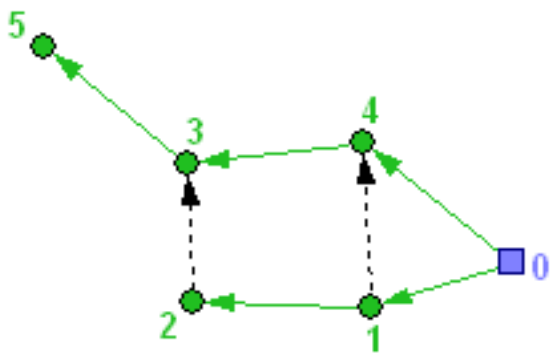
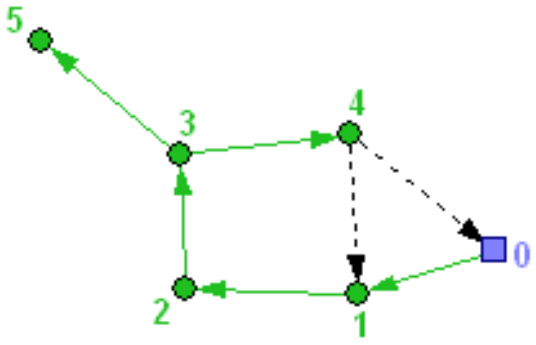




Przeszukiwania grafów

(dla kierunku informatyka to powtórka,
dla kierunku matematyka stosowana to nowe zagadnienia)

dr Anna Beata Kwiatkowska, UMK Toruń

Literatura

Thomas H. Cormen, Charles E. Leiserson, Ronald L Rivest **Wprowadzenie do algorytmów**

Witold Lipski

Kombinatoryka dla programistów

Kenneth A. Ross, Charles R.B. Wright

Matematyka dyskretna

Maciej M. Sysło, Narsingh Deo, Janusz Kowalik

Algorytmy optymalizacji dyskretnej

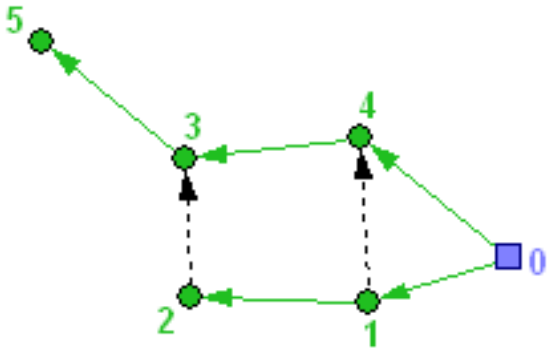
http://www.oi.edu.pl

Przeszukiwanie grafu

Przeszukiwanie grafu to systematyczne przechodzenie wzdłuż jego krawędzi w celu odwiedzenia wszystkich wierzchołków.

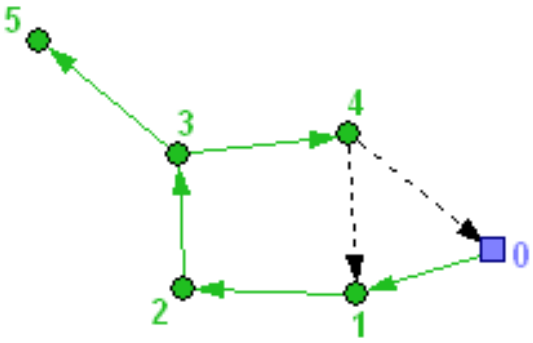
Przeszukiwanie wszerz (BFS- ang. breadth-first search)

W tym algorytmie odwiedzamy najpierw wszystkich sąsiadów danego wierzchołka, a potem wszystkich sąsiadów tych sąsiadów, etc. Aby zachować taką kolejność odwiedzeń korzystamy z struktury nazywanej kolejką.

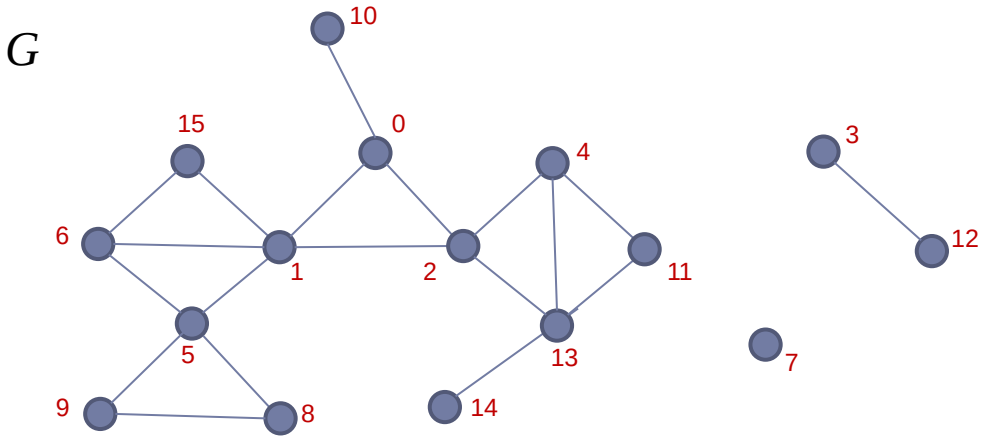


Przeszukiwanie w głąb (DFS – ang. depth-first search)

W tym algorytmie z danego wierzchołka przechodzimy coraz głębiej do następnych. Gdy już nie możemy przejść dalej, wycofujemy się i sprawdzamy inne możliwe drogi. Jest to strategia przeszukiwania z nawrotami. Kolejność powrotów wyznacza niejawnie tworzony przez rekurencję stos.



