Федеральное государственное автономное образовательное учреждение высшего образования «Казанский (Приволжский) федеральный университет»

Семестровая работа

по Алгоритмам и структурам данных на тему “Поиск ближайшей пары точек методом “разделяй и

влаcтвуй” (Препарата, Шеймос)”

Выполнила студентка группы

11-002 Института ИТИС

Галеева Л.Р.

Казань 2021

1. **Разделяй и властвуй** (divide and conquer) в информатике — парадигма разработки алгоритмов, заключающаяся в рекурсивном разбиении решаемой задачи на две или более подзадачи того же типа, но меньшего размера, и комбинировании их решений для получения ответа к исходной задаче; разбиения выполняются до тех пор, пока все подзадачи не окажутся элементарными.

Ранними примерами таких алгоритмов являются в первую очередь «Уменьшай и властвуй» — исходная задача последовательно разбивается на отдельные подзадачи, и в действительности может быть решена итеративно.

Бинарный поиск, алгоритм «Уменьшай и властвуй», в котором подзадачи имеют примерно половину исходного размера, имеет долгую историю. Хотя четкое описание алгоритма на компьютерах появилось еще в 1946 году в статье Джона Мочли. Идея использования отсортированного списка элементов для облегчения поиска восходит, по меньшей мере, к Вавилонии в 200 году до нашей эры. Еще один древний алгоритм уменьшай и властвуй - это алгоритм Евклида для вычисления наибольшего общего делителя из двух чисел путем уменьшения чисел до меньших и меньших эквивалентных подзадач, который датируется несколькими веками до нашей эры.

Ранний пример алгоритма «Разделяй и властвуй» с несколькими подзадачами — Гауссовое (1805) описание того, что сейчас называется быстрое преобразование Фурье Кули-Тьюки.

Ранний алгоритм «Разделяй и властвуй» из двух подзадач, который был специально разработан для компьютеров и должным образом проанализирован, является алгоритмом сортировки слиянием, изобретенный Джоном фон Нейманом в 1945 году.

1. Многие алгоритмы по природе рекурсивны (гесursive algorithms): решая некоторую задачу, они вызывают самих себя для решения её подзадач. Идея метода «разделяй и властвуй» состоит как раз в этом. Сначала задача разбивается на несколько подзадач меньшего размера. Затем эти задачи решаются (с помощью рекурсивного вызова - или непосредственно, если размер достаточно мал). Наконец, их решения комбинируются и получается решение исходной задачи. Для задачи сортировки эти три этапа выглядят так. Сначала мы разбиваем массив на две половины меньшего размера. Затем мы сортируем каждую из половин отдельно. После этого нам остаётся соединить два упорядоченных массива половинного размера в один. Рекурсивное разбиение задачи на меньшие происходит до тех пор, пока размер массива не дойдёт до единицы (любой массив длины 1 можно считать упорядоченным).

1. Вышеупомянутый алгоритм делит все точки на два набора и рекурсивно вызывает два набора. После разделения он находит полосу за время O (n). Кроме того, требуется O (n) времени, чтобы разделить массив вокруг средней вертикальной линии. Наконец, находит ближайшие точки в полосе за время O (n). Таким образом, T (n) можно выразить следующим образом:  
   T (n) = 2T (n / 2) + O (n) + O (n) + O (n)  
   T (n) = 2T (n / 2) + O ( n)  
   T (n) = T (nLogn)



1. «Разделяй и властвуй» — это мощный инструмент для решения концептуально сложных задач: все, что требуется для этого, — это найти случай разбивания задачи на подзадачи, решения тривиальных случаев и объединения подзадач в исходную задачу. Парадигма «Разделяй и властвуй» часто помогает в открытии эффективных алгоритмов. Это послужило ключом, например, к быстрому методу умножения Карацубы, алгоритмам быстрой сортировки и сортировки слиянием, алгоритму Штрассена для умножения матриц и быстрых преобразований Фурье. Во всех этих примерах подход «Разделяй и властвуй» привел к улучшению асимптотической стоимости решения в самом решении. Например, если (a) базовый вариант имеет размер, ограниченный постоянной, то работа по разбиению задачи и объединению частичных решений пропорциональна размеру задачи n, и (b) существует ограниченное число p подзадач размера ~ n/p на каждом этапе, тогда эффективность алгоритма «Разделяй и властвуй» будет равна O(*n* log*pn*). Недостатки алгоритма: В рекурсивных реализациях алгоритмов «Разделяй и властвуй» необходимо убедиться, что для стека рекурсии выделено достаточно памяти, иначе выполнение может завершиться неудачей из-за переполнения стека. Алгоритмы «Разделяй и властвуй», которые эффективны по времени, часто имеют относительно небольшую глубину рекурсии. Например, алгоритм, быстрой сортировки может быть реализован таким образом, что он никогда не требует больше, чем log2 n вложенных рекурсивных вызовов для сортировки n элементов.

