INSTITUTO DE COMPUTAÇÃO - UNICAMP

MC202EF - Estrutura de Dados

2º Semestre de 2017

Professor: Julio Cesar dos Reis
Monitores: Wellington Lucas Moura (PED)
Victor Luccas Soares Villas Boas Antunes (PAD)
José Carlos Vasques Moreira (PAD)

Laboratório 06: Gerenciador de Memória

Prazo de entrega: 25/10/2017 23:59
Peso: 2
Em dupla

1 - Descrição

O sistema operacional (SO) de um computador gerencia a execução de programas (processos). Parte importante do SO envolve iniciar ou finalizar processos. Para permitir a execução de um processo, é necessário que seja concedida à ele uma porção da memória. A dupla precisa desenvolver uma técnica de alocação de memória[1] que divide a memória em partições de potências de 2, ou seja 2^i, onde *i* é um inteiro não negativo. Nessa técnica, o SO mantém uma árvore binária na qual cada nó representa um pedaço da memória. Cada partição de memória alocada para um processo (nó folha da árvore) pode estar em um dos três estados seguintes (veja a Figura 1):

- **Livre**: um nó folha na árvore que não está sendo usado por nenhum programa. Caso haja um programa de tamanho X, e o nó no estado Livre tenha tamanho 2^i, então se $2^{i-1} < X \leq 2^i$, o processo X pode ocupar esse nó folha, passando-o para o estado Ocupado.

- **Ocupado**: um nó folha na árvore que foi alocado para um programa específico. Caso o programa seja finalizado, duas situações podem ocorrer: **1)** Se o irmão desse nó estiver no estado Livre, os nós referentes a esses dois irmãos são retirados da árvore e o nó pai deles, que era um nó não-folha em estado Particionado, se torna um nó folha com

[1] https://pt.wikipedia.org/wiki/Buddy_memory_allocation

estado Livre. **2)** Se o irmão desse nó estiver no estado Ocupado ou Particionado, esse nó passa para o estado Livre.

- **Particionado**: um nó interno da árvore (ou seja, não-folha). Em um nó desse estado não podem ser alocados processos.

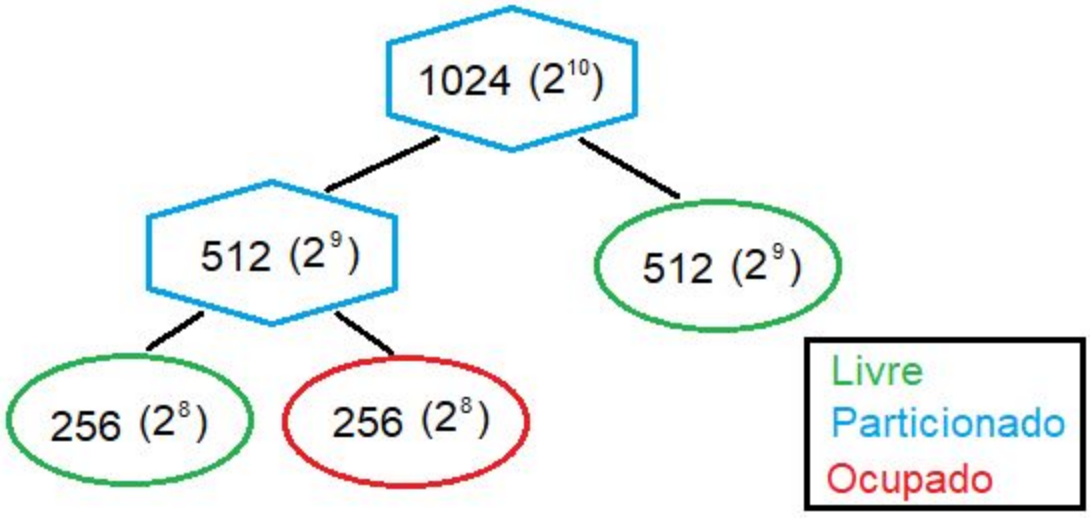

Figura 1: Exemplo de uma árvore binária de alocação de memória

Fatos sobre a árvore binária usada para gerenciamento de memória:

1) Ela nunca está vazia, existe nela sempre ao menos um nó na raiz, que concentra o tamanho total da memória (que deve ser uma potência de 2).
2) Ela é uma árvore binária completa. Ou seja, nenhum nó possui apenas um único filho.
3) Todos os nós tem tamanho igual a uma potência de 2. Nós na mesma altura possuem o mesmo tamanho. O menor tamanho possível de um nó é 1 (2 elevado a 0).
4) Independente do estado em que se encontra, o nó presente na raiz de uma árvore é aquele de maior tamanho dentre todos os seus descendentes.
5) Como a maioria dos programas (processos) não ocupam toda a memória que é dada a eles, essa técnica pode gerar muito desperdício de espaço. Esse desperdício é chamado de fragmentação. A fragmentação de um nó da árvore é a diferença entre o tamanho do nó e o tamanho do processo alocado nele. Por exemplo, se o nó tem tamanho 256 e o processo alocado tem tamanho igual a 248, a fragmentação será de 8. Apenas nós no estado **Ocupado** contribuem para a fragmentação total da árvore, pois apenas eles possuem processos alocados neles. A fragmentação total da árvore é a soma da fragmentação de todos os nós ocupados da árvore.

Assim como uma árvore binária qualquer, se aplicam a essa árvore de gerenciamento de memória os conceitos de caminhos (em ordem simétrica, pré-ordem ou pós-ordem). Entende-se por um caminho na árvore uma ordem na qual os nós dela são acessados (geralmente nesse caminho pode-se efetuar algum algoritmo específico, como por exemplo, contar o número de nós da árvore). Para alguns algoritmos, a ordem na qual os nós são visitados altera o resultado final. Um caminho em ordem simétrica (também chamado de ín-ordem) é aquele no qual recursivamente visita-se (processa) primeiro a subárvore esquerda, depois o pai (raiz) e por último a subárvore direita. Em um caminho em pré-ordem, primeiro visita-se o pai, depois as subárvores esquerda e direita, respectivamente. Por último, em um caminho em pós-ordem, primeiro visita-se as subárvores esquerda e direita, respectivamente, e depois o pai (veja a Figura 2).

Uma maneira prática de armazenar as informações de uma árvore consiste em gerar “sementes” da mesma. Uma semente geradora é uma representação sequencial dos nós da árvore, indicando a ordem na qual eles são visitados ao utilizarmos algum dos três tipos de percurso descritos (ordem simétrica, pré-ordem e pós-ordem). As sementes geradoras podem ser úteis para a reconstrução de uma árvore à partir de percursos dados na mesma.

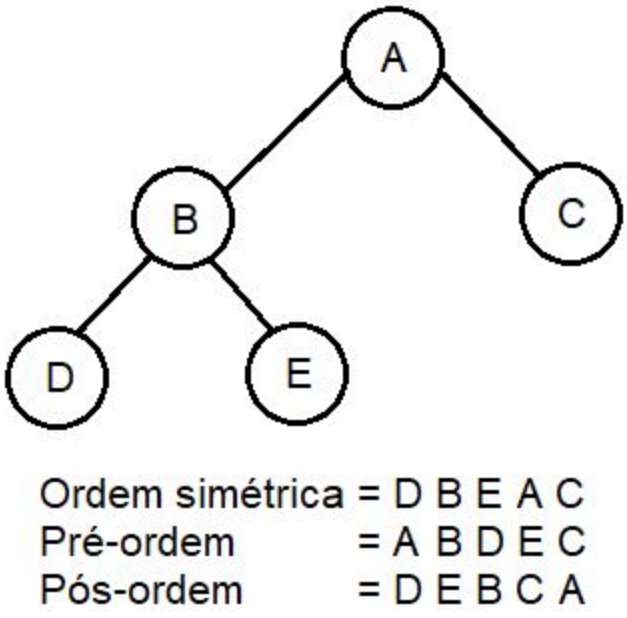

Figura 2: Exemplo de percursos em uma árvore binária. Cada percurso equivale a uma semente geradora

Dentre outras operações (veja a Tabela 1), as duas operações mais básicas que uma árvore binária de gerenciamento de memória necessita é a inserção e a remoção de processos.

