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

Edwino Alberto L. Stein[1], **Pedro Daniel da S. Gohl**[1], **Hebert O. Rocha**[1]

[1] Departamento de Ciência da Computação
Universidade Regional de Blumenal (UFRR) – Boa Vista, RR

{edwino.stein,danielgohl13,heberthb12}@gmail.com

Abstract. *This report describes a solution to the Bin Packing, using the first-fit heuristic with binary tree to determine a sequence of close to optimal consumption to wish list Steam considering a fixed investment per month and trying hard to keep order user-defined wish list.*

Resumo. *Este relatório descreve uma solução para o Bin Packing, utilizando a heurística first-fit com Árvore binária, para determinar uma sequência de consumo próxima do ótimo para a lista de desejo da Steam, considerando um investimento fixo por mês e tentando ao máximo manter a ordem da lista de desejo definida pelo usuário.*

1. Introdução

Atualmente, o Steam – uma plataforma de jogos digitais – possui mais de 3,500 títulos e mais de 9 milhões de usuários ativos segundo [Valve 2016], com essa grande quantidade de títulos disponíveis é muito complicado o usuário decidir como vai gastar seu dinheiro. Nossa proposta para esse problema é estabelecer um método de como o usuário gastar seu dinheiro de forma eficiente.

O objetivo desse trabalho é mostrar como uma solução para combinações de compra usando o Problema da Mochila na lista de desejos da steam pode ser viável aos usuários da plataforma, tornando mais fácil a escolha de títulos na plataforma, ajudando o usuário a escolher entre a melhor possibilidade de compra, dentro de suas limitações.

O problema da Bin Packing é uma variação do problema da mochila e pode ser definido com a seguinte situação: uma dona de casa precisa ir no supermercado e comprar todos os itens de uma lista de compras. Cada item tem seu tamanho e prioridade, entretando ela tem que usar o minímo de sacolas possível levando em consideração a prioridade. O problema consiste em determinar quantas sacolas são necessárias para comprar toda a lista e quais itens devem ser colocados nessas sacolas. Uma aplicação prática deste problema aparece na análises de combinações para lista de desejo da Steam, voltada a eficiência e quantidade de Bins.

O problema resolvido por esse artigo pode ser definido com a seguinte pergunta *Dada a quantia que o usuário pode usar por mês o algoritmo estipula em quantos meses ele irá comprar toda sua lista de desejos.*

2. Problema da Mochila

Segundo [Martello e Toth 1990] o problema da mochila consiste em definir restrições para encher a mochila com uma variada possibilidade de itens. O problema pode ser matema-

ticamente representado enumerando-se os objetos de *1* até *n* e os inserindo em um vetor de binários $x_j(j = 1, ..., n)$ para a seguinte condição:

$$x_j = \begin{cases} 1, & \text{se o objeto} j \text{for selecionado} \\ 0, & \text{caso contrário} \end{cases}$$

Para demonstrar, [Martello e Toth 1990] utilizou-se do seguinte exemplo: Supondo que um mochileiro tenha que encher sua mochila com vários itens disponíveis, mas de uma forma que seja confortável para ele. Se s_j é a medida de conforto dada pelo item *j*, e p_j é o peso do item e *c* é o tamanho da mochila, nosso problema vai selecionar dentre os valores do vetor binário *x* para satisfazer a regra.

$$\sum_{j=1}^{n} p_j x_j \leq c$$

e para satisfazer a restrição de conforto

$$\sum_{j=1}^{n} s_j x_j.$$

[Marques e Arenales 2002] define uma variação do problema da mochila: a mochila compartimentada que é semelhante àquele, porém neste o agrupamento de itens é feito por subconjuntos.

"Um alpinista deve carregar sua mochila compossíveis itens de seu interesse. A cada item atribui-se o seu peso e um valor de utilidade (até aqui, o problema coincide com o clássico Problema da Mochila). Entretanto, os itens são de agrupamentos distintos (alimentos, medicamentos, utensílios, etc.)" [Marques e Arenales 2002]

2.1. Bin Packing

De acordo com [Martello e Toth 1990] o problema da Bin Packing pode ser descrito com a mesma terminologia que o problema da mochila, dado *n* itens e *n* mochilas, onde:

$$p_j = peso \text{ de cada itens} j,$$

$$c = capacidade \text{ de cada mochila}$$

Adicionando cada item a mochila sem que o peso dos itens não ultrapassem o tamanho mochila, tentando minimizar a quantidade de mochilas usadas.

