Avoid Backwash Issue in Percolation Problem (Without 2 WQUUF)

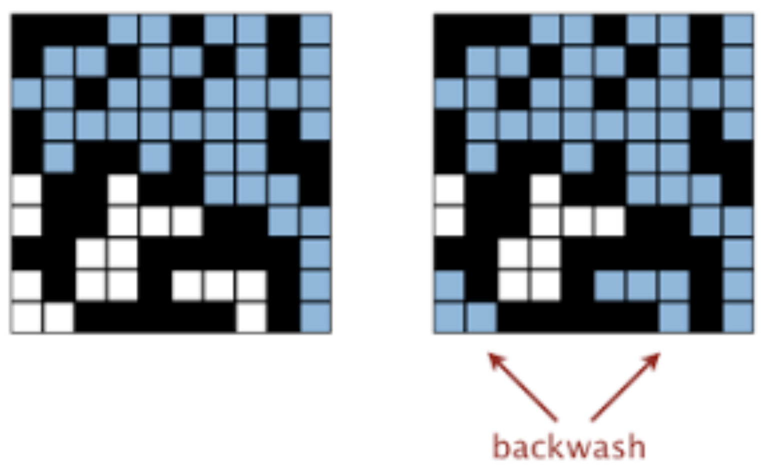
Posted on September 16, 2014 by sigmainfy — 14 Comments ↓ Pageview: 3,962

Preface:

This is a summary of all different solutions to the backwash problem within the percolation simulation including: 1) Associate each component root with an additional information (Without 2 WQUUF), I summarize two methods here, one involves modification of the given API while the other not; 2) using TWO Weighted Quick Union Union Find (WQUUF) objects. If you want to check the one without 2 WQUUF and no need to modify the given API which is what most of the people are looking for, you can directly jump to the the end of section 3 “Solutions to Resolve the issue” for the details.

Backwash Problem Definition:

In the context of Percolation, the backwash issue is that some site might be mistakenly judged as a full site (A *full* site is an open site that can be connected to an open site in the top row via a chain of neighboring (left, right, up, down) open sites.) if we directly adopt the dummy nodes suggested in the course material, i.e., a top virtual node connected to each site in the first first top row, another bottom virtual node connected to each site in the last bottom row. More concretely, it is just as the following pic shows (adopted from this post)



Solutions to Resolve the Backwash Issue:

1) The easiest way to tackle this problem is to use two different Weighted Quick Union Union Find objects. The only difference between them is that one has only top virtual site (let’s call it uf_A), the other is the normal object with two virtual sites top one and the bottom one (let’s call it

Search

search here … Go

HOT:

1. "LeetCode Handbook: All Problem Solution Index"
2. "Summary for LeetCode 2Sum, 3Sum, 4Sum, K Sum"
3. "LeetCode All Problems Solution Index: Simulation, Math, Hash, Bit Operation and Others"
4. "LeetCode All Problems Solution Index: Linked List and Array"
5. "LeetCode All Problems Solution Index: Search, Sum and Tree Problems"
6. "LeetCode All Problems Solution Index: Backtracking, Greedy and DP"

Notice

2. The site is recently rebuilt to be fully mobile friendly, any comments or feedback would be highly appreciated.

3. Since the domain name and the web hosting all cost real money, I have to create ads units to support myself and I will try my best to minimize the

uf_B) suggested in the course, uf_A has no backwash problem because we "block" the bottom virtual site by removing it. uf_B is for the purpose to efficiently determine if the system percolates or not as described in the course. So every time we open a site (i, j) by calling Open(i, j), within this method, we need to do union() twice: uf_A.union() and uf_B.union(). Obviously the bad point of the method is that: semantically we are saving twice the same information which doesn't seem like a good pattern indeed. The good aspect might be it is the most straightforward and natural approach for people to think of.

2) And there turned out to be more elegant solutions without using 2 WQUUF if we can modify the API or we just wrote our own UF algorithm from scratch. The solution is from "Bobs Notes 1: Union-Find and Percolation (Version 2)": Store two bits with each component root. One of these bits is true if the component contains a cell in the top row. The other bit is true if the component contains a cell in the bottom row. Updating the bits after a union takes constant time. Percolation occurs when the component containing a newly opened cell has both bits true, after the unions that result from the cell becoming open. Excellent! However, this one involves the modification of the original API. Based on this and some other discussion from other threads in the discussion forum, I have come up with the following approach which need not to modify the given API but adopt similar idea by associating each connected component root with the information of connection to top and/or bottom sites.

(3) Here we go for the approach based on Bob's notes while involving no modification of the original given API:

In the original Bob's notes, it says we have to modify the API to achieve this, but actually it does not have to be like that. We create ONE WQUUF object of size N * N, and allocate a separate array of size N * N to keep the status of each site: blocked, open, connect to top, connect to bottom. I use bit operation for the status so for each site, it could have combined status like Open and connect to top.