Для некоторых задач разветвленная рекурсия может привести к многократной оценке одной и той же подзадачи. В таких случаях, возможно, стоит определить и сохранить решения этих перекрывающихся подзадач.

5. Источники:

1. Кормен. Алгоритмы - построение и анализ.
2. Седжвик. Фундаментальные алгоритмы на C++
3. Препарата, Шеймос. Вычислительная геометрия: введение
4. <https://en.wikipedia.org/wiki/Divide-and-conquer_algorithm>

6. Код программы:

#include <iostream>  
#include<fstream>  
#include <chrono>  
#include <float.h>  
#include <stdlib.h>  
#include <math.h>  
using namespace std;  
using namespace std::chrono;  
  
struct Point  
{  
 int x, y;  
};  
  
int compareX(const void\* a, const void\* b)  
{  
 Point \*p1 = (Point \*)a, \*p2 = (Point \*)b;  
 return (p1->x - p2->x);  
}  
  
int compareY(const void\* a, const void\* b)  
{  
 Point \*p1 = (Point \*)a, \*p2 = (Point \*)b;  
 return (p1->y - p2->y);  
}  
  
  
float dist(Point p1, Point p2)  
{  
 return sqrt( (p1.x - p2.x)\*(p1.x - p2.x) +  
 (p1.y - p2.y)\*(p1.y - p2.y)  
 );  
}  
  
float bruteForce(Point P[], int n)  
{  
 float min = FLT\_MAX;  
 for (int i = 0; i < n; ++i)  
 for (int j = i+1; j < n; ++j)  
 if (dist(P[i], P[j]) < min)  
 min = dist(P[i], P[j]);  
 return min;  
}  
  
float min(float x, float y)  
{  
 return (x < y)? x : y;  
}  
  
  
float stripClosest(Point strip[], int size, float d)  
{  
 float min = d; // Initialize the minimum distance as d  
  
 for (int i = 0; i < size; ++i)  
 for (int j = i+1; j < size && (strip[j].y - strip[i].y) < min; ++j)  
 if (dist(strip[i],strip[j]) < min)  
 min = dist(strip[i], strip[j]);  
  
 return min;  
}  
  
float closestUtil(Point Px[], Point Py[], int n)  
{  
 // If there are 2 or 3 points, then use brute force  
 if (n <= 3)  
 return bruteForce(Px, n);  
  
 // Find the middle point  
 int mid = n/2;  
 Point midPoint = Px[mid];  
  
  
 Point Pyl[mid+1]; // y sorted points on left of vertical line  
 Point Pyr[n-mid-1]; // y sorted points on right of vertical line  
 int li = 0, ri = 0; // indexes of left and right subarrays  
 for (int i = 0; i < n; i++)  
 {  
 if (Py[i].x <= midPoint.x)  
 Pyl[li++] = Py[i];  
 else  
 Pyr[ri++] = Py[i];  
 }  
  
 float dl = closestUtil(Px, Pyl, mid+1 );  
 float dr = closestUtil(Px+mid+1, Pyr, n-mid-1);  
  
 // Find the smaller of two distances  
 float d = min(dl, dr);  
  
 Point strip[n];  
 int j = 0;  
 for (int i = 0; i < n; i++)  
 if (abs(Py[i].x - midPoint.x) < d)  
 strip[j] = Py[i], j++;  
  
 return min(d, stripClosest(strip, j, d) );  
}  
  
float closest(Point P[], int n)  
{  
 Point Px[n];  
 Point Py[n];  
 for (int i = 0; i < n; i++)  
 {  
 Px[i] = P[i];  
 Py[i] = P[i];  
 }  
  
 qsort(Px, n, sizeof(Point), compareX);  
 qsort(Py, n, sizeof(Point), compareY);  
  
 // Use recursive function closestUtil() to find the smallest distance  
 return closestUtil(Px, Py, n);  
}  
  
// Driver program to test above functions  
int main()  
{  
  
 ifstream file;  
 file.open("C:\\Users\\user\\IdeaProjects\\untitled3\\output2.txt");  
  
 ofstream res;  
 res.open("C:\\Users\\user\\CLionProjects\\untitled5\\res.txt");  
 for (int i = 0; i < 50; ++i) {  
 int k;  
 file >> k;  
 Point P[k];  
 for (int i = 0; i < k; i++) {  
 file >> P[i].x;  
 file >> P[i].y;  
 }  
 auto start = steady\_clock::now();  
 int n = sizeof(P) / sizeof(P[0]);  
 closest(P, n);  
 bruteForce(P, n);  
 auto stop = steady\_clock::now();  
 auto duration = duration\_cast<microseconds>(stop - start);  
 res << k << "\t" << duration.count() << endl;  
 }  
 res.close();  
 file.close();  
 return 0;  
}

7. Входные данные: <https://github.com/galeisan/homework-aisd/blob/main/sem2/output2.txt>