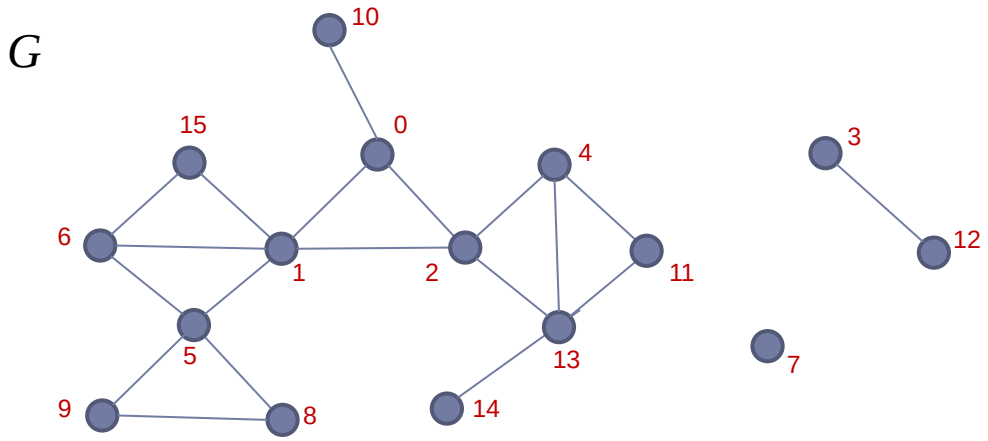
Przeszukiwanie grafu wszerz



Struktury danych:

- graf G(V,E) o n wierzchołkach i m krawędziach reprezentowany jako lista sąsiedztwa.
- L[0..n-1] – tablica/lista taka, że L[v] = true (1), gdy element v był już odwiedzony, L[v] = false (0) w przeciwnym wypadku.
- F[0..n-1] – tablica/lista poprzedników (ojców) - informacja, z którego wierzchołka przyszliśmy do aktualnie rozpatrywanego.
- Q – kolejka, do której wkładamy sąsiadów aktualnie przetwarzanego wierzchołka. Wyznacza kolejność ich przetwarzania.

Przeszukiwanie grafu wszerz – symulacja

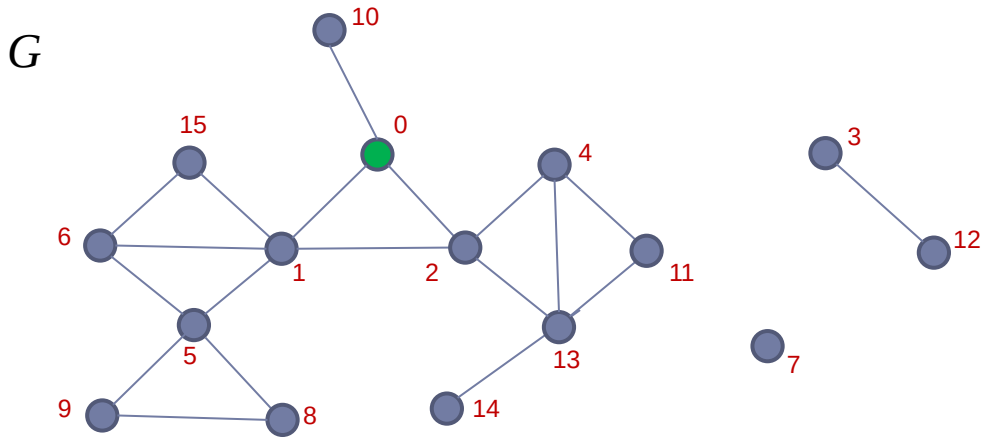


	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
L:	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

v= Q:

Przeszukiwanie grafu wszerz



L:

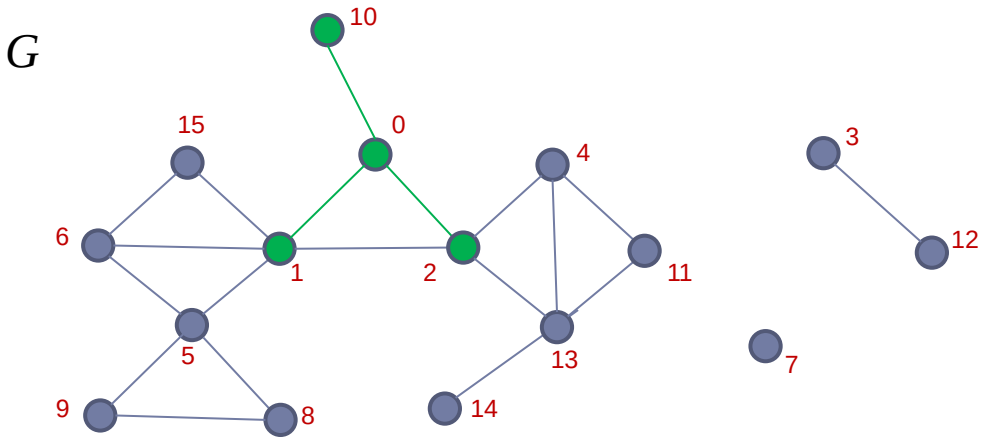
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

