Micael Andrade Dos Santos (201900051051)

1. Questão

`Ideia` Podemos adapatar o algoritmo de `LongaSubseqComum` para esse objetivo.

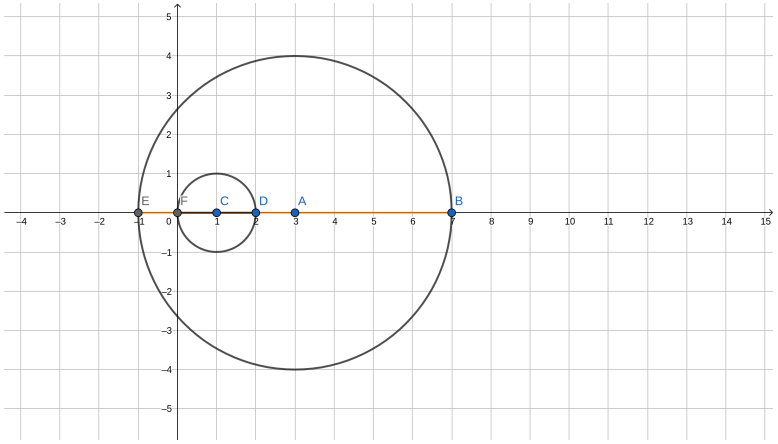
```
algoritmo LongaSubseqComum(X,m,Y,n,LCS)
{- entrada: sequências X e Y de tamanhos m e n;
saída: matriz LCS armazenando o tamanho da mais longa subsequência comum
de X e Y na posição (m,n) e indicadores de onde a solução foi obtida. -}
início
-- casos base, uma das sequências é vazia
para i = 0 até m faça {LCS[i,0].tam := 0; LCS[i,0].dir := ‘*’}
para j = 0 até m faça {LCS[0,j].tam := 0; LCS[0,j].dir := ‘*’}
-- caso geral
para i = 1 até m
  para j = 1 até n
    se X[i] = Y[j] então {LCS[i,j].tam := 1 + LCS[i-1,j-1].tam; LCS[i,j].dir := ‘D’}
    senão
      se LCS[i-1,j].tam >= LCS[i,j-1].tam então {LCS[i,j].tam := LCS[i-1,j].tam; LCS[i,j].dir := ‘A’
      senão{LCS[i,j].tam := LCS[i,j-1].tam; LCS[i,j].dir := ‘E’
fim
```

2. Questão

a. `Ideia`

Podemos transformar o problema dos discos em um problema de segmentos horizontais, visto que os pontos centrais são de coordenada y=0. Sendo assim, um disco está contido em outro disco se o segmento que representa o diâmetro de um determinado disco estiver contido em um dico maior. Podemos usar uma AVL para gerenciar os eventos para manter uma complexidade baixa.

Veja a imagem que gerei no geogebra:



Note que o diâmetro do disco menor está contido no diâmetro do disco maior. Logo o primeiro disco, está contido no segundo.

Temos os segnte discos MAIOR = (3,0) de raio 4, logo seu segmento é {(3-r, 0), (3+r, 0) = {(-1,0), (7,0)}, a mesma lógica vale para o círculo menor.

b. `Algoritmo`

```
algoritmo sweepDisc(P, n)
{--
ENTRADA: Conjunto de pontos de tamanho n representando os diâmetros de cada disco.
SAÍDA: A quantidade de segmentos de retas(diâmetros) contido em outro segmento de reta(diâmetro. Ou seja, a quantide
```

```
de discos contidos em outros.
--}
inicio
    countDisc := 0
    heapSort(P,n) -- Ordenando todos os pontos do meu conjunto para obter o efeito sweep-line
    A := criarAVL() -- AVL para manter os conjunto de segmentos abertos.
    para i=1 ate n faça
        se p[i] = 'ESQ' então -- Ponto esq vai para AVL de eventos
            A.insere(p[i])
        senão --Encontrou um fim de segmento DIR
            res = checaDentro((p[i].esq.x,p[i].dir.x), AVL) -- Passando os pontos extremos do segmento para verificar se está dentr
            se(res) {countDisc = countDisc+1} -- está dentro
            A.remove(segmento[i]) -- Removendo um segmento do conjunto de candidastas
    retorne countDisc

procedimento checaDentro(segmento, AVL):
{--
ENTRADA: Segmento de reta horizontal e uma AVL com os eventos abertos
SAÍDA:
--}
resposta := AVL.encontre(segmento) -- Verificando o segmento com os elentos em abertos.
se(segmento.xEsq >=  resposta.cordenada.x && segmento.Xdir <= segmentoxDir) -- a o segmento está dentro
    retorne True
fim
```

c. `Complexidade`

a. Temos que realizar uma ordenação na cordenada x dos segmentos. Estou usando o hepSort com complexidade `O(nlogn)` , além disso, estou usando uma árvore balanceada para gerenciar os eventos. Temos que as funções `encontre` `remove` `insere` também tem complexidade `O(nlogn)`.

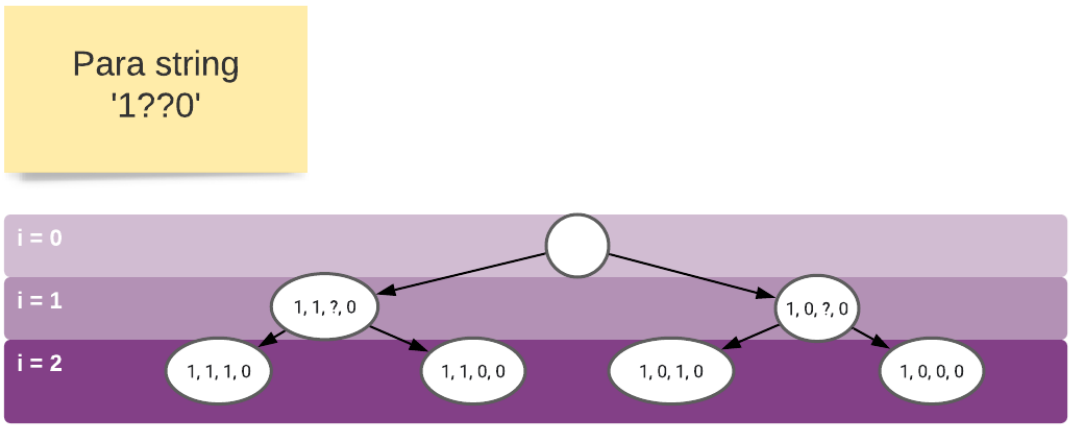
3. Questão

`Ideia`

Posso usar o algoritmo probabilístico LasVegas para construir uma senha e depois checar se essa senha contém os requisitos exigidos. Caso não, precisamos gerar outra, e para cada senha gerada devemos checar se é válido.

4. Questão

`Árvore`



`Algoritmo`

```
algoritmo backtracking(S, n, i)
iniciar
{--
ENTRADA: String S de tamanho n e inteiro i para realizar o controle do backtracking
SAÍDA: Saída uma string binária completa sem '?'.
```

```
--}
se(i == 2 && checkPrint(string, n)) faça -- 2 pois é 0 ou 1 (binário) estou verificando se a string pode ser impressa.
    escreva(string)
senão
    para i=1 até n faça
        se(string[i] = '?') faça
            string[i] = '1'
            K_backtracking(string, i+1)
            string[i] = '0'
            K_backtracking(string, i+1)
            string[i] = '?' -- reconstituindo a string
fim

procedimento checkPrint(S, n):
{--
ENTRADA: String S de tamanho n
SAÍDA: True caso S não tenha '?' False caso contrário
--}
inicio
    para i =1 até n faça
        se(S[i] == '?') então
            retorne False -- Não é imprimível
    return True -- Pode imprimir
fim
```