**LAPORAN TUGAS KECIL 3**

**STRATEGI ALGORITMA**

**KELAS 02**

**Penyelesaian Persoalan 15-Puzzle dengan Algoritma Branch and Bound**

Oleh :

Kevin Austin Stefano (13518104)

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA** **INSTITUT TEKNOLOGI BANDUNG  
2020**

**DASAR TEORI**

**Algoritma *Branch and Bound***

Algoritma *Branch and Bound* adalah jenis algoritma pencarian pada pohon/graf dengan pencarian solusi berupa pembentukan pohon ruang status. Algoritma ini akan memngkas setiap simpul yang tidak mengarah ke simpul tujuan. Algoritma *Branch and Bound* digunakan untuk masalah optimasi dimana meminimalkan atau memaksimalkan fungsi objektif yang ada, namun tetap mengikuti batasan atau *constraint* yang ada.

Pada dasarnya algoritma *Branch and Bound* adalah algoritma perpaduan antara BFS dan algoritma *least cost search.* Namun perbedaannya dengan BFS adalah algoritma *Branch and Bound* tidak melakukan ekspansi berdasarkan urutan kapan dibangkitkan seperti BFS, algoritma *Branch and Bound* akan memberikan cost untuk setiap simpul yang ada dan membangkitkan sesuai dengan *cost* dari simpul-simpul yang ada. *Cost* disini berarti nilai yang dimiliki simpul yang menghitung “harga” perjalanan simpul tersebut dari akar hingga sekarang dan “harga” perjalanan simpul dari sekarang sampai ditemukannya simpul tujuan. Nilai taksiran *cost* yang dimiliki setiap simpul inilah yang akan menandai simpul mana saja yang akan dibangkitkan terlebih dahulu. Sama seperti algoritma *least cost search,* algoritma *branch and bound* akan mencari atau membangkitkan simpul dengan taksiran harga *cost* yag paling kecil.

Selain mencari *cost* terendah untuk simpul yang akan dibangkitkan, algoritma *Branch and Bound* juga mengedepankan fungsi pembatas yang akaan melakukan pemangkasan terhadap simpul yang tidak mengarah ke solusi. Pemangkasan ini dilakukan oleh fungsi pembatas yang ada dengan kriteria bahwa nilai simpul yang tdipangkas tidak lebih baik dari nilai simpul saat ini, simpul melanggar batasan / *constraint*, dan solusi *feasibke* pada simpul tersebut hanya terdiri pada satu titik saja.

Banyak sekali persoalan yang mengguanakan algoritma *Branch and Bound.* Pada tugas kecil kali ini, kita akan mencoba untuk merepresentasikan penggunaka algoritm *branch and bound* untuk pemecahan masalah dalam *game* Puzzle-15.

**ANALISIS**

**ALGORITMA *BRANCH AND BOUND***

Algoritma *Branch and Bound* pada Puzzle-15

1. Memasukkan simpul awal (matriks masukkan awal) ke dalam *queue*. Jika simpul awal adalah simpul tujuan, maka solusi ditemukan dan program berhenti.
2. Melakukan pengecekan apakah simpul awal (matriks masukkan awal) apakah *Reachable Goal* / dapat diselesaikan atau tidak. Fungsi ini dilakukan dengan mengecheck dengan fungsi KURANG(i).
3. Jika tidak *Reachable Goal*, program berhenti dan mengeluarkan pesan “Tidak dapat diselesaikan”
4. Jika *Reachable Goal*, maka persoalan dapat diselesaikan dengan melakukan 3 hal:

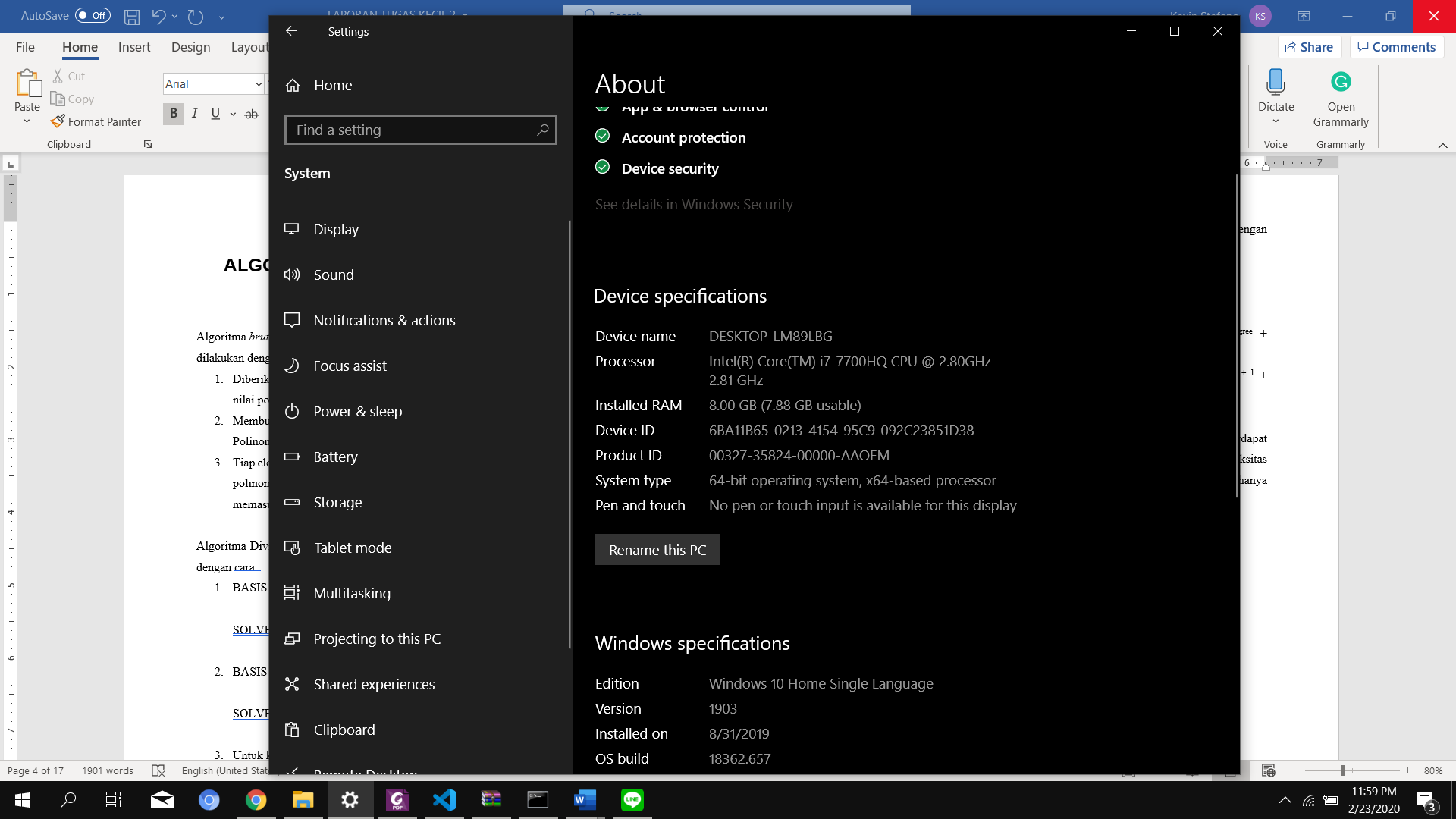
**Pertama,** mempop simpul yang dicheck dari *queue* (simpul ekspand).

**Kedua,** membangkitkan simpul anak-anaknya (yaitu jika ubin kosong melakukan up, down, left, dan right) dan menghitung *costnya* / f(i) + g(i) (f(i) adalah panjang lintasan dari simpul akar ke simpul sekarang, g(i) adalah jumlah ubin tidak kosong yang tidak terapat pada susunan akhir)

**Ketiga,** memasukkan kedalam *queue* simpul anak-anak yang dibangkitkan dengan cara memasukkan ke depan *queue* untuk simpul anak dengan *cost* lebih kecil dari *cost* simpul sebelumnya dan memasukkan de belakang *queue* untuk simpul anak dengan *cost* lebih besar.

1. Jika *queue* sudah kosong atau simpul *goal* sudah ditemukan ( simpul yang berbentuk simpul dengan matriks susunan akhir), program berhenti
2. Jika *queue* belom kosong atau belom ditemukan simpul *qoal,* maka terus melakukan langkah ke-4.

**SPESIFIKASI KOMPUTER**

Berikut ini adalah spesifikasi dari komputer yang digunakan untuk membuat dan menjalankan program.

