Klasa problemów #P

Paweł Gora

Agenda

- Problemy klasy #P
- Problemy #P-zupełne
- Przykład problemu #PC: zliczanie roszerzeń liniowych
- Przykładowe algorytmy zliczania rozszerzeń liniowych

Klasa poblemów #P

Klasa #P – klasa problemów zliczania związanych z problemami decyzyjnymi z klasy NP

Przykłady:

NP	**#P**
Czy istnieje w grafie cykl Hamiltona o koszcie mniejszym niż 100?	Ile istnieje w grafie cykli Hamiltona o koszcie mniejszym niż 100?
Czy istnieje wartościowanie spełniające formułę CNF?	Ile istnieje wartościowań spełniającyh formułę CNF?
Czy istnieje doskonałe skojarzenie w grafie dwudzielnym?	Ile istnieje doskonałych skojarzeń w grafie dwudzielnym?

Klasa problemów #P-zupełnych

Klasa #P-complete – klasa problemów z #P takich, że każdy problem z #P da się do nich zredukować w czasie wielomianowym.

Przykłady:

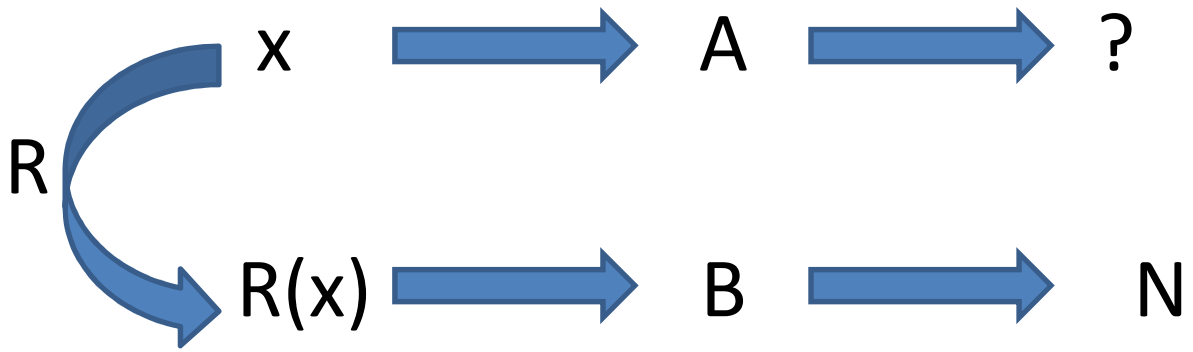
- **#SAT** (ile jest wartościowań spełniających zadaną formułę?)
- **#HAMILTON_PATH** (ile jest ścieżek Hamiltona w grafie?)
- **#PERMANENT** (ile jest idealnych skojarzeń w grafie dwudzielnym?)

Dowodzenie przynależności do #PC

Redukcja problemu A do problemu B:

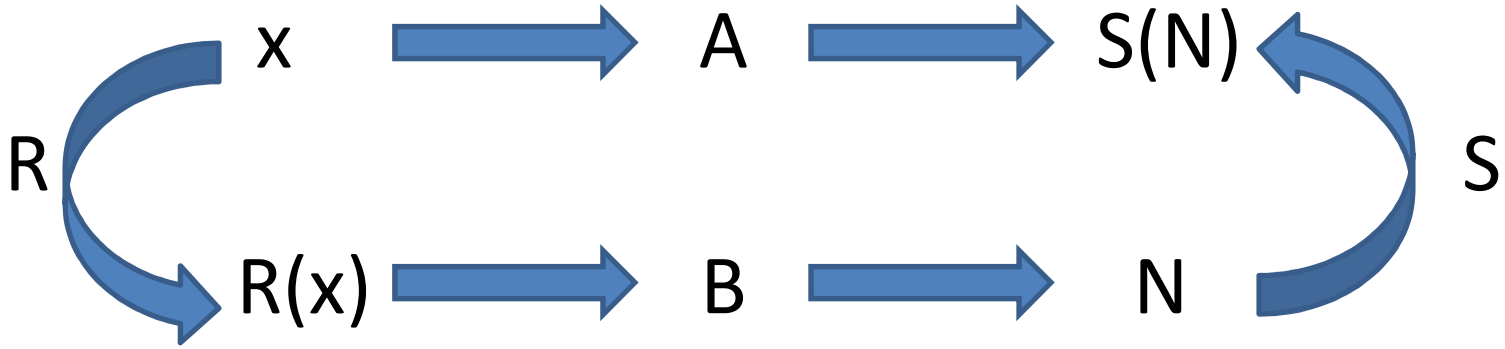
Dowodzenie przynależności do #PC

Redukcja problemu A do problemu B:



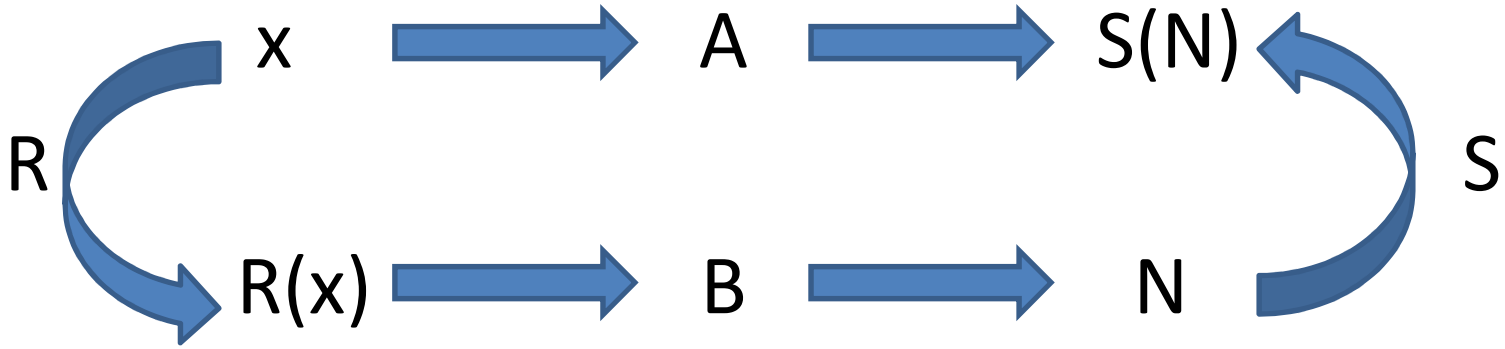
Dowodzenie przynależności do #PC

Redukcja problemu A do problemu B:



Dowodzenie przynależności do #PC

Redukcja problemu A do problemu B:



Jeśli S jest funkcją identycznościową, to jest to **redukcja oszczędna**.

Trudność problemów z #P i #PC

- Problemy z klasy #P są co najmniej tak trudne jak odpowiadające im problemy z klasy NP

- Jeżeli istnieje wielomianowy algorytm dla problemu z #PC, to P = NP (bo pokazano, że #SAT jest w klasie #PC). Taki algorytm nie jest jeszcze znany ... ☺

Pytanie kontrolne

$\#P_{NPC}$ - klasa problemów z #P, które odpowiadają problemom z NPC.

Czy $\#PC = \#P_{NPC}$?

Pytanie kontrolne

$\#P_{NPC}$ - klasa problemów z #P, które odpowiadają problemom z NPC.

Czy $\#PC = \#P_{NPC}$?

NIE!!

Problemy z #PC

Do klasy #PC należą też problemy, dla których odpowiedni problem decyzyjny posiada rozwiązanie wielomianowe.

Przykład: Problem istnienia idealnego skojarzenia w grafie dwudzielnym

Ważny przykład problemu z #PC

Problem znajdowania ilości rozszerzeń liniowych zbioru częściowo uporządkowanego *(posetu*).

Poset = (P, ≤), |P| = n

Relacja ≤ jest:

1. Zwrotna: a ≤ a dla a z S
2. Antysymetryczna: jeśli a ≤ b i b ≤ a, to a = b
3. Przechodnia: jeśli a ≤ b i b ≤ c, to a ≤ c

Poset

- Elementy x, y są **porównywalne**, jeżeli $x \leq y$ lub $y \leq x$
- Wpp x,y – **nieporównywalne** (x || y)
- **Porządek liniowy**: każde 2 elementy P są porównywalne
- **Łańcuch** w P: podziór P będący porządkiem liniowym
- **Antyłańcuch** w P: podzbiór P, w którym wszystkie elementy są nieporównywalne
- **Szerokość** P: rozmiar największego antyłańucha w P (w(P))

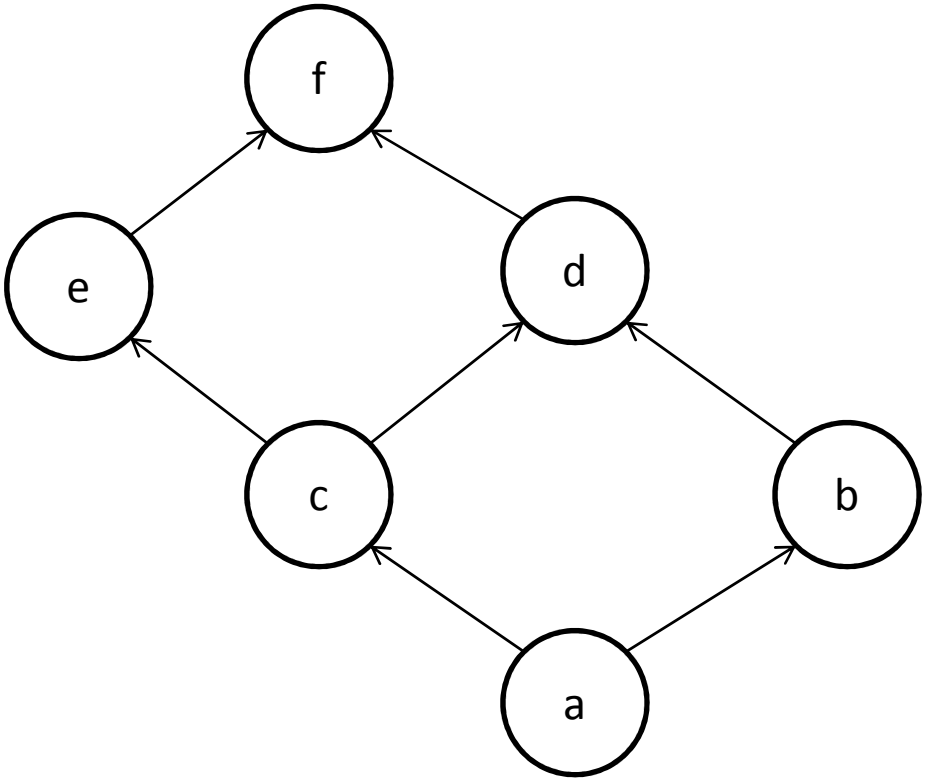
Rozszerzenia liniowe posetu

- Poset $(P,\leq_1)$ jest **rozrzeszeniem** $(P,\leq)$, gdy dla x,y ze zbioru P $x \leq y$ implikuje $x \leq_1 y$

- **Rozszerzenie liniowe** = rozszerzenie będące łańcuchem

Diagram Hassego

- Przykład diagramu Hassego:



Problemy związane z posetami

- Jak znajdować wszystkie rozszerzenia liniowe?
- Ile jest wszystkich rozszerzeń liniowych?
- Ile jest roszerzeń liniowych, w których x jest na pozycji ustalonej pozycji?
- Ile jest rozszerzeń liniowych, w których x jest przed y?

Pierwszy problem da się rozwiązać w zamortyzowanym czasie stałym (O(e(P)), gdzie e(P) – ilość rozszerzeń liniowych)

Ostatnie 3 problemy należą do klasy #PC.

