Penyelesaian *Number Fill-in Puzzle* dengan Algoritma *Backtrack*

Bram Musuko Panjaitan / 13517089
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13517089@std.stei.itb.ac.id, bram.musuko@gmail.com

***Abstract*— *Number Fill-in Puzzle* adalah permainan yang terdiri dari rangkaian kotak hitam dan putih. Kotak putih berarti dapat diisi dan kotak hitam berarti tidak dapat diisi. Kemudian tersedia juga kumpulan dari angka-angka. Tujuan dari permainan ini adalah bagaimana caranya untuk memasukkan semua angka-angka tersebut ke dalam rangkaian kotak hitam putih. Strategi Algoritma (IF2211) adalah mata kuliah yang diajarkan di Teknik Informatika Institut Teknologi Bandung. Salah satu strategi yang diajarkan pada mata kuliah ini adalah backtrack(runut – balik). Dengan adanya makalah ini penulis berharap masyarakat dapat lebih gemar untuk bermain permainan *puzzle* yang sangat bersifat positif untuk perkembangan otak seseorang, sehingga dapat meningkatkan kecerdasan seseorang Pada makalah ini penulis akan menyelesaikan *Number Fill-in Puzzle* dengan menggunakan algoritma *backtrack* (runut – balik).**

Keywords – Number Fill-in Puzzle, backtrack

I. PENDAHULUAN

Pada zaman sekarang masyarakat sering sekali menggunakan waktunya untuk bermain sebuat *game*. Hal ini bukanlah hal yang baru terjadi, karena sejak zaman dulu manusia telah bermain *game*, tetapi stigma masyarakat pada *game* bersifat negatif dan memberi dampak buruk kepada kehidupan, tetapi tidak semua *game* memberi efek buruk terhadap kita. *Game* yang bersifat teka – teki atau *puzzle* sangat bagus untuk otak kita, karena kita dapat mengasah otak dan kita juga mendapat keseruan yang ditawarkan oleh permainan tersebut

Banyak sekali jenis teka – teki yang kita ketahui seperti sudoku, *Crossword*, *Fill-in Puzzle*, dan masih banyak lainnya. *Number Fill-in Puzzle* cukup populer di Indonesia, khususnya pada masyarakat generasi 1980-2000, dimana teka-teki ini sering dijual pada toko buku atau penjual majalah dan koran. Teka – teki ini sangat populer karena peraturan dari teka – teki mudah dimengerti, yaitu kita harus memasukkan semua angka yang tersedia pada kotak yang diberikan. Harga TTS ini juga tergolong murah pada kisaran 1000-2000, oleh karena itu teka-teki ini sangat digemari oleh masyarakt.

Namun seiring dengan perkembangan zaman, teknologi sudah dapat menggantikan beberapa media fisik, perubahan ini menyebabkan kurangnya minat masyarakat untuk mengkonsumsi puzzle – puzzle yang dijual di toko buku atau majalah dan koran, sehingga perlahan – lahan *Number Fill-in Puzzle* atau yang sering disebut dengan TTS angka mulai tidak dimainkan lagi oleh masyarakat Indonesia dan perlahan – lahan akan dilupakan

Number Fill-in puzzle dapat diselesaikan dengan beberapa algoritma. Salah satu algoritma yang dapat menyelesaikan teka – teki ini adalah algoritma *backtrack*. Algoritma *backtrack* digunakan karena dengan algoritma ini lebih pintar sehingga dapat memberikan solusi lebih cepat dari beberapa algortima seperti algoritma *brute force*, greedy, dan lain lain