A inserção de um processo na árvore é feita percorrendo a árvore em pré-ordem até que se chegue em um nó **Livre** adequado para a inserção. Se o tamanho do processo a ser inserido é maior do que o desse nó, a busca em pré-ordem continua. Se o processo é menor que o tamanho do nó livre, e ocupa mais da metade de seu tamanho, o processo é inserido nesse nó e o algoritmo termina. Caso contrário, se o tamanho do processo é menor ou igual a metade do tamanho do nó, devem ser criados dois filhos para esse nó com metade de seu tamanho, e esse pai passa a ter o estado de **Particionado**. Caso isso ocorra, deve-se continuar o percurso em pré-ordem para inserir o processo. É necessário que esse processo ocorra em cascata, visto que os filhos **Livres** gerados ainda podem ser muito maiores que o processo, necessitando que os filhos dos filhos continuem sendo particionados até que se gere um nó **Livre** com tamanho adequado. Caso a árvore seja percorrida inteiramente e nenhum nó tenha sido capaz de comportar o processo, a inserção falha pois não há espaço para ele no sistema.

A **Figura 3** apresenta um exemplo passo-a-passo da inserção. Considere um computador com capacidade de memória 1024 e inicialmente sem nenhum processo **(Quadro a na Figura 3)**. Imagine que deseja-se inserir um processo de tamanho 195. Iniciamos o percurso em pré-ordem a partir da raiz; como no início a raiz é um nó **Livre** de tamanho 1024, e 195 é menor que a metade de 1024, então essa raiz é transformada em um nó **Particionado** e recebe dois filhos no estado **Livre** com tamanho de 512, respectivamente. Continua-se o percurso em pré-ordem visitando a subárvore esquerda primeiro, chegando ao nó 512 **Livre** recém criado **(Quadro b na Figura 3)**. Como 195 ainda é menor que a metade de 512, esse nó também é transformado em um nó **Particionado** e recebe dois filhos no estado **Livre** com tamanho 256. O percurso em pré-ordem visita a subárvore esquerda desse nó 512, chegando ao nó 256 **Livre** **(Quadro c na Figura 3)**. Como 195 é menor que 256, mas ocupa mais da metade do nó (195 > 128), o processo de tamanho 195 é alocado nesse nó, que agora passa para o estado **Ocupado**, finalizando a inserção com sucesso **(Quadro d na Figura 3)**.

Agora desejamos inserir nessa árvore um outro processo de tamanho 492 **(Quadro e na Figura 3)**. O percurso se inicia da raiz de tamanho 1024, mas esta está no estado **Particionado**, portanto seguimos para a subárvore esquerda. A raiz da subárvore esquerda é um nó de tamanho 512, também **Particionado**, portanto a busca poderia continuar em suas subárvores esquerda e direita, mas podemos notar que qualquer nó da subárvore enraizada em 512 terá no máximo tamanho 256, que não é capaz de comportar o processo de tamanho 492 independente de seu estado. Partimos então para a subárvore direita da raiz 1024. Encontramos um nó **Livre** de tamanho 512 capaz de comportar o processo de tamanho 492, sem necessidade de ser particionado. O processo é inserido e o algoritmo finaliza.

Por fim, um processo de tamanho 257 precisa ser inserido **(Quadro f na Figura 3)**. Se tentar executar o percurso em pré-ordem, nenhum nó **Livre** será capaz de comportar esse processo. Mas, poderíamos economizar o tempo dessa busca se já soubéssemos que do espaço total disponível de 1024, já temos uma partição de 512 e outra de 256 ocupadas (ou seja, 768 de memória em uso), portanto no máximo uma partição de 256 está livre, e não é capaz de suportar um processo de 257.