Potrzebne pojęcia

- Poset P jest **kratą**, jeżeli dla dowolych x, y z P istnieje inf{x, y} = x ∧ v oraz sup{x, y} = x v y.
- **Ideał**: podzbiór D posetu P:

$$x \in D, y \in P, y \leq x \Rightarrow x \in D$$

- **Filtr**: podzbiór U posetu P:

$$x \in U, y \in P, x \leq y \Rightarrow y \in U$$

Zliczanie rozszerzeń liniowych

- Algorytm dynamiczny

Poset **(P, ≤)** o szerokości *k*. Z tw. Dilwortha da się go pokryć *k* rozłącznymi łańcuchami:

$$x_{1,1} < x_{1,2} < \ldots < x_{1,n_1}$$

$$x_{2,1} < x_{2,2} < \ldots < x_{1,n_2}$$

$$\ldots\ldots\ldots \ldots\ldots\ldots \ldots\ldots\ldots ..$$

$$x_{k,1} < x_{k,2} < \ldots < x_{1,n_k}$$

Algorytm dynamiczny

- Każdemu ideałowi I można jednoznacznie przyporządkować krotkę:

$$I = \bigcup_{i=1}^{k} \{x_{i,j} : j < p_i\} \rightarrow (p_1, p_2, \ldots, p_k)$$

- Różne ideały są reprezentowane przez różne krotki, np krotka (0,0,...,0) reprezentuje ideał pusty.

Algorytm dynamiczny

- $e[p_1,p_2,...,p_k]$ – ilość rozszerzeń liniowych ideału reprezentowanego przez tą krotkę

- $e[0,0,...,0] = 1$

- Jeśli $(p_1,p_2,...,p_k)$ nie reprezentuje ideału, to

$$e[p_1,p_2,...,p_k] = 0$$

- Jeśli $(p_1,p_2,...,p_k)$ reprezentuje ideał, to

$$e[p_1,p_2,...,p_k] = e[p_1-1,p_2,...,p_k] + e[p_1,p_2-1,...,p_k] + ... + e[p_1,p_2,...,p_k-1]$$

Algorytm dynamiczny

```
for all 0 ⩽ i_1 ⩽ n_1, ..., 0 ⩽ i_k ⩽ n_k do e[i_1, ..., i_k] := 0
e[0, ..., 0] := 1
fifo := ∅
fifo ← (0, ..., 0)
while fifo ≠ ∅ do begin
    fifo → (p_1, ..., p_k)
    for i := 1 to k do begin
        if p_i < n_i then begin
            isDownSet := true
            for j := 1 to k do
                if j ≠ i and p_j < n_j and x_{j,p_j+1} ≺ x_{i,p_i+1} then
                    isDownSet := false
            if isDownSet then begin
                if e[p_1, ..., p_i + 1, ..., p_k] = 0 then
                    fifo ← (p_1, ..., p_i + 1, ..., p_k)
                e[p_1, ..., p_i + 1, ..., p_k] := e[p_1, ..., p_i + 1, ..., p_k] +
                                                  + e[p_1, ..., p_i, ..., p_k]
            end
        end
    end
end
return e[n_1, ..., n_k]
```

Algorytm dynamiczny

- Na koniec obliczeń:

$$e[n_1, n_2, \ldots, n_k] = \text{liczba rozszerzeń liniowych}$$

- Wszystkich krotek jest co najwyżej:

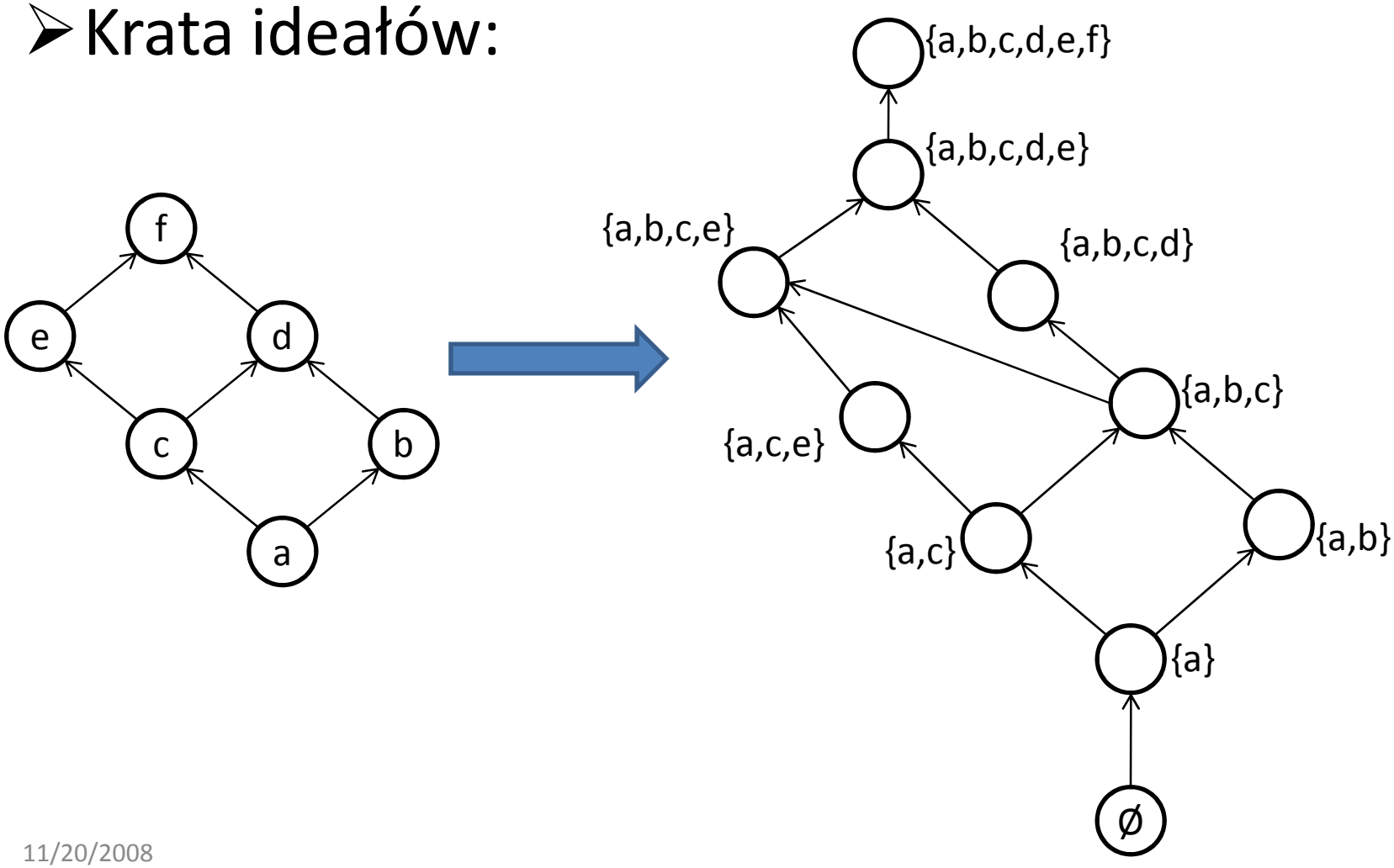
$$\prod_{i=1}^{k}(n_i+1) \leqslant \left(\frac{\sum_{i=1}^{k}(n_i+1)}{k}\right)^k = \frac{(n+k)^k}{k^k}$$

Algorytm dynamiczny

- Ilość operacji na przetworzenie krotki: k^2
- Łączna ilość operacji algorytmu: $k^{2-k}(n+k)^k$
- Dla ustalonego *k* złożoność: $\mathrm{O}(n^k)$
- Dla k = Θ(n) złożoność: $\mathrm{O}(n^2 2^n)$

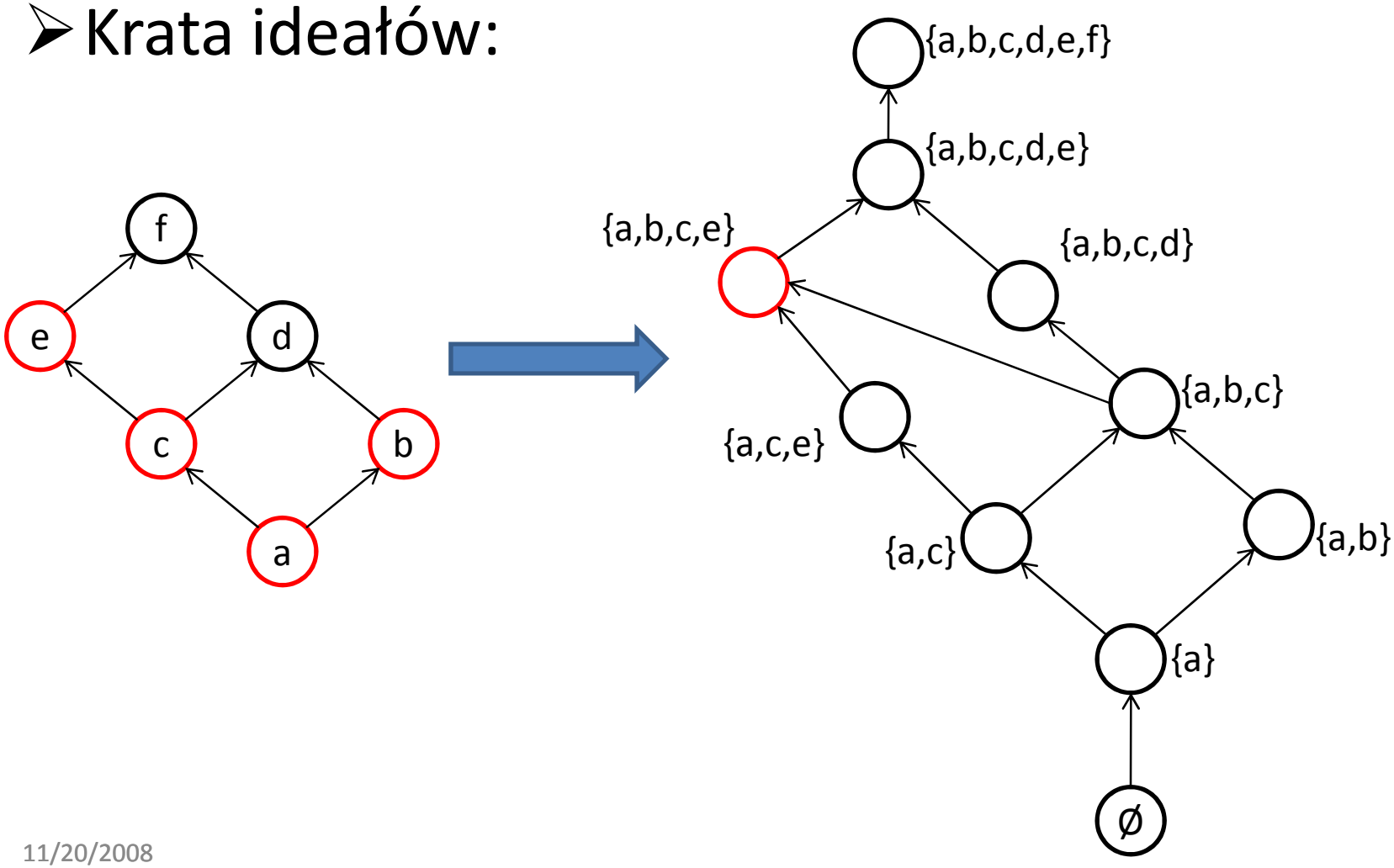
Algorytm II – krata ideałów

- Krata ideałów:



Algorytm II – krata ideałów

- Krata ideałów:



