Organização e Recuperação de Dados
Profa. Valéria

9ª Lista de Exercícios

1. A função *hashing* para um sistema de *hashing* extensível também pode fazer uso da função *módulo*, mas o motivo, se comparado ao seu uso em um sistema de *hashing* estático, é diferente. Explique.

   No hashing extensível o espaço de endereços é dinâmico e o endereço *hash* das chaves é analisado bit-a-bit. Dessa forma, não há necessidade de usar a função módulo para limitar o endereço *hash* gerado. Mesmo assim, a função módulo pode ser usada pela função *hash* como parte do processo de transformação da chave.

2. Para sistemas de *hashing* extensível, conforme visto em aula, descreva as características que a função *hash* deve ter (além da questão de gerar um bom espalhamento das chaves) para que tal sistema funcione.

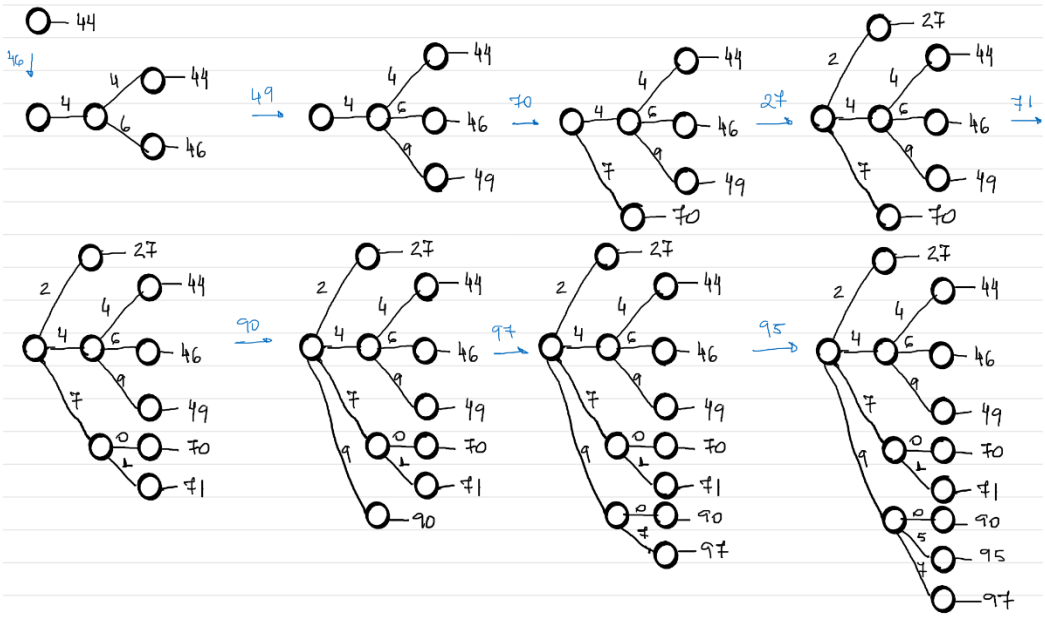
   Além da transformação da chave para um endereço, a função também deve permitir que se extraia uma quantidade específica de bits do endereço gerado.

3. Dada uma lista de números reais, qual seria o fator de divisão da *trie* que indexaria essa lista?

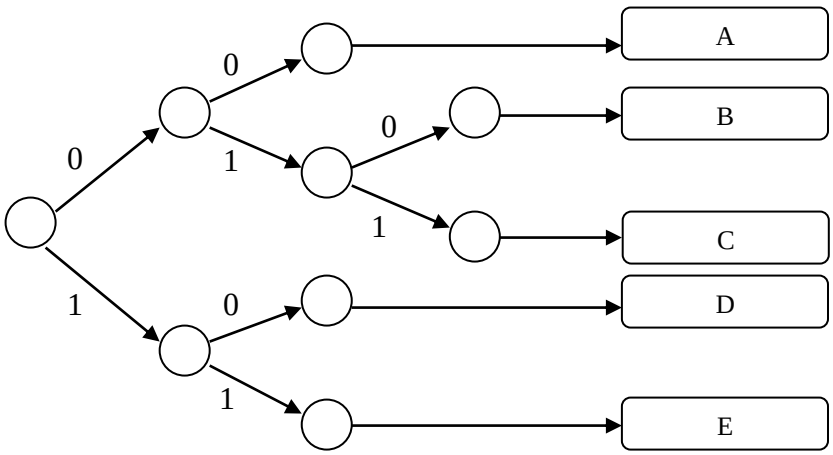
   O fator de divisão seria 11: 10 dígitos numéricos (0 a 9) e um caractere vírgula (,).

