Mencari Banyak Posisi yang dapat Dijangkau sebuah Lingkaran dengan Dynamic Programming

Jehian Norman Saviero - 13515139
Program Sarjana Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha, No. 10, Bandung 40132, Indonesia
jehiannormansaviero@students.itbac.id

***Abstract*—Terinspirasi dari permasalahan mencari titik yang dapat di-*cover* oleh lingkaran berjari-jari *r*, banyak sekali permasaalahan tersebut yang dapat dikatikan dengan kehidupan sekarang ini seperti yang kita rasakan adalah aplikasi transportasi *online*. Kemudian tidak hanya itu, algoritma ini dapat digunakan untuk memetakan posisi tertentu apakah *strategis* untuk dibangun sebuah *mall* dengan parameter-parameter tertentu. Pada makalah ini, akan dibahas menyelesaikan permasalahan tersebut dengan memodelkannya dengan sederhana serta menjabarkan prosesnya secara matematika dan *dynamic programming*. Lalu dibuktikan pula algoritma tersebut mangkus dibandingkan dengan pendekatan *brute-force*.**

Keywords—dynamic programming, geometri, point, radius

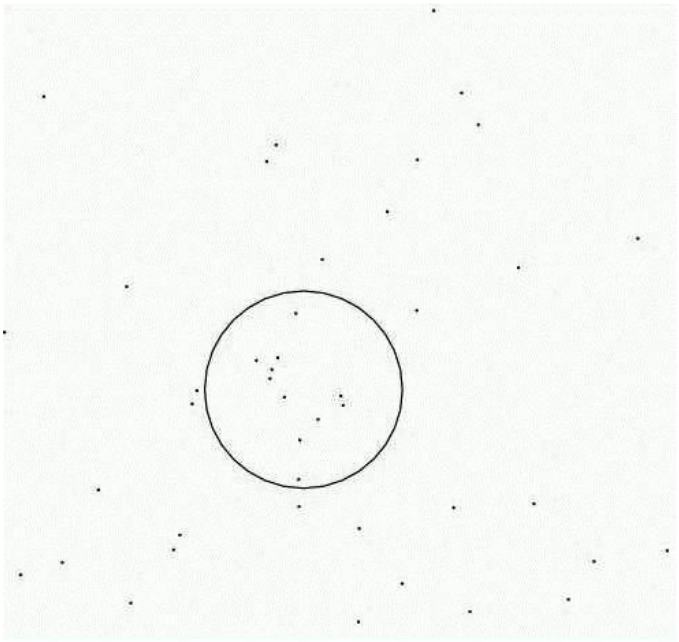
I. PENDAHULUAN

Di era saat ini, tidak jarang kita menemukan sebuah permasalahan yang dikaitkan dengan posisi suatu benda, seperti posisi suatu lokasi, posisi teman saat ini, hingga penentuan posisi kita saat ini.

Menentukan banyak poisisi yang dapat dijangkau banyak sekali manfaatnya dalam kehidupan ini. Contoh pengerapan yang membutuhkan posisi dan radius adalah pengambilan keputusan untuk membbangun sebuah menara, membangun *mall*, membangun tempat wisata, bahkan dalam menentukan *driver* tranportasi *online* yang feasible untuk dijangkau *customer* dalam waktu yang singkat.

Sebenarnya, algoritma ini dapat diselesaikan dengan menggunakan *brute force* dengan kompleksitas waktu $O(N^3)$. Tetapi, jika kita mengacu dengan perbandingan 1 detik $\sim 10^7$proses, maka kita hanya dapat memproses paling banyak 220 titik dalam waktu satu detik.

Diperlukan algoritma mangkus yang lebih *efisien* terhadap hal tersebut. Setelah ditelusuri, *dynamic programming* dan bantuan analisis geometri, kita dapat memotong kompleksitas algoritma tersebut hingga $O(N^2 \times \log N)$ yang artinya kita dapat memproses setidaknya 2000 titik dalam satu detik dalam hal ini 10 kali lebih cepat dibanding sebelumnya.



Gambar 1. Permasalahan mencari yang dapat dicover sebuah lingkaran berjari-jari *r*.