F:

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

v=0 Q: 0

Przeszukiwanie grafu wszerz



L:

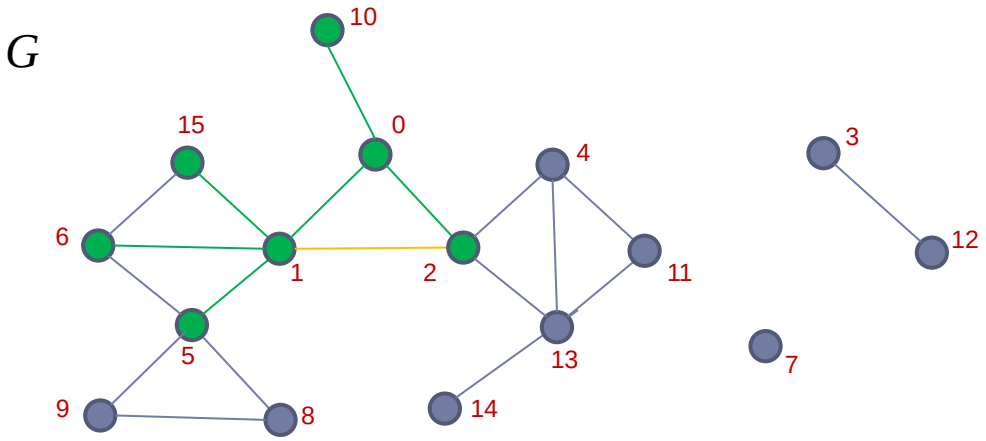
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0

F:

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

v=0 Q: ~~0,~~ 1, 2, 10, #kolejność wkładania sąsiadów do kolejki jest uzależniona #od ich kolejności w liście sąsiadów

Przeszukiwanie grafu wszerz



L:

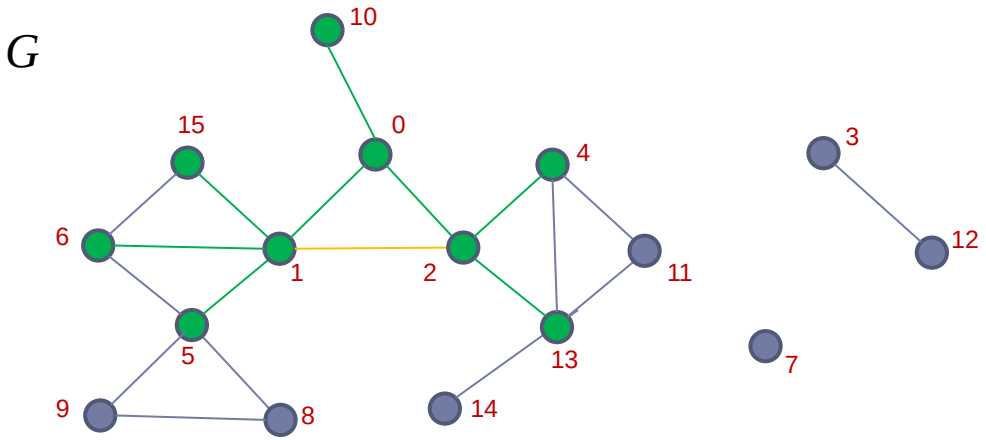
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	0	0	1	1	0	0	0	1	0	0	0	0	1

F:

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

v=1 Q: ~~0, 1,~~ 2, 10, 15, 5, 6 #krawędzie zaznaczone kolorem żółtym #zamykają cykle proste

Przeszukiwanie grafu wszerz



L:

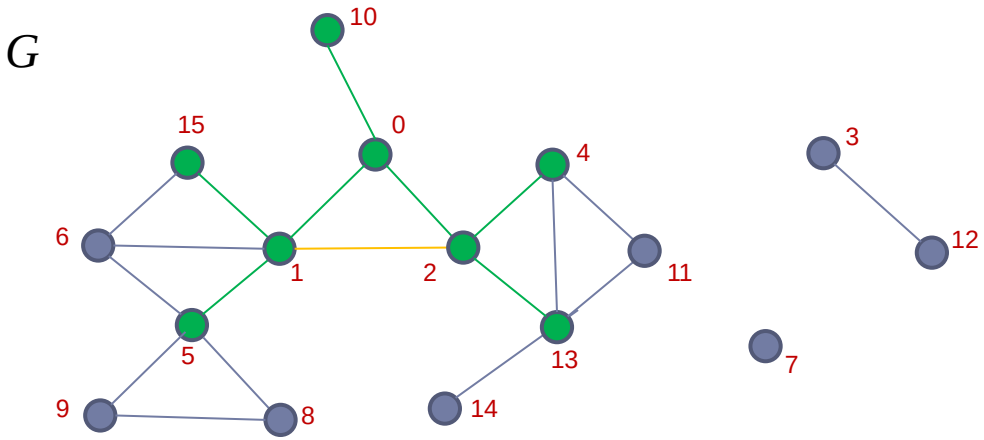
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	0	1	1	1	0	0	0	1	0	0	1	0	1

