Análise e Síntese de Algoritmos

2013/2014

Projeto

Parte 2

Prefácio

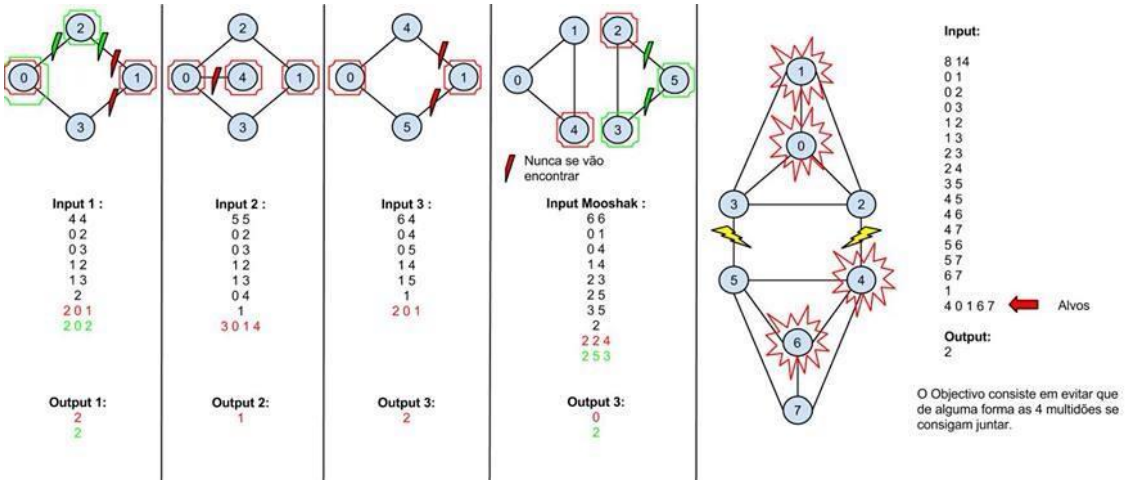
Introdução

O objectivo deste Relatório consiste em explicar a nossa implementação do problema para solucionar a quebra de comunicação entre pontos críticos e consequentemente o seu isolamento através da *Teoria de Grafos*.

Problema

De uma forma breve, o nosso problema consiste em modelar um controlo de multidões, onde existem pontos, onde as pessoas se podem concentrar e ligações entre esses pontos. O objectivo consiste em saber qual o numero mínimo de ligações a barrar bastando isolar um e um só ponto crítico dos restantes.

Exemplo: passamos a explicar o Problema através dos seguintes exemplos do Enunciado.



Solução

A solução implementada por nós passaria por retirar partido de um Algoritmo que modele o problema em termos de Fluxos Máximos, sendo que daqui se calcularia o *Corte Mínimo* a partir do *Teorema do Fluxo Máximo – Corte Mínimo*. Neste caso teríamos uma oferta de Algoritmos tais como:

- *Edmonds-Karp*;
- *Ford-Fulkerson*;
- *Relable-to-Front*;
- *Push-Relable*;

No nosso caso optamos por utilizar o *Algoritmo de Edmonds-Karp*, sendo este uma variante do *Algoritmo Ford-Fulkerson* com a particularidade de ter um final garantido e com um tempo de execução independente do valor do Fluxo Máximo, sendo que também implementa uma Busca em Largura (BFS) com a finalidade de encontrar o Caminho de Aumento mais curto.

Uma das razões que nos levou a implementar o *Algoritmo de Ford-Fulkerson* foi o facto de sendo o *Grafo Não Dirigido*, usar um algoritmo que usasse um baixo número de *Arcos do Grafo* como este o faz.

Algoritmo de Edmonds-Karp – Descrição

Como dissemos em cima, o *Algoritmo de Edmonds-Karp* é uma implementação do *Algoritmo de Ford-Fulkeron* para uma resolução do problema de *Fluxo Máximo* numa Rede de Fluxos. É distinguido pela característica de ter um caminho de aumento mais curto sendo usado a cada interação com o Grafo, garantindo assim findo o calculo. Na nossa implementação o Caminho mais Curto é encontrado através de uma Busca em Largura (BFS).

Uma das grandes diferenças entre o *Algoritmo de Edmonds-Karp* e o *Algoritmo de Ford-Fulkerson* é o facto de o primeiro ter uma ordem de busca quando encontra que o caminho de aumento de fluxo definido, sendo que o caminho encontrado deve ser o caminho mais curto com capacidade disponível.

Note-se que neste Algoritmo, o comprimento do Caminho de Aumento encontrado nunca diminui, assim como também, os caminhos encontrados são os mais curtos possíveis.

O Fluxo encontrado é igual à capacidade que cruza o menor corte possível no Grafo separando a fonte e o seu consumo.

Algoritmo de Edmonds-Karp – Complexidade

Em termos de Complexidade o *Algoritmo de Edmonds-Karp* implementado uma busca em Largura (BFS) cuja a complexidade em tempo de execução é de **O(|V| + |E|)**, que completado pelo *Algoritmo de Edmonds-Karp* obtemos uma complexidade assimptótica de **O(VE^2)**.

