universidade de aveiro deti departamento de electrónica, telecomunicações e informática

Catálogo criminal com Bloom Filter e MinHash

Grupo: 91322 – Isadora Loredo
92972 – Gonçalo Matos

Introdução

Neste projeto foi-nos proposto a criação de dois módulos para resolver o problema de acesso à base de dados de um catálogo criminal, de forma a determinar a existência de um nome na lista de criminosos e determinar criminosos parecidos segundo um conjunto de características selecionadas.

A base de dados deve conter os dados de criminosos com os seus crimes cometidos e suas características físicas.

Classes criadas

Para implementar a solução para o problema proposto, criámos algumas classes que nos auxiliaram a "esboçar" a realidade para a qual a aplicação que desenvolvemos foi criada. A principal define o criminoso (*Criminal*), nomeadamente o seu nome, código de identificação, data de nascimento, características físicas e crimes praticados. Os dois últimos atributos são por sua vez duas novas classes, respetivamente a *Traits*, que especifica a altura, o sexo, a nacionalidade e a cor da pela e a *Crime*, que estabelece um nome, uma categoria e uma data em que o mesmo foi praticado. Auxiliar a estas classes que desenvolvemosde raíz é a classe *Date*, que reutilizámos das aulas de POO, tendo feito algumas modificações, em particular com a criação dos métodos estáticos *numberDaysOfMonth()* e *randomDate()*.

Módulos

Bloom Filter

Implementação

A implementação deste módulo foi feita com a criação da classe *BloomFilter*, tendo como estrutura base um *array* de booleanos. Escolhemos trabalhar com este tipo de dados, pois é o que ocupa menos espaço em memória (1 bit apenas), cumprindo assim com a eficiência em memória na representação dos conjuntos.

Para além do *array* referido, são ainda atributos da classe o número de elementos adicionados (*nElements*) e o número máximo de elemento que se podem adicionar (*elementsInsert*), ambos inteiros e cujo objetivo é garantir que não são adicionados mais elementos do que os inicialmente previstos aquando da criação do objeto, evitando que ocorra *overflow* e/ou que a probabilidade de existirem falsos positivos se torne significativa.

São ainda criados dois *arrays* de inteiros gerados aleatoriamente entre 1 e um número primo, cada um com tantos elementos quanto o número de funções de dispersão especificadas na chamada ao construtor.

Como precisávamos de vários *hashcodes* para cada elemento introduzido no filtro, a solução que adotámos foi a de manipular o código gerado por uma única função de dispersão (neste caso a função de dispersão do objetivo a inserir – *hashCode()*), multiplicando e adicionando por dois números aleatórios, que são gerados no construtor, como descrito no parágrafo anterior. Como estes *arrays* de números aleatórios estão associados ao filtro, irão mapear o mesmo elemento sempre para os mesmos índices.

Funções

Entrando em detalhe nas funções criadas nesta classe, o construtor recebe como argumentos o número de elementos a inserir e o número de funções de dispersão que devem ser calculadas para cada elemento, determinando através da fórmula lecionada nas aulas teóricas o comprimento do *array* de booleanos, que inicia de seguida, tal como as variáveis de controlo e os *arrays* com números aleatórios.

O método *addElement()* recebe como argumento um tipo de dados genérico e começa por verificar se já foi adicionado o número máximo de elementos ao filtro, devolvendo *false* em caso afirmativo. Caso não se verifique, determina o *hashcode* do elemento com recurso à sua função de dispersão, valor que é manipulado posteriormente para cada índice dos *arrays* *hashMultiplier* e *hashIncrementer* (respetivamente multiplicado e somado), e finalmente mapeado para cada índice calculado com o resto da divisão inteira entre o *hashcode* manipulado e o comprimento do *array* de booleanos. Devolve *true* quando termina a execução.

O *isMember()* calcula os vários *hashcodes* do elemento passado como argumento, de forma idêntica à descrita na função anterior e verifica se todos os índices mapeados estão a 1 (valor *true*). Caso algum não esteja devolve *false*, caso contrário *true*.

As restantes funções são simples *getters*.

MinHash

Implementação

A implementação deste módulo foi feita com a criação da classe *MinHash*, tendo como princípio a comparação de similaridade entre um conjunto de características definidos, podendo esses conjuntos de características serem físicas ou registos criminais.

Para cada criminoso na base de dados são geradas 100 assinaturas (a função *makeMinHash()* é usada para popular as matrizes de assinaturas) a partir do conjunto de características escolhido para ser comparado, e armazenados nas matrizes de assinaturas *minHashTraits* para características físicas e registos criminais, é então uma matriz bidimensional com n números de criminosos e 100 colunas de assinaturas. Para as assinaturas são geradas permutações aleatórias através da função *hashCode()* de maneira a determinar o mínimo dos valores da função *hash* usada.

Funções

Esse módulo possui ainda duas funções as quais têm a finalidade de proporcionar a realização de comparações em busca de similaridades entre o conjunto de características. A função *getSimilar()* recebe um dos dois conjuntos de características possíveis, gera um vetor de 100 assinaturas para esse conjunto e compara com a matriz de assinaturas de criminosos correspondente ao conjunto de características selecionadas, leva em consideração a distância de Jaccard, calculada pela função *getDistance()*, que especifica o quanto similar se deseja que os dois conjuntos comparados sejam. A função retorna um conjunto de criminosos que possuam valor de similaridade maior ou igual à desejada.

Testes

Para implementar o nosso projeto, criámos as classes *Test* e *CriminalDataBase*. A primeira apenas para testar os módulos descritos acima e a segunda para implementar algumas funcionalidades que o utilizador pode testar de forma interativa.

A execução de ambas começa por ler do ficheiro *src/Data/traits.csv* um conjunto de características físicas e dos crimes que serão armazenadas em *arrays*, para posteriormente serem gerados os criminosos, em número introduzido pelo utilizador durante a execução no programa interativo ou 100 no caso do *Test*. Os dados dos criminosos são gerados aleatóriamente com equiprobabilidade, com exceção do sexo, da altura e da data de nascimento, que são baseados nos dados estatísticos para Portugal,retirados do Pordata e referenciados no final deste relatório.

Depois de gerados os criminosos, são criados os filtros de Bloom para armazenar a informação relativa aos criminosos em si, ao seu nome completo e aos seus nomes individualmente (um nome completo é dividido em cada um dos seus nomes constituintes e cada um mapeado individualmente no filtro). Antes desta operação é também pedido ao utilizador para introduzir o número de funções de dispersão (no *Test* são consideradas 3) que deve ser utilizado em cada filtro (o único filtro que não segue esta regra é o que mapeia

os nomes individuais, que como vai mapear mais nomes vai ter o dobro das funções de dispersão introduzidas e o quíntuplo do comprimento – pois no pior caso os criminosos gerados aleatoriamente têm cinco nomes - para evitar falsos positivos).

Aqui as execuções começam a seguir diferentes rumos. No *CriminalDataBase*, depois de "inicializado" o programa, é apresentado o menu ao utilizador, que pode explorar as várias funcionalidades desenvolvidas, nomeadamente a verificação se um criminoso com um determinado nome (singular ou completo) consta na base de dados, com recurso aos filtros de Bloom criados para esse efeito e também a busca de criminosos similares a suspeitos de crimes.

É gerada uma lista de suspeitos, com quantidade determinada pelo utilizador, dos quais queremos encontrar uma lista de criminosos similares a cada suspeito na base de dados gerada possui o mesmo conjunto de características físicas e do crime cometido. As características físicas dos suspeitos assim como a dos criminosos são enquadras padrões como faixa etária e média de altura, uma vez que não é possível saber com exatidão as características de um suspeito, e dessa forma potencializar a quantidade de correspondências nas buscas por similaridades físicas. É realizada a leitura do ficheiro *src/Data/suspects.txt* com um conjunto de características físicas e dos crimes que serão armazenadas em *arrays*, para posteriormente serem gerados os suspeitos.

A classe *MinHash* é chamada para a busca de similaridades. Se encontradas similaridades retorna quantos e quais os criminosos que correspondem às características procuradas, para cada suspeito.

No *Test,* são adicionados ao filtro de Bloom os criminosos nos índices mútiplos de 10 (0,10,20,…,90) e verifica se os que têm índices múltiplos de 5 (0,5,10,15,…,90,95) estão no filtro, fazendo o *output* do resultado do teste para cada um dos elementos. São também gerados 5 suspeitos com deferentes conjuntos de característica e realizadas as buscas de criminosos por similaridade, de forma estática.