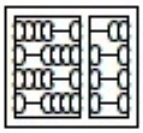


Instituto de Computação
Universidade Estadual de Campinas
MC202 Estruturas de Dados (A, B, C, D)

Cobertura de Conjuntos

1 Problema

O Problema de Cobertura de Conjuntos tem como entrada um universo $\mathcal{U}$ de n elementos e uma coleção $\mathcal{S}$ de subconjuntos de $\mathcal{U}$. A saída é uma coleção dos subconjuntos de $\mathcal{S}$ tal que sua união resulta em $\mathcal{U}$. Existem diversas abordagens para este problema, uma delas se baseia em escolher, iterativamente, o subconjunto que possui o maior número de elementos ainda não cobertos. Esse é um método *greedy* dado pelo algoritmo seguinte

$U = \mathcal{U}$; $S = \mathcal{S}$; $R = \{\}$
while $U \neq \varnothing$
    $M =$ item de S que cobre a maior quantidade de elementos em U
    $S = \{C \setminus M \mid C \in S\}$
    $U = U \setminus M$
    $R = R \cup \{M\}$
return R

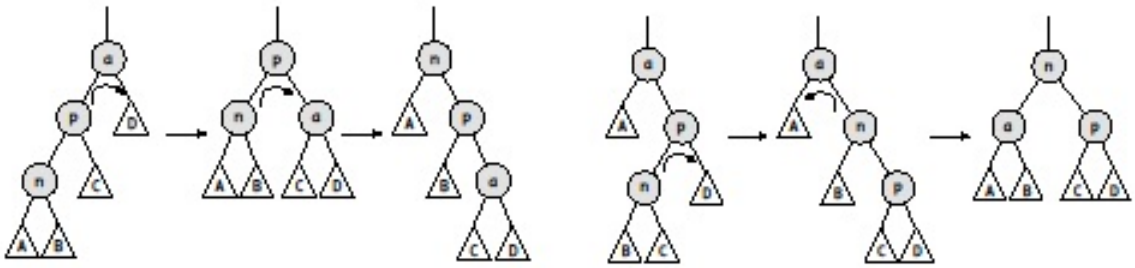
Desta vez, sua tarefa é implementar um programa que realiza a cobertura de conjuntos pelo método *greedy* descrito. Os conjuntos serão formados por números inteiros e devem ser representados utilizando árvores binárias de busca (ABB) autoajustáveis por afunilamento.

Esse tipo de árvore seria exatamente igual a ABB se não fosse pelo afunilamento. O afunilamento consiste em realizar rotações sempre que uma operação é aplicada sobre a árvore, como inserir ou remover um elemento. Existem três tipos de rotações que podem ser feitas de acordo com o posicionamento de um nó n:

1. Um nó n é filho esquerdo de seu pai p, e p é filho esquerdo do avô, a, de n. Neste caso, primeiro é feita uma rotação em torno de p e depois uma em torno de n;
2. Um nó n é filho esquerdo de seu pai p, e p é filho direito do avô, a, de n. Neste caso, são feitas duas rotações em torno de n;
3. Um nó n é filho esquerdo de seu pai p, e p é a raiz da árvore. Apenas uma rotação é realizada em torno de n.

Figura 1: Árvore autoajustável (Jorge Stolfi, 1987)

Os três tipos precisam considerar as situações simétricas, como um nó n ser filho direito de seu pai p e p ser filho direito do avô de n. Estas rotações devem ser aplicadas enquanto o nó n não se tornar a raiz da árvore. As Figuras 2 e 3 demonstram como as rotações são realizadas.



(a) n é filho esquerdo de p e p é filho esquerdo de a (b) n é filho esquerdo de p e p é filho direito de a

Figura 2: Rotações 1 e 2, respectivamente

Na sua implementação, o afunilamento será aplicado após a conclusão de cada operação. Por exemplo, após inserir um elemento numa ABB, rotações serão feitas até que o novo elemento se torne a raiz da árvore.

