Estruturas de Dados - Prof. Claudson - 2016.1

Matéria da Prova:

✓ 1) Heap binomial
✓ 2) Heap de Fibonacci
~~3) Heap esquerdista~~
4) Análise amortizada
✓ 5) Conjuntos disjuntos
✓ 6) Tabela hash
✓ 7) Árvore digital / patrícia
✓ 8) Árvore de Huffman

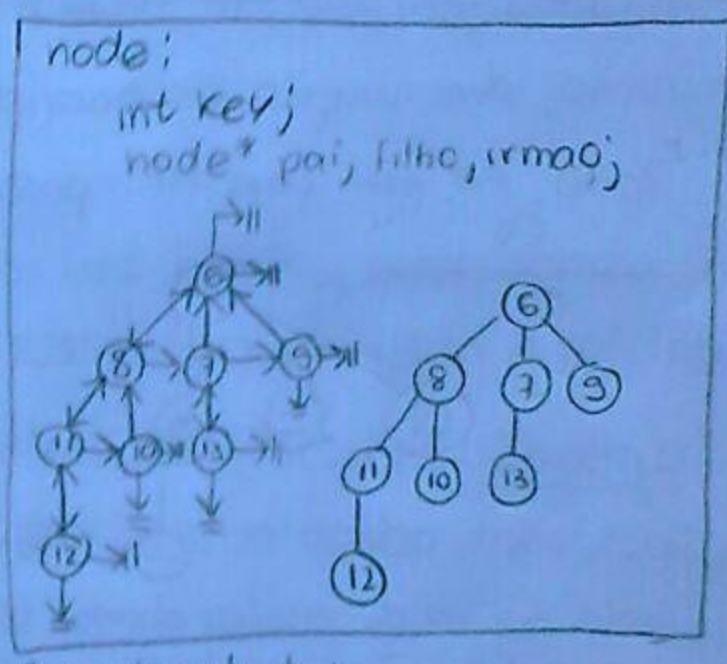
DETONADO

1) Heap Binomial

Ele nunca pediu nada de heap binomial que não fosse escrever o algoritmo de "descer" com sua complexidade em O(n)

```
void descer_bino(node* pt)
    while (pt != NULL){
        node* aux = pt -> filho;
        node* nmin = aux;
        while (aux != NULL){
            if (aux->key < nmin->key)
                nmin = aux;
            aux = aux -> irmao; }
        if (nmin->key < pt->key){
            int tmp = nmin->key;
            nmin->key = pt->key;
            pt->key = tmp;
            pt = nmin; }
        else
            return;
    }
```

```
node:
    int key;
    node* pai, filho, irmao;
```



Complexidade:

Árvore binomial de ordem k possui k nós na altura.

A busca do menor filho custa sempre entre 0 e k.

$k + k = 2k$; $O(k)$

Por precaução:

Remoção de mínimo: itera-se pela lista de raízes da heap e remove-se a menor delas. Forma-se uma heap com as sub-árvores da raiz removida. Faz-se a união entre a heap original de raiz removida e a heap das sub-árvores.

União: unimos árvores binomiais de mesma ordem até isso não ser mais possível.

Inserção: cria-se uma heap com uma árvore binomial de ordem 0 com o novo elemento e une-se ela com a original.

Decréscimo de chave: decrementamos o valor de um nó e subimos ele.

Remoção: decrementa-se o nó até $-\infty$ e removemos o mínimo.

2) Heap de Fibonacci

Ele já pediu: escrever algoritmo de busca de menor chave, inserir elementos em uma HF com e sem avaliação tardia, remoção de mínimo, diminuição de prioridade. É bom entender bem como funciona.

Todo nó em uma HF possui um ponteiro pro seu pai, para um filho ~~de menor chave~~, para seu irmão à esquerda e para seu irmão à direita. A heap sempre contém o ponteiro para a menor raiz.

Inserção: criada árvore de ordem 0 com a chave. Se for com avaliação tardia, adiciona o nó dessa árvore à lista de raízes e a atualiza quanto ao nó de menor chave (caso necessário). Se for sem avaliação tardia, adiciona este nó à lista de raízes e une árvores de ordens iguais tal qual uma heap binomial.

União de heaps: une as listas de raízes e atualiza o nó de menor chave.

Decréscimo de chave: decrementa-se o valor de um nó e fazemos com que ele seja cortado da árvore (perde relações com o pai), adicionando-o à lista de raízes. Se o seu pai estiver marcado, ele também é cortado da árvore, e o pai do pai sofre marcação e assim por diante até chegar em alguém que não foi marcado, que é marcado.

Remoção de mínimo: mínimo é removido. Seus filhos vão para a lista de raízes. Se o mínimo era o único nó na lista de raízes, o menor de seus filhos vira a mínima raiz. Caso contrário, todas as árvores de ordens iguais são mescladas até isso ser impossível.

Exercício: função que encontra o menor filho de um nó em HF.

```
node* menorf (node* pt) {
    node* aux = pt -> filho;
    if (aux == NULL) return NULL;
    while (aux -> prev != NULL)
        aux = aux -> prev;
    node* nmin = aux;
    while (aux -> next != NULL) {
        if (aux -> key < nmin -> key)
            nmin = aux;
        aux = aux -> next; }
    return nmin;
```

← considerando que cada nível da árvore é uma lista duplamente encadeada não-circular. Se for circular, algumas alterações triviais precisam ser realizadas

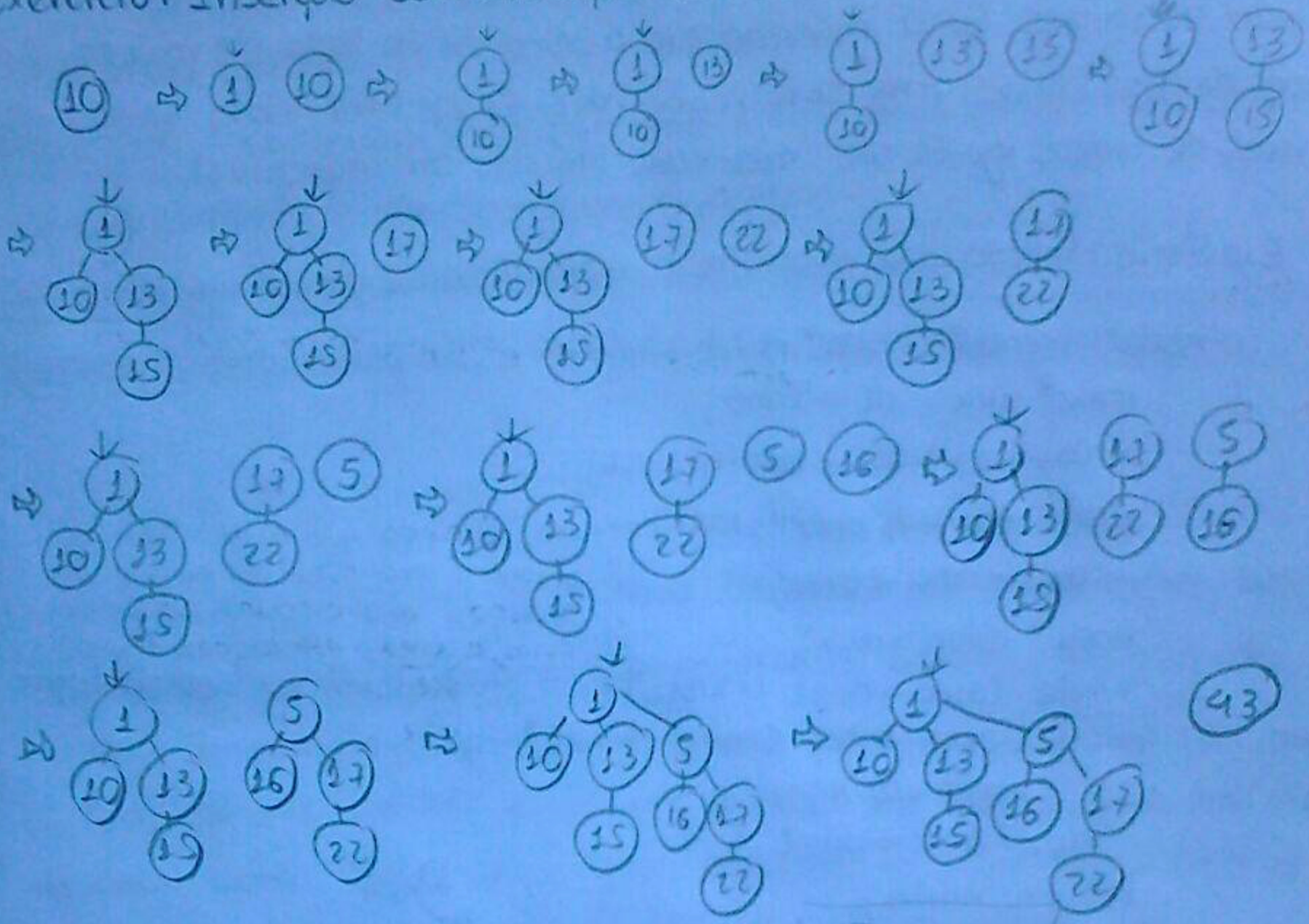
Exercício: usando função acima, função que desce um nó em HF

```
void desce_hf (node* pt)
    node* aux = menorf (pt);
    while (aux != NULL) {
        if (aux -> key < pt -> key) {
            int tmp = aux -> key;
            aux -> key = pt -> key;
            pt -> key = tmp;
            aux = pt; }
        else return;
        aux = menorf (pt);
    }
```