**KODE PROGRAM**

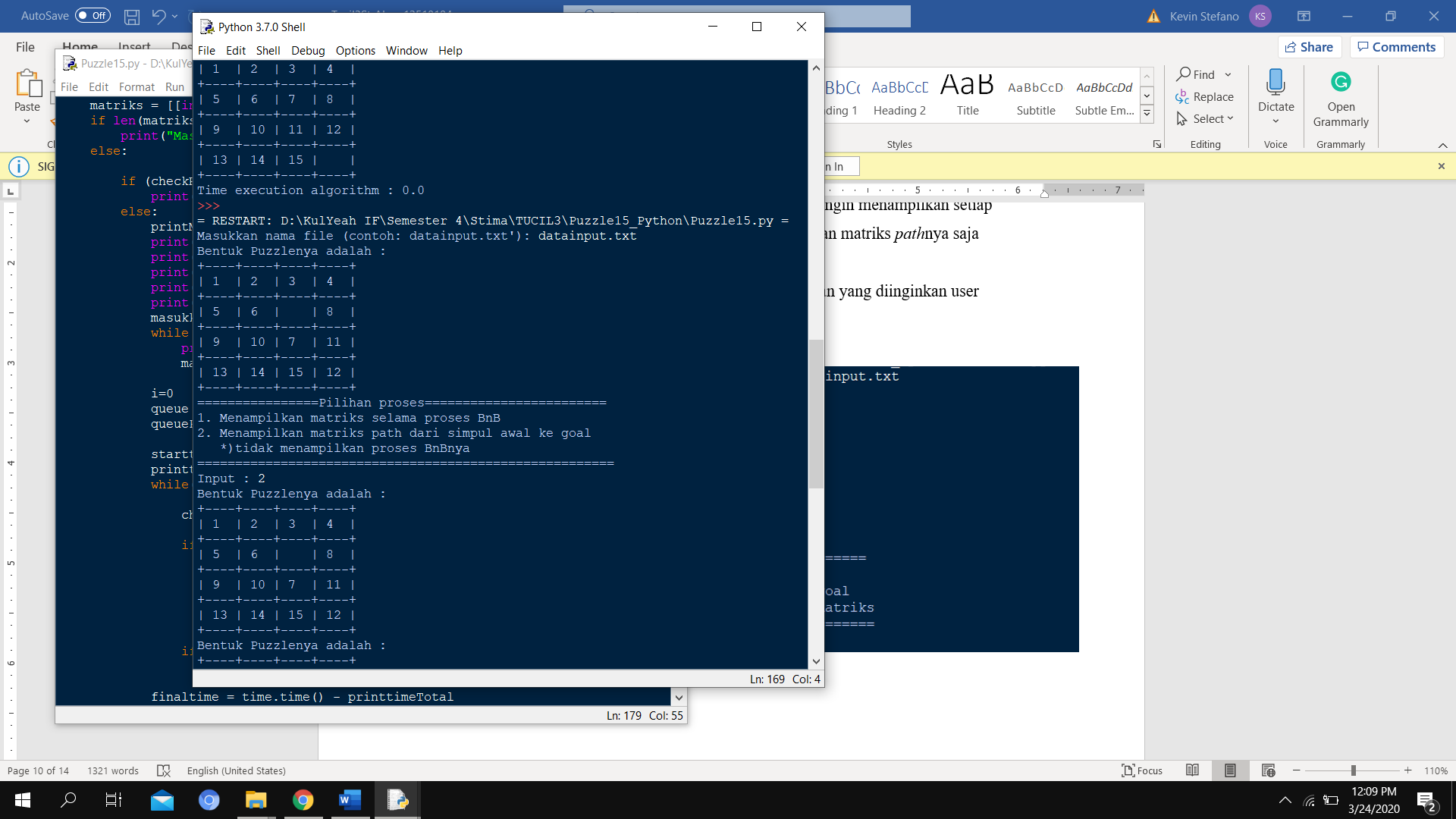
|  |
| --- |
| #Name : Kevin Austin Stefano  #NIM : 13518104  import collections  import time  **def printMatriks (matriks):**  print("Bentuk Puzzlenya adalah : ")  print("+----+----+----+----+")  for i in range(4):  print("", end='| ')  for j in range(4):  if matriks[i][j]<10 and matriks[i][j]>0:  print(matriks[i][j],end=' | ')  elif matriks[i][j]==0:  print(" ",end=' | ')  else:  print(matriks[i][j],end=' | ')    print()  print("+----+----+----+----+")    **def checkReachableGoal (matriks):**  #cari index yang kosong  for i in range(4):  for j in range(4):  if matriks[i][j]==0:  break  if matriks[i][j]==0:  break  #Cari nilai X  if ((i==0 or i==2) and (j==1 or j==3)) or ((i==1 or i==3) and (j==0 or j==2)):  x = 1  else:  x = 0    #cari nilai fungsi KURANG(i)  kurang = 0  #itterasi tiap element matriks  for k in range(4):  for l in range(4):  #check tiap element matriks  if l==3 and k<3:  m= k+1 #baris  n= 0 #kolom  stop=0  elif l==3 and k==3:  stop =1  else:  m = k  n = l+1  stop=0  while (m<4 and stop==0):  while (n<4):  if (matriks[m][n]<matriks[k][l] and matriks[m][n]!=0) or matriks[k][l]==0:  kurang=kurang+1  n=n+1  n=0  m=m+1  #hitung total  kurang = kurang+x  #check apakah genap atau bukan  if kurang%2==0:  return 1 #bisa  else:  return 0 #tidakbisa      **def checkDenganSusunanAkhir(matriks):**  count = 0  check = 1  for i in range(4):  for j in range(4):  if matriks[i][j]!=0 and matriks[i][j] != check:  count=count+1  check = (check+1)%16  return count  **def whereKotakKosong(matriks):**  for i in range(4):  for j in range(4):  if matriks[i][j]==0:  break;  if matriks[i][j]==0:  break;  return i,j  **def swipeAndCheck(matriks, queue, kosongI, kosongJ, swipeI, swipeJ,langkah, cost0):**  costNow = 0  langkah = langkah +1  if (swipeI==kosongI-1 and swipeJ==kosongJ) or (swipeI==kosongI+1 and swipeJ==kosongJ) or (swipeJ==kosongJ+1 and swipeI==kosongI) or (swipeJ==kosongJ-1 and swipeI==kosongI):  matriks[kosongI][kosongJ] =matriks[swipeI][swipeJ]  matriks[swipeI][swipeJ]=0  costNow = checkDenganSusunanAkhir(matriks)+langkah    matriksacuan = []  for k in range(4):  new=[]  for l in range(4):  new.append(matriks[k][l])  matriksacuan.append(new)    if (len(queue)!=0):  if cost0>=costNow:  queue.appendleft((matriksacuan,costNow,langkah))    else:  queue.append((matriksacuan,costNow,langkah))    else:  queue.append((matriksacuan,costNow,langkah))      matriks[swipeI][swipeJ]=matriks[kosongI][kosongJ]  matriks[kosongI][kosongJ] = 0    **def checkUDLRandAddQueue(queue,queuePath,sumsimpul):**  matriks,c,langkah = queue[0]  kosongI,kosongJ = whereKotakKosong(matriks)  count =checkDenganSusunanAkhir(matriks)  if count==0:  return 1  else:  matriksPath= []  for k in range(4):  new=[]  for l in range(4):  new.append(matriks[k][l])  matriksPath.append(new)  queuePath.appendleft((matriksPath,c,langkah))  del queue[0]    #checkjika UP  if (kosongI-1>=0):  swipeAndCheck(matriks,queue,kosongI,kosongJ,kosongI-1,kosongJ,langkah,c)  sumsimpul.append(1)    #checkjika DOWN  if (kosongI+1<=3):  swipeAndCheck(matriks,queue,kosongI,kosongJ,kosongI+1,kosongJ,langkah,c)  sumsimpul.append(1)    #checkjika LEFT  if (kosongJ-1>=0):  swipeAndCheck(matriks,queue,kosongI,kosongJ,kosongI,kosongJ-1,langkah,c)  sumsimpul.append(1)    #checkjika RIGHT  if (kosongJ+1<=3):  swipeAndCheck(matriks,queue,kosongI,kosongJ,kosongI,kosongJ+1,langkah,c)  sumsimpul.append(1)    mat,cos,walk = queue[0]  matB, cosB, walkB = queuePath[0]  if(walk==walkB):  del queuePath[0]  elif(walkB>walk):  while(walk>walkB):  del queuePath[0]  matB, cosB, walkB = queuePath[0]  return 0    **#MAIN**  #Masukan file eksternal  filemasuk = input("Masukkan nama file (contoh: datainput.txt'): ")  with open(filemasuk, 'r') as f:  matriks = [[int(idx) for idx in line.split(' ')] for line in f]  if len(matriks)!= 4 or len(matriks[0]) != 4:  print("Masukan file matriks harus berukuran 4x4");  else:  if (checkReachableGoal(matriks)==0):  print("Matriks tidak bisa di proses")  else:  printMatriks(matriks)  print("================Pilihan proses========================")  print("1. Menampilkan matriks selama proses BnB")  print("2. Menampilkan matriks path dari simpul awal ke goal")  print(" \*)tidak menampilkan proses BnBnya")  print("=======================================================")  masukkan = input("Input : ")  while (masukkan!='1' and masukkan!='2'):  print("Nilai input salah")  masukkan = input("Input : ")  i=0  sumsimpul = collections.deque()  queue = collections.deque([(matriks,999,i)])  queuePath = collections.deque()  starttime = time.time()  printtimeTotal = 0  while(queue):  checker = checkUDLRandAddQueue(queue,queuePath,sumsimpul)  if (masukkan=='1'):  printtimestart = time.time()  ma,co,i = queue[0]  print("----Jarak simpul ke akar :",i,"----")  printMatriks(ma)  printtimefinal = time.time()  printtimeTotal = printtimeTotal + (printtimefinal-printtimestart)  if checker==1:  break  finaltime = time.time() - printtimeTotal  if(masukkan=='2'):  p=1  i= len(queuePath)  while (i!=0):  print("----Langkah ke-",p,"----")  ma, c, l = queuePath[i-1]  printMatriks(ma)  i=i-1  p=p+1    print("====Langkah ke-",p,"====")  ma, c, l = queue[0]  printMatriks(ma)  print("Time execution algorithm :", finaltime-starttime)  print("Jumlah simpul yang dibangkitkan :",len(sumsimpul)) |

**TES UJI**

Keterangan :

1. Masukkan data input yang akan diuji
2. Maka yang di *outputkan* pertama adalah bentuk *puzzle* untuk matriks dari data input
3. Kemudian memasukkan inputan kedua yaitu pilihan ingin menampilkan setiap *puzzle*nya selama proses BnB atau hanya menampilkan matriks *path*nya saja
4. Program akan memproses matriks inputan
5. Program akan menampilkan hasil *puzzle* sesuai dengan yang diinginkan user, waktu eksekusi pengerjaannya, dan jumlah simpul dibangkitkan

