Lista com saltos - generalidades

- Foram propostas em 1989 por William Pugh, professor da Universidade de Maryland;
- Permitem operações de consulta, inserção e remoção mais eficientes do que as listas ligadas simples e duplas;
- O aumento na eficiência é, no entanto, obtido à custa de um aumento de recursos (memória) necessários.
- A ideia base é a seguinte:
  - Procurar o nome de "Manuel Silva" numa lista de nomes ordenados alfabeticamente, que constam num dado documento;
  - Ao abrir o documento, verificamos que estamos apenas nos nomes cuja inicial é "A";
  - Acção inteligente:
    - Avançar 1 página (provavelmente avança-se para uma página com a mesma letra) - ERRADO
    - Avançar N páginas! (provavelmente avança-se para a letra h?, j?) – CORRETO

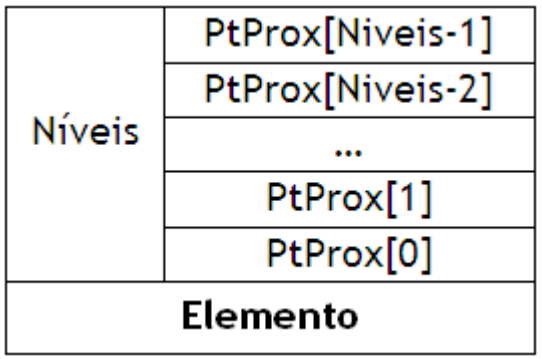
Lista com saltos - definição

- É uma lista ligada simples
  - em que cada nodo tem um número variável de *ligações*,
  - sendo que cada nível de *ligação* implementa uma lista ligada simples formada por um número diferente de nodos.
- Está-se perante uma estrutura de listas paralelas,
  - todas elas terminadas com o ponteiro nulo (NULL),
  - que permitem implementar diferentes travessias,
  - sendo que cada travessia avança sobre os nodos com um número de *ligações* inferior ao nível em que ela decorre.
- Quanto maior é o nível da travessia,
  - maior é o espaçamento de nodos percorridos,
  - sendo que uma travessia pode descer de nível para prosseguir de uma forma menos espaçada ao longo da lista.
- Logo, está-se perante uma lista ligada com atalhos.
- Permite obter travessias mais eficientes do que a pesquisa sequencial,
  - apesar da travessia ao longo de um nível ser sequencial,
  - o facto dos níveis terem espaçamentos diferentes, tanto maiores quanto maior é o nível.

Lista com saltos - implementação

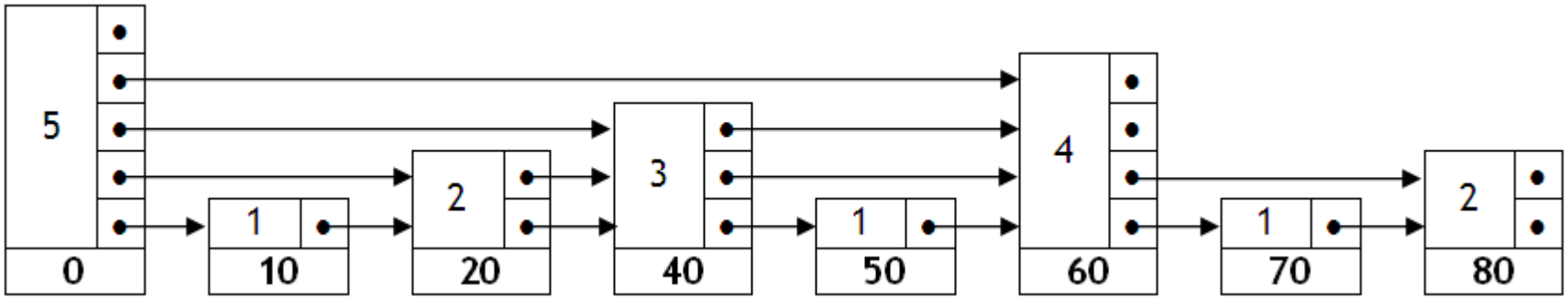
- Várias forma de implementar uma lista com saltos, dependendo da forma como se armazenam os ponteiros para cada nível:
  - "arrays" estáticos,
  - "arrays" dinâmicos,
  - listas ligadas.
- Vai-se usar os "arrays" dinâmicos:

```
typedef  struct  Nodo  *PNodo;
struct  Nodo {
    Info      Elemento;
    int       Niveis;
    PNodo     *PtProx;
};
```