F:

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

v=2 Q: ~~0, 1, 2,~~ 10, 15, 5, 6, 4, 13

Przeszukiwanie grafu wszerz



L:

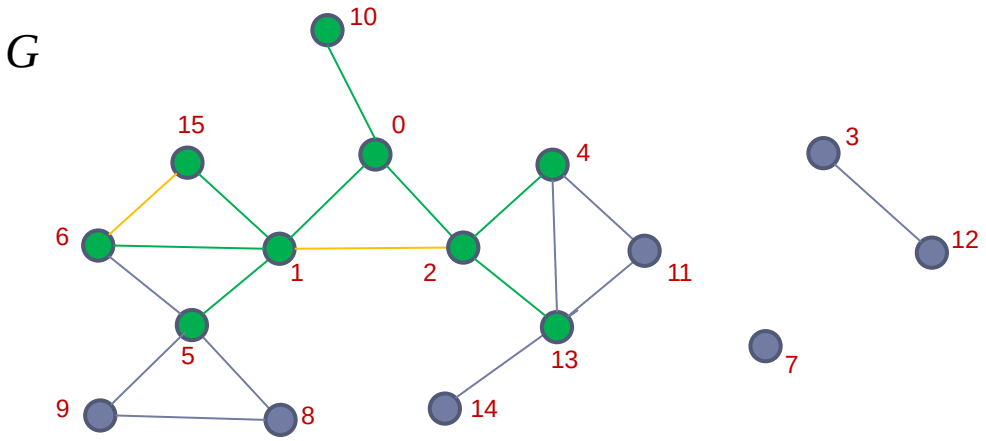
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
l	l	l	0	l	l	l	0	0	0	l	0	0	l	0	l

F:

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

v=10 Q: ~~0, 1, 2, 10,~~ 15, 5, 6, 4, 13

Przeszukiwanie grafu wszerz



L:

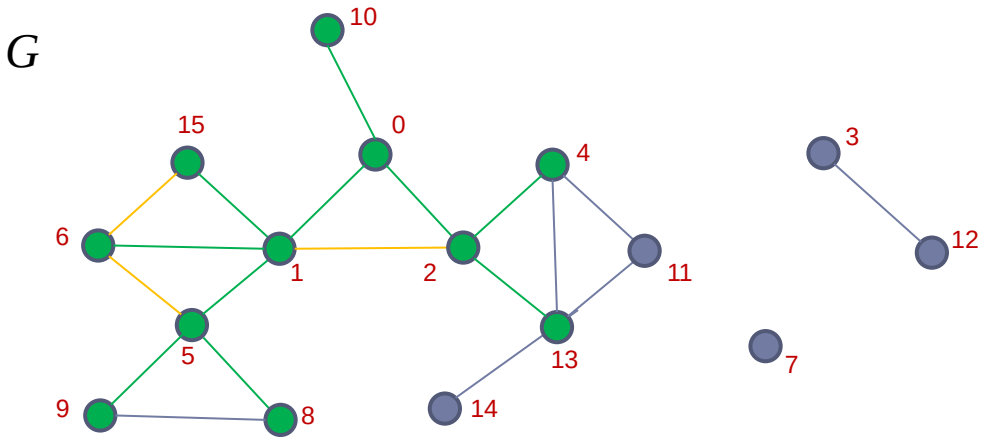
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	0	1	1	1	0	0	0	1	0	0	1	0	1

F:

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

v=15 Q: ~~0, 1, 2, 10, 15,~~ 5, 6, 4, 13

Przeszukiwanie grafu wszerz



L:

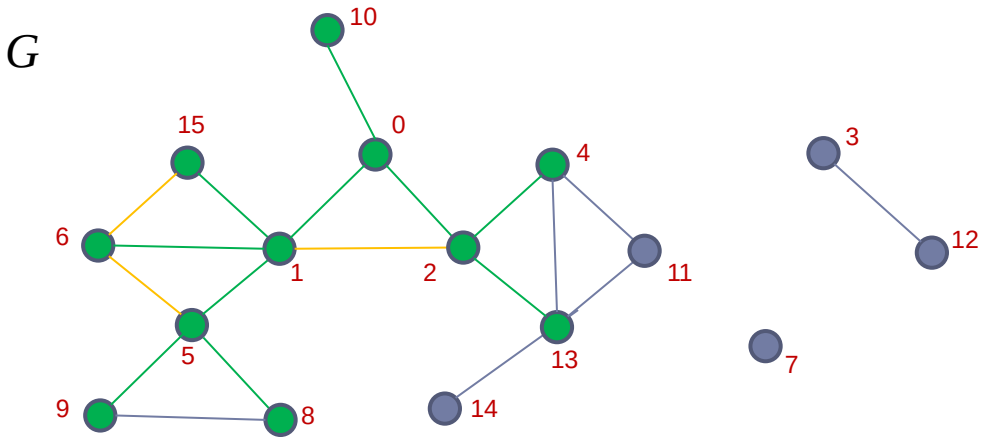
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	0	1	1	1	0	1	1	1	0	0	1	0	1