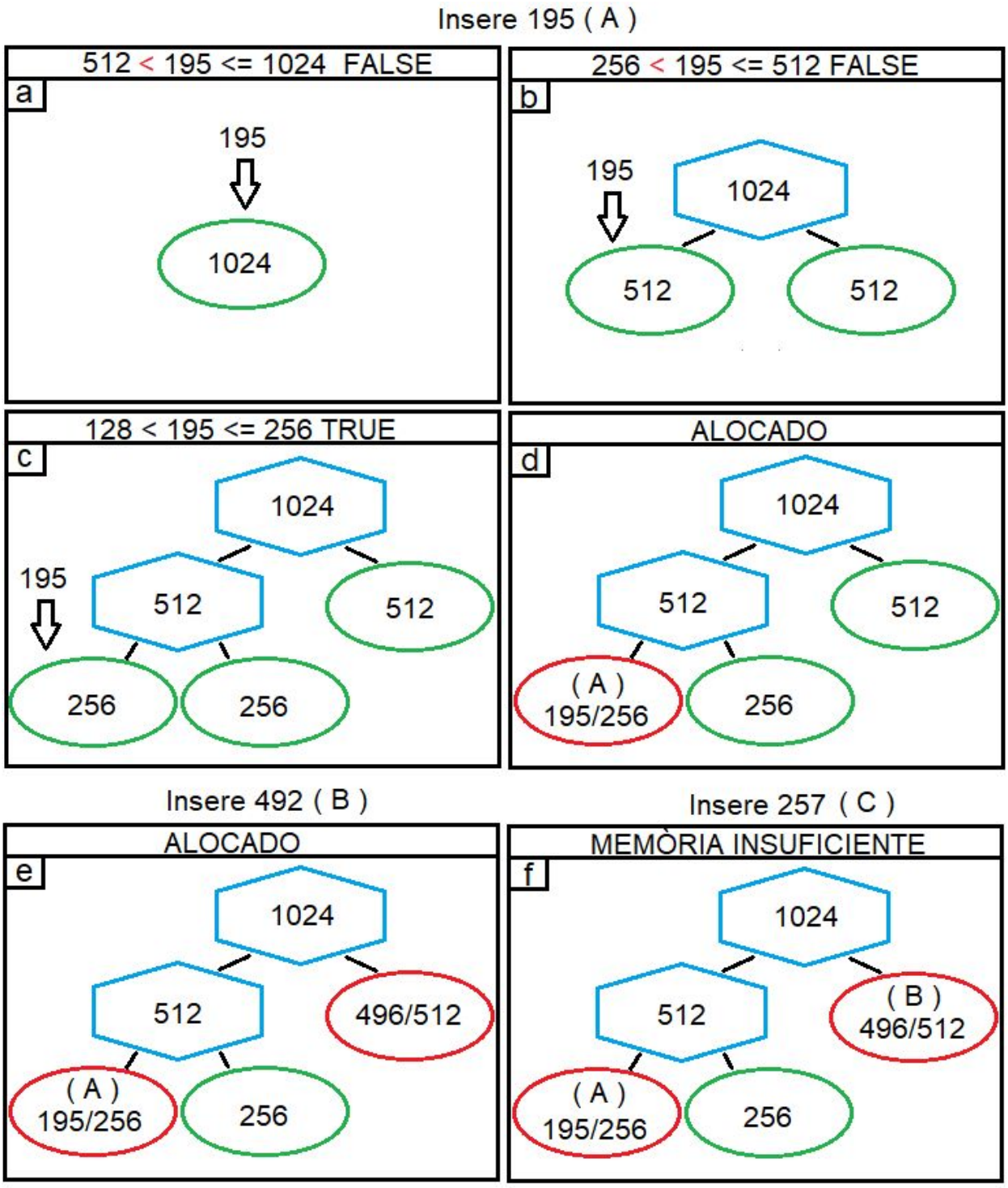

Figura 3: Exemplos de inserção na árvore binária de alocação de memória

A remoção de um processo na árvore de gerenciamento de memória é feita percorrendo a árvore em pré-ordem, até que o processo que deseja-se remover seja encontrado em um nó **Ocupado**. Encontrado o nó no qual o processo esteja alocado, deve-se transformar esse nó em **Livre** e remover as informações relativas ao processo desse nó. Ao transformar esse nó em **Livre**, caso o irmão dele também esteja **Livre**, ambos devem ser removidos da árvore, e o pai de ambos passa do estado **Particionado** para **Livre**. Assim como na inserção, esse procedimento deve ser feito em cascata, visto que ao transformar o nó pai em **Livre**, caso o irmão dele (tio do nó inicialmente removido) também seja **Livre**, o procedimento deve continuar, transformando o pai do pai em **Livre**, e assim por diante. Os nós filhos precisam ser desalocados para manter as definições de **Livre** (nó folha) e **Particionado** (nó interno).

