```
// Bit visitado
bool insere(int k){
int chave = hash(k);
bool pego = false;
int posM = -1;
while(T[chave] != -1 && T[chave] !=
k){
if(!pego && T[chave]==-2){
posM = chave;
pego = true;
}
if(BitVisitado[chave] == 0)
BitVisitado[chave] = 1;
chave = rehash(chave);
}
if(T[chave] == -1){
if(pego)T[posM] = k;
else T[chave] = k
return true;
}
return false;
}
bool busca(int k){
int chave = hash(k);
while(T[chave] != k &&
BitVisitado[chave]==1){
chave = rehash(chave);
}
if(T[chave] == k) return true;
return false;
}
bool remove(int k){
/*
Créditos a Karina Santos
*/
int i, anterior;
i = hash(k);
anterior = -1;
while(T[i]!=k && BitVisitado[i]==1){
anterior = i;
i = rehash(i);
}
if(T[i]!=k)
return false;
T[i] = -2;
if(BitVisitado[i]==0 && anterior!=-
1)
BitVisitado[anterior] = 0;
return true;
}

// Metodos Hash
int Divisao(int k){
return k%TS;
}
int Multiplicacao(int k){
double c = 0.6180339887; //
constante definida pelo usuario
double aux = k*c;
double aux2 = (double) (int) temp;
double fracao = aux-aux2;
double chave = TS*fracao;
return ((int)chave)%TS;
}
int QuadradoMedio(int k){
double aux = (double)k;
double c = pow(8,7) // constante
definida pelo usuario
double chave = ((aux*aux)/c)%TS;
return (int)chave;
}
int AlfaNumerico(char * chave){
int length = (int)strlen(chave);
double result;
for(int i = 0;i<length;i++){
int temp = chave[i];
result += temp*pow(26,(length-1-i));
}
return (int)result;
}
int Dobra(int key){
/*
Créditos ao Henrique Leme
*/
int n =
(int)(log((double)TS)/log(2.0))+1;
int p = pow(2,n)-1;
int d = 0;
int s = sizeof(int)*8;
while (s>0) {
d=d^(key&p);
key=key>>n;
s=s-n;
}
return ((int)d)%TS;}
```

```
// Btree
int BinarySearch(NO* p, int x){
int ini = 0;
int fim = p->numElementos;
while(ini<=fim){
int meio = (ini+fim/2);
if(p->chave[meio]==x) return meio;
if(p->chave[meio]>x){
fim = meio-1;
}
else ini = meio+1
}
return ini;
}
bool Search(NO* p, int x){ // metodo
Recursivo
if(!p) return false;
int index = BinarySearch(p,x);
if(p->chave[index]==x)return true;
return Search(p->filho[index],x);
}
bool Search(NO* p, int x){ // metodo
não-Recursivo
while(p){
index = BinarySearch(p,x);
if(p->chave[index]==x) return true;
else p = p->filho[index];
}
return false;
}
void imprime(NO* p){
// IMPRIME BTREE EM ORDEM-CRESCENTE
if(!p)return;
int i = 0;
while(i<p->numElementos){
imprime(p->filhos[i]);
printf("%d ", p->chave[i]);
i++;
}
imprime(p->filhos[i]);
}


-


// Lista Generalizada
void remove(NO* atual){ // recursivo
if(!atual) return ;
atual->ref--;
remove(atual->prox);
if(atual->ref==0){
if(atual->info==2)remove(atual-
>listainfo);
free(atual);
}
}
bool Mark(){ // Marca todos os nós
que precisam ser removidos, não-
recursivamente
for(int i=0;i<MAX;i++){
Nodes[acc[i]].mark = true;
}
int i = 0;
int j;
while(i<NumNodes){
j = i+1;
if(Nodes[i].mark){
if(Nodes[i].type == 2 &&
Nodes[Nodes[i].info].mark == false){
Nodes[Nodes[i].info].mark = true;
if(Nodes[i].info<j){
j=Nodes[i].info;
}
}
if(Nodes[i].next>0 &&
Nodes[Nodes[i].next].mark == false){
Nodes[Nodes[i].next].mark = true;
}
if(Nodes[i].next<j){
j = Nodes[i].next;
}
}
i = j;
}
}
```

```
// Predictor
int bool insert(int k){
int ant = -1;
int chave = hash(k);
if(T[chave]==k)return false;
if(T[chave]==-1){
T[chave]= k;
return true;
}
else{
bool pego = false;
int auxC = -1;
while(T[chave]!=-1&&T[chave]!=k){
if(!pego){
if(T[chave]==-2){
auxC = chave;
pego = true;
}
if(predictor[chave]==0 && !pego){
ant = chave;
pego = true;
}
}
if(predictor[chave]==0)chave =
rehash(chave);
else chave = (predictor[chave] +
chave)%TS;
}
if(T[chave]==-1){
if(auxC==-1){
T[chave] = k;
predictor[ant] = chave - ant;
}
else T[auxC] = k;
return true;
}
else return false;
}
}
bool search(int k){
int chave = hash(k);
if(T[chave]== k)return true;
while(predictor[chave]!=0){
chave = (chave +
predictor[chave])%TS;
if(T[chave] == k) return true;
}
return false;
}
bool remove(int k){


int chave = hash(k);


if(T[chave]==k){
T[chave]=-2;
return true;}
int ant;
while(predictor[chave]!=0){
ant = chave;
chave = (chave +
predictor[chave])%TS;
if(T[chave]==k){
T[chave]=-2;
if(predictor[chave]==0){
predictor[ant] = 0;}
return true;}}
return false;}
```

Inserção H ordenado

```
int insert(int key) {
    bool first = true;
    int i, j, newK, tmp, tk;
    i = h(key);
    newK = key;
    while (T[i] > newK) {
        i = rh(i);
    }
    tk = T[i];
    while ((tk != -1) && (tk != newk)) {
        tmp = T[i];
        if (newk > tk) {
            T[i] = newk;
            newk = tmp;
            if (first) {
                j = i;
                first = false;
            }
        }
        i = rh(i);
        tk = T[i];
    }
    if (tk == -1) T[i] = newk;
    if (first) return i;
    else return j;
}
```

Insert encadeado

```
bool insere(int k) {
    int chave;
    chave = h(k);
    if (T[chave] == -1) {
        T[chave] = k;
        return true;
    }
    while ((P[chave] != -1) && (T[chave] != k)) {
        chave = P[chave];
        if (T[chave] == k) return false;
    }
    T[marcador] = k;
    P[chave] = marcador;
    while (T[marcador] != -1) marcador--;
    return true;
}
```

Remove encadeado

```
bool remove(int k) {
    int chave = h(k);
    if (T[chave] == k) {
        if (P[chave] == -1) {
            T[chave] = -1;
            return true;
        }
        else {
            T[chave] = T[P[i]];
            T[P[chave]] = -1;
            P[chave] = P[P[chave]];
            P[P[chave]] = -1;
        }
    }
    while ((P[chave] != -1) && (T[P[chave]] == k)) {
        chave = P[chave];
    }
    if (P[chave] == -1) return false;
    T[P[chave]] = -1;
    P[chave] = P[P[chave]];
    P[P[chave]] = -1;
    return true;
}
```

Remove ordenado

```
bool remove(int k) {
    int i, tmp;
    i = h(k);
    while (T[i] > k) {
        i = rh(i);
    }
    if (T[i] < k) return false;
    T[i] = -1;
    i = rh(i);
    while (T[i] != -1) {
        tmp = T[i];
        T[i] = -1;
        insere(tmp);
        i = rh(i);
    }
    return true;
}
```

-