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, cada um identificado por um inteiro, e uma sequência com pares de objectos (**p-q**), representando uma ligação entre si, o **problema da Conectividade** consiste em saber se, dado um par de objectos arbitrários, existe uma ligação entre ambos, directa ou indirectamente.

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 saber que Algoritmo implementar pois, dependendo do grau de complexidade do problema, a eficácia de resolução 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,foi feita uma análise da 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, representando objectos ligados, 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

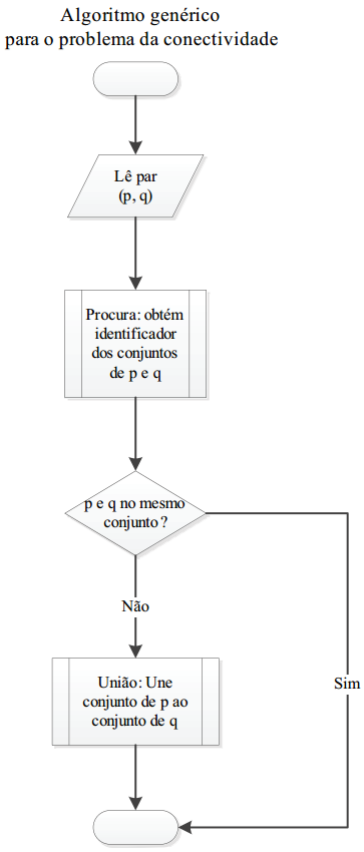

Figure 1: Fluxograma genérico de um algoritmo de resolução do problema da conectividade (por aglomeração de conjuntos)

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 QF - 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.

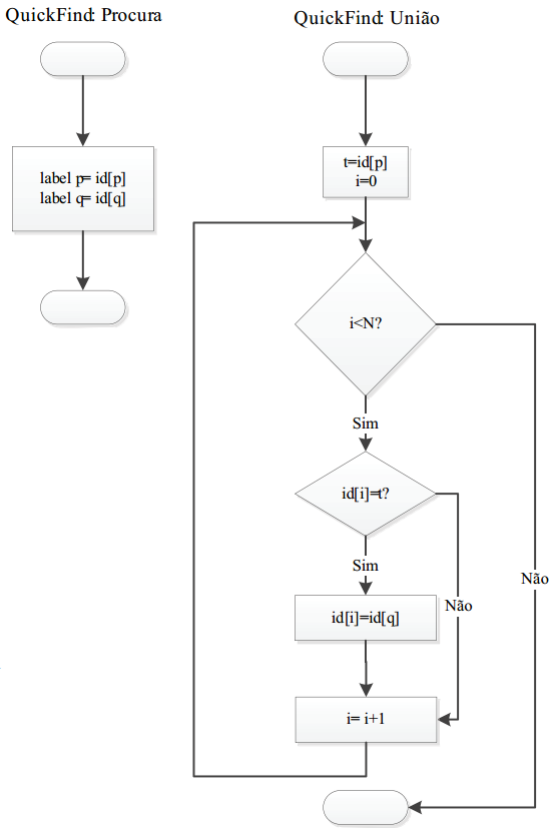

Figure 2: Operações de procura e união de algoritmo Quick-Find. A operação de procura é executada num só passo inquirindo se id[q]=id[p]. A operação de união já exige a leitura e actualização de todos os identificadores antigos da tabela.

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 positiva 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 no máximo (**N-1**), e portanto o número de operações expectável é aproximadamente $\mathbf{N}^2$. Finalmente este algoritmo também tem a propriedade de aumentar em velocidade assim que a união entre todos os pontos fica estabelecida pois o processo de união nunca mais é chamado.

2.2 QU - 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. Em oposição ao Algoritmo de União rápida, este Algoritmo continua a demorar o mesmo tempo de execução mesmo após estabelecida a ligação entre os pontos.

2.3 WQU - 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[1] 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 CWQU - 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 considerávelmente o número de operações de procura necessárias, no entanto para **N** e **M** muito pequenos poderá representar trabalho desnecessário.

[1] Ref. Algorithms in C, Robert Sedgewick

3 Resultados experimentais

Para colocar os Algoritmos á prova foi necessário contabilizar o número de operações de procura e união efectuadas por cada um para problemas de diferente complexidade. Utilizou-se para esse efeito o programa fornecido, juntamente com os dados, ao qual apenas foi necessário implementar os contadores necessários.

f_cnt e **u_cnt** são respectivamente os contadores de operações de procura e de união aos quais se vai incrementar sucessivamente o valor para uma operação de leitura ou actualização da tabela.

