CCM128 EXERCÍCIO-PROGRAMA 4

O MODELO DE KLEINBERG

Y. KOHAYAKAWA

Data de entrega: 9/7/2014 (23:55)

1. Introdução

Este EP é apenas uma especificação mais detalhada do Exercício 4.5.44, *Kleinberg graph model*, de Sedgewick e Wayne (Creatve Exercises).

Para entender o contexto em que se insere este exercício, recomanda-se uma leitura cuidadosa da Seção 4.5 de Sedgewick e Wayne. A leitura do Capítulo 20 de *Networks, Crowds, and Markets*: *Reasoning About a Highly Connected World*, de David Easley e Jon Kleinberg, é também fortemente recomendada. Veja `http://www.cs.cornell.edu/home/kleinber/networks-book/`. Aliás, este livro é uma excelente fonte para diversas linhas de pesquisa da ciência da computação atual.

2. O modelo de Kleinberg

Descrevemos aqui o *modelo de Kleinberg* de grafo aleatório. Consideraremos apenas o caso bidimensional desse modelo. Seja dado um inteiro positivo N. O grafo aleatório $\mathrm{KM}(N)$ pode ser gerado da seguinte forma: inicialmente, considere o *grafo-grade* $N \times N$, que é o grafo com conjunto de vértices $[N] \times [N]$ (aqui, $[N] = \{0, 1, \ldots, N-1\}$) e com dois vértices (x, y) e (x', y') adjacentes se e só se $|x - x'| + |y - y'| = 1$. Agora, adicionamos uma aresta adicional aleatória e_v para cada vértice $v = (x, y)$ desse grafo-grade. A aresta e_v incide em v. Para gerar a outra ponta de e_v, escolhemos um vértice $w = (x', y') \in [N] \times [N]$, $w \neq v$, aleatoriamente, com a probabilidade de gerarmos w proporcional a $1/d(v, w)$. Aqui, $d(v, w)$ é a distância de v e w na assim chamada métrica ℓ_1 ou *Manhattan distance*: $d(v, w) = d((x, y), (x', y')) = |x - x'| + |y - y'|$. (Lembrem-se do método `discrete(double a[])` de `StdRandom`.) Para gerarmos $\mathrm{KM}(N)$, adicionamos ao grafo-grade $N \times N$ todas as arestas e_v assim geradas. (Note que pode ocorrer de termos $e_v = e_{v'}$ para algum par $v \neq v'$.)

No grafo-grade $N \times N$, dois vértices típicos v e w estão a uma distância $\geq cN$, para alguma constante absoluta $c > 0$. Neste EP, você verá que a adição de apenas uma aresta nova por vértice torna essa distância típica muito menor: ela torna-se algo como $O((\log N)^2)$. Ademais, você verá que um algoritmo natural e simples de roteamento fornece caminhos curtos entre quaisquer dois vértices.

3. O algoritmo de Kleinberg

Sejam dados dois vértices $v = (x, y)$ e $w = (x', y')$ em $\mathrm{KM}(N)$. O algoritmo de Kleinberg procede de forma muito natural: sejam $v_1, \ldots, v_k$ os vizinhos de v em $\mathrm{KM}(N)$. Escolha um v_i que

Versão de 9 de julho de 2014, 22:48.

seja o mais próximo possível de w na distância ℓ_1. Agora execute este algoritmo recursivamente entre v_i e w.

4. Seu programa

Seu EP deve conter os seguintes componentes: `KMGraph.java` e `KAlgorithm.java` e quaisquer outros módulos adicionais que você achar interessante (por exemplo, para fazer o bônus abaixo, talvez você queira implementar um módulo gráfico separado).

4.1. **A classe `KMGraph.java`.** A classe `KMGraph.java` deve gerar grafos aleatórios de Kleinberg KM(N). A saída da chamada

```
$ java-introcs KMGraph N
```

deve ser um KM(N), no formato que temos usado para representar grafos (os vértices de KM(N) devem ser coisas como `314.159`, que representa o vértice $(314, 159)$). A classe `KMGraph.java` deve ter um construtor, a ser usado da seguinte maneira por seus clientes:

```
Graph G = new KMGraph(N);
```