F:

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

v=5 Q: ~~0, 1, 2, 10, 15, 5~~, 6, 4, 13, 9, 8

Przeszukiwanie grafu wszerz



L:

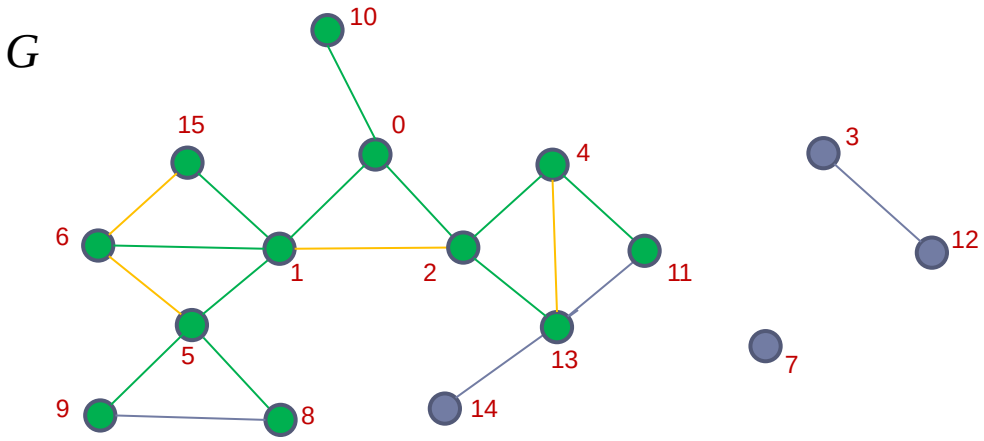
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	0	1	1	1	0	1	1	1	0	0	1	0	1

F:

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

v=6 Q: ~~0, 1, 2, 10, 15, 5, 6,~~ 4, 13, 9, 8

Przeszukiwanie grafu wszerz



L:

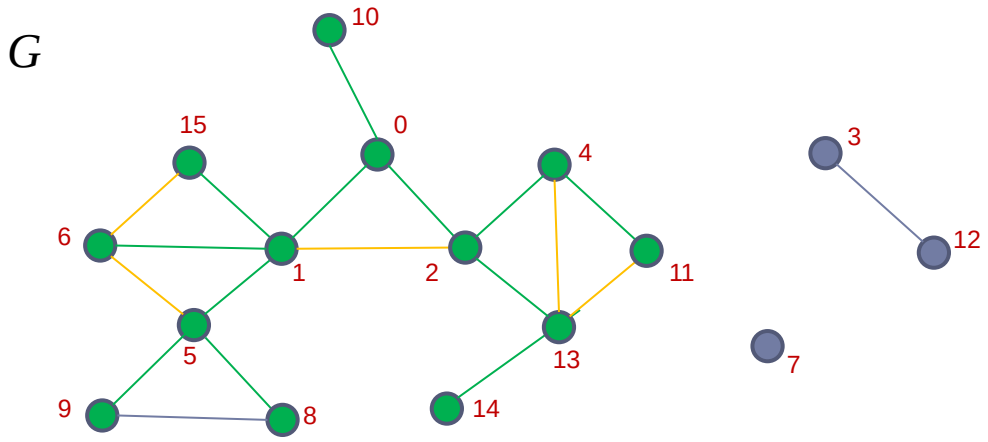
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
∞	∞	∞	0	∞	∞	∞	0	∞	∞	∞	∞	0	∞	0	∞

F:

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

v=4 Q: ~~0, 1, 2, 10, 15, 5, 6, 4,~~ 13, 9, 8, 11

Przeszukiwanie grafu wszerz



L:

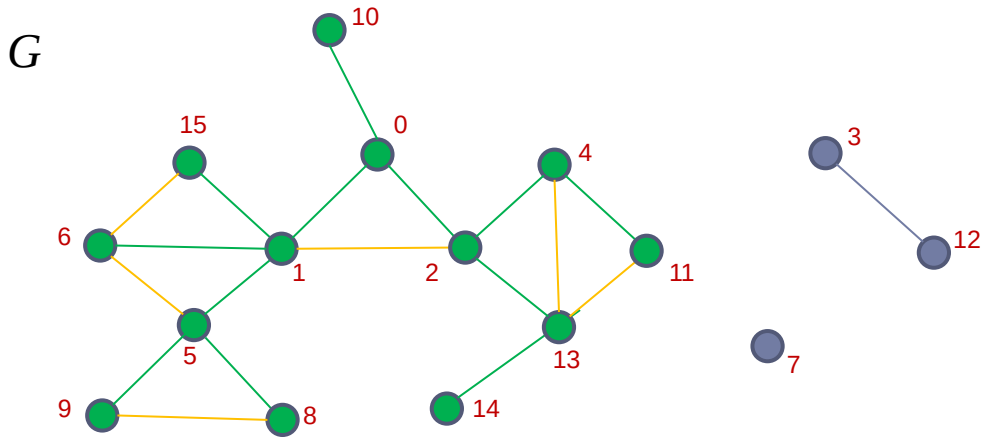
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	0	1	1	1	0	1	1	1	1	0	1	1	1