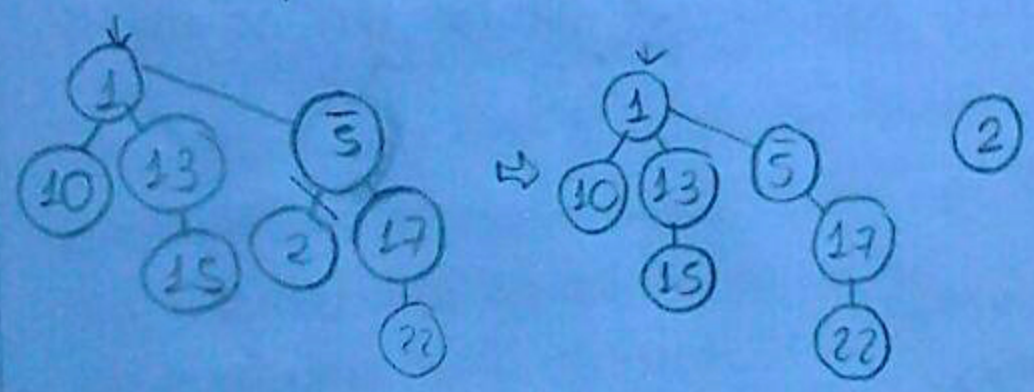
Complexidade:

$O(n)$ por motivos similares ao descer da heap binomial.

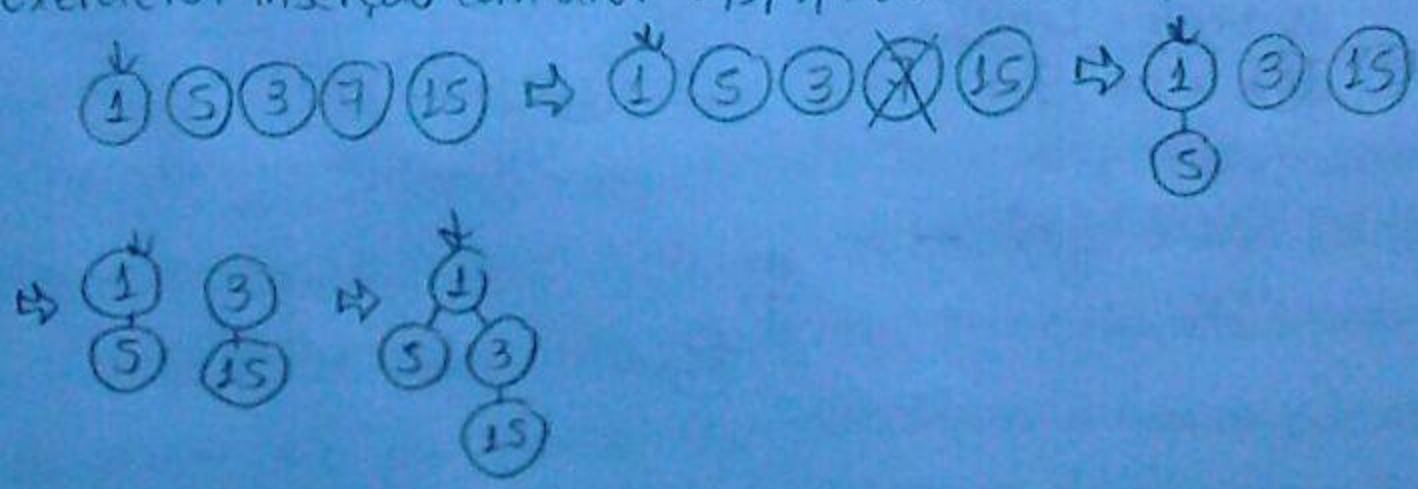
Exercício: Inserção sem avaliação tardia 10, 1, 13, 15, 17, 22, 5, 16 e



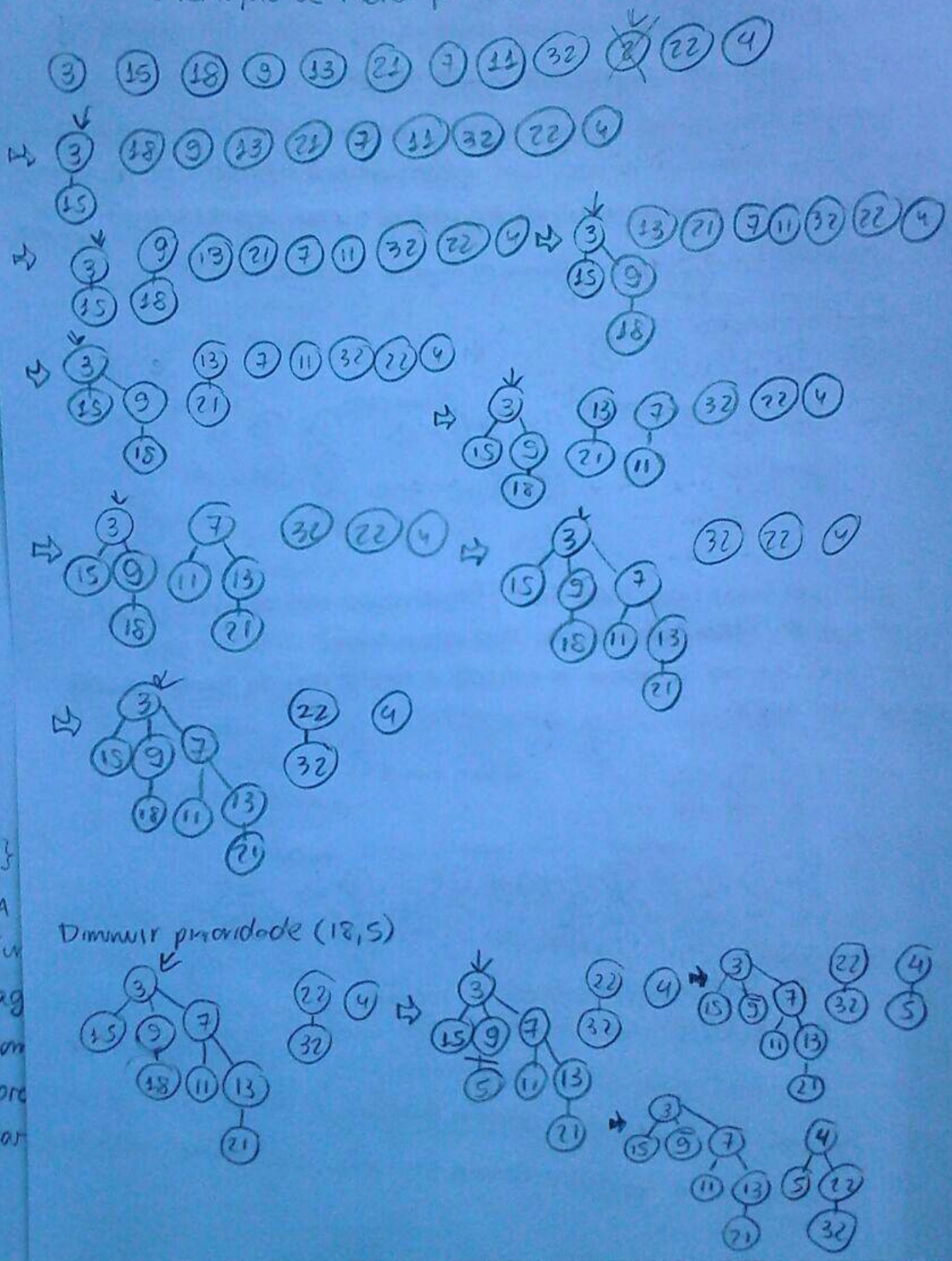
Diminuir prioridade (16, 2)