Lista com saltos - implementação

- Representação de uma lista com saltos:



  - O primeiro nodo/elemento é o se encontra à cabeça da lista (nodo extra):
    - 0 (valor do elemento do nodo, que é o menor da lista e não faz parte da lista),
    - 5 (quantidade de níveis do nodo, que é o máximo de níveis de cada nodo pode ter),
    - 5 ponteiros para nodos seguintes.

Lista com saltos – operações

- Uma lista com saltos caracteriza-se pelos seguintes parâmetros:
  - ***cabeça da lista*** (PCab) que é um nodo extra e que contém um valor sempre inferior a qualquer um dos valores contidos nos restantes nodos da lista;
  - ***máximo de níveis*** da lista (MaxNiv), que corresponde ao número de níveis da cabeça.

Lista com saltos - operações

- Criar um nodo

Entrada: um elemento X e o seu nível (niv)

Saída: um ponteiro para um nodo (informação + ligações)

```
PNodo CriarNodo (Info X, int niv) {
    int  k;
    PNodo P;
    P = (PNodo) malloc (sizeof (struct Nodo));
    if (P == NULL)
        return  NULL;
    P→PtProx = (PNodo *) malloc (niv * sizeof (PNodo));
    if (P→PtProx == NULL)
        return NULL;
    P→Elemento = X;
    P→Niveis = niv;
    for (k = 0; k < niv; k++)
        P→PtProx[k] = NULL;
    return  P;
}
```

Lista com saltos – operações

- Determinar o nível dum elemento (nodo) duma lista
  - Para se obter uma pesquisa com eficiência logaritmica, O(log n), a lista deve ter listas ligadas com espaçamentos do tipo 2^{MaxNiv}.
  - Uma possível função é a seguinte:

    Entrada: nível máximo a considerar (maxNiv)

    Saída: o nível do elemento (gerado aleatoriamente)

```
int GerarNivelAleatorio (int maxNiv) {
    int  i, Niv = maxNiv;
    float  t, fator = 1/(pow(2,maxNiv)-1);   // probabilidade do menor nível
    srand(time(NULL));
    t = rand();
    for (i = 0; i < maxNiv-1; i++) {
        if (t <= pow(2,i)*fator)
            return Niv;
        Niv--;
    }
    return 1;
}
```

- Criar uma lista
  - consiste em criar um nodo associado à cabeça da lista, cujos campos são os seguintes:
    - Elemento = X, tal que X é menor do que qualquer elemento da lista;
    - Niveis = MaxNiv;
    - PtProx[k] = NULL, k = 0,…,MaxNiv-1
  - Uma possível função é a seguinte:

    Entrada: um elemento X (X menor da lista) e máximo de níveis a considerar (MaxNiv)

    Saída: um ponteiro para um nodo (cabeça da lista)

```
PNodo CriarLista (Info X, int maxNiv) {
    PNodo P;
    P = CriarNodo(X, maxNiv);
    return  P;
}
```

Lista com saltos – operações

- Libertar/destruir um nodo de uma lista
  - consiste em atribuir a todos os ponteiros dos campos associados aos nodos seguintes de cada nível o ponteiro nulo (NULL),
  - libertar para o sistema o ponteiro para a sequência daqueles ponteiros e, por fim,
  - libertar o ponteiro para o nodo a destruir.
- Uma possível função é a seguinte:

Entrada: um ponteiro para o nodo que se pretende destruir

Saída: o ponteiro a apontar para NULL

```
PNodo LibertarNodo (PNodo P) {
    int  k;
    for (k = 0; k < P→Niveis; k++)
        P→PtProx[k] = NULL;
    free(P→PtProx);
    free(P);
    P = NULL;
    return P;
}
```