F:

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

v=13 Q: ~~0, 1, 2, 10, 15, 5, 6, 4, 13,~~ 9, 8, 11, 14

Przeszukiwanie grafu wszerz



L:

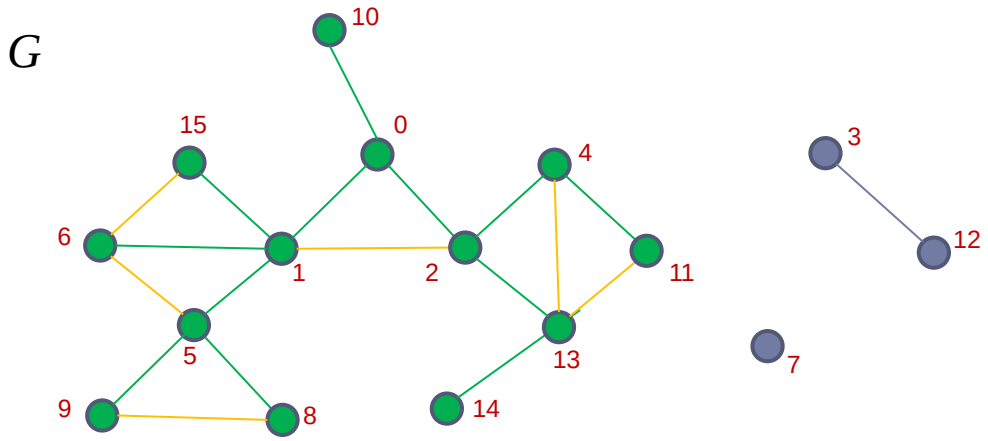
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	0	1	1	1	0	1	1	1	1	0	1	1	1

F:

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

v=9 Q: ~~0, 1, 2, 10, 15, 5, 6, 4, 13, 9,~~ 8, 11, 14

Przeszukiwanie grafu wszerz



L:

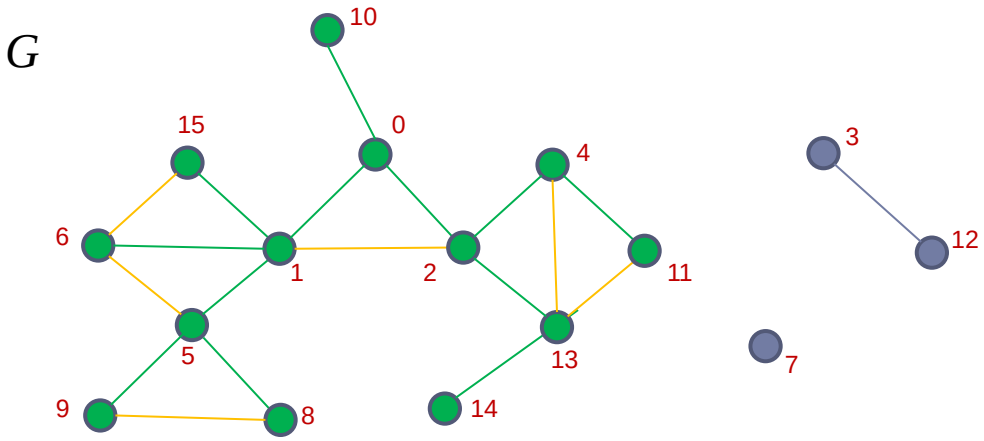
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	0	1	1	1	0	1	1	1	1	0	1	1	1

F:

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

v=8 Q: ~~0, 1, 2, 10, 15, 5, 6, 4, 13, 9, 8,~~ 11, 14

Przeszukiwanie grafu wszerz



L:

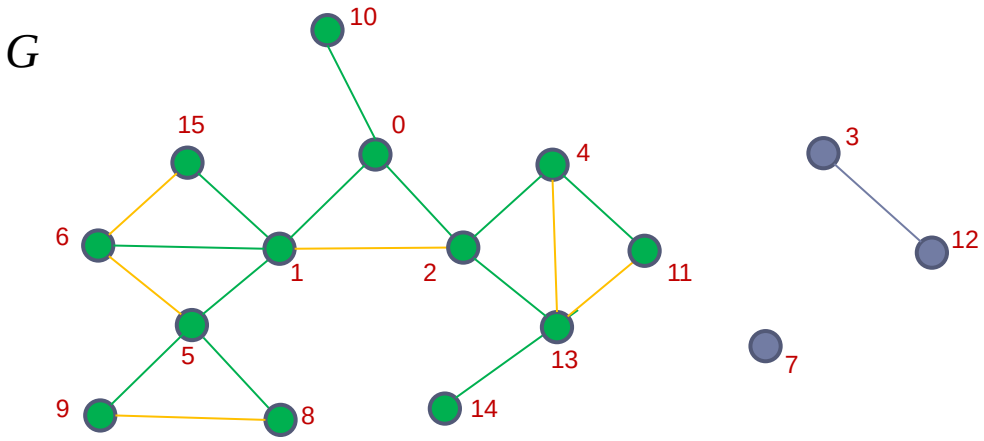
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	0	1	1	1	0	1	1	1	1	0	1	1	1

