Análisis de Algoritmos y Estructuras para Datos Masivos 2023

Unidad 6 - Tarea

David Aarón Ramírez Olmeda

Introducción:

En este ejercicio se planteó el problema de la intersección de listas de posteo, es decir, encontrar los documentos que contienen todas las palabras de una consulta. Para esto, se implementaron los algoritmos de unión y búsqueda dados en la lectura 3.1 Melding Algorithms y 3.2 Search algorithm de [@BLOLS2010]. Se realizó un experimento para medir el tiempo y el número de comparaciones necesarios para realizar la intersección de pares, tercias y cuartetas de listas de posteo seleccionadas aleatoriamente.

Desarrollo:

import itertools import time import random import numpy as np import matplotlib.pyplot as plt import json

with open('listas-posteo-100.json') as f: data = f.readlines()

postings_lists = {} for line in data: term, postings = json.loads(line) postings_lists[term] = postings

Definimos los algoritmos

def melding_algorithm(lists): merged = [] for lst in lists: merged.extend(lst) return sorted(merged)

def binary_search(lst, x): low = 0 high = len(lst) - 1 while low <= high: mid = (low + high) // 2 if lst[mid] == x: return True elif lst[mid] < x: low = mid + 1 else: high = mid - 1 return False

def galloping_search(A, key): pos = 0 jump = 1 while pos < len(A) and A[pos] < key: pos += jump jump *= 2 left = pos // 2 right = min(pos, len(A)-1) while left <= right: mid = (left + right) // 2 if A[mid] == key: return mid elif A[mid] < key: left = mid + 1 else: right = mid - 1 return None

#postings_lists = {'_url': [1, 2, 3, 4, 6], '_date': [2, 4, 5, 6, 8], '_lol': [1,5,9]}

Prueba pequeña

pairs = list(itertools.combinations(postings_lists.values(), 2)) random.shuffle(pairs) A = pairs[:1000]

triplets = list(itertools.combinations(postings_lists.values(), 3)) random.shuffle(triplets) B = triplets[:1000]

quadruplets = list(itertools.combinations(postings_lists.values(), 4)) random.shuffle(quadruplets) C = quadruplets[:1000]

Se generaron aleatoriamente conjuntos de pares, tercias y cuartetas de listas de posteo

binary search

results = {'A': [], 'B': [], 'C': []} for i, lsts in enumerate(A): start_time = time.time() intersection = melding_algorithm(lsts) comparisons = len(lsts) - 1 # número de comparaciones en el algoritmo de fusion for j in range(len(lsts) - 1): comparisons += len(intersection) - 1 # número de comparaciones en el algoritmo de intersección intersection = [x for x in intersection if binary_search(lsts[j+1], x)] end_time = time.time() elapsed_time = end_time - start_time results['A'].append({'time': elapsed_time, 'comparisons': comparisons, 'length': len(intersection)})

for i, lsts in enumerate(B): start_time = time.time() intersection = melding_algorithm(lsts) comparisons = len(lsts) - 1 for j in range(len(lsts) - 1): comparisons += len(intersection) - 1 intersection = [x for x in intersection if binary_search(lsts[j+1], x)] end_time = time.time() elapsed_time = end_time - start_time results['B'].append({'time': elapsed_time, 'comparisons': comparisons, 'length': len(intersection)})

for i, lsts in enumerate(C): start_time = time.time() intersection = melding_algorithm(lsts) comparisons = len(lsts) - 1 for j in range(len(lsts) - 1): comparisons += len(intersection) - 1 intersection = [x for x in intersection if binary_search(lsts[j+1], x)] end_time = time.time() elapsed_time = end_time - start_time results['C'].append({'time': elapsed_time, 'comparisons': comparisons, 'length': len(intersection)})

Para cada conjunto, se realizó la intersección de las listas usando los algoritmos de unión y búsqueda definidos anteriormente. Se midió el tiempo y el número de comparaciones necesarios para realizar cada intersección.

fig1, ax1 = plt.subplots() ax1.boxplot(tiempos) ax1.set_title('Tiempos de Intersección') ax1.set_xticklabels(['A', 'B', 'C']) ax1.set_ylabel('Tiempo (segundos)') ax1.set_ylim([0, 0.03])