A **Figura 4** apresenta um exemplo passo-a-passo da remoção. Partindo da árvore criada na **Figura 3**, deseja-se remover o processo **A**. O percurso em pré-ordem é iniciado na raiz de tamanho 1024, que está em estado **Particionado**. A busca é efetuada pelo primeiro processo na subárvore esquerda, chegando ao nó de tamanho 512 também em estado **Particionado**. Continua-se a busca na subárvore esquerda, chegando ao nó de tamanho 256 **Ocupado,** que contém o processo que se deseja remover **(Quadro a na Figura 4)**. As informações referentes ao processo são removidas e o nó passa do estado **Ocupado** para o estado **Livre** **(Quadro b na Figura 4).**

Ao voltar para o nó pai do nó que comportava o processo removido, nota-se que o filho direito dele estava no estado **Livre**, e seu filho esquerdo que antes estava **Ocupado** também passou para o estado **Livre** **(Quadro c na Figura 4)**. Como ambos filhos estão livres, eles são removidos da árvore e o nó pai se torna um nó folha, passando do estado **Particionado** para **Livre** **(Quadro d na Figura 4)**. O procedimento volta para o pai do nó 512, a raiz de tamanho 1024. O nó filho esquerdo da raiz passou para o estado **Livre**, mas seu filho direito ainda está no estado **Ocupado**, portanto a raiz não pode efetuar a mesma retração descrita, e a remoção do processo termina.

Em seguida, tenta-se remover um nó C, mas após percorrer a árvore inteira em pré-ordem, nenhum processo C é encontrado, portanto a remoção falha **(Quadro e na Figura 4)**. Por fim, deseja-se remover o processo B. O percurso se inicia na raiz 1024 de estado **Particionado**, e segue para sua subárvore esquerda, encontrando um único nó de tamanho 512 **Livre**. Depois, passa para sua subárvore direita encontrando o nó de tamanho 512 **Ocupado** que contém o processo B. Efetua-se o mesmo processo de remover as informações do processo do nó e transformá-lo em **Livre**. Ao voltar para seu pai, a raiz de tamanho 1024, seus dois filhos estão no estado **Livre**, portanto é executado o procedimento que remove os nós filhos, e o nó raiz volta a ser o único nó da árvore, de tamanho 1024 e em estado **Livre** **(Quadro f na Figura 4).**

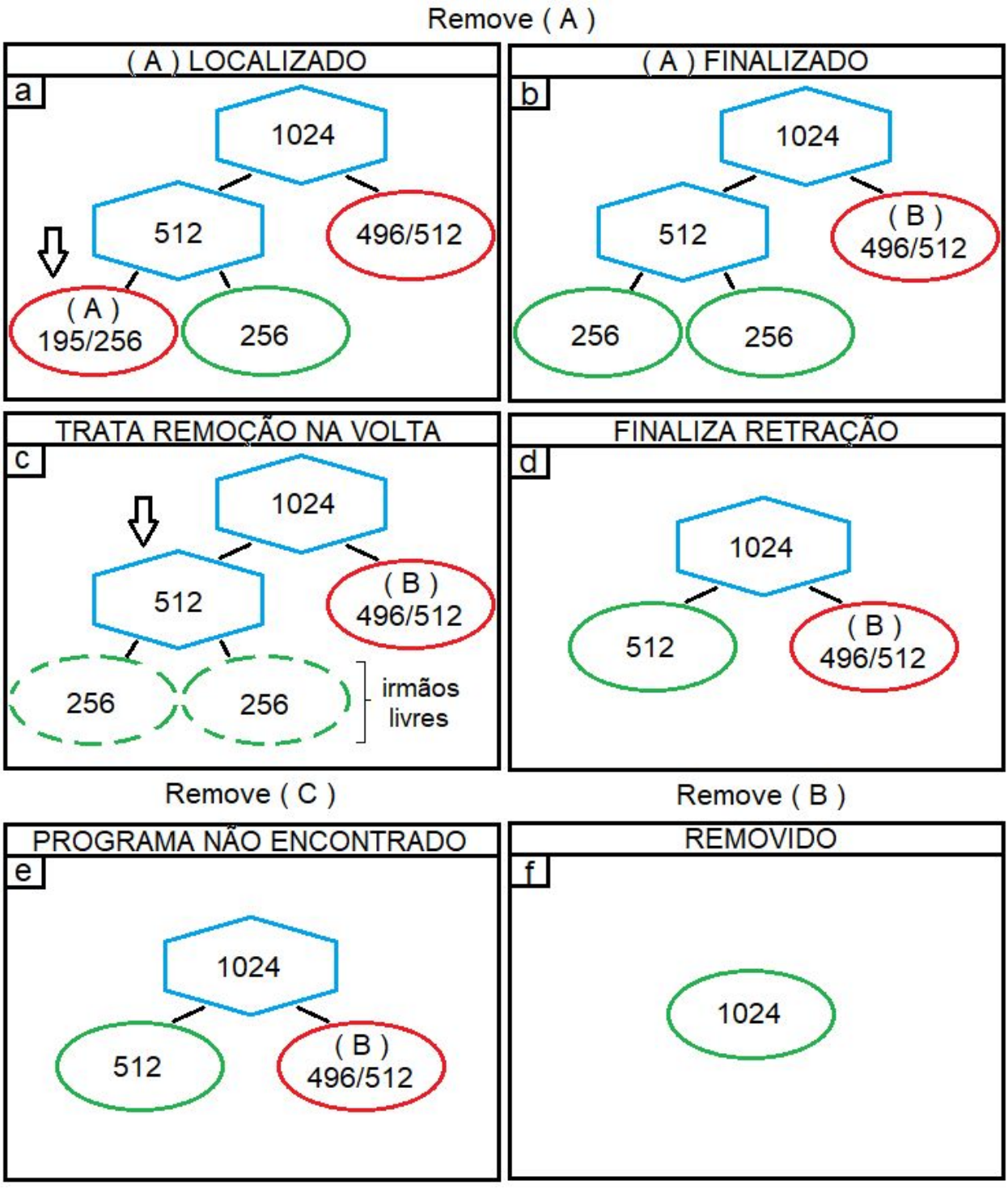

Figura 4: Exemplos de remoção na árvore binária de alocação de memória

Sua dupla foi designada para implementar a árvore de gerenciamento de memória de um SO. O programa deve seguir fielmente a estrutura descrita, possuindo as operações básicas de inserção e remoção, e outras que são apresentadas na Tabela 1.

2 - Entrada

A primeira linha deve conter um número inteiro **M**, esse número indica que o tamanho total da memória do computador será 2 elevado a **M** (2^{M}). Em seguida, deve-se informar as operações de entrada. Seu sistema deve implementar todas as operações descritas na Tabela 1. Cada operação é indicada por um comando numérico nas linhas de entrada e exigem a entrada de parâmetros adicionais. Uma entrada deve ser fornecida com várias linhas. Cada linha deve conter vários números inteiros, sendo o primeiro deles o código da operação (de acordo com #Operação na Tabela 1) e os demais os parâmetros conforme a coluna Parâmetros de cada operação.

Tabela 1: Operações de Entrada

#Operação	Nome	Parâmetros	Descrição
1	Inicia processo	Int : cod Int : size	Inicia um processo de código **cod** e tamanho **size** no SO, caso haja memória suficiente para comportá-lo. Para isso, deve-se encontrar ou criar um nó folha na árvore de alocação de memória tal que seu tamanho seja a primeira potência de dois maior ou igual a **size** (veja exemplos na Figura 3).
2	Finaliza processo	Int : cod	Dado um código **cod**, finaliza o processo identificado por ele no SO caso ele exista. Para isso, remova o nó correspondente da árvore de alocação de memória, lembrando de efetuar as devidas retrações de folhas quando necessário (veja exemplos na Figura 4).
3	Calcula fragmentação	-	Para todos os processos presentes atualmente no sistema (alocados em nós do estado **Ocupado**), calcula a quantidade total de memória desperdiçada por conta da fragmentação no sistema. Entende-se por fragmentação a porção de memória que cada processo recebe, mas não utiliza (veja o fato 5 do enunciado).
4	Relatório do sistema	-	Percorre a árvore de alocação de memória, contando quantos nós pertencem a cada estado (**Particionado**, **Ocupado** e **Livre**) e quantos por cento da memória total está sendo utilizada. Essa porcentagem é dada pela soma dos tamanhos de todos os processos presentes na árvore (em nós

			Ocupado) dividido pelo quantidade total de memória da árvore (ou seja, o valor presente na raiz da árvore, dado na entrada do programa).
5	Imprime sementes geradoras	-	Imprime as três sementes geradoras da árvore de alocação de memória. Devem ser impressas as sementes em ordem simétrica, pré-ordem e pós-ordem, respectivamente. As informações de cada nó devem ser impressas de acordo com o modelo descrito na Tabela 2.
6	Imprime processos	-	Imprime uma lista de processos presentes (ou seja, aqueles que estão alocados em nós no estado **Ocupado**) na árvore de alocação de memória atualmente. A impressão deve ser feita em ordem simétrica.

