

Programa F.8 Refinamento final do algoritmo Insere

```

void Ins(TipoRegistro Reg, TipoApontador Ap, short *Cresceu,
        TipoRegistro *RegRetorno, TipoApontador *ApRetorno)
{
    long i = 1; long j;
    TipoApontador ApTemp;
    if (Ap == NULL)
    {
        *Cresceu = TRUE; (*RegRetorno) = Reg; (*ApRetorno) = NULL;
        return;
    }
    while (i < Ap->n && Reg.Chave > Ap->r[i-1].Chave) i++;
    if (Reg.Chave == Ap->r[i-1].Chave)
    {
        printf(" Erro: Registro ja esta presente\n"); *Cresceu = FALSE;
        return;
    }
    if (Reg.Chave < Ap->r[i-1].Chave) i--;
    Ins(Reg, Ap->p[i], Cresceu, RegRetorno, ApRetorno);
    if (!*Cresceu) return;
    if (Ap->n < MM) /* Pagina tem espaco */
    {
        InsereNaPagina(Ap, *RegRetorno, *ApRetorno);
        *Cresceu = FALSE;
        return;
    }
    /* Overflow: Pagina tem que ser dividida */
    ApTemp = (TipoApontador) malloc(sizeof(TipoPagina));
    ApTemp->n = 0; ApTemp->p[0] = NULL;
    if (i < M + 1)
    {
        InsereNaPagina(ApTemp, Ap->r[MM-1], Ap->p[MM]);
        Ap->n--;
        InsereNaPagina(Ap, *RegRetorno, *ApRetorno);
    }
    else InsereNaPagina(ApTemp, *RegRetorno, *ApRetorno);
    for (j = M + 2; j <= MM; j++)
        InsereNaPagina(ApTemp, Ap->r[j-1], Ap->p[j]);
    Ap->n = M; ApTemp->p[0] = Ap->p[M+1];
    *RegRetorno = Ap->r[M]; *ApRetorno = ApTemp;
}

void Insere(TipoRegistro Reg, TipoApontador *Ap)
{
    short Cresceu;
    TipoRegistro RegRetorno;
    TipoPagina *ApRetorno, *ApTemp;
    Ins(Reg, *Ap, &Cresceu, &RegRetorno, &ApRetorno);
    if (Cresceu) /* Arvore cresce na altura pela raiz */
    {
        ApTemp = (TipoPagina *) malloc(sizeof(TipoPagina));
        ApTemp->n = 1;
        ApTemp->r[0] = RegRetorno;
        ApTemp->p[1] = ApRetorno;
        ApTemp->p[0] = *Ap; *Ap = ApTemp;
    }
}

```

Programa F.9 Função Retira

```

void Reconstitui(TipoApontador ApPag, TipoApontador ApPai,
                int PosPai, short *Diminiu)
{
    TipoPagina *Aux; long DispAux, j;
    if (PosPai < ApPai->n) /* Aux = TipoPagina a direita de ApPag */
    {
        Aux = ApPai->p[PosPai+1]; DispAux = (Aux->n - M + 1) / 2;
        ApPag->r[ApPag->n] = ApPai->r[PosPai];
        ApPag->p[ApPag->n + 1] = Aux->p[0]; ApPag->n++;
        if (DispAux > 0) /* Existe folga: transfere de Aux para ApPag */
        {
            for (j = 1; j < DispAux; j++)
                InsereNaPagina(ApPag, Aux->r[j-1], Aux->p[j]);
            ApPai->r[PosPai] = Aux->r[DispAux-1]; Aux->n -= DispAux;
            for (j = 0; j < Aux->n; j++) Aux->r[j] = Aux->r[j + DispAux];
            for (j = 0; j <= Aux->n; j++) Aux->p[j] = Aux->p[j + DispAux];
            *Diminiu = FALSE;
        }
    }
    else /* Fusao: intercala Aux em ApPag e libera Aux */
    {
        for (j = 1; j <= M; j++)
            InsereNaPagina(ApPag, Aux->r[j-1], Aux->p[j]);
        free(Aux);
        for (j = PosPai + 1; j < ApPai->n; j++)
        {
            ApPai->r[j-1] = ApPai->r[j];
            ApPai->p[j] = ApPai->p[j+1];
        }
        ApPai->n--;
        if (ApPai->n >= M) *Diminiu = FALSE;
    }
}

else /* Aux = TipoPagina a esquerda de ApPag */
{
    Aux = ApPai->p[PosPai-1]; DispAux = (Aux->n - M + 1) / 2;
    for (j = ApPag->n; j >= 1; j--) ApPag->r[j] = ApPag->r[j-1];
    ApPag->r[0] = ApPai->r[PosPai-1];
    for (j = ApPag->n; j >= 0; j--) ApPag->p[j+1] = ApPag->p[j];
    ApPag->n++;
    if (DispAux > 0) /* Existe folga: transf. de Aux para ApPag */
    {
        for (j = 1; j < DispAux; j++)
            InsereNaPagina(ApPag, Aux->r[Aux->n - j],
                          Aux->p[Aux->n - j + 1]);
        ApPag->p[0] = Aux->p[Aux->n - DispAux + 1];
        ApPai->r[PosPai-1] = Aux->r[Aux->n - DispAux];
        Aux->n -= DispAux; *Diminiu = FALSE;
    }
}
else /* Fusao: intercala ApPag em Aux e libera ApPag */
{
    for (j = 1; j <= M; j++)
        InsereNaPagina(Aux, ApPag->r[j-1], ApPag->p[j]);
    free(ApPag); ApPai->n--;
    if (ApPai->n >= M) *Diminiu = FALSE;
}
}
}

```