Problema da Conectividade

Análise da implementação de algoritmos em linguagem C

Laboratórios de Algoritmos e Estruturas de Dados

Grupo X - 2ªfeira
Beatriz Ferreira 78794; Henrique Nogueira 78927
Professor ——-

21 de Setembro de 2016

Resumo: Perante o problema da Conectividade, foi feita uma análise da eficácia de resolução para 4 Algoritmos distintos para diferentes níveis de complexidade do problema.

1 Introdução e Motivação

Dado um número **N** de objectos, identificados separadamente por um inteiro, e uma sequência de pares desses inteiros (**p-q**), representando uma ligação entre dois objectos, o **problema da Conectividade** consiste em saber se, dado um par de objectos arbitrários, existe uma ligação directa/indirecta entre eles.

Este problema tem interesse em ser estudado pois objectos abstractos podem ser traduzidos em objectos reais, permitindo a resolução de problemas do mundo real, como por exemplo:

- numa rede de computadores, saber se um está ligado a outro
- saber se dois pontos de contacto de um circuito estão ligados entre si
- saber se existe uma ligação directa ou indirecta entre duas pessoas numa rede social

É portanto vital ser capaz de implementar o Algoritmo certo para cada situação pois a eficácia de resolução do problema pode ser melhorada em várias ordens de grandeza permitindo tomar maior partido do poder de computação de uma dada máquina.

2 Implementação dos Algoritmos

Em laboratório, pretendeu-se analisar a eficácia de diferentes Algoritmos para diferentes níveis de complexidade diferente. Para este efeito foram implementados 4 Algoritmos diferentes.

Visto que para a resolução deste problema não é necessário saber o caminho percorrido que liga dois objectos mas apenas se estes estão de facto ligados, basta-nos distribuir os objectos por diferentes conjuntos e ir aglomerando á medida que é recebida informação acerca das ligações entre si, perdendo no processo informação relativa á ordem de ligação. Podemos portanto dividir este processo em 2 fases distintas, a Procura (**Find**), onde se identifica se dois objectos estão contidos num mesmo conjunto, e a União (**Union**) onde se faz a junção de dois conjuntos distintos. O fluxograma representado na **figura 1** exprime precisamente este processo:

1. é lido um par de objectos vindo da sequência de pares
2. a procura obtém o identificador de conjunto de cada objecto
3. o identificador exprime a coexistência dos objectos no mesmo conjunto, permitindo saber se **p** e **q** se encontram ligados
4. **caso não se encontrem ligados** segue-se o processo de União onde se quer igualar os identificadores de cada objecto de ambos os conjuntos

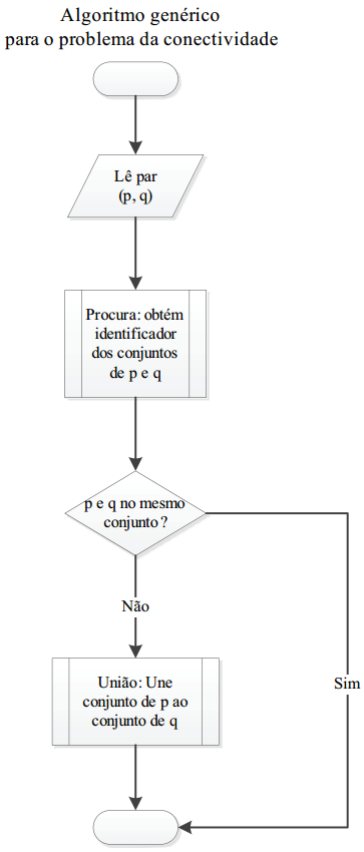

Figure 1: Fluxograma genérico de um algoritmo de resolução do problema da conectividade (por aglomeração de conjuntos)

Para cada um dos algoritmos que se seguem é usada a mesma estrutura de dados, uma tabela de identificadores **id** cujo índice representa o objecto.

2.1 Algoritmo de Procura rápida

O primeiro algoritmo a analisar é o algoritmo de procura rápida, que como o nome indica tem uma fase de procura mais rápida, necessitando apenas de uma operação para

concretizar a verificação. A ideia é que o identificador do conjunto está imediatamente contido na identificador da tabela, isto é, se **p** e **q** estão contidos num mesmo conjunto então **id[p]=id[q]**.

Em contrapartida de uma operação de procura rápida, a sua união é mais lenta pois implica que para cada união de conjuntos é necessário percorrer a tabela inteira e redefenir o identificador de todos os objectos que pertenciam a um desses conjuntos.