4.2. **A classe `KAlgorithm.java`.** A classe `KAlgorithm.java` deve implementar o algoritmo de Kleinberg para encontrar caminhos curtos entre vértices dos grafos KM(N). Deve haver três modos de execução de `KAlgorithm`:

```
$ java-introcs KAlgorithm N
```

```
$ java-introcs KAlgorithm N file.gr
```

e

```
$ java-introcs KAlgorithm
```

No primeiro caso, `KAlgorithm` deve gerar um KM(N) e trabalhar com este grafo. O segundo caso é igual ao primeiro, mas seu programa deve, adicionalmente, gravar o grafo gerado KM(N) no arquivo de nome `file.gr`. Na terceira forma de execução de `KAlgorithm`, seu programa deve ler um grafo de Kleinberg KM(N) de `stdin`, e deve trabalhar com ele. Finalmente, para trabalhar com um grafo fixo, gravado em um arquivo (digamos, `f.gr`), o usuário poderá também dizer

```
$ java-introcs KAlgorithm - f.gr
```

onde `f.gr` é um arquivo que foi gerado fazendo-se

```
$ java-introcs KMGraph N > f.gr
```

ou

```
$ java-introcs KAlgorithm N f.gr
```

Uma vez que `KAlgorithm` é invocado e um grafo KM(N) está fixo, ele deve entrar em um modo interativo (ele passa a ler "comandos" do `stdin` e passa a executá-los). O usuário poderá então emitir comandos da forma `v:`, `v:max`, `v:w`, `v:w:p`, `rand:`, `rand:max`, `rand:rand` e `rand:rand:p`. O significado desses comandos é como segue.

`v:w`. neste caso, seu programa deve calcular e imprimir dois valores: (1) a distância entre `v` e `w` em KM(N) e (2) o comprimento do caminho encontrado pelo algoritmo de Kleinberg entre `v` e `w` em KM(N).

`v:w:p`. este caso é como `v:w`, mas devem também ser impressos os dois caminhos encontrados (um caminho de comprimento mínimo entre `v` e `w` e o caminho encontrado pelo algoritmo de Kleinberg).

`v:max`. neste caso, seu programa deve calcular e imprimir dois valores: (1) o *máximo* das distâncias entre `v` e os outros vértices de KM(N) e (2) o máximo dos comprimentos dos caminhos encontrados pelo algoritmo de Kleinberg entre `v` e todos os outros vértices de KM(N).

`v:` neste caso, seu programa deve calcular e imprimir dois valores: (1) a distância *média* de `v` a todos os outros vértices de KM(N) e (2) o comprimento *médio* dos caminhos encontrados pelo algoritmo de Kleinberg entre `v` e todos os outros vértices de KM(N).

`rand:rand`. neste caso, seu programa deve calcular e imprimir dois valores: (1) a distância entre dois vértices sorteados uniforme e independentemente ao caso em KM(N) e (2) o comprimento do caminho encontrado pelo algoritmo de Kleinberg entre estes dois vértices sorteados (isto é o mesmo que executar `v:w` com `v` e `w` gerados de forma aleatória).

`rand:rand:p`. isto é como se executássemos `v:w:p` com `v` e `w` gerados de forma aleatória (como acima).

`rand:max`. isto é como se executássemos `v:max` com `v` gerado de forma aleatória (como acima).

`rand:` isto é como se executássemos `v:` com `v` gerado de forma aleatória (como acima).

4.3. **Desempenho do algoritmo de Kleinberg.** Escreva um programa para verificar experimentalmente que o algoritmo de Kleinberg encontra caminhos de comprimento $O((\log N)^2)$ em KM(N) típicos. Faça vários experimentos para vários valores de N (grandes).

4.4. **Bônus.** Implemente métodos gráficos que facilitem ver como as coisas funcionam. Por exemplo, seu EP poderia desenhar os grafos KM(N) gerados ou sendo manipulados. Seu EP poderia também indicar em tais desenhos os caminhos mínimos encontrados por busca em largura e aqueles encontrados pelo algoritmo de Kleinberg.

Observações. Seguem algumas observações importantes.

1. *Este EP pode ser feito em quartetos* (isso mesmo, em quartetos!). Serão aceitos EPs feitos em grupos menores `:-)`
2. Seja cuidadoso com sua programação (correção, documentação, apresentação, clareza de código, etc). A correção não é baseada apenas na correção de seu programa.
3. Comparem entre vocês o desempenho de seus programas.
4. Entregue seu EP no Paca.
5. Não deixe de incluir em seu material um *relatório* para discutir seu EP: faça quaisquer comentários que você achar interessante e inclua exemplos de execuções de seu programa.

Observação final. Enviem dúvidas para a lista de discussão da disciplina.

Instituto de Matemática e Estatística, Universidade de São Paulo, Rua do Matão 1010, 05508–090 São Paulo, SP

Endereço eletrônico: `yoshi@ime.usp.br`