4. Crie uma *trie* para indexar a seguinte lista de números inteiros: 44, 46, 49, 70, 27, 71, 90, 97, 95. Dado que essa *trie* indexa números inteiros, qual seria o seu fator de divisão?

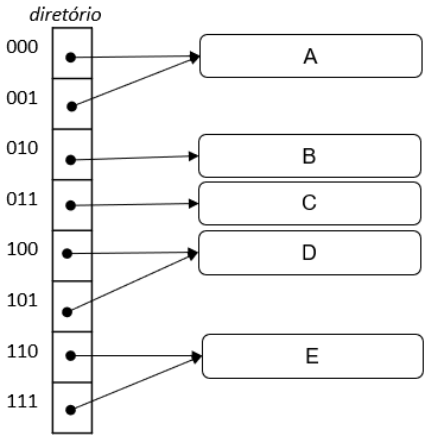
   O fator de divisão seria 10: 10 dígitos numéricos (0 a 9).



5. Na figura abaixo, temos uma *trie radix* 2 endereçando 5 *buckets* (A, B, C, D e E).



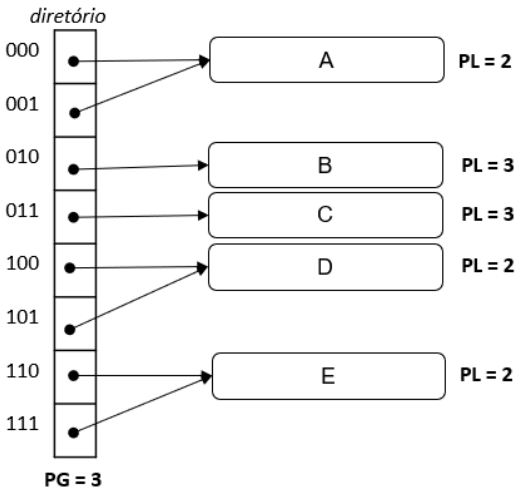
a. Transforme a *trie* da figura acima em um diretório mostrando como fica o endereçamento dos referidos *buckets*.



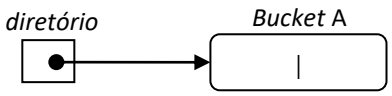
b. Com base na figura acima, a chave 001100 seria mapeada e inserida em qual *bucket*? E a chave 101100?

A chave de endereço 001100 seria inserida no *bucket* A e a chave 101100 seria inserida no *bucket* D.

c. De acordo com a figura acima, qual é a profundidade global (do diretório) e qual é profundidade local de cada *bucket*?