[Cormen et al. 2001]acrescenta que dado um determinado número de itens j, onde o peso p_n do n-ésimo item satisfaz $0 < p_n < 1$, onde queremos usar a menor quantidade de mochilas. Essas mochilas podem armazenar quantos itens quiser, contanto que peso não exceda o limite definido como 1.

3. Heurística First-Fit

Segundo [Cormen et al. 2001] a heurística First-fit voltada para o problema do Bin Packing aloca o primeiro item encontrado dentro da primeira na mochila que couber o item.

$$c = capacidade \text{ da mochila ;}$$

$$j = quantidade \text{ de itens} j;$$

$$p = peso \text{ dos itens}$$

$$S = \sum_{n=1}^{j} p_n$$

De acordo [Martello e Toth 1990] a complexidade do algoritmo de Bin Packing usando a heuristíca é O($nlogn$).

Um exemplo que ilustra a heurística First-Fit é a alocação de grupo de turistas em micro ônibus pra um para excursão, onde os micro ônibus contém 7 lugares (bins com capacidade 7), e os grupos de turistas dispostos da seguinte forma:

- 3 turistas americanos;
- 2 turistas japoneses;
- 6 turistas alemães;
- 4 turistas canadenses;
- 5 turistas ingleses;
- 2 turistas espanhóis.

Considerando que a quantidade de pessoas por grupo seja os pesos e os grupos não podem ser quebrados em subgrupos, utilizando a heurística First-Fit será preciso 4 micro ônibus e os grupos distribuídos da seguinte forma:

- Grupo americano (3), grupo japonês (2), grupo espanhol (2);
- Grupo alemão (6);
- Grupo canadense (4);
- Grupo inglês (5);

3.1. First-fit com Árvore Binária

A inserção da árvore binária de busca se baseia na ideia de comparação entre o valor a ser inserido e os nós da árvore, caso o valor seja menor que o valor de um nó, o mesmo será um novo nó da subárvore da esquerda, caso contrário, será um novo nó da subárvore da direita. Essa ideia garante bastante eficiência durante a inserção e busca de valores em árvores binárias de busca.

Realizando um paralelo com o First-fit, é possível notar que podemos nos beneficiar dessa característica da árvore binária de busca. Para isso, é necessário realizar algumas mudanças na estrutura da árvore binária. Primeiramente devemos definir que para cada nó da direita será um pacote (bin), e cada nó da esquerda será um item de um pacote, dessa forma, ao invés de verificarmos se um valor é maior ou menor, como é realizado pela árvore binária de busca, podemos verificar se um determinado item irá caber ou não no pacote, caso caiba, o item será inserido como filho da esquerda do nó, caso contrário, deve verificar se o item cabe no próximo pacote (nó da direita). Com essas considerações, podemos distribuir os itens em vários pacotes apenas com inserções em uma árvore binária, e após inserir o último item, basta percorrer a árvore em pré-ordem para recuperar todos os pacotes e seus respectivos itens.

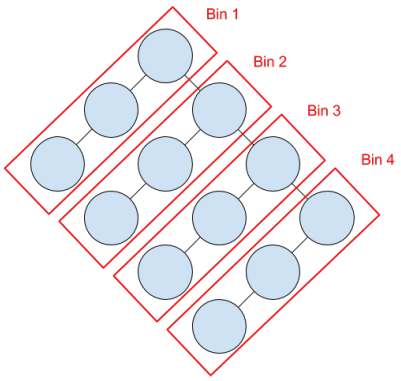

Figura 1. Árvore Binária com First-Fit

4. Ambiente de desenvolvimento

Este trabalho foi implementado utilizando a linguagem de programação C++ utilizando o padrão 2011 e desenvolvido e testado nos ambientes:

- Mac OS X 10.11.5 utilizando o compilador clang versão 703.0.31 (Apple LLVM);
- Ubuntu 16.04 utilizando o compilador GNU G++ versão 5.4.0;

O programa espera dois parâmetros de entrada, o primeiro é um arquivo JSON contendo a lista de desejos do usuário (itens), o seguindo parâmetro é o valor máximo que será gasto por mês (tamanho dos pacotes).

Para parsear o arquivo JSON, foi utilizada a biblioteca JSON for Modern C++ desenvolvida por Niels Lohmann e distribuída sob a licença do MIT através do Github.

4.1. Ambiente de testes

Para os testes, foram utilizados como entrada de 10 a 10000 itens gerados aleatórios com valores entre 5 e 75 reais, com 4 variações de tamanhos de bins: R$ 100, R$ 150, R$ 200, R$ 300 e R$ 400. Os testes foram executados com uma maquina equipada com um Intel Core i7 2600 com 12GB de memória RAM com o sistema operacional Mac OS X 10.11.5.

