Análise

Até aqui tudo bem.

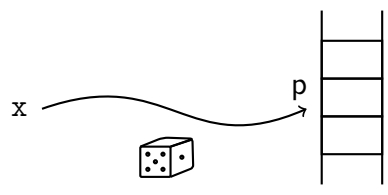
Mas será que esse negócio funciona bem na prática?

Bom, vamos ver.

Quer dizer, vamos fazer a análise.

E para fazer a análise, a gente faz a nossa suposição usual

→ *Imagine que as funções de dispersão* $\mathsf{h_i}(.)$ *tem comportamento aleatório*



E daí, a gente faz uma observação bem fácil

→ *Quando a tabela está completamente cheia, uma busca sem sucesso sempre leva tempo* $O(n)$

Porque a busca só pára quando ela encontra uma posição vazia.

Mas uma tabela completamente vazia não tem posições vazias.

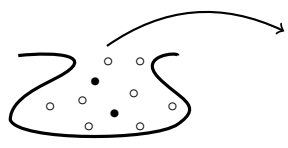
Certo.

Então para a coisa funcionar direito, a tabela não pode estar completamente cheia.

Daí que, nós vamos assumir que 10% das posições da tabela estão vazias.

E que as nossas funções de dispersão tem comportamento aleatório.

Nesse caso, aplicar uma função de dispersão a um elemento `x` é como selecionar uma bolinha de um saco onde 90% das bolinhas são brancas, e 10% das bolinhas são pretas



Quer dizer,

- a probabilidade de obter uma posição ocupada é igual a 9/10
- a probabilidade de obter uma posição vazia é igual a 1/10

E o tempo médio da busca é dado pela resposta da seguinte pergunta

- *Quantas bolinhas precisam ser selecionadas do saco (em média) até obter uma bolinha preta ?*

Bom, isso também é a mesma coisa que lançar uma moeda que dá `Cara` com probabilidade 1/10 até obter `Cara` pela primeira vez.

E a resposta nesse caso é

10 lançamentos (em média)

— (*porque?*)

Quer dizer, uma busca sem sucesso examina (em média) 10 posições na tabela.

E isso leva tempo $O(1)$.

Mas a coisa é ainda melhor para as buscas com sucesso.

Quer dizer, no momento em que a tabela ainda está vazia e os elementos começam a ser inseridos, a função de dispersão localiza uma posição vazia na primeira ou na segunda tentativa.

Daí, mais tarde, quando nós fizermos uma busca por esses elementos, eles também serão encontrados na primeira ou na segunda tentativa — (*mesmo que a tabela esteja cheia*)

— (*porque?*)

Quer dizer, o tempo que é gasto na busca por um elemento que está na tabela é o mesmo tempo que foi gasto no momento da sua inserção.

Mas a maioria dos elementos foi inserida quando a tabela ainda estava relativamente vazia.

Logo, o tempo de busca da maioria dos elementos da tabela é bem pequeno.

Certo.

Mas aqui nós não estamos levando em conta as remoções.

Quer dizer, quando as remoções acontecem, elas marcam as posições que ficaram vazias como apagadas.

E daí, se 90% das posições da tabela estão ocupadas, então os outros 10% vão ser divididos entre posições vazias e posições apagadas.

Mas, o desempenho da busca depende apenas do número de posições vazias.

Quer dizer, a medida que os elementos vão sendo inseridos e removidos, começam a surgir posições apagadas.

E nesse momento, o tempo de execução da busca começa a se degradar.

Daí, quando esse problema se torna muito grave, a solução é reconstruir a tabela.

Quer dizer,

- todos os elementos são removidos
- e depois eles são inseridos em uma tabela completamente vazia

Desa maneira, as posições apagadas são eliminadas.

E a busca volta a ter o seu desempenho ótimo.