Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Projeto e Análise de Algoritmos
Trabalho Prático 1

Orkato

Alberto Hideki Ueda

Matrícula: 2014765817

Belo Horizonte
9 de outubro de 2014

1 Introdução

Dado um conjunto de amigos, deseja-se descobrir qual é a melhor configuração possível entre todas as possibilidades de unir todos os amigos em uma única rede. Para cada amizade entre duas pessoas, são fornecidas duas informações: o *nível* da amizade e a *distância* física entre elas. Define-se a qualidade de uma configuração a divisão da soma de todos os valores de amizade desta configuração pela soma de todas as distâncias. Em outras palavras, deseja-se a configuração de amizades de maior qualidade possível.

2 Modelagem

Seja cada indivíduo um vértice e cada amizade entre duas pessoas uma aresta ligando dois vértices, podemos definir o conjunto de amigos original como um grafo G contendo todas as amizades possíveis de acordo com o conjunto fornecido. Deste modo, cada configuração de amizades pode ser vista como um possível *subgrafo* de G.

Considerando a restrição de que uma configuração viável deve alcançar todas as pessoas do conjunto, é natural considerar como subgrafos válidos apenas os que conectam todos os vértices do grafo original G. Assim, caso não seja possível conectar todos os amigos, o valor da função qualidade para todo subgrafo possível não é definido e consequentemente o problema não tem solução. Neste caso, seguindo a especificação do trabalho, o algoritmo implementado devolve um valor pré-estabelecido.

A grande peculiaridade deste problema é que o *peso* de uma aresta não pode ser simplesmente a) o nível de amizade, b) a distância, ou c) a divisão da amizade pela distância, pois nestes casos o algoritmo descartará informações relevantes. Por exemplo, se for utilizada a divisão amizade/distância como peso da aresta, amizades como (2000/1000) e (2/1) serão consideradas equivalentes, o que não é verdadeiro, dada a função de qualidade.

Porém, se nível de amizade e distância forem considerados separadamente, a única solução conhecida para maximizar a qualidade seria utilizar algoritmos de força-bruta, testando configuração por configuração. Sendo E o número de arestas no grafo original G, o número de subgrafos possíveis é da ordem de 2^E e o algoritmo teria complexidade exponencial.

Felizmente, podemos adotar o custo de cada aresta i como $f_i - (d_i * r)$, onde f e d são os valores do nível da amizade e da distância, respectivamente. e r um candidato ao valor da qualidade ótima do subgrafo. Tal atribuição vem diretamente da definição da função de qualidade.

Seja r a qualidade de um subgrafo. Temos que:

$r = (f_1 + f_2 + ... + f_n)/(d_1 + d_2 + ... + d_n) \Rightarrow$

$(f_1 - (d_1 * r)) + (f_2 - (d_2 * r)) + ... + (f_n - (d_n * r)) = 0$

Podemos então adotar tal peso para as arestas e solucionar o problema buscando o valor de r que satisfaça a equação, conforme descrito a seguir.

3 Breve Descrição da Solução

Dado um grafo G não-direcionado, verificamos primeiramente se ele é conectado (i.e. se há caminhos entre todos os vértices de G). Caso não seja, paramos a execução e informamos que ele não é conectado. Caso contrário, definimos dois limitantes - um inferior e um superior - para escolher um candidato r_{cand} à qualidade ótima. Calculamos então a árvore espalhadora máxima de G, baseando-se nos pesos calculados com o valor de r_{cand}. Em seguida, adicionamos à árvore todas as arestas que *aumentam* a qualidade do subgrafo. Verificamos então se o somatório de todos os pesos equivale a zero, conforme equação da seção anterior. Em caso positivo, r_{cand} é o valor ótimo para qualidade de G e o programa é finalizado. Em caso negativo, atualizamos ou o limitante superior ou o inferior, dependendo do valor resultante da soma dos pesos. Definimos em seguida um novo r_{cand} (seguindo em uma busca binária para o r_{cand} ótimo), recalculamos todos os pesos das arestas repetindo o processo.

