Trabalho 1 (Heaps k-ários e algoritmo de Dijkstra)

Eduardo Menges Mattje

# Detalhes de implementação

O heap k-ário foi implementado com o fixado em tempo de compilação, através do recurso de *templates* do C++. Isso permitiu otimizar os cálculos dos índices da heap, ao custo de não poder defini-los em tempo de execução, o que para este trabalho não importava.

Ao invés de utilizar a geração de grafos disponibilizada pelo professor, construiu-se uma função que seleciona dois nodos aleatórios e liga-os caso não haja uma aresta prévia e eles não sejam iguais. Isso permite fixar o número de arestas *m* nos casos gerados, acelerar significativamente a criação de novos grafos, e eliminar a dependência da Boost. Além disso, o programa de teste constrói os grafos em tempo de execução, ao invés de primeiro gravar num arquivo e posteriormente lê-lo, já que sua construção só é relevante para uma primeira avaliação. Para acelerar a geração dos grafos, utilizou-se a biblioteca OpenMP, assim cada grafo é gerado em paralelo, e utilizou-se o *mimalloc* como implementação do *malloc*, na esperança de acelerar a alocação dinâmica (que é notoriamente lenta no Windows).

# Ambiente de teste

## Máquina

A máquina de teste possui Windows 11, um processador AMD Ryzen 5 7600X e 32 GB de memória RAM. O compilador utilizado foi o MSVC, versão 14.43.34808, com todas as otimizações padrões habilitadas.

## Parâmetros

Testou-se *K* no intervalo [2, 16], *n* (número de nós) no intervalo de 1000 até 512000 (dobrando a cada iteração), e *m* no intervalo de 2 \* n e dobrando até , ou seja, 90% do número máximo de arestas, com exceção de que os grafo com nós superiores a 64000 têm apenas 2% das arestas totais, em vista de que valores maiores demoraram muito para ser gerado devido à limitação da memória RAM.

Para cada entrada desse conjunto, escolheu-se 10 pares de vértices aleatórios (que são garantidamente diferentes) e rodou-se a implementação do algoritmo do Dijkstra 10 vezes, a fim de obter uma média que minimizasse a interferência da memória cache e não houvesse vieses quanto aos nós escolhidos.

# Resultados

## Determinação do K ótimo

Devido à necessidade de avaliar o tempo em relação a e , escolheu-se apresentar os dados com coloração para cada linha (independente das demais) onde regiões verdes representam os menores tempos para um mesmo e , e regiões vermelhas representam mais tempo, como um mapa de calor. A exibição conta somente com os grafos de 1000, 64000 e 512000 nós, para que ela caiba nas páginas.

