Homework 3 Dec. 29, 2020

3. Triple Sum

given an array $A[0...n-1]$ with n elements, each ele of A is in the range $0 \le A[i] \le n$. If there exist indices i, j, K such that $A[i] + A[j] + A[K] = n$.

Design an $O(n \log n)$ algorithm. Don't need to return the indices, just yes or no.

Solution:

Using Exponentiation to convert multiplication to addition.

eg. $[1,3] \rightarrow x^1 + x^3$

$[2,4] \rightarrow x^2 + x^4 \quad \times \rightarrow x^3 + 2x^5 + x^7 \quad [3,5,7]$

Note that: $[1,3] + [2,4] = [3,5,7]$ possible sum

def $p(x) = x^{A[0]} + x^{A[1]} + \cdots + x^{A[n-1]}$

$q(x) = p(x)^3 = \sum_{i,j,k} p^{A[i]+A[j]+A[k]} \quad (0 \le i, j, k \le n-1)$

indices i, j, k exist $\Leftrightarrow$ x^n in q is nonzero for n.

$r(n)$:

	Time	Note:
convert $p(x)$	$O(n)$	$0 \le A[i] \le n$
compute $q(x)$	$O(n \log n)$	so FFT is efficient
coefficient x^n	$O(1)$	If not, it's a
return	$O(n \log n)$	3SUM problem (solved in $O(n^2)$)

4. Protein Matching

g is a length-n string (gene)

... is a length-m string (full DNA sequence), $m \geq n$. Find the starting locations of all length n-substrings of s which match g in at least $n-k$ positions.

eg. g = ACT s = ACTCTA $k=1$

output: [0, 2]

(a) Given a $O(mn)$ time algorithm.

Solution: Two pointers

For i in $[0 \ldots m-n]$ starting points, check if the substring differs from g in at most k positions.

T(n):

starting point $O(m-n+1)$

each point's check $O(n)$

overall $O(mn)$

(b) Assume g and s are given as bit strings, i.e. each character is 0 or 1. Give a $O(n \log n)$ algorithm that works for any k.

Solution: g', s' be g, s with 0 replaced by -1.

$P_1(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}$, $a_d = g'(n-d-1)$ (inverse order)

$P_2(x) = b_0 + b_1 x + \cdots + b_{m-1} x^{m-1}$, $b_d = s'(d)$ (positive order)

$P_3(x) = P_1(x) \cdot P_2(x)$

the coefficient of x^{n-1+j} in P_3 is the dot product of the substrings in s' starting at index j and string g'.

differ at most $k \Rightarrow$ dot product $\geq n-2k$

Step: compute $P_3(x)$, output all the j's between 0 and $m-n$ such that $c_{n-1+j} \geq n-2k$.

5. Practice with SCCs

(a) Design an algorithm that a directed Graph G, outputs the set of all vertices v that ...

... is a cycle containing v.

I.e. output all v that there is a non-empty path from v to itself.

Solution: Compute SCCs of G, output all vertices in SCC that is larger or equal to 2.

Note: SCC means a path $u \to v$ and a path $v \to u$, or there is a cycle in SCC.

Vertices in SCC can be connected strongly vertices in the same SCC. $T(n) = O(|V|+|E|)$.

(b) Design an algorithm that a directed Graph G, determines whether there is a vertex v from which can be reached. vertices

Solution: 1) Compute the SCC and metagraph.

Find size of source node on metagraph.

True $\Leftrightarrow$ size ≥ 1 False $\Leftrightarrow$ size $= 0$

$T(n)$:

	Time
SCC	$O(\|V\|+\|E\|)$
find source	$O(\|V\|+\|E\|)$

$T(n) = O(|V|+|E|)$

2) step: ① DFS on G form a forest.

② v is the root of the last DFS tree.

③ DFS starting from v to see if reachable.

$T(n) = O(|V|+|E|)$ DFS twice.

