UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
INF05515 – COMPLEXIDADE DE ALGORITMOS – 2016/2

(Gunt3r_XML_F4t0r14l)

LEONARDO BISSANI
PAULO RICARDO DELWING

TRABALHO #1

Prof. Árton Pereira Dorneles

Porto Alegre, setembro de 2016

1. **Visão Geral da Solução**

Passo 1: Um laço *while (1)* e uma flag *flagporta* inicializada com zero como condição de parada para o mesmo. Dentro do laço o hacker percorre o mapa em busca da porta, salvando as coordenadas da mesma ao encontrá-la, tendo assim a distância até a porta.

Passo 2: O hacker então percorre o mapa procurando por um sensor que esteja a uma distância menor da porta e que ainda não tenha sido hackeado.

Passo 3: Se a porta de saída estiver mais distante do que algum sensor, o hacker irá se deslocar para o sensor mais próximo. Ele irá analisar o problema do sensor, e para resolve-lo irá usar a função *encontraPrimo*. Tendo em mãos uma chave encontrada na função em questão, o sensor é hackeado. O dispositivo hackeado é colocado então na lista dos já hackeados e retorna ao Passo 2. Se a porta de saída estiver mais próxima do que algum sensor, o hacker irá se deslocar até a porta e irá inspecionar o problema da porta, setando *flagporta* como 1, saindo assim do laço *while (1)*.

Passo 4: Um laço *while (2)* com as flags *flagp* e *flagq* inicializadas em zero como condição de parada. Ou seja, até encontrar as chaves p e q utilizadas para hackear a porta.

Passo 5: O hacker percorre o mapa em busca de sensores ou computadores que não tenham sido hackeados, novamente fazendo uso da flag *jafoihack* para conferir se o dispositivo se encontra na lista, e da flag *sensorcomp* para diferir computadores de sensores.

Passo 6: Se a flag *sensorcomp* for 1, então o dispositivo em questão é um sensor. O hacker é deslocado até ele e utiliza a função *encontraPrimo* para encontrar a chave e hackear o sensor. Se o dispositivo não está na lista dos já hackeados, ele é então adicionado e retorna para o Passo 5.

Passo 7: Se a flag *sensorcomp* for 2, então o dispositivo em questão é um computador. O hacker é deslocado até ele, e utiliza a função *encontraSeq* para encontrar a chave e hackear o computador. Se o dispositivo não está na lista dos já hackeados, então ele é adicionado.

Passo 8: Após hackear o computador e colocá-lo na lista dos já hackeados, o hacker percorre a lista, computador a computador, testando se o tamanho da chave do último adicionado a lista somado ao tamanho da chave de cada computador da mesma lista é igual ao tamanho da porta.

Passo 9: Caso não encontre resultado, então retorna ao Passo 5. Caso encontre resultado, o hacker utiliza a função *multiplica* para fazer o produto entre as duas chaves dos computadores e então testa se a mesma resolve o problema da porta.

Passo 10: Solucionando o problema da porta, *flagp* e *flagq* são setadas para 1 saindo do *while (2)*. Caso contrário, retorna ao Passo 5.

Passo 11: O hacker então escolhe a maior chave entre P e Q e a usa para hackear a porta.

2. Funções Utilizadas

a) *encontraPrimo*
Função que encontra o maior divisor primo de uma string numérica.
Recebe como parâmetros um inteiro *n*, representando o tamanho da string de entrada e *stringA*, a string da qual se deseja obter o maior primo.
Inicialmente a variável long *valor* irá guardar o valor de *stringA*. Dentro de um laço *for*, se o resto da divisão do *valor* pelo índice do laço for zero, então a variável *divisor* recebe o índice do laço, que será o maior divisor. O laço *while* interno irá decrementar o valor da string, para que quando este seja 1, saia do laço externo *for*.
Em sequência é executado um laço while com a condição de saída que o valor divisor (o maior número primo) seja maior do que zero. Neste laço, divisor é convertido em uma string, que ainda necessita de um segundo laço para ser colocado na ordem correta.
A função termina copiando a string encontrada como maior primo para a própria string *stringA* que foi passada como parâmetro da função.

b) encontraSeq
Função que encontra a maior sequência de números presente em duas strings.
Recebe como parâmetros um inteiro *n*, representando o tamanho de ambas as strings, e duas strings numéricas *stringA* e *stringB*.
Dentro de dois laços *for* encadeados, que irão percorrer as strings, são criadas variáveis tam e dois índices *k* e *l*. Enquanto o caractere de índice *k* da *stringA* for igual ao caractere de índice *l* da *stringB* e *k* e *l* forem menores que o tamanho das strings, *tam*, *k* e *l* serão incrementados. Se *tam* for maior do que a variável *maior*, indicando que há uma sequência maior presente, então *maior* receberá o valor de *tam* e *posi_maior* receberá o índice *i*, tendo o índice onde inicia a sequência numérica de maior tamanho.
Finalmente um laço *while* faz com que a variável *resposta* receba os caracteres da maior sequência de números presente nas strings.

c) intToString: como o próprio nome sugere, simplesmente converte este valor em uma string.

d) stringtovetor: função que recebe uma string e um vetor de inteiros e faz a conversão como o nome da mesma indica.

e) max: recebe dois inteiros e retorna o maior valor entre eles.

f) vetorToString: função que converte um vetor em uma string de caracteres.

g) multiplica: função que multiplica dois vetores, necessária na resolução do problema da porta.

3. Análise de Complexidade Pessimista

a) *encontraSeq*
Analisando primeiro o *while* interior dos dois laços *for* encadeados, o pior caso será quando stringA e stringB forem completamente iguais.
Assim, na primeira volta do laço ele será executado n vezes. Na segunda volta será n-1, e assim por diante, sempre decrementando n, ou seja, *n+(n-1)+(n-2)+ … + 1*.
Como temos dois laços *for* encadeados, e um laço *while* que pode ser escrito como um for de k=0 até n, então a complexidade desta função pode ser dada por:

$$\sum_{i=0}^{n} \sum_{j=0}^{n} \sum_{k=0}^{n} k$$

Desenvolvendo os somatórios, chegamos ao resultado O(**n^3**).

b) *multiplica*
Como essa função multiplica dois vetores de inteiros, são utilizados dois laços *for* encadeados, onde o pior caso é que os dois vetores são de tamanho *n*, onde n é um número tão grande quanto possível, dado por:

$$\sum_{i=0}^{n} \sum_{j=0}^{n} 1$$

Desenvolvendo os somatórios, chegamos ao resultado O(**n^2**).

c) Complexidade Geral do Hacker
Como o algoritmo do hacker essencialmente corresponde a executar a função encontraSeq n vezes, onde n é o número de computadores do mapa, que no pior dos casos será tão grande quanto possível. Ou seja, a complexidade pessimista será n * O(n^3) = O(**n^4**).