| n | m | K=2 t(ms) | K=3 t(ms) | K=4 t(ms) | K=5 t(ms) | K=6 t(ms) | K=7 t(ms) | K=8 t(ms) | K=9 t(ms) | K=10 t(ms) | K=11 t(ms) | K=12 t(ms) | K=13 t(ms) | K=14 t(ms) | K=15 t(ms) | K=16 t(ms) |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1000 | 2000 | 7,43E-02 | 6,66E-02 | 6,26E-02 | 6,53E-02 | 6,44E-02 | 6,43E-02 | 6,60E-02 | 7,04E-02 | 7,28E-02 | 7,50E-02 | 7,65E-02 | 7,91E-02 | 8,07E-02 | 8,17E-02 | 8,14E-02 |
| 1000 | 4000 | 9,67E-02 | 8,57E-02 | 8,16E-02 | 8,19E-02 | 8,16E-02 | 8,13E-02 | 8,08E-02 | 8,62E-02 | 8,74E-02 | 9,16E-02 | 9,29E-02 | 9,48E-02 | 9,93E-02 | 1,00E-01 | 9,99E-02 |
| 1000 | 8000 | 1,22E-01 | 1,08E-01 | 1,06E-01 | 1,06E-01 | 1,04E-01 | 1,04E-01 | 1,08E-01 | 1,07E-01 | 1,08E-01 | 1,14E-01 | 1,15E-01 | 1,16E-01 | 1,20E-01 | 1,21E-01 | 1,21E-01 |
| 1000 | 16000 | 1,83E-01 | 1,66E-01 | 1,62E-01 | 1,64E-01 | 1,59E-01 | 1,57E-01 | 1,58E-01 | 1,59E-01 | 1,62E-01 | 1,71E-01 | 1,69E-01 | 1,71E-01 | 1,76E-01 | 1,79E-01 | 1,79E-01 |
| 1000 | 32000 | 2,54E-01 | 2,39E-01 | 2,36E-01 | 2,36E-01 | 2,31E-01 | 2,27E-01 | 2,30E-01 | 2,25E-01 | 2,27E-01 | 2,42E-01 | 2,43E-01 | 2,35E-01 | 2,46E-01 | 2,47E-01 | 2,48E-01 |
| 1000 | 64000 | 4,34E-01 | 4,25E-01 | 4,20E-01 | 4,24E-01 | 4,14E-01 | 4,07E-01 | 4,09E-01 | 4,01E-01 | 4,07E-01 | 4,31E-01 | 4,21E-01 | 4,05E-01 | 4,27E-01 | 4,31E-01 | 4,26E-01 |
| 1000 | 128000 | 7,61E-01 | 7,49E-01 | 7,39E-01 | 7,73E-01 | 7,41E-01 | 7,24E-01 | 7,29E-01 | 7,08E-01 | 6,96E-01 | 7,74E-01 | 7,43E-01 | 7,06E-01 | 7,55E-01 | 7,64E-01 | 7,48E-01 |
| 1000 | 256000 | 1,42E+00 | 1,42E+00 | 1,39E+00 | 1,45E+00 | 1,40E+00 | 1,37E+00 | 1,40E+00 | 1,35E+00 | 1,33E+00 | 1,48E+00 | 1,40E+00 | 1,36E+00 | 1,44E+00 | 1,46E+00 | 1,40E+00 |
| 64000 | 128000 | 1,47E+01 | 1,35E+01 | 1,30E+01 | 1,32E+01 | 1,30E+01 | 1,32E+01 | 1,35E+01 | 1,37E+01 | 1,36E+01 | 1,38E+01 | 1,41E+01 | 1,46E+01 | 1,50E+01 | 1,51E+01 | 1,54E+01 |
| 64000 | 256000 | 1,92E+01 | 1,74E+01 | 1,71E+01 | 1,73E+01 | 1,69E+01 | 1,70E+01 | 1,71E+01 | 1,74E+01 | 1,78E+01 | 1,80E+01 | 1,82E+01 | 1,85E+01 | 1,87E+01 | 1,92E+01 | 1,95E+01 |
| 64000 | 512000 | 2,10E+01 | 1,91E+01 | 1,86E+01 | 1,84E+01 | 1,81E+01 | 1,83E+01 | 1,83E+01 | 1,82E+01 | 1,86E+01 | 1,92E+01 | 1,95E+01 | 1,97E+01 | 1,98E+01 | 1,98E+01 | 2,02E+01 |
| 64000 | 1024000 | 2,61E+01 | 2,41E+01 | 2,37E+01 | 2,40E+01 | 2,36E+01 | 2,38E+01 | 2,37E+01 | 2,35E+01 | 2,38E+01 | 2,45E+01 | 2,43E+01 | 2,45E+01 | 2,49E+01 | 2,51E+01 | 2,51E+01 |
| 64000 | 2048000 | 3,47E+01 | 3,26E+01 | 3,17E+01 | 3,21E+01 | 3,17E+01 | 3,19E+01 | 3,12E+01 | 3,07E+01 | 3,09E+01 | 3,18E+01 | 3,24E+01 | 3,19E+01 | 3,24E+01 | 3,22E+01 | 3,24E+01 |
| 64000 | 4096000 | 4,79E+01 | 4,57E+01 | 4,41E+01 | 4,48E+01 | 4,39E+01 | 4,42E+01 | 4,39E+01 | 4,40E+01 | 4,38E+01 | 4,55E+01 | 4,48E+01 | 4,44E+01 | 4,56E+01 | 4,59E+01 | 4,52E+01 |
| 64000 | 8192000 | 6,92E+01 | 6,74E+01 | 6,70E+01 | 6,84E+01 | 6,61E+01 | 6,55E+01 | 6,53E+01 | 6,51E+01 | 6,57E+01 | 6,93E+01 | 6,75E+01 | 6,53E+01 | 6,64E+01 | 6,83E+01 | 6,68E+01 |
| 64000 | 16384000 | 1,12E+02 | 1,10E+02 | 1,07E+02 | 1,13E+02 | 1,08E+02 | 1,07E+02 | 1,06E+02 | 1,06E+02 | 1,06E+02 | 1,14E+02 | 1,10E+02 | 1,06E+02 | 1,09E+02 | 1,11E+02 | 1,08E+02 |