Lista com saltos – operações

- Destruição de uma lista
  - a travessia da lista é feita no nível zero, para destruir todos os seus nodos, ficando apenas a cabeça da lista, de forma sequencial.
  - Uma possível função é a seguinte:

Entrada: um ponteiro para a cabeça da lista

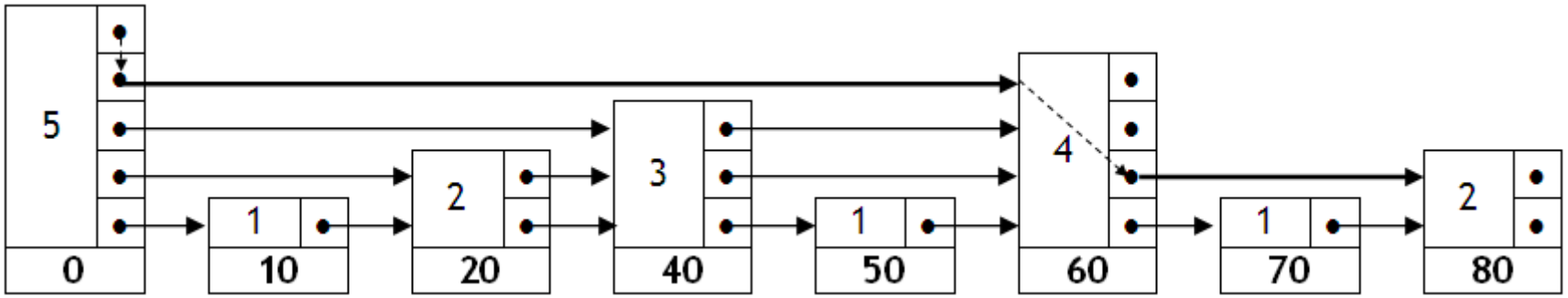
Saída: a lista vazia (cabeça da lista com todos os seus ponteiros seguintes nulos)

```
void DestruirLista (PNodo PCab) {
    int  k;
    PNodo PAux, P = PCab→PtProx[0];
    while (P != NULL) {
        PAux = P;
        P = P→PtProx[0];
        PAux = LibertarNodo(PAux);
    }
    for (k = 0; k < PCab→Niveis; k++)
        PCab→PtProx[k] = NULL;
}
```

Lista com saltos - operações

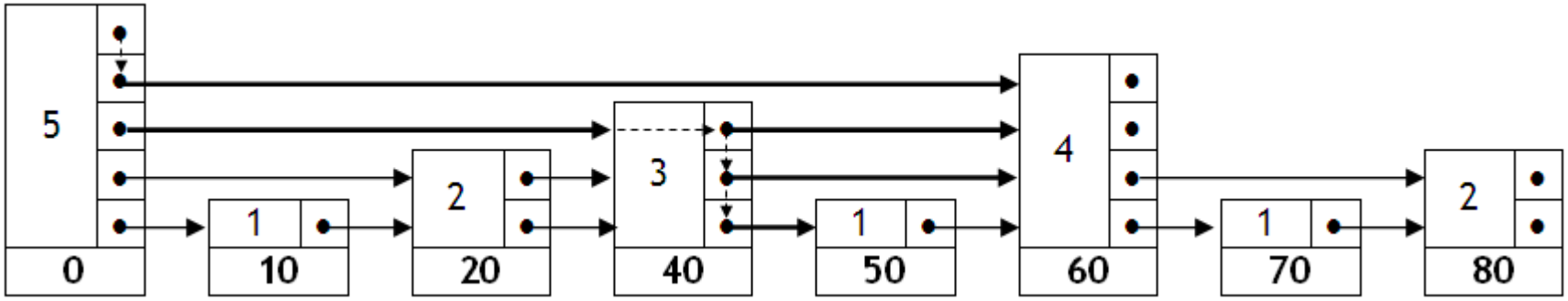
- Pesquisa dum elemento numa lista
  - A pesquisa de um elemento segue a seguinte estratégia:
    - começar no ponteiro de maior nível na cabeça da lista; percorrer até ao nodo que seja igual ou que exceda o elemento que se procura;
    - se encontro um elemento igual, terminar com sucesso; senão, descer de nível e usar a mesma estratégia;
    - se depois de atingir o nível 0 não encontrar o nodo com o elemento procurado, então concluir que ele não está presente.