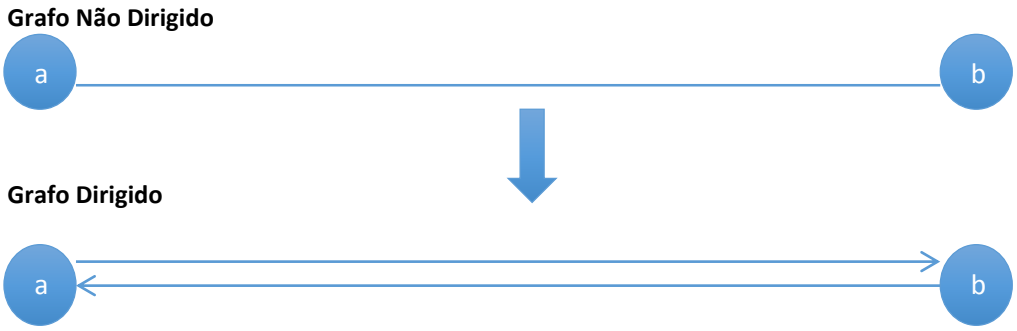
Pelo motivo de utilizarmos uma Matriz de Adjacências e sendo o grafo um Grafo Esparso, não obtivemos aprovação à totalidade dos Testes com uma reprovação dos mesmos por *Memory Limit Exceeded*. Algo que não foi resolvido.

Implementação

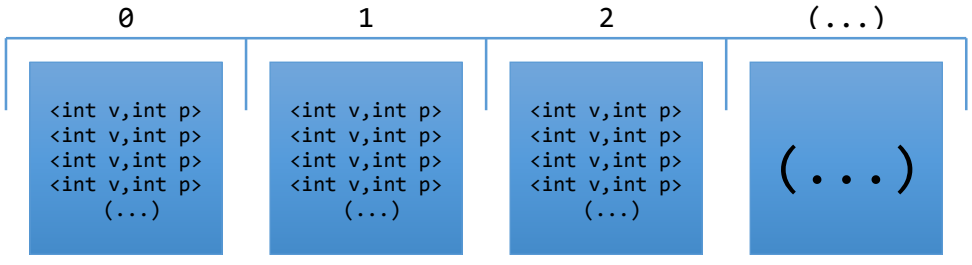
Para a implementação deste projeto utilizamos a Linguagem de Programação C++ pois tem um excelente rácio entre simplicidade de Programação e Eficiência de execução não retirando grandes recursos da Máquina.

No que toca à nossa implementação do *Algoritmo de Edmonds-Karp*, passamos por algumas questões no decorrer da programação do problema. Inicialmente começamos por programar uma solução baseada no *Algoritmo de Ford-Fulkerson*, no entanto chegamos à conclusão, que para além de mais eficiente também seria bastante mais útil uma Busca em Largura (BFS) consecutivamente iria originar com que a nossa implementação final fosse o *Algoritmo de Edmonds-Karp*.

O primeiro problema que aqui tivemos foi a necessidade de conversão de um Grafo em um Grafo Dirigido e Pesado, sendo tal necessário, pois teria que ser aplicado ao Algoritmo de Fluxos Máximos sempre um Grafo Dirigido e Pesado. Assim sendo, uma forma simples e intuitiva de resolver o problema foi ligar dois vértices unidirecionais com sentidos opostos um do outro:



Em termos computacionais, optamos por implementar um vector de vectores, tal como fizemos para a Primeira Parte do Projeto, simplesmente desta vez foi uma simples implementação de vectores de vectores de inteiros, onde cada elemento é um par de inteiros, os quais, o primeiro elemento do par representa o segundo vértice '*v*' que define o Arco e o segundo elemento do par indica o peso '*p*' desse mesmo Arco, a ideia essencial da nossa Estrutura de Dados está em seguida exemplificada:



Não esquecendo que o peso dos Arcos é definido com o valor um `Capacities[from][to] = 1` e `Capacities[to][from] = 1`.

Implementando uma a Busca em Largura (BFS) a qual retorna Verdadeiro no caso em que existe um dado caminho proveniente da origem 's' (source) que vai até ao final 't' (trace) do Grafo Residual, sendo que, também preenche a `parentsList[]`, criando Vértices visitados e marca os restantes como não visitados. Após isto, cria uma lista onde guarda a origem dos Vértices e marca-os consoante a sua origem tenha já sido, ou não, visitada, atualizamos os pais do futuro Nó para que este passe a ser o Nó atual. Por fim, Se chegamos ao fundo da BFS partindo da origem, então retornamos Verdadeiro, caso contrario retornamos Falso.

Também foi implementada uma DFS com a funcionalidade básica de procurar todos os vértices alcançáveis a partir de s. A função marca como visitado no caso em que o índice do Array dos vértices visitados é alcançável a partir de s. Os valores iniciais do nosso Array de vértices visitados devem ser falsos. Também podemos sempre usar a BFS para encontrar os vértices alcançados, mas era algo não tão útil nesta situação e não tão direto.

Para o Cálculo do *Corte Mínimo* aplicamos o *Algoritmo de Ford-Fulkerson* que no fundo nos irá dar o Fluxo Máximo e o Corte Mínimo num Grafo, alocados num vector `vector<AEK> minCut`. Após isto, cria um Grafo Residual, como termo de comparação, e preenche-o com as capacidades dadas no Grafo original como se tratassem de capacidades residuais dentro do próprio Grafo. Por fim, procura pela capacidade residual mínima dos vértices pelo caminho preenchido pela BFS, por outras palavras, procura o Fluxo Máximo através dos caminhos já percorridos e já encontrados, sendo que, quando o Fluxo Máximo já foi atingido, procura agora o vértice alcançável '*s*', alcançando '*s*', contabiliza agora o número de Cortes Mínimos possíveis e assim que contar um, esse é o devolvido.

Conclusão

Em suma, provavelmente esta implementação poderá não ter sido a mais correta, pois originou *Memory Limit Exceeded* cuja origem provem de utilizarmos uma *Matriz de Adjacências*. E assim sendo não obtivemos a aprovação total aos testes submetidos, no entanto sabendo a causa, seria de fácil resolução este problema. Contudo achamos que o *Algoritmo de Edmonds-Karp* seria o mais correto nesta situação.