Projeto 1 - Algoritmos de Ordenação

[João Gabriel Dornellas Joaquim] - Matricula: 3913

8 de outubro de 2018

1 Introdução e Objetivos

Neste relatório são reportadas análises de tempo de execução dos algoritmos de ordenação selection sort, insertion sort, shell sort, merge sort, quick sort e heap sort, a fim de, compará-los computacionalmente sua eficiencia e tempo de execução.

2 Materiais e Métodos

Os algoritmos estudados neste trabalho foram implementados sobre a linguagem de programação Python e avaliados em um *hardware* composto com um processador FX-8300, 1 Terabyte de HDD e 16 gb de RAM, e uma placa de video Radeon RX 560.

Para avaliar os algoritmos, foram utilizados vários casos de teste (entradas de tamanhos diversos), desde 10 (dez) elementos até 10^6 (1 milhão) de elementos a serem comparados.

2.1 Selection Sort

O algoritmo selection sort (ou algoritmo de classificação por seleção) ordena uma lista de elementos localizando repetidamente o valor minimo da lista (considerando que esteja ordenando em ordem crescente) da parte não classifica e coloca-o no inicio. O algoritmo mantém dois subarrays em um determinado array.

1. O subarray que já está classificado.
2. Subarray restante que não está classificado.

Em cada iteração de classificação de seleção, o elemento mínimo (considerando a ordem crescente) do subarray não classificado é selecionado e movido para o subarray classificado.

- Pior Caso: $O(n^2)$
- Melhor Caso: $O(n^2)$

2.2 Insertion Sort

Existem muitos modos diferentes de ordenar. Conforme a ordenação por seleção é executada, o subarray no começo do array é ordenado, mas o subarray no final não. A ordenação por seleção (selection sort) escaneia o subarray não ordenado em busca do próximo elemento a ser incluído nele.

Porém na ordenação por inserção um novo elemento é colocado na lista e necessita ser colocado em sua posição correta, considerando que temos uma lista com toda sequencia correta, porém o novo elemento deve ser verificado com a lista anterior (já ordenada) para ser inserida na posição correta.

Uma boa analogia atrás da ordenação por inseção é de se utilizar um baralho em mãos, em que a cada nova carta você precisa colocar na posição correta para voltar a sequência certa.

- Pior Caso: $O(n^2)$
- Melhor Caso: $O(n^2)$

2.3 Shell sort

Criado por Donald Shell em 1959, publicado pela Universidade de Cincinnati, Shell sort é o mais eficiente algoritmo de classificação dentre os de complexidade quadrática. É um refinamento do método de inserção direta. A ideia do Shell Sort é permitir a troca de itens distantes, através da divisão da lista de elementos várias vezes. Através de um vetor h que será dividido até que o mesmo tenha valor 1. Um vetor é classificado com h quando todas as sub listas de todos os elementos forem classificados.

- (Depende da sequência do gap. Melhor conhecida) - Pior Caso: $O(nlog_2 n)$
- Melhor Caso: $O(nlog2n)$

Fonte: $https://pt.wikipedia.org/wiki/Shell_sort$

2.4 Merge sort

O algoritmo de ordenação Merge Sort funciona de maneira recursiva, dividindo um arranjo pela metade até chegar no tamanho de um. Isso se deve por causa de seu método chamado de Divisão e Conquista, funciona através da chamada dessas duas metades e então faz merge (união, junta as duas partes) dessas duas metades. A função merge fica responsável pela união dessas duas metades. O arranjo é divido recursivamente até chegar no valor 1, assim que o valor 1 é alcançado, a função merge é acionada e começa a união dos vetores até que todo o arranjo esteja unificado novamente (em ordem crescente).

- Pior Caso: $O(nlog(n))$
- Melhor Caso: $O(nlog(n))$

2.5 Quick sort

O algoritmo de ordenação Quick Sort é feito com divisão e conquista assim como o Merge Sort. Ele pega um elemento como um pivô e particiona a patir deste pivô em dois arranjos. Existe várias maneiras de escolher o pivô no Quick Sort:

1. Sempre pegando o primeiro elemento como pivô.
2. Sempre pegando o ultimo elemento como pivô.
3. Pegando um elemento aleatório
4. Pegando a mediana como pivô
5. Pegando a média como pivô

O processo principal do Quick Sort são as partições. O alvo das partições é, dado um vetor e um elemento x deste vetor como pivô, coloca-se o x em sua posição correta em um vetor ordenado e todos os elementos menores (do que x) antes de x, e todos os elementos maiores (do que x) depois de x. Tudo isso deve ser feito em um tempo linear.

- Pior Caso: $O(n^2)$
- Melhor Caso: $O(nlog(n))$

2.6 Heap sort

Heap Sort é um algoritmo de ordenação baseado na estrutura de Heap Binária (Arvore binária quase completa). É similar ao algoritmo de seleção, no qual procura-se o elemento e coloca o elemento máximo no final, em seguida é feito o mesmo processos aos elementos restantes.

Já que o Heap Binário é uma arvore completa, este pode ser facilmente representado como um arranjo e representações baseadas em arranjos são efficientes em questão de espaço. Se o nó pai for armazenado no índice I, o filho da esquerda pode ser calculado por 2 * I + 1 e filho da direita por 2 * I + 2 (assumindo que a indexação começa em 0).

Operações do algoritmo Heap Sort

- Heapify: Manter as propriedades da heap (minima ou máxima)
- Build-Heap: Contruir uma heap a partir do arranjo de elementos.
- Heapsort: Ordenar uma sequencia de elementos através da heap construida.

Algoritmo Heap Sort para ordenação em ordem crescente

1. Contruir um "max heap"através dos dados de entrada
2. A partir desse ponto, o maior elemento é designado como raiz da arvore. Substituindo ele como o ultimo item do heap e reduzindo o tamanho da heap em 1. Finalmente aplicando "heapfiy"na raiz da arvore.
3. Repetir os passos enquanto o tamanho da heap for maior do que 1.

- Pior Caso: $O(nlog(n))$
- Melhor Caso: $O(nlog(n))$

3 Resultados

- Tempos de execução com elementos em ordem aleatória (em segundos)

Algoritmos	Tamanho da entrada (n)				
*	10	1000 (10^3)	10000 (10^4)	100000 (10^5)	1000000 (10^6)
Selection Sort	0.000018	0.084204	2.849129	285.135197	
Insertion Sort	0.000021	0.084204	8.179446	967.012656	
Shell Sort	0.000024	0.005289	0.080124	1.244556	14.534917
Merge Sort	0.000060	0.009441	0.126121	1.457441	18.998034
Quick Sort	0.000022	0.004424	0.059699	0.723756	8.941398
Heap Sort	0.000039	0.009971	0.141920	1.833765	23.470744

- Tempos de execução com elementos em ordem crescente (em segundos)

Algoritmos	Tamanho da entrada (n)				
*	10	1000 (10^3)	10000 (10^4)	100000 (10^5)	1000000 (10^6)
Selection Sort	0.000021	0.035809	3.385523	344.476028	———
Insertion Sort	0.000008	0.000482	0.003774	0.044003	0.411550
Shell Sort	0.000017	0.002183	0.031757	0.407817	5.065068
Merge Sort	0.000054	0.008450	0.102171	1.192201	14.337507
Quick Sort	0.000029	——	———-	———	———
Heap Sort	0.000041	0.010126	0.139230	1.722163	21.732482

- Tempos de execução com elementos em ordem decrescente (em segundos)

Algoritmos	Tamanho da entrada (n)				
*	10	1000 (10^3)	10000 (10^4)	100000 (10^5)	1000000 (10^6)
Selection Sort	0.000021	0.041745	4.067894	425.531311	———
Insertion Sort	0.000021	0.158128	14.630378		———
Shell Sort	0.000023	0.003744	0.049709	0.638046	7.934836
Merge Sort	0.000055	0.008495	0.108266	1.278130	14.468340
Quick Sort	0.000023	———	———	———	———-
Heap Sort	0.000033	0.008625	0.126750	1.690909	20.100499

Observações:

- Os códigos de Quicksort estão com problemas de profundidade de recursão excedida na comparação (por isso falta a maioria dos dados crescentes e decrescentes)
- Os tempos que faltam de selection e insertion se deve a demanda de tempo real para ordenar (principalmente em casos de 10^5 e 10^6)

A partir dos resultados das tabelas é possível verificar que os algoritmos Insertion Sort e Selection Sort não são viáveis para ordenação de arranjos com números muito grandes de elementos, principalmente levando o caso de tempo de ordenação em todas as tabelas. Heap Sort entre os algortimos Shell, Merge e Quick foi um dos mais demorados no processo, e entre todos os algoritmos de ordenação, o Quick Sort leva como execução o algoritmo mais rapido para ordenar, em seguida com o Merge Sort e o Shell Sort competindo entre si, no tempo de execução de elementos arranjados de maneira aleatória, pois de maneira quando arranjados de maneira decrescente e crescente, o ganho é favorável ao Shell Sort.

4 Conclusões

Os tempos de execução dos algortimos de ordenação são condizentes com anotação O(n) disponivél na literatura, os algoritmos de ordenação selection e insertion sort principalmente são inviáveis para conjuntos de elementos muito grandes.

O Quick Sort é um dos algoritmos de ordenação mais rápido entre a lista, seguido do Shell Sort, Merge Sort e Heap Sort.