II. DASAR TEORI

A. Algoritma Brute-Force

Algoritma *brute-force* merupakan algoritma yang mempunyai prinsip pasti ditemukan tanpa memilah solusi yang seharusnya tidak perlu ditelusuri. Algoritma ini termasuk algoritma yang memakan waktu cukup bahkan sangat lama dibanding algoritma lain untuk menyelesaikan suatu permasalahan. Algoritma ini dipakai ketika tidak ada solusi mangkus yang dapat ditemukan. [1]

Pada kali ini penyelesaian masalah ini dengan *brute-force* diselesaikan dengan algoritma berikut

```
43
44  Point p[maxn];
45
46  inline int solve(Point a, Point b){
47      Point mid = Point((a.x + b.x) / 2, (a.y + b.y) / 2);
48      double angle = atan2(a.y - b.y, a.x - b.x) + acos(-1.0)/2;
49      double len = sqrt(1 - dis_pp(a,b)/ 4);
50      mid.x += cos(angle) * len;
51      mid.y += sin(angle) * len;
52      int ret = 0;
53      for (int i = 0; i < n; i ++){
54          if (dis_pp(mid,p[i]) <= 1 + EPS) ++ret;
55      }
56      return ret;
57  }
58
59  int main(){
60      long double t0, t1;
61      t0 = clock()*1000/CLOCKS_PER_SEC;
62      while (~scanf("%d",&n) && n){
63          for (int i = 0; i < n; i ++){
64              scanf("%lf%lf", &p[i].x, &p[i].y);
65          }
66          sort(p, p + n, cmp);
67          int ans = 1;
68          for (int i = 0; i < n; i ++){
69              for (int j = i + 1; j < n && (p[j].x - p[i].x <= 2); j ++){
70                  if (dis_pp(p[i],p[j]) <= 4 + EPS){
71                      ans = max(ans, max(solve(p[i],p[j]),solve(p[j],p[i])));
72                  }
73              }
74          }
75          printf("%dn",ans);
76      }
77      t1 = clock()*1000/CLOCKS_PER_SEC;
78      printf("Time elapsed: %.3Lf milisecond(s)\n", t1-t0);
79      return 0;
80  }
```

Gambar 2. Implentasi *brute force*

B. *Geometri Lingkaran dan Titik*

Titik merupakan noktah yang mempunyai posisi. Apabila terdapat dua titik yang memiliki posisi yang berbeda, maka dapat terbentuk sebuah garis lurus sederhana.

Secara umum, garis yang sering digunakan untuk memecahkan masalah geometri meruapakan garis lurus sederhana. Garis lurus sederhana mempunyai atribut antara lain kemiringan, serta posisi dari tiap-tiap titiknya, Garis lurus sederhana dapat digunakan mencari persamaan garis lainnya yang mempunyai sifat sejajar, tegak lurus, maupun hanya berpotongan dengan dirinya.

Selain garis lurus sederhana, bentuk lingkaran kerap digunakan untuk menentukan area yang dapat dilingkupi oleh lingkaran tersebut dengan jari-jari tertentu dari posisi tertentu. Lingkaran memiliki atribut titik pusat dan jari-jari lingkaran. Secara Aljabar, bentuk umum persamaan garis adalah

$$y = mx + C$$

dimana m meruapakan kemiringan garis tersebut dan sebuah kontanta C yang merupakan besarnya pergeseran titik pada garis tersebut pada saat memotong sumbu-X diukur dari titik $(0,0)$.

