Projeto Prático de Desenvolvimento de Software

Sistema de Encriptação com Recurso à Manipulação de Inteiros e Strings

Usando Funções e Estruturas Dinâmicas de Dados em C

Linguagens de Programação 1
Rui Silva Moreira
rmoreira@ufp.edu.pt

João Viana
jviana@ufp.edu.pt

Algoritmos e Estruturas de Dados 1
José Torres
jtorres@ufp.edu.pt

Célio Carvalho
celio.carvalho@ufp.edu.pt

(versão 1.0)
Outubro de 2022
Universidade Fernando Pessoa
Faculdade de Ciência e Tecnologia

Descrição do problema

Em computação ubíqua (UbiComp) os sistemas possuem capacidades de processamento e memória mais limitadas do que os portáteis ou computadores pessoais. A disponibilidade de alimentação é também mais limitada, obrigando-nos a desenvolver aplicações e algoritmos que consumam menos energia, promovendo dessa forma a autonomia energética desses sistemas. A segurança em sistemas distribuídos, por exemplo, utiliza algoritmos de encriptação baseados em pares de chaves (cf. chave pública e respectiva chave privada) que exigem bastante poder computacional e recursos energéticos. Estes algoritmos limitam portanto a sua aplicação em sistemas UbiComp, que pela sua natureza mais limitada exigem algoritmos mais simples e computacionalmente mais económicos. Pretende-se, portanto, neste projecto, desenvolver uma alternativa aos algoritmos de encriptação fortes, baseada na utilização de números bipolares.

Um número bipolar é um inteiro positivo que contém uma ou várias ocorrências de apenas dois dígitos distintos, mas em que todas as repetições de um dos dígitos precedem, obrigatoriamente, todas as ocorrências do outro dígito. A tabela seguinte mostra vários exemplos de números bipolares:

Tabela 1: Exemplos de números bipolares

Números Bipolares					
8	9				
6	6	0	0	0	
1	2	2	2		
4	4	4	3	3	3

Pretende-se neste projecto que dada uma chave pública constituída por um número inteiro positivo, seja criada a correspondente chave privada. A chave privada deverá ser o mais pequeno número bipolar, múltiplo e maior do que a respectiva chave pública. Na Tabela 2 temos vários exemplos de chaves públicas e respectivas chaves privadas. Como se pode verificar, cada chave privada é constituída pelo menor número bipolar múltiplo da chave pública associada.

Tabela 2: Exemplos de chaves públicas (K^+) e respectivas chaves privadas (K^-) bipolares.

K^+					
8					
1	8				

K^-					
1	6				
3	6				

2	5	1			
2	0	1	3		

5	5	2	2		
5	5	5	5	8	8

O armazenamento das chaves privadas deve ser efectuado usando uma codificação *run-length*, ou seja, cada dígito deve ser precedido pela sua frequência na chave. A Tabela 3 mostra a codificação das chaves privadas previamente apresentadas.

Tabela 3: Exemplos de representações run-length de números bipolares.

K^-					
8	9				
6	6	0	0	0	
1	2	2	2		
4	4	4	3	3	3

run-length (K^-)					
1	8	1	9		
2	6	3	0		
1	1	3	2		
3	4	3	3		

Requisitos

De seguida resumem-se os requisitos a cumprir neste projecto. Todos os requisitos deverão ser implementados com base em funções e estruturas de dados usando **duas representações distintas**:

- **Números inteiros**: as chaves devem ser representadas através dos dígitos inteiros (0 a 9) constituintes das chaves;
- **Strings**: as chaves devem ser representadas através de strings contendo os caracteres dos dígitos (‘0’ a ‘9’) das chaves.

Pretende-se portanto que os alunos utilizem um conjunto de estruturas de dados dinâmicas (e.g. arrays dinâmicos, listas ligadas e combinações de ambos) e respectivas funções de manipulação de modo a obterem as **duas soluções** que cumpram os requisitos funcionais abaixo indicados (quer para a representação usando números inteiros, quer para a representação usando strings). As estruturas e funções associadas devem utilizar apontadores e estruturas dinâmicas para agregar e manipular os dados necessários. Deverão ser desenvolvidos também os algoritmos de processamento, pesquisa e gestão da informação, adequados às funcionalidades e requisitos elencados.

NB: NÃO deve ser desenvolvida uma interface interactiva com o utilizador, ou seja, NÃO devem desenvolver menus ou interface gráfica.

O programa a desenvolver deverá permitir para ambas as representações (cf. **números inteiros** e **strings**) as seguintes funcionalidades:

1. As estruturas de dados para guardar as chaves **públicas** e **privadas** devem basear-se em matrizes dinâmicas:
    a. No caso da representação com **inteiros** devem utilizar-se matrizes dinâmicas dos dígitos inteiros (i.e. array dinâmico de vários arrays dos dígitos das chaves);
    b. No caso da representação com **strings** devem utilizar-se matrizes dinâmicas de caracteres dos respectivos dígitos (i.e. array dinâmico de várias strings contendo os caracteres representativos dos dígitos das chaves);
2. Dada uma chave pública, calcular a chave privada com base em dois algoritmos diferentes à escolha, i.e., dada uma chave pública n:
    a. Gerar múltiplos de n até encontrar um nº bipolar (pode ser pouco eficiente);
    b. Gerar números bipolares sucessivos e verificar se são ou não múltiplos de n.
3. Dada uma chave privada, calcular a respectiva chave codificada utilizando uma representação *run-length*. A representação codificada (*run-length*) das chaves deve ser mantida/gerida numa matriz de inteiros e strings (tal como para as chaves públicas e privadas);
4. Inserir e remover uma ou mais chaves públicas e respectivas chaves privadas e codificadas (run-length) nas estruturas de dados. Inicialmente a inicialização deverá ser efectuada por geração aleatória das chaves públicas e cálculo das respectivas chaves privadas e codificadas. Mais tarde deve ser considerada também a leitura/escrita de chaves através de ficheiros;
5. Efectuar pesquisas de chaves, retornando as chaves privadas e codificadas para uma dada chave pública ou partes dessa chave, ou seja, cada pesquisa poderá utilizar subsequências/partes da chave pública na expressão de pesquisa;
6. Ordenar a lista de chaves públicas e respectivas chaves privadas e codificadas, por ordem crescente ou decrescente das chaves;
7. Listar as chaves públicas e respectivas chaves privadas e codificadas, na ordem em que se encontram inseridas ou noutra ordem indicada (e.g. crescente ou decrescente dos tamanho das chaves, i.e. número de dígitos);
8. Agregar numa estrutura de dados designada por KEY_HOLDER (porta-chaves) um conjunto de matrizes de chaves públicas, privadas e codificadas, tal como foram descritas anteriormente (3 matrizes para a versão com dígitos inteiros e 3 matrizes para a versão com strings). Considerar ainda a representação de uma lista ligada de vários porta-chaves, i.e., uma lista ligada em que cada nó contém os dados de 6 matrizes de chaves públicas, privadas e codificadas (versão inteiros e strings). Cada porta-chaves deverá ter uma ordem cronológica na sequência de nós da lista. Portanto, cada nó da lista deve ter duas datas: i) data da geração/criação do porta-chaves; ii) data da última actualização/modificação de chaves. Os nós da lista devem ser ordenados cronologicamente pelas datas de modificação dos porta-chaves. Devem ainda ser criadas várias funções para gerir a lista ligada de porta-chaves:
    a. Criar e inserir um porta chaves numa dada posição da sequência;
    b. Eliminar um porta-chaves que está numa dada posição;
    c. Pesquisar chaves apenas em determinados porta-chaves da lista, definidos numa expressão de pesquisa;

9. Será necessário representar também vários utilizadores, através do seu nome, email e conjunto de chaves do utilizador, i.e., cada utilizador poderá ter vários porta-chaves que deverão ser armazenados numa lista ligada de porta-chaves (como descrito anteriormente). Deverá existir também uma lista ligada de utilizadores. Deverão implementar funções de gestão da lista de utilizadores, nomeadamente:
    a. Inserir e remover utilizadores, à cabeça, cauda e ordenadamente pelo nome;
    b. Pesquisar utilizadores por nome ou parte do nome;
    c. Ordenar uma dada lista pelo nome dos utilizadores;
10. Utilizar ficheiros de texto para output e input de chaves públicas e privadas, i.e. escrever e ler informação de um ou mais porta-chaves em ficheiros de texto. As funções de pesquisa de chaves definidas anteriormente podem também ter como output ficheiros de texto;
11. Utilizar ficheiros binários para output e input de chaves públicas e privadas, i.e. escrever e ler informação de um ou mais porta-chaves em ficheiros binários.

Cotação dos Requisitos em AED1 e LP1

Req	**R1**	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**	**R9**	**R10**	**R11**
AED1	0,5	1,5	1	1,5	1,5	2	2	4	4	2	0
LP1	0,5	1,5	1	1,5	1,5	1,5	1	4	4	2	1,5

(NB: cotação 0-20)

Processo de Submissão

A aplicação final (código fonte) deve estar depurada de todos os erros de sintaxe e de acordo com os requisitos funcionais pedidos. Só serão considerados os projetos de software que não contenham erros de sintaxe e que implementam as funcionalidades pedidas total ou parcialmente.

Deve ser criado um repositório privado Git para conter todos os componentes do projeto. O conteúdo deve ser mantido atualizado. O repositório deve ser partilhado com os docentes através de autorização dada no próprio Git (partilha através dos IDs/E-mails dos docentes).

Adicionalmente, o projecto contendo a documentação (html ou pdf), juntamente com todo o código-fonte desenvolvido, deve ser submetida num ficheiro zip/rar (project.zip) na plataforma inforestudante (https://inforestudante.ufp.pt/).

Código Fonte e Documentação a entregar

As estruturas de dados e os algoritmos especificados devem ser implementados em linguagem C, juntamente com os comentários apropriados, inseridos no código fonte desenvolvido, de modo a que facilitem a compreensão do mesmo. Todas as funções devem

estar anotadas em formato doxygen (www.doxygen.nl) incluindo: uma breve explicação dos algoritmos implementados; uma menção ao desempenho dos algoritmos (quando aplicável) assim como dos testes efetuados/implementados.

As principais estruturas de dados e variáveis devem também estar anotadas neste formato. Deverão usar o software doxygen para gerar a documentação com base nos comentários.

Os alunos deverão manter no repositório git do projecto, um ficheiro de texto no formato doxygen (.dox), descrevendo explicitamente quais foram: i) os requisitos implementados, ii) parcialmente implementados e iii) não implementados. Devem mencionar sempre o número do requisito de acordo com a numeração utilizada neste documento de especificação:

- Funcionalidades implementadas: devem identificar todas as funções desenvolvidas para assegurar os requisitos funcionais solicitados.
- Funcionalidades parcialmente implementadas: devem identificar as funcionalidades parcialmente implementadas e apontar as dificuldades na sua conclusão.
- Funcionalidades não implementadas: devem identificar as funcionalidades não implementadas e as dificuldades que impediram o seu desenvolvimento.