CHALLENGE PROMPT

Kayak is thinking of providing weather information to users to help them plan trips. As an experiment before doing a full integration, they’d like to create a fun tool that tells users where the warmest temperatures are within 30 miles of their destination.

Create a basic version of this tool on any platform you'd like (web or mobile). Use the[WeatherBug weather API](http://developer.weatherbug.com/docs/read/WeatherBug_API_JSON) for weather information and the [Geonames API](http://www.geonames.org/export/web-services.html) for finding populated cities within 30 miles of a given location (search for**api.geonames.org/findNearbyPostalCodes** and**api.geonames.org/findNearbyPlaceName** on the linked page). If you want to build a fully functional app, feel free to use the username **ms\_test201302** and the premium Geonames server **ws.geonames.net** for your requests and the WeatherBug keys**dpkdcj78p5mqnukn8sfv8jkf** for WeatherBug GEO Basic and **rpjv5wkg9q465bkuzrhdrqbg**for WeatherBug REST XML.

Your final product must take an input city and give an output of the top 10 warmest temperatures (from warmest to coldest). This list of top 10 warmest temperatures should include - 1) The location of the temperature and 2) Which day it happens on. Use only data from the 7 day forecasts, and cities within the US.

Submit your code (or pseudo-code with calls to the aforementioned APIs) and a brief paragraph describing your algorithm. Feel free to wow the Kayak team by customizing your weather tool with more cool features. Bonus points for submitting a link to a real demo!

Winning Proposal

As chosen by Kayak

PROPOSAL

My algorithm:

query the geonames API via HTTP GET with a postal code and a 48 km (30 mile) radius

sort the results of the query  O(nlogn)

create an empty quadtree

for each result r : O(n)

- skip if the placeName is empty or if the placeName is equal to the previous entry's placeName

- query the quadtree for points within a 10km square of the current point O(m) or O(log m)

- if there are already points in the quad tree that are within the 10k square, then add the current point to that quad tree's array

(this iteration results in a quad tree where every point contains an array of cities that are closest to that point)

create a hash from the quad tree where the key is the postal code of the first city in the quad tree node's array. e.g. { 1234 => [{city1}, {city2},{city3}]} O(m)

query the weatherbug API via HTTP GET with all of the postal codes from the hash

create an empty minheap

for each weather result r: O(m)

if the tempature of r is greater than the min value of the heap

delete the min value of the heap

add current r to the heap

if size of heap is less than 10- add current r to the heap create empty array

while heap is not empty

*extract the min of the heap and add it to the front of a new array*

return the array

(n = number of cities from geonames, m = number of "groupings" in the quad tree)

Explanation:

I decided to filter the results from geonames using a quad tree because the results were enormous if I didn't. For example, querying New York city resulted in 862 different postal codes. I had hoped to perhaps ignore cities with a very small population (since high population cities are more interesting for travelers) but there was no convenient way to do that without making another HTTP request. So I decided to create a quad tree and group the cities that were within a 10km square of each other. I figured that cities that were \*that\* close to each other probably had very similar weather (weather reporting is not super accurate anyway).

After grouping, my results for New York city were reduced to around ~100 postal codes that I had to query weatherbug for. After getting the results from weatherbug, I inserted each city into a minheap with the temperature as the key, and I ignored any cities where the temperature was less than the min. This way any insertion into the heap would be at most log10.

After popping each element from the heap, you get your sorted array of temperatures!

The biggest bottleneck in terms of performance is of course the HTTP requests. Populating the initial quad tree is also time consuming, as each "search" in the quad tree could potentially be log n, working out to nlogn iterations. If this were a large scale app with millions of requests per day, and it is required to use geonames to find the 30 miles data, then perhaps it would be better to use an object caching system like memcached or a database with Rquad capabilities to store the data from geonames (since postal codes are unlikely to change frequently).

You can see a working version of this project at <http://hotspots.herokuapp.com/,> or view the code at <https://github.com/kolauren/hotspots> (the main algorithm described above can be found in app.rb)

EDIT: unfortunately it seems like I've hit the cap on queries to weatherbug, so the above link might not work properly. You can still run it locally by cloning the repository and typing "bundle install" and then "thin start".