| 64000 | 32768000 | 2,00E+02 | 2,00E+02 | 1,94E+02 | 2,04E+02 | 1,96E+02 | 1,93E+02 | 1,92E+02 | 1,91E+02 | 1,89E+02 | 2,10E+02 | 1,97E+02 | 1,90E+02 | 1,97E+02 | 2,01E+02 | 1,95E+02 |
| 64000 | 65536000 | 3,55E+02 | 3,53E+02 | 3,50E+02 | 3,75E+02 | 3,55E+02 | 3,50E+02 | 3,51E+02 | 3,44E+02 | 3,40E+02 | 3,73E+02 | 3,55E+02 | 3,43E+02 | 3,54E+02 | 3,63E+02 | 3,49E+02 |
| 64000 | 1,31E+08 | 6,69E+02 | 6,68E+02 | 6,58E+02 | 7,03E+02 | 6,73E+02 | 6,54E+02 | 6,62E+02 | 6,55E+02 | 6,47E+02 | 7,08E+02 | 6,73E+02 | 6,44E+02 | 6,73E+02 | 6,92E+02 | 6,59E+02 |
| 64000 | 2,62E+08 | 1,27E+03 | 1,27E+03 | 1,26E+03 | 1,32E+03 | 1,27E+03 | 1,25E+03 | 1,26E+03 | 1,25E+03 | 1,23E+03 | 1,34E+03 | 1,28E+03 | 1,24E+03 | 1,28E+03 | 1,31E+03 | 1,25E+03 |
| 64000 | 5,24E+08 | 2,34E+03 | 2,35E+03 | 2,32E+03 | 2,44E+03 | 2,35E+03 | 2,30E+03 | 2,32E+03 | 2,30E+03 | 2,28E+03 | 2,46E+03 | 2,37E+03 | 2,29E+03 | 2,36E+03 | 2,42E+03 | 2,32E+03 |
| 64000 | 1,05E+09 | 4,62E+03 | 4,66E+03 | 4,59E+03 | 4,85E+03 | 4,67E+03 | 4,55E+03 | 4,58E+03 | 4,55E+03 | 4,50E+03 | 4,86E+03 | 4,68E+03 | 4,52E+03 | 4,66E+03 | 4,77E+03 | 4,59E+03 |
| 512000 | 1024000 | 1,79E+02 | 1,61E+02 | 1,59E+02 | 1,62E+02 | 1,65E+02 | 1,66E+02 | 1,66E+02 | 1,72E+02 | 1,68E+02 | 1,72E+02 | 1,72E+02 | 1,74E+02 | 1,76E+02 | 1,77E+02 | 1,78E+02 |
| 512000 | 2048000 | 2,03E+02 | 1,83E+02 | 1,79E+02 | 1,83E+02 | 1,84E+02 | 1,86E+02 | 1,84E+02 | 1,90E+02 | 1,89E+02 | 1,91E+02 | 1,92E+02 | 1,92E+02 | 1,95E+02 | 1,96E+02 | 1,97E+02 |
| 512000 | 4096000 | 2,29E+02 | 2,08E+02 | 2,03E+02 | 2,06E+02 | 2,04E+02 | 2,08E+02 | 2,08E+02 | 2,12E+02 | 2,10E+02 | 2,15E+02 | 2,13E+02 | 2,13E+02 | 2,16E+02 | 2,19E+02 | 2,18E+02 |
| 512000 | 8192000 | 2,73E+02 | 2,48E+02 | 2,47E+02 | 2,47E+02 | 2,47E+02 | 2,48E+02 | 2,49E+02 | 2,50E+02 | 2,49E+02 | 2,57E+02 | 2,58E+02 | 2,55E+02 | 2,57E+02 | 2,62E+02 | 2,60E+02 |
| 512000 | 16384000 | 3,48E+02 | 3,25E+02 | 3,19E+02 | 3,23E+02 | 3,22E+02 | 3,24E+02 | 3,25E+02 | 3,24E+02 | 3,25E+02 | 3,34E+02 | 3,28E+02 | 3,23E+02 | 3,32E+02 | 3,37E+02 | 3,33E+02 |
| 512000 | 32768000 | 5,26E+02 | 4,96E+02 | 4,80E+02 | 4,99E+02 | 4,91E+02 | 4,88E+02 | 4,79E+02 | 4,85E+02 | 4,79E+02 | 4,97E+02 | 4,94E+02 | 4,85E+02 | 4,99E+02 | 5,03E+02 | 4,88E+02 |
| 512000 | 65536000 | 6,90E+02 | 6,73E+02 | 6,53E+02 | 6,78E+02 | 6,65E+02 | 6,60E+02 | 6,55E+02 | 6,56E+02 | 6,47E+02 | 6,89E+02 | 6,68E+02 | 6,51E+02 | 6,80E+02 | 6,89E+02 | 6,64E+02 |
| 512000 | 1,31E+08 | 1,30E+03 | 1,25E+03 | 1,20E+03 | 1,26E+03 | 1,21E+03 | 1,19E+03 | 1,17E+03 | 1,18E+03 | 1,17E+03 | 1,24E+03 | 1,20E+03 | 1,17E+03 | 1,21E+03 | 1,22E+03 | 1,18E+03 |
| 512000 | 2,62E+08 | 1,96E+03 | 1,94E+03 | 1,88E+03 | 1,99E+03 | 1,92E+03 | 1,89E+03 | 1,88E+03 | 1,89E+03 | 1,87E+03 | 2,00E+03 | 1,93E+03 | 1,86E+03 | 1,92E+03 | 1,96E+03 | 1,90E+03 |
| 512000 | 5,24E+08 | 3,33E+03 | 3,34E+03 | 3,26E+03 | 3,44E+03 | 3,31E+03 | 3,27E+03 | 3,25E+03 | 3,23E+03 | 3,19E+03 | 3,44E+03 | 3,29E+03 | 3,19E+03 | 3,30E+03 | 3,35E+03 | 3,24E+03 |
| 512000 | 1,05E+09 | 5,81E+03 | 2,96E+04 | 5,73E+03 | 6,03E+03 | 5,78E+03 | 5,72E+03 | 5,71E+03 | 5,70E+03 | 5,65E+03 | 6,08E+03 | 5,87E+03 | 5,64E+03 | 5,83E+03 | 5,93E+03 | 5,73E+03 |

