Trabalho Prático
Laboratórios de Informática

Grupo 6

Guilherme Viveiros e Gonçalo Faria

Departamento de Informática
Universidade do Minho

Contents

1 Motivação

No âmbito da Unidade Curricular Laboratórios de Informática III, foi proposto pelo corpo docente a análise dos dados gerada pelo website Stack Overflow.

A análise foi realizada por forma a responder às 11 interrogações propostas pelos mesmos.

2 Carregamento dos dados:

A informação do Stack Overflow encontra-se dividida em comunidades, cada uma delas com um tema específico. A informação relativa a cada comunidade encontra-se no formato xml e subdividida por 8 ficheiros (i.e., Votes, Tags, Users, PostLinks, Posts, PostHistory, Comments e Badges). No entanto, consideramos que apenas os ficheiros Votes,Tags, Users e Posts são necessários para responder corretamente a todas as interrogações.

Na leitura dos ficheiros xml optamos pela biblioteca recomendada pelos professores, Libxml2[1] ,**versão 2.9.7**. O tipo de dados abstrato, Community, que encapsula os dados carregados, é composto por 3 Hash Tables, uma dedicada a guardar a informação dos utilizadores, outra das publicações e uma outra a fazer correspondência entre as tags [2] e o identificador destas. Adicionalmente para facilitar as travessias que envolvem um intervalo de datas foi também incluída uma Lista de publicações que as guardará apenas por referência evitando assim ocupar espaço adicional.

A selecção das estruturas teve em conta minimizar os tempos de acesso e travessia por forma a reduzir o tempo de resposta às interrogações. Evidentemente, para isso foi necessário sacrificar tempo no pré-processamento destas mesmas.

[1] http://www.xmlsoft.org/

[2] Tags - Termo geralmente usado como sinónimo de etiqueta ou palavra-chave.

3 Estruturas principais

A Hash Table é uma ferramenta ideal para efetuar a correspondência entre chaves e valores sempre que a ordem dos elementos não é relevante. A Hash Table contém um tempo amortizado de consulta e remoção constante, quando complementada com uma função hash[3] apropriada à distribuição das chaves.

Uma função hash deve assegurar que o conjunto imagem tem uma distribuição o mais próximo do uniforme possível, para que todos os dados sejam perfeitamente distribuídos pelo array associativo que a suporta, tanto que, idealmente, esta função seria injectiva, no entanto na maioria das aplicações isto não é possível.

Encontrar a hash ideal é o maior desafio no uso de uma hashtable, pois tem de haver um equilíbrio entre o tempo de processamento e qualidade dos mapeamentos.

Como para fazer a correspondência das tags e armazenar o conjunto de utilizadores não existe uma necessidade de ordem foi claro que Hash Table era a estrutura ideal.

A decisão de usar uma Hash Table para o conjunto de publicações exigiu um grau superior de deliberação. Dado que uma porção significativa das interrogações requerem operar num subconjunto das publicações cuja data de criação se encontra num indicado intervalo, a noção de ordem cronológica é crucial para rapidamente operar desta forma neste conjunto.

Como abdicar dos acessos constantes às publicações seria terrivelmente dispendioso tivemos que investir mais memória nesta tarefa. Mantivemos as publicações armazenadas na Hashtable e complementamos o tipo abstrato de dados com uma nova estrutura, uma Lista[4], cuja única função seria facilitar estas travessias intervalares.

Após a fase inicial do carregamento dos dados não há novas adições na Community, logo não é necessário uma estrutura dedicada a assegurar adições ordenadas em tempo reduzido. Esta informação sobre a natureza do problema levou-nos a decidir implementar esta Lista na forma de um Array [5], assim sendo no final do carregamento de toda a informação para as Hash tables esta será criada e ordenada.

Tendo introduzida a ordem na Lista é então possível usar todo um conjunto de algoritmos eficientes para encontrar o segmento deste array que corresponde ao intervalo indicado.

Para a implementação em concreto da Hash Table foi usada a biblioteca Glib[6], **versão 2.56.1** ,principalmente desenvolvida pela GNOME. A Glib é uma excelente biblioteca que os professores permitiram usar como ferramenta na elaboração do projeto.

[3] Função hash é um algoritmo que mapeia dados de comprimento variável para dados de comprimento fixo.

[4] Lista é uma coleção de elementos sequencial que pode conter elementos repetidos.

[5] Array é um lista de elementos do mesmo tipo guardados conjuntamente em memória.

[6] http://www.gtk.org

A Hash Table da Glib permite-nos usar qualquer função hash que pretendamos, no entanto como precisamos necessariamente de aceder ao valores com o identificador, um inteiro de 64 bit (long), tanto no conjunto dos utilizadores como no das publicações e como este valor é único para cada elemento no seu respectivo conjunto, o nosso trabalho fica significativamente facilitado, pois temos unicamente de converter este identificador num inteiro de 32-bit.

4 Estruturas auxiliares:

Em todas as interrogações que envolvem encontrar o top N de uma métrica em questão usamos uma fila prioritária limitada.

Inicialmente decidimos implementar a fila prioritária limitada na forma de uma Heap binária introduzida em [1] e posteriormente melhora significativamente em [2].

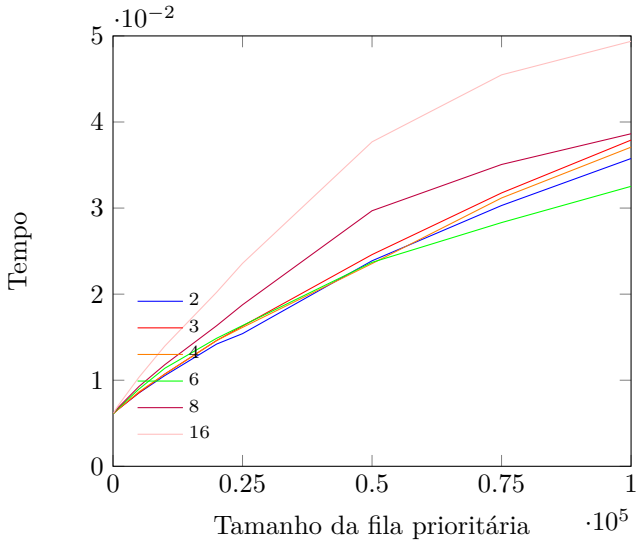
Uma Heap binária é uma estrutura implícita que mantêm a propriedade invariante de ter os filhos menores ou maiores que o pai (dependendo sé é uma max-Heap ou min-Heap), cuja a análise da complexidade e comportamento foi alvo de estudo na U.C. de Algoritmos e Complexidade.

Criar	$\Theta(N)$
Adicionar	$O(log_{\mathrm{k}} N)$
Remover	$O(k * log_{\mathrm{k}} N))$
Mínimo(1)	$\Theta(N)$

(1) Encontrar máximo caso seja uma max-Heap ou minimo no caso de min-Heap
(2) Sendo n o número de elementos na Heap e k o número de ramos por nodo (fator de ramificação).
(3) k=2 para a Heap binária.

No entanto, durante o cálculo dos tempos teóricos, apercebemos-nos que se escolhe-se-mos aumentar o número de filhos de cada nodo para 3 ou 4 teríamos a garantia de obter um tempo teórico inferior.

Tendo em conta esta descoberta decidimos implementar uma variação de Heap binária, denominada d-Heap, introduzida em [3], onde a propriedade invariante é a mesma que a da Heap, no entanto suporta um número arbitrário de ramos em cada nodo. Na tentativa de escolher este fator de ramificação realizamos os testes em baixo.



(1)Os testes foram realizado na interrogação #2 os valores em cada entrada são o tempo médio nas 1000 vezes que a interrogação #2 foi repetida nos dados android, o eixo das abscissas corresponde à variação do argumento N. (2)Todos os testes foram realizado num portátil com : 8,00 GB (RAM) ,Intel(R) Core ,i7-4720HQ e CPU 2.60 GHZ. (3)Todos os testes usaram o parâmetro -Ofast no gcc.

Acabamos por decidir usar uma d-Heap com fator de ramificação de 6. O tipo de dados de cada elemento do array onde a d-Heap foi implementada é void* para que esta seja genérica e assim fácil de reutilizar nas diferentes interrugações.

Como a propriedade invariante da Heap nada diz quanto à relação que os nodos do mesmo nível têm, não há um mecanismo imediato para manter esta limitada.

A nossa solução foi a de inverter a ordem da Heap,tal que se quisermos saber os N maiores, usamos uma min-Heap e caso queiramos saber os menores uma max-Heap.

Isto possibilitará verificar se o elemento a adicionar é maior ou menor que o que se encontra no topo, no caso afirmativo substituímo-lo e verificamos a propriedade invariante.

Para esclarecer melhor esta ideia, tomemos a Interrogação #2 como exemplo. Nesta Interrogação temos de devolver os N utilizadores mais ativos e para isso usamos a fila prioritária limitada. Ao iniciar, é criada uma min-Heap com N elementos do conjunto de utilizadores. Dado que o número de elementos do conjunto será muito maior que N, os restantes elementos serão iterativamente testados para adesão na Heap. Sendo que é uma min-Heap o utilizador menos ativo, chamemo-lo de x, pode ser consultado em tempo constante logo se o utilizador em que estamos a iterar for maior que x , trocamos x por este novo e verificamos nele a propriedade invariante. No final teremos a Heap repleta com os N utilizadores mais ativos onde os menores destes encontrar-se-ão no topo.

5 Publicação

A estrutura que criamos para representar as publicações é essencialmente composta por nome, identificador, identificador do utilizador que a criou, data de criação , número de comentários, lista com as tags usadas, tipo de publicação, classificação, número votos, no caso de ser resposta o identificador da pergunta correspondente, no caso de ser pergunta o número de respostas e também uma lista ligada com os identificadores das respetivas respostas.

6 Utilizador

O módulo que criamos para representar cada uma das utilizações foi composta por identificador , nome , número de perguntas efetuadas , número de respostas efetuados , uma pequena biografia, reputação e uma Hash Table que regista as intervenções do utilizador nas diferentes publicações. Este último campo, foi especialmente construído para suportar intersecções de forma eficiente.

7 Gestão de memória:

Para assegurar que não há perdas de memória nas diferentes partes do projeto usamos o software Valgrind[7], que indica os blocos de memória que foram reservados pela aplicação mas que não contêm nenhuma variável com o endereço desta.

[7] http://valgrind.org/

8 Interrogacões:

8.1 Interrogação 1:

Devido à escolha de estruturas, esta interrogação foi particularmente simples. A função que a realiza, acede ao conjunto das publicações, obtém a publicação com o identificador discriminado nos argumentos desta e de seguida usando os métodos que possibilitam comunicar com com os atributos privados, acede ao identificador do utilizador que a criou.

Tendo acesso tanto à publicação em questão como ao utilizador que a criou a função unicamente retorna o nome de ambos.

8.2 Interrogação 2:

A função que implementa a interrogação faz uma travessia no conjunto de utilizadores, por forma a preencher uma fila prioritária limitada.

São adicionados N dos primeiros elementos a um Array e logo que este esteja totalmente preenchido é convertido para uma fila prioritária limitada de tamanho N. A prioridade desta fila é o número de publicações e os restantes elementos a ser percorridos são iterativamente testados para adesão nesta, até que todos os utilizadores tenham sido percorridos e a fila prioritária contenha o top N de utilizadores mais ativos.

8.3 Interrogação 3:

Nesta Interrogação usufruímos da ordenação cronológica que a Lista em Community possui. Dado que temos de contar o número de perguntas e respostas num determinado intervalo de tempo, usamos pesquisa binária, descrita em [4], para identificar os dois índices na lista que contém o indicado intervalo de datas. Após identificado o segmento a ser percorrido, é efetuada uma travessia neste, aplicando uma função a cada elemento. Esta função, verifica se o elemento é uma pergunta ou uma resposta e regista a contagem no resultado que devolve.

8.4 Interrogação 4:

Sendo que existe em Community uma Hash Table que efetua o mapeamento entre tag e o seu identificador e como cada publicação contém uma listagem do identificador das suas tags, teremos unicamente de verificar em cada publicação se a tag recebida está listada nesta. Dado que apenas a informação no intervalo recebido é relevante na travessia, esta é efetuada na Lista em Community com o mecanismo que consta na Interrogação 3.

8.5 Interrogação 5:

O utilizador é obtido, mais uma vez, consultando o conjunto de Utilizadores. Dado que existe um campo, no tipo de dados que representa o utilizador, dedi-

cado a guarda o nome e outro a biografia temos unicamente que que chamar os métodos apropriados.

Como para aceder as 10 últimas publicações temos de percorrer o conjunto que contém todas as intervenções do utilizador é construída com os mecanismos elucidados anteriormente uma fila prioritária limitada de tamanho 10 onde a prioridade é a mais recente data de criação.

8.6 Interrogação 6:

A função que realiza esta interrogação é extremamente semelhante à interrogação 2, dado que temos que listar o Top N de respostas com mais votos num determinado intervalo de tempo.

Em quase todas as interrogações em que é usada uma fila prioritária existe uma função comum a todas elas, dedicada a fazer a travessia e criação da respectiva fila, tendo apenas que ser proporcionada a função de comparação dos elementos.

Tal como na interrogação 3 é encontrado o segmento da Lista de Community e de seguida usado o método de criação da fila prioritária, onde a prioridade é o número de votos.

8.7 Interrogação 7:

Nesta interrogação, mais uma vez, o procedimento é o mesmo, no entanto os elementos adicionados são unicamente as perguntas e a prioridade na fila prioritária é o número de respostas.

8.8 Interrogação 8:

Como a diferença entre maiúsculas e minúsculas não pode alterar os resultados, há a necessidade de pré-processar tanto a palavra dada como o título da publicação. O pré-processamento, para além de converter as letras maiúsculas para minúsculas coloca um espaço antes e após as cadeias de caracteres , tanto no título como na palavra dada. Esta última operação é para prevenir que um título que contenha a palavra com um afixo de ser considerado como uma possível solução.

Após o pré-processamento, dá-se a travessia das publicações cuja data se encontra no intervalo indicado usando a Lista de Community e o mecanismo de seleção do segmento introduzido na interrogação 3. No percorrer de cada publicação é usada a função strstr[8] para identificar a ocorrência da palavra no título. .

[8]strstr - Retorna um ponteiro que indica a primeira ocorrência de uma dada string numa outra ou NULL caso este não exista.

8.9 Interrogação 9:

Esta Interrogação foi especialmente desafiante, tanto que requereu vários campos dedicados no módulo Utilizador.

Nesta interrogação foi necessário devolver as últimas N perguntas em que dois utilizadores intervêm conjuntamente.

Para isso mantivemos no módulo que o representa uma hashTable cujas chaves são os identificadores das perguntas em que o utilizador interviu. Caso esse utilizador tenha sido apenas o criador da pergunta, então o valor correspondente a essa chave será NULL, caso contrário é o identificador resposta. Usando esta estrutura que cada um dos utilizadores contém, primeiro calculamos a interseção das chaves, obtida iterando pelos elementos da menor e verificado se o elemento que está a ser iterado está contido na maior.

Tendo a interseção destes conjuntos ter-se-á unicamente que criar uma fila prioritária de tamanho N sobre ela e devolver na forma de uma lista ordenada.

8.10 Interrogação 10:

Como todas publicações que são perguntas contêm uma lista com o identificador de todas as suas respostas. Para resolver esta interrogação é necessário percorrer esta lista acedendo à publicação a que corresponde o identificador e obtendo a resposta cujo valor da expressão de cálculo é máximo.

8.11 Interrogação 11:

Primeiro, na função que implementa esta interrogação, são obtidos, usando uma fila prioritária de tamanho N, o top N de utilizadores com maior reputação. Isto é alcançado percorrendo o conjunto de utilizadores e usando os mecanismos de criação de fila prioritária referidos anteriormente. Após encontrados, estes utilizadores são todos eles adicionados a uma nova hash table e de seguida é criado um histograma com todas as tags que estes usam.

O histograma, também implementado numa HashTable, é preenchido percorrendo o intervalo recebido como argumento da interrogação na Lista em Community. Assim que este esteja preenchido é também ele percorrido procedendo ao processo de criação de uma fila prioritária onde a prioridade é valor de cada tag no histograma.

No final ,teremos estas tags ordenadas decrescentemente pelo número de ocorrências e teremos unicamente de a devolver como resposta à interrogação.

9 Modularização:

Todas as componentes do nosso trabalho, foram criadas em ficheiros fonte separados e com atributos privados, para que fosse impossível aceder a estes sem usar os métodos diretamente associados a cada uma das estruturas de dados criadas.

A legibilidade e a flexibilidade do nosso projeto permitiu testar todas os diferentes componentes dos módulos criados sem que fosse necessário alterar as centenas de linhas externas que usufruem destes.

10 Esquema de tempo:

Android	4.1919491 s
Ubuntu	25.34053 s

(1)Todos os testes foram realizado num portátil com : 8,00 GB (RAM) ,Intel(R) Core ,i7-4720HQ e CPU 2.60 GHZ. (2)Todos os testes usaram o parâmetro -Ofast no gcc.

11 Convenções para entradas inválidas:

Dado que não existem identificadores com o valor 0, caso o número de elementos que são resposta seja inferior ao número enviado nos argumentos da interrogação, ao enviar os resultados, serão primeiro enviadas as soluções encontradas na ordem que é pedida e os restantes conterão o valor 0.

12 Referências:

[1] - J.W. J. Williams, Algorithm 232: Heapsort, Communications of the ACM 7 (1964), 347-348.

[2] - R.W. Floyd, Algorithm 245: Treesort 3, Communications of the ACM 7 (1964), 701

[3] - D. B. Johnson, Priority queues with update and finding minimum spanning trees, Information Processing Letters 4 (1975), 53-57.

[4] - Willams, Jr., Louis F. , A modification to the half-interval search (binary search) method , Proceedings of the 14th ACM Southeast Conference (1975) , 95-101.