UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

INSTITUTO DE INFORMÁTICA

Guilherme Krüger Araujo (150821)

Trabalho 1 (Heaps binários)

Table of Contents

1 Objetivos

- Implementar um heap binário e verificar a complexidade das operações experimentalmente.
- Implementar o algoritmo de Dijkstra usando o heap implementado.
- Comparar a complexidade teórica pessimista com a complexidade real. Em particular verificar que a complexidade real respeita o limite teórico.

2 Solução

Implementei o algoritmo de Dijkstra utilizando um heap binário de acordo com o material e pseudo algoritmos retirados do material do prof. Marcus Ritt.

Porem na implementação realizada, o algoritmo termina ao encontrar o menor caminho para o vértice destino.

Porque cursivo?

3 Ambiente de testes

Os resultados foram obtidos num desktop, com processador Intel Core 2 Duo E8400 de 3 GHz, e 4 GB de RAM. Rodando Fedora 16, Kernel Linux 3.6.6-1.fc16.x86_64.

4 Como executar e testar

Os programas ~~fazem uso~~ da biblioteca Boost de C++. Para compilar ~~você precisa~~ ter essa biblioteca instalada.

A implementação do algoritmo de Dijkstra aceita um grafo no formato da DIMACS challenge na entrada padrão (stdin) e os índices de dois vértices origem e destino na linha de comando. Ele imprime o valor do caminho mais curto na saída padrão (stdout). Caso não existe caminho entre os dois vértices imprimir “inf”. Exemplo (em UNIX):

```
./dijkstra 1 2 < NY.gr 803
```

Compilando

Compile o gerador de teste:

```
c++ -o gen gen.cpp Compile o programa dijkstra c++ dijkstra dijkstra.cpp
```

Script de testes

O script de testes gera grafos de testes (aleatórios) e executa dijkstra para o grafo resultante. Ele repete isso ~~10~~ vezes para cada uma das quantidades de vertices (100, 200, 300, ..., 2000) dado uma determinada probabilidade de existência de arestas entre cada um dos vertices (input).

```
./run_tests.sh
```

O script de teste irá gerar os seguintes arquivos, para uma probabilidade de 0.5:

- experiment-0.5-time, contem o maior tempo de execução, a média e o menor tempo de execução para o determinado número de vertices.

- experiment-0.5-deletemin, contem o maior número de chamadas, a média e o menor número de chamadas a insert para o determinado número de vertices.
- experiment-0.5-insert, contem o maior número de chamadas, a média e o menor número de chamadas a insert para o determinado número de vertices.
- experiment-0.5-update, contem o maior número de chamadas, a média e o menor número de chamadas a update para o determinado número de arestas.
- experiment-0.5-all, contem os dados experimentais de cada uma das execuções.

Gerando casos de teste (grafos)

Para gerar casos de teste use gen.cpp, ex:

./gen 100 0.4 > test.gr Esse comando gera um grafo no formato DIMACS challenge.

Executando o algoritmo de Dijkstra

Para rodar dijkstra, passe como argumentos o vértice inicial, vértice final e o arquivo do grafo na entrada padrão, ex:

./dijkstra 1 2 < test.gr

5 Experimentos e Resultados

Os dados foram gerados, randomicamente, utilizando o gerador de casos de testes fornecido (gen.cpp) com número de vértices variando de 100, 200, 300,..., 2000 e a chance de existir arestas entre cada vertice 0.1, 0.5, 0.7 e 0.9. Cada combinação de teste foi repetida 10 vezes, gerado a média, maxima e minima dos seguintes dados:

- Tempo de execução
- Chamadas a deletemin()
- Chamadas a insert()
- Chamadas a update()

Porque: isso é incluido no resto.

Analise e discussão dos resultados

O algoritmo de Dijkstra possui complexidade:

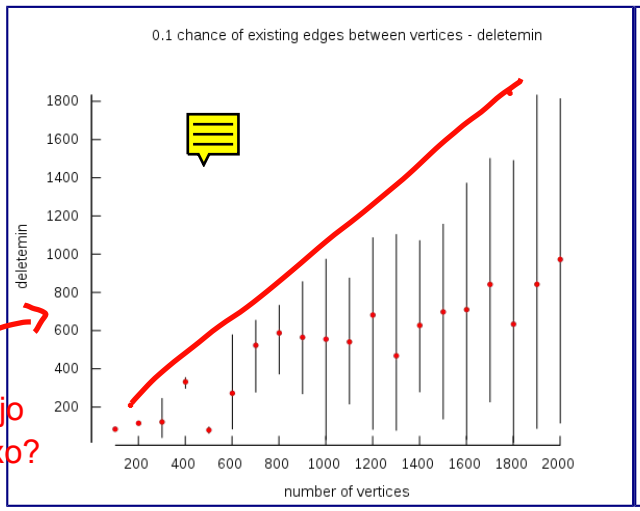
O(n) + n x deletemin + n x insert + m x update.

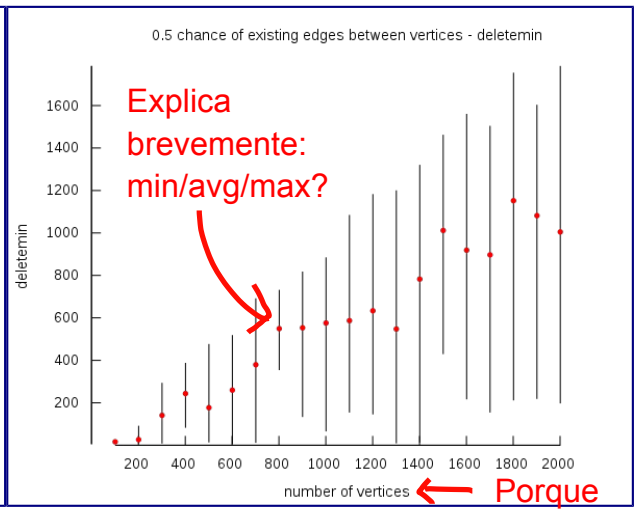
Por isso a utilização de uma boa estrutura de dados para a fila de prioridades utilizada no algoritmo de dijkstra é crucial para se obter complexidade melhor que quadrática. Utilizando um heap binário tem-se complexidade O(n log n + m log n). Isso porque é possivel realizar as operações de insert, deletemin e update em O(log n).

Veja abaixo os resultados das experimentações práticas.

deletemin

Podemos ver que o número de operações deletemin nos piores casos tem um crescimento linear O(n) em reação ao número de vértices do grafo (máximo obtido). O que esta de acordo com a complexidade teórica de Dijkstra. Na prática, a média fica próximo de n/2 para um grafo esparso (poucas arestas - 0.1) e para grafos mais densos (0.5 - 0.9) ainda possui a média bem abaixo do limite máximo. Nos melhores casos, o número de operações deletemin podem ser mínimas.

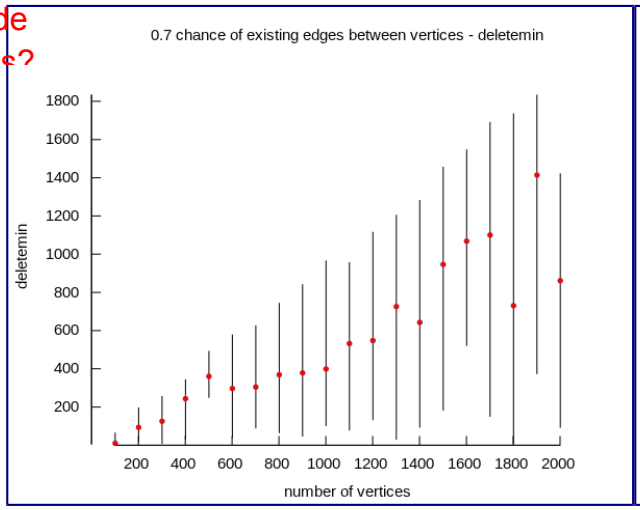


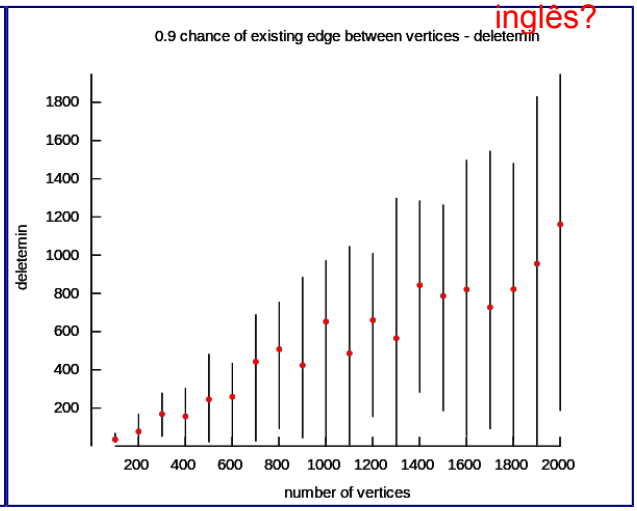


Explica brevemente: min/avg/max?

O que vejo nesse eixo? Tempo? Número de chamadas?

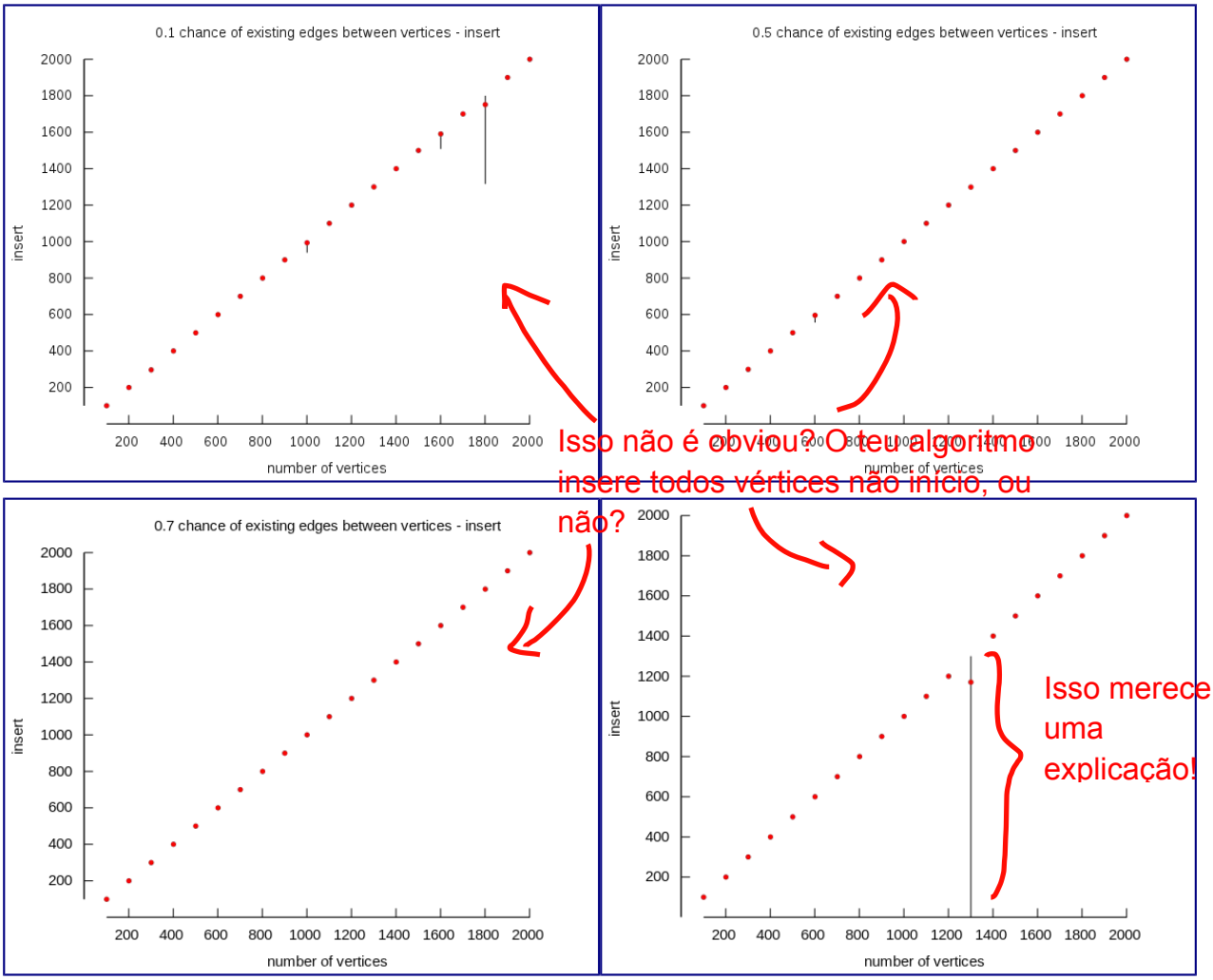
Porque inglês?





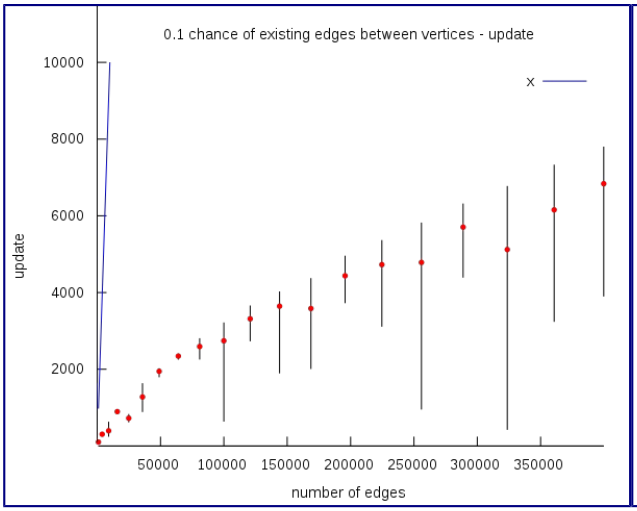
insert

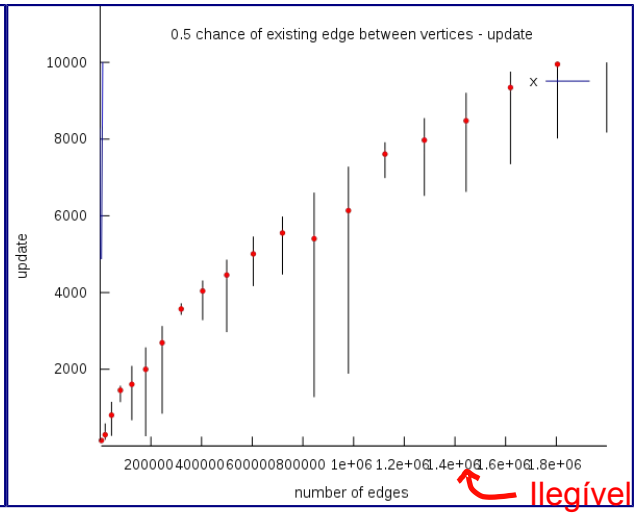
Em todos casos de testes, o algoritmo apresentou um comportamento bastante consistente no número de operações insert, ou seja, um crescimento linear ao número de vertices do grafo.

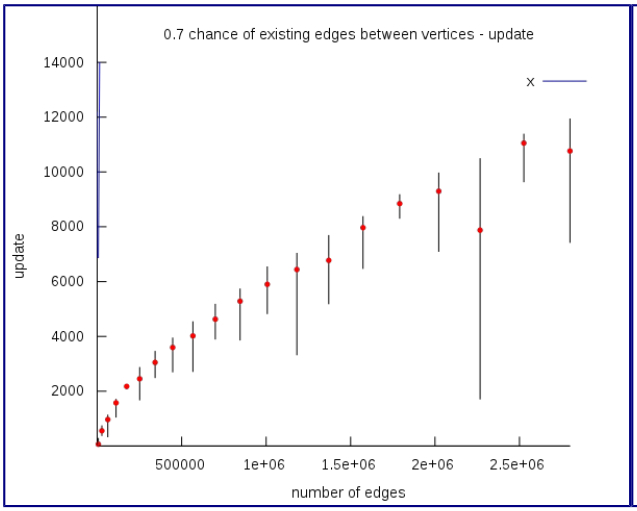


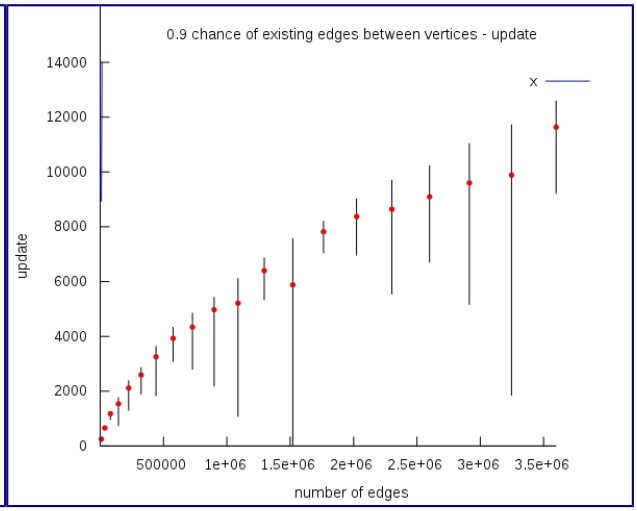
update

Teoricamente temos um crescimento linear no número de operações update em relação ao número de arestas do grafo, o que considera o pior caso. No entanto, na prática podemos verificar que isso é muito difícil de acontecer, seja para grafos densos ou esparsos. Pelo gráfico podemos ver que na prática é realmente muito menor que o valor esperado (reta x em azul).



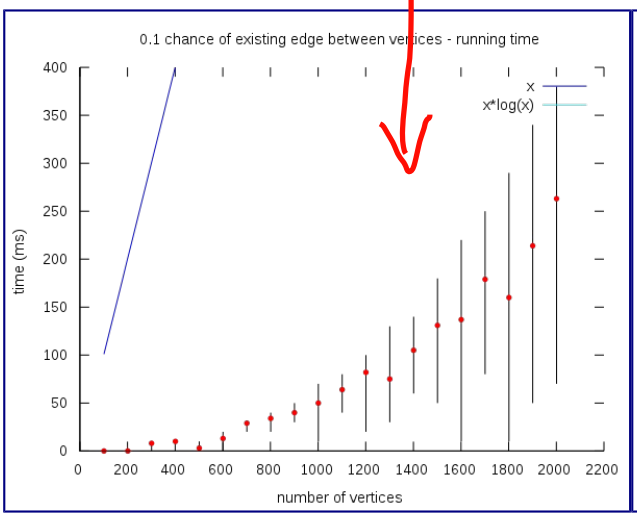


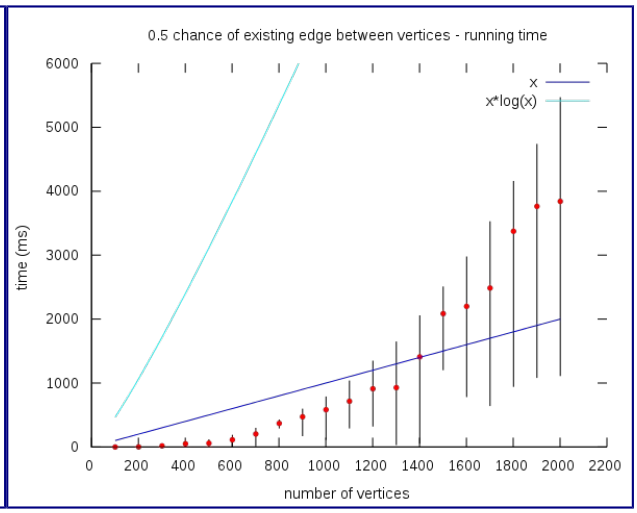


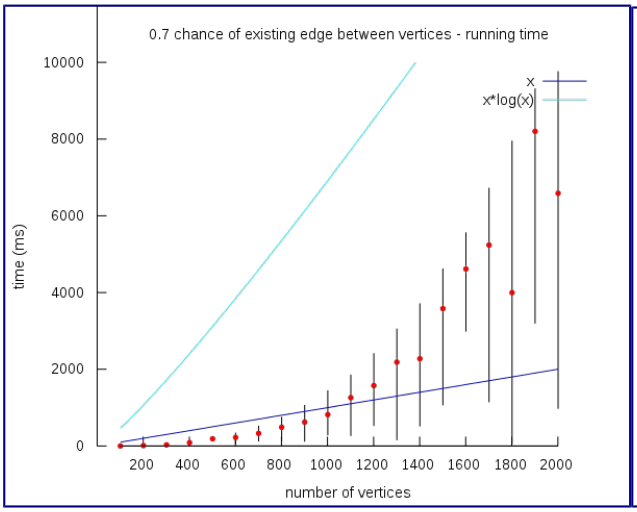


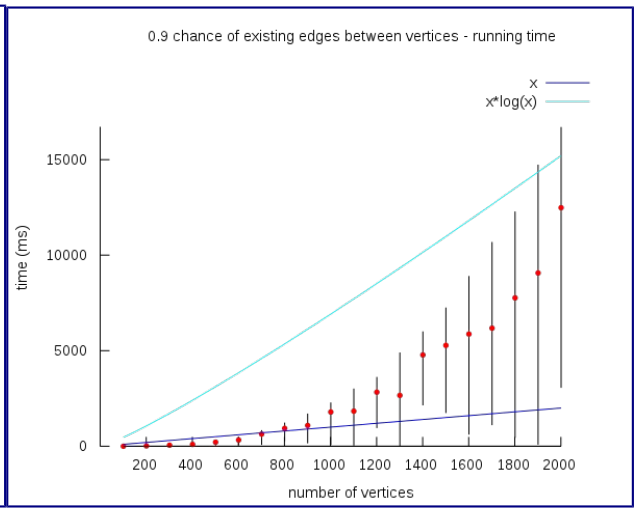
Tempo de execução

Por fim temos abaixo os resultados dos tempos de execução medidos de acordo com o crescimento no número de vértices do grafo. Nos resultados, para os números de vértices testados, obtivemos uma média abaixo da curva x*log x para o grafo mais denso (0.9), se aproximando do limite nos maiores números de vertices. Como era de se esperar, para grafos menos densos, a curva de tempo foi bastante inferior, ficando abaixo da curva x para o grafo mais esparso (0.1), ultrapassando a curva x apenas acima de 1400 vertices quando havia 0.5 de probabilidade de haver uma aresta entre cada vertice e acima de 1100 para 0.7.









Conclusão

Os resultados vão de encontro com a teoria em que temos O(n log n + m log n) para Dijkstra utilizando heap binário em sua fila de prioridades. Podemos concluir que o algoritmo respeita o limite teórico e possui um desempenho muito bom na prática.

Resumo

Relatório: Aborda a maioria dos objetivos, mas não verifica a complexidade das operações individuais, somente no contexto do Dijkstra. A análise da complexidade geral do Dijkstra é incorreto. O texto é legível, porém com muitos erros ortográficos e inconsistências e omissões na formatação. O texto também foca demais em detalhes técnicas (compilação, chamada de scripts etc.) 7.5pt.

Implementação: Ok, 5pt.

Nota: 7.5