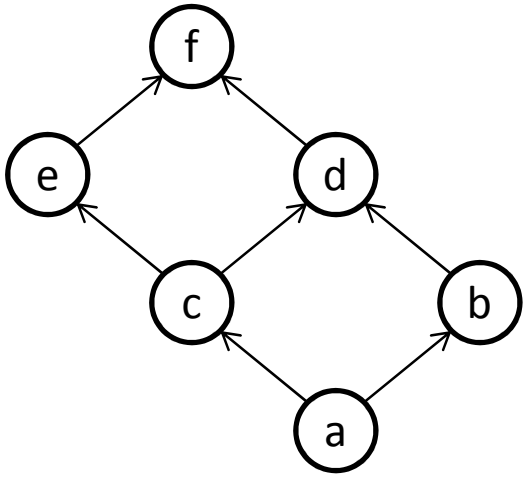
Oznaczenia

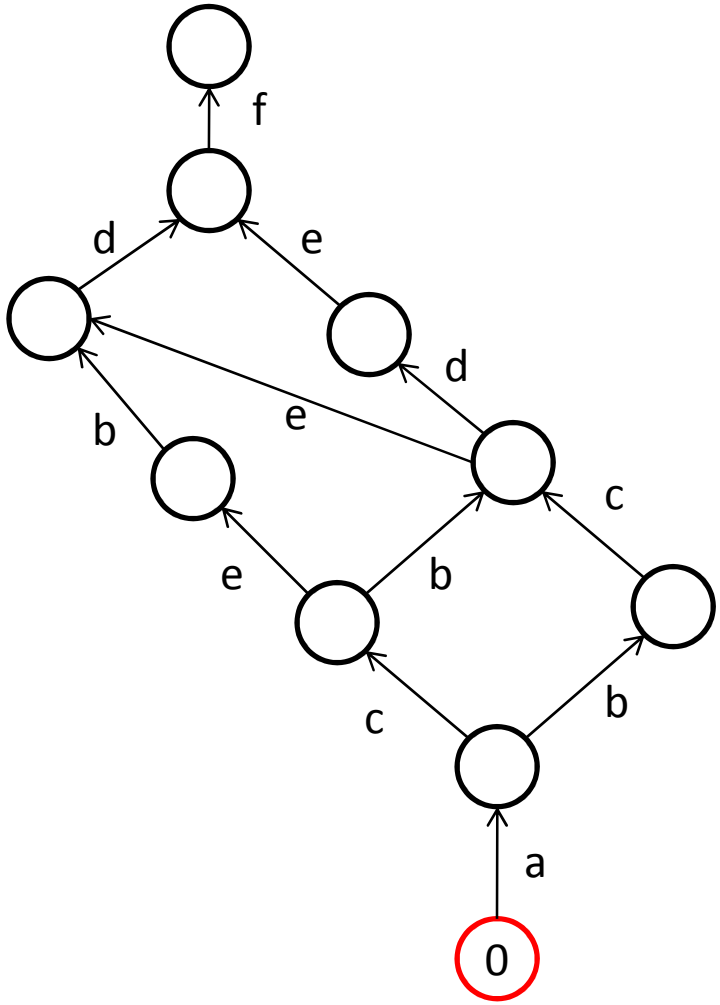
- ImSucc(I) – lista bezpośrednich następników I
- Child(I) – lista bezpośrednich poprzedników I
- LinExtFilter(I) – ilość rozszerzeń liniowych filtru P\I
- VisitedIdeal(I) – flaga: czy I był już odwiedzony przez algorytm

Algorytm

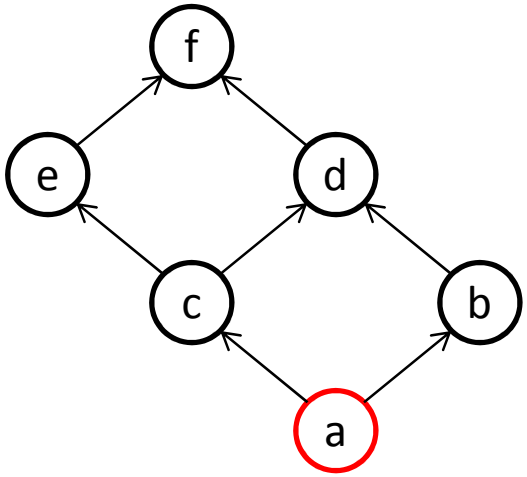
```
Build (Poset P)
    zbuduj kratę ideałów P;
    count <- Assign(Ø);

Assign (Ideal I)
     VisitedIdeal(I) <- true;
     extensions <- 0;
     for each ideal I' in ImSucc(I) do
          if I' = P then extensions <- extensions + 1;
          else
               if not VisitedIdeal(I') then  extensions <- extensions + Assign(I');
               else extensions <- extensions + LinExtFilter(I');
          LinExtFilter(I) <- extensions;
      return extensions;
```

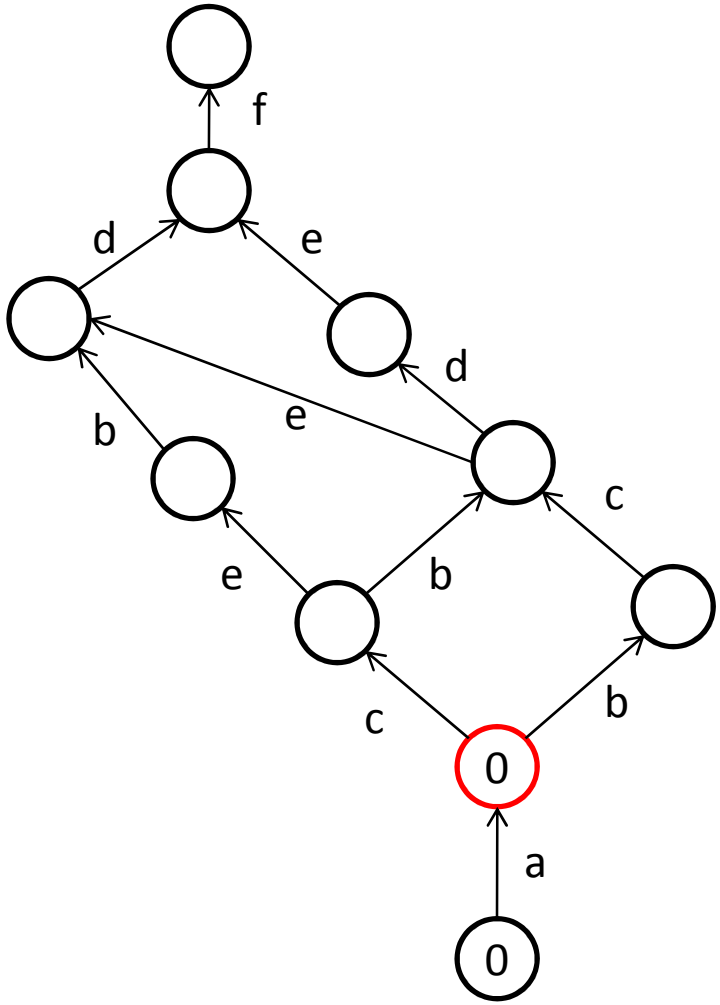
Przykład



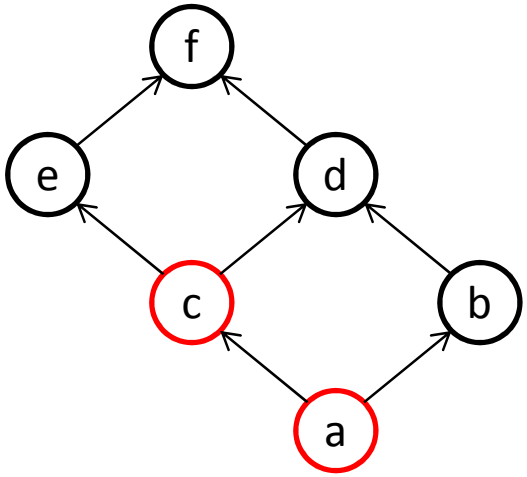


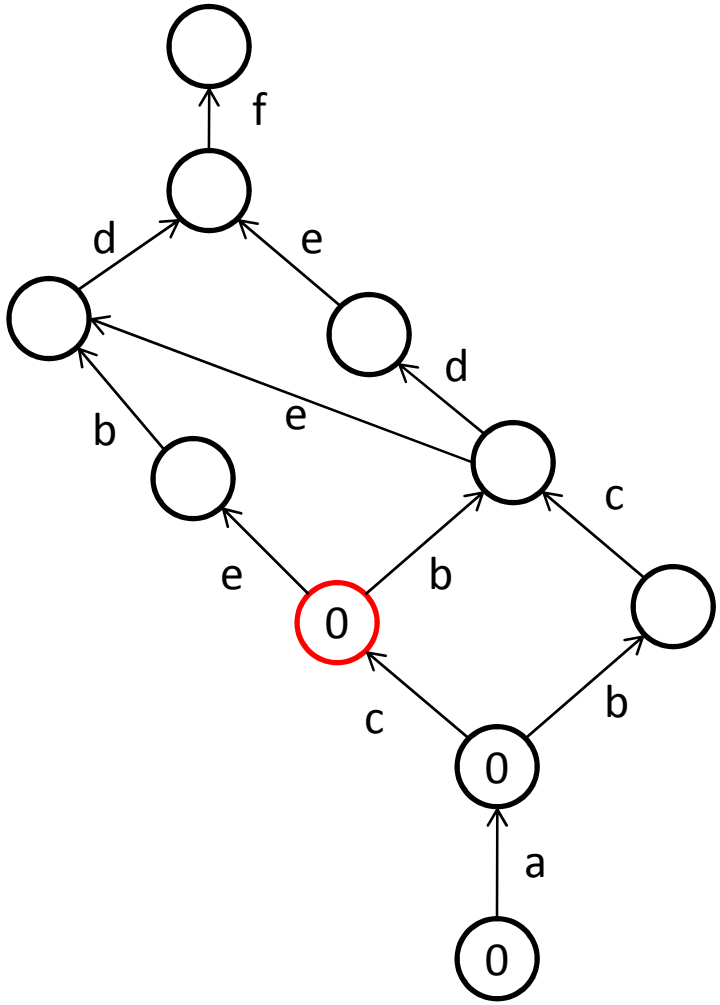
Przykład



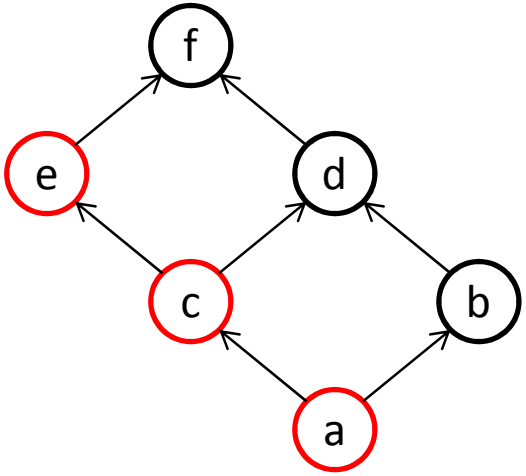


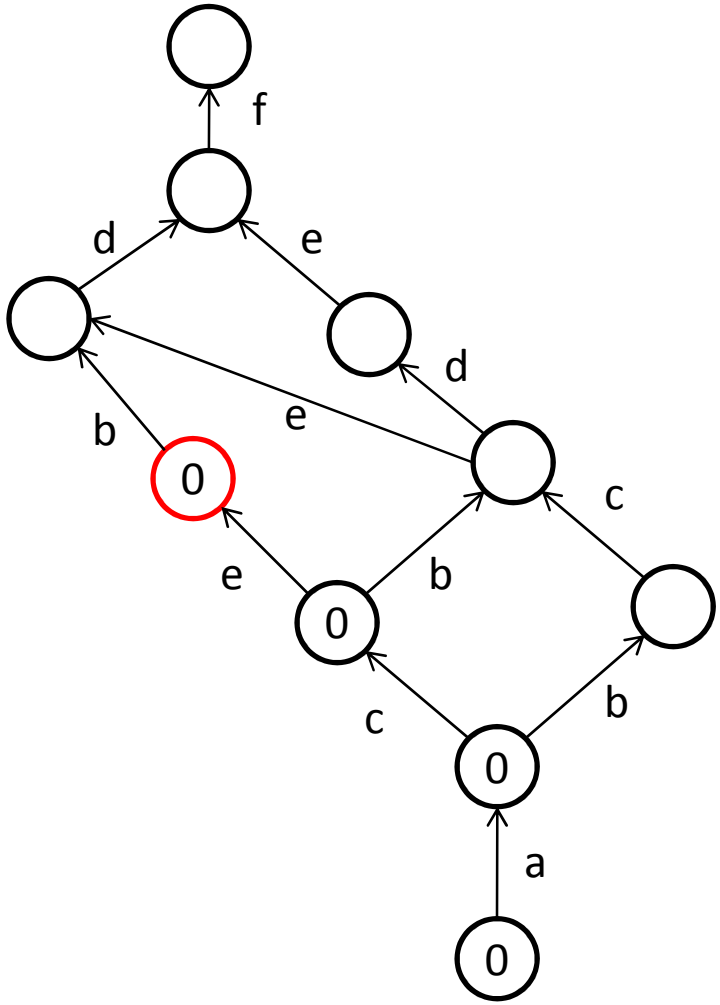
Przykład



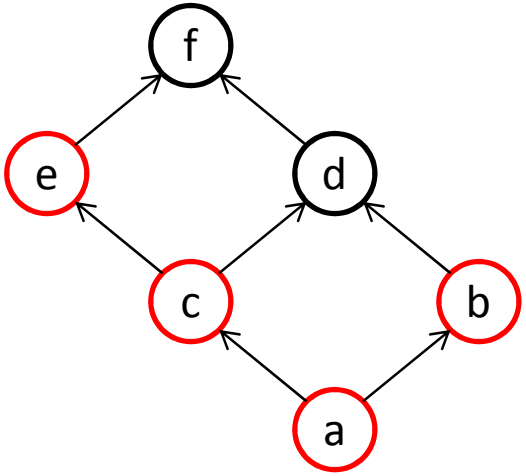


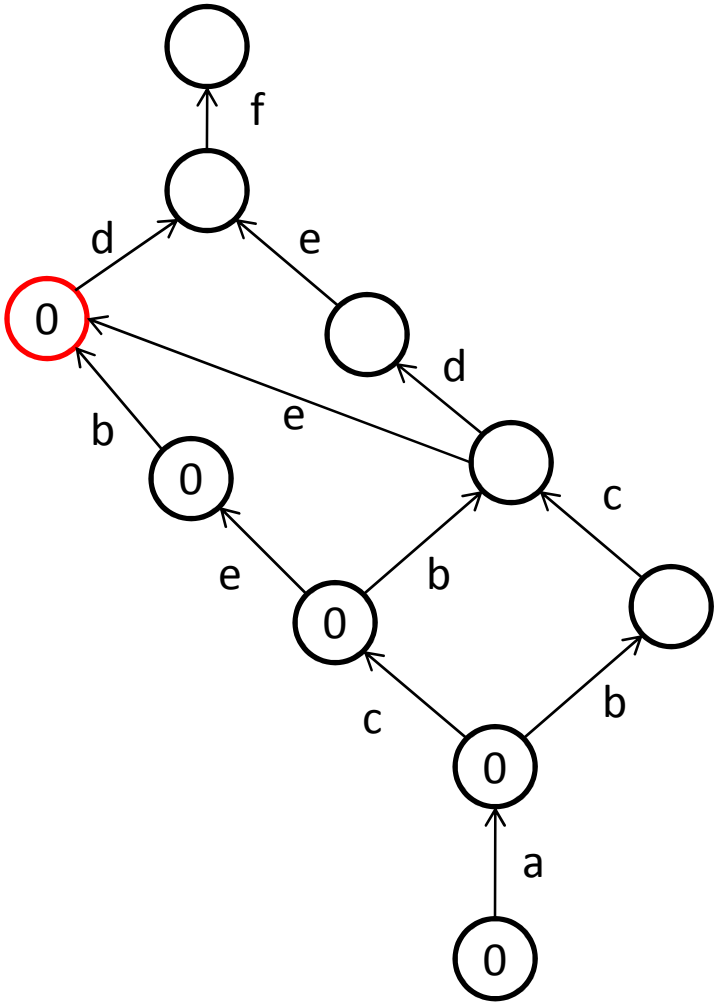
Przykład



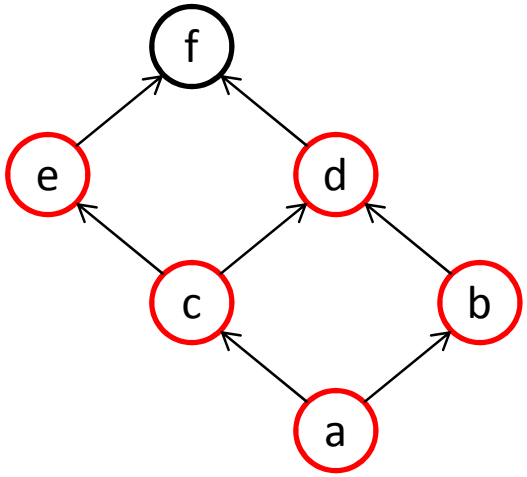


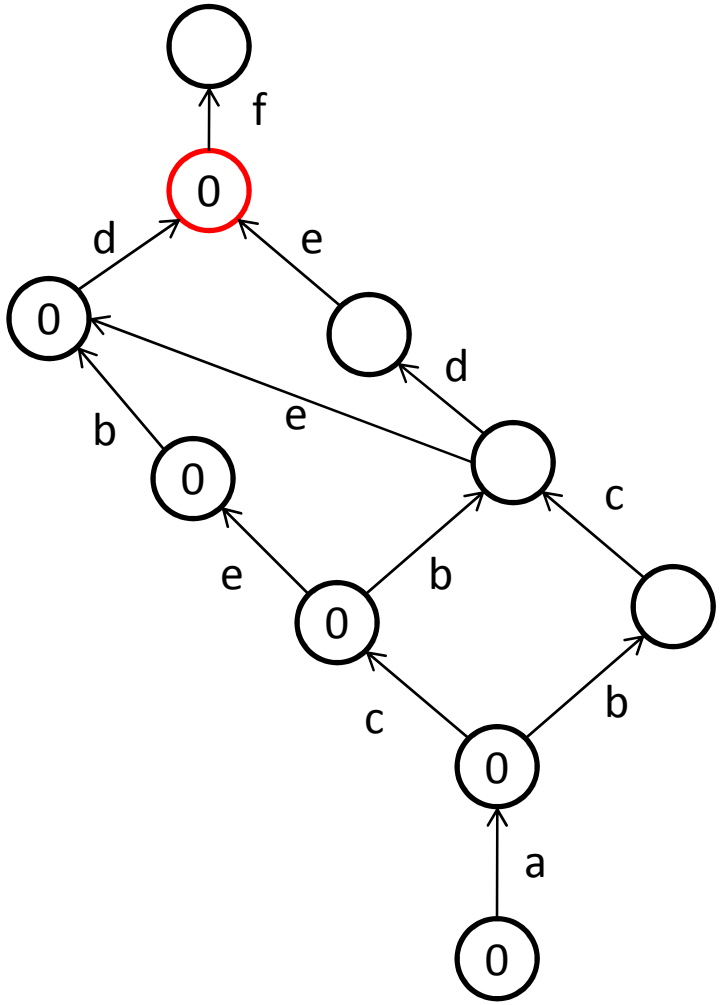
Przykład



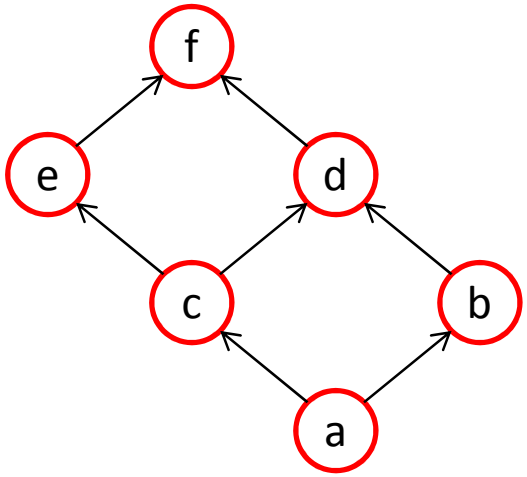


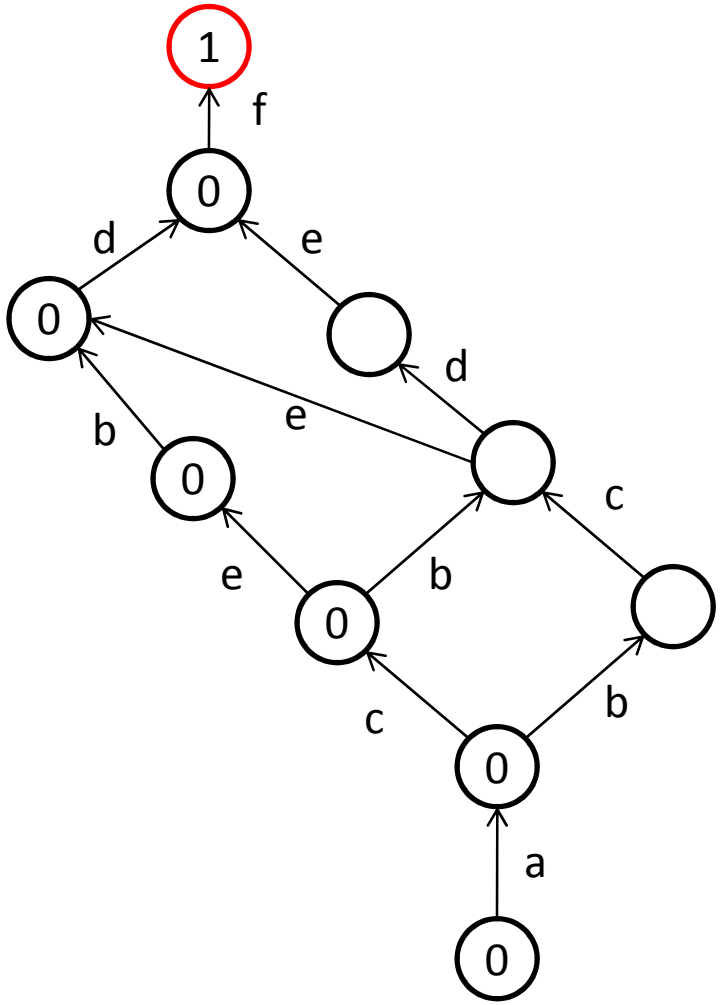
Przykład



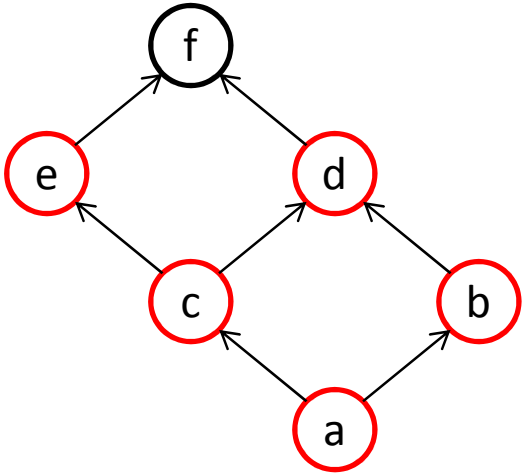


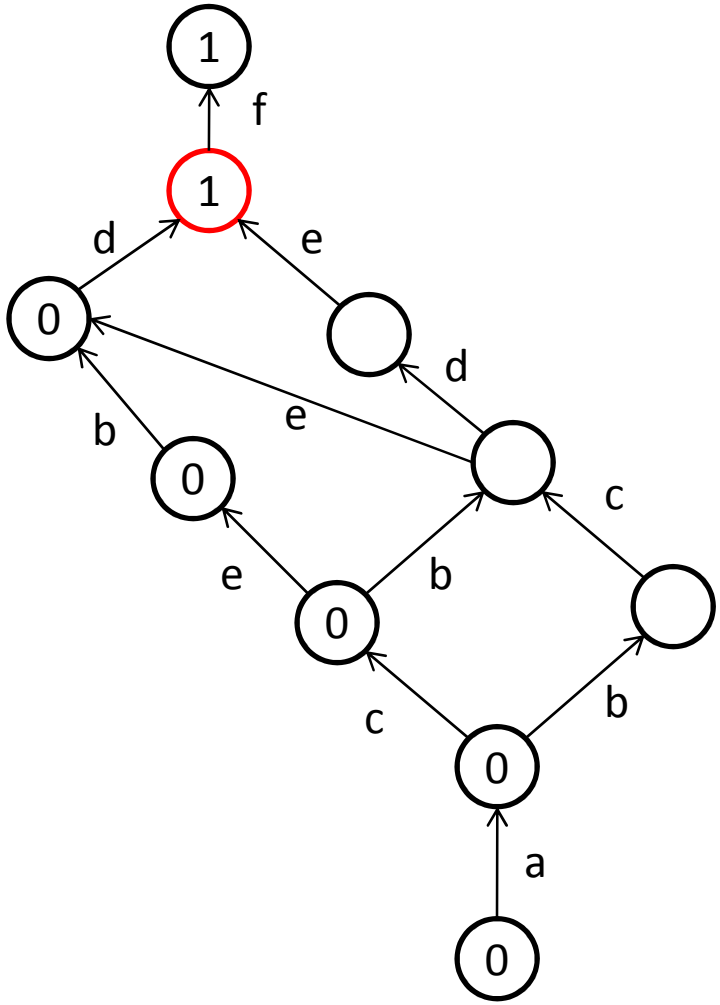
Przykład



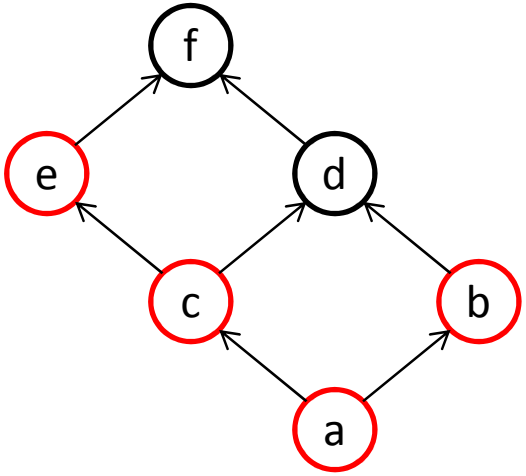


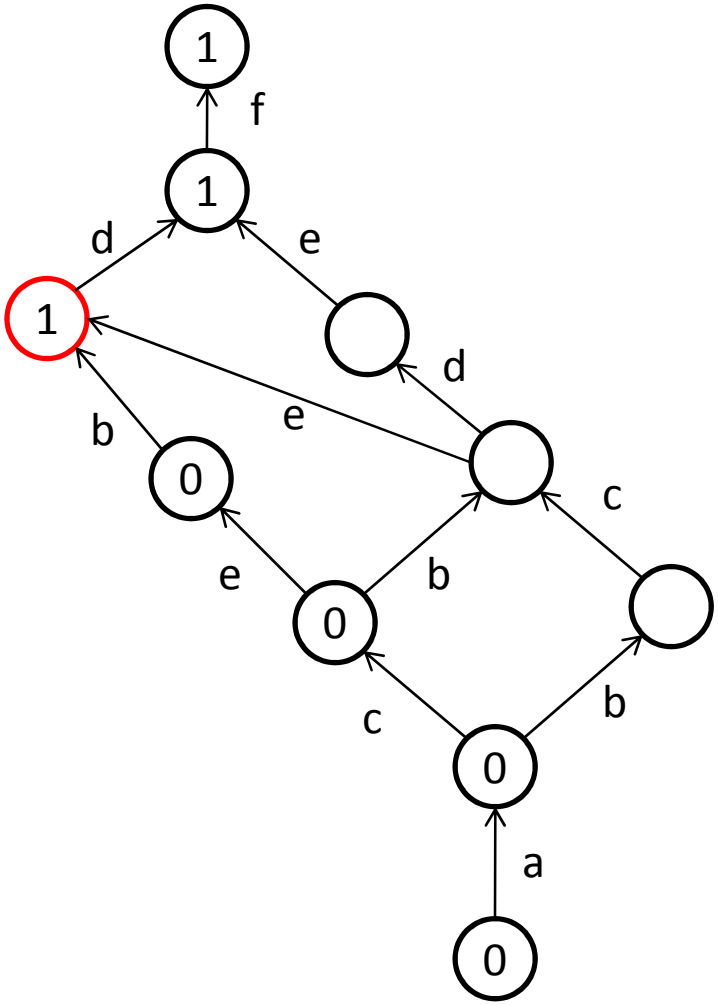
Przykład



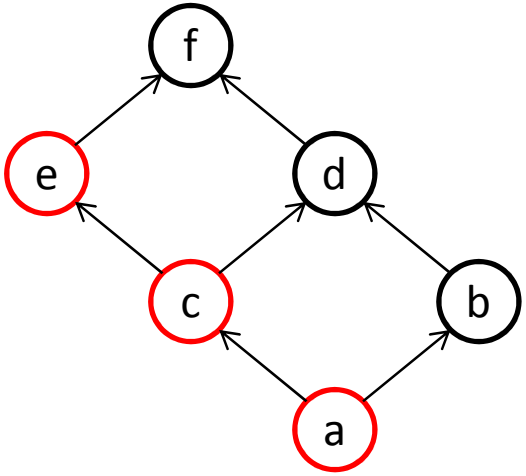


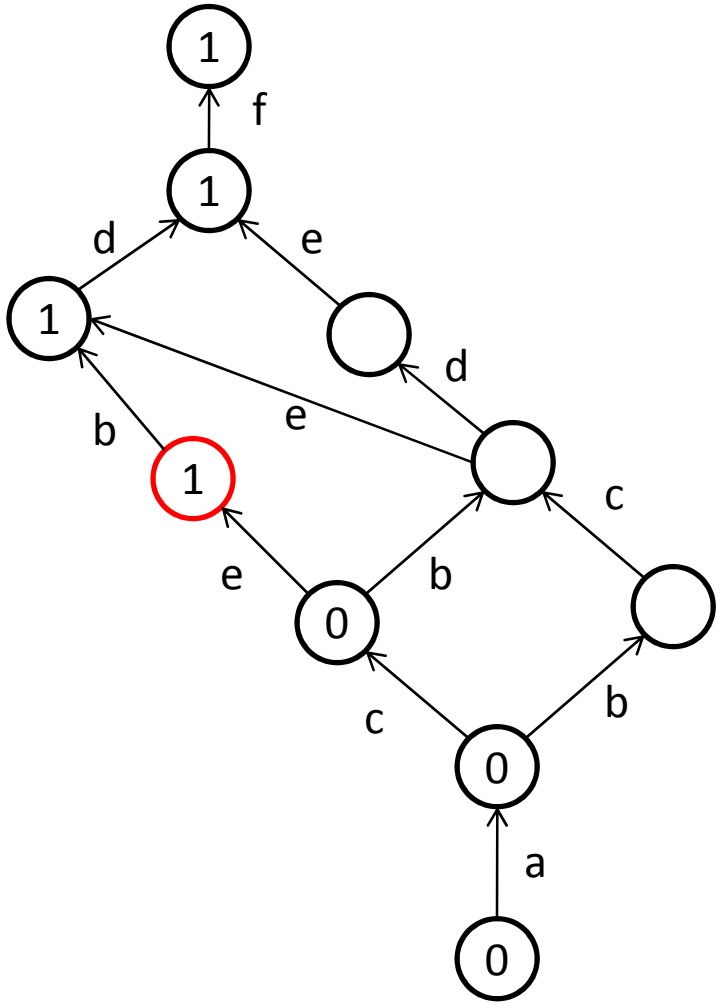
Przykład



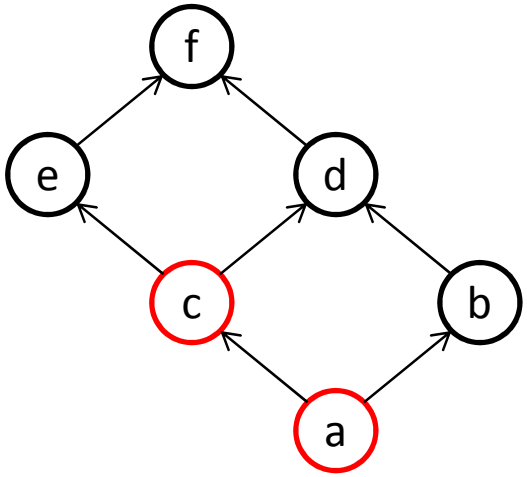


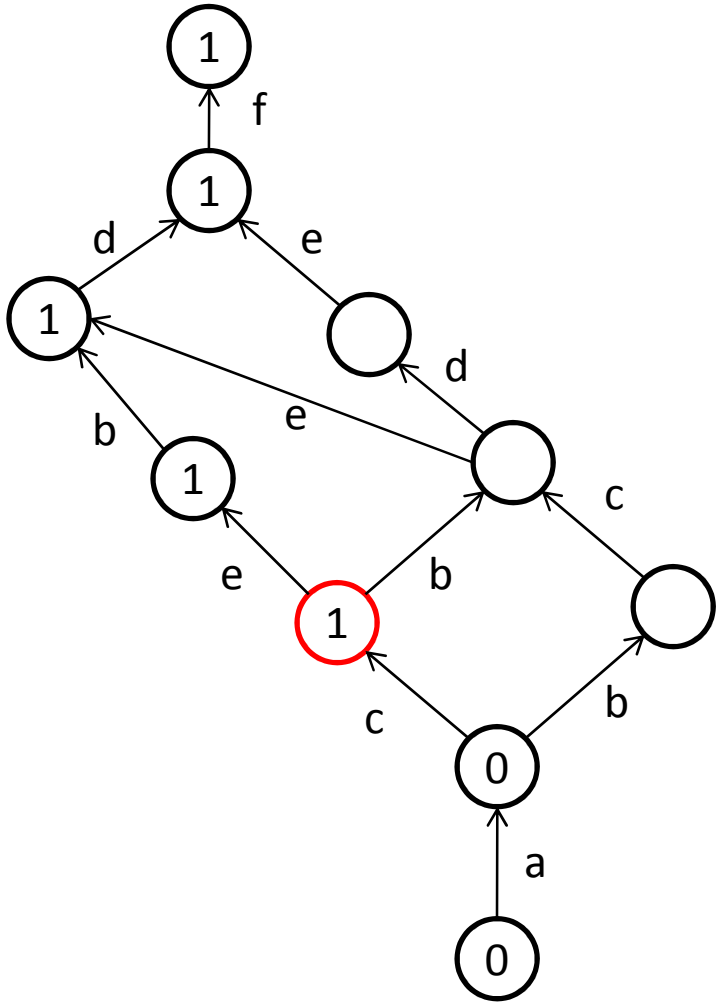
Przykład



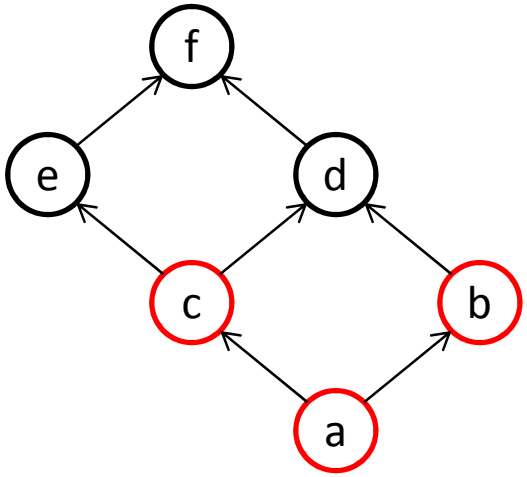


