移除MST中的相关边，这会将其断开成两个连接的组件。在其中一个组件的端点上运行DFS，标记所有顶点在一个组件中。未标记的顶点来自于另一个连接的组件。现在追踪所有边，保持追踪最小权重边的边。将那个最小权重边加入到MST，恢复一个连接的组件。DFS找到一个连接的组件将运行在O(E)时间，同样，找到最小生成树中未连接组件的最小权重边也是在O(E)时间内。因此，运行时间是O(E)。（2分）


19:14 12月20日周三

Update the MST when | Solved Which of the fo | Minimum Spanning Tre | courses.engr.illinois.ed | Minimum Spanning Tre

cs.princeton.edu

1. You are given an edge-weighted undirected graph, using the adjacency list representation, together with the list of edges in its minimum spanning tree (MST). Describe an efficient algorithm for updating the MST, when each of the following operations is performed on the graph. Assume that common graph operations (e.g., DFS, BFS, finding a cycle, etc.) are available to you, and don't describe how to re-implement them.
   a. Update the MST when the weight of an edge that was not part of the MST is decreased. Give the order-of-growth running time of your algorithm as a function of V and/or E.
   b. Update the MST when the weight of an edge that was part of the MST is decreased. Give the order-of-growth running time of your algorithm as a function of V and/or E.
   c. Update the MST when the weight of an edge that was not part of the MST is increased. Give the order-of-growth running time of your algorithm as a function of V and/or E.
   d. Update the MST when the weight of an edge that was part of the MST is increased. Give the order-of-growth running time of your algorithm as a function of V and/or E.

Answers

   a. Add the updated edge to the MST, which will create a cycle. Run a cycle detector on the augmented MST, to find the edges in that cycle. Loop over those edges, keeping track of the one having maximum weight. Remove that maximum-weight edge, restoring an MST. Finding a cycle (using something like DFS) is normally O(E), but the augmented MST has only V edges, hence runs in O(V) time. Similarly, finding the max-weight edge in that cycle can be done in O(V) time. So, the running time is O(V). (6 points)
   b. Do nothing, since this can't affect the MST. Constant time. (2 points)
   c. Do nothing, since this can't affect the MST. Constant time. (2 points)
   d. Remove the edge in question from the MST, which will disconnect it into two connected components. Run DFS from one of the endpoints of that edge, marking all vertices in one of those components. (The unmarked vertices form the other connected component.) Now loop over all edges, keeping track of the minimum-weight edge having one endpoint in the marked set and its other endpoint in the unmarked set. Add that minimum-weight edge, restoring an MST. DFS to find a connected component will run in O(E) time, as will finding the minimum-weight edge spanning the two connected components. So, the running time is O(E). (6 points)

2. Given a minimum spanning tree T of a weighted graph G, describe an O(V) algorithm for determining whether or not T