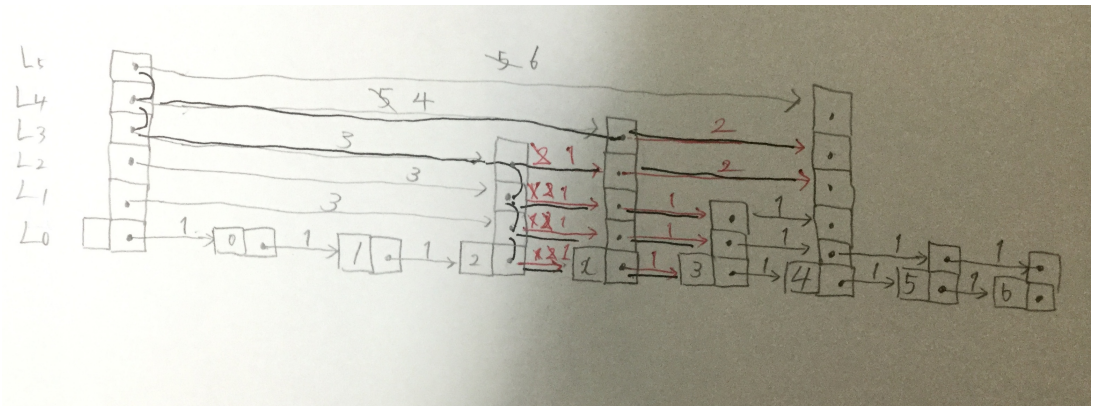
情報数理科学第３回課題　08-192025 林橘平

Ex 4-5

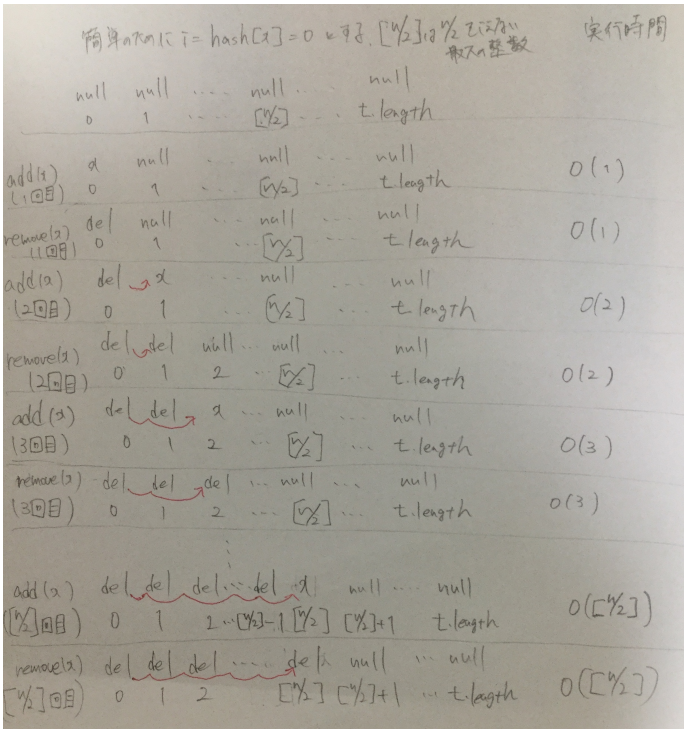


add(3,x)では、まずxを要素にもつ新たなノードwを作成する。この時wの高さ(w->height)はpickHeight()で決まる。この場合は4である。w->heightとスキップリストの高さhを比べてhの方が小さければ、hにw->heightを代入する。(今回はh=5, w->height=4であるためこれは実行されない。)
次に、ノードwをスキップリストに挿入する。kにw->height(=4)を代入し、i=3の探索経路をたどる。L5から順に探索する。5 > kであるのでL4に移動する。L4は、4 <= kであるのでL4にwを継ぎ足す。L4のsentinelノードuで、辺u.next[4]の長さを1つ大きくする。(u->length[4]++;) w->next[4] = u->next[4]; u->next[4] = w;でwをL4に挿入する。 w->length[4] = u->length[4] - (3 - 1); u->length[r] = 3 - 1; で、　L４の辺の長さをwの挿入に応じて変更する。 これをr=4 ~ 0まで実行することでadd(3, x)が実行される。

Ex 5-5

add(x),remove(x)を１回ずつやる操作を1ペアとして、それを[n/2]回やる操作を考える。([n/2]はn/2を超えない最大の整数)。これはO(n)個の操作である。この時LinearHashTableは全ての要素がnullであり、i=hash(x)=0であると仮定する。 1回目のadd(x)では、i=0よりt[0]から順にnullのエントリーを探す。今t[0]はnullなのでそこにxを入れる。その後remove(x)では同様にしてt[0]にdelを入れる。期待実行時間はそれぞれO(1)。
2回目のadd(x)でもt[0]から順にnullのエントリーを探す。t[0]はdelのエントリーなので、t[1]にxを入れる。remove(x)では同様にしてt[1]にdelを入れる。期待実行時間はそれぞれO(2)。
3回目以降も同様に、a回目のadd(x)では、まずt[0]から順にnullのエントリーを探していく。t[0] ~ t[a-2]まではdellのエントリーなので、nullエントリーであるt[a-1]にxを入れる。remove(x)も同様に辿ってt[a-1]にdelを入れ

る。期待実行時間はそれぞれO(a)。 以下の図は上で説明した操作をまとめたものである。



よって操作回数が増えるにつれて期待実行時間は増えていくことがわかる。この操作全体の実行時間は、$$\sum_{i=1}^{n/2}i*2 = \frac{(2+n)(n/2)}{2} = \frac{n^2+2n}{4}$$ より、O(n^2)である。 一方で簡略化する前の、delエントリーを探すadd(x)では、常にt[0]にxを入れるため期待実行時間はadd,remove共に常にO(1)である。そのため操作全体の実行時間はO(n)である。
よってaddSlowでは実行が非常に遅いということがわかる。 これは、addSlow(x)ではdelエントリーを探さず、nullエントリーのみを探しているからである。delエントリーを探すようにすれば、実行のたびにいちいちnullエントリーを探す動作を繰り返す必要がない。