

Bin packing utilizando árvore binária para determinar ordem de compra para lista de desejos da Steam

Edwino Stein e Daniel Gohl

Problema da Mochila

O que é o problema da mochila?

Problema da Mochila

O que é o problema da mochila?

Uma mochila com uma certa capacidade, itens com valores e pesos variados que devem ser colocados na mochila, deve-se maximizar a quantidade de itens que devem estar dentro da mochila.

Problema da Mochila

Exemplos e Aplicações:

Problema da Mochila

Exemplos e Aplicações:

Um ladrão roubando uma loja encontra n **objetos**, cada objeto tem seu **valor** v e **peso** p . O ladrão deve levar em consideração o **valor**, e encher sua mochila com os **objetos** mais valiosos sem que sua mochila fique pesada demais.

Problema da Mochila

Exemplos e Aplicações:

Cortar objetos grandes para a produção de itens menores em quantidades bem definidos;

Empacotar itens pequenos dentro de espaços bem definidos;

Bin Packing

O que é o Bin Packing?

Bin Packing

O que é o Bin Packing?

Similar ao problema da mochila, onde se deve armazenar itens de forma eficiente, a **Bin Packing** tem outra abordagem, ela visa usar a menor quantidade de **Bins** que são nossas mochilas de tamanho fixo, para armazenar uma quantidade n de objetos dada as restrições.

Bin Packing

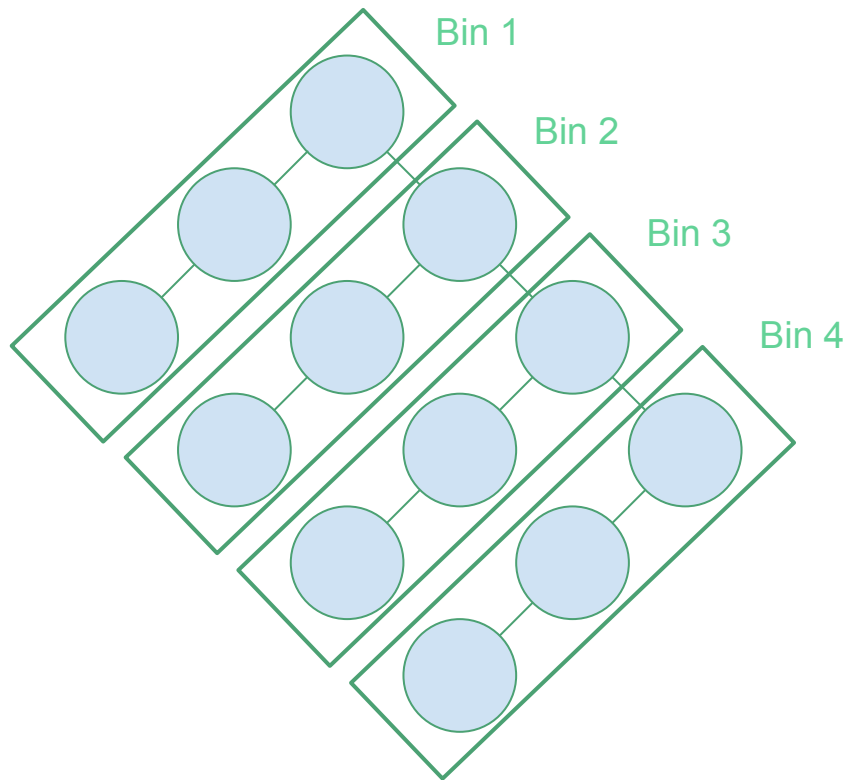
Aplicação no problema:

Bin Packing

Aplicação no problema:

Dada uma lista com n jogos, o tamanho da **Bin**, uma **ordem p** e **valor v** , calcular as combinações de quantas **Bins** são necessárias para comprar toda a lista de jogos.

