Aula Prática 7

1 - Bateria de Testes

Primeiramente, devemos definir os parâmetros que serão usados no Shell Sort. Isso é, o valor inicial de h e sua regra de atualização. Já sabemos a regra de atualização empiricamente ótima de h, que é a sequência de Knuth(h = h * 3 + 1). Porém, utilizaremos nossa própria regra:

- h inicial: 377
- Regra de Atualização de h: Iremos utilizar a Sequência de Fibonacci de maneira descendente, começando de 377 até chegar em seu primeiro elemento, que é 1.

Para efetivamente testarmos a eficiência do nosso algoritmo, devemos analisá-lo de acordo com vetores(entradas) de tamanhos diferentes preenchidos de forma aleatória. O tamanho do vetor, é claro, deve ser menor ou igual ao h em qualquer um dos seus valores disponíveis.

Entretanto, para que esses testes sejam verdadeiramente representativos, devemos ter vetores com um número de combinações diferentes PROPORCIONAL ao tamanho daquele vetor. Ou seja, enquanto que para um vetor de 10 elementos, testaríamos 2 combinações aleatórias diferentes dele, para um vetor de 100 elementos, testaríamos 20 combinações aleatórias diferentes.

Faremos nossos testes em duas etapas. Na primeira, com 10 vetores cujos tamanhos variam de 377 até 2000. Na segunda, com 10 vetores que variam de 37700 até 200000.

- Aproximadamente, na primeira etapa, cada vetor terá uma diferença de 180 de tamanho. Na segunda etapa, cada vetor terá diferença de 18000 de tamanho.
- Iremos variar o número de combinações testadas para cada vetor de acordo com a seguinte fórmula: 10 + 10*n, onde n = índice do vetor quando os 10 vetores são ordenados(por tamanho) de forma crescente(varia de 0 até 10).

Em cada teste, será medido o tempo de execução do algoritmo inteiro para que posteriormente, possamos comparar sua eficiência com o Heap Sort.

2 - Relatório

Após executar nosso programa e registrar seus resultados, podemos chegar a algumas conclusões. Não foi necessário executá-lo várias vezes, pois ele já possui inúmeros casos de teste.

Além disso, o programa soma o tempo de execução de todos os casos de teste, gerando um tempo de execução total para todos os testes. Como temos o mesmo número de testes para os dois algoritmos, o tempo de execução médio e o tempo de execução total podem ambos ser usados para comparação.

Etapa 1:
O ShellSort se provou mais eficiente na etapa 1 da bateria de testes, na qual os vetores possuem tamanhos menores.
Shell sort: 0.292932 ns Heap sort: 0.367514 ns

Etapa 2:
O HeapSort se provou mais eficiente na etapa 2 da bateria de testes, na qual os vetores possuem tamanhos 100 vezes maiores.
Shell sort: 105.331 ns Heap sort: 63.2717 ns

Ao executar o programa, é possível verificar o tempo de cada teste e o tamanho de sua entrada individualmente caso seja de interesse. Também foi disponibilizado um arquivo de texto com todos esses resultados. O volume de dados é MUITO grande, portanto não será apresentado completamente neste documento.