F:

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

v=11 Q: ~~0, 1, 2, 10, 15, 5, 6, 4, 13, 9, 8, 11,~~ 14

Przeszukiwanie grafu wszerz



L:

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

F:

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

v=14 Q: ~~0, 1, 2, 10, 15, 5, 6, 4, 13, 9, 8, 11, 14~~

Kończymy przeszukiwanie, gdy kolejka jest pusta. W ten sposób przeszukujemy jedną składową grafu, w której był wierzchołek startowy.

Przeszukiwanie grafu wszerz - pseudokod

```
bfs(v)
begin
        Q := Ø; Q ← v; L[v] ← true;
        while Q ≠ Ø do
        begin
                p← head[Q];        #z kolejki
                for u ∈N(p) do   if (not L[u]) then      #dla wszystkich sąsiadów u wierzchołka p
                  begin
                    Enqueue(Q,u) ;          #sąsiad u do kolejki
                    L[u] ← true; F[u] ← p;
                end;
end;
```

Wywołanie **bfs(v)** powoduje odwiedzenie wszystkich wierzchołków składowej spójności grafu zawierającej wierzchołek v, każdy wierzchołek jest odwiedzony dokładnie raz. Każdy wierzchołek jest umieszczany w kolejce i usuwany dokładnie raz, a liczba wszystkich iteracji pętli for jest rzędu liczby krawędzi grafu.

Całkowita złożoność tego algorytmu wynosi zatem **O(n+m)**.

Przeszukiwanie grafu wszerz dla wielu składowych

```
bfs(v)
begin
        Q := Ø; Q ← v; L[v] ← true;
        while Q ≠ Ø do
        begin
                p← head[Q];
                for u ∈N(p) do   if (not L[u]) then
                  begin
                    Enqueue(Q,u) ;
                    L[u] ← true; F[u] ← p;
                end;
end;
```

Przeszukiwanie wszerz dowolnego grafu, niekoniecznie spójnego

```
begin
        for v ∈ V do
        begin
                L[v] ← false;
                F[v] ← 0;
        end;
        for v ∈V do if not (L[v]) then bfs(u);
end.
```

Wywołujemy bfs dla każdej składowej. Pierwsze wywołanie wykonujemy dla wierzchołka o numerze 0. Po jego zakończeniu sprawdzamy kolejne wartości w tablicy L. Jeśli natrafimy na nieodwiedzony wierzchołek (wartość false), to znów startujemy przeszukiwanie od tego wierzchołka.

Złożoność algorytmu również wynosi **O(n+m)**.

Przeszukiwanie wszerz - składowe spójności

Przeszukując graf możemy pozyskać więcej informacji o składowych spójności, np.:

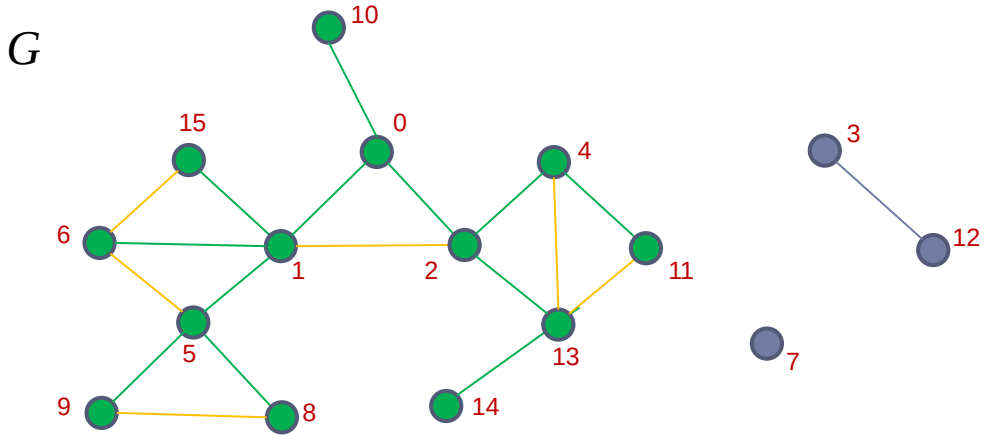
- ile jest składowych spójności
- które wierzchołki są w poszczególnych składowych

Przy kolejnych wywołaniach przeszukiwania wypełniamy L innymi wartościami (np. kolejnymi wartościami zmiennej s):

```
for v:=0 to n-1 do L[v]:=0; #jest teraz tablicą składowych:
s:=0;
for v:= 0 to n-1 do if L[v] = 0 then
begin
        s=s+1;
        bfs(v);   #wszystkim wierzchołkom osiągniętym z wierzchołka v
                  # ustalamy w L wartość równą s
  end;
```

Przeszukiwanie wszerz – najkrótsze drogi

1. Po wykonaniu przeszukiwania tablica ojców F zawiera drzewo najkrótszych dróg z wierzchołka startowego (na rysunku kolor zielony) do wszystkich pozostałych wierzchołków w grafie. Drogę tę reprezentujemy ścieżką odczytaną z F poczynając od wierzchołka docelowego.