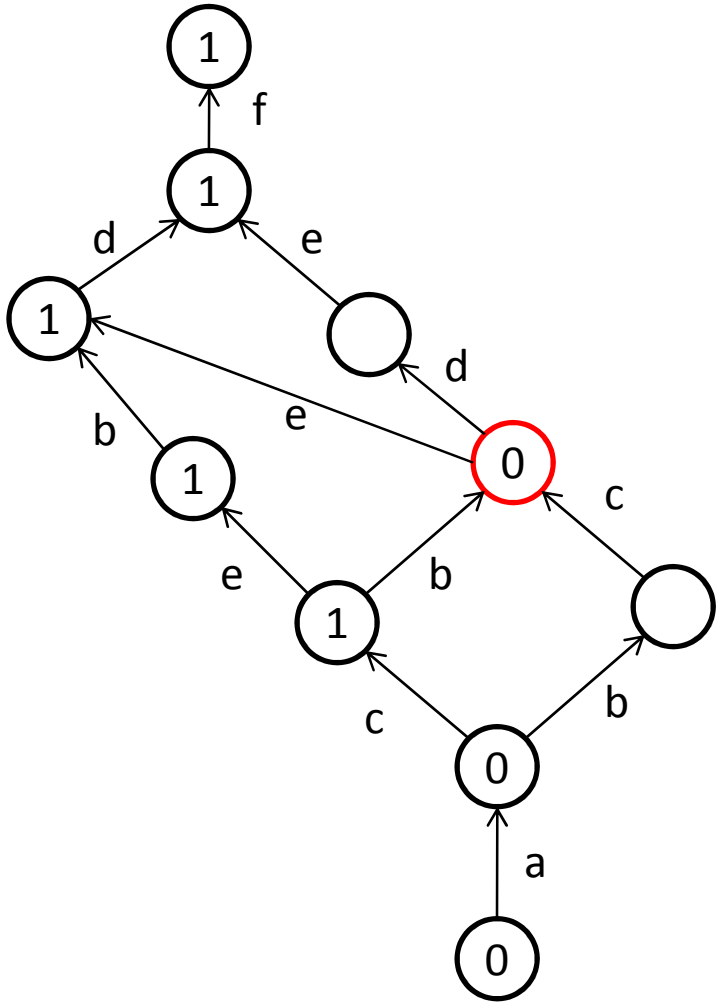
Przykład



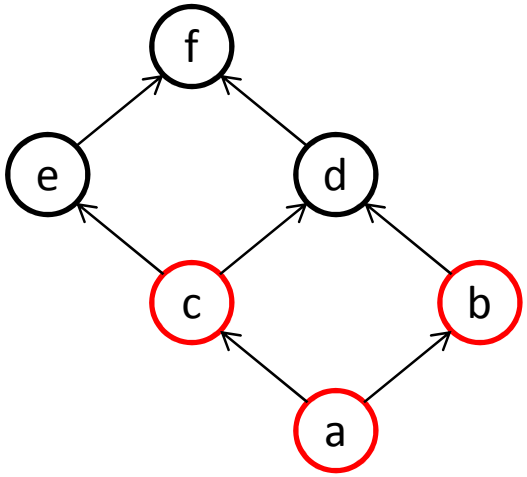


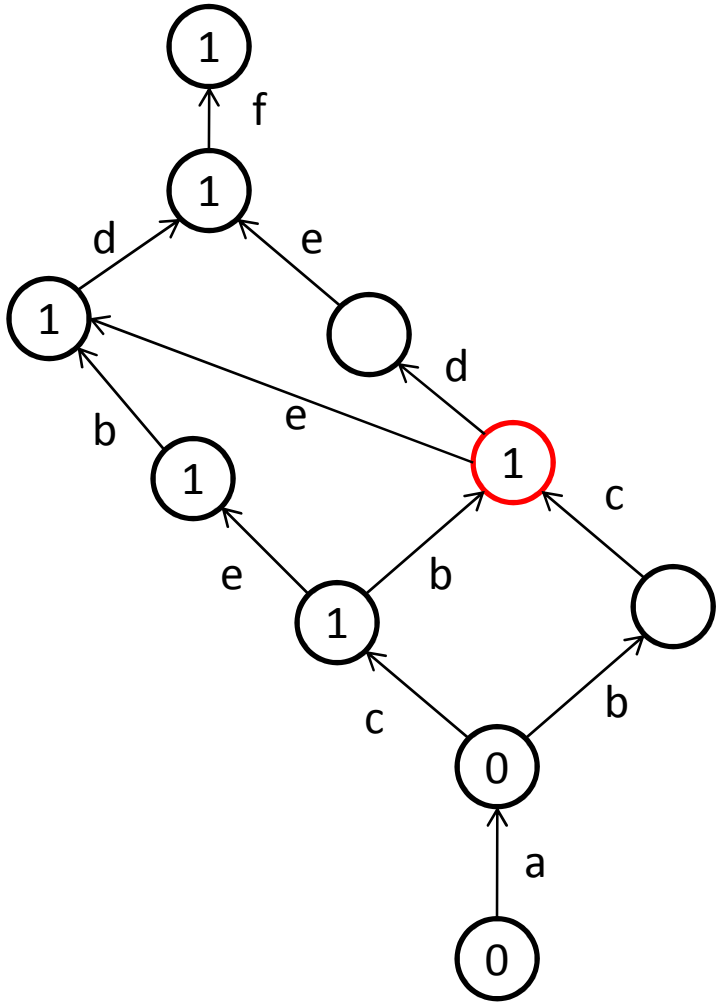
Przykład



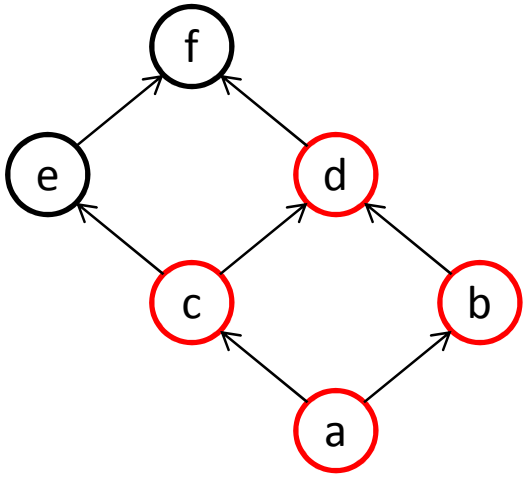


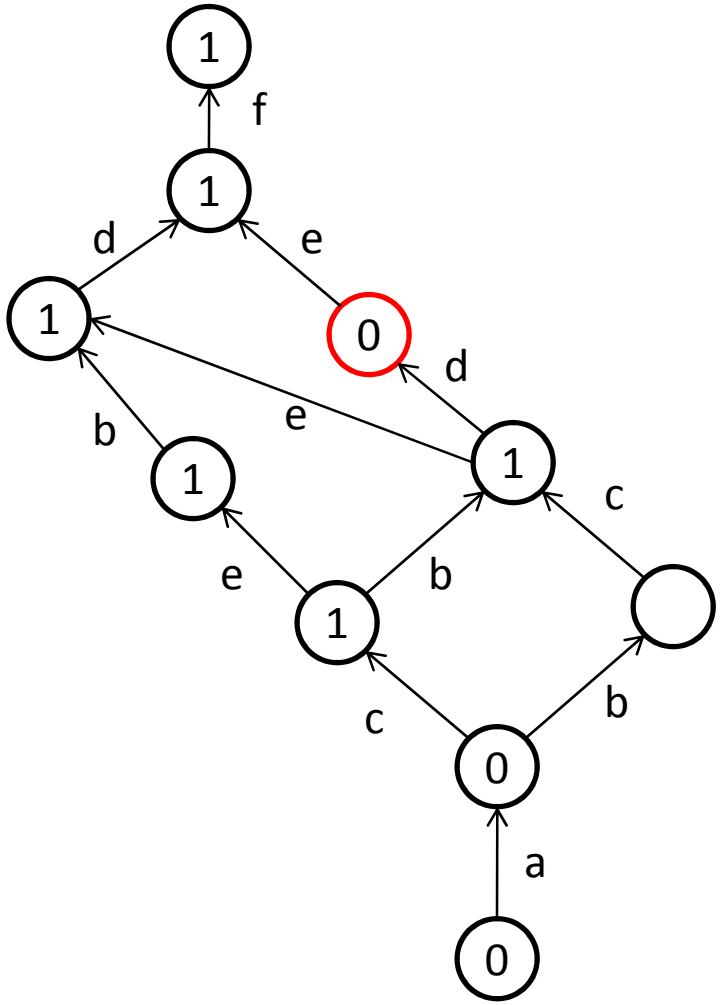
Przykład



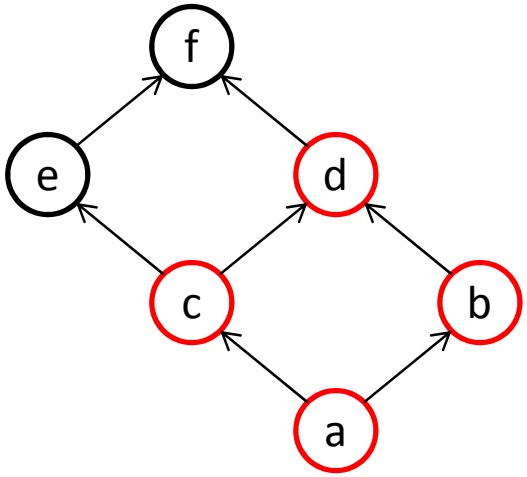


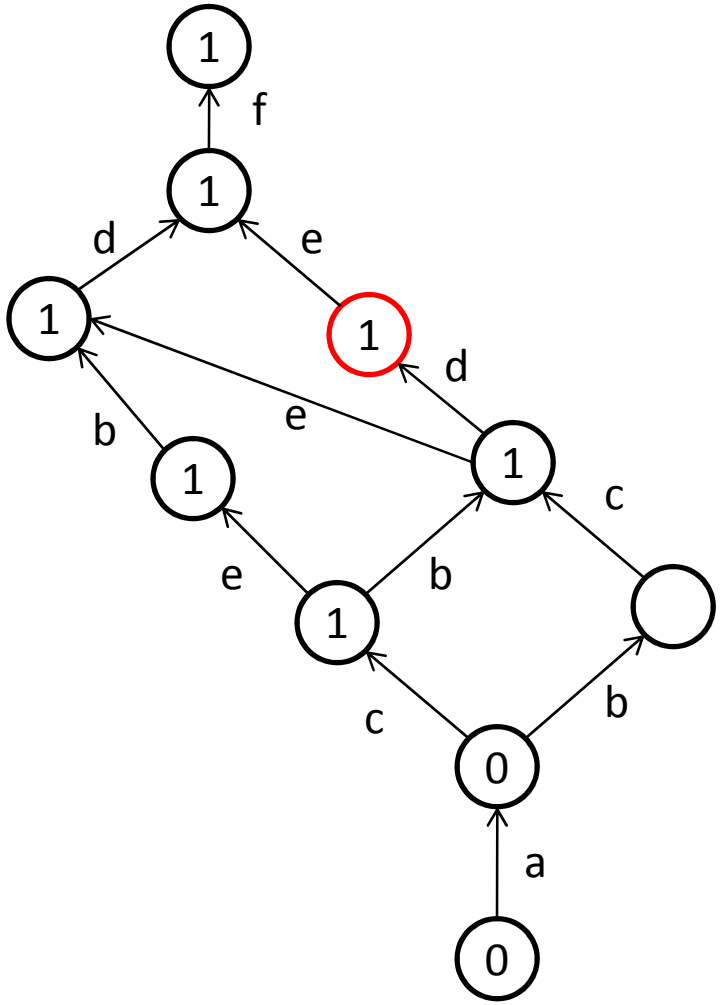
Przykład



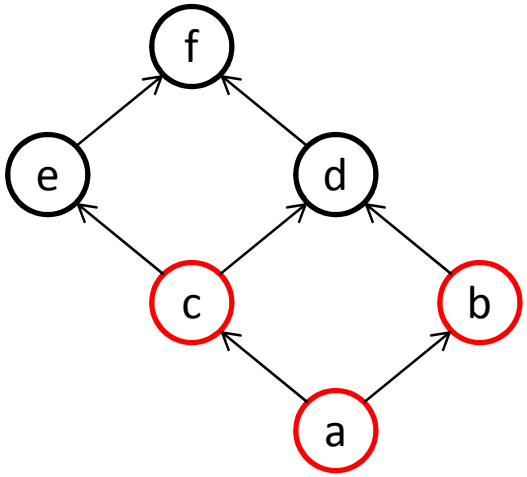


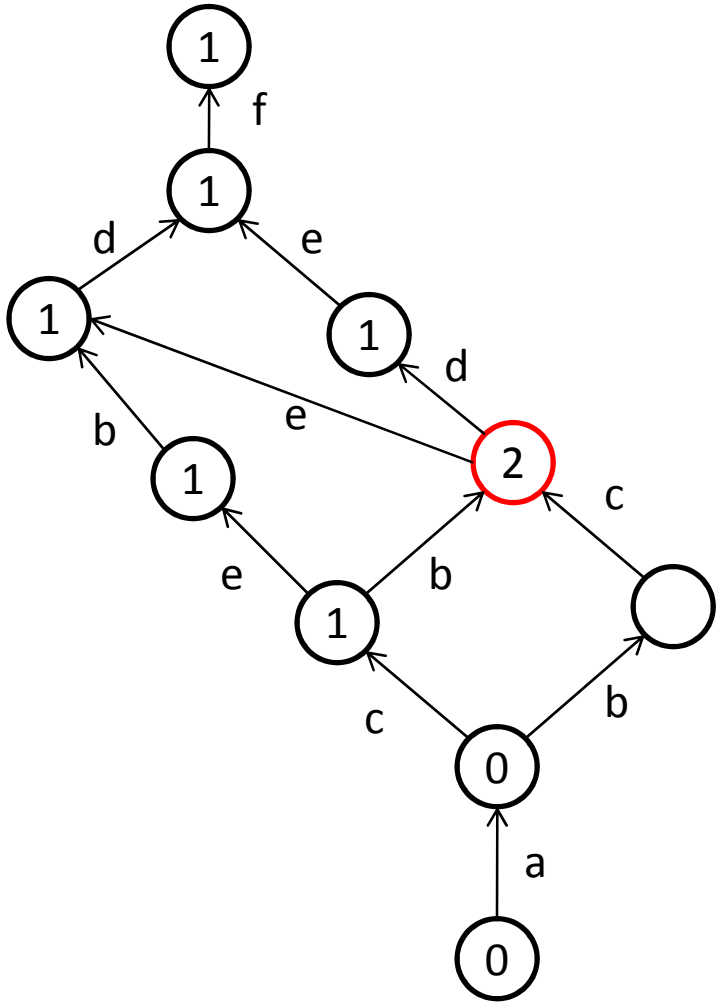
Przykład



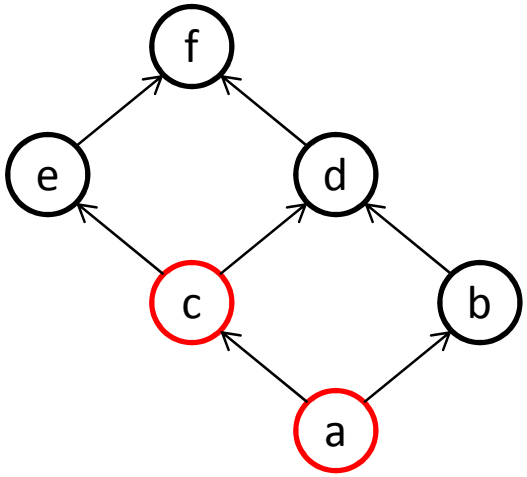


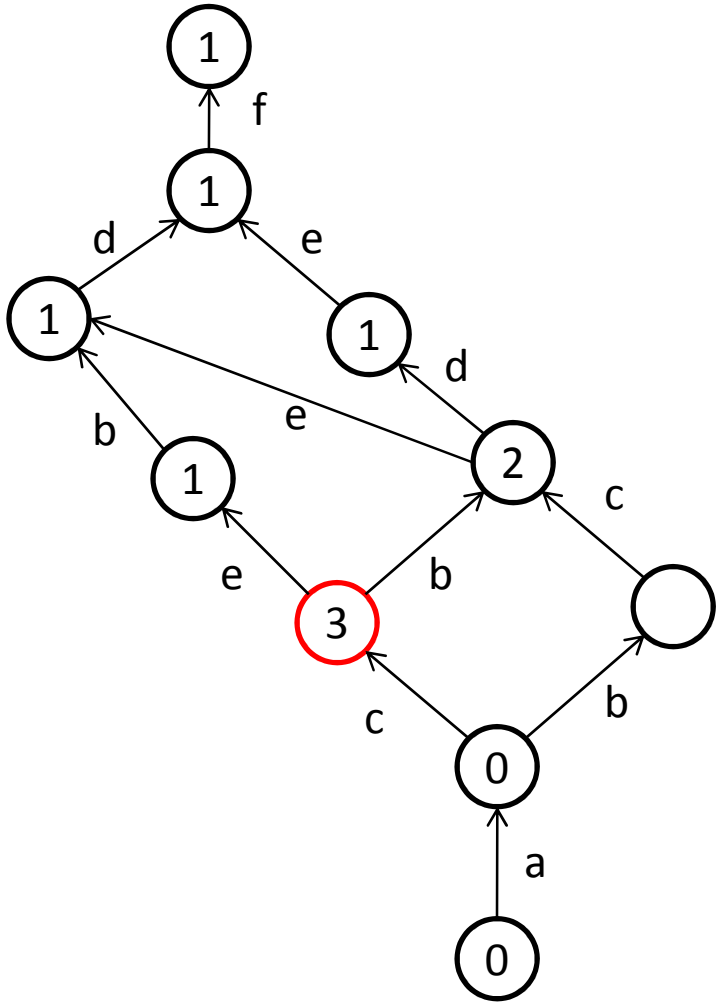
Przykład



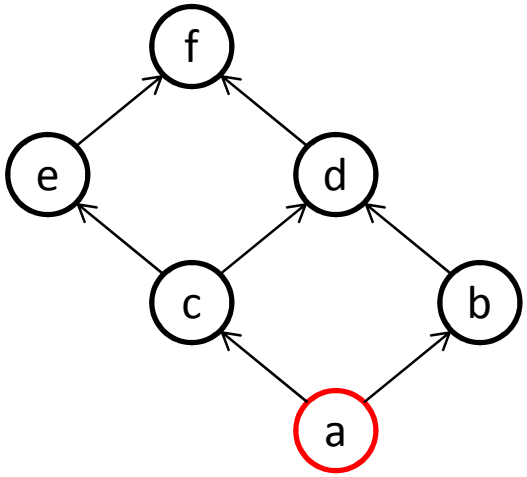


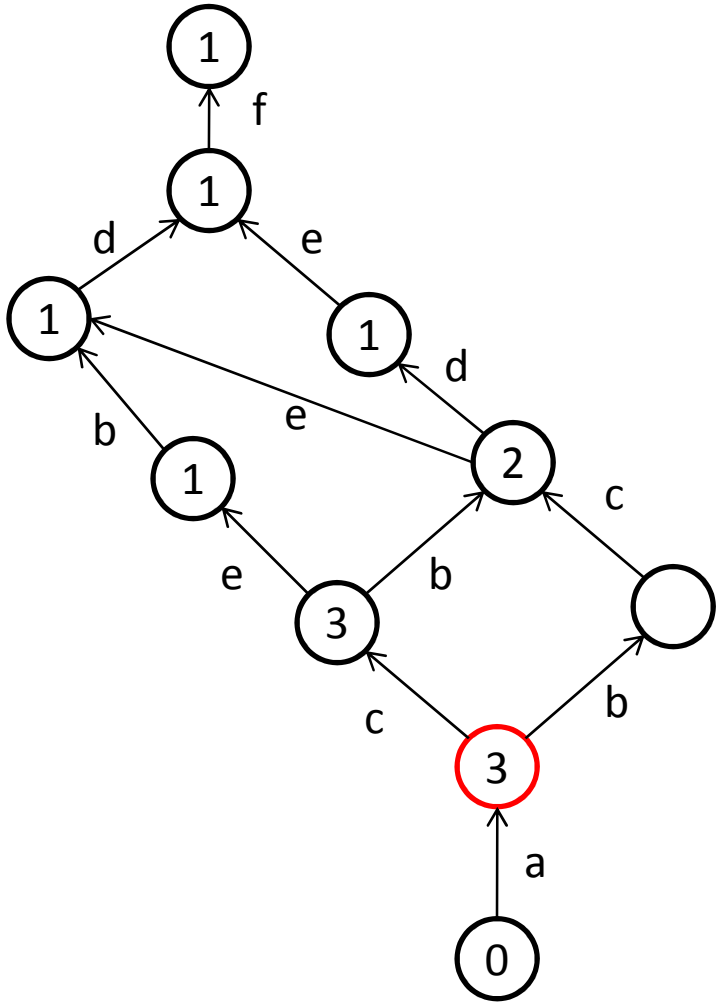
Przykład



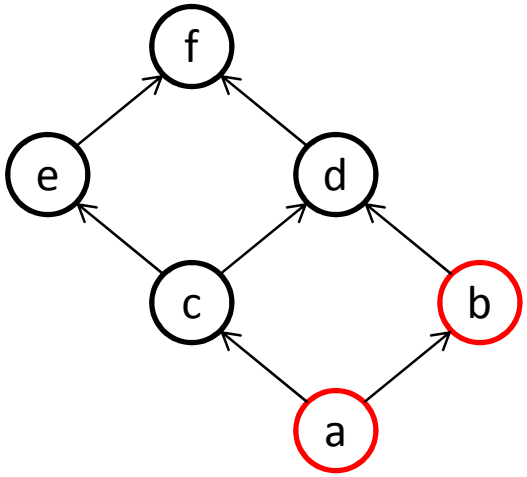


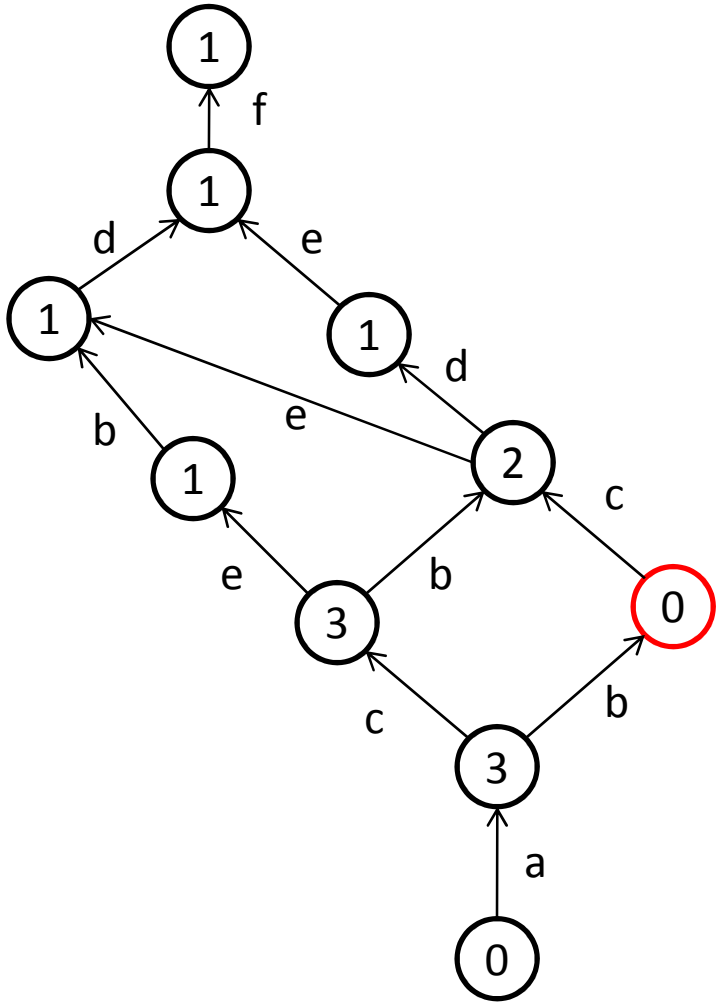
Przykład



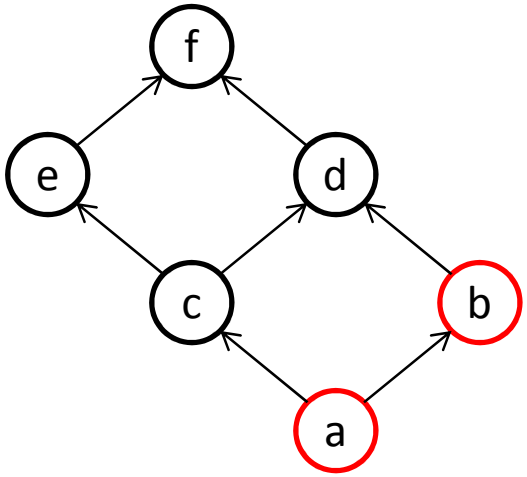


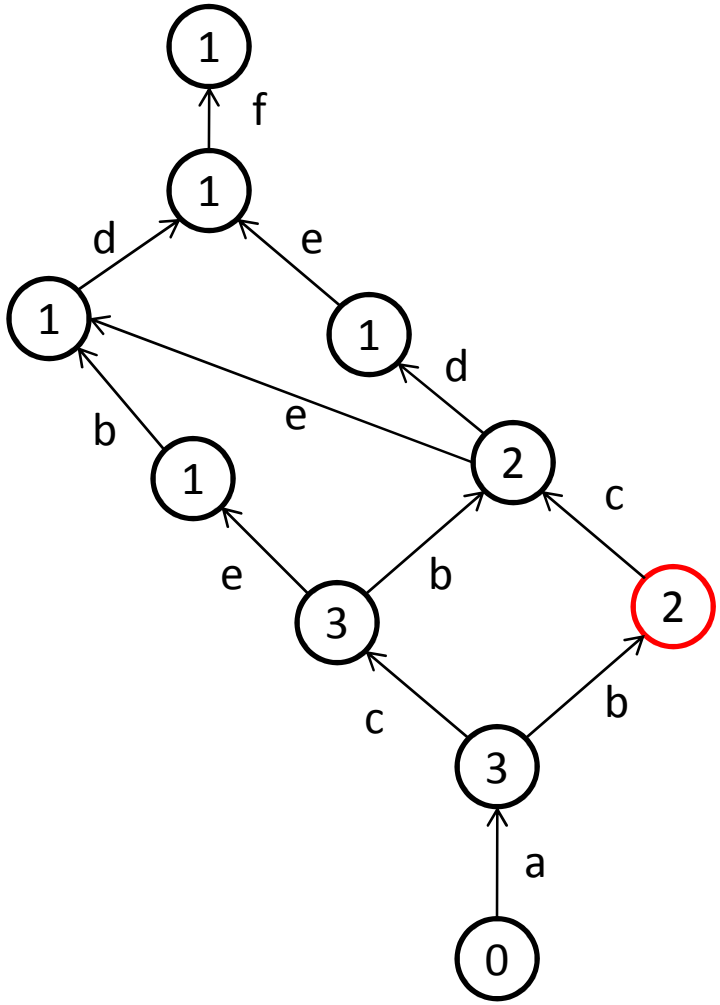
Przykład



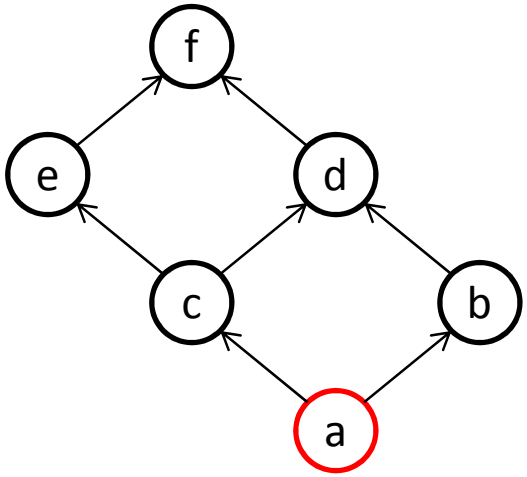


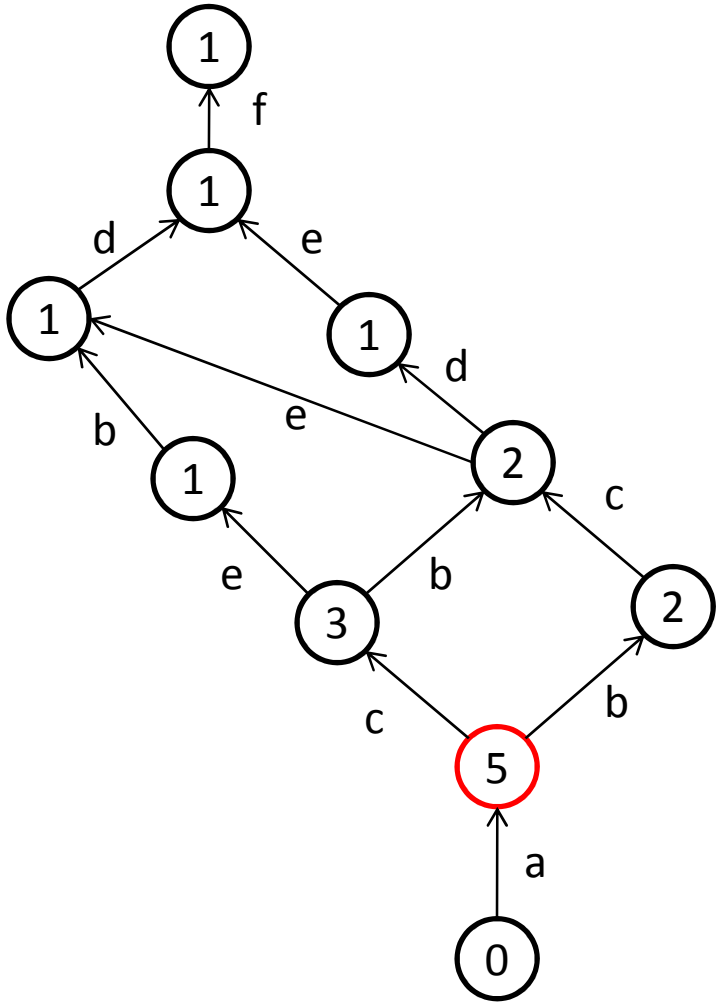
Przykład



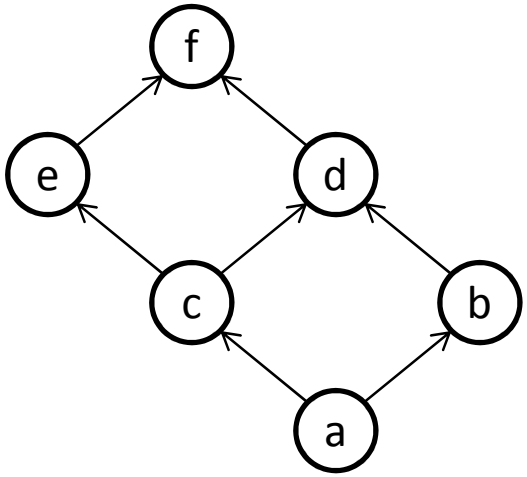


