Zadanie 2.

Paweł Jasiak – 308313

12 czerwca 2020

1 Zadanie

Ułóż algorytm rozwiązujący problem znajdowania najbliżej położonej pary punktów na płaszczyźnie oparty na następującej idei. Niech d będzie odległością pomiędzy parą najbliżej położonych punktów spośród punktów $p_1, p_2, \ldots, p_{i-1}$. Sprawdzamy, czy p_i leży w odległości mniejszej niż d, od któregoś z poprzednich punktów. W tym celu dzielimy płaszczyznę na odpowiednio małe kwadraty, tak by w każdym z nich znajdował się nie więcej niż jeden punkt. Te „zajęte" kwadraty pamiętamy w słowniku.
Twój algorytm powinien działać w oczekiwanym czasie liniowym. Jeśli nie potrafisz zbudować algorytmu opartego na powyższej idei, możesz opracować algorytm oparty na innej (ale spełniający te same wymagania czasowe).

2 Idea

Załóżmy, że dla każdej pary (p_i, p_j) prawdopodobieństwo, że jest ona parą o najmniejszej odległości jest takie samo. W przypadku złośliwego adwersarza możemy losowo przepermutować ciąg punktów.

Idea jest taka, że w każdym kroku mamy wyznaczone d będące aktualnie najmniejszą odległością pomiędzy parą punktów oraz płaszczyznę podzieloną na kwadraty o boku długości $\frac{d}{2}$. W każdym kwadracie będzie znajdował się najwyżej jeden punkt. Kwadrat będzie istotny, tylko jeśli znajduje się w nim punkt. Wszystkie istotne kwadraty będziemy pamiętać w tablicy haszującej, pamiętając jakie punkty są z nimi skojarzone. Kiedy będziemy rozpatrywać kolejny punkt sprawdzimy wszystkie kwadraty, które znajdują się dookoła kwadratu w którym znajduje się nowy punkt. Jeśli ten punkt stworzy krótszy odcinek, to będzie on krótszy niż d. W takim razie drugi punkt musi znajdować się w sąsiedztwie pierwszej lub drugiej warstwy kwadratów okalających ten kwadrat. Aby uniknąć niejasności załączam obrazek, 0 oznacza kwadrat z nowym punktem, 1 pierwsza warstwę, a 2 drugą warstwę.

2	**2**	**2**	**2**	**2**
2	1	1	1	2
2	1	0	1	2
2	1	1	1	2
2	2	2	2	2

Kwadratów tych jest zaledwie 25. Będziemy mieli dwa przypadki

- nie znaleźliśmy krótszego odcinka, wtedy nie dzieje się nic ciekawego, jedynie umieszczamy nasz punkt w odpowiednim kwadracie
- znaleźliśmy nowy odcinek, a więc mamy nową wartość d zgodnie z którą przehaszowujemy płaszczyznę

3 Algorytm

Niech d oznacza nasz wynik, p_i i–ty punkt, a H tablicę haszującą.

Algorithm 1 Najkrótszy Odcinek

$d \leftarrow distance(p_0, p_1)$
$rehash(H, d)$
$insert(H, p_0)$
$insert(H, p_1)$
for i from 2 to $n-1$ **do**
  $tmp \leftarrow d$
  for x in $neighbors(H, p_i)$ **do**
    $d = min(distance(x, p_i), d)$
  end for
  if $tmp \neq d$ **then**
    $rehash(H, d)$
  end if
  $insert(H, p_i)$
end for

Na koniec działania, nasz wynik będzie znajdował się w d.

4 Poprawność

Chcemy utrzymywać niezmiennik, że po i–tym kroku mamy poprawnie policzoną wartość d dla zbioru punktów do p_i. Dla pierwszych dwóch punktów jest to oczywiste. Zastanówmy się nad $(i+1)$–krokiem. Jeśli punkt p_{i+1} tworzy mniejszą odległość z jakimś p_j, gdzie $j < i+1$ to taki punkt musi być w odległości mniejszej niż d (stare) od p_{i+1}, ale wszystkie takie punkty muszą znajdować się we wskazanym sąsiedztwie.

5 Złożoność

Zastanówmy się jaki jest czas przetworzenia w i–tym kroku. Mamy $i(i-1)$ wszystkich par, z czego $i-1$ to nowe pary, po przyjściu i–tego punktu. Skoro założyliśmy równe prawdopodobieństwa, to szansa na to, że otrzymamy nowe d wynosi $\frac{i-1}{i(i-1)} = \frac{1}{i}$. Operacja *rehash* jest liniowa względem rozmiaru tablicy haszującej. Operacja *insert* wykonywana w oczekiwanym czasie stałym. W obu przypadkach dodamy nowy element do tablicy oraz porównać go ze wszystkimi potencjalnymi sąsiadami (jest ich 25), co jest wykonywalne w czasie stałym. Otrzymamy więc oczekiwany czas $O(1) + \frac{1}{i}O(i)$ w i–tym kroku. Ostatecznie mamy $\sum_i (O(1) + \frac{1}{i}O(i)) = O(n)$.