Exercício: inserção com a.t. 1, 5, 3, 7 e 15 e remoção do 7



...cício: inserção com a.t. de 3, 15, 18, 9, 13, 21, 7, 11, 32, 2, 22 e 4
e remoção de menor prioridade



}

A
fu
ag
com
pre
tar

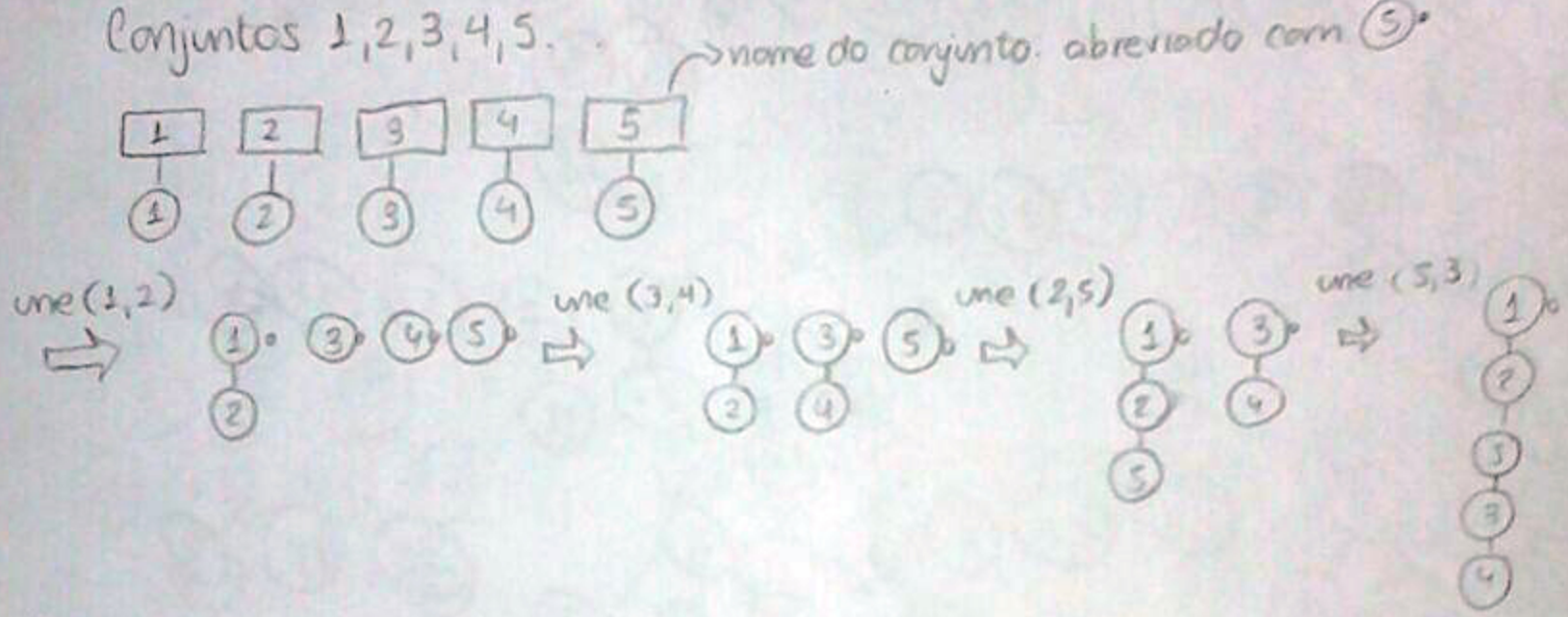
Diminuir prioridade (18, 5)

5) Conjuntos Disjuntos

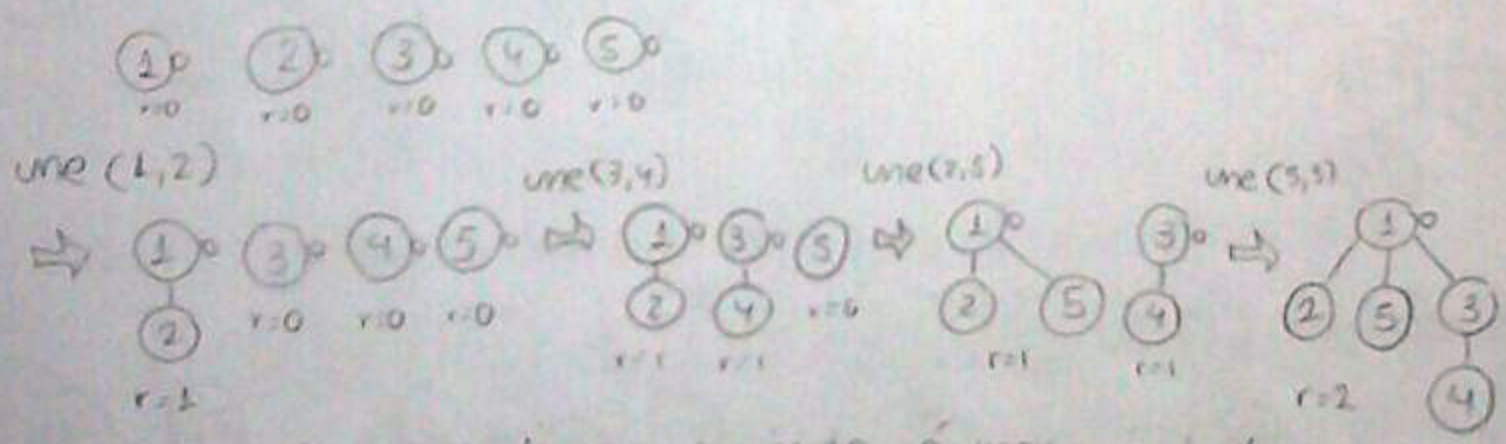
Essa é meio tricky. Ele pode pedir vários coisas. Com/sem união por rank, com/sem compressão de caminhos.

Sempre começamos com conjuntos unitários disjuntos, e vamos fundindo eles conforme indicado. Sem união por rank, fundir(a,b) geralmente faz com que a raiz de b aponte para a raiz de a, embora a ordem seja esclarecida pelo enunciado.

Conjuntos 1,2,3,4,5.