Berikut ini adalah bentuk tampilan awalnya

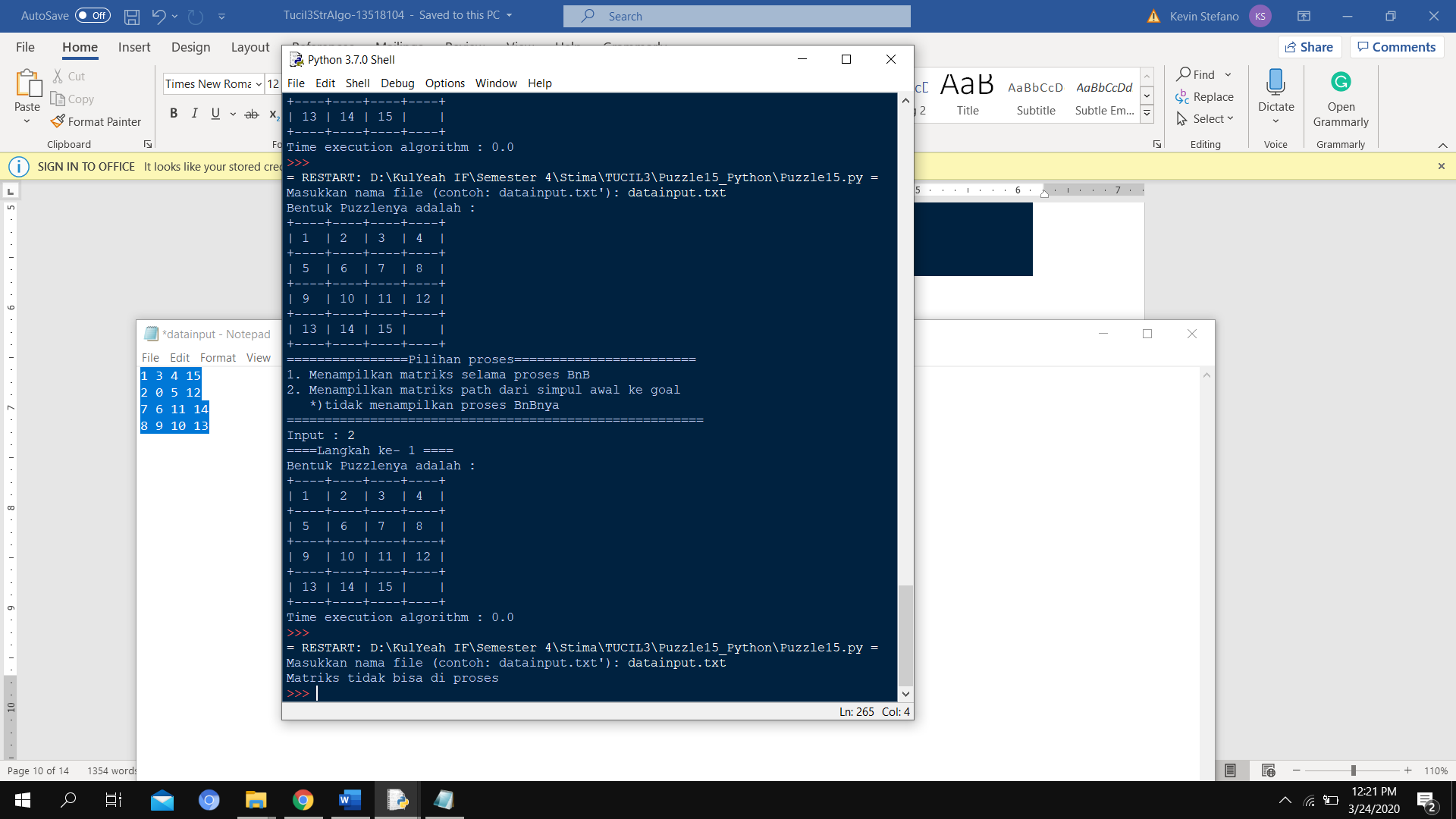


**Test Uji 1 –** Tidak reachable goal

Berikut adalah matriks inputnya (file: datainput.txt)