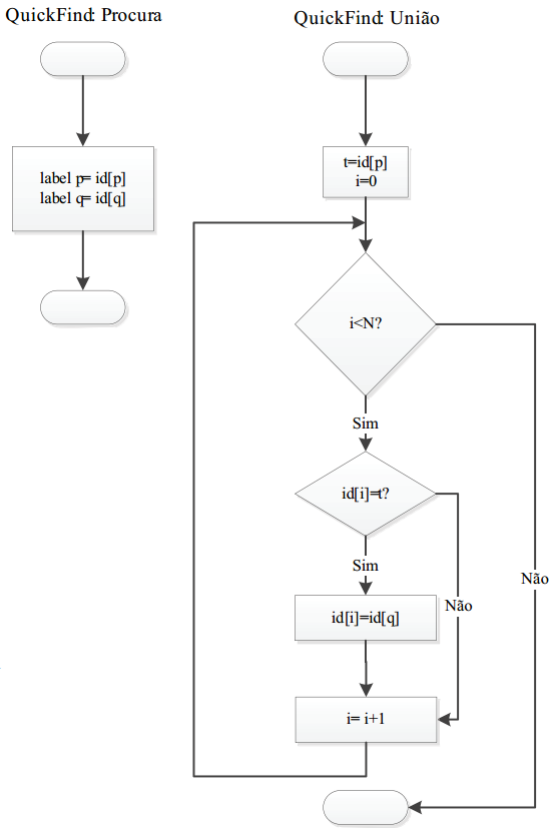

Figure 2: Operações de procura e união de algoritmo Quick-Find. A operação de procura é executada num só passo inquirindo se id[q]=id[p]. A operação de união já exige a leitura e actualização de todos os identificadores antigos da tabela.

Para cada um dos algoritmos que se seguem é usada a mesma estrutura de dados, uma tabela de identificadores **id** cujo índice representa o objecto.

Podemos dizer com certeza que o número de operações de procura é sempre igual a **M**, número de pares, pois para cada novo par a procura é feita num só inquérito (**id[q]==id[p]**)

Quanto á união é mais incerto pois este dependerá do número de ligações, **L**, pois caso o resultado da procura seja negativa não haverá união por fazer. Podemos no entanto garantir que pelo menos **(N+1)L** operações de união são feitas pois para cada ligação a tabela é lida N vezes e escrita pelo menos 1 vez.

No entanto, para **M>>N** com **N>>1**, temos na maior parte dos casos **L** na mesma ordem de grandeza de **N**, pois o número máximo de ligações é sempre (**N-1**), e portanto o número de operações expectável é aproximadamente $\mathbf{N}^2$.

2.2 Algoritmo de União Rápida

Este algoritmo por sua vez tem um processo de união rápida em contrapartida com um processo de procura mais lenta. A ideia desta vez está em vez em estabelecer uma hierarquia de ponteiros que convirjam numa raíz do conjunto, isto é, se a procura indicar uma nova ligação entre **p** e **q** faz-se **id[p]=q** estabelecendo a ligação ao apontar arbritáriamente **p** á raíz de **q**. Desta forma cria-se uma estrutura de conjuntos em forma de árvore cujo cada elemento aponta para o próximo até chegar á raíz que por sua vez aponta para si mesma. Percebe-se portanto que o número de operações de procura é superior pois o identificador de conjunto está "escondido" a cada objecto, sendo necessário seguir o caminho de ponteiros para o encontrar.

O processo de união é sempre feito numa operação dada uma nova ligação, pelo que no total é feita em **L** operações.

Por sua vez, o número de operações de procura é incerto pois o número de procuras por par dependerá da estrutura de árvore dos conjuntos em causa. No entanto para valores de **N** pequenos iremos ter uma estrutura de árvores mais favorável a uma procura mais rápida pois para cada procura temos que seguir no máximo uma cadeia de **N** objectos, pelo que teóricamente será mais eficaz nesses casos.

2.3 Algoritmo de União Rápida equilibrada

Este algoritmo é equivalente ao algoritmo anterior de uma maneira geral, excepto que em vez de ser feita arbitráriamente a estrutura em árvore dos conjuntos, escolhe-se sempre apontar o objecto com o menor conjunto em árvore da qual ele é raíz, permitindo uma melhor distribuição dos nodos da árvore e por sua vez diminuindo a quantidade de ponteiros a seguir para chegar á raíz do conjunto aglomerado. No entanto, a memória gasta é duplicada já que é necessário acrescentar uma tabela com os tamanhos dos conjuntos correspondentes bem como acrescentar ás operações de união os passos relativos á actualização desta nova tabela.

É possível provar que para cada processo de procura cada objecto tem de seguir no máximo **Log(N)** ponteiros até chegar á raíz, pelo que as operações de procura não devem exceder **MLog(N)** operações.

Já as operações de união são no máximo 3 por par: leitura e comparação de tamanhos, associação de ponteiro e por fim update da tabela de tamanhos.

2.4 •Algoritmo de União Rápida equilibrada comprimida

Semelhante ao Algoritmo de União Rápida equilibrada, este por sua vez acrescenta mais ao processo de união fazendo ligar todos os objectos que procedem o ponteiro inicial directamente á raíz do seu objecto par.

Torna-se óbvio que para **M** e **N** grandes este processo díminui em bastante, mesmo relativamente ao algoritmo imediatamente anterior, o tempo necessário á procura de cada par, no entanto para **N** e **M** muito pequenos poderá representar trabalho desnecessário.

3 Resultados experimentais

4 Análise de resultados

5 Conclusão

6 Referências

- Algorithms in C, Robert Sedgewick
- Aulas teóricas, Prof. Carlos Bispo; Instituto Superior Técnico, Departamento de Engenharia Electrotécnica e de Computadores