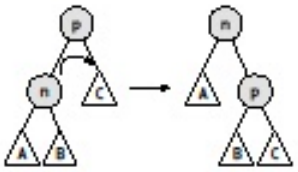


Figura 3: Rotação 3

Apesar de todas as operações serem iguais àquelas feitas em ABB, a remoção requer um pouco mais de atenção. No caso, o elemento que se tornará a nova raiz será o pai do nó removido. Em especial, se o nó k a ser removido tiver dois filhos, então a nova raiz será o pai do nó de maior chave da subárvore esquerda de k (também poderia ser o menor da subárvore direita de k). A Figura 4 exemplifica esta situação. Na árvore mais a esquerda da Figura 4 temos a estrutura atual da qual deseja-se remover a chave 7. A operação de remoção localiza a chave 7 e escolhe a chave 4 para a substituí-la, produzindo a segunda estrutura desta Figura. Por último é necessário realizar o afunilamento. Como o nó que originalmente guardava a chave 4 foi removido, o nó com chave 3 é aquele que deve ser rotacionado até atingir a raiz.

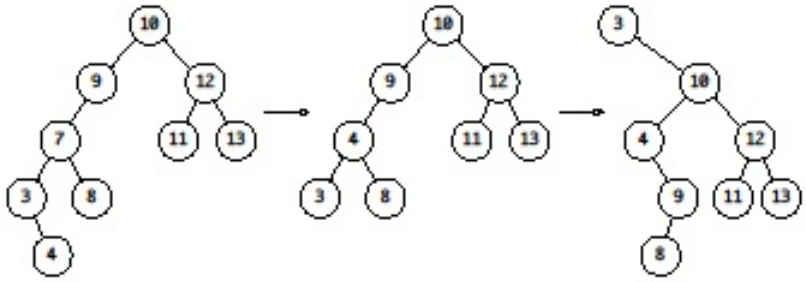


Figura 4: Exemplo de remoção da chave 7 seguido de afunilamento

Caso a operação de remoção seja utilizada com uma chave que não existe na árvore, o afunilamento deve ser feito no último nó acessado.

2 Entrada

A primeira linha da entrada contém dois números inteiros positivos, u e n. O número u representa o tamanho do conjunto universo e indica o conjunto a ser coberto: $\{1, 2, \ldots, u\}$. Em seguida há n linhas, cada uma representando um subconjunto de u. Cada uma destas linhas inicia com o caractere "s" depois apresenta um espaço em branco e, logo depois, uma sequência de números inteiros, separados por espaços, que compõem o subconjunto. Estes subconjuntos são enumerados a partir do número 1.

Exemplo:

```
10 5
s 1 2 3 8 9 10
s 1 2 3 4 5
s 4 5 7
s 5 6 7
s 6 7 8 9 10
```

3 Saída

Sua solução deve indicar (em um única linha) quais subconjuntos foram escolhidos pela abordagem *greedy* – seguindo a ordem das escolhas. A linha deve terminar com ":)" se for possível cobrir $\mathcal{U}$, caso contrário com ":(". O formato do exemplo adiante deve ser seguido. No caso, o conjunto 1 foi escolhido primeiro, depois o conjunto 4 e por último o conjunto 3. Se durante a operação do algoritmo *greedy* houver empate no tamanho de conjunto a selecionar, você deve escolher aquele que aparece por último na entrada.

Exemplo:

```
S1 U S4 U S3 :)
```

4 Dicas

1. Utilize uma lista para armazenar os tamanhos dos conjuntos e decidir pelo próximo a ser utilizado

5 Avaliação

A nota deste laboratório será dada por:

$$nota = \frac{CS\left\{8\left\lfloor\overbrace{\frac{\sum_{i=1}^{n}(teste_i == \text{correto})\,?\,1:0}{n}}^{SuSy}\right\rfloor\right\} + 2\,DQ}{S}$$

A variável DQ corresponde a documentação adequada (comentários no código fonte) e qualidade do código entregue. A variável booleana CS indica se o código em `C` está coerente com o enunciado da Seção 1. A variável contínua S está definida no intervalo $[1, 3]$; $S = 1$ corresponderá ao uso de afunilamento em todas as operações em ABB, enquanto que $S = 3$ indicará ausência do uso da mesma.

Seu trabalho deverá estar dividido da maneira seguinte:

`datastruct.h` : Contém os protótipos das estruturas de dados utilizadas;

`datastruct.c` : Contém toda a implementação das estruturas de dados;

`setcover.h` : Contém os protótipos das funções para realizar a cobertura de conjuntos *greedy*;

`setcover.c` : Contém toda a implementação do problema de cobertura de conjuntos *greedy*;

`main.c` : Trata a entrada e chama as funções definidas nos demais arquivos