A estrutura final ficou toda torta. Como vimos com as árvores, seria melhor que ela fosse balanceada. Para isso, usamos união por rank. Fundir(a,b) ignora a ordem de entrada e faz a raiz do menor apontar pra raiz do maior.



O rank de um conjunto é incrementado por 1 quando ele é mesclado com outro de rank equivalente.

algoritmo de união sem ser por tamanho:

```
fundir (conj* a, conj* b){
    conj* a_raiz = buscar (a);
    conj* b_raiz = buscar (b);
    b_raiz -> pai = a_raiz;
}
```

```
conj {
    conj* pai;
}
```

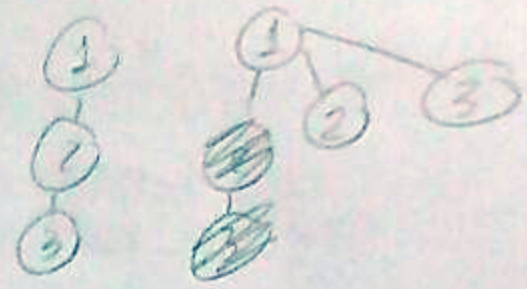
```
buscar (conj* a){
    while (a -> pai != a)
        a = a -> pai;
    return a;
}
```

O algoritmo da união por rank/tamanho:

```
fundir (conj* a, conj* b){
    conj* a_raiz = buscar (a);
    conj* b_raiz = buscar (b);
    if (a_raiz -> rank < b_raiz -> rank)
        a_raiz -> pai = b_raiz;
    else if (a_raiz -> rank > b_raiz -> rank)
        b_raiz -> pai = a_raiz;
    else //if (a_raiz -> rank == b_raiz -> rank)
        a_raiz -> pai = b_raiz;
        a_raiz -> rank ++;
}
```

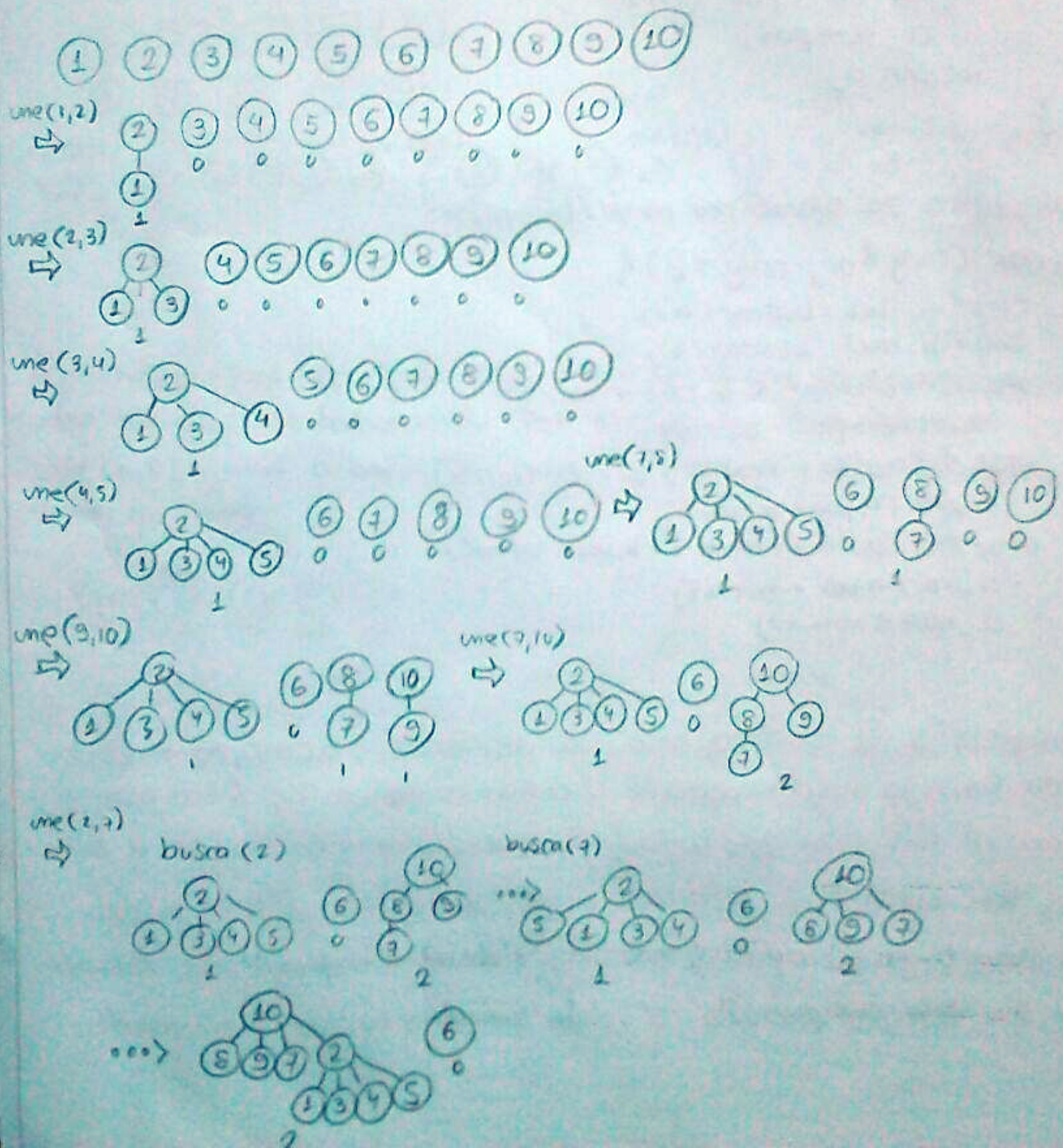
A compressão de caminho funciona alterando o buscar, ao invés do fundir. Como o buscar percorre o caminho de um nó à sua raiz, agora ele faz com que todos os nós nesse caminho tenham a raiz como pai direto. Isso facilita o acesso, pois eles já teriam que procurar a raiz no futuro mesmo. Porém, é importante ressaltar que esta compressão só ocorre quando o buscar é chamado.

```
buscar_comp (conj *a)
    if (a -> pai != a)
        a -> pai = buscar_comp(a -> pai);
    return a -> pai;
```

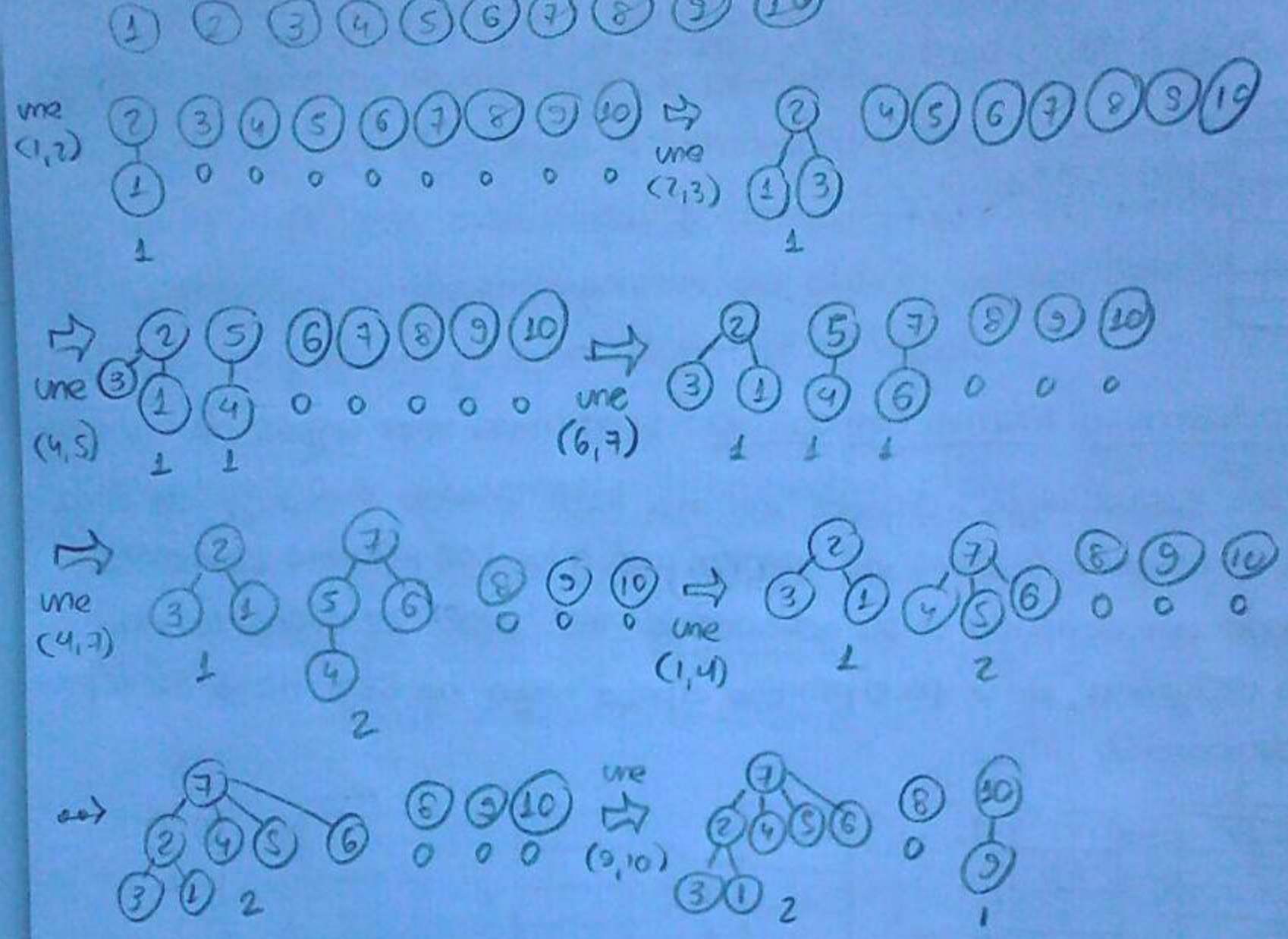


buscar(3);

Exercício: união por rank, compressão. unir a a b em caso de empate de rank. 1, 2, 3, ..., 10



xercício: com rank e compressão - 1, 2, 3, ..., 10



6) Tabela Hash

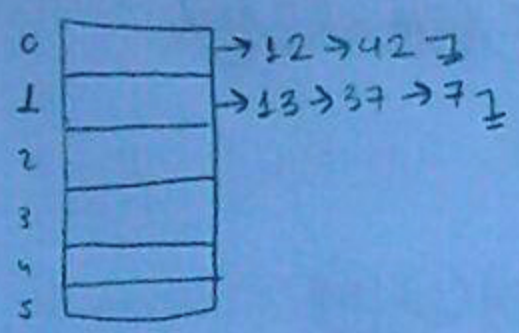
Também é meio tricky. Existem algumas espécies diferentes de hash, mas todas elas seguem o mesmo princípio: para procurar uma palavra em um dicionário, sempre vemos qual a primeira letra da palavra e vamos direto nessa seção. Mas números não tem letras, então temos que inventar uma forma de codificar esse número em outro: essa é a hash. Uma hash table é uma tabela com esses números que a hash pode gerar.

Hash table de endereçamento externo: usa memória fora da tabela para alocar dados. Exemplo:

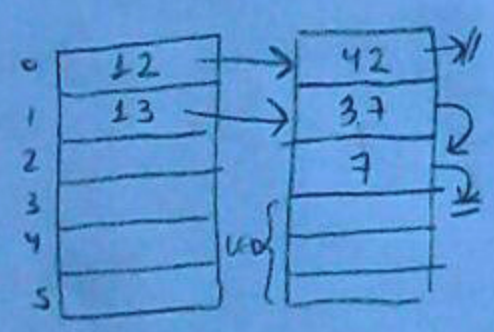
Ex: Inserir 12, 42, 13, 37 e 7 em uma hash de encadeamento exter

$h(x) = x \pmod 6$

$h(12) = 0$; $h(42) = 0$; $h(13) = 1$; $h(37) = 1$; $h(7) = 1$



<u>Encadeamento</u> <u>interno</u> <u>com</u> <u>colisão</u>: separamos uma região de memória com espaços vagos indicados em uma lista. Quando o espaço de uma hash foi ocupado, fazemos ele apontar para o local do próximo elemento. Quando um elemento a ser adicionado não pode ser alocado na hash designada, ele ocupa o primeiro espaço vago na LED (lista de espaços disponíveis).

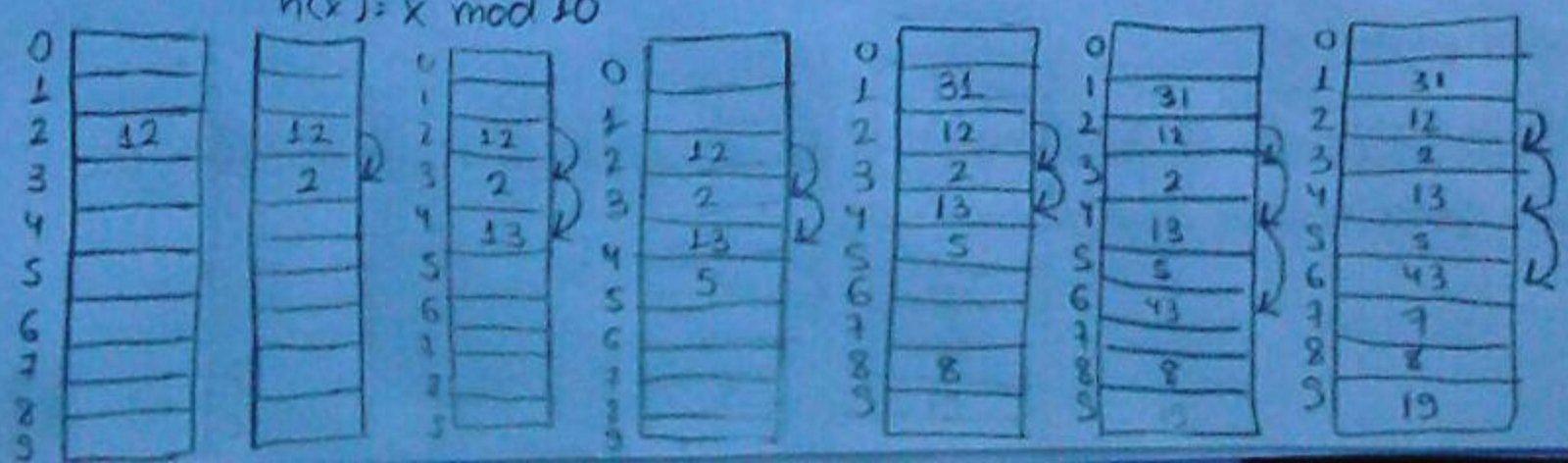


<u>Encadeamento</u> <u>interno</u> <u>sem</u> <u>colisão</u> <u>com</u> <u>coalescência</u>:

Não há espaço alocado fora da hashtable. Quando queremos inserir um elemento e sua vaga está ocupada, inserimos ele na próxima vaga e o último elemento de mesmo hash aponta para uma vaga.