Apabila diberikan posisi dua titik sebut saja (x_1, y_1) dan (x_2, y_2), secara umum persamaan garis tersebut adalah

$$\frac{y - y_1}{y_2 - y_1} = \frac{x - x_1}{x_2 - x_1}$$

dan kemiringan persamaan garisnya adalah:

$$\frac{y_2 - y_1}{x_2 - x_1}$$

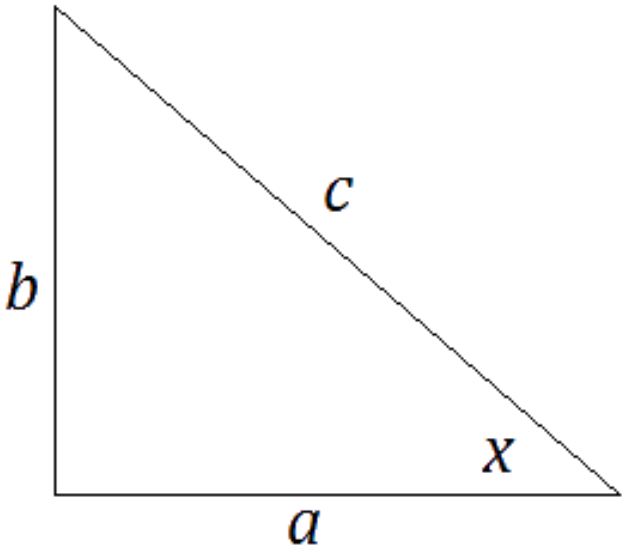
Sebuah titik memiliki hubungan dengan sebuah lingkaran yang terdiri dari *point incircle*, titik yang berada di dalam lingkaran tersebut, kemudian *point circumference*, titk yang berada pada keliling lingkaran tersebut, serta *point excircle* yang merupakan titik yang berada di posisi luar lingkaran tersebut. Untuk persamaan lingkaran sendiri sebuah lingkaran berjar-jari r dan memiliki titik pusat (x_1, y_1) memiliki bentuk aljabar dengan acuan diagram karterius adalah sebagai berikut

$$(x - x_1)^2 + (y - y_1)^2 = r^2$$

Tidak hanya itu, lingkaran juga memiliki hubungan dengan sebuah garis lurus sederhana. Hubungan tersebut terdiri dari berpotongan, menyinggung, atau tidak keduanya. Sebuah garis dikatangan berpotongan apabila terdapat dua buah titik yang berada terletak pada garis tersebut sekaligus terletak pada keliling lingkaran tersebut. Sebuah garis lurus sederhana dikatakan menyinggung lingkaran apabila tepat memiliki sebuah titik bersama yang berada garis lurus sederhana tersebut serta berada pada lingkaran tersebut. Kemudian, sebuah garis lurus sederhana tidak termasuk pada keduanya apabila tidak terdapat titik yang memenuhi kondisi ia berada pada garis lurus tersebut maupun pada lingkaran tersebut.

C. *Trigonometri*

Untuk mempelajari hubungan tersebut, diperlukan ilmu trgionometri yang mencakup persamaan garis dan persamaan lingkaran tersebut. Dasar dari trigonometri yang perlu diketahui adalah sebagai berikut:



Gambar 3, Segitiga Trigonometri

dimana:

$$\sin x = \frac{b}{c}$$

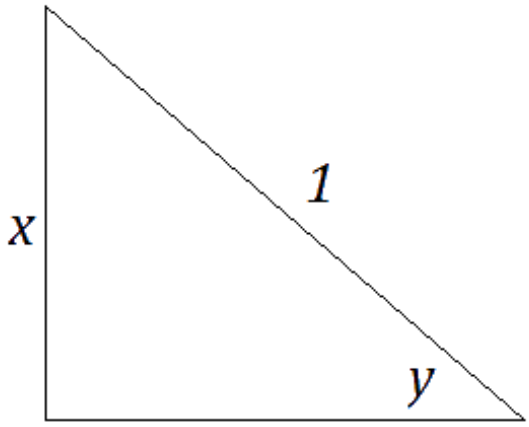
$$\cos x = \frac{a}{c}$$

$$\tan x = \frac{b}{a}$$

dan berlaku:

$$\sin^2 x + \cos^2 x = 1$$

Kemudian untuk mencari besar sudut x digunakan persamaan *arc* dimana persamaan tersebut adalah:



Gambar 4. Segitiga Trigonometri dengan sisi

$$\arcsin x = y$$

$$\arccos \sqrt{1 - x^2} = y$$

$$\arctan \frac{\sqrt{1 - x^2}}{x} = y$$

Kemudian, untuk $arctan$ sendiri mempunyai ketentuan

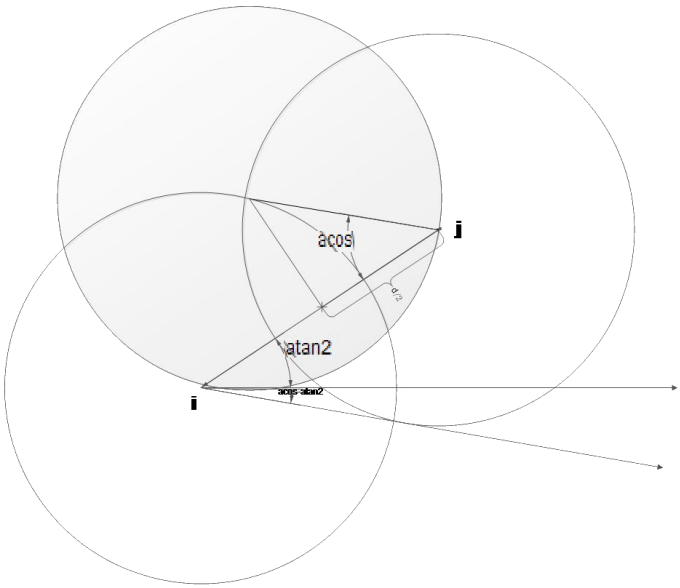
$$\text{atan2}(y, x) = \begin{cases} \arctan\left(\frac{y}{x}\right) & x > 0 \\ \arctan\left(\frac{y}{x}\right) + \pi & y \geq 0, x < 0 \\ \arctan\left(\frac{y}{x}\right) - \pi & y < 0, x < 0 \\ +\frac{\pi}{2} & y > 0, x = 0 \\ -\frac{\pi}{2} & y < 0, x = 0 \\ \text{undefined} & y = 0, x = 0 \end{cases}$$

III. Analisis Masalah

Setelah memiliki *basic-basic* di atas, kita langsung ke permasalahan untuk menyelesaikan mencari banyak titik yang dapat di-*cover* oleh sebuah lingkaran berjari-jari *r*.

A. Pemilihan titik

Karena data yang diberikan adalah banyak titik, kumpulan titik serta jari-jari lingkaran yang ingin meng-*cover* kumpulan titik, maka kita perlu patokan titik. Bila kita lihat secara geometri misal titik acuan adalah i dan titik yang akan dibandingkan adalah titik j, perhatikan sketsa berikut:



Gambar 5. Penggambaran masalah

Dari sini kita daftarkan jangkauan lingkaran dengan pusat baru yang dimana titik pusat tersebut dapat diambil. Misal jarak $i-j$ adalah d, sehingga $i-j$ merupakan talibusur untuk lingkaran baru yang dapat menjangkau menjangkau kedua titik tersebut.

Tarik sebuah garis apotema dari titik pusat lingkaran teresbut ke talibusur. Kita tau bahwa dari titik j maupun titik i ke lingkaran tersebut berjarak r, maka besarnya sudut yang dibentuk oleh jari-jari lingkaran baru dengan talibusur adalah

$$\operatorname{acos}\frac{d}{2r}$$

yang akan kita misalkan dengan θ.

Perhatikan kemiringan sebuah garis yang dibentuk oleh titik i dengan titik j. Besar sudut yang dibuat oleh dua buah titik tersebut dengan sumbu-X adalah

$$\operatorname{atan}\left(\frac{y_i - y_j}{x_i - x_j}\right)$$

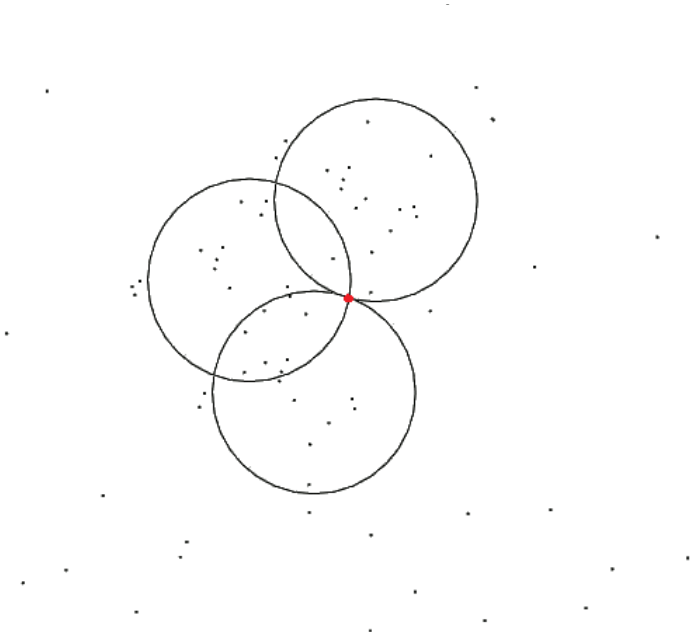
yang akan kita misalkan dengan ϕ.

Tinjau bahwa lingkaran tersebut dapat kira rotasi dengan pusat i dimulai dari $\theta - \phi$ hingga $\theta + \phi$ agar dapat menjangkau kedua titik tersebut. Dengan cara yang sama, kita daftarkan semua titik pasangan (P_i, P_j) dimana $i \neq j$ dengan atribut awal sudut yang dapat menjangkau keduanya dan akhir sudut yang dapat menjangkau keduanya.

B. Pengolahan Data

Dengan bantuan dari *dynamic programming*, kita dapat mendaftarkan pasangan titik-titik tersebut dengan menyimpan id, $boolean$ telah diambil, $boolean$ telah diletakkan, dan $counter$ (penghitung), serta besar sudut (baik itu $\theta + \phi$ maupun $\theta - \phi$).

Pertama-tama, kita $sort$ data-data yang telah kita punya dengan acuan $sort$ adalah besar sudut dengan kompleksitas *sorting* terbaik adalah $O(N \times \log N)$. Alasan menggunakan besar sudut adalah kita seolah-olah akan memutar lingkaran tresebut dengan skema seperti berikut.



Gambar 6. Teknik Rotasi[2]

Bila dimodelkan dengan sebuah permasalahan, bisa saya modelkan sebagai berikut. Misal kita mempunyai sejumlah $2n$-botol yang dinomori $1-n$ sehingga ada tepat dua botol yang memiliki nomor yang sama dan n bendera. Dijamin bahwa botol dengan nomor k untuk k berapapun selalu ditemukan bendera dengan nomor k terlebih dahulu lalu kemudian botol lain yang bernomor k tidak ada bendera. Diberikan aturan main sebagai berikut:

a. Ambil bendera bila ditermukan bendera pada botol
b. Letakkan bendera bernomor-k apabila botol bernomor-k kosong.

Masalah yang muncul dari permainan ini adalah, pada saat bermain, berapa yang yang dipegang tangan paling banyak dalam waktu bersamaan.

Untuk menyelesaikan masalah ini, misal kita punya variabel bendera awal $maks$ adalah 1 (karena sudah jelas kita pasti memegang bendera di awal). Dari sini kita cukup melakukan penambahan $variabel\ counter$ yang diset menjadi 1. Setiap kita mendapatkan *update counter* menjadi *counter*+1. Kemudian update $maks$ didalam *looping* dengan algoritma `maks = max(maks, counter)` sehingga dijamin bahwa nilai `maks` akan selalu *update*.

Setelah dikalkulasi, kompleksitas total adalah $O\big(N \times (N \times \log N + N)\big) = O(N^2 \times \log N)$ seperti yang dijanjikan di awal.

C. *Implementasi Algoritma*

Untuk *input* dan *output* untuk kasus ini, menggunakan format awal banyaknya titik dan jari-jari r kemudian koordinat titik-titik. Lalu *output* dari program ini adalah banyak titik yang dapat *dicover* sebuah lingkaran dengan jari-jari r.

```cpp
#include <bits/stdc++.h>
#define jehian using
#define mau namespace
#define libur std
#define MAX_N 1000 + 100

jehian mau libur;
typedef double p_type;
struct Point{
    p_type x, y;

    Point(){}

    Point(p_type x, p_type y) : x(x), y(y){}
} ps[MAX_N];

struct PolarAngle{
    p_type angle;
    bool flag;
    const bool operator<(const PolarAngle &other){
        return angle < other.angle;
    }
} as[MAX_N];

inline p_type distance_of(const Point &P, const Point &Q){
    return sqrt((P.x - Q.x) * (P.x - Q.x) + (P.y - Q.y) * (P.y - Q.y));
}

int solve(const int& n, const p_type& r){
    int result = 1;
    for (int i = 0; i < n; ++i){
        int m = 0;
        long double d;
        for (int j = 0; j < n; ++j){
            if (i != j && (d = distance_of(ps[i], ps[j])) <= r){
                long double phi = acos(d / 2);
                long double theta = atan2(ps[j].y - ps[i].y, ps[j].x - ps[i].x)
                as[m].angle = theta - phi, as[m++].flag = 1;
                as[m].angle = theta + phi, as[m++].flag = 0;
            }
        }
        sort(as, as + m);
        for (int sum = 1, j = 0; j < m; ++j){
            if (as[j].flag){
                ++sum;
            } else {
                --sum;
            }
            result = max(result, sum);
        }
    }
    return result;
}

int main(int argc, char *argv[]){
    int N;
    long double r;
    while (scanf("%d %d", &N, &r), N, r){
        for (int i = 0; i < N; ++i){
            scanf("%lf%lf", &ps[i].x, &ps[i].y);
        }
```

Gambar 7. Implementasi *dynamic programming*.

IV. KESIMPULAN DAN SARAN

A. *Hasil Data*

Setelah dibandingkan dengan menggunakan *testcase* tertentu, berikut waktu eksekusi *output* untuk tiap-tiap *testcase*

TABLE I. TABEL PERBANDINGAN

No.	Besar *Test Case*	*Brute Force*	Perbaikan Algoritma
1	10	0 ms	0 ms
2	25	2 ms	0 ms
3	100	42 ms	7 ms
4	250	187 ms	59 ms
5	1000	Seg-Fault	149 ms

B. *Kesimpulan*

Berdasarkan data yang diperoleh, terbukti bahwa perbaikan algoritma dengan *dynamic programming* lebih mangkus dibandingkan dengan *brute-force*. Perbedaannya cukup *significant* hampir mencapai 10 kali sesuai dengan kompleksitasnya.

C. *Saran*

Setelah dilihat, kita dapat membuat algoritma tersebut dengan memotong waktu kompleksitasnya. Tapi bila kita ingin mengambil solusinya kita cukup mengedit sedikit solusinya dengan menambahkan tipe data *stack* yang berfungsi untuk memasukkan dan meneluarkan *id point* yang dipilih. Bila hasil pengambilan bendera sudah maksimum, maka *stack* tidak perlu di *pop* dan disimpan hingga akhir program untuk dicetak titik-titiknya

V. PENUTUP

Ucapan terima kasih penulis berikan kepada Tuhan Yang Maha Esa, Institut Teknologi Bandung dan kedua dosen mata kuliah IF2211-Strategi Algoritma yaitu Bapak Rinaldi Munir Ibu Ulfa Maulidevi, dan Ibu Masayu Leylia Khodra. Diharapkan makalah ini dapat memberi manfaat bagi pembaca dan menginspirasi untuk dibuat pengembangannya yang lebih lanjut.

DAFTAR PUSTAKA

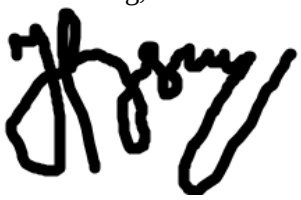
[1] Munir, Rinaldi. 2009. Diktat Kuliah Strategi Algoritma. Bandung : Program Studi Teknik Informatika Institut Teknologi Bandung.

[2] *Peking University Online Judge* http://poj.org/problem?id=1981. Diakses pada 19 Mei 2017

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 19 Mei 2017



Jehian Norman Saviero
(13515139)