Hopscotch CSR Proof

1 Insertion Time Complexity Proof

The following discussion about *hopscotch CSR(HCSR)* is based on the assumption that we know that for each newly inserted edge, the probability of its source vertex being the i_{th} vertex is a_i. This describes how the graph evolves during the time. This assumption is quite strong since in practice it is usually impossible for users to know exactly how the graph will evolve beforehand, but its main purpose is to help give an analysis of the time complexity.

Assume at the time of initiation, the HCSR has N occupied slots and $\sum_{i=0}^{|V|-1}\lceil KNa_i\rceil$ empty slots, where K is a configurable constant.

Theorem 1. *In add-only scenario, if for each newly inserted edge, the probability of its source vertex being the i_{th} vertex is a_i, and in the hopscotch CSR the number of empty slots after the outgoing edges of i_{th} vertex is $\lceil KNa_i\rceil$, then the expected time complexity of inserting the j_{th} edge is $\mathcal{O}(\frac{1}{2}(1+\frac{1}{(1-\alpha)^2}))$, where $\alpha = \frac{j}{\sum_{i=0}^{|V|-1}\lceil KNa_i\rceil}$.*

Here we prove this theorem by first proving that a variation of HCSR(briefly denoted as *vHCSR*) is no worse than open address hashing and then showing our algorithm is no worse than that variation. In vHCSR, each new edge is given a randomly generated integer as its tag. An edge $u \to v$ with tag t is firstly hashed to i_{th} vertex's a_i slots, and then randomly hashed to one of these a_i slots by its tag. If this slot is empty, then the edge is placed here. Otherwise this edge is placed to nearest empty slot at the right side of this slot.

Lemma 2. *In add-only scenario, the probing length of insertion in vHCSR is no worse than in open address hashing for an hash table whose size if $\sum_{i=0}^{|V|-1}\lfloor KNa_i\rfloor$ and which initially is empty and is fed with the same insertion sequence.*

Proof. Lemma 2 is straightforward if KNa_i is an integer for all i, since in this case every edge is hashed to every slot with equal probability, so the vHSCR is exactly an equivalent definition of open address hashing.

□

Lemma 3. *In vHSCR, the expected time complexity of inserting the j_{th} edge is $\mathcal{O}(\frac{1}{2}(1+\frac{1}{(1-\alpha)^2}))$.*

Proof. This follows lemma 2 and the analysis of open address hashing in [1]. □

Then we can prove theorem 1 by proving that for every possible insertion sequence, the probing length of HSCR is no larger that vHCSR:

Proof. Suppose the to-be-inserted edge is $u \to v$. We classify the situation into 2 cases:

1). If in vHCSR, the slot this edge is hashed to is empty, then in HCSR, there must be at least one empty slot after the outgoing edges of u. So in both data structure the probing length is 1. 2). If in vHCSR, the slot this edge is hashed to is not empty, and this edge is placed to the nearest empty slot of this slot, then in HCSR, the distance between the nearest empty slot and the outgoing edges of u is guaranteed no larger than in vHCSR.

So in both cases, the probing length of HCSR is no larger than vHCSR. And since in HCSR the entries that are accessed during the process of moving the empty slot to the desired slot is equal to the probing length, so the total expected time complexity of insertion in HCSR is $\mathcal{O}(\frac{1}{2}(1 + \frac{1}{(1-\alpha)^2}))$. □

To give an straightforward example for the efficiency of HCSR, if there are 1 million edges and 200 thousand empty slots at the initiation, then after the insertion of 100 thousand new edges, the α becomes 50%, and the upper bound of expected time complexity for the next insertion operation is 2.5. We claim the expected time complexity is constant because we set an upper bound for α. Once α exceeds this upper bound, we will re-hash the edges to a larger HCSR.

And actually in many cases HCSR performs better than this. Because during the linear probing, if there are several edges belonging to the outgoing edges of the same vertex, they can be skipped at one operation instead of probing them one by one as in open address hashing.

As stated before, the assumption of above analysis requires that we know a_i for each vertex before hand. Generally speaking, in real world, the graph usually evolves in a way between two extremities: (1). If updates are completely random and are totally blind of previous graph topology, then we have $a_i = \frac{1}{|V|}$ for all vertices. (2). If the updates completely comply to graph topology, then we have $a_i = \frac{OutDegree_i}{|E|}$, i.e. a_i is proportional to the i_{th} vertex's out degree. Although our implementation and the above analysis is based on the second occasion, evaluation in our paper shows even if this assumption is broken, i.e. if updates are completely different from our expectation, hopscotch CSR can still produce a high throughput.

References

[1] D. E. Knuth. *The art of computer programming: sorting and searching*, volume 3. Pearson Education, 1997.