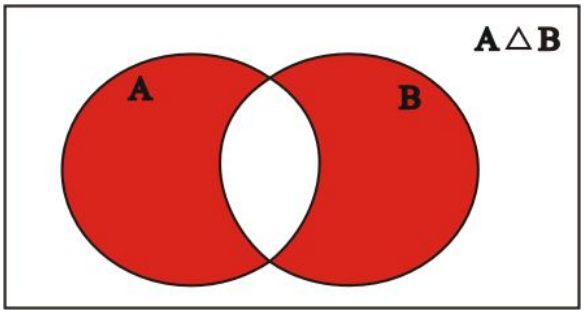
MC202 — ESTRUTURAS DE DADOS

Laboratório 11 — Diferença Simétrica

Neste laboratório, vamos revisitar a tarefa do laboratório 01, mas dessa vez usando uma estrutura de dados mais eficiente para resolver o problema.

Diferença simétrica

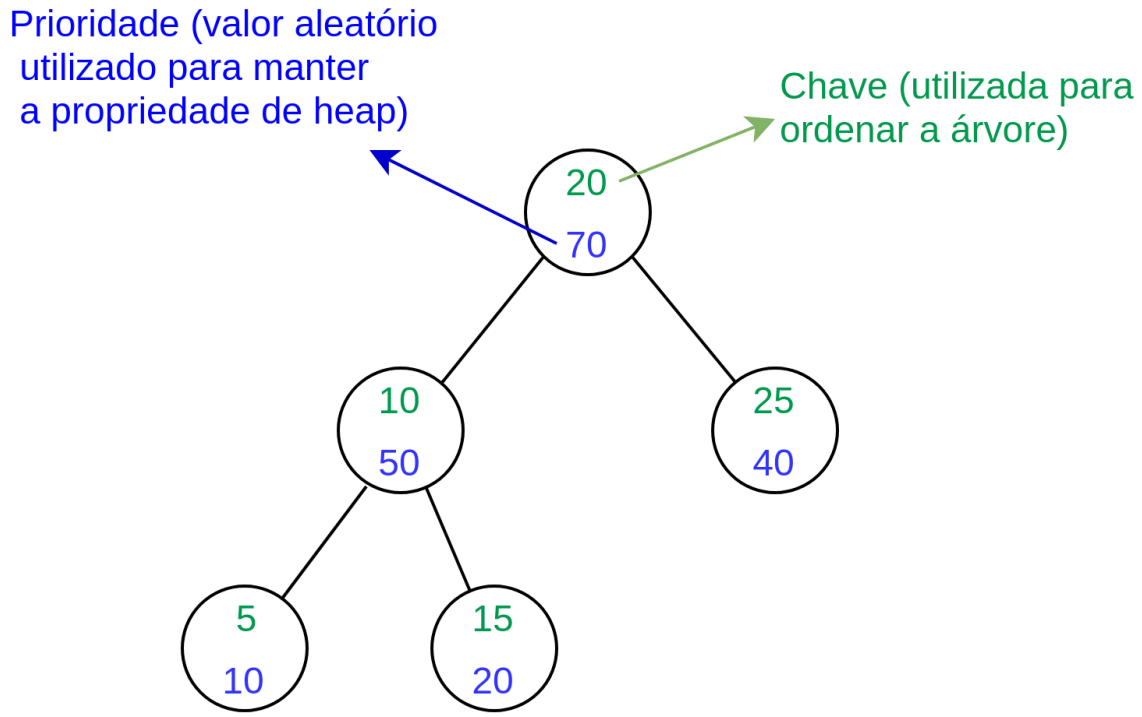
No laboratório 01, o programa recebia 2 vetores de inteiros e deveria achar a **diferença simétrica** deles, que consiste em elementos que pertencem a algum dos conjuntos mais não em ambos. Veja o diagrama de Venn que representa a diferença simétrica de $\mathbf{A}$ e $\mathbf{B}$.



Desta vez, não há limite para o tamanho dos conjuntos e pode-se esperar que eles sejam muito grandes. Para tornar mais eficiente o processo de verificar se um elemento aparece em um conjunto, será necessário armazenar o conjunto em uma árvore balanceada.

Árvore binária aleatória de busca

Uma árvore binária aleatória de busca do tipo Treap, assim como as árvores AVL e Rubro-Negra, é uma árvore binária de busca balanceada, mas não possui altura garantidamente de no máximo $O(log\ N)$. A ideia desse tipo de árvore é usar a aleatoriedade e a propriedade de Heap binário máximo para manter a árvore balanceada com alta probabilidade. Dessa forma, o tempo esperado para inserção, remoção e busca é $O(log\ N)$. Na Treap, cada nó possui uma *chave* e uma *prioridade*. A chave é o valor inserido na árvore e segue a mesma ideia de uma árvore binária comum (os menores valores vão para esquerda e os maiores para a direita). Já a prioridade de cada nó é dada por um valor obtido aleatoriamente durante a inserção e é utilizada para manter a chamada propriedade de Heap da árvore. Observe um exemplo de Treap na imagem a seguir.



Propriedade de heap binário máximo

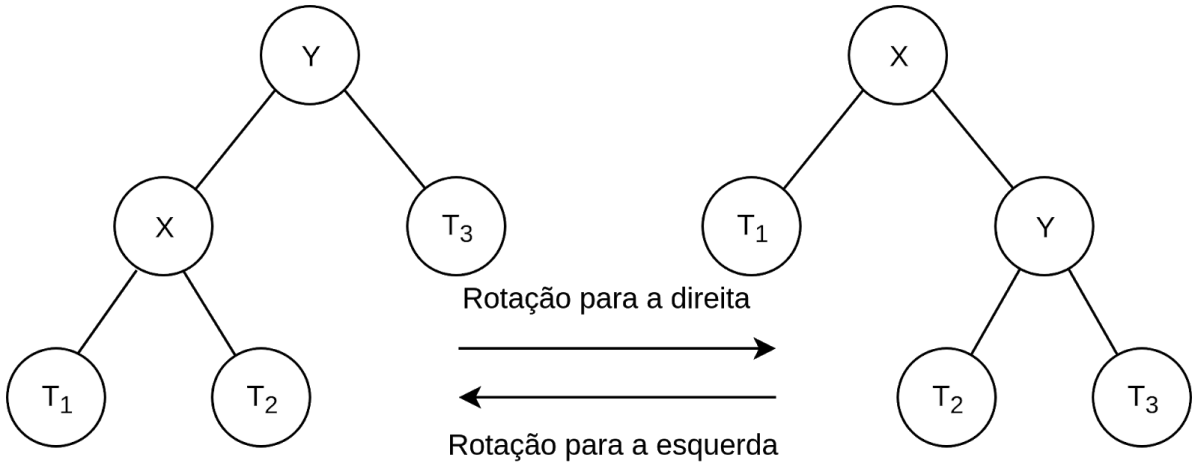
Dizemos que uma árvore satisfaz a propriedade de heap binário máximo se ela respeitar o seguinte critério:

- A prioridade de cada nó deve ser maior ou igual que as prioridades dos filhos (da esquerda e da direita).

Para manter esse critério, teremos que transformar a nossa árvore binária. É isso que vai deixá-la “provavelmente” balanceada (neste laboratório, não vamos entrar em detalhes no motivo disso).

Rotação

Assim como as demais árvores balanceadas, a Treap utiliza as operações de rotação para a esquerda e rotação para a direita durante a inserção e remoção. Em uma árvore Treap, essas operações são utilizadas para manter a propriedade de Heap máxima.



No exemplo mostrado acima, a rotação acontece entre os nós X e Y. Suponha que o valor aleatório de X é maior do que o de Y, nesse caso é necessário alterar a árvore para que X seja pai de Y. Trocar X e Y de lugar não é uma opção, pois o valor do conjunto de X é menor que o de Y, então trocar eles de lugar desrespeitaria o primeiro critério apresentado acima. Fazemos então uma rotação à direita. Para isso, removemos a subárvore a direita de X e substituímos ela com Y, que agora não vai mais possuir uma subárvore esquerda, já que ela era o X. Por fim, pegamos a subárvore da direita removida de X e colocamos ela a esquerda de Y.