Exemplo de Entrada:

```
10
1 111 195
1 222 496
1 333 257
3
4
5
6
2 111
2 333
2 222
3
4
5
6
```

3 - Saída

Tabela 2: Especificação da Saída das Operações

#Operação	Saída
1	Processo (**cod**) de tamanho **size** inicializado com sucesso Ou Memoria insuficiente

2	Processo (**cod**) finalizado com sucesso Ou Nao existe processo de codigo **cod** inicializado no sistema
3	Quantidade total de memoria desperdicada por fragmentacao: **mem**
4	[RELATORIO DE SISTEMA] **ocup** Ocupados **livre** Livres **part** Particionados Memoria utilizada = **mem** / 100 Obs.: A porcentagem deve ser expressa como um inteiro truncado. Considere que a memória ocupada pelo processo é o tamanho da partição inteira, não apenas da parte que o processo ocupa (ou seja, o tamanho do processo). Use variáveis do tipo double para precisão correta. Por exemplo, se a quantidade de memória ocupada é 194 de um total de 256, temos a porcentagem 0.7578, multiplicado por 100 e truncado, ficamos com 75.
5	[SEMENTES GERADORAS] Sim = **(L:512)(P:1024)(O:496/512[862])** Pre = **(P:1024)(L:512)(O:496/512[862])** Pos = **(L:512)(O:496/512[862])(P:1024)** Obs.: Um nó **Livre** é representado no formato **(L:cap)**. Um nó **Particionado** é representado no formato **(P:cap)** Um nó **Ocupado** é representado no formato **(O:p/cap[cod])** **cap** indica a capacidade de armazenamento do nó **p** indica o tamanho de um processo alocado no nó **Ocupado** **cod** indica o código do processo alocado no nó **Ocupado** Obs2.: O exemplo acima se refere ao quadro **e** da Figura **4**
6	[PROCESSOS PRESENTES NA MEMORIA] **Mem_ocupada0** [Processo: **cod0**] **Mem_ocupada1** [Processo: **cod1**] … **Mem_ocupadaN** [Processo: **codN**]

	Ou [PROCESSOS PRESENTES NA MEMORIA] Nenhum processo presente Obs.: **Mem_ocupadaX** se refere ao tamanho da partição que o processo **codX** está utilizando.

Exemplo de saída:

```
Processo (111) de tamanho 195 inicializado com sucesso
Processo (222) de tamanho 496 inicializado com sucesso
Memoria insuficiente
Quantidade total de memoria desperdicada por fragmentacao: 77
[RELATORIO DE SISTEMA]
2 Ocupados
1 Livres
2 Particionados
Memoria utilizada = 75 / 100
[SEMENTES GERADORAS]
Sim = (O:195/256[111])(P:512)(L:256)(P:1024)(O:496/512[222])
Pre = (P:1024)(P:512)(O:195/256[111])(L:256)(O:496/512[222])
Pos = (O:195/256[111])(L:256)(P:512)(O:496/512[222])(P:1024)
[PROCESSOS PRESENTES NA MEMORIA]
256 [Processo: 111]
512 [Processo: 222]
Processo (111) finalizado com sucesso
Nao existe processo de codigo 333 inicializado no sistema
Processo (222) finalizado com sucesso
Quantidade total de memoria desperdicada por fragmentacao: 0
[RELATORIO DE SISTEMA]
0 Ocupados
1 Livres
0 Particionados
Memoria utilizada = 0 / 100
[SEMENTES GERADORAS]
Sim = (L:1024)
Pre = (L:1024)
Pos = (L:1024)
[PROCESSOS PRESENTES NA MEMORIA]
Nenhum processo presente
```

4 - Informações Adicionais

- O laboratório é em dupla;
- Não há número máximo de submissões;

- O programa deve estar completamente contido em um único arquivo nomeado lab06.c;
- Apenas um dos integrantes da dupla deve submeter a solução;
- No início do arquivo inclua como comentário, o nome e o RA dos integrantes da dupla, além de uma breve descrição dos objetivos do programa, as entradas e as saídas esperadas;
- Documente a solução através de comentários ao longo do programa e indente corretamente o código para melhor legibilidade;

Submissões detectadas como plágio receberão conceito zero.

5 - Critérios de Avaliação

- Seu programa deve passar pelos casos de teste definidos para o laboratório. Se positivo, os critérios de avaliação em seguida serão analisados:
    - Respeitou o enunciado;
    - Usou a estrutura de dados mais indicada para a solução;
    - Alocou e liberou memória adequadamente;
    - Organizou e indentou bem o código;