First-fit com árvore binária



Complexidade inserção na árvore

Algoritmo 1 Inserção na Árvore

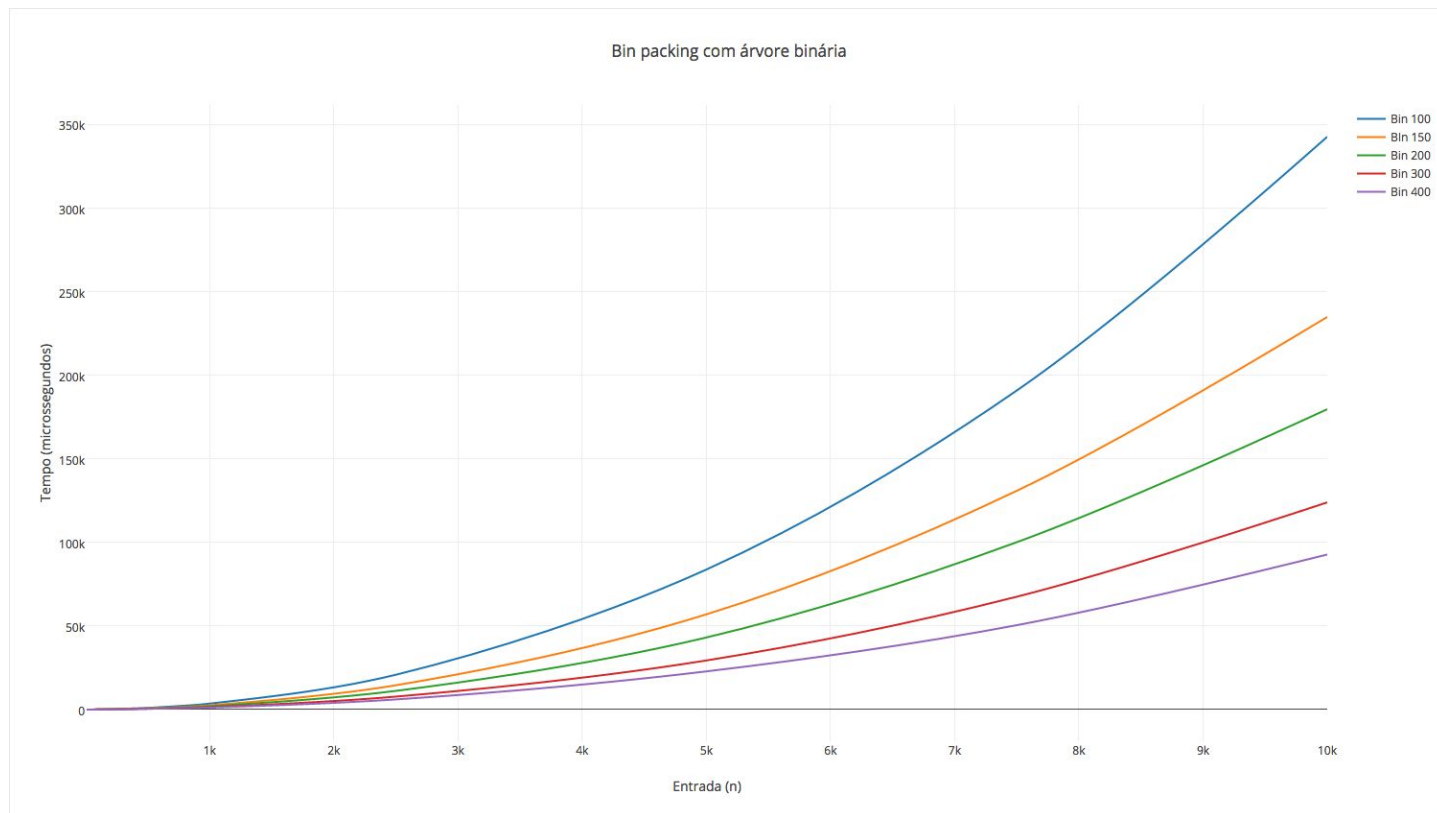
```
1: função INSERT(item, root)
2:   se fits(game, root) então ▷  $O(1)$ 
3:     node ← root
4:     enquanto hasLeft(node) faça ▷  $O(h_l)$ 
5:       node ← getLeft(node)
6:     fim enquanto
7:     setLeft(node, item)
8:     updateSize(root, item)
9:   senão
10:    se hasRight(root) então
11:      insert(item, getRight(root))
12:    senão
13:      setRight(root, item)
14:      updateSize(getRight(root), item)
15:    fim se
16:  fim se
17: fim função
```

Complexidade empacotamento

Algoritmo 2 Empacotamento

```
função DOPACKING(wishlist, binSize)  $O(n^2)$ 
2:   binTree  $\leftarrow$  newtree(binSize)
   para todo  $i \in$  wishlist faça
4:     insert( $i$ , binTree)  $\triangleright O(n)$ 
   fim para
6:   root  $\leftarrow$  binTree
   enquanto root  $\neq$  null faça  $\triangleright O(n)$ 
8:     node  $\leftarrow$  root
     bin  $\leftarrow$  newBin(binSize)
10:    enquanto node  $\neq$  null faça
        pushbackBin(bin, node)
12:    node  $\leftarrow$  getLeft(node)
     fim enquanto
14:    pushbackBinList(bin)
    root  $\leftarrow$  getRight(root)
16: fim enquanto
   fim função
```

Testes



Conclusão

Resultados;

Impressões;

OBRIGADO