Exemplo: Inserir 12, 2, 13, 5, 8, 31, 43, 19 e 7

$h(x) = x \bmod 10$

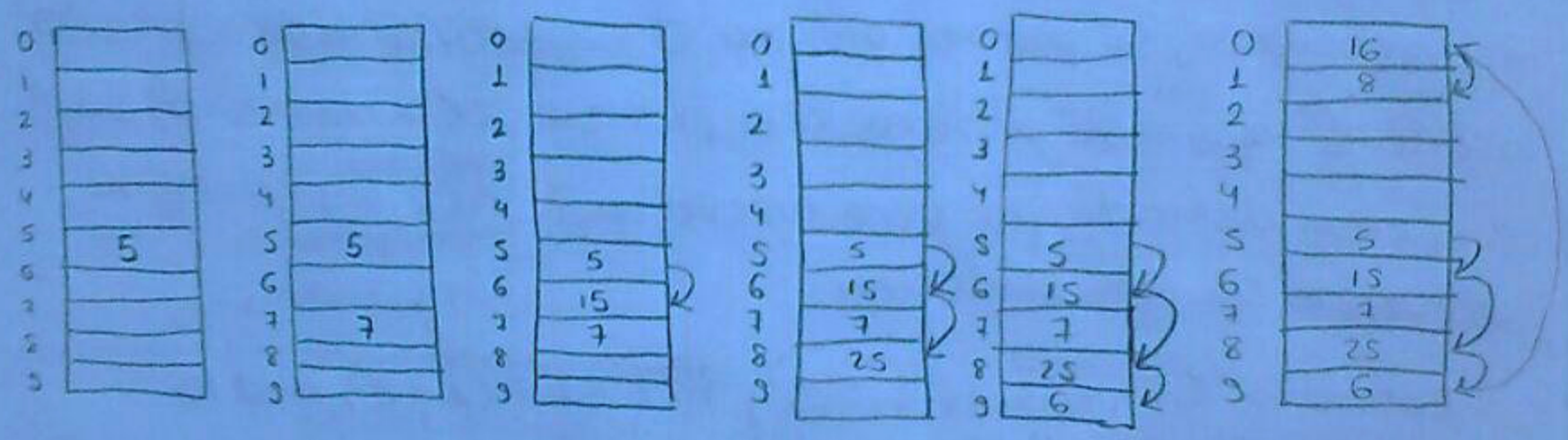


cadeamento interno sem coalescência: quando um elemento a ser inserido está com sua hash ocupada, vemos se o elemento na posição pertence àquela hash. Se pertencer, inserimos o elemento na próxima vaga disponível e fazemos o último elemento da hash apontar pra essa vaga. Se não pertencer, tira o errado e bota o certo. Se alguém apontava pro errado, agora aponta pro errado na próxima posição disponível na tabela.

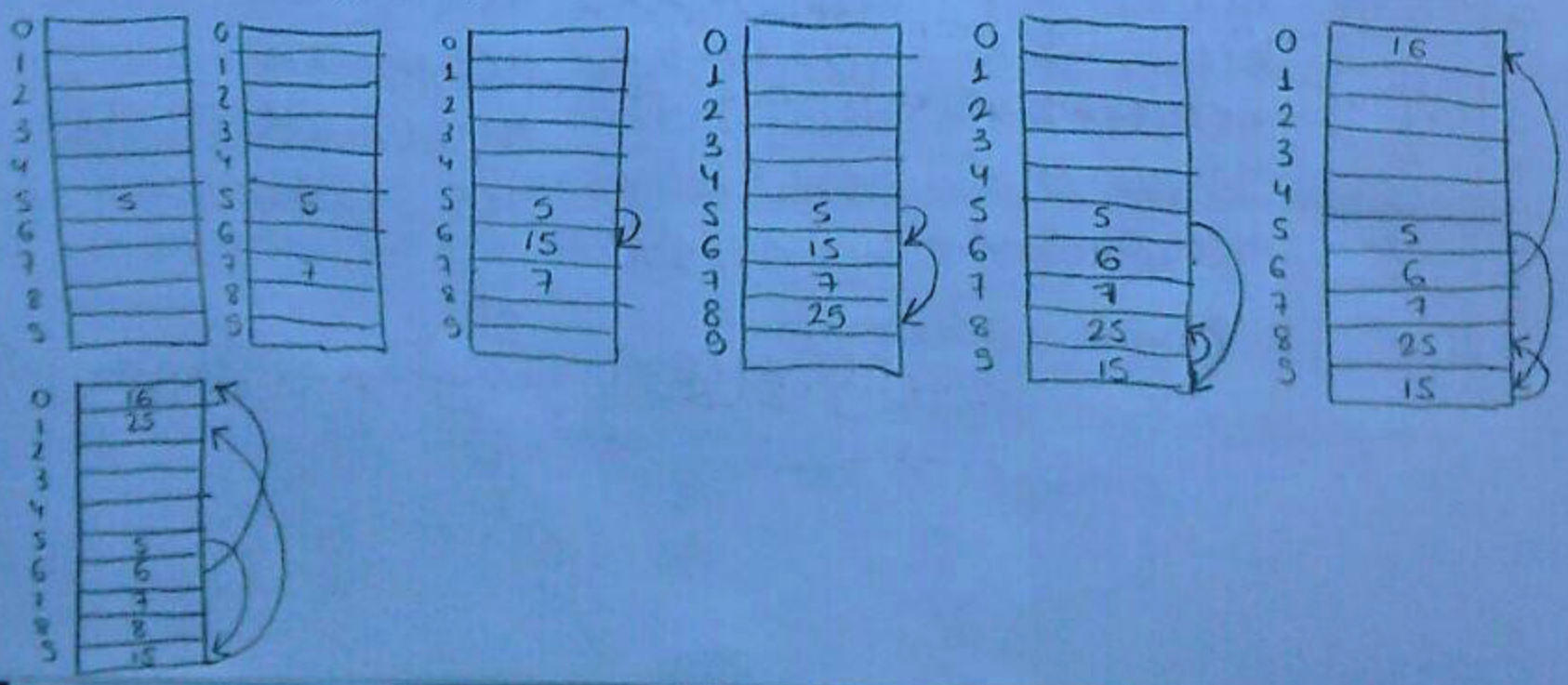
Ele sempre pede para inserir determinadas chaves em certas tabelas.

Exercício: Inserir 5, 7, 15, 25, 6, 16, 8 com $h(x) = x \bmod 10$

interno s/ colisão c/ coalescência



interno s/ colisão s/ coalescencia



Exercício: 1, 11, 4, 13, 12, 2, 9, 5 ; $h(x) = x \bmod 10$

enc. interno s/ área de colisão s/ coalescência

0	
1	1
2	
3	
4	
5	
6	
7	
8	
9	

0						
1	1	1	1	1	1	1
2	11	11	11	12	12	12
3			13	13	13	13
4		4	4	4	4	4
5				11	11	11
6					2	2
7						
8						
9						9

0	
1	1
2	12
3	13
4	4
5	5
6	2
7	11
8	
9	9

Encadeamento Aberto:

sem ponteiros. Botamos um elemento na tabela: se sua posição estiver ocupada, só põe na próxima. O segredo do endereçamento aberto é que essa próxima não precisa ser a de baixo, ela é a casa definida por uma função hash $h(x,k)$ sendo k o numero da sua tentativa.