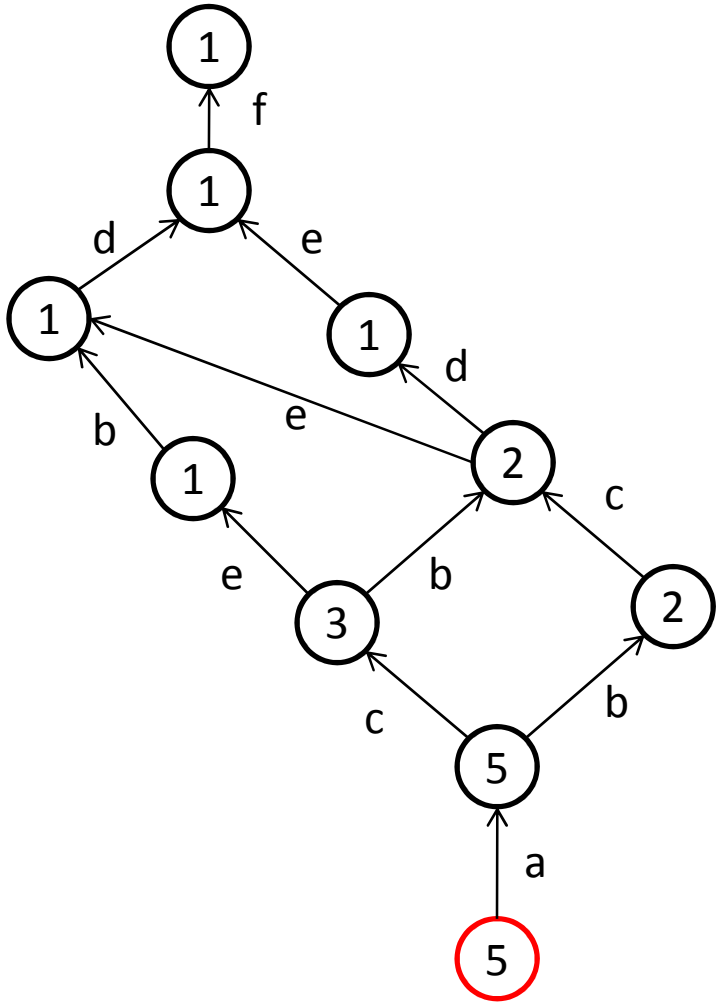
Przykład





Przykład





Znajdowanie rozszerzeń liniowych

- Każde rozszerzenie liniowe odpowiada ścieżce od najmniejszego do największego ideału

Znajdowanie rozszerzeń liniowych

```
Generate (Krata L)
I<-Ø;
E<-"";
while (I ≠ P)
   cumulAssignment  <-  0;
   rand  <-  random number from {1,...,LinExtFilter(I)};
   for each ideal I'  in  ImSucc(I)  do
      cumulAssignment  <-  cumulAssignment  +  LinExtFilter(I');
      if  rand ≤ cumulAssignment  then
          E.add(I'\I);
          I <- I';
          break;
return E;
```

Zliczanie liniowych rozszerzeń

- Pytanie: Ile jest rozszerzeń liniowych, w których element *x* występuje na pozycji *i* ?

Rozwiązanie:

Stosujemy algorytm **Assign** dla porządku odwróconego i otrzymujemy wartości **LinExtIdeal(I)** – ilość liniowych rozszerzeń ideału I.

Zliczanie rozszerzeń liniowych

```
ComputeRanks(Poset P)
   zbuduj kratę ideałów;
   Assign(Ø);
   for each element E in P do