The most important operation is open(int i, int j): we need to union the newly opened site (let's call it site 'S') S with the four adjacent neighbor sites if possible. For each possible neighbor site(Let's call it 'neighbor'), we first call find(neighbor) to get the root of that connected component, and retrieves the status of that root (Let's call it 'status'), next, we do Union(S, neighbor); we do the similar operation for at most 4 times, and we do a 5th find(S) to get the root of the newly (copyright @sigmainfy) generated connected component results from opening the site S, finally we update the status of the new root by combining the old status information into the new root in constant time. I leave the details of how to combine the information to update the the status of the new root to the readers, which would not be hard to think of.

For the isFull(int i, int j), we need to find the the root site in the connected component which contains site (i, j) and check the status of the root.

For the isOpen(int i, int j) we directly return the status.

For percolates(), there is a way to make it constant time even though we do not have virtual top or bottom sites: think about why?

So the most important operation open(int i, int j) will involve 4 union() and 5 find() API calls.

Conclusion and Summary of Backwash Problem:

I outline the tree methods here to resolve the backwash (copyright @sigmainfy) issues:

1) Using 2 WQUUF objects:

- open(): would involves 8 union() API calls at most but the time complexity is still 8 * log (N^2) = O(logN)

influence.

Recent Comments

- ZerocoolCoder on 4Sum Problem Analysis: O(N^2) VS O(N^2logN) VS O(N^3)
- Sigmainfy on 4Sum Problem Analysis: O(N^2) VS O(N^2logN) VS O(N^3)
- Xue on 4Sum Problem Analysis: O(N^2) VS O(N^2logN) VS O(N^3)
- [LC] 3Sum | codingarms on Summary for LeetCode 2Sum, 3Sum, 4Sum, K Sum
- Java practice: Percolation problem revisited | Data Recipe on Avoid Backwash Issue in Percolation Problem (Without 2 WQUUF)
- Srikanth on Avoid Backwash Issue in Percolation Problem (Without 2 WQUUF)

Recent Posts

- What Startup Files Are Read by the Shell? (Shell Configuration)
- Shell Pattern Explained with Shell Script Examples
- Difference Between Single and Double Quotes in Bash
- LeetCode Remove Linked List Elements: Iterative vs Recursive
- LeetCode Delete Node in a Linked List: O(1) Time and Space
- LeetCode Palindrome Linked

- isOpen(): O(1) by memorizing in a 2-d array
- isFull(): by call connected() API and thus is O(logN)

2) Using 1 QQUUF objects and store additional information for keeping track of each connected component root:

- open(): would involves 4 union() and 5 find() API calls at most but the time complexity is still 9 * log (N^2) = O(logN)
- isOpen(): O(1) by memorizing in a 2-d array
- isFull(): by call find() API and thus is O(logN)

A final remark: one key observation here is that to test a site is full or not, we only need to know the status of the root of that connected component containing the site, whether a site is full is the same question interpreted by asking whether the connected component is full or not: this make things simple and saves time, someone in the forum proposed to linear scan the whole connected component to update the status of each site which is not necessary and inefficient. each time we only update the status of the root site rather than each site in that component.

(End. Original articles, please associate the author information and original URL with any reprints)

‹ Interview Questions 5: Successor with delete

Two Sum Problem Analysis 1: Sort and Hash with unique solution ›

Tagged with: backwash, percolation, Union-Find

Posted in Algorithm

14 comments on "Avoid Backwash Issue in Percolation Problem (Without 2 WQUUF)"

yijiem says:
September 29, 2014 at 20:33

Thanks a lot! It works like a magic!

Reply

> sigmainfy says:
> September 29, 2014 at 21:32
>
> Yijie, you are very welcome! 🙂
>
> Reply

watabou34 says:
January 30, 2015 at 13:08

List: O(1) Space

Tags

3sum 4sum Amazon Array Azure backtracking bfs binary search Bit Operation Blob BST C/C++ dfs dp duplicate git greedy hash HTML index inorder Java javascript LeetCode linked list linux Math Overflow Palindrome pat php quick sort S3 set simulation Solution sort stack String Swap tree Two Pointers Two Sum Union-Find wordpress

Categories

- Algorithm
- Android
- C/C++
- Git
- Interview
- Java
- JavaScript
- Languages
- LeetCode
- Tech
- Web
- WindowsProgramming
- WordPress
- 癫笑痴狂

Archives

- September 2015
- August 2015
- July 2015
- June 2015
- May 2015
- April 2015
- March 2015
- February 2015
- January 2015
- December 2014
- November 2014
- October 2014

Thanks for this explanation. However I'm still having a little bit of trouble.

So for each open() call, are we updating the status of both S and the new root of S after all of the potential unions have been made? Also I'm confused as to how percolates() can work. I can't figure out what site we would check to figure if the grid is percolating.

I'm having problems with certain information from the status from being "erased". Like sometimes sites in the top row will lose the bit that contained the information on "connected to top" and I'm not sure why it's happening.

Reply

Sigmainfy says:
February 2, 2015 at 16:59

Hi watabou34, please check the reply to nemo below, essentially, as long as we keep the status like connect to the top, connect to the bottom, whether it percolates or not could be determined by testing any site which is connected to both top and bottom, and this could be done as an additional constant time operation each time we do open(). Hope it helps!

Reply

nemo says:
February 1, 2015 at 20:05

博主你好，我昨天用两个WQUUF实现了percolation。这种实现存储了重复的信息，所以肯定有更好的方法。今天搜到了您的文章，对于保存节点状态的思想表示理解，只是您第三部分省略的改变节点状态的方法我想了半天没想明白，可否给点更明确的提示?

Reply

Sigmainfy says:
February 2, 2015 at 16:55

Well, the motivation of the third approach here is that instead of modifying the the original API, we can keep useful information for each site in a separate array. More concretely, take the status of "connect to top" for example, and we need to connect two site s1 and s2, this "connect to top" status applies to every site within the connected component where s1 resides, and we only need to associate it with the root, after union(s1, s2), it becomes a single connected component, and for this component, whether it is connected to the top or not only depends on any of the roots for s1 and s2 connected to the top or not, if so, then the new component is connected to the top, otherwise, it is not. And in order to determine whether it percolates, we only need to see whether there is any site connected to both top and bottom.

Reply

nemo says:
February 4, 2015 at 01:38

我反应过来了。。这种方法是不需要开始和结束两个vitual nodes的吧？我之前以为还要连着那俩，怎么想都觉得还是会有bashwash的现象。
如果不带那俩的话，就需要多一个属性标记是否percolate，每次open之后检查一

- September 2014
- August 2014
- July 2014
- June 2014
- May 2014
- April 2014
- March 2014
- February 2014
- January 2014
- December 2013
- November 2013
- October 2013
- September 2013
- August 2013
- July 2013
- June 2013
- May 2013
- April 2013
- March 2013
- February 2013
- January 2013
- December 2012

次。这样percolates()方法才能在线性时间内完成。
非常感谢你的解答！:)

Reply

Yunfeng says:
February 6, 2015 at 22:08

感觉一个status数组不够标记四个状态:open, blocked, connect-to-top, connect-to-bottom。如果有个node既连top，又连bottom怎么标记，求详解。

Reply

Sigmainfy says:
February 7, 2015 at 12:27

As I mentioned in the post, you can use bit operation to maintain the different status or any combinations~ 😐

Andrew says:
February 5, 2015 at 17:42

Thanks for this post. Super helpful!

Reply

Sigmainfy says:
February 5, 2015 at 18:51

You are very welcome, Andrew!

Reply

Srikanth says:
June 10, 2015 at 23:04

Hello,

I am bit confused and sure that I am missing something here.
If suppose, I have a connected component of 100 sites and all of them are not connected to the top site yet. So, all these 100 sites will have a status like “opened”. Now, if a union operation between a site in the top row and a site in this connected component, then all these nodes will be a full site. So, how we update the status of all these 100 sites (to open and connected to top)? Are we going to update all the 100 sites? Please clarify.

Reply

Sigmainfy says:
June 12, 2015 at 18:42

Hey Srikanth, in your case, we do not have to update all of the 100 sites which is considered as inefficient, as I mentioned in the end of the post: “each time we only update the status of the root site rather than each site in that component.” Hope this helps! And feel free to ask if you still have any other questions.

Reply

Srikanth says:
June 13, 2015 at 07:20

Thank you for the reply. But I am sorry! I couldn’t understand, how we will have the corrected status of each and every site just by looking at the root. Because, the connected component that causes the backwash problem would be connected to top site through the virtual site.

Can you please share your code? I will try to understand.

Reply

2 Pings/Trackbacks for "Avoid Backwash Issue in Percolation Problem (Without 2 WQUUF)"

[Coursera] Algorithm(Princeton) HW1-Percolation | JYUAN says:
October 8, 2014 at 16:02

[...] A good Summary for this Assignment:Avoid Backwash Issue in Percolation Problem (Without 2 WQUUF) [...]

Java practice: Percolation problem revisited | Data Recipe says:
June 24, 2015 at 11:16

[...] I have a previous post about the percolation problem using a recursive approach. Here’s another approach using weighted quick union. This is also the first programming assignment of Coursera course Algorithms, Part I. The tricky part is determine if the site is full. I took an intuitive approach by creating two WeightedQuickUnionUF objects. There is a discussion about a more creative method. [...]

Leave a Reply

Your email address will not be published. Required fields are marked *

Name *
E-mail *
Website

Comment

Post Comment