A tabela 1 apresenta o resultado obtido correndo cada Algoritmo com 9 configurações de pares, **N**, **M** e **L**.

				Quick Find		**Quick Union**	
Dados	Nós	Pares	Ligações	Find	Union	Find	Union
10.txt	10	5	4	5	47	8	5
100.txt	100	200	49	200	5,340	2,910	200
1000.txt	1,000	2,000	499	2,000	531,892	185,503	2,000
10000.txt	10,000	20,000	4,998	20,000	53,232,431	18,479,087	20,000
a.txt	1,000	6,206	999	6,206	1,120,579	1,211,723	6,206
b.txt	2,500	20,236	2,499	20,236	7,013,153	10,603,943	20,236
c.txt	5,000	41,913	4,999	41,913	28,182,683	45,839,698	41,913
d.txt	10,000	83,857	9,999	83,857	112,960,455	186,599,079	83,857
e.txt	25,000	309,802	24,999	309,802	704,627,920	1,782,620,002	309,802

				Weighted Quick Union		**Compressed WQU**	
Dados	Nós	Pares	Ligações	Find	Union	Find	Union
10.txt	10	5	4	18	12	15	13
100.txt	100	200	49	1,119	147	595	195
1000.txt	1,000	2,000	499	12,074	1,497	6,620	2,002
10000.txt	10,000	20,000	4,998	127,290	14,994	66,693	20,253
a.txt	1,000	6,206	999	40,067	2,997	22,058	4,009
b.txt	2,500	20,236	2,499	131,737	7,497	71,116	10,087
c.txt	5,000	41,913	4,999	275,402	14,997	148,030	20,198
d.txt	10,000	83,857	9,999	565,155	29,997	298,584	40,616
e.txt	25,000	309,802	24,999	2,139,355	74,997	1,116,831	101,506

Table 1: Tabelas de resultados experimentais

4 Análise de resultados

Numa análise superficial, é vísivel que os algoritmos **QF** e **QU**, embora a um nível de eficácia comparável para graus de complexidade muito baixos, assim que se aumenta a dificuldade o número de operações explode. Isto indica desde logo que ambos os algoritmos reajem não linearmente com **N**. Em oposição, **WQU** e **CWQU** apresentam um comportamento mais linear, sendo observável que por cada ordem de grandeza de **N** o número de operações aumenta também aproximadamente em uma ordem de grandeza, indicando comportamento aproximadamente linear com **N**.

4.1 QF - análise de resultados

Fazendo um gráfico de dispersão logarítmico para o número de operações de união (**U**) podemos investigar se há uma relação exponêncial com **N**.

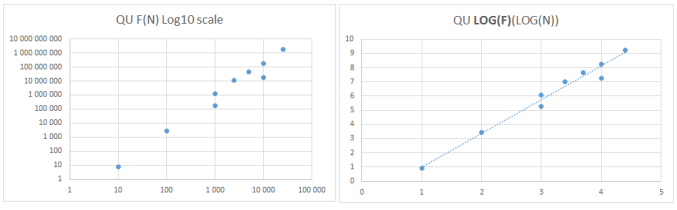

Figure 3: QF - Á esquerda: plot de U(N) em escala logarítmica; á direita: regressão linear sobre **Ln(U)**(Ln(N))

Observando os gráficos da figura 3, com excepção de dois dos pontos experimentais obtidos, podemos concluir que a regressão linear foi correctamente aplicadamente aplicada ao gráfico devolvendo um declive de 2. Temos portanto uma verificação do comportamento previsto de que $\mathbf{U} \simeq \mathbf{N}^2$.

Como expectável, o número de operações de procura é igual a **M**, não sendo necessária fazer uma análise gráfica sobre **F**.

4.2 QU - análise de resultados

Repetindo o mesmo processo da análise imediatamente anterior...

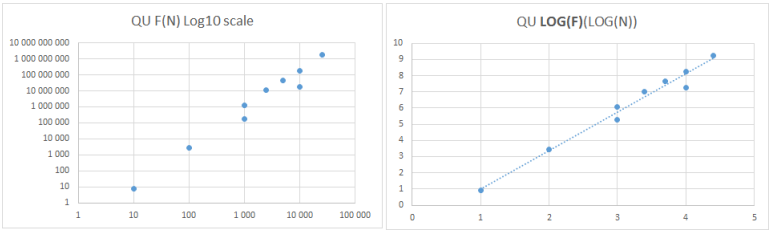

Figure 4: QU - Á esquerda: plot de F(N) em escala logarítmica; á direita: regressão linear sobre **Ln(F)**(Ln(N))

Paralelamente ao primeiro algoritmo, a regressão linear realizada sobre o plot logaritmico de **F(N)** devolve um declive de 2, podendo também ser feita a conclusão $\mathbf{F(N)} = \mathbf{N}^2$.

O número de operações de união está exactamente de acordo com o previsto.

Por fim á que acrescentar que o número de operações deste algoritmo é superior ao algoritmo **QF** para valores de **N** muito grandes, e mais especificamente **M** >> **N**. Uma razão mais óbvia para que tal aconteça é pois, mesmo quando as ligações entre todos os objectos já estão estabelecidas, que se pode confirmar ser o caso devido a **L=N-1**, este algoritmo continua a ter o mesmo número de operações para cada novo par enquanto **QF** apenas verifica a já existente ligação.

4.3 WQU e CWQU - análise de resultados

Por fim, ambos os algoritmos **WQU** e **CWQU** provaram ser verdadeiramente eficazes para todos os graus de complexidade testados. Fazendo os respectivos gráficos de dispersão obtivemos as imagens da figuras 5 e 6.

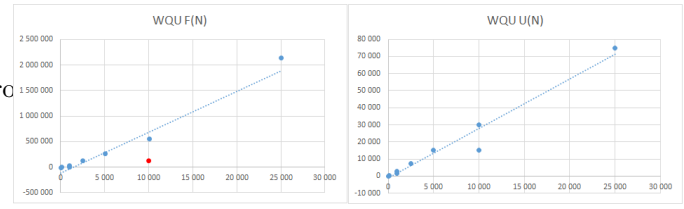

Figure 5: WQU - Á esquerda: regressão linear do plot de F(N) ; á direita: regressão linear do plot U(N)

Embora a amostra de resultados não seja suficiente, a regressão linear destes gráficos de dispersão parece que foi bem sucedida a menos de um dos pontos.

Para o algoritmo **WQU** obtivemos o valor de declive representando $\mathbf{F(N)} \simeq \mathbf{60N}$ e $\mathbf{U(N)} \simeq \mathbf{3N}$ e para **CWQU** os

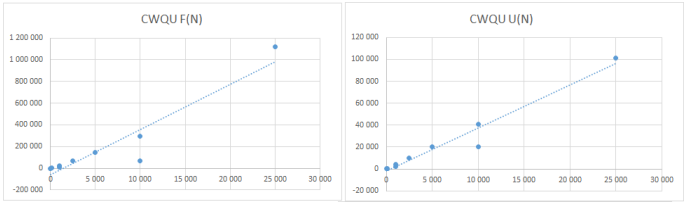

Figure 6: CWQU - Á esquerda: regressão linear do plot de F(N); á direita: regressão linear do plot F(N)

respectivos valores **F(N)≃40N** e **U(N)≃4N**, verificando-se assim uma maior eficácia na procura por parte de **CWQU** ao custo de uma ligeiro aumento no tempo de união e compressão.

5 Análise crítica e Conclusão

Após a análise dos resultados experimentais podemos concluir que estes estavam de acordo com os valores esperados. O algoritmo **CWQU** deve ser portanto implementado para uma aplicação usual do problema da conectividade, tendo no entanto em conta que para valores baixos na ordem de **N=10** este fica atrás de **QU** (28 operações totais contra apenas 13), devido a uma maior complexidade nas operações de compressão e equilibrio associadas. No entanto, com alguma surpresa, verifica-se que se pode esperar uma resposta que aumenta aproximadamente linearmente com o aumento de **N** tanto para **WQU** como **CWQU**.

Quanto aos algoritmos **QU** e **QF**, podemos esperar na maior parte dos casos um tempo quadrático em ordem ao número de objectos.

No entanto, é importante acrescentar que o comportamento destes algoritmos pode ser imprevisivel, como se pode inferir dos dados **1000.txt** e **10000.txt** que estavam claramente fora das linhas obtidas por regressão linear. Portanto na aplicação destes algoritmos deve-se ter tido em conta a aleatoriadade ou não da forma como estão distribuidas as ligações.

6 Extra - Impressão dos conjuntos na função de Algoritmo Quick-Find

Por fim, no ponto 4 do enunciado é pedida a implementação de código que imprima dos diferentes conjuntos obtidos. Para tal o código segue as seguintes instruções:

1. O número de conjuntos diferentes é dado por **N-L=set_cnt**
2. Usar codificação: **id[i]==-1⇒ já foi impresso**
3. Encontrar o primeiro **id[i]!=-1** e guardar o conjunto a ser impresso numa variável (**t**
4. Imprimir todos os objectos com **id[i]==t**
5. Repetir o processo **set_cnt** vezes até imprimir todos os objectos

7 Referências

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