4 Etapas da Solução

1. **Leitura da entrada padrão**
   A leitura dos dados é trivial. Apenas algumas observações:
   - Como não existe o significado de *direção* neste problema, para toda aresta $i \rightarrow j$ a aresta $j \rightarrow i$ também é adicionada;
   - Espera-se uma entrada válida. Ou seja: as linhas em branco separando as instâncias, quatro inteiros em cada linha para aresta, um único inteiro para o número de vértices, etc;
   - Não são consideradas arestas que partem e chegam ao mesmo vértice, por exemplo: 2 2 10 5;
   - É esperado um $V > 1$ para o número de vértices.

2. **Verificação de grafo conectado**
   Por meio de uma DFS partindo de um único ponto já é possível verificar se o grafo é conectado, dado que as arestas invertidas também são adicionadas no passo anterior. O custo é $O(V + E)$ em termos de tempo e $O(V + E)$ para o grafo armazenado com listas ligadas, em termos de memória.

3. **Escolha do candidato à qualidade ótima**
   O limitante inferior utilizado é 0 (zero) e o superior é a soma dos níveis de amizade (sempre maior que a qualidade ótima). O candidato é então a média entre os dois limitantes.

4. **Cálculo da àrvore espalhadora máxima**
   Utilizado Kruskal [1], com algoritmo de ordenação Mergesort ($O(E \log(E))$.

5. **Adição de arestas que aumentam a qualidade**
   Com a equação dos pesos definida na seção de modelagem, segue que, para cada f e d de uma aresta qualquer:

   $$peso > 0 => f - r * d > 0 => f > r * d => f/d > r$$

   Portanto adicionamos à árvore espalhadora máxima todas as arestas que possuem peso positivo.

6. **Atualização dos limitantes**
   Caso a soma dos pesos seja maior que zero, segundo a equação definida na seção de modelagem, significa que escolhemos um candidato muito baixo para qualidade. Portanto, atualizamos o limitante inferior para o valor do candidato e mantemos o superior como antes. Analogamente para a soma de pesos negativa: atualizamos o limitante superior para o valor do candidato e mantemos o limitante inferior. Assim, o tempo que o algoritmo leva para encontrar o valor correto é no pior caso $O(\log(\sum f_i))$, ou seja, a soma de todos os níveis de amizade do grafo original.

7. **Condições de parada**
   O algoritmo para quando encontra um candidato suficientemente bom (soma dos pesos é suficientemente próxima de zero) ou se os limitantes inferior e superior estão próximos demais (diferença entre os limitantes está suficientemente próxima de zero).

Portanto, o custo total do algoritmo é $O(E \log(E) \log(\sum(f_i)))$ em termos de tempo e $O(V+E)$ para a memória utilizando listas ligadas. Para todas as instâncias fornecidas e também algumas criadas manualmente, o programa executou em menos de 0.02 segundo na máquina de teste.

5 Dificuldades Encontradas

- Suporte às arestas duplicadas: como apresentadas no grafo do terceiro exemplo do enunciado do TP, assim como no terceiro do arquivo *test.in* disponibilizado no Moodle, algumas arestas poderiam estar duplicadas no grafo de entrada. Embora no fórum tenha sido dito que as chances disso acontecer na correção eram pequenas, a probabilidade positiva causou uma mudança considerável na implementação do programa no final do trabalho. Em um dado momento optei por substituir a estrutura de dados, de matriz a listas de adjacências, visando maior qualidade na modelagem e de código.

- Cálculo do peso/custo das arestas: sem o auxílio do professor ou dos monitores seria bastante difícil encontrar a solução do problema.

- Linguagem C++: já há algum tempo não programava com C ou C++ e a utilização de ponteiros mostrou-se desafiadora em alguns momentos. Porém serviu como uma ótima revisão.

6 Conclusão

Apesar de ser um problema aparentemente muito difícil, a atribuição de pesos utilizada no trabalho o torna factível em tempo polinomial. O uso de algoritmos como o de Kruskal dentro de um laço com busca binária foi bastante interessante, assim como de visualizar o algoritmo "buscando" a melhor configuração de amizades possível.

Muitos erros foram encontrados e corrigidos no decorrer da implementação, porém a maioria deles relacionados ao uso de ponteiros em C++, não aos algoritmos em si. Outro ponto a destacar foi o aumento do conhecimento em relação às estruturas de dados, que tiveram de ser substituídas durante a implementação do programa.

Referências

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.