Esta gráfica muestra los tiempos de intersección para tres experimentos. Cada experimento contiene múltiples mediciones de tiempo de intersección. Los boxplots representan la distribución de estos tiempos. Podemos observar claramente que el tiempo aumenta dependiendo de los grupos (pares, tercias y cuartetas de listas de posteo)

fig2, ax2 = plt.subplots() ax2.boxplot(comparaciones) ax2.set_title('Número de Comparaciones') ax2.set_xticklabels(['A', 'B', 'C']) ax2.set_ylabel('Comparaciones') ax2.set_ylim([0, 16000])

Esta gráfica muestra el número de comparaciones para los mismos tres experimentos ('A', 'B' y 'C'). Cada experimento contiene múltiples mediciones del número de comparaciones realizadas. Los boxplots representan la distribución de estos números de comparaciones. La observación es muy parecida a la que hemos hecho anteriormente.

fig3, ax3 = plt.subplots() ax3.boxplot(intersecciones) ax3.set_title('Longitud de Intersecciones') ax3.set_xticklabels(['A', 'B', 'C']) ax3.set_ylabel('Longitud de Intersección') ax3.set_ylim([0, 1400])

Por último, esta gráfica muestra la longitud de las intersecciones para los tres experimentos ('A', 'B' y 'C'). Cada experimento contiene múltiples mediciones de la longitud de la intersección. Los boxplots representan la distribución de estas longitudes. Aquí observamos que, entre más corta

sea la agrupación (pares) más es la longitud de intersecciones, por eso en el caso de C, cuartetas, la longitud de intersecciones se ruduce.

galloping search

for i, lsts in enumerate(A): start_time = time.time() intersection = melding_algorithm(lsts) comparisons = len(lsts) - 1 for j in range(len(lsts) - 1): comparisons += len(intersection) - 1 intersection = [x for x in intersection if galloping_search(lsts[j+1], x)] end_time = time.time() elapsed_time = end_time - start_time results['A'].append({'time': elapsed_time, 'comparisons': comparisons, 'length': len(intersection)})

for i, lsts in enumerate(B): start_time = time.time() intersection = melding_algorithm(lsts) comparisons = len(lsts) - 1 for j in range(len(lsts) - 1): comparisons += len(intersection) - 1 intersection = [x for x in intersection if galloping_search(lsts[j+1], x)] end_time = time.time() elapsed_time = end_time - start_time results['B'].append({'time': elapsed_time, 'comparisons': comparisons, 'length': len(intersection)})

for i, lsts in enumerate(C): start_time = time.time() intersection = melding_algorithm(lsts) comparisons = len(lsts) - 1 for j in range(len(lsts) - 1): comparisons += len(intersection) - 1 intersection = [x for x in intersection if galloping_search(lsts[j+1], x)] end_time = time.time() elapsed_time = end_time - start_time results['C'].append({'time': elapsed_time, 'comparisons': comparisons, 'length': len(intersection)})

tiempos = [] comparaciones = [] intersecciones = []

for experimento in ['A', 'B', 'C']: tiempos.append([result['time'] for result in results[experimento]]) comparaciones.append([result['comparisons'] for result in results[experimento]]) intersecciones.append([result['length'] for result in results[experimento]])

fig1, ax1 = plt.subplots() ax1.boxplot(tiempos) ax1.set_title('Tiempos de Intersección') ax1.set_xticklabels(['A', 'B', 'C']) ax1.set_ylabel('Tiempo (segundos)') ax1.set_ylim([0, 0.05])

fig2, ax2 = plt.subplots() ax2.boxplot(comparaciones) ax2.set_title('Número de Comparaciones') ax2.set_xticklabels(['A', 'B', 'C']) ax2.set_ylabel('Comparaciones') ax2.set_ylim([0, 16000])

fig3, ax3 = plt.subplots() ax3.boxplot(intersecciones) ax3.set_title('Longitud de Intersecciones') ax3.set_xticklabels(['A', 'B', 'C']) ax3.set_ylabel('Longitud de Intersección') ax3.set_ylim([0, 1400])

print('Para el algoritmo galloping_search se tiene:') plt.show()

Analogamente y como hemos hecho con el algorimos anterior, las mismas observaciones aplican aquí. Vemos que el tiempo y número de intersecciones aumentan si el grupo es más grande y la longitud disminuye para grupos más grandes.

Conclusión

Se observó que los algoritmos implementados lograron encontrar la intersección de las listas de posteo, se encontró que el número de comparaciones necesarias aumenta con el número de listas de posteo que se intersectan.

Al comparar los algoritmos "galloping_search" y "binary_search" en función de las gráficas generadas, podemos evaluar su rendimiento en términos de tiempo de ejecución, eficiencia de

In [ ]: