# Treaps

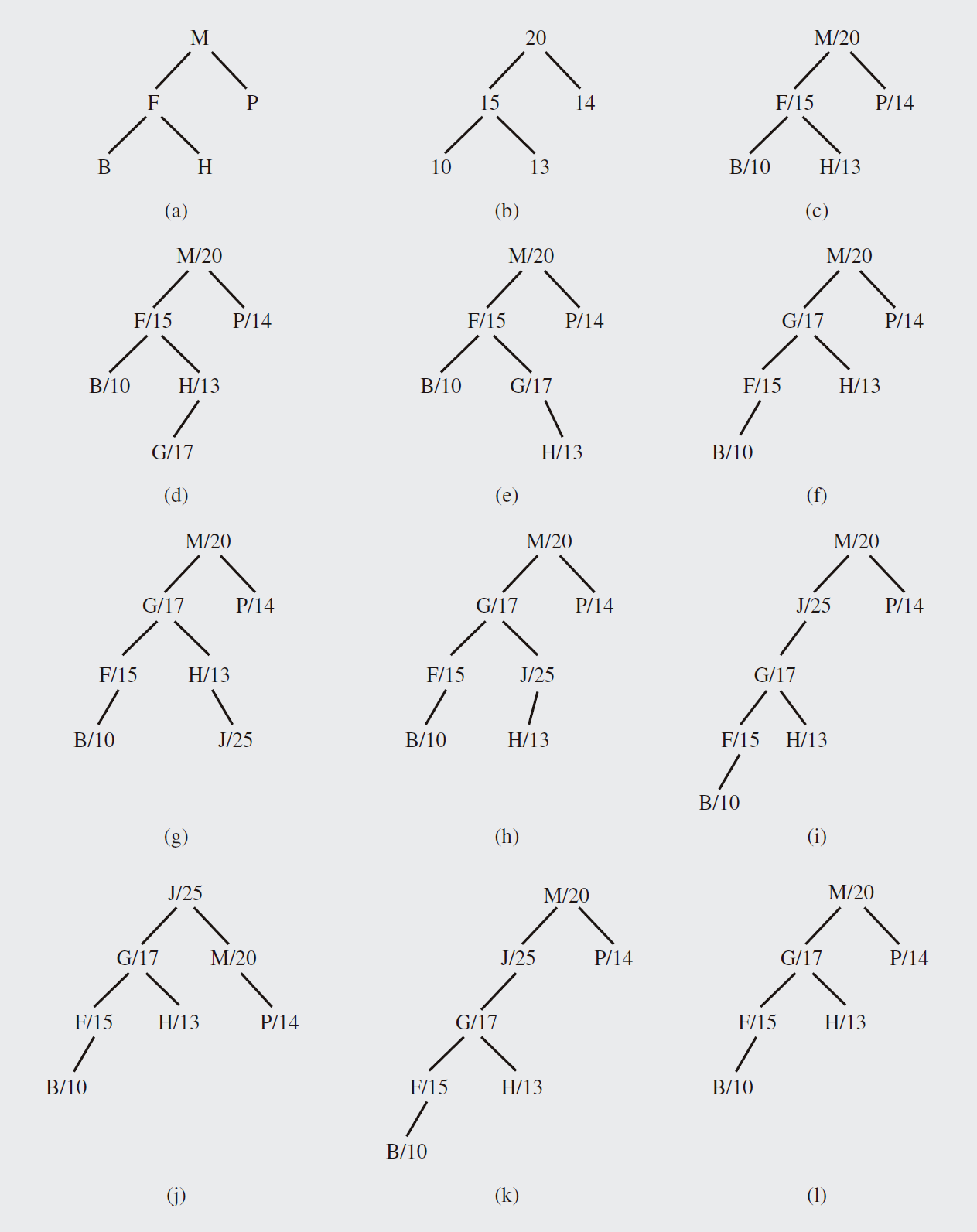
Os *Treaps* são muito atraentes porque são árvores perfeitamente equilibradas e permite acesso imediato ao maior elemento no heap máximo. No entanto, eles não não têm permissão para acesso rápido a qualquer outro elemento.

A pesquisa é realizada de forma muito eficiente na árvore de busca binária, mas a forma da árvore depende da ordem das inserções e exclusões e a árvore podem se tornar severamente disformes se nenhuma provisão for feita para equilibrá-la. No entanto, é possível combinar uma árvore de pesquisa binária e um heap em uma estrutura de dados, um treap, que também é refletido em seu nome.