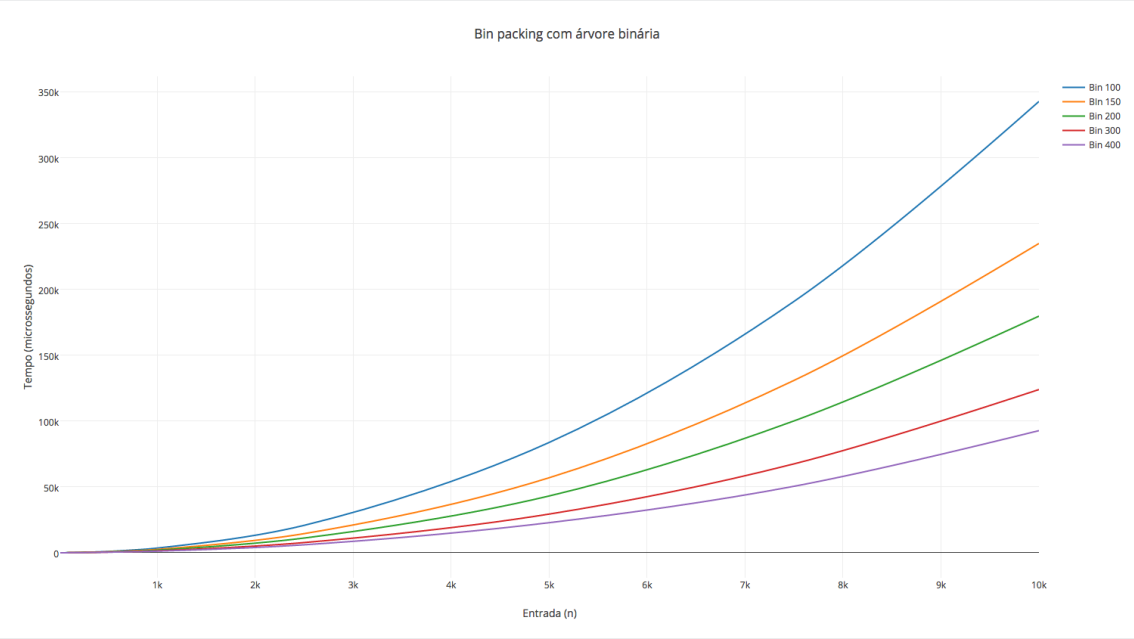

Figura 2. Benchmark de execução

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
```

Algoritmo 2 Empacotamento

```
    função DOPACKING(wishlist, binSize)
 2:     binTree ← newtree(binSize)
        para todo i ∈ wishlist faça
 4:         insert(i, binTree)                                  ▷ O(n)
        fim para
 6:     root ← binTree
        enquanto root ≠ null faça                              ▷ O(n)
 8:         node ← root
            bin ← newBin(binSize)
10:         enquanto node ≠ null faça
                pushbackBin(bin, node)
12:             node ← getLeft(node)
            fim enquanto
14:         pushbackBinList(bin)
            root ← getRight(root)
16:     fim enquanto
    fim função
```

5. Conclusão

Por mais que o first-fit não atinja uma solução muito próxima do ótimo devido sua característica gulosa. Com este trabalho verificamos que a utilização do first-fit utilizando árvores binárias se adequou muito bem ao nosso problema, pois manter a ordem dos itens da lista de desejos do usuário é uma restrição importante, ao contrario de outras heurísticas para o problema do Bin packing.

Com base em nossos testes, definimos que a complexidade do algoritmo apresentado para o pior caso é O(n), onde este caso é alcançado quando apenas um item é alocado por pacote, forçando a árvore binária de empacotamento cresça somente para direita consequentemente aumentando a largura da árvore. Em contra partida, quando aumentamos o tamanho do pacote, verificamos que a árvore tende manter um equilíbrio entre a largura e altura, resultando em uma redução de tempo significativa, como ilustra o gráfico [figura 2].

Referências

[Cormen et al. 2001] Cormen, T., Leiserson, C., Rivest, R., e Stein, C. (2001). *Introduction To Algorithms*. MIT Press.

[Marques e Arenales 2002] Marques, F. d. P. e Arenales, M. N. (2002). O problema da mochila compartimentada e aplicações. *Pesquisa Operacional*, 22(3):285–304.

[Martello e Toth 1990] Martello, S. e Toth, P. (1990). *Knapsack problems: algorithms and computer implementations*. Wiley-Interscience series in discrete mathematics and optimization. J. Wiley & Sons.

[Valve 2016] Valve, C. (2016). Steam. `http://store.steampowered.com/about/`. Accessed: 2016-08-17.