Weekly Report Aug 6,2018-Aug 12,2018

Haoyu Wang
School of Software Engineering
Shanghai Jiao Tong University
gogowhy@sjtu.edu.cn

Abstract

In a nutshell, I continued the work of last week.

1 Tuesday

1.1 current state

1. I finished week 7 of Algorithm(graph searching)

1.2 The record of my Algorithms

1.2.1 HEAP

1. Firstly, according to my buddy's advice, I learned about several data structures, namely lists, stacks, queues, heaps, search trees, hash tables, bloom filter and union-find,etc. First of all ,is the HEAP structure mentioned by me last week.
   HEAP is a container of ojects that have keys, here are several common operations on a HEAP:
   INSERT:add a new objextt to a HEAP running time:O($\log n$)
   EXTRACT-MIN(MAX): remove an object in a HEAP with a minimum (or maximum)key value running time:O($\log n$)
   HEAPIFY: n batch insert in O(n) time
   DELETE: from a middle running time:O($\log n$)

2. Then I learned about the first application of heap– sorting, which is used as a fast way to do repeated recall for the minimum
   HEAP SORT:
   1. insert into the heap
   2.extract one by one
   The running time can be 2n$\log n$
   MEDIUM MAINTENCE:
   X1,X2......Xm divided into Hlow and Hhigh and the medium is just between them
   SPEEDING UP DIJKSTRA:
   By using heaps, we can do the graph search with the running time of O(m$\log n$)

3. Then I learned about the HEAP property, it;s conceptionally a rooted binary complete tree, at every node X,key[X]$\leq$all children's keys ,Thus we say that the root is the minimum
   NOTE:
   parent(i)= $\frac{i}{2}$ if i is even
   parent(i)=$[\frac{i}{2}]$ if i is odd
   children of i=2i,2i+1

4. Then I went through the INSERTION AND BUBBLING UP of HEAP:
   INSERTION:
   step1: stick k at the end of the last leave
   step2:swap key (if violate the rule then bubble up k until it's properly stored)
   EXTRACT:
   step1: delete root
   step2:move last leaf to be the new root
   step3:swap the smaller child

1.2.2 BINARY SEARCH TREE

Here is a comparison of the traditional sorted array and BINARY SEARCH TREE:
SORTED ARRAY:
SEARCH O($\log n$)
SELECT O(1)
MIN/MAX O(1)
PRED/SUCC O(1)
RANK O($\log n$)
OUTPUT O(n)
BINARY SEARCH TREE:
SEARCH O($\log n$)
SELECT O($\log n$)
MIN/MAX O($\log n$)
PRED/SUCC O($\log n$)
RANK O($\log n$)
OUTPUT O(n)
INSERT O($\log n$)
DELETE O($\log n$)

5. Then I learned about the BST structure
   -exactly one node per key
   -each node has three nodes:
   1.left child
   2.right child
   3.parent

6. Then I learned about its property:
   the key left is less and the key right is bigger
   The Print procedure is a recursive tricky call:
   -let r = root of search tree, with subtrees Tl,Tr
   -recurse on Tl
   -print out r
   -recurse on Tr
   The running time is O(n)

7. Then I learned about its deletion
   EASY (k has no children) delete directly
   MEDIUM (k has one child) delete k and swap
   DIFFICULT (k has two children)
   compute k's PRES L
   swap k and L
   delete k

8. Then I learned about the select and rank:
   SELECT
   -start at x ,with children y and z
   -let a =size(y)

-if a=i-1, return x's key
-if a ¿i-1,recurse on Y
-if a ¡ i-1,recurse on z

1.2.3 RED BLACK TREE

9. how to balance the tree to make it as complete as possible? We here use the RED BLACK TREE, we can ensure that the height is always O($\log n$)
The princple of it is actually quite simple:
-each node is red or black
-the root is black
-no two red in one row

2 Tuesday

2.1 current state

1. I finished week 8 of Algorithm(graph searching)

2.2 records of my algorithm

2.2.1 HASH TABLE

1. The purpose od hash table is to maintain a set of stuff, we use a key to insert,delete and look up, these operations all run in a O(1) constant time and there are some applications of the HASH TABLE:

2. DE-DUPLICATION a goven stream of objects, we linear scan the objects, the goal is to remove the duplicates, the solution is as follows:
-look up i nhash table
-if not found, insert x

3. 2-sum problem , we input an unsorted array od n integers ,target sum t, we want to figure out whether or not there are 2 numbers x,y that x+y=t
NAIVE:
-O(n^2) running time exhaustive search
BETTER:
-sort A O($n \log n$)
-for each x in A, look for t- x in A with O(nlog n)
AMAZING:
-insert elements of A
-for each x in A, look up t-x in H
HISTORICAL APPLICATION:
-symbol tables in compiles
-blocking network traffic
-speed up search algorithm

4. Then I learned about the high level idea:
-picl n= number of buckets
-choose a hash function h(x)
-use array A of length n , store x in A(h(x))
but in a hash function ,sometimes the different elements collide in a same bucket, there are two solutions:
CHAINING:
-keep linked lists in each bucket
-given a key (object x, perform insert delete look up)in the list in A(h(x))
OPEN ADDRESSING

-hash funtion now specifies probe sequence h1(x), h2(x) (keep trying until find a open slot)
-linear probing (look consecutively)

5. HASH FUNTIONS
   1. It should be easy to store and be very fast to evaluate
   2.It should lead to good performance
   Beacuse all of the hash functions has there disadvantages, so we should design it accoring to the problem, one of the method is the QUICK AND DIRTY HASH FUNCTION, which links the objects with integers and integers with buckets

6. Then I learned how to choose na s buckets
   -n should be a prime
   -not too close to the power of 2
   -not too close to the power of 10

2.2.2 BLOOM FILTER

7. Then I learned about the use of bloom filter, it has some advantages and disadvantages
   ADVANTAGE:
   -move space efficiently
   DISADVANTAGE:
   -can not store an associated object
   -no deletions
   -small false positives
   Then we can look up if the elements exists in the list easily!

3 summary

These days there are too many other trifles, spoiling the learning procedure.

4 Saturday

4.1 current state

1. I finished week 9 of Algorithm(greedy algorithm)

4.2 records of my algorithm

4.2.1 APPLICATION of the GREEDY ALGORITHM

1.1. internet-graph, we take the vertices as end hosts and router
1.2.web-graph, we take edges as hyperlinks
we use BELL-Ford algorithm to calculate the shortest path
2.1 sequence alignment, we use two strings over the alphabet , to check out how similar they are(in gene similarity)

4.2.2 Paradigms of design algorithm

-divide and conquer
-randomized algorithm
-greedy algorithm
-dynamic algorithm

4.2.3 Greedy algorithm

-DEF: we take myopic decision, hope everything work at the end

-EXAMPLE: the Dijkstra's shortest path algotithm, it's processed once on each step irrevocably what kind of problems can be solved by greedy algorithm? for an example, we can solve the caching problem ,using the BELADY(1960's) theorem "the furthest-in-future", as guideline for practical algorithm ,and such problem can be extended to a new stage–scheduling problem

4.2.4 scheduling problem

we assume the job j has weight wj, and length lj, we want the minimum of the comletion time Cj, obviously:

-if the jobs has equal length and different weight, we put heavier ahead

-if the jobs has equal wight and different length,we put shorter ahead

we assume the score in two forms: Wj-Lj and $\frac{Wj}{Lj}$, by using examples we find that the former is wrong, then we use the induction method to prove that the score can be expressed as the later form.

-we sort the jobs in the order 1 ,2 ,.....n as $\frac{W1}{L1} > \frac{W2}{L2} >$ ...... $> \frac{Wn}{L1n}$

-if ther is a better way than the greedy method, there should be i¿j, if we exchange i and j , the cost is WiLj, and the benifit is WjLi, obviously benifit is bigger than cost ,if we exchange like this for at most C^n2 times ,the score can be proved right.

4.2.5 PRIM's MST ALOGORITHM

INPUT: undirected graphG(V,E)

-assume adjacency lust representation

-negative is ok

OUTPUT: minimum cost tree that spanns all vertices

the operations are ass follows:

-initialize xX=(s)

-T=

-while X$\neq$T

let e(u,v)=the cheapest edge

add e to T

add v to X

if we use the HEAP structure, the running time can be easily O(m$\log n$)