|  |
| --- |
| 1 3 4 15  2 0 5 12  7 6 11 14  8 9 10 13 |

Keluaran dalam program

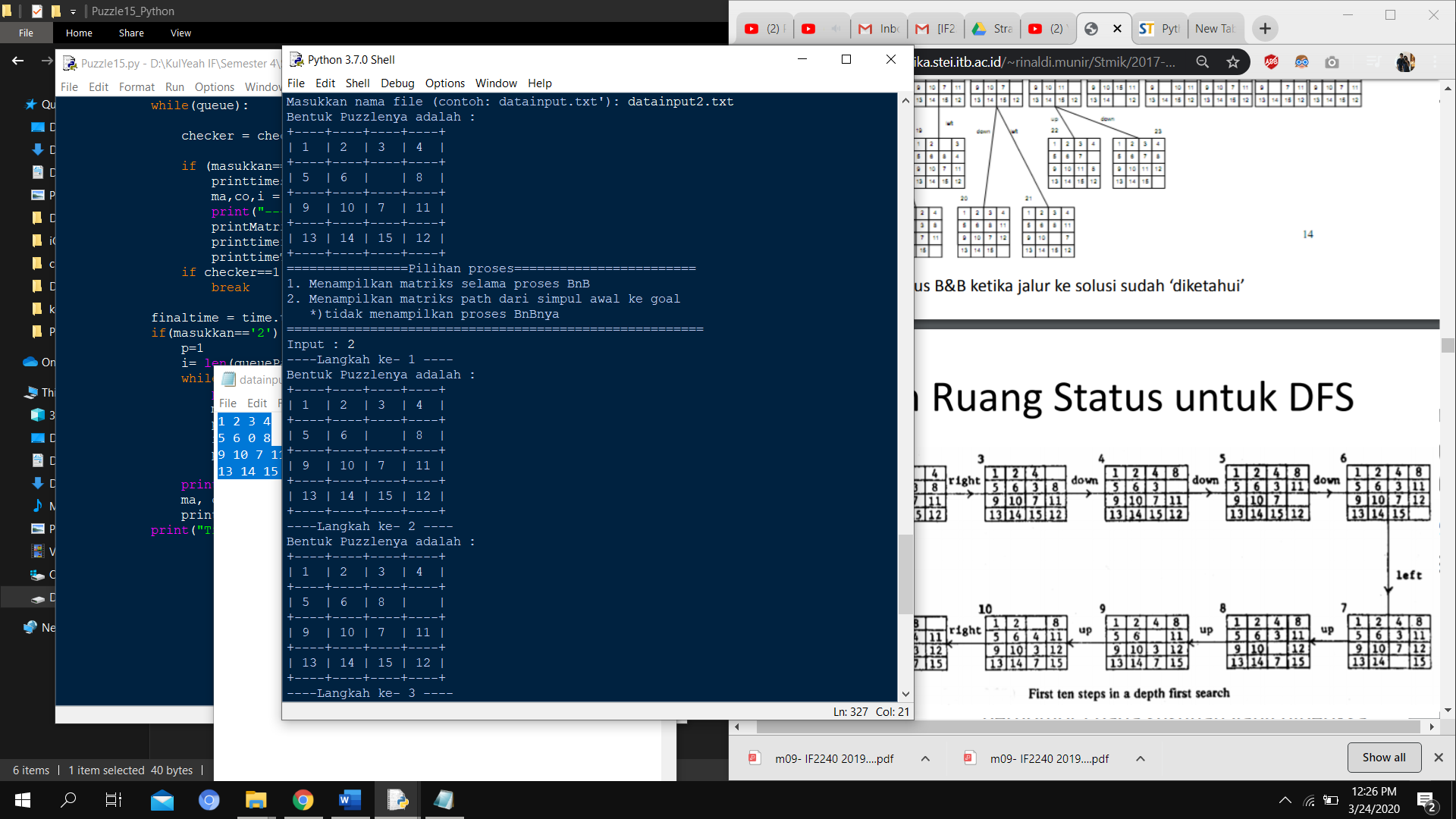


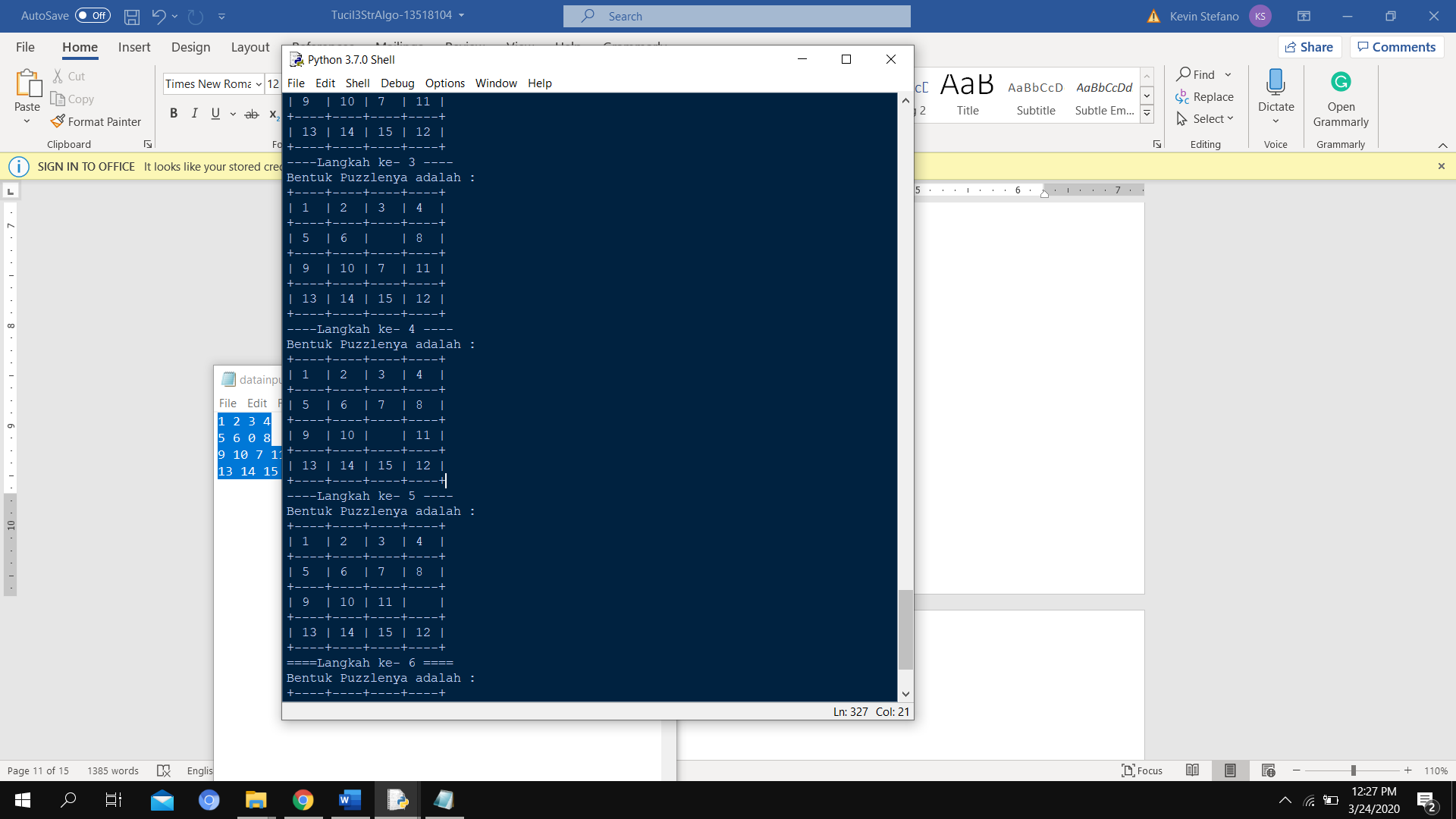
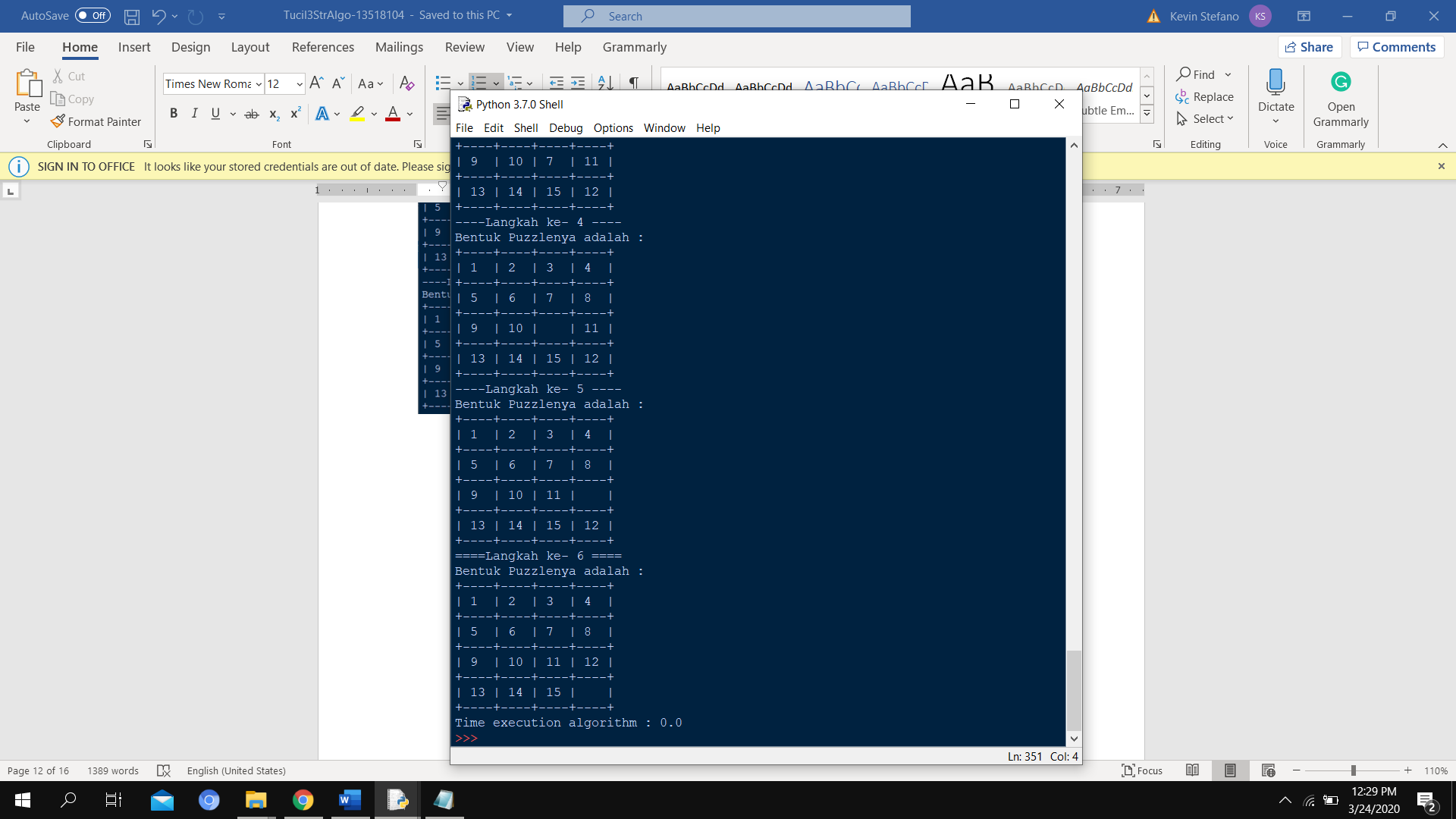
**Test Uji 2**

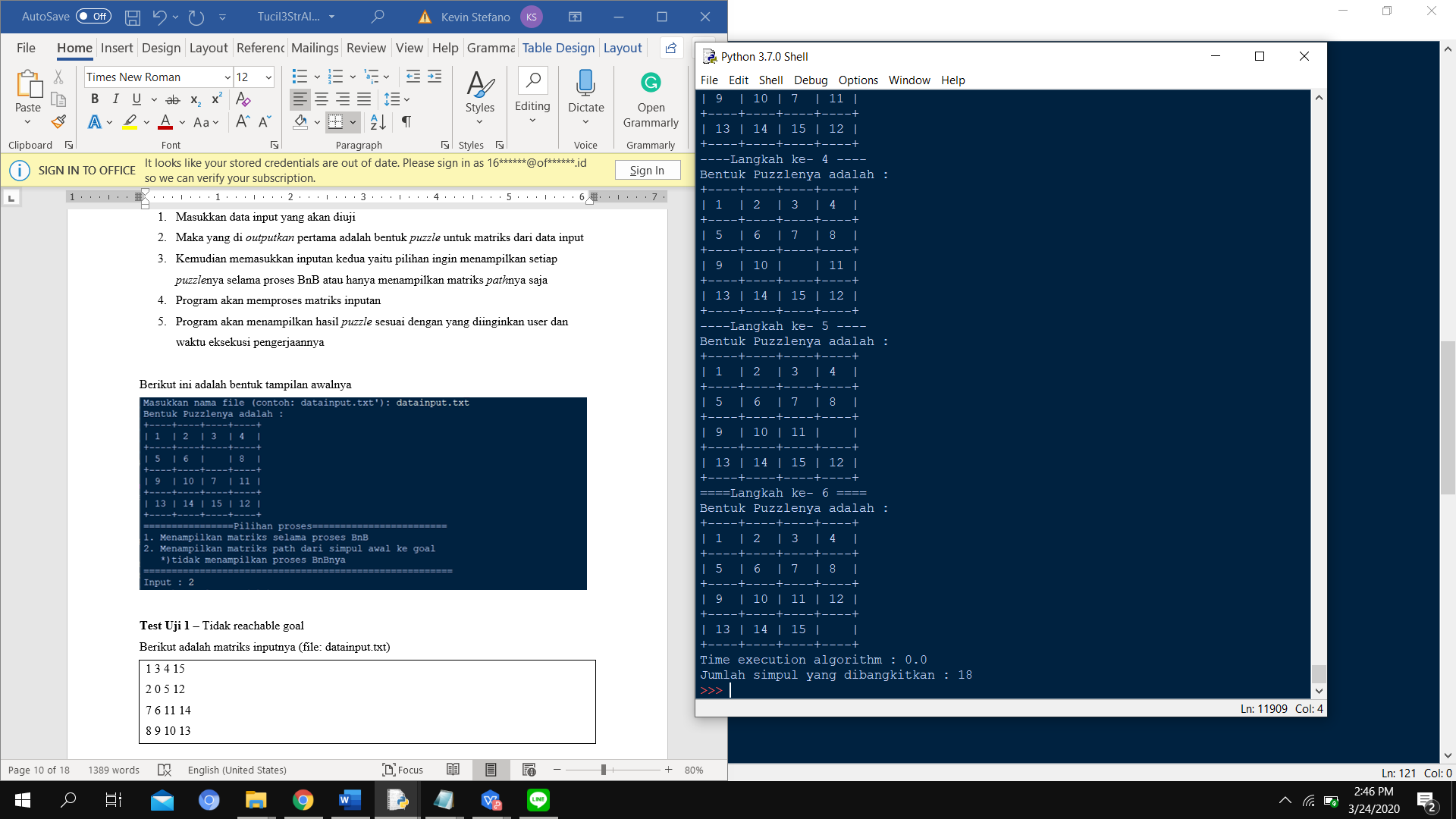
Berikut adalah matriks inputnya (file: datainput2.txt)