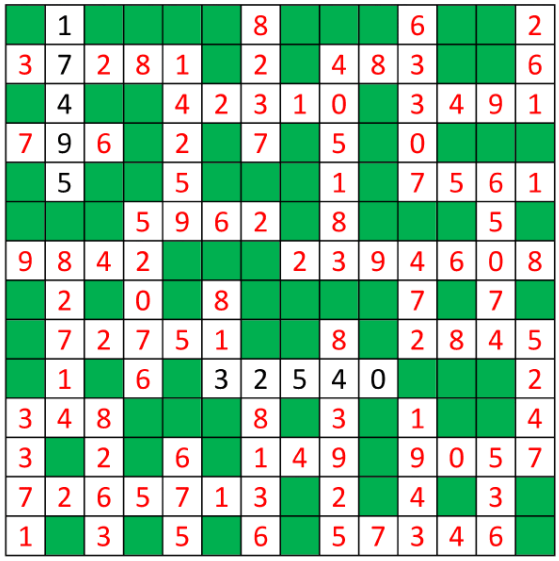
Pada Makalah ini, penulis akan coba membuat algoritma *backtrack* yang dapat menyelesaikan *Number Fill-in Puzzle*, penulis akan membuat kode solver dalam bahasa python. Penulis juga berharap dengan dibuatnya makalah ini, kesadaran masyarakat untuk bermain teka-teki dapat ditingkatkan. Karena permainan teka-teki tidak hanya memberikan sesorang efek keseruan saat bermain, tetapi juga memberikan seseorang efek positif, yaitu otak semakin terasah

II. DASAR TEORI

A. *Number Fill-in Puzzle*

Number fill-in puzzle adalah bagian dari *fill-in puzzle* yaitu kita mempunya matriks dari kotak yang dapat dibagi menjadi 2 jenis, yaitu petak yang bisa diisi dan petak yang tidak dapat diisi. Setiap matriks memiliki kombinasi dari kedua jenis petak ini yang berbeda – beda. Setelah itu akan diberikan list angka dengan jumlah digit yang berbeda – beda. Kemudian kita akan mencari cara bagaimana cara untuk setiap bagian dari list angka masuk ke dalam matriks awal kita. Kondisi menang pada teka – teki ini adalah ketika kita dapat memasukkan semua angka ke dalam matriks kotak secara tepat satu kali untuk setiap list angka. *Number fill-in puzzle* umumnya bersifat unik, yaitu hanya ada 1 jawaban untuk setiap teka- teki nya.

Dari gambar dibawah kita dapat melihat cara memainkan permainan *Number Fill-in Puzzle*, untuk angka yang berwarna hitam, angka tersebut telah ditetapkan sebelum permainan ini dimulai yang bertujuan untuk membantu pemain dalam mengerjakan teka – teki ini. Angka yang berwarna merah adalah angka yang harus diisi, disinilah tugas algoritma backtrack untuk mencari bagaimana caranya mengisi semua angka yang belum ditaruh di matriks kotak ke dalam matriks kotak secara benar



3 DIGITS		4 DIGITS		5 DIGITS		6 DIGITS	7 DIGITS
149	536	1943	7561	14259	52076	405183	2394608
261	675	2845	8237	~~17495~~	57346	843925	7265713
348	796	3371	8263	28136	63307		
472	813	3491	9057	~~32540~~	65074		
483		5247	9842	37281	72751		
		5962		42310	82714		

Gambar 1. Contoh *Number Fill-in Puzzle* https://www.math-salamanders.com/image-files/number-fill-in-puzzles-6ans.gif dengan sedikit perubahan

B. Algoritma Backtrack

Algoritma *backtrack* (runut – balik) adalah sebuah fase di dalam algoritma traversal DFS yaitu ketika menelusuri simpul – simpul pada sebuah graf. Sebagai sebuah metode pencarian solusi yang terstruktur, sistematis, dan mangkus

Algoritma *backtrack* adalah pengembangan dari algoritma *exhaustive search*, di dalam *exhaustive search* akan digenerate semua kemungkinan solusi untuk sebuah masalah. Di dalam algoritma backtrack, semua solusi tidak akan di generate, hanya pilihan yang mengarahkan kepada solusi lah yang akan di generate, sedangkan pilihan yang tidak akan mendapatkan solusi tidak akan dipertimbangkan lagi. Proses membuang pilihan yang tidak akan mencapai solusi disebut *pruning*.

Properti umum metode *backtrack* :

1. Solusi Persoalan

- Solusi dinyatakan sebagai vektor dengan *n-tuple*:

  $X = (x_1, x_{2,...,} x_n), x_i \in S_i$

- Mungkin saja $S_1 = S_2 = ... = S_n$
- Contoh : $S_i = \{0,1\}$, $x_i = 0$ atau 1

2. Fungsi pembangkit nilai x_k

- Dinyatakan sebagai predikat:

  T(k)

- T(k) membangkitkan nilai untuk xk , yang merupakan komponen vektor solusi.

3. Fungsi Pembatas

- Dinyatakan sebagai predikat B($x_1, x_{2,...,} x_k$)
- B bernilai benar jika ($x_1, x_{2,...,} x_k$) mengarah pada solusi
- Jika bernilai benar**,** maka akan dibangkitkan nilai untuk x_{k+1} dilanjutkan, tetapi jika bernilai salah, maka ($x_1, x_{2,...,} x_k$) akan dibuang

```
boolean solve(Node n) {
        if n is a goal node, return true
        foreach option O possible from n {
                if solve(O) succeeds, return true
        }
        return false
}
```

Berikut adalah *pseudocode* untuk algoritma *backtrack* yang bersifat rekursif karena akan memanggil fungsi solve() sampai ia mendapatkan solusi. Return true yang pertama adalah untuk menghentikan proses pencarian solusi rekursif dan solusi ditemukan. Return true yang kedua berfungsi untuk mencari solusi dengan kedalaman lebih tinggi, karena pilihan ini dipercaya berpotensi untuk menemukan solusi. Return false yang terakhir adalah untuk mengaktifkan *pruning* dari *backtrack*, sehingga program tidak akan menelusuri pilihan tersebut karena tidak berpotensi untuk menghasilkan solusi

III. PENYELESAIAN *NUMBER FILL-IN PUZZLE* DENGAN ALGORITMA *BACKTRACK*

Setelah membahas beberapa teori dan latar belakang dalam penyelesaian masalah ini, kita akan membahas cara menyelesaikan masalah ini dengan algoritma *backtrack*, tetapi sebelum itu berikut adalah cara merepresentasikan permainan *Number Fill-in Puzzle* dalam bahasa python:

A. Board

Board atau papan permainan yang akan digunakan dalam *Number Fill-in Puzzle* digunakan sebagai akan direpresentasikan dengan array of character, dimana setiap karakternya dapat bernilai "#", "-", dan angka. Representasi char terhadap *board* ini akan disimpan di dalam file eksternal yang akan dipanggil saat menjalankan program

- # = bagian yang tidak dapat diisi (dinding)
- - = bagian kosong yang dapat diisi dan belum diisi
- 0 – 9 = bagian yang sudah terisi angka

```
-----#-#-#
-#-#------
---#-#-#-#
-#-##-----
#----##-#-
#-#-#-----
----#-#-##
#-#---#---
#-#-#-###-
----#-----
```

Contoh file external yang menyimpan *board*

B. List Jawaban

Salah satu bagian yang terpenting dari teka-teki ini adalah kumpulan jawaban yang akan dimasukkan ke dalam *board*, kumpulan jawaban ini akan disimpan di dalam file eksternal dengan menuliskan angka dari list jawaban secara urutan membesar. Penulisan list jawaban secara berurut dilakukan untuk mengurangi beban program, sehingga program tidak perlu melakukan fungsi sort lagi dan langsung dapat menggunakan fungsi reverse. Alasan kedua adalah karena setiap angka sesungguhnya disimpan dalam bentuk array of char sehingga fungsi sort tidak dapat dilakukan, kecuali kita mengubah tipe bentukan angka yang bersifat array of char tersebut menjadi integer.

```
143
267
468
659
748
946
2619
3580
6034
7129
7519
9256
19076
20537
32706
41037
41853
58321
84192
210496
420358
725692
```

Contoh file external yang menyimpan list jawaban dari *board* diatas

C. Solver

Komponen terakhir dan yang paling penting adalah solver, solver berfungsi untuk menyelesaikan permainan *Number Fill-in Puzzle* secara rekursif, fungsi ini menerima satu buat parameter yaitu jumlah clue yang telah ditelusuri atau dengan istilah lain adalah kedalaman solusi yang telah ditelusuri. Fungsi solver ini akan berhenti saat parameter dari solver telah sama dengan jumlah list jawaban yang artinya semua list jawaban telah dimasukkan ke dalam *board* pertanyaan.

```
def solver(x):
  if(x == totalClue):
    return True
  else:
    empty,s = EmptySpace[x]
    global board
    global clue
    boardTemp = copyBoard(board)
    for c in clue:
      if (len(c)==s):
        if(isSafe(c,empty)):
          putToBoard(c,empty)
          cluetemp.append(c)
          clue.remove(c)
          if(solver((x+1))):
            return True
          clue.insert(0,cluetemp.pop())
          board = copyBoard(boardTemp)
    return False
```