  - Exemplo: pesquisar o elemento com o valor 80



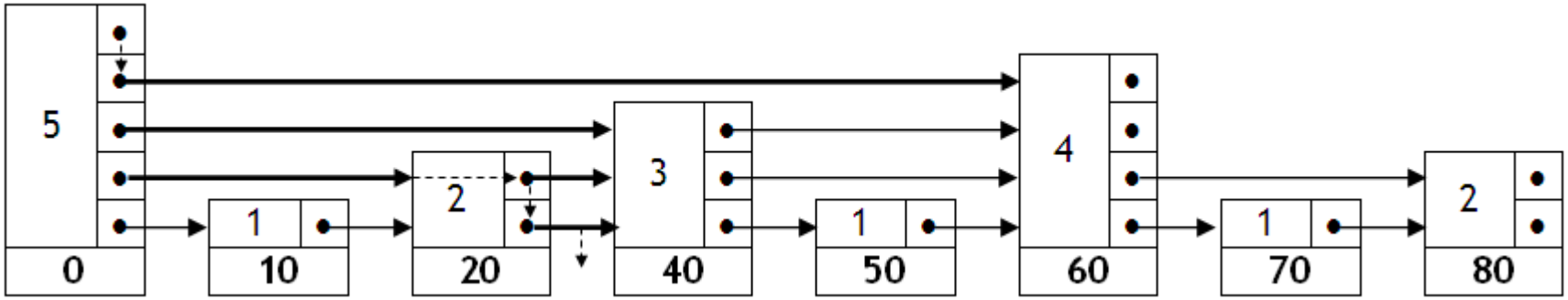
Lista com saltos - operações

- Pesquisa dum elemento numa lista
  - Exemplo: pesquisar o elemento com o valor 50



Lista com saltos - operações

- Pesquisa dum elemento numa lista
  - Exemplo: pesquisar o elemento com o valor 30



Lista com saltos – operações

- Pesquisa dum elemento numa lista
  - Uma possível função é a seguinte:

Entrada: um ponteiro para a cabeça da lista e o elemento a pesquisar

Saída: 1 (elemento está na lista); 0 (elemento não está na lista)

```
int PesquisarElemento (PNodo PCab, Info X) {
    int k;
    PNodo P = PCab;
    for (k = PCab→Niveis; k > 0; k--) {
        while (P→PtProx[k-1] != NULL && (P→PtProx[k-1])→Elemento < X)
            P = P→PtProx[k-1];
        if (P→PtProx[k-1] != NULL && (P→PtProx[k-1])→Elemento == X)
            return 1;
    }
    return 0;
}
```

Lista com saltos – operações

- Pesquisa do nodo anterior dum elemento duma lista
  - As operações de inserir e remover elemento duma lista precisam de determinar quais os ponteiros para cada nível dos nodos anteriores ao elemento a inserir (quando este for colocado na lista) ou a remover.

Lista com saltos – operações

- Pesquisa do nodo anterior dum elemento duma lista
  - Uma possível função é a seguinte (pré-requisito da existência do elemento na lista):

Entrada: um ponteiro para a cabeça da lista e o elemento a pesquisar

Saída: sequência de ponteiros para os elementos anteriores ao elemento a procurar