      for each Height in 1,...,|P| do
         Rank(E, Height) <- 0;
    ComputeRank(Ø, 1);
    return Rank;

ComputeRank(Ideal I, Integer Height)
     VisitedIdeal(I) <- true;
     for each ideal I' in ImSucc(I) do
        Rank(I'\I, Height)  += LinExtIdeal(I) * LinExtFilter(I');
     if (I' ≠ P) and not VisitedIdeal(I') then
        ComputeRank(I', Heigh + 1);
```

Zliczanie rozszerzeń liniowych

```
ComputeMutualRank (Ideal I, Integer Height)
Build the ideal lattice of P;
Assign(Ø);
For each element E in P
    For each element F in P do
        MR(E, F) <- 0;
 ComputeMutualRankDFS(Ø,1);
 return MR;
```

```
// MR(E, F) – ilość rozszerzeń, w których
// element F jest przed elementem E
```

```
ComputeMutualRankDFS(Ideal I, Integer Height)
declare a local array HasVisited;
VisitedIdeal(I)<-true;
for each ideal I' in ImSucc(I) do
     HasVisited(I'\I) <- true;
     for each element E in P do
         if HasVisited(E) = True then
             MR(I'\I, E) <- MR(I'\I, E) + LinExtIdeal(I) * LinExtFilter(I')
     if I' ≠ P and not VisitedIdeal(I') then
         ComputeMutualRankDFS(I', Height + 1);
```

Złożoność

- Zliczanie rozszerzeń liniowych: O(I(P) * w(P))
- Znajdowanie kolejnych rozszerzeń liniowych: O(P * w(P))
- Zliczanie ilości rozszerzeń, dla których *x* jest na pozycji *i* : O(I(P) * |P| * w(P))
- Zliczanie ilości rozszerzeń, dla których *x* jest przed *y* : O(I(P)*|P| * w(P))

Bibliografia

- M. Peczarski „Liczenie rozszerzeń liniowych w czasie wielomianowym”
- M. Peczarski „New reults in Minimum-Comparison Sorting”
- G.Pruesse, F. Ruskey „Generating linear extensions fast”
- K. De Loof, B. De Baets, H. De Mayer „Exploiting the lattice of ideals representation of posets”
- M.Habib, R.Medina, L. Nourine, G.Steiner „Efficient algorithm on distributive lattices”
- Ch. Papadimitriou „Złożoność obliczeniowa”
- M. Wells „Elements of combinatorial Computing”

Dziękuję za uwagę!!

- Pytania?