Algoritma Solver dari Number Fill-in Puzzle

Untuk menjalankan *Number Fill-in Puzzle* solver dibutuhkan 2 argument yaitu file eksternal dari *board* dan list jawaban, berikut adalah cara memanggil solver dan keluaran dari program python tersebut :

```
E:\sekuy\makalah>py fillin.py a.txt b.txt
- - - - - # - # - #
- # - # - - - - - -
- - - # - # - # - #
- # - # # - - - - -
# - - - - # # - # -
# - # - # - - - - -
- - - - # - # - # #
# - # - - - # - - -
# - # - # - # # # -
- - - - # - - - - -

2 0 5 3 7 # 6 # 7 #
6 # 8 # 4 2 0 3 5 8
1 4 3 # 8 # 3 # 1 #
9 # 2 # # 8 4 1 9 2
# 7 1 2 9 # # 9 # 6
# 2 # 1 # 4 1 0 3 7
3 5 8 0 # 1 # 7 # #
# 6 # 4 6 8 # 6 5 9
# 9 # 9 # 5 # # # 4
9 2 5 6 # 3 2 7 0 6
Time:  0.024244596
```

Gambar 2. Keluaran *Number Fill-in Puzzle Solver*

Dari gambar diatas kita dapat melihat bahwa solver menemukan sebuah solusi dan penyelesaian *Number Fill-in Puzzle* ini hanya dengan 0.0242 s.

IV. ANALISIS PENYELESAIAN *NUMBER FILL-IN PUZZLE* DENGAN MENGGUNAKAN ALGORITMA *BACKTRACK*

Untuk mempermudah kasus penyelesaian, akan digunakan *board* yang lebih kecil, sehingga kita dapat melakukan analisis secara detail.

***Board* yang digunakan :**

```
-#-###
-----#
-#-#-#
##----
##-###
```

Board untuk Analisis

List jawaban yang digunakan :

```
246
549
1397
40215
42035
```

List jawaban untuk Analisis

Setelah *board* dan List jawaban dibaca oleh program python, selanjutnya akan dicari bagian board mana saja yang dapat diisi dengan angka, berikut adalah hasil pencarian bagian bagian yang dapat diisi oleh program :

```
[(0, 2), (1, 2), (2, 2), (3, 2), (4,
2)] 5
[(1, 0), (1, 1), (1, 2), (1, 3), (1,
4)] 5
[(3, 2), (3, 3), (3, 4), (3, 5)] 4
[(1, 4), (2, 4), (3, 4)] 3
[(0, 0), (1, 0), (2, 0)] 3
```

Hasil pencarian tempat yang dapat diisi di *board*

Setiap elemen dari Hasil pencarian ini adalah *tuple* yang terdiri dari panjang kotak kosong dalam satu baris dan list of *tuple* yang merepresentasi kan index baris dan kolom pada setiap kotak kosong dalam satu baris

Setelah itu akan dicek apakah ada angka yang telah dimasukkan ke dalam board sebelumnya. Hal ini biasa dilakukan jika *puzzle* ini sudah memiliki ukuran yang cukup besar, sehingga memberikan kita acuan bagaimana mengisi *puzzle* tersebut, tidak hanyak dengan menebak-nebak saja. Tetapi untuk kasus kali ini kita tidak menemukan angka pada file eksternal. Sehingga proses ini akan dilewati. Setelah ini program menyelesaikan menjalankan solver(0), 0 adalah angka basis yang menandakan belum ada kata yang dimasukkan ke dalam *board*.

a. Solver(0) dijalankan :

Maka akan diambil kotak kosong yang terpanjang yaitu ([(0 2), (1, 2), (2, 2), (3, 2), (4, 2)], 5) dan jawaban yang akan dicocokkan pertama adalah 42035, karena board dalam posisi belum diisi, maka ketika dicek 42035 dapat memasuki board, sehingga kondisi board sekarang menjadi :

```
- # 4 # # #
- - 2 - - #
- # 0 # - #
# # 3 - - -
# # 5 # # #
```