|  |
| --- |
| 1 2 3 4  5 6 0 8  9 10 7 11  13 14 15 12 |

Keluaran dalam program





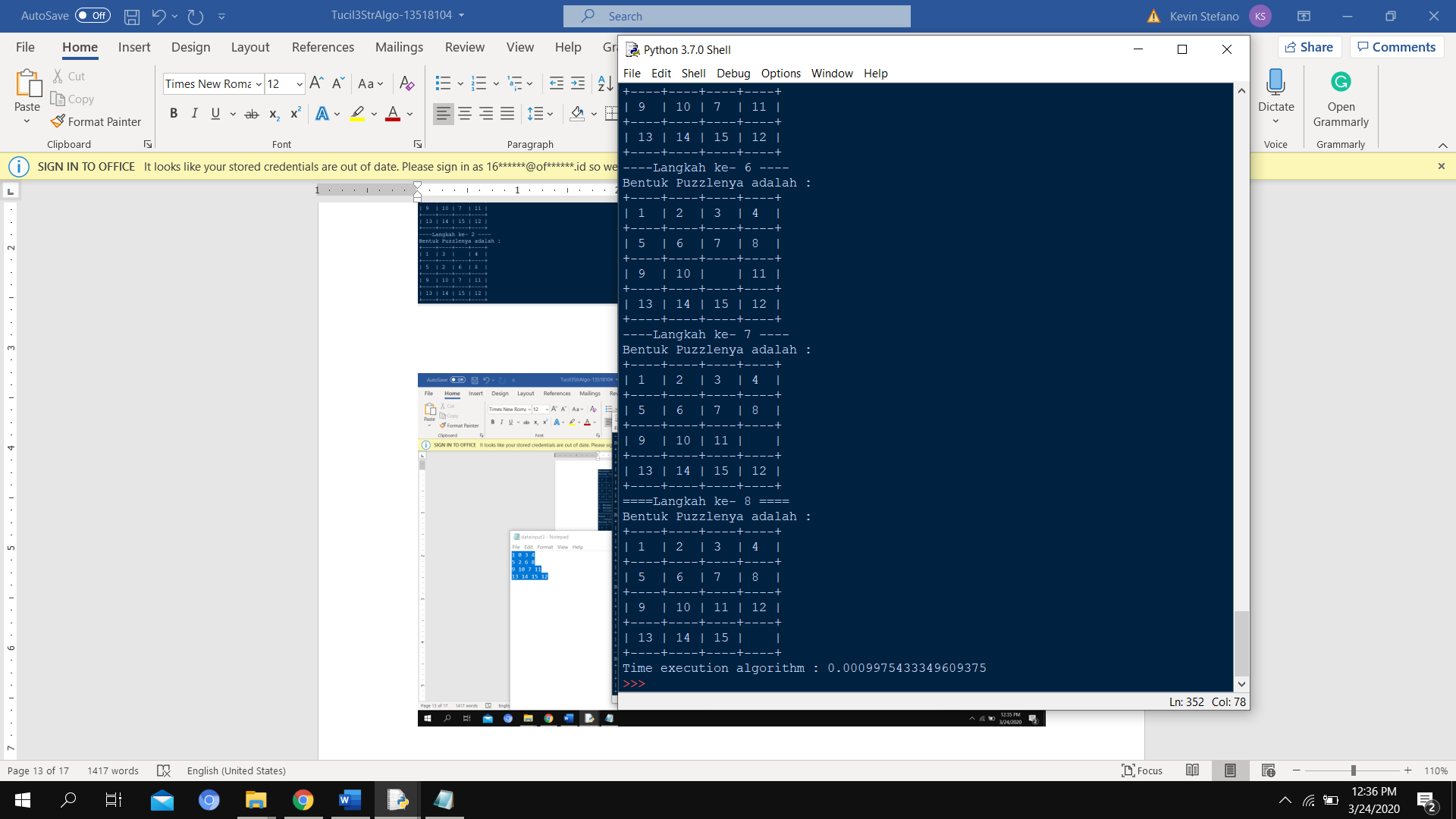
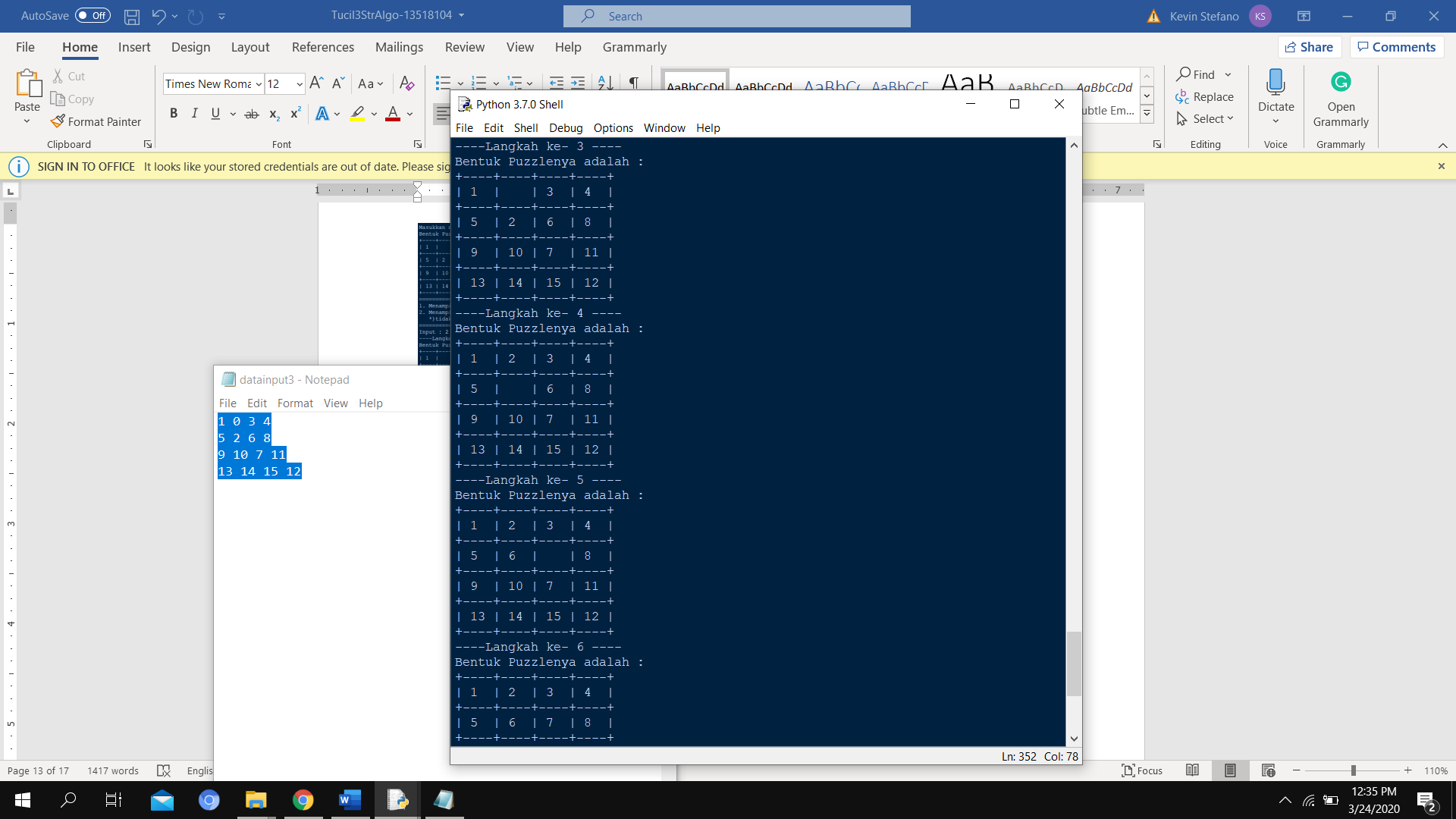
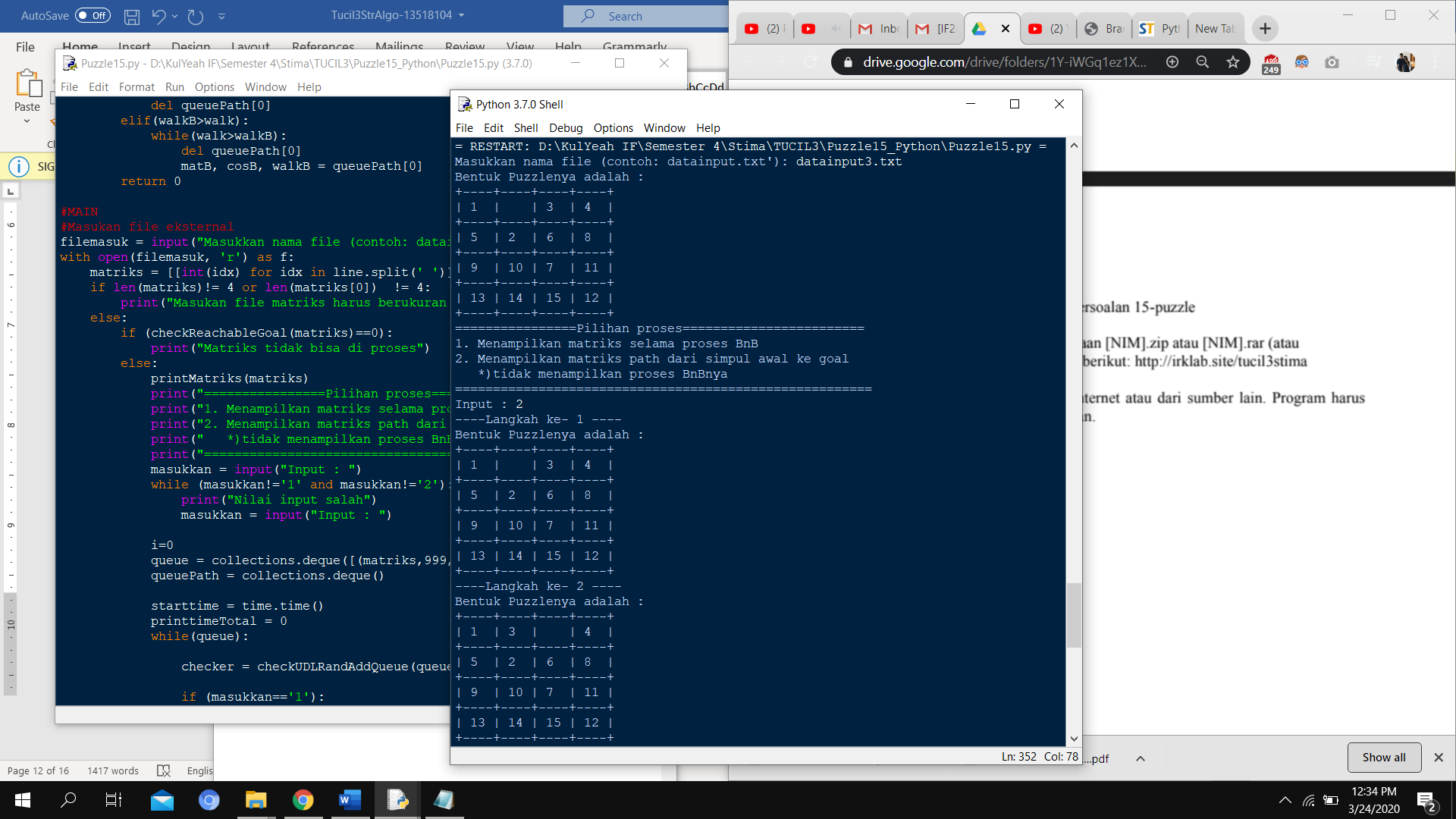