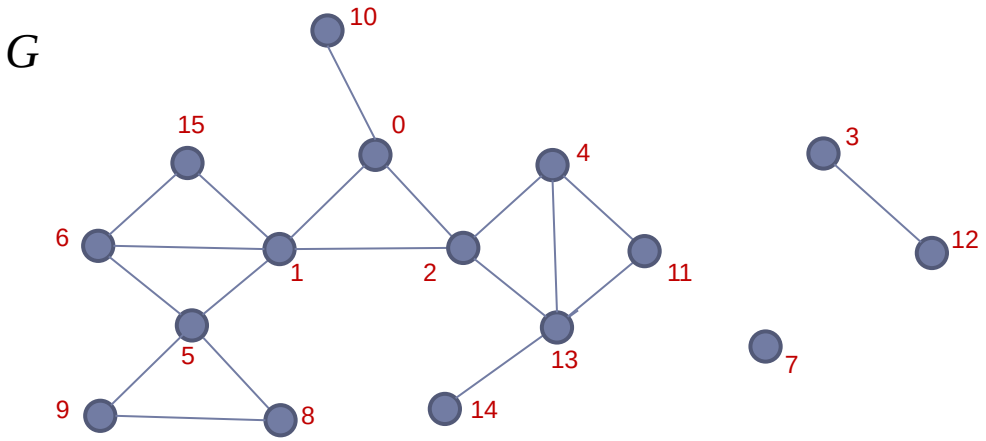
F:

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

Na przykład: aby znaleźć najkrótszą drogę z wierzchołka 0 do 8, zaczynamy w F od miejsca 8, jest tam informacja, że przyszliśmy do 8 z 5, w 5 jest informacja, że przyszliśmy do niej z 1, w 2 jest informacja, że przyszliśmy do niej z 0 (wierzchołek początkowy).

Zatem najkrótsza droga jest wyznaczona ścieżką: 0, 1, 5, 8.

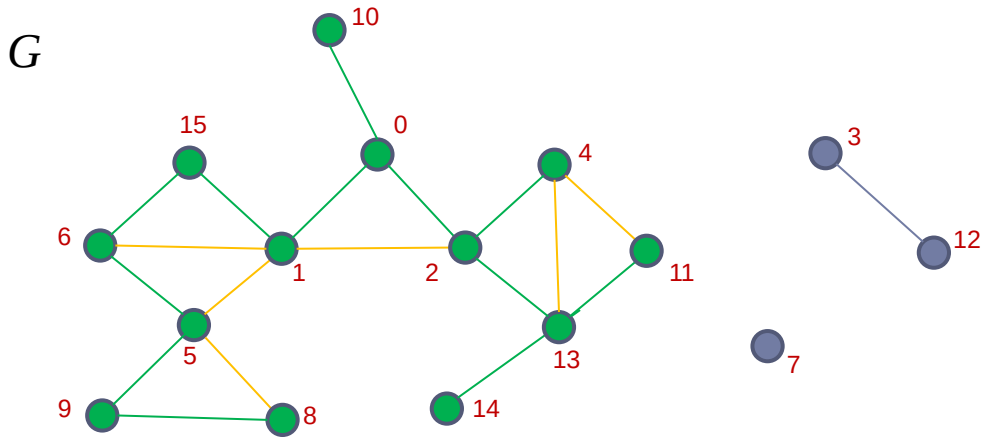
Przeszukiwanie grafu w głąb



Struktury danych:

- graf G(V,E) o n wierzchołkach i m krawędziach reprezentowany jako lista sąsiedztwa.
- L[0..n-1] – tablica/lista taka, że L[v] = true (1), gdy element v był już odwiedzony, L[v] = false (0) w przeciwnym wypadku.
- F[0..n-1] – tablica/lista poprzedników (ojców) - informacja, z którego wierzchołka przyszliśmy do aktualnie rozpatrywanego.
- Przeszukiwanie dfs bazuje na niejawnym stosie rekurencyjnym.

Przykładowe drzewo przeszukiwania w głąb



- Przykładowe drzewo poszukiwań algorytmu dfs dla jednej składowej grafu G zaznaczone jest kolorem zielonym.
- Krawędzie żółte, to krawędzie powracające – te z których wycofuje się rekurencja.

- Algorytm wyznacza między dwoma wierzchołkami dowolna drogę. W tym drzewie mamy drogę z wierzchołka 0 do 8 długości 6 (liczba krawędzi) wyznaczona wierzