TÉCNICO LISBOA

ALGORITMIA E DESEMPENHO
EM REDES DE COMPUTADORES

Conectividade em Digrafos

Gonçalo Ribeiro, 73294

Ricardo Amendoeira, 73373

Docente: Prof. João Luís Sobrinho

10 de Dezembro de 2014

1 Caminhos Disjuntos entre Dois Nós

Na primeira parte do projecto é-nos pedido um algoritmo que dado um digrafo, um nó de origem e um de destino determine qual o número mínimo de ligações que é preciso quebrar para que o nó de origem deixe de conseguir chegar ao nó de destino.

Podem existir inúmeros caminhos entre o nó de origem e o de destino mas sabemos que se esses caminhos não forem disjuntos em termos de arestas ao quebrarmos uma aresta comum quebramos todos caminhos. Parece portanto óbvio que este problema é equivalente a encontrar o número de caminhos disjuntos que permitem chegar do nó de origem ao nó de destino. Há que notar também que como estamos na presença de um digrafo não é recíproco ir de um nó A para B ou de B para A.

O problema pode ser traduzido para um problema de fluxos máximos em que cada ligação tem capacidade unitária e em que o fluxo permitido é ou unitário ou nulo. Para resolver este problema podemos usar o algoritmo de Ford–Fulkerson ou o de Edmonds–Karp. Optámos por nos basear no segundo algoritmo, que faz as procuras com BFS. O pseudo-código da nossa implementação pode ser visto no Algoritmo 1.

```
count_disjoint(graph, src, dest):
  nr_disjoint = -1
  while there is a path from src to dest:
    nr_disjoint = nr_disjoint + 1
    path = BFS(graph, src, dest)
    if a path was found:
      reverse links in path

  return nr_disjoint
```

Algoritmo 1: algoritmo para contagem de caminhos disjuntos num digrafo

Sendo n o número de nós e m o número de arestas, o algoritmo de Edmonds–Karp é $O(nm^2)$ e a complexidade do Algoritmo 1 é a mesma. Verificámos que é possível melhorar este resultado usando o algoritmo de Goldberg que é $O(n^2m)$. No entanto considerámos que os grafos a considerar são relativamente esparsos e que portanto a melhoria não seria significativa. Segundo [1] existe ainda o algoritmo "relabel-to-front" que é $O(n^3)$.

2 Imunidade a Quebras

Nesta secção é pedido que se calculem para cada $k \in \mathbb{N}$, a fracção de pares de nós que são separados quebrando apenas k ligações. A primeira parte deste projecto calcula o número de caminhos disjuntos entre um par de nós, o que equivale ao número de ligações que têm de ser quebrados para separar o par (um por cada caminho disjunto).

Assim, o método utilizado para resolver a 2ª parte foi correr o Algoritmo 1 para cada par de nós (em ambos os sentidos, uma vez que o grafo é direccionado), incrementando o número de pares separados com k ligações cada vez que se descobre um par com k caminhos disjuntos entre si, obtendo no fim o numerador da fracção para cada valor de k. O denominador da fracção é simplesmente o número de pares origem–destino do grafo. O pseudo-código deste processo pode ser visto no Algoritmo 2

```
Redundancy(graph):
  int separated_by[k]
  for each src in graph:
    for each dest in graph:
      if src ≠ dest:
        k = count_disjoint(graph, src, dest)
        if k ≠ 0:
         separated_by[k] = separated_by[k] + 1

  return separeted_by
```

Algoritmo 2: algoritmo que determina a k-conectividade de um digrafo

O Algoritmo 2 usa o Algoritmo 1 (Edmonds–Karp), de complexidade $O(nm^2)$, $n(n-1)$ vezes, levando a uma complexidade de $O(n^3m^2)$ para o Algoritmo 2.

3 k-conectividade de um Grafo

No último problema deste trabalho é pedido um algoritmo que calcule a k-conectividade de um digrafo em termos de arestas (*k-edge-connectivity)*, ou seja dado um digrafo fortemente conexo qual o número mínimo de ligações que é preciso remover para que o grafo deixe de ser fortemente conexo. Para além disso queremos encontrar um conjunto de ligações que uma vez quebradas fazem o grafo deixar de ser conexo.

Da definição anterior segue que para um digrafo ser k-conexo todos os pares de nós têm que ser pelo menos k-conexos, ou seja é necessário que existam pelo menos k caminhos disjuntos de *cada nó* para cada um dos outros. Esta condição poderia ser relaxada no caso de estarmos na presença de um grafo não direccionado i.e. num grafo não direccionado basta que *um nó* tenha k caminhos disjuntos para cada um dos outros. Esta diferença deve-se ao facto de que num grafo não direccionado os caminhos disjuntos de A para B são os mesmos que de B para A, mas num digrafo estes caminhos não são coincidentes e podem ser em número diferente.

Tendo em conta que o objecto de estudo deste trabalho são digrafos, um algoritmo possível para determinar a k-conectividade é veririficar qual o mímino de caminhos disjuntos que existe entre cada par de nós. O par que tiver entre si o menor número de caminhos disjuntos impõe esse número como o valor de k. O Algoritmo 3 faz uso deste raciocínio.

O Algoritmo 3 usa o Algoritmo 1 $n(n-1)$ vezes, pelo que a complexidade do primeiro será $O(n^3m^2)$.

Para descobrir um exemplo de k ligações que quebradas façam a rede não ser conexa usámos o raciocínio de que quando o Algoritmo 1 termina as ligações que fazem parte de caminhos disjuntos estarão invertidas em comparação com o grafo original. Consequentemente se do conjunto de ligações invertidas

```
k_connectivity(graph):
  min_k = ∞
  for each src in graph:
    for each dest in graph:
      if src ≠ dest:
        k = count_disjoint(graph, src, dest)
        if k == 0:
          return not strongly connected
        if k < min_k:
          min_k = k

  return min_k
```

Algoritmo 3: algoritmo que determina a k-conectividade de um digrafo

escolhermos k que pertençam a caminhos disjuntos obtemos um conjunto de ligações que se quebradas tornam o grafo desconexo. Uma forma simples de escolher ligações de caminhos disjuntos é escolher k ligações invertidas de entre as que saem da origem ou de entre as que entram no destino.

4 Visualizador de Grafos

Visto que para o trabalho anterior criámos um visualizador de grafos, fizemos pequenas adaptações a esse visualizador de forma a podermos ver digrafos criados para o trabalho actual.

5 Conclusão

Este 3º mini-projecto tem grande utilidade para ajudar a identificar as zonas mais vulneráveis de uma rede (no caso de perder certas ligações) e, consequentemente, a aumentar a sua robustez. Como falado nas aulas, um problema de $min - cut$ é muito próximo de um problema de $max - flow$, pelo que este mini-projecto pode ser facilmente alterado para resolver esse tipo de problemas. Foi notada a falta da existência um algoritmo conhecido para grafos direccionados que resolva o problema da $edge - connectivity$ de forma mais eficiente que usando $n(n-1)$ vezes um algoritmo de $min - cut/max - flow$ de um nó para outro como o de $Edmonds - -Karp$ usado neste mini-projecto. Sendo ainda uma área de estudo, é possível que uma solução assinptóticamente mais eficiente seja apresentada no futuro.

Referências

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms. 3rd ed.* Cambridge, MA: MIT Press, 2009.

[2] Prof. João Luís Sobrinho. Slides das Aulas de Algoritmia e Desempenho em Redes de Computadores.