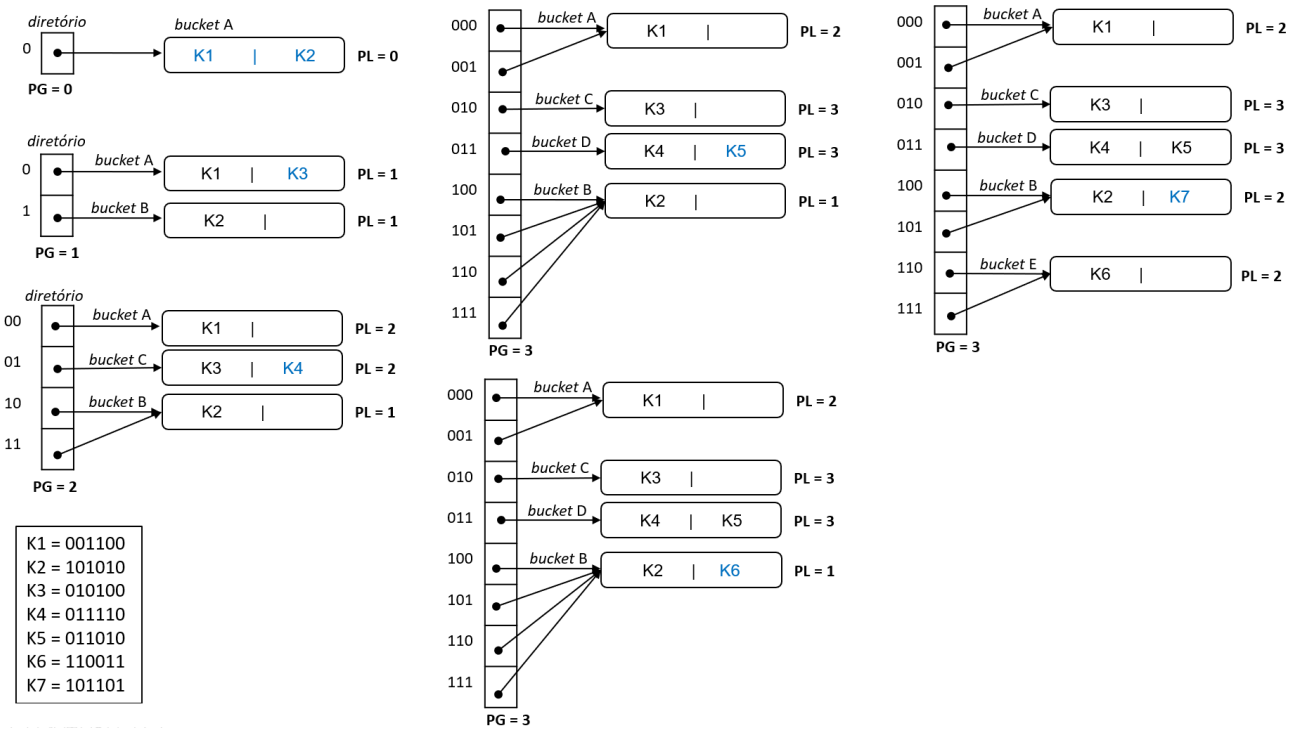
6. Suponha um sistema de *hashing* extensível em que cada *bucket* pode armazenar duas chaves. Inicialmente o sistema está vazio, como mostrado na figura abaixo.



a. Estando o sistema vazio, qual é a profundidade global e qual é a profundidade local de A?

A profundidade global é zero e a profundidade local do *bucket* A também é zero.

b. Mostre como fica o sistema após a inserção das chaves k1, k2, k3, k4, k5, k6, k7, nesta ordem, cujos endereços base são dados pelas respectivas sequências de bits: 001100, 101010, 010100, 011110, 011010, 110011, 101101.



c. Após a inserção das 7 chaves (item b), qual é a profundidade global e a profundidade local de cada *bucket*?

Indicado na figura acima como PG (profundidade global) e PL (profundidade local)

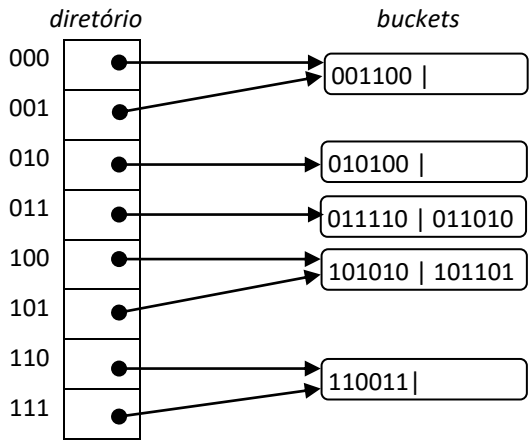
d. Dê um exemplo de uma nova chave cujo endereço provocaria a expansão do diretório.

K8 = 011111

7. Em um sistema de *hashing* extensível, sempre que o diretório se expande, ele dobra de tamanho. Por que isso ocorre?

Porque foi aumentado um bit na representação dos endereços. A adição de um bit dobra a quantidade de endereços possíveis de serem representados.