6. Splitting Edges.

$T = (V, E)$ is an undirected connected Tree.

For each $e \in E$, $T \setminus e$ has exactly two connected ...

sizes of these two components are split numbers of e.

(a) Describe a $O(n)$ algorithm compute the split number of all edges $e \in E$.

Solution: An arbitrary node $r \in V$. For any $v \in V$, define $N(v)$ as the number of nodes rooted at v.

Run a DFS of T from r, on the post-order visit of vertex $v \in V$, compute $N(v) = 1 + \sum_{c \in \text{children of } v} N(c)$ (size of tree)

Any edge $e = \{p, c\}$ p is the parent of c in DFS tree rooted at r. It's split numbers are $N(c)$ and $|V| - N(c)$.

Note: post-order: from sink to source, to get $N(\text{sink})$ first, then use $N(s) = \sum N(\text{sink}) + 1$ to get $N(\text{source})$.

$T(n)$: Computing $N(v)$ each v $\quad O(\text{degree}(v))$

Computing $N(V)$ $\quad O(|E|)$

Overall $\quad O(|E|) = O(|V|)$

(b) For any $u, v \in V$, $d(u,v)$ is the number of edges on the unique path from u to v. Suppose we know the split numbers for each edge $e \in E$. given only the split numbers and not the original graph, how to compute the total pairwise distance between all vertices.

$$R := \sum_{u,v \in V} d(u,v)$$

Input: edge e: (x_e, y_e).

Solution: e lies on $u \to v \iff u$ and v are divided by e. (x_e, y_e).

Each edge e: $x_e \xrightarrow{e} y_e$ ($u \to v$)

contribution of e: ways of choosing starter and ender is $x_e \cdot y_e$ (starter and ender are

$R = \sum_{e \in E} x_e \cdot y_e$.

divideen /.

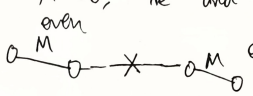
(c) Perfect matching: a set of edges $M \subseteq E$ every vertex incident to exactly one edge in M.

Let T be a tree has a perfect Matching M.

Prove: $M = \{e \in E : x_e \text{ is odd and } y_e \text{ is odd}\}$.

Solution:

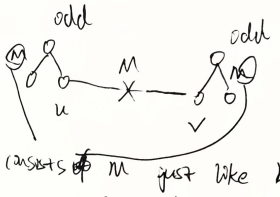
Suppose e is not in M. Each component of $T \setminus e$ still has a perfect matching. (Because e is not in M, whether e there still a M). So, x_e and y_e are even.

eg. 

Suppose e is in M. T' is the component of v of $T \setminus e$.

$T' \setminus u$ has a perfect matching by matching vertices in M.

So $|T' \setminus u|$ is even, $|T'|$ is odd. The next component of v is the same.

eg. 

(consists of M just like before.

even size, every vertex is used once.

7. Vertex separators,

$G = (V, E)$ an undirected, unweighted graph.

Call a set of vertices S a u-v separator

if 1) S doesn't contain u or v

2) delete vertices and edges adjacent, disconnect u and v.

I.e. S is a u-v separator if every path from u to v goes through some vertex in S.

Given an algorithm takes G, u and v as input and finds a u-v separator of size at most $\frac{n-2}{d-1}$, where d is length of shortest path u->v.

Solution:

run BFS from u to compute dist from u.

Let S_i be the set of all vertices at distance i from u.

Output whichever of $S_1, S_2, \dots S_{d-1}$ is the smallest.

Correctness: For all i to d-1, u->v is d, so u->v must pass through some vertex in S_i. So deleting S_i can disconnect u and v. (dist d is obtained by (d-1)+1, also d-1 = (d-2)+1, so  v. must pass through).

total size is at most n-2, average size = $\frac{n-2}{d-1}$.

So smallest $\leq \frac{n-2}{d-1}$.

$T(n) = O(|V|+|E|)$ (BFS)