List jawaban sekarang berkurang satu menjadi :

```
246
549
1397
40215
```

Kotak kosong tersedia tetap sama, tetapi indeksnya mengalami *increment* dan menjadi x = 1, hal ini dilakukan pada saat rekursif selanjutnya, maka akan diambil kotak kosong dengan indeks sesudah rekursif sebelumnya.

b. Solver(1) dijalankan :

Akan diambil kotak kosong yang kedua yaitu ([(1, 0), (1, 1), (1, 2), (1, 3), (1, 4)] 5), kemudian list jawaban akan dicari dari jawaban yang memiliki ukuran terpanjang, angka 40215 akan di uji, dan bisa dimasukkan ke dalam board. Sehingga kondisi board menjadi :

```
- # 4 # # #
4 0 2 1 5 #
- # 0 # - #
# # 3 - - -
# # 5 # # #
```

Kondisi List jawaban sekarang menjadi

```
246
549
1397
```

Kemudian indeks kotak kosong akan di*increment*, menjadi x = 2, sehingga akan dipanggil solver selanjutnya

c. Solver(2) dijalankan:

Akan diambil kotak kosong ketika yaitu ([(3, 2), (3, 3), (3, 4), (3, 5)], 4) dan akan dicek bilangan selanjutnya yaitu 1397, karena 1397 tidak dapat memasuki *board* dan hanya 1397 yang memiliki ukuran 4, program akan mengirimkan nilai False dan akan melakukan pruning, sehingga akan kembali ke solver 1. Kondisi *board* dan List jawaban sama seperti kondisi (b)

d . Solver(1) dilanjutkan (setelah di *pruning)*:

Tahap ini melanjutkan bagian (b), tetapi tidak ada lagi angka yang berukuran 5, sehingga program akan mengirimkan nilai False dan melakukan *pruning* lagi. Akan dilanjutkan kembali solver(0). Kondisi *board* dan List jawaban sama seperti kondisi (a)

e. Solver(0) dilanjutkan (setelah di *pruning*):

Tahap ini melanjutkan bagian (a), kita memiliki angka yang berukuran 5 selain 42035, dengan demikian angka selanjutnya yang berukuran 5 yaitu 42015 akan dimasukkan ke dalam *board*. Kondisi *board* menjadi :

```
- # 4 # # #
- - 0 - - #
- # 2 # - #
# # 1 - - -
# # 5 # # #
```

Kondisi List jawaban menjadi :

```
246
549
1397
42035
```

Indeks kotak kosong akan di *increment*, setelah itu akan dipanggil fungsi rekursif solver(1).

f. Solver(1) dijalankan

Akan diambil kotak kosong yang kedua yaitu ([(1, 0), (1, 1), (1, 2), (1, 3), (1, 4)] 5), kemudian list jawaban akan dicari dari jawaban yang memiliki ukuran terpanjang, angka 42035 akan di uji, dan bisa dimasukkan ke dalam board. Sehingga kondisi board menjadi :

```
- # 4 # # #
4 2 0 3 5 #
- # 2 # - #
# # 1 - - -
# # 5 # # #
```

Kondisi List Jawaban menjadi :

```
246
549
1397
```

Indeks kotak kosong akan di*increment* menjadi x = 2, kemudian akan dipanggil solver(2)

g. Solver(2) dijalankan

Akan diambil kotak kosong ketiga yaitu ([(3, 2), (3, 3), (3, 4), (3, 5)],4). Kemudian akan di cek angka yang memiliki panjang 4. Angka 1397 adalah satu-satunya yang memiliki angka 4 dan angka tersebut dapat dimasukkan ke dalam *board*. Dengan demikian Kondisi *board* menjadi :

```
- # 4 # # #
4 2 0 3 5 #
- # 2 # - #
# # 1 3 9 7
# # 5 # # #
```

Kondisi List Jawaban menjadi :

```
246
549
```

Indeks Kotak Kosong akan ditambahkan, sehingga akan dipanggil solver(3).

h. Solver(3) dijalankan:

Akan diambil kotak kosong ke-empat yaitu ([(1, 4), (2, 4), (3, 4)], 3). Kemudian akan diambil angka yang memiliki ukuran 3. Pertama-tama akan diambil 246, tetapi 246 tidak dapat memenuhi ketentuan dari peraturan permainan, sehingga angka 246 tidak dimasukkan. Setelah itu akan diambil angka 549 dan dilakukan pengecekan. 549 dapat memasuki *board*. Dengan demikian kondisi *board* menjadi :

```
- # 4 # # #
4 2 0 3 5 #
- # 2 # 4 #
# # 1 3 9 7
# # 5 # # #
```

Kondisi List Jawaban menjadi :

```
246
```

Indeks kotak kosong akan di *increment* menjadi x = 4, kemudian akan dipanggil solver(4).

i. Solver(4) dijalankan

Akan diambil kotak kosong yang kelima yaitu ([(0, 0), (1, 0), (2, 0)], 3). Kemudian akan diambil angka terakhir yaitu 246. Setelah dilakukan pengecekan, ternyata 246 memenuhi peraturan permainan dan bisa diletakkan ke dalam *board*, dengan demikian kondisi *board* menjadi :

```
2 # 4 # # #
4 2 0 3 5 #
6 # 2 # 4 #
# # 1 3 9 7
# # 5 # # #
```

Sekarang List Jawaban tidak berisi apapun, karena semua angka telah dipakai ke dalam board. Kemudian indeks kotak kosong akan di*increment* menjadi x = 5, dengan itu akan dipanggil solver(5)

j. Solver(5) dijalankan

Solver(5) masuk ke dalam kondisi ketemu, yaitu ketika x == (total list jawaban). Program akan mengirimkankan nilai true dan penelusuran secara rekursif pun diakhiri.

Berikut adalah hasil yang diberikan program saat kita menjalankan kondisi *board* dan list jawaban seperti diatas :

```
E:\sekuy\makalah>py fillin.py easyboard.txt easyans.txt
- # - # # #
- - - - - #
- # - # - #
# # - - - -
# # - # # #

2 # 4 # # #
4 2 0 3 5 #
6 # 2 # 4 #
# # 1 3 9 7
# # 5 # # #
Time:  0.004526708000000004
```

gambar 3 penyelesaian *Number Fill-in*

V. KESIMPULAN

Algoritma *backtrack* dapat digunakan untuk menyelesaikan beberapa permainan – permainan puzzle dengan syarat kita harus dapat menentukan kapan melakukan *pruning*. Penyelesaian dengan *backtrack* bukanlah penyelesaian yang paling efektif, karena jika kita memainkan permainan *Number Fill-in Puzzle* dengan ukuran *board* yang sangat besar, maka eksekusi waktunya akan sangat lama. Tetapi algoritma ini jauh lebih efektif dari algoritma *exhaustive search*.

Saran saya untuk makalah selanjutnya yang ingin membahas topik yang serupa adalah menggunakan algoritma yang lebih efektif dari *backtrack* atau dapat menggunakan algoritma *backtrack* di dalam permainan – permainan yang belum pernah diselesaikan dengan menggunakan algoritma *backtrack*

VI. UCAPAN TERIMA KASIH

Saya ingin mengucapkan terima kasih kepada Tuhan Yang Maha Esa yang memberi saya kesempatan untuk berkuliah di ITB. Kemudian saya juga ingin berterima kasih kepada dosen pengampu Mata Kuliah IF2211 Strategi Algoritma yang telah memberikan saya pedoman ilmu – ilmu informatika, sehingga saya dapat menyelesaikan makalah "Penyelesaian Number Fill-in Puzzle dengan Algoritma Backtrack".

REFERENSI

[1] https://www.math-salamanders.com/number-fill-in-puzzles.html tanggal akses 25 April 2019

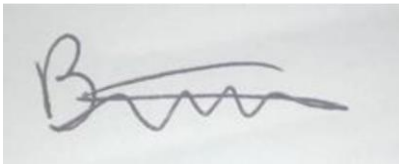
[2] https://www.geeksforgeeks.org/backtracking-algorithms/ tanggal akses 26 April 2019

[3] https://www.interviewbit.com/tutorial/backtracking-pseudocode/ tanggal akses 24 April 2019

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, 26 April 2019



Bram Musuko Panjaitan / 13517089