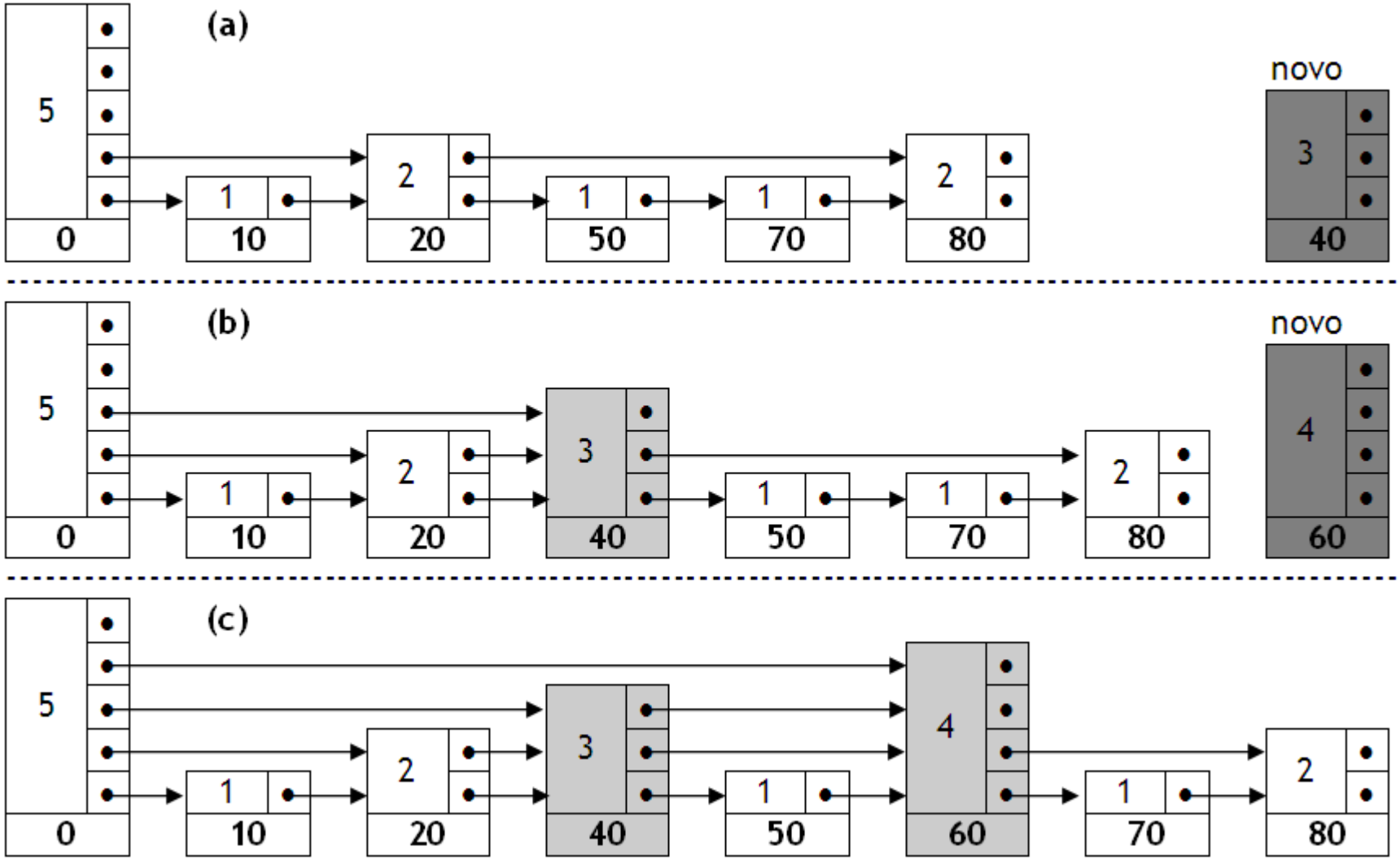
```
PNodo PesquisarAnterior (PNodo PCab, Info X) {
    int k;
    PNodo *PAnt, P = PCab;
    PAnt = (PNodo *) malloc(PCab→Niveis * sizeof(PNodo));
    if (PAnt == NULL)
        return NULL;
    for (k = PCab→Niveis; k > 0; k--) {
        while (P→PtProx[k-1] != NULL && (P→PtProx[k-1])→Elemento < X)
            P = P→PtProx[k-1];
        PAnt[k-1] = P;
    }
    return PAnt;
}
```

Lista com saltos – operações

- Inserção de um elemento numa lista
  - implica a atualização de todas as listas ligadas que são intercetadas pelo nodo a inserir;
  - segue a seguinte estratégia:
    1. alocar o novo nodo;
    2. escolher aleatoriamente o nível do elemento de acordo com a distribuição de probabilidade;
    3. determinar quais os nodos anteriores, para cada nível, ao elemento a inserir (após colocação do novo elemento na lista);
    4. inserir o novo nodo na lista.
  - o algoritmo proposto
    - realiza uma travessia começando na cabeça da lista e no nível máximo,
    - até chegar ao local de inserção e ao nível 1, sendo nesta altura que o processo de atualização das ligações se inicia;
    - só após a conclusão daquelas ligações é que o nodo é efetivamente inserido e o algoritmo termina.
    - Os ponteiros para cada nível (associados a cada lista ligada) determinam o local de inserção do novo nodo (isto é, o seu nodo anterior), fazendo depois as ligações do novo nodo ao nodo seguinte do seu nodo anterior, e do nodo anterior ao novo nodo.

Lista com saltos - operações

- Inserção de um elemento numa lista
  - Exemplo: inserção do elemento 40 (3 níveis) e posterior inserção do elemento 60 (4 níveis)



Lista com saltos – operações

- Inserção de um elemento numa lista - uma possível função é a seguinte:

Entrada: um ponteiro para a cabeça da lista e o elemento a inserir

Saída: a lista atualizada

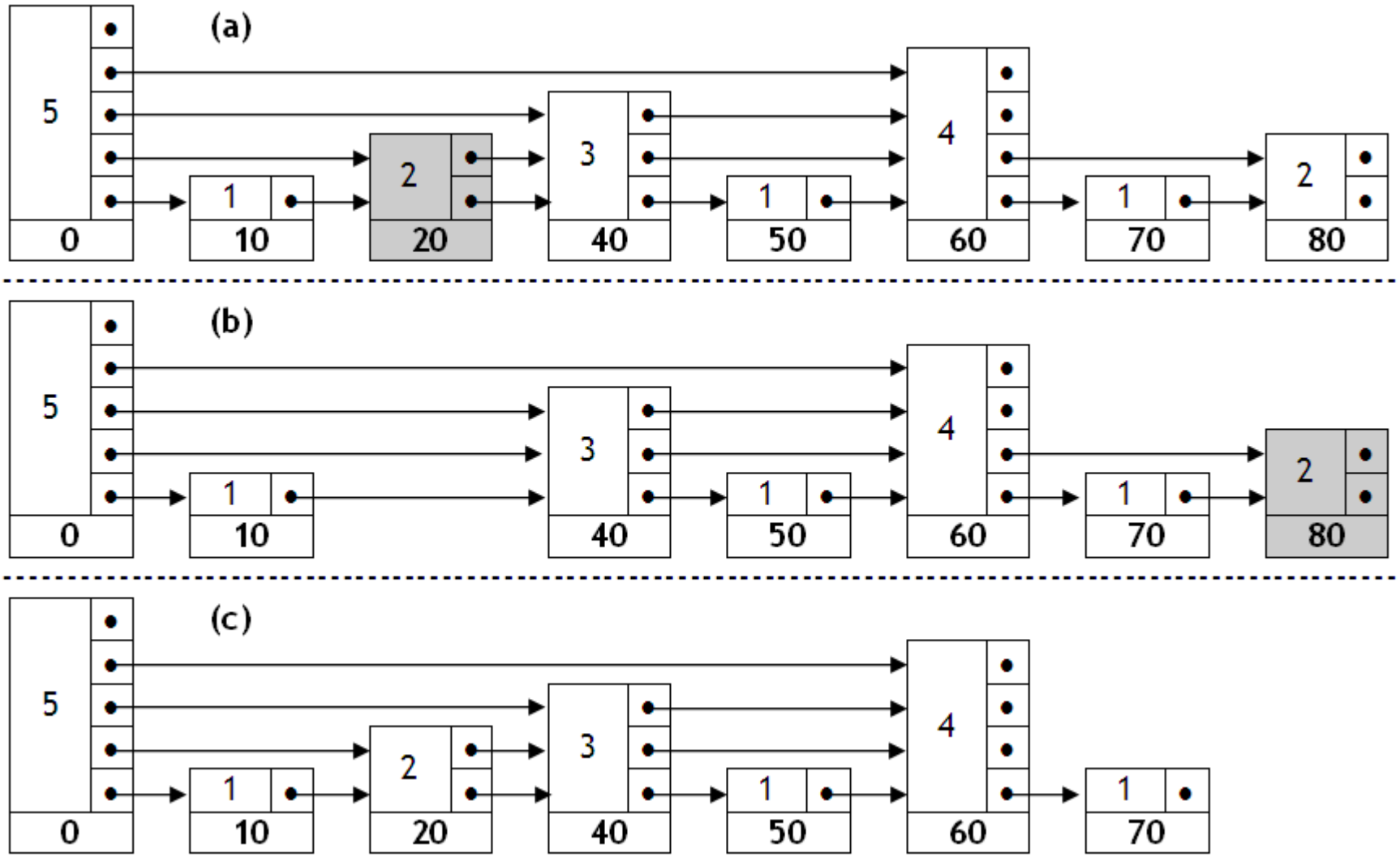
```
void InserirElemento (PNodo PCab, Info X) {
    PNodo P, *PAnt;
    int  Nivel, k;
    if (PCab→Elemento < X)      // o valor da cabeça da lista tem de ser o menor
        PCab→Elemento = X – 100;  // atualizar o valor da cabeça da lista
    Nivel = GerarNivelAleatorio(MaxNiv);
    P = CriarNodo(X, Nivel);
    if (P == NULL)   // criação de nodo sem sucesso - lista sem alteração
        return PCab;
    PAnt = ProcurarAnterior(PCab, X);
    for (k = Nivel; k > 0; k--) {
        P→PtProx[k-1] = PAnt[k-1]→PtProx[k-1];
        PAnt[k-1]→PtProx[k-1] = P;
    }
}
```

