Principles of Information Security

Indranil Pradhan

2019202008

MTech in CSIS

[R]
Computing [*m; k*]-(*s; t*)-**Weak-Connectivity:** A digraph *G* = (*V;A*) is said to be **k-**(*s; t*)**-connected** if, on the deletion of any *k* nodes (other than *s* and *t*) from G, there still exists a path from *s* to *t*. There are several famous polynomial-time algorithms to compute the maximum **k** such that *G* is **k-**(*s; t*)**-connected** (for example, many are based on algorithms for max-flow). Let V^* denote set of vertices that have a path to *t*, i.e. $V^* = \{u \mid \exists$ path from u to t in $G\}$. The digraph *G* = (*V;A*) is said to be **k-**(*s; t*)**-weak-connected** if, the underlying undirected graph G_u (that is, the graph obtained by replacing every arc in *G* by an undirected edge) when induced on V^*, denoted $G_u[V^*]$ (that is, we delete all the vertices in $V \setminus V^*$ in G_u), is **k-**(*s; t*)**-connected**. The digraph *G* = (*V;A*) is said to be **[m,k]-**(*s; t*)**-weak-connected** if, on the deletion of any *m* nodes from *G*, the remaining graph is **k-**(*s; t*)**-weak-connected**. The question is: *given a digraph G, and integers m and k, design an efficient algorithm to decide if G is* **[m,k]-**(*s; t*)**-weak-connected**.

Answer:-

Here according to the problem statement given, I have been provided with a digraph G and integers m and k. So I have to design an efficient algorithm to decide if G is [m,k]-(s; t)-weak connected.

First, I need to decide which m vertices of the graph G to be left out. For that efficient approach as follows-

The standard algorithm to find m vertices of the graph which can be removed is to remove m vertices that have degree less than- 'K' from the input graph in the increasing order of their degree. We must be careful that removing a vertex reduces the degree of all the vertices adjacent to it, hence the degree of adjacent vertices can also drop below 'K'. And thus, we may have to remove those vertices also. This process may/may not go until there are no vertices left in the graph.

If the process does not go then we terminate.

To implement above algorithm, we do a modified DFS on the input graph and delete m vertices having degree less than 'K' in the increasing order of their degree, then update degrees of all the adjacent vertices, and if their degree falls below 'K' we will delete them too.

So, the time complexity will be O(V+E).

After removing the m vertices from the G, we are now left with new graph G1.

Now we build G1' , the underlying undirected graph when induced in V* (where V* is the set of vertices that have a path to t, $V^* = \{u \mid \exists$ path from u to t in $G1\}$) that is we delete all the vertices in V\V* in the G1'.

on receiving a graph *G*1', a vertex *s* (the source) and a set of available vertices *N* (vertices that can be chosen as the sink), the algorithm randomly picks a vertex $t \in N - \{s\}$, and runs the max-flow algorithm to determine the max-flow from *s* to *t*. After this step, we will obtain two min-cut, (*S*, *T*). Since the connectivity of *s*, *t*, say *x*, is assigned to edge (*s*, *t*) as the connectivity between *s* and *t*. We also set (*S*, *T*) to the corresponding min-cut (for the case where *G* is undirected, (*S*, *T*) is already the desired min-cut). Then, an edge (*s*, *t*) with weight *x* is added to the auxiliary graph *A*. The procedure then calls itself recursively, first with *S* as the set of available vertices and *s* as the source. The recursive calls terminate when *S* is reduced to a single vertex.

Algorithm:

```
If N = {s}
    Return.
Randomly pick a vertex t from N − {s}.
(x, S, T) ≔ S-T MAX-FLOW(G1, s, t).
Add edge (s, t) with weight x to A
constructing (G1, s, N ∩ S)
```

After the auxiliary graph *A* is constructed, for each query *k*, the *k* -connected graph can be easily determined as follows: traverse the auxiliary graph *A* and delete all edges with weights less than *k*. Then the resulting graph represents a *k* -connected graph. So it can shown that ***G is*** **[m,k]-(*s; t*)-weak-connected.**

[Q]
Recall that a *pointer* to data X stores the *address* of X, denoted by &X. Similarly, let a *hashpointer* to data X be the address of X along with the cryptographic hash (say, H) of X, denoted <&*X;H*(*X*)>. Further, let a *hash&sign-pointer* to data X be the address of X, along with the cryptographic hash (say, H) of X that is digitally signed by the owner of X, denoted ⟨&*X;H*(*X*)*;* sigma⟩. Let D be your favourite data structure among stacks, queues, lists, trees, forests, graphs, priority-queues along with their variants (like Binomial heaps, skip-lists, red-black-tress and so on). Consider a **pointer-based** implementation of D. It is straight-forward to visualize the corresponding **hash-pointer-based** implementation of D, wherein every pointer &X is replaced by ⟨&*X;H*(*X*)⟩.What do you think are the advantages of the hash-pointer-based implementation of D over the pointer-based implementation of D? Specifically, can you think of one application/problem/setting/protocol, say A_{hash}, wherein a hash-pointer-based implementation of D is more suitable? Analogously, list the advantages of the hash&sign-pointer-based implementation of D and give an application A_{sign} where it is (more) suitable. Justify your answers.

Answer:-

Part1:-

The advantages of the hash pointer-based implementation of D over the pointer-based implementation of D-

1. the hash can be used to verify that information hasn't been changed which is not possible in normal pointer-based implementation.

2. In case of block chain we only need to keep the hash pointer to the last block. Then when somebody shows the whole blockchain later and claim the data in it is not modified, we can tell if any block in the chain is tampered by traversing the blocks backwards and verifying the hashes one by one. The normal pointer-based implementation does not give any surety about tampering of the data.

3. If we implement a binary tree with hash pointer and keep the hash-pointer in the root and then we can traverse down to any leaf data block to check if a node is in the tree or has it been tampered.

4. If we build a binary tree with hash pointer and sort it, the membership of any element can easily be said. if the data before and after the given data are both in the tree and they're consecutive, so there's no

space between them, this proves that the given data is not in three. So this is more efficient than the normal pointer-based implementation.

5. A regular pointer gives you a way to retrieve the information. A hash pointer will let us ask to get the information back and verify that the information hasn't changed. A hash pointer tells us where something is and what its value was. It also stores the hash of the value that this data had when we last saw it.

Part 2:-

one application/problem/setting/protocol, say A_{hash}, wherein a hash-pointer-based implementation of D is more suitable.

Hash pointer can be implemented any data structure which does not have cycle. If there is cycle, we can't match up the hashes. In acyclic data structure we can sort starting from the leaves from where no pointer is coming out and move backward towards the beginning. A directed acyclic graph can be constructed where we can verify membership efficiently or linked list like data structure can be built using hash pointer where data integrity can be pertained.

The application of the hash pointer is in block chain. Here the linked list of hash pointer is built. Here, every block contains data and the hash-pointer to the previous block. The hash pointer tells about the hash of what is the previous block and the data in it. And there is head just like it is present in linked list.
So, if an adversary moves forward from the back and modifies any bocks the hash changes, then there will be consistency in the following block which helps us to detect the tampered block. This is how data integrity and any attack can be detected.

Part 3:-

Hash and sign pointer based application has advantages over the hash pointer based implementation. Hash pointer based implementation gives the data integrity. Hash and sign pointer based application whereas provides data integrity with addition it also provides authenticity of data, the identity of owner. The hash message is signed by the owner with its private key and one can verify it with the public key of the owner. The owner also can't disagree with the authenticity. So it also provides non-repudiation.

The application of hash and sign pointer based implementation is in crypto currencies, online transaction, digital documents and certificates.