DESAIN DAN ANALISIS ALGORITMA
ALGORITMA BRUTE FORCE

Dosen Pengampu :
Dr. Dra. Luh Gede Astuti,M.Kom.

Disusun Oleh :
Aditya Chandra Nugraha 2308561092

PROGRAM STUDI INFORMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS UDAYANA
2024

1. Match String

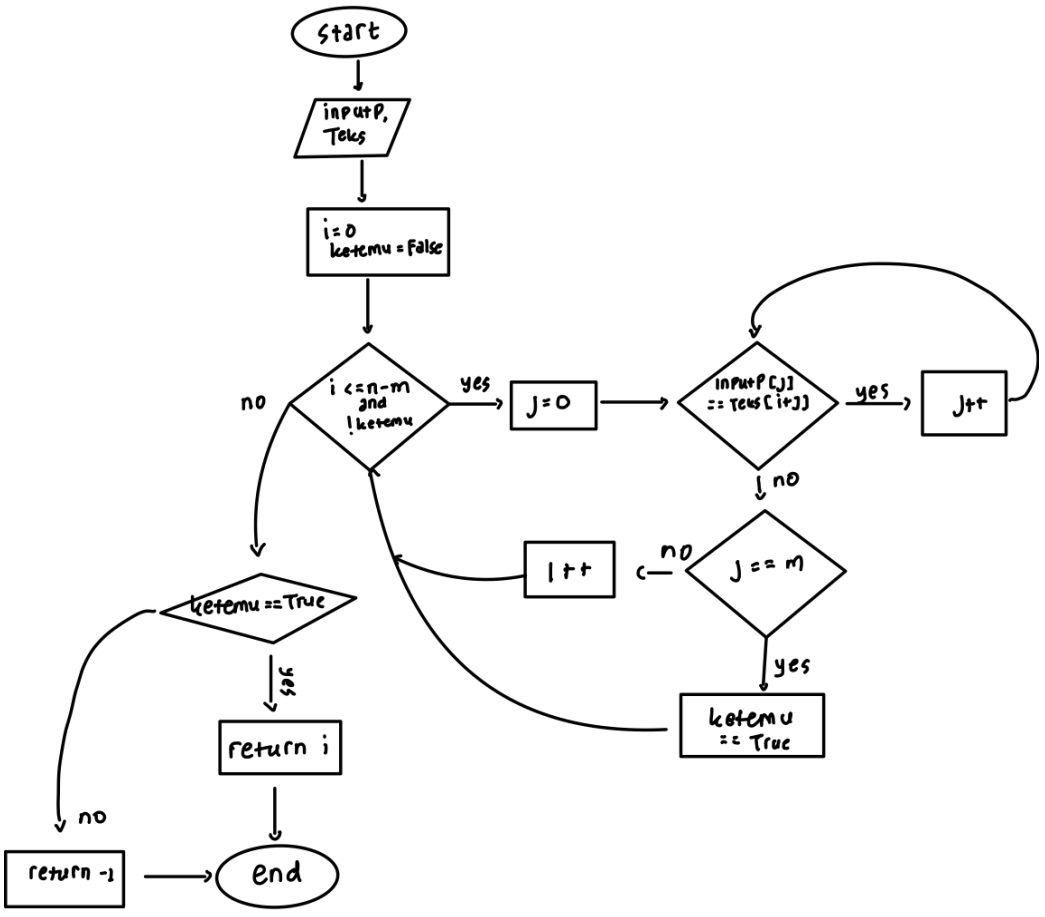
1.1 Pseudocode

```
procedure PencocokanString(input P : string, T : string,
                           n, m : integer,
                           output idx : integer)
{ Masukan: pattern P yang panjangnya m dan teks T yang
panjangnya n. Teks T direpresentasika sebagai string
(array of character)
Keluaran: lokasi awal kecocokan (idx)
}
Deklarasi
   i : integer
   ketemu : boolean

Algoritma:
   i←0
   ketemu←false
   while (i ≤ n-m) and (not ketemu) do
      j←1
      while (j ≤ m) and (Pj = Ti+j ) do
         j←j+1
      endwhile
      { j > m or Pj ≠ Ti+j }

      if j = m then    { kecocokan string ditemukan }
         ketemu←true
      else
         i←i+1  {geser pattern satu karakter ke kanan teks }
      endif
   endfor
   { i > n - m or ketemu }
   if ketemu then
      idx←i+1
   else
      idx←-1
   endif
```

1.2 Flowchart



1.3 Implementasi Algoritma (Python)

Contoh soal untuk digunakan:
Teks (Teks) : “aditya chandra” (Panjang 21 karakter).
Pattern (inputP) : chan (Panjang 4 karakter).

```
def stringMatching(inputP, Teks):
    # inputP is the pattern to find in Teks
    m = len(inputP)   # Length of the pattern
    n = len(Teks)     # Length of the text
    i = 0             # Starting index for the text search
    ketemu = False    # A flag to indicate if a match is found

    while (i <= n - m) and not ketemu:
        j = 0
        while (j < m) and (inputP[j] == Teks[i + j]):
            j += 1
        if j == m:
            ketemu = True
        else:
            i += 1

    if ketemu:
        return i
    else:
        return -1
strInput = "aditya chandra"
strDicari = "chan"
print(strDicari ,"ditemukan dari index
ke-",stringMatching(strDicari,strInput))
```

output

1.4 Tracing

Iterasi	Indeks	Potongan Teks	pencocokan inputP[j] == Teks[i + j]
1	i=0	“adit”	Tidak cocok inputP[0]!=Teks[0]
2	i=1	“dity”	Tidak cocok inputP[0]!=Teks[1]
3	i=2	“itya”	Tidak cocok inputP[0]!=Teks[2]
4	i=3	“tya ”	Tidak cocok inputP[0]!=Teks[3]

5	**i=4**	**“ya c”**	**Tidak cocok inputP[0]!=Teks[4]**
6	**i=5**	**“a ch”**	**Tidak cocok inputP[0]!=Teks[5]**
7	**i=6**	**“ cha”**	**Tidak cocok inputP[0]!=Teks[6]**
8	**i=7**	**“chan”**	**Cocok inputP[0]==Teks[7]**

1.5 Analisis kompleksitas algoritma

1. Kompleksitas Kasus Terburuk
Algoritma harus membandingkan setiap karakter dalam pattern inputP dengan setiap substring dari Teks sebanyak n - m + 1 kali. Dikarenakan pattern tidak ditemukan,atau hanya ditemukan di bagian akhir teks. Oleh karena itu, O(n*m) adalah notasi kompleksitas waktu dalam kasus terburuk yang tepat.

2. Kompleksitas dalam Kasus Terbaik,
Dalam kasus terbaik, pattern P ditemukan di awal teks, dan inner loop berhenti setelah menemukan kecocokan penuh dalam iterasi pertama. Pada kondisi ini, outer loop berjalan hanya satu kali, dan inner loop berjalan sebanyak m kali, menghasilkan kompleksitas O(m).

2. Nearest Point

2.1

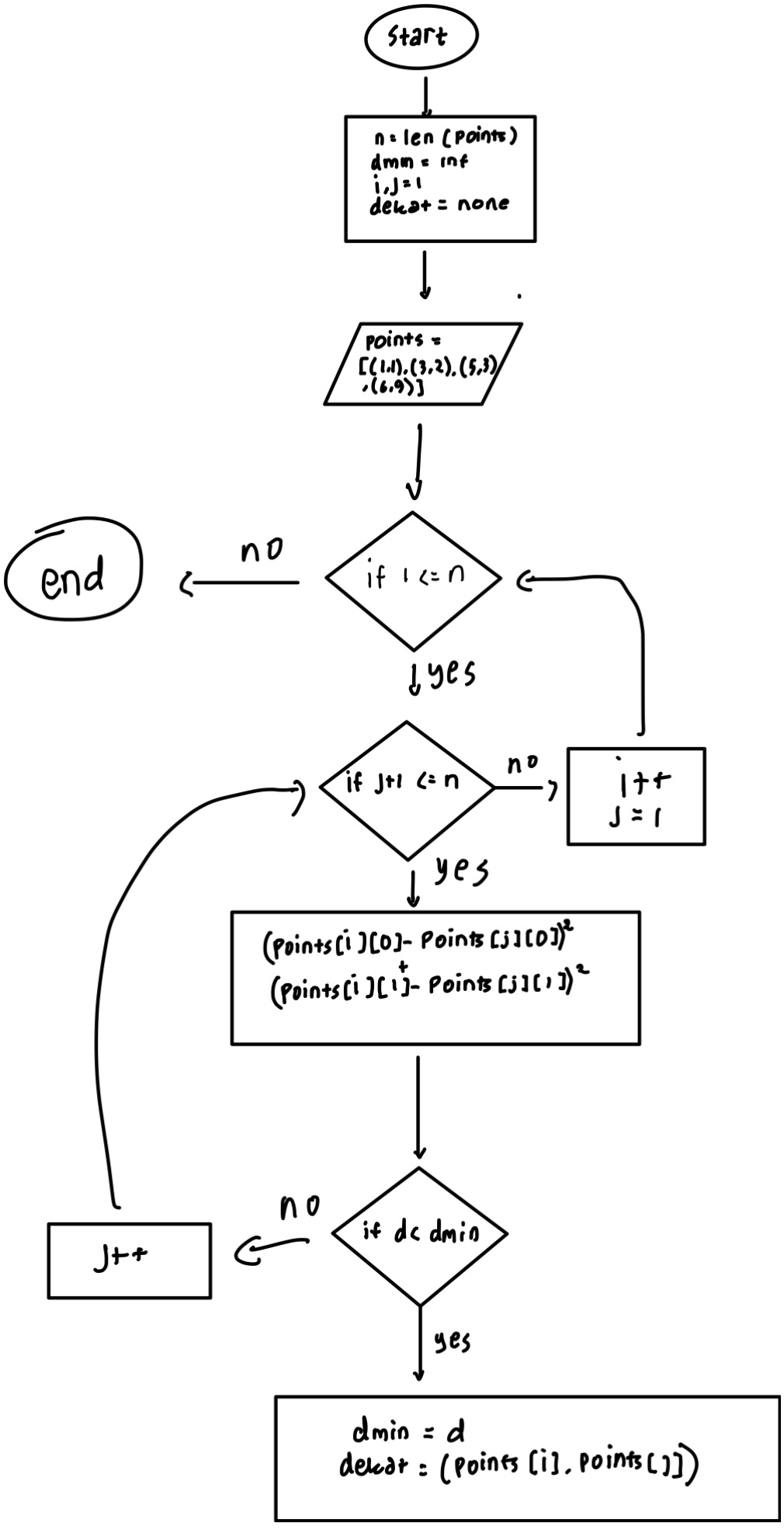
```
procedure CariDuaTitikTerdekat(input P : SetOfPoint,
                               n : integer,
                               output P1, P2 : Point)
{ Mencari dua buah titik di dalam himpunan P yang jaraknya
terdekat.
Masukan: P = himpunan titik, dengan struktur data sebagai
berikut
        type Point = record(x : real, y : real)
        type SetOfPoint = array [1..n] of Point
Keluaran: dua buah titik, P1 dan P2 yang jaraknya
terdekat.
}
Deklarasi
  d, dmin : real
  i, j : integer

Algoritma:
  dmin←9999
  for i←1 to n-1 do

    for j←i+1 to n do
      d←√((Pi.x-Pj.x)^2 + ((Pi.y-Pj.y)^2)
      if d < dmin then          { perbarui jarak terdekat }
        dmin←d
        P1←Pi
        P2←Pj
      endif
    endfor

  endfor
```

2.2 Flowchart


start
n = len (points)
dmin = inf
i, j = 1
dekat = none
points = [(1,1),(3,2),(5,3),(6,9)]
if i <= n
no
end
yes
if j+1 <= n
no
i++
j = i
yes
(points[i][0] - points[j][0])^2
(points[i][1] - points[j][1])^2
if d < dmin
no
j++
yes
dmin = d
dekat = (points[i], points[j])

2.3 Implementasi Algoritma (Python)

```
def nearestPoint(Points):
   n = len(Points)
   dmin = float('inf')
   dekat = None

   for i in range(n):
       for j in range(i +1,n):
           d = m.sqrt((Points[j][0] - Points[i][0])**2 +
(Points[j][1]-Points[i][1])**2)

           if d < dmin :
               dmin=d
               dekat = (Points[i],Points[j])

   return dekat
setPoints =[(1,1),(3,2),(5,4),(6,9)]
print(nearestPoint(setPoints))
```

output

2.4 Tracing

Iterasi i	Iterasi J	Point [i]	Point [j]	d	dmin	Dekat
1	**2**	**1,1**	**3,2**	$\sqrt{(1-3)^2+(1-2)^2}$ **√5 =2,2360**	**2,2360**	**P1 = (1,1) P2 = (3,2)**
1	**3**	**1,1**	**5,4**	$\sqrt{(1-5)^2+(1-4)^2}$ **√20=4,472**	**2,2360**	**P1 = (1,1) P2 = (3,2)**
1	**4**	**1,1**	**6,9**	$\sqrt{(1-6)^2+(1-9)^2}$ **√89=9,433**	**2,2360**	**P1 = (1,1) P2 = (3,2)**
2	**3**	**3,2**	**5,4**	$\sqrt{(3-5)^2+(2-4)^2}$ **√8=2,828**	**2,2360**	**P1 = (1,1) P2 = (3,2)**
2	**4**	**3,2**	**6,9**	$\sqrt{(3-6)^2+(2-9)^2}$ **√58=7,615**	**2,2360**	**P1 = (1,1) P2 = (3,2)**
3	**4**	**5,4**	**6,9**	$\sqrt{(5-6)^2+(4-9)^2}$ **√26=5,090**	**2,2360**	**P1 = (1,1) P2 = (3,2)**

2.5 Analisis kompleksitas algoritma

1. Kompleksitas dalam Kasus Terburuk

Karena penggunaan nested loop, jumlah perbandingan yang dilakukan oleh algoritma adalah kombinasi dari setiap dua titik dalam himpunan. Jumlah total iterasi yang dilakukan adalah sebesar:

C(n, 2) = n(n − 1)/2

Jadi, kompleksitas waktu untuk keseluruhan algoritma adalah O(n2) dalam kasus terburuk.

2. Kompleksitas dalam Kasus Terbaik,

ketika dua titik pertama sudah merupakan pasangan dengan jarak terdekat, algoritma tetap harus memeriksa semua pasangan titik yang mungkin. Maka, kompleksitas dalam kasus terbaik juga adalah O(n2), karena algoritma brute-force ini tetap melakukan semua perhitungan jarak.