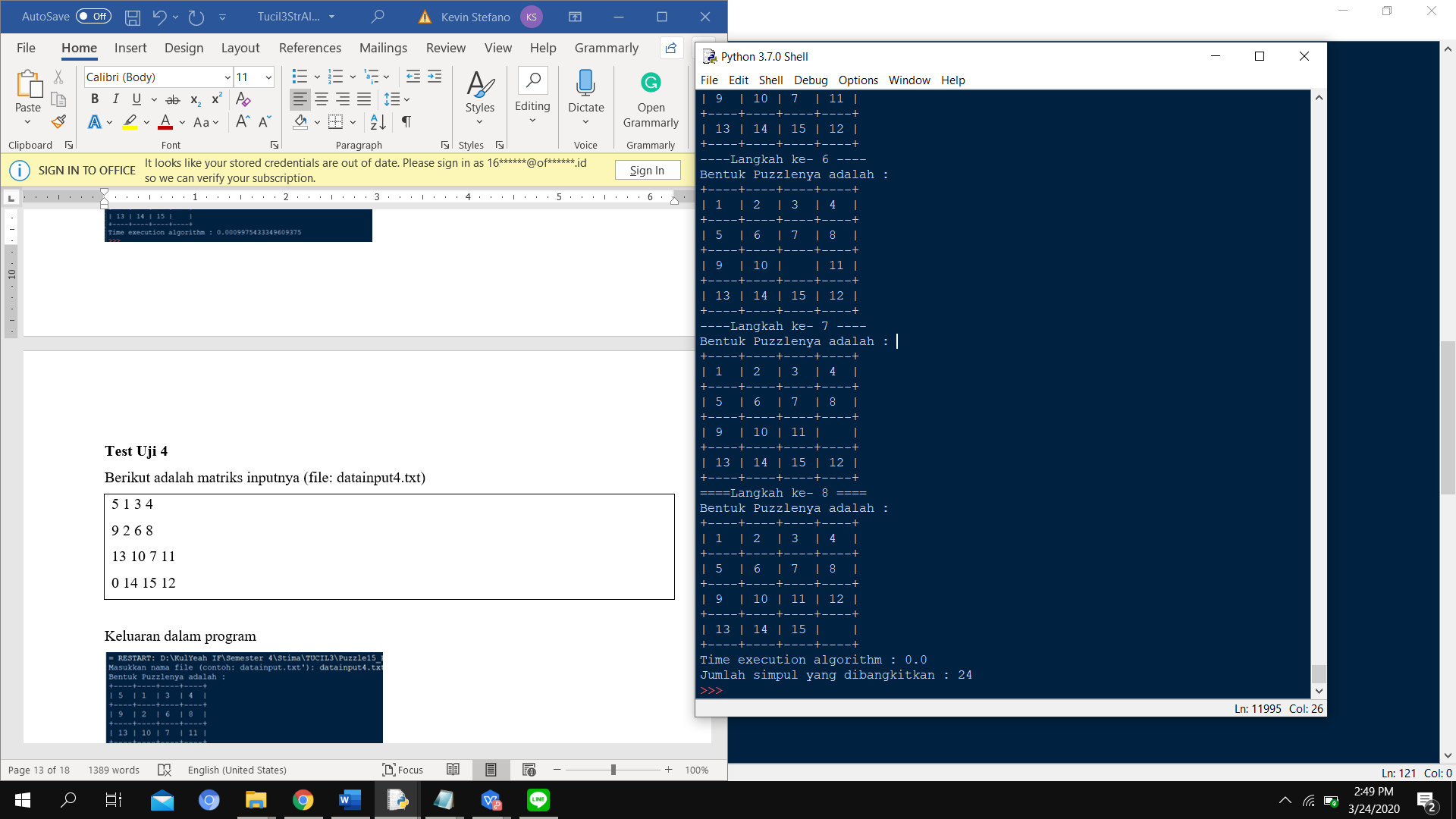
**Test Uji 3**

Berikut adalah matriks inputnya (file: datainput3.txt)

|  |
| --- |
| 1 0 3 4  5 2 6 8  9 10 7 11  13 14 15 12 |

Keluaran dalam program



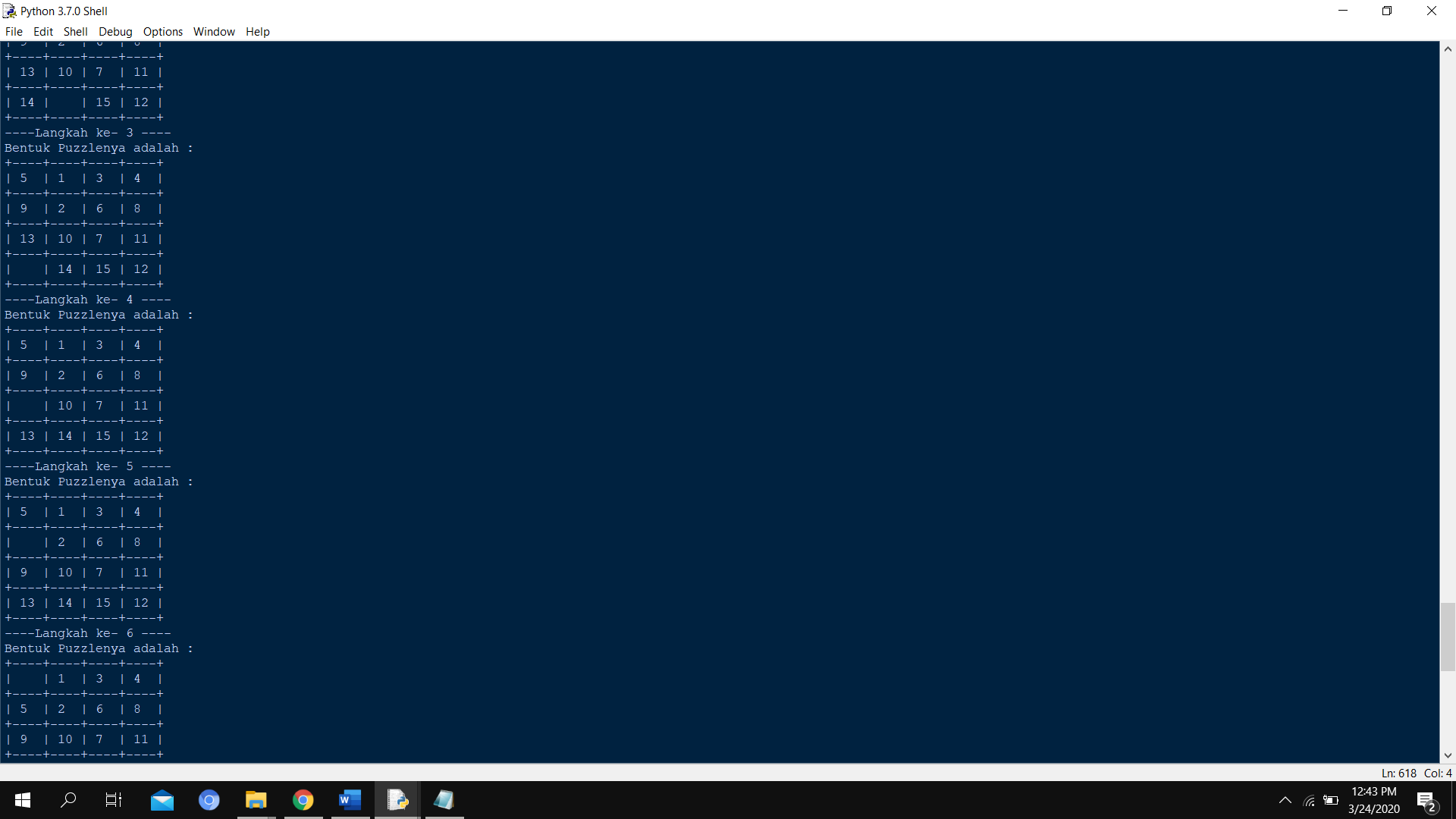
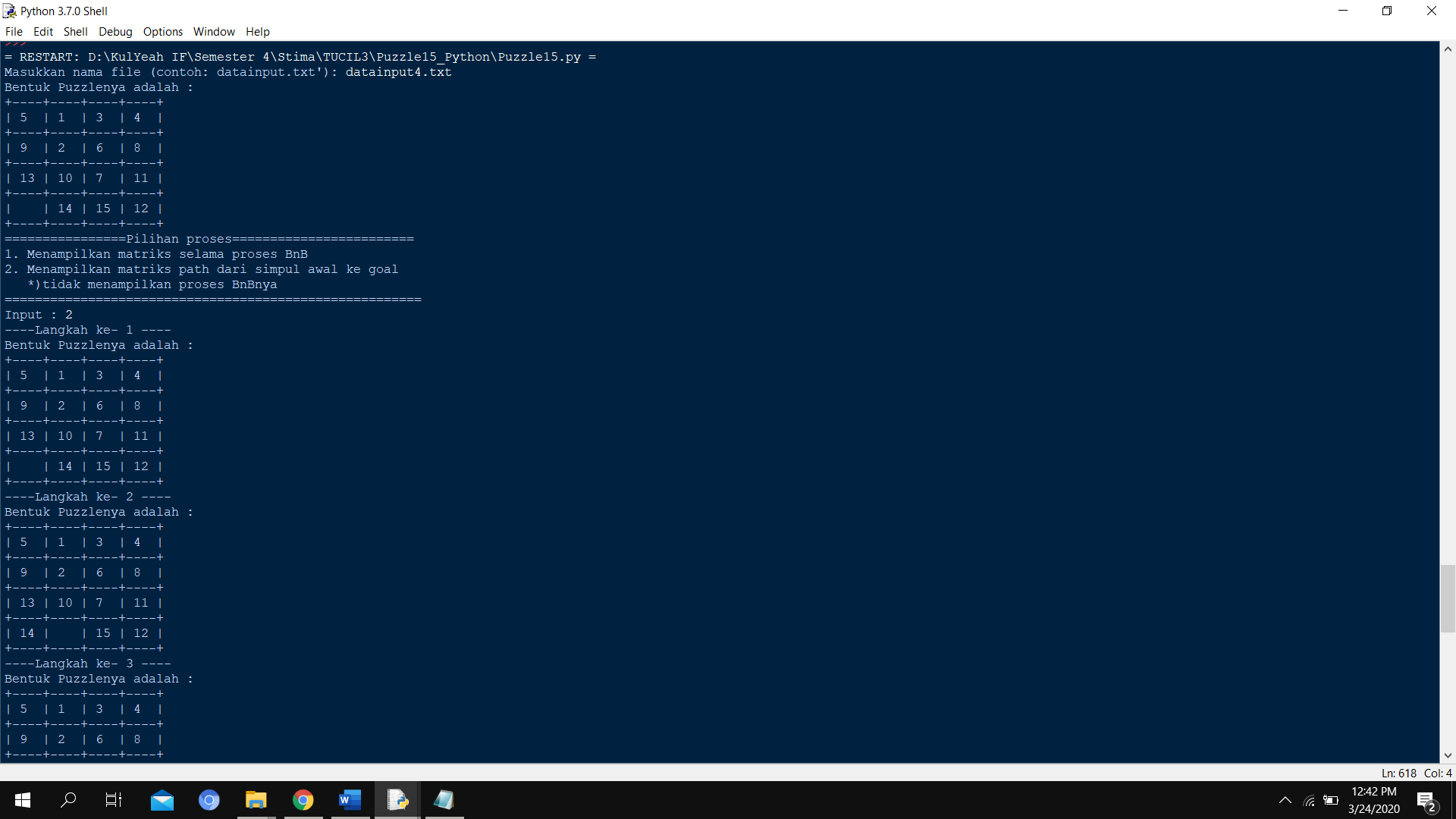