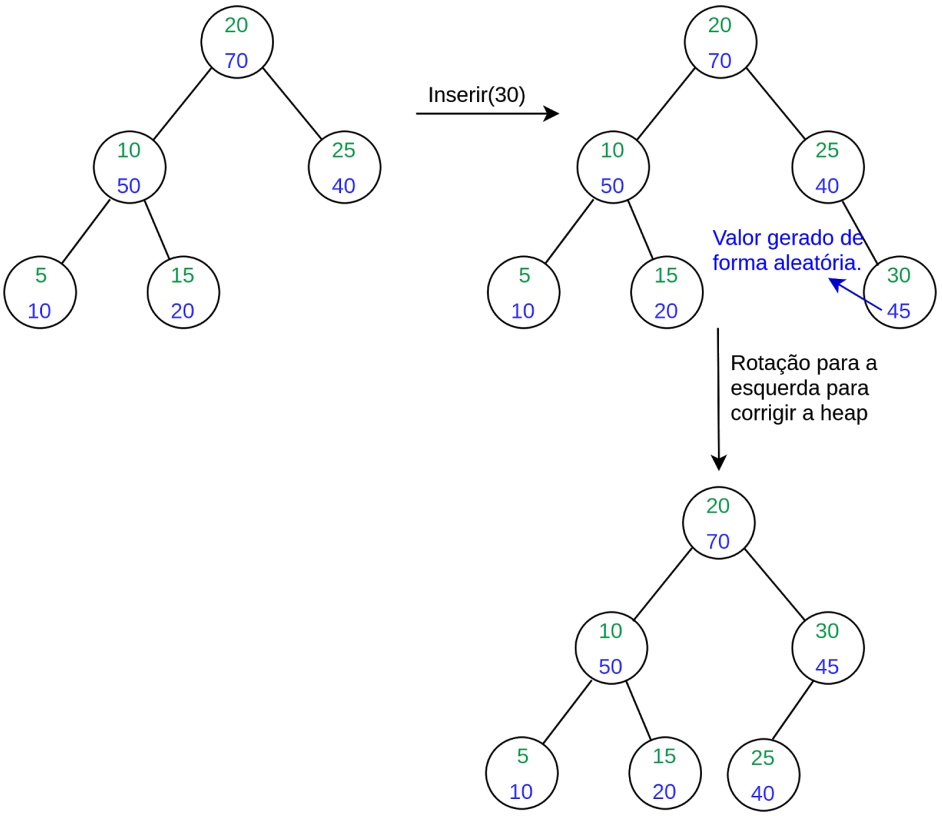
Isso faz com que a subárvore com X como raiz respeito os dois critérios, mas uma vez feito isso devemos verificar se X também é maior que o seu novo pai (antigo pai de Y) e possivelmente realizar uma nova rotação. Como todos os nós inseridos inicialmente serão folhas, provavelmente serão necessárias várias rotações para fazer com que toda a árvore respeite os dois critérios.

Busca

A busca em uma árvore Treap é realizada da mesma maneira que a busca em uma árvore binária comum.

Inserção

Durante a inserção, é criado um novo nó com a chave recebida como parâmetro e com uma prioridade aleatória. Esse nó é inserido na árvore da mesma maneira que a inserção é realizada em uma árvore binária comum. Após a inserção do novo nó, a propriedade de Heap máxima pode ter sido perdida, nesse caso é necessário utilizar as operações de rotação para refazer a Heap.



Ponteiros para o nó pai

Normalmente uma árvore binária não precisa que cada nó contenha um apontador para o pai, mas nesse caso isso será necessário por dois motivos:

- É uma informação importante para realizar a rotação.
- Simplifica iterações sobre a árvore caso não se opte por utilizar a recursão para fazer fazer isso.

Tarefa

Neste laboratório, o programa deve receber **N** conjuntos de números inteiros (onde **N** será especificado na entrada do programa), denotados por A_1, A_2, **...,** A_n, e determinar a diferença simétrica entre todos eles, denotado por $S = ((A_1 \Delta A_2) \Delta ...) \Delta A_n$.

O programador deve utilizar a Treap para armazenar os elementos de um conjunto. Os números aleatórios usados para garantir a propriedade de Heap binária máxima devem ser gerados pelo aluno através da função rand() da biblioteca `stdlib.h`.

Apesar de normalmente a ordem dos elementos de um conjunto não ser relevante, para esse programa, **é necessário que o conjunto de saída seja impresso em ordem decrescente.**

Entrada

A primeira linha da entrada é composta por um inteiro **N** que representa o número de conjuntos que serão informados a seguir.

Cada uma das **N** linhas consecutivas representa um conjunto da seguinte forma: o primeiro inteiro **M** é o tamanho do conjunto, os **M** inteiros seguintes serão os elementos do conjunto.

Saída

O seu programa deve produzir uma única linha de saída correspondente ao conjunto $\mathbf{S}$. O formato da saída apresentada abaixo deve ser seguido rigorosamente, lembrando que os elementos devem ser impressos em ordem **decrescente**.

Para facilitar a impressão a saída deve conter um espaço após o conjunto e antes da quebra de linha. (Normalmente o último elemento impresso não deveria vir acompanhado de um espaço antes da quebra de linha, mas para facilitar a implementação desse trabalho colocamos o espaço no final).

Exemplo

Entrada

```
4
5 0 1 2 3 4
2 4 5
8 2 4 6 8 10 12 14 16
4 10 11 12 13
```

Saída

```
16 14 13 11 8 6 5 3 1 0
```

Dicas

Usar estruturas diferentes

Uma ideia para simplificar a implementação do lab é primeiro usar estruturas mais simples (como árvores não balanceadas, vetores ou até listas) e depois que o lab estiver

funcionando para instâncias pequenas trocar a estrutura de dados por uma mais eficiente, terminando na Treap. (O lab entregue deve implementar uma Treap).

Uma forma mais simples de fazer isso seria usar interfaces semelhantes para acesso a estrutura, por exemplo:

`void inserir_elemento(tipo_conjunto *c, unsigned int elemento);`
Função que insere um elemento no conjunto c

`int checar_elemento(tipo_conjunto *c, unsigned int elemento);`
Função que verifica se o elemento já está presente no conjunto c

`tipo_conjunto* criar_conjunto_vazio(void);`
Função que retorna um conjunto vazio para depois ser preenchido.

Lembre-se que isso é apenas um exemplo. A interface vai acabar tendo outras funções, a ideia é tentar criar uma interface que não esteja atrelada a estrutura de dados que será usada.

Critérios específicos

- Para as turmas E e F, este laboratório tem peso 4.
- Para as turmas G e H, este laboratório tem peso 2.
- Deverão ser submetidos os seguintes arquivos:
    - treap.c (implementação da treap)
    - treap.h (interface da treap)
    - lab11.c (programa principal cuja solução utiliza treap)
- Tempo máximo de execução: 1 segundo.

Testando

Para compilar com o Makefile fornecido e testar um caso de teste:

```
make
./lab11 < arq01.in > arq01.out
diff arq01.res arq01.out
```

onde `arq01.in` é a entrada (casos de testes disponíveis no SuSy), `arq01.out` é a saída do seu programa e `arq01.res` é a solução provida para a entrada arq01.in. O Makefile também contém uma regra para testar todos os testes de uma vez; nesse caso, basta digitar:

`make testar_tudo`

Observações gerais

No SuSy, haverá 3 tipos de tarefas com siglas diferentes para cada laboratório de programação. Todas possuirão os mesmos casos de teste. As siglas são:

1. **SANDBOX:** Esta tarefa serve para testar o programa no SuSy antes de submeter a versão final. Nessa tarefa, tanto o prazo quanto o número de submissões são ilimitados, porém arquivos submetidos aqui **não serão corrigidos**.
2. **ENTREGA:** Esta tarefa tem limite de **uma única submissão** e serve para entregar a versão final dentro do prazo estabelecido para o laboratório. Não use essa tarefa para testar o seu programa: submeta aqui quando não for mais fazer alterações no seu programa.
3. **FORAPRAZO:** Esta tarefa tem limite de **uma única submissão** e serve para entregar a versão final após o prazo estabelecido para o laboratório, mas com nota reduzida (conforme a ementa). O envio nesta tarefa irá substituir a nota obtida na tarefa ENTREGA apenas se o aluno tiver realizado as correções sugeridas no feedback ou caso não tenha enviado anteriormente em ENTREGA.

Observações sobre SuSy:

- Versão do GCC: C-ANSI 4.8.2 20140120 (Red Hat 4.8.2-15).
- Flags de compilação:
  `-ansi -Wall -pedantic-errors -Werror -g -lm`
- Utilize comentários do tipo `/* comentário */`;
  comentários do tipo `//` serão tratados como erros pelo SuSy.

Além das observações acima, esse laboratório será avaliado pelos critérios gerais:

- Indentação de código e outras boas práticas, tais como:
  - uso de comentários (apenas quando forem relevantes);
  - código simples e fácil de entender;
  - sem duplicidade (partes que fazem a mesma coisa).
- Organização do código:
  - tipos de dados criados pelo usuário e funções bem definidas e tão independentes quanto possível.
- Corretude do programa:

- programa correto e implementado conforme solicitado no enunciado;
- inicialização de variáveis sempre que for necessário;
- dentre outros critérios.