Exercício: 1, 11, 4, 13, 12, 2, 9, 5 ; $h(x,k) = (x+k) \bmod 10$

0	
1	1
2	11
3	13
4	4
5	12
6	2
7	5
8	
9	9

$h(1,0) = 1$ ✓
$h(11,0) = 1$ ✗
$h(11,1) = 2$ ✓
$h(4,0) = 4$ ✓
$h(13,0) = 3$ ✓

$h(12,0) = 2$ ✗
$h(12,1) = 3$ ✗
$h(12,2) = 4$ ✗
$h(12,3) = 5$ ✓
$h(2,0) = 2$ ✗
⋮
$h(2,4) = 6$ ✓

$h(9,0) = 9$ ✓
$h(5,0) = 5$ ✗
$h(5,1) = 6$ ✗
$h(5,2) = 7$ ✓

...goritmos de Hash:

Endereçamento aberto:

```
int busca (hashtable H, int x)
    int k=0;
    int pos=h(x,k);
    while (H[pos] != vazia)
        if (H[pos] == x) return pos;
        pos = h(x,++k);
    return pos;

void insere (hashtable H, int x)
    int pos = busca(H,x);
    H[pos] = x; //ou está vazio ou contém x, esta linha funciona
                em ambos os casos

    return;
```

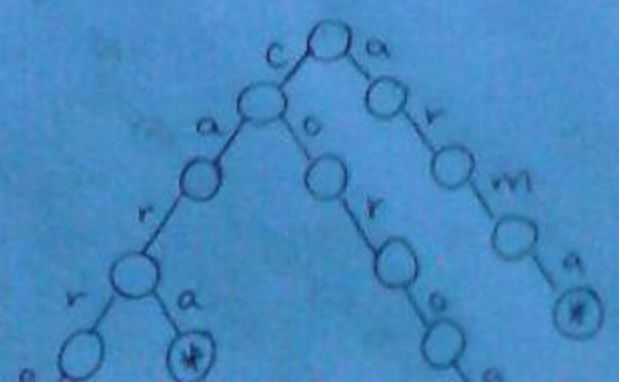
Encadeamento Externo:

```
key* busca (ht H, int x)
    int pos=h(x);
    key* pt = H[pos];
    while (pt != NULL)
        if (pt->key == x) break;
        pt = pt->prox;
    return pt;
```

```
void insere (ht H, int x)
    key* aux = busca(H,x);
    if (aux == NULL)
        aux = malloc(sizeof(key));
        aux->key = x;
        int pos=h(x);
        aux->prox = H[pos];
        H[pos] = aux;
```

7) Árvore Digital e Árvore PATRICIA

Árvore digital: cada caractere é uma aresta e o caminho deriva a palavra. "carro", "cara", "coroa", "arma":



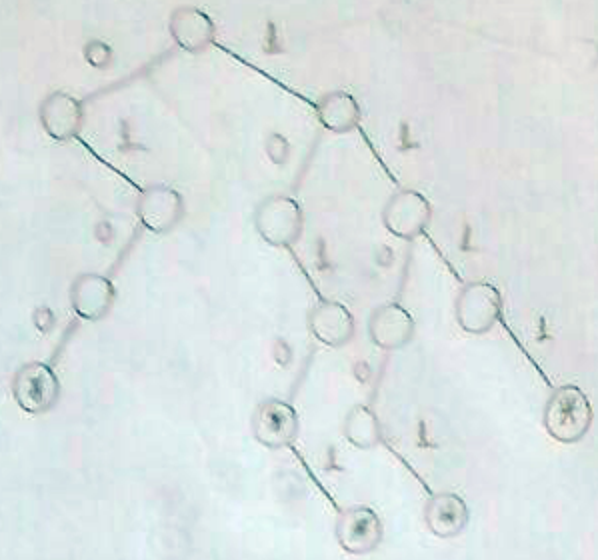
(*) marca fim de palavra

As letras são como decisões em um fluxograma.

Árvore Binária Digital:

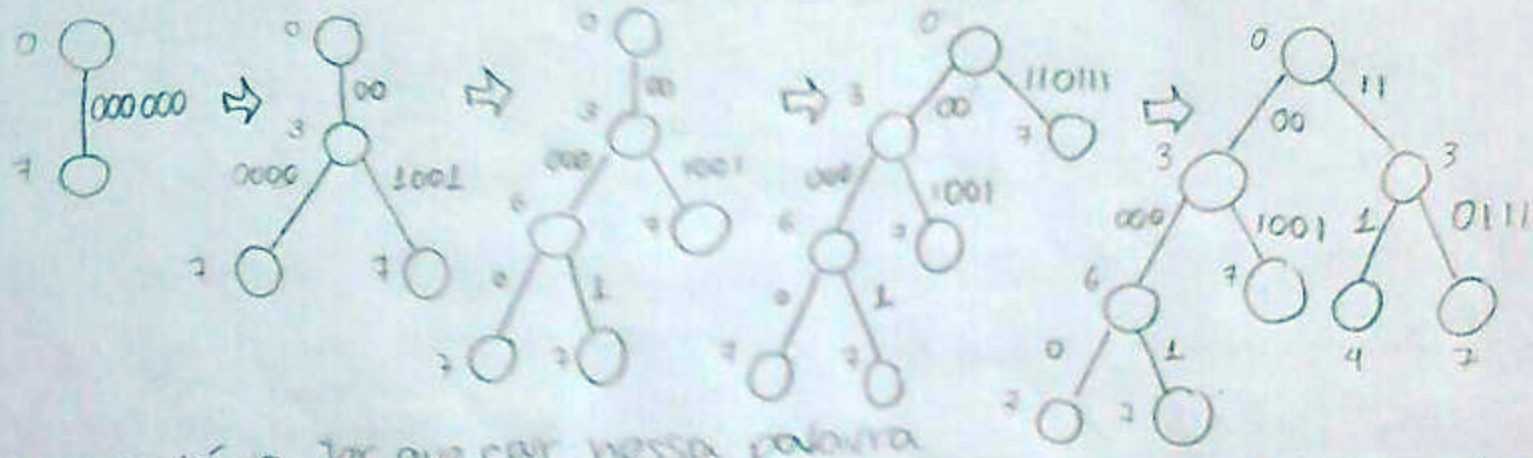
Ex: 11001, 10101, 1111, 0100, 101010

Esquerda: 0
Direita: 1



Árvore Patricia: cada aresta não contém 1 caractere e sim um conjunto de caracteres cujo caminho até uma ⊛ não possui decisões

Exercício: inserir na ordem: 000000, 001001, 00001, 110111, 111



"se eu já vou ter que cair nessa palavra mesmo, pra que tantas arestas?"

Os números indicam qual a posição do caractere no qual as ramificações diferem

8) Árvore de Huffman

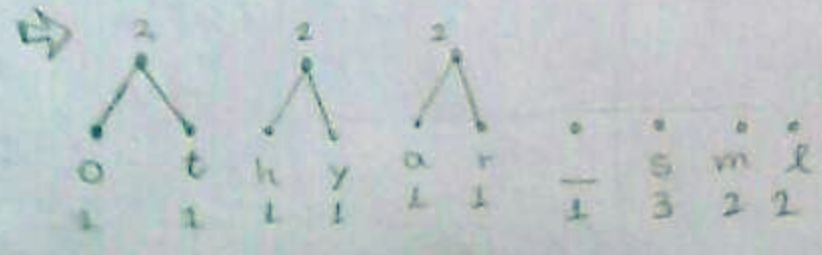
Ele pede pra codificar uma string e dizer o comprimento.

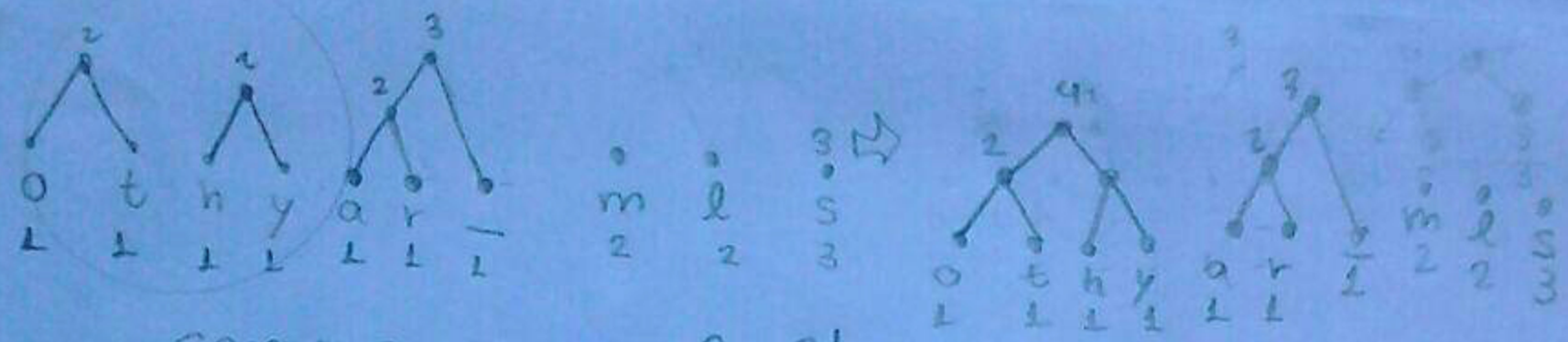
Fazemos a frequência de cada caractere.

Exemplo: "mostly harmless". (esqueci do e na hora de fazer e não vou refazer)

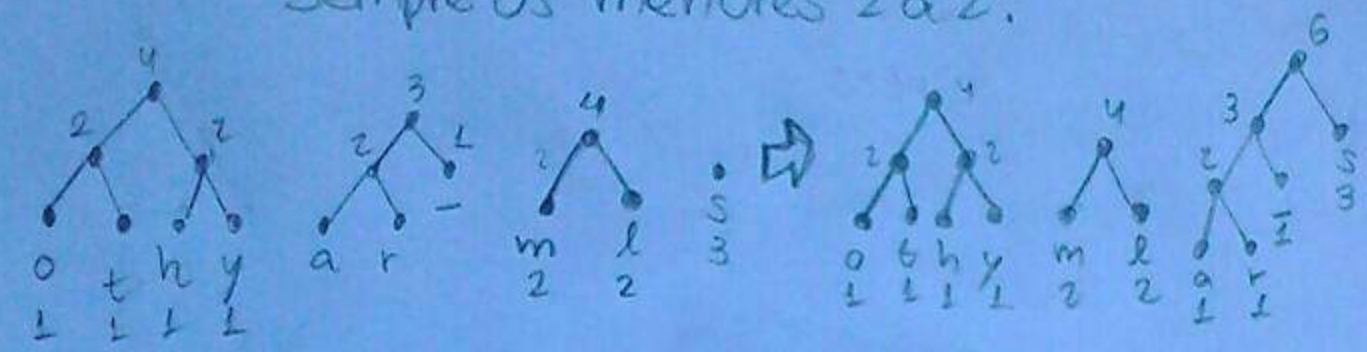
Unimos os menos frequentes 2 a 2.

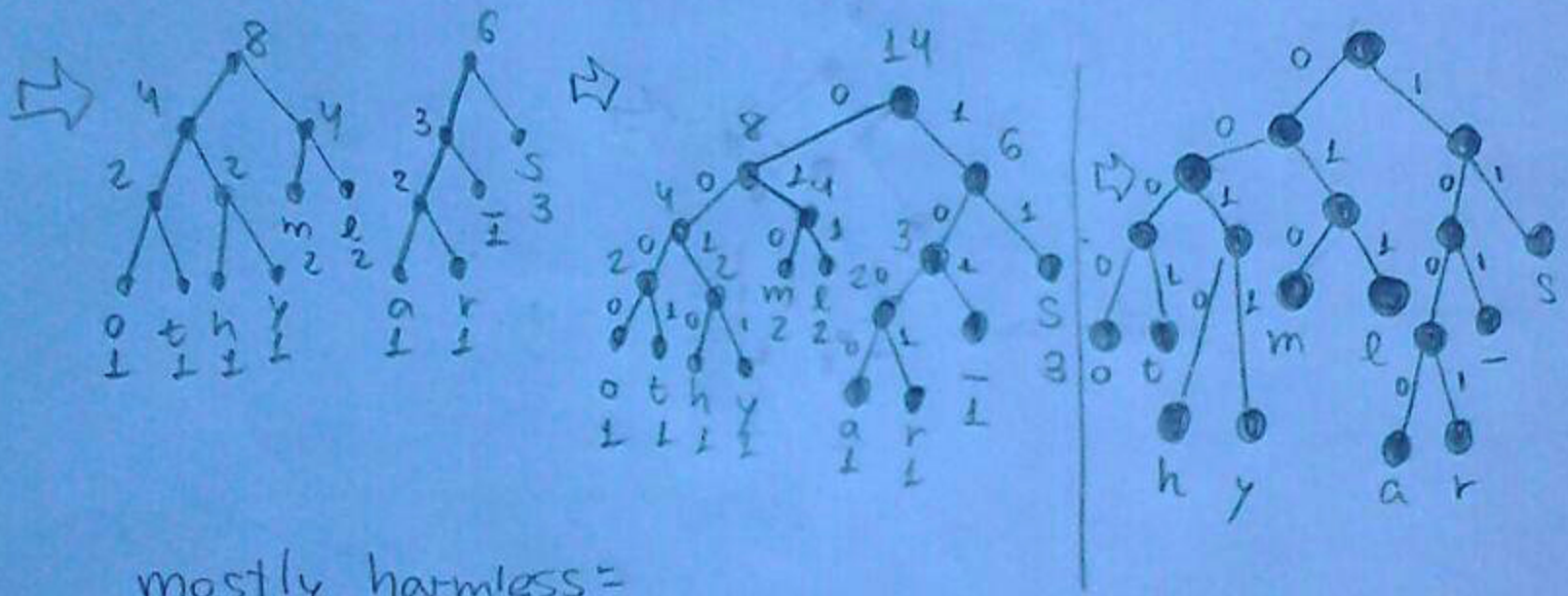
m: 2
o: 1
s: 3
t: 1
l: 2
h: 1
y: 1
a: 1
r: 1
_: 1



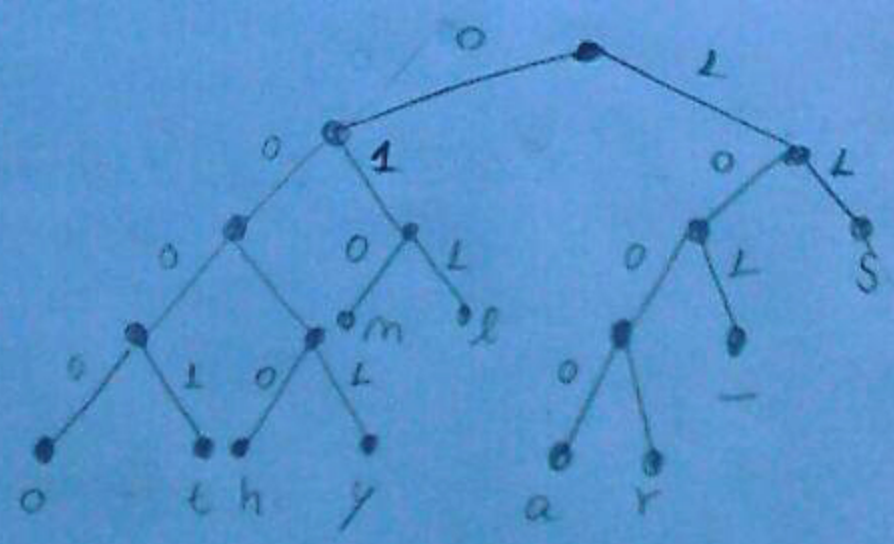


Sempre os menores 2 a 2!





mostly_harmless=

$\frac{010}{m}$ $\frac{0000}{o}$ $\frac{11}{s}$ $\frac{0001}{t}$ $\frac{011}{l}$ $\frac{0011}{y}$ $\frac{101}{_}$ $\frac{0010}{h}$ $\frac{1000}{a}$ $\frac{1001}{r}$

$\frac{010}{m}$ $\frac{011}{l}$ $\frac{11}{s}$ $\frac{11}{s}$