Note, no entanto, que heap é entendido nesta seção em um sentido mais fraco como a árvore binária com a propriedade heap, mas a condição estrutural que requer que a árvore seja perfeitamente balanceada e as folhas nas posições mais à esquerda seja desconsiderada, embora destinam-se a ser tão perfeitamente equilibrados quanto possível.

Um treap é uma árvore de pesquisa binária, ou seja, uma árvore que usa uma chave de dados como na árvore de busca binária regular, e uma chave adicional, uma prioridade, e a árvore é também um heap organizado de acordo com tais prioridades. Por exemplo, uma árvore de pesquisa binária com a figura abaixo (a), e um heap máximo de acordo a figura abaixo (b) juntos formam um tripe na (c).

Considere uma árvore para pares de números (x, y). Isso torna uma árvore cartesiana onde xy são coordenadas de um ponto no plano cartesiano, x é usado para organizar o ponto treap como uma árvore de pesquisa binária e y para organizá-la como um heap (Vuillemin 1980). Outro possibilidade é gerar aleatoriamente o número y para uma determinada chave de dados x em cima a inserção da chave x na árvore (Seidel and Aragon 1996). Isso torna um árvore randomizada com o tempo esperado de operações de pesquisa, inserção e exclusão sendo de ordem O (lg n).

Quando um item deve ser inserido em um treap, primeiro, uma prioridade é gerada aleatoriamente para este item, então um lugar para este item é encontrado na árvore, onde está inserido como uma folha. Se a prioridade de seu pai for maior que a prioridade que o nó acabou de inserir, nada mais precisa ser feito. Caso contrário, o novo nó é rotacionado sobre seu pai e, possivelmente, sobre seu novo pai, etc., até que um pai seja encontrado, cuja prioridade é maior que a prioridade do novo nó (a menos que, é claro, o novo nó seja rotacionado o caminho para a raiz). Por exemplo, a inserção de G e a prioridade gerada 17 em o treap na como mostra na figura (c) fornece uma árvore de busca binária (d) que requer dois rotações do nó G para restaurar a propriedade heap na árvore (e, f). Inserção de J com prioridade 25 para este treap dá uma árvore de pesquisa binária (g) que requer três rotações para torná-lo um heap também (h-j). 

## Remoção

Para excluir um nó x de um treap, seu filho com maior prioridade deve ser rotacionado sobre o nó, então o mesmo deve ser feito para um novo filho de x, até que o nó x tem apenas um filho ou se torna uma folha, caso em que pode ser facilmente excluído. Por exemplo, para excluir o nó J do tripe na Figura (j), primeiro, o filho M de J com maior prioridade do que a outra criança G é girada sobre J (k), após o qual o nó J pode ser removido já que tem apenas um filho (l).

Em outro método, os nós são armazenados em uma matriz. O treap que funciona como um mini caixa que armazena apenas o índice i do local ocupado por um item; o índice também é usado como a prioridade deste item. Para inserir um item, um índice i ≤ n é gerado aleatoriamente. E se i = n, o item é colocado na posição n do array e o item é inserido no treap. Caso contrário, um item atualmente ocupando a posição i é movido para a posição n através de uma série de rotações para se tornar uma folha, sendo assim rebaixado para prioridade n; o novo item é colocado na posição i e depois inserido no tripe com prioridade i. Considere um treap nas Figuras acima (a) e na figura abaixo (a), que é realmente armazenado como mostrado na Figura abaixo (b): um binário árvore com índices em campos de informação de seus nós, onde índices indicam posições de itens nos dados da matriz [] e também funcionam como prioridades. Suponha que para o item com o chave G a ser inserida, o índice i = 1 é gerado aleatoriamente. Ou seja, a situação no array deve ser como ilustrado na Figura abaixo (c): G é colocado na posição 1, depois que F o liberou sendo movido para a posição 5. Para refletir essa situação no treap, primeiro, o campo de informações no nó correspondente à tecla F é alterado para 5 (c) e depois, após dois rotações, o nó torna-se uma folha (Figuras abaixo d, e com uma visão conceitual do trecho (somente chaves) e sua aparência real (somente prioridades)). Em seguida, a tecla G é usada para inserir nó no treap (Figura abaixo (f)). Para transformar a árvore em um heap, o novo nó é girado três vezes em seu devido lugar (Figura abaixo (i)).