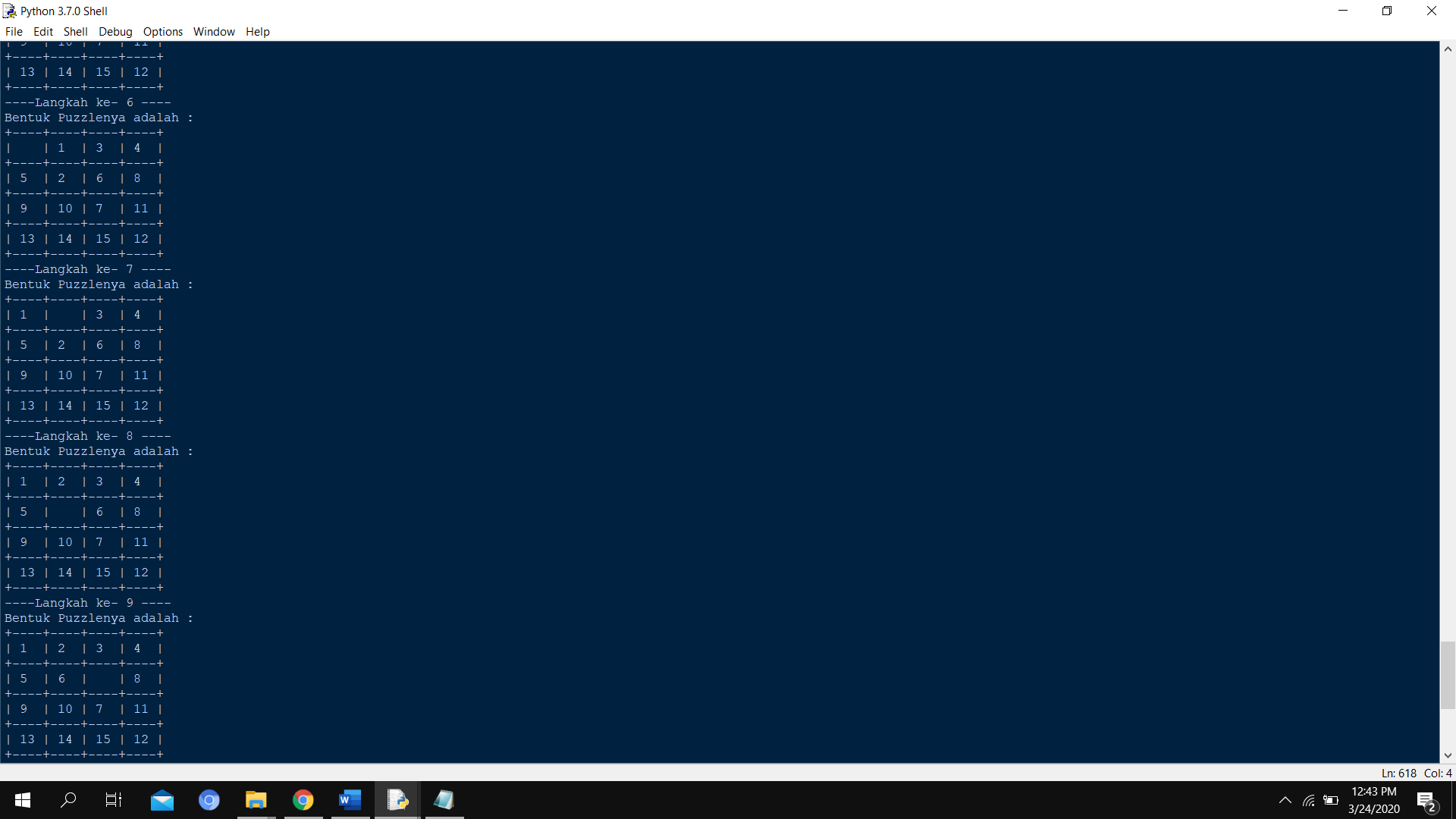
**Test Uji 4**

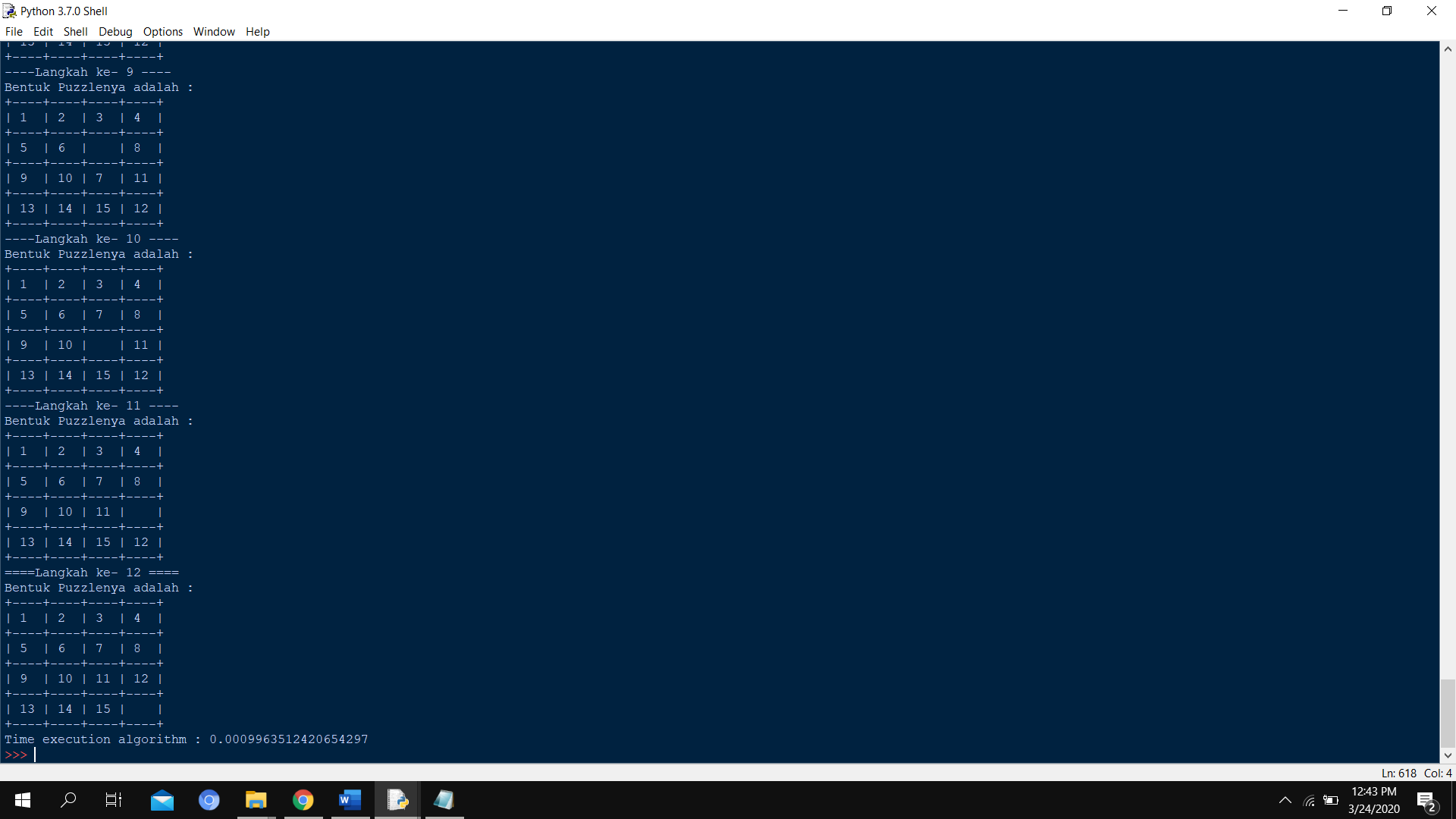
Berikut adalah matriks inputnya (file: datainput4.txt)