Valores na região mostraram-se melhores no caso médio. Há um notório viés aos valores superiores conforme o grafo se torna mais denso, atingindo um limite quando . No entanto, mostrou-se um valor repetidamente bom para grafos densos e de grande número de nós, o que talvez se dê por ser um ponto de boa localidade espacial com a memória cache do sistema, otimizando a troca de páginas realizada pelo SO. É possível que pontos similares estejam localizados para , porém, na montagem desses algoritmos, consideramos otimizar para o caso médio de uso, além de que tais otimizações não costumam ser portáteis, sendo altamente dependente do tipo do hardware e da configuração do SO.

# Contagem de operações

## Heap

O para as operações que envolvem *sift up* mostrou-se independente da quantidade de nós , conforme esperado. Além disso, há uma notável curva de decaimento logarítmico, confirmando a complexidade teórica.

O mesmo se aplica para o do *sift down*.

## Algoritmo

Conforme esperado, as operações de *push* e *pop* tem como cota máxima, em vista de que no máximo todos os nós entrarão na heap e todos os que entrarem sairão dela.

Já o número de *updates* é menor do que a quantidade esperada, calculada com . A quantidade cresce conforme o grafo se torna mais conexo, havendo uma assíntota.

# Complexidade por tempo

Ao fixar e variar a quantidade de arestas , há uma linearização nos grafos mais conexos.

Isso é esperado, em vista de que a complexidade do Dijkstra utilizando um *heap* é , desta forma, conforme atinge valores muito maiores do que , ele se torna o principal expoente da equação. Isso pode ser mais facilmente visualizado se utilizarmos escala logarítmica no eixo :

Note como, em valores baixos de , o gráfico cresce linearmente. Isso ocorre porque o termo dominante da equação ainda é , havendo um ponto em que se torna tão grande que o crescimento é exponencial (na escala logarítmica).

# Escalonamento

Para o grafo das estradas de Nova Iorque, selecionando 50 pares de nós garantidamente diferentes, e utilizando o Dijkstra com *heap* 4-ária, o algoritmo demorou 28,8201ms em média para executar e consumiu 14,9MB de memória (incluindo a memória utilizada para carregar o grafo do arquivo).

Já para o grafo das estradas dos Estados Unidos, utilizando a mesma metodologia, o algoritmo demorou 4472,51ms em média para executar e consumiu 1993MB de memória (também incluindo a memória utilizada para carregar o grafo do arquivo).

Essa diferença é coerente. A razão entre as complexidades pessimistas dos dois casos é 112, enquanto a razão do tempo médio é 159, diferença que pode ser explicada devido ao manejo de páginas do sistema operacional.

# Análise

Através deste experimento, podemos verificar que a complexidade prática respeita a teórica. Além disso, que a contagem de operações é mais precisa para a comparação da complexidade do que o tempo de execução, devido a este estar intimamente ligado ao hardware e ao gerenciamento do sistema operacional.