8. Dada estrutura de *hashing* extensível abaixo, na qual os *buckets* acomodam duas chaves, dê exemplos de duas chaves cujos endereços causem *overflow*: (a) uma que cause expansão do diretório e (b) uma que cause a criação de um novo *bucket*, mas sem precisar expandir o diretório. Para ambos os casos (a e b), explique porque houve ou não a necessidade de expansão do diretório.



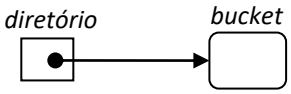
(a) 011111

(b) 100011

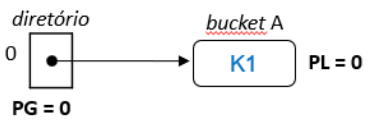
9. Em um sistema de *hashing* extensível haverá dois arquivos: um arquivo que armazena os *buckets* e um que armazena o diretório. Entretanto, o diretório deve ser mantido em memória sempre que possível, enquanto os *buckets* vão sendo trazidos para a memória conforme necessário. Sendo os *buckets* de tamanho fixo, qual é o conteúdo de cada célula do diretório?

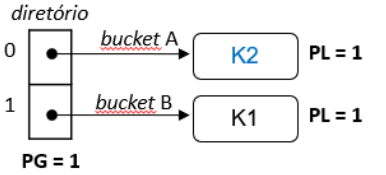
   Cada célula do diretório armazena o RRN de um *bucket*.

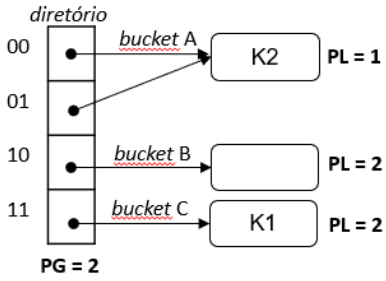
10. Suponha um sistema de *hashing* extensível em que cada *bucket* pode armazenar um registro. Inicialmente o sistema está vazio.



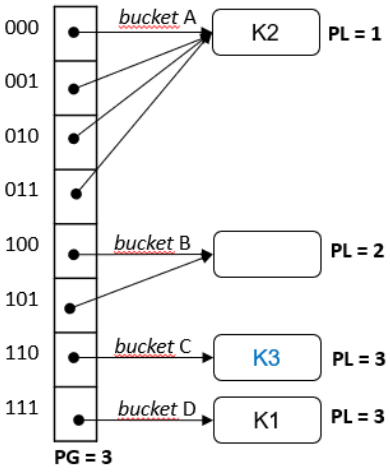
   Mostre como ficam o diretório e os *buckets* após a inserção das chaves k1, k2, k3, k4, k5 e k6, nesta ordem, sabendo que os respectivos endereços bases são: 1111, 0000, 1100, 1010, 0101, 1001. Indique no gráfico a profundidade global e as profundidades locais.