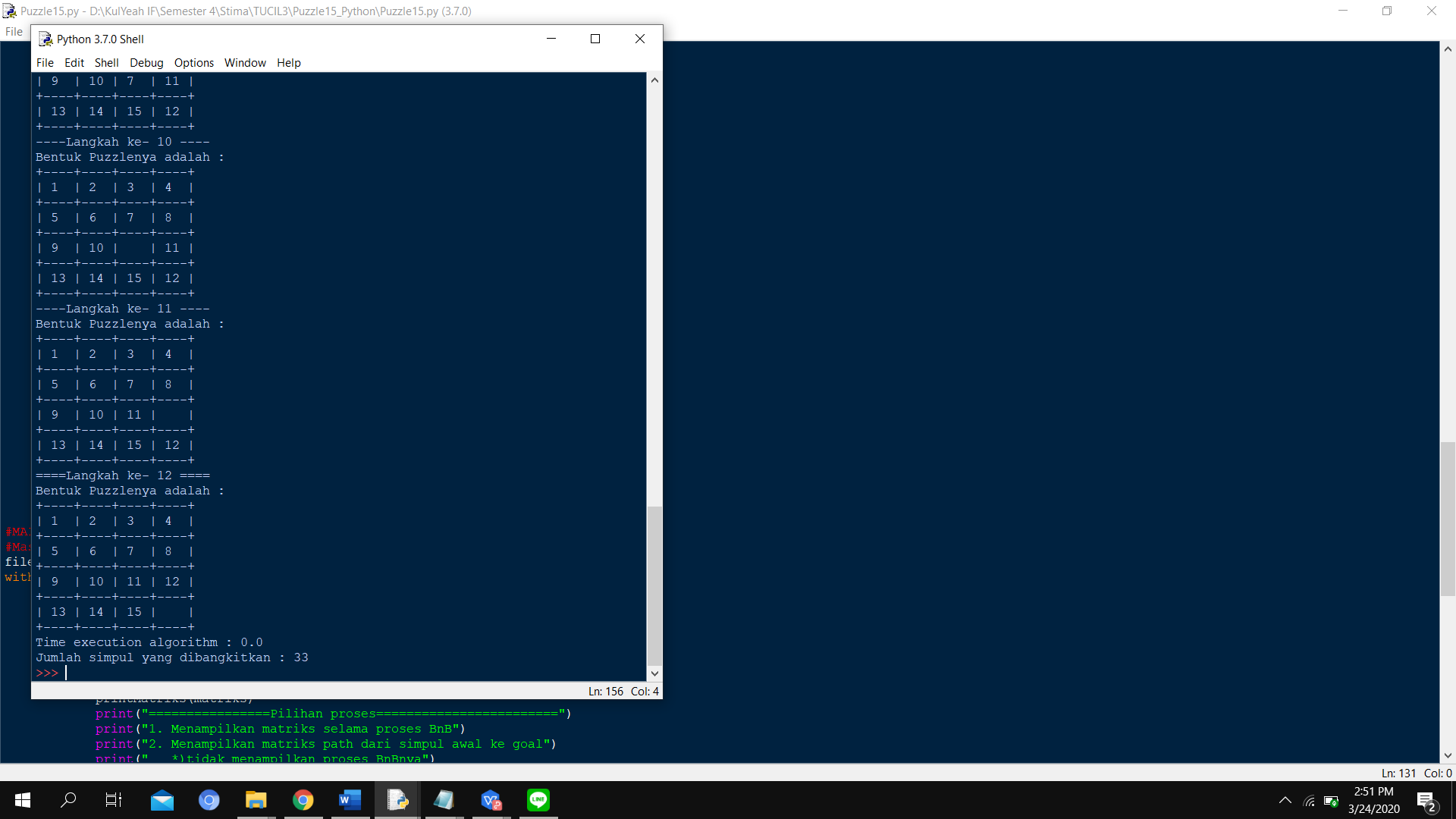
|  |
| --- |
| 5 1 3 4  9 2 6 8  13 10 7 11  0 14 15 12 |

Keluaran dalam program







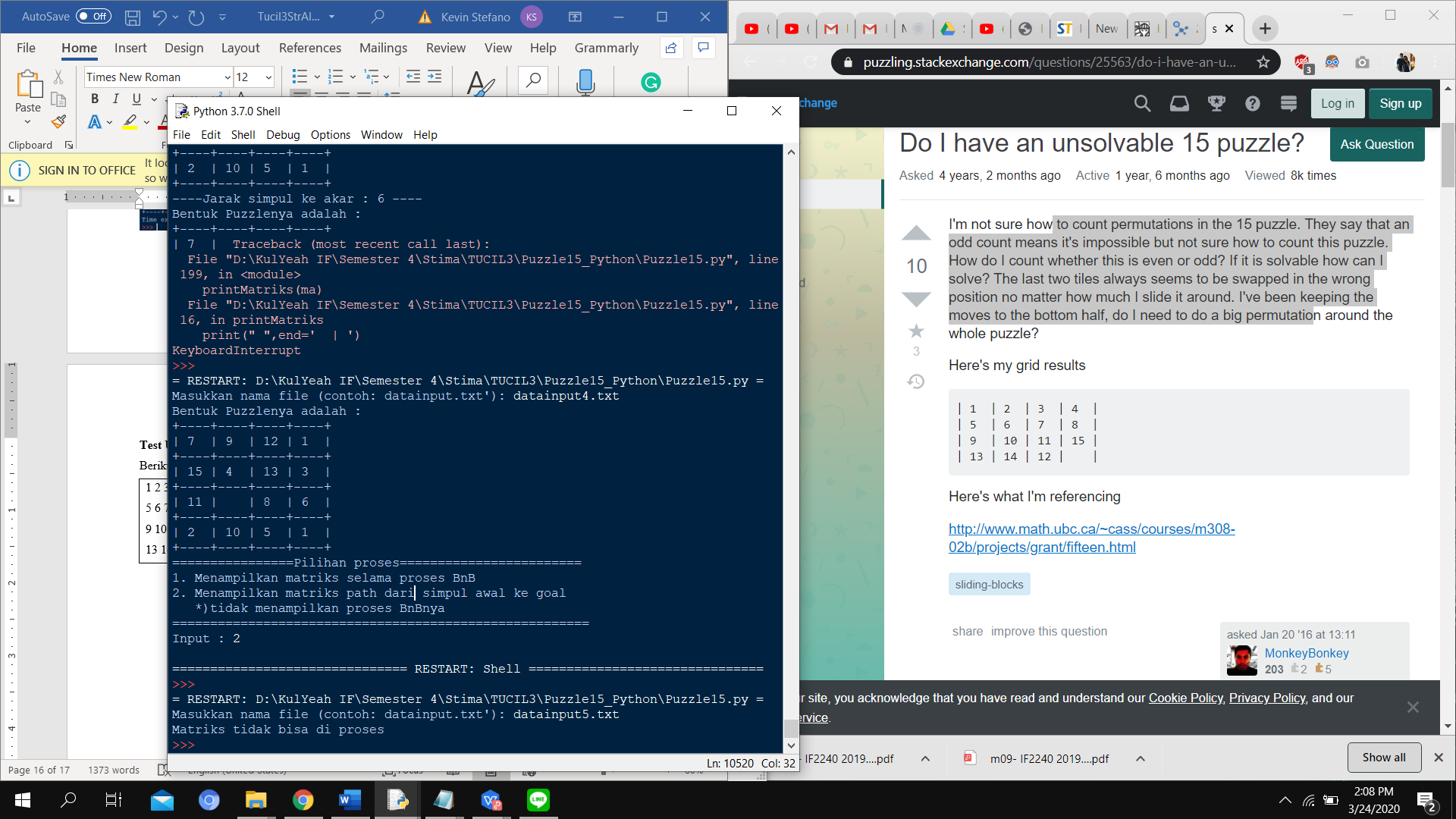


**Test Uji 5 –** Tidak reachable goal

Berikut adalah matriks inputnya (file: datainput5.txt)

|  |
| --- |
| 1 2 3 4  5 6 7 8  9 10 11 15  13 14 12 0 |

Hasil Keluaran program,



**KESIMPULAN**

Algoritma *Branch and Bound* merupakan algoritma yang lebih baik dibandingkan BFS dan efektif dijalankan saat simpul goal terletak dekat dengan simpul akar. NAMUN, algoritma *Brnach and Bound* sangatlah tidak efektif untuk

|  |  |  |
| --- | --- | --- |
| Poin | Ya | Tidak |
| 1. Program berhasil dikompilasi | v |  |
| 2. Program berhasil *running* | v |  |
| 3. Program dapat menerima input dan menuliskan output. | v |  |
| 4. Luaran sudah benar untuk semua n | v |  |

**REFERENSI**

[1] Munir, Rinaldi. 2007*. Diktat Kuliah IF2251 Strategi Algoritmik*. Program Studi Teknik

Informatika STEI ITB.