Lista com saltos – operações

- Remoção de um elemento da lista
  - implica a atualização de todas as listas ligadas que são intercetadas pelo nodo a removar;
  - segue a seguinte estratégia:
    1. determinar quais os nodos anteriores, para cada nível, do elemento a remover;
    2. removar o nodo da lista.
  - o algoritmo proposto
    - realiza uma travessia pela lista, começando na cabeça da lista e no nível máximo,
    - até chegar ao nível 1 do nodo a remover, sendo nesta altura que o processo de atualização das ligações se inicia;
    - só após a conclusão daquelas ligações é que o nodo é efetivamente removido e o algoritmo termina.
    - Os ponteiros para cada nível (associados a cada lista ligada) determinam os nodos anteriores ao nodo a remover, fazendo depois as ligações destes nodos (anteriores) com os nodos seguintes do nodo a remover, e a libertação do nodo que se pretende remover. No caso de haver elementos

Lista com saltos - operações

- Remoção de um elemento da lista
  - Exemplo: remoção do elemento 20 (2 níveis) e posterior remoção do elemento 80 (2 níveis)



Lista com saltos – operações

- Remoção de um elemento da lista
  - Uma possível função é a seguinte:

Entrada: um ponteiro para a cabeça da lista e o elemento a remover

Saída: a lista atualizada

```
void RemoverElemento (PNodo PCab, Info X) {
    PNodo P, *PAnt;
    int  Nivel, k;
    PAnt = ProcurarAnterior(PCab, X);
    P = PAnt[0]→PtProx[0];  // nodo a remover
    Nivel = PAnt→Niveis;
    for (k = Nivel; k > 0; k--)
        PAnt[k-1]→PtProx[k-1] = P→PtProx[k-1];
    P = LibertarNodo(P);
}
```