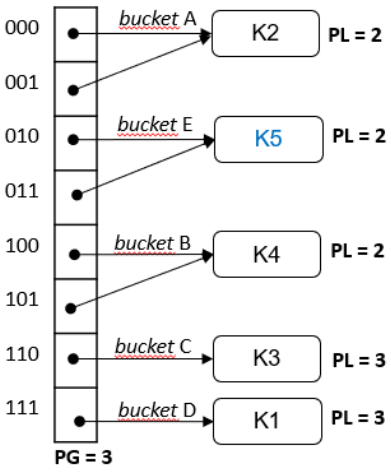


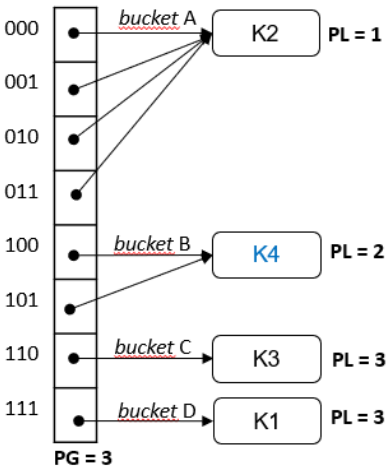


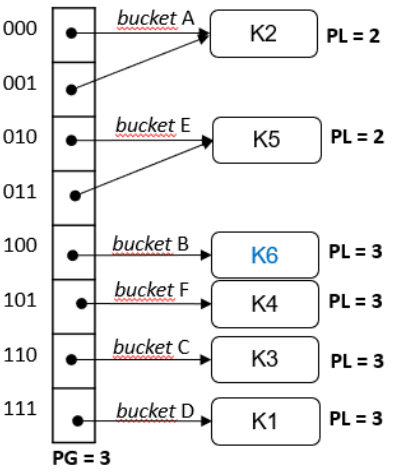


K1 = 1111
K2 = 0000
K3 = 1100
K4 = 1010
K5 = 0101
K6 = 1001

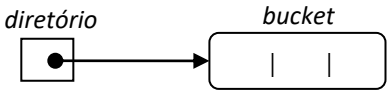




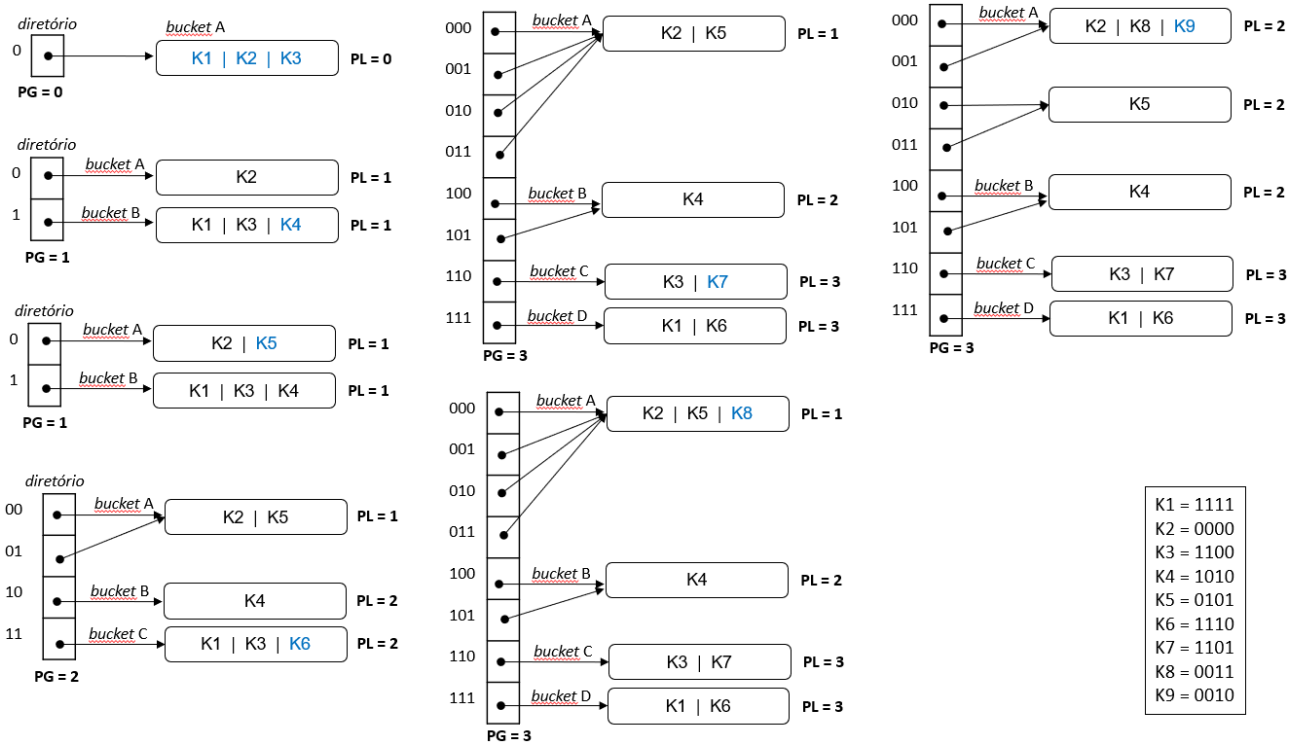




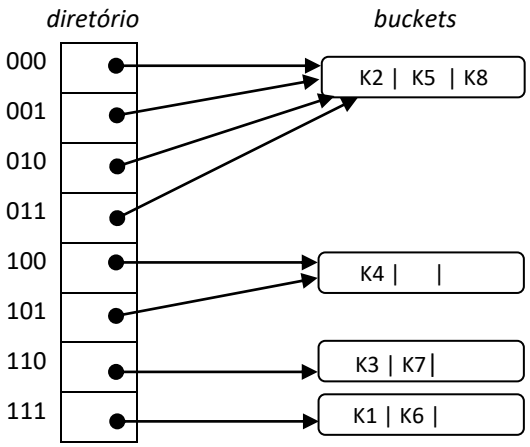
11. Suponha um sistema de *hashing* extensível em que cada *bucket* pode armazenar três registros. Inicialmente o sistema está vazio (a profundidade global é zero)

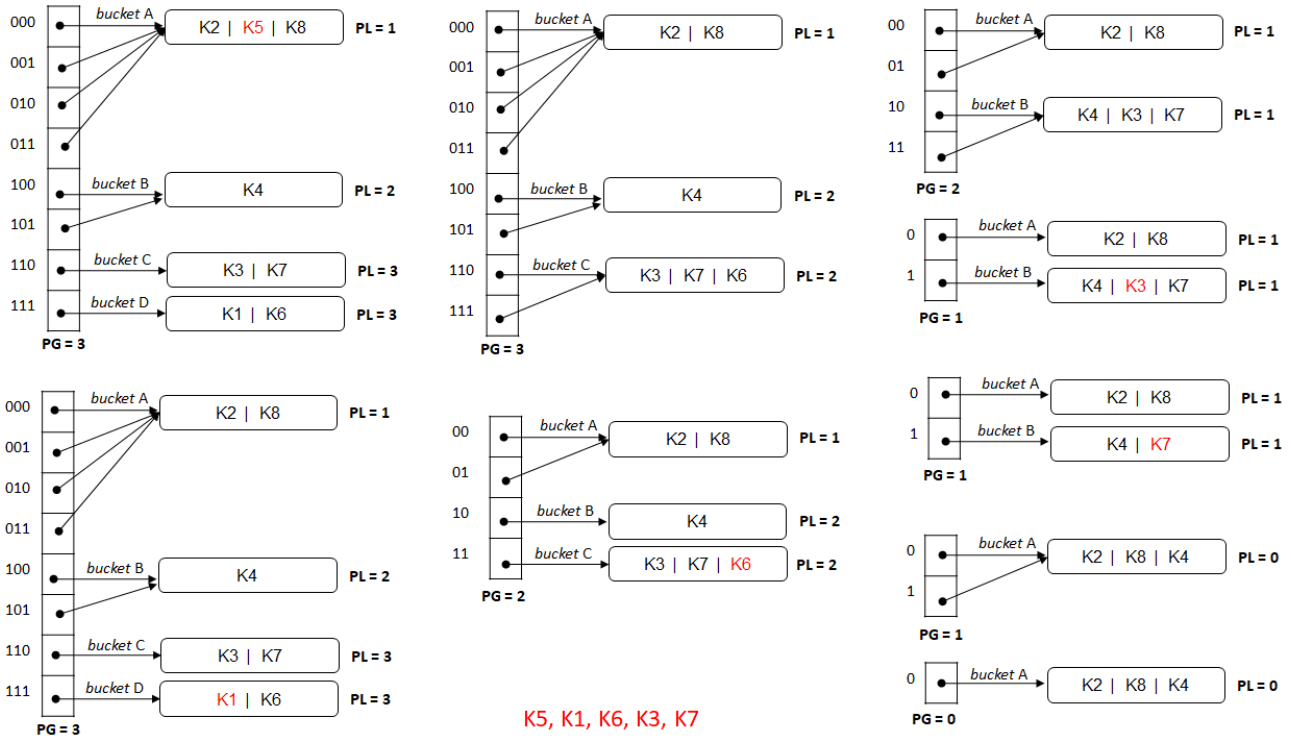


Mostre como ficam o diretório e os *buckets* após a inserção das chaves k1, k2, k3, k4, k5, k6, k7, k8 e k9, nesta ordem, sabendo que os respectivos endereços bases são: 1111, 0000, 1100, 1010, 0101, 1110, 1101, 0011, 0010. Indique no gráfico a profundidade global e as profundidades locais.

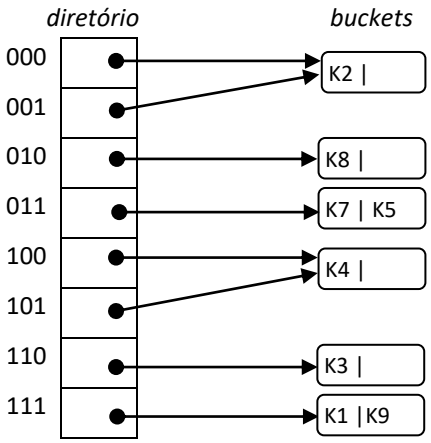


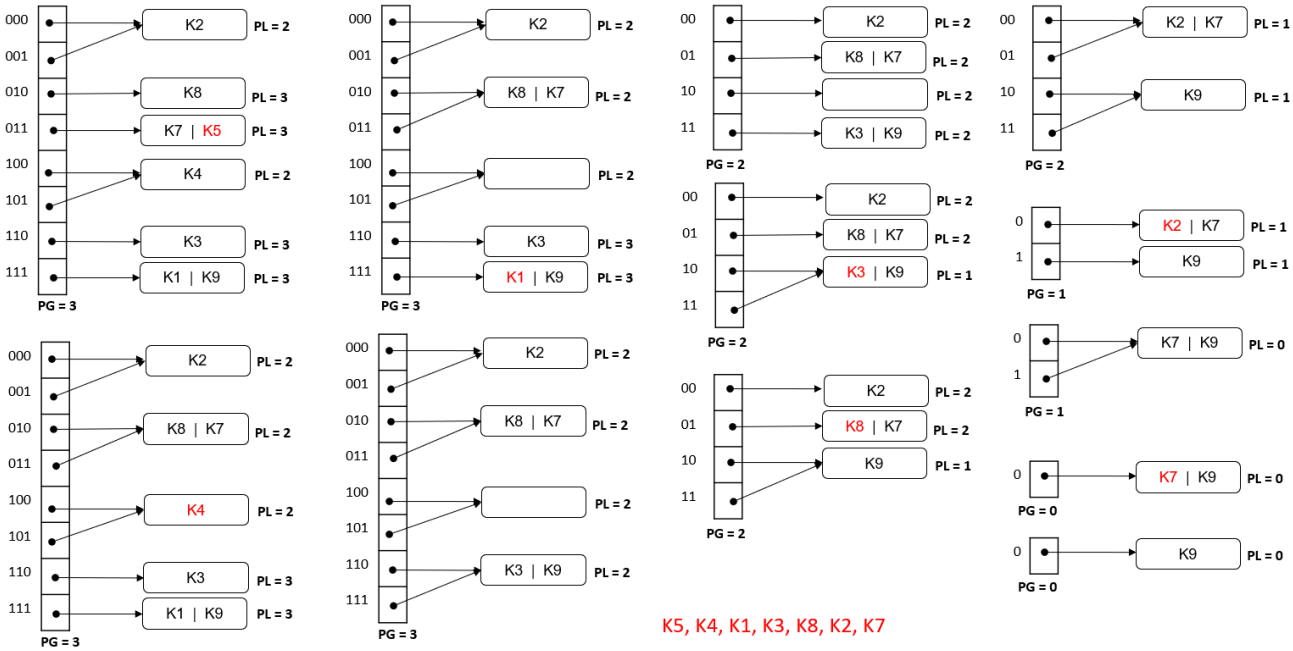
12. Considere o *hashing* extensível mostrado na figura (*buckets* com até três chaves) e simule as remoções das seguintes chaves, nesta ordem: K5, K1, K6, K3, K7.





13. Considere o *hashing* extensível mostrado na figura (*buckets* com até duas chaves) e simule as remoções das seguintes chaves, nesta ordem: K5, K4, K1, K3, K8, K2, K7.





K5, K4, K1, K3, K8, K2, K7