Na początku programu zamieniamy dane wejsciowe na slownik zaleznosci. (klucz litera, wartosci, tablica z literami na ktorych operuje)

```
translated = {}
for key, val in A.items():
   # [key, dependencies]
   main_id = val[:1]
   dep = []
   for digit in val[2:]:
       if digit.isalpha():
           dep.append(digit)
   translated[key] = [main_id, dep]
```

Nastepnie tworzymy dwa sety, dla relacji zaleznosci i niezaleznosci. Robimy petle ktora idzie przez wszystkie elementy, jesli jeden ktorys jest zalezny od czegos z drugiego elementu, dodajemy pare do tablicy zaleznosci, a jesli nie - do niezaleznosci

```
D = set()
I = set()
for key1, val1 in translated.items():
   for key2, val2 in translated.items():
       if val1[0] in val2[1] or val2[0] in val1[1] or key1 == key2:
           D.add((key1, key2))
       else:
           I.add((key1, key2))
```

Postać normalna Foaty FNF. Na poczatku przygotowywujemy tablice tablic. Dla kazdej litery z podanego slowa przechodzimy przez juz wpisane litery do postaci Foaty, jesli obecna litera jest zalezne z jakas z natrafionych juz wpisany do Foaty, to dodajemy ja do listy zaraz po niej.

```
FNF = [[]]
for idx, letter in enumerate(w):
   flag = False
   for i in range(len(FNF) - 1, -2, -1):
       if i == -1:
           FNF[0].append((letter, idx))
           break
       for el, _ in FNF[i]:
           if (letter, el) in D:
               if i == len(FNF) - 1:
                   FNF.append([(letter, idx)])
               else:
                   FNF[i + 1].append((letter, idx))
               flag = True
               break
       if flag:
           break
```

Ponizej kod służacy do stworzenia minimalnego grafu zaleznosci, dziala na zasadzie ze zapisujemy w tablicy wszystkie rozgalezienia zalensci, jesli jest jakies odgalezienie, badz nowy korzen, to dodajemy to jako osobna galaz, a kolejnie iterujemy przez kazda galaz i jak znajdziemy zaleznosc to dodajemy. Dokladniej opisane w komentarzach w kodzie

```
G = [[] for _ in range(len(w))]
# w branches przechowywane sa poszczegolne galezie zaleznosci
branches = []
# lecimy po wszystkich literach ze slowa
for idx, letter in enumerate(w):
   # potrzebujemy przetrzymywac flage, czy dodalismy cos w trakcie, bo jesli
nie dodajemy nowy korzen
   added = False
   for branch in branches.copy():
       # jesli jest zalezny od ostatniego elementu w galezi, po prostu dodaj
go na koniec
       if (branch[-1][0], letter) in D:
           # przy dodaniu do branches, od razu zapisaujemy zaleznosci w
grafie
           G[branch[-1][1]].append(idx)
           added = True
           branch.append((letter, idx))
           continue
       # w przeciwnym wypadku, przegladamy po wczesniejszych literach w tej
galezi i jesli jest zalezne,
       # tworzymy nowa odnoge w branches
       for i in range(len(branch) - 2, -1, -1):
           if (branch[i][0], letter) in D:
               G[branch[-i][1]].append(idx)
               added = True
               branches.append([(letter, idx)])
               break
   # tutaj dodajemy jesli nie bylo zalezne od niczego wczesniej
   if not added:
       branches.append([(letter, idx)])
```

Kod sluzacy do wizualizacji

```
A = {
   "a": "x := x + y",
   "b": "y := y + 2z",
   "c": "x := 3x + z",
   "d": "z := y - z"
}
w = "baadcb"

B = {
   "a": "x := x + 1",
   "b": "y := y + 2z",
   "c": "x := 3x + z",
```

```
    "d": "w := w + v",
    "e": "z := y - z",
    "f": "v = x + v"
}
w2 = "acdcfbbe"
```

```
def show_results_for(tab, word):
    D, I, fnf, G = interpreter(tab, word)

    formatted_fnf = ""
    for tab in fnf:
        formatted_fnf += "("
        for el in tab:
            formatted_fnf += el[0]
        formatted_fnf += ")"

    print(f"D = {D}")
    print(f"I = {I}")
    print(f"FNF = {formatted_fnf}")
    print("\nGraph (format .dot)")
    print("digraph G {")
    for fr, to_tab in enumerate(G):
        for to in to_tab:
            print(f"  {fr} -> {to};")
    for id, letter in enumerate(word):
        print(f"  {id}[label=\"{letter}\"];")
    print("}")


print("Results for test data 1:")
show_results_for(A, w)

print("\n-------------------------")
print("\nResults for test data 2:")
show_results_for(B, w2)
```

Wizualizacja wyników:
Results for test data 1:
D = {('b', 'b'), ('a', 'c'), ('c', 'd'), ('b', 'a'), ('d', 'd'), ('d', 'c'), ('c', 'a'), ('a', 'b'), ('b', 'd'), ('a', 'a'), ('d', 'b'), ('c', 'c')}
I = {('b', 'c'), ('d', 'a'), ('a', 'd'), ('c', 'b')}
FNF = (b)(ad)(a)(cb)

Graph (format .dot)

```
digraph G {
  0 -> 1;
  0 -> 3;
  1 -> 2;
  2 -> 4;
  2 -> 5;
  3 -> 4;
  3 -> 5;
  0[label="b"];
  1[label="a"];
  2[label="a"];
  3[label="d"];
  4[label="c"];
  5[label="b"];
}
```

Results for test data 2:
D = {('c', 'e'), ('a', 'c'), ('b', 'b'), ('e', 'e'), ('c', 'f'), ('d', 'd'), ('f', 'f'), ('a', 'f'), ('f', 'a'), ('e', 'b'), ('c', 'a'), ('d', 'f'), ('b', 'e'), ('a', 'a'), ('f', 'c'), ('f', 'd'), ('c', 'c'), ('e', 'c')}
I = {('b', 'f'), ('c', 'd'), ('e', 'd'), ('b', 'a'), ('a', 'e'), ('a', 'd'), ('f', 'b'), ('d', 'c'), ('d', 'e'), ('c', 'b'), ('e', 'f'), ('b', 'c'), ('f', 'e'), ('a', 'b'), ('e', 'a'), ('b', 'd'), ('d', 'b'), ('d', 'a')}
FNF = (adb)(cb)(c)(fe)

Graph (format .dot)

```
digraph G {
  0 -> 1;
  1 -> 3;
  2 -> 4;
  3 -> 4;
  3 -> 7;
  5 -> 6;
  6 -> 7;
  0[label="a"];
  1[label="c"];
  2[label="d"];
  3[label="c"];
```

```
  4[label="f"];
  5[label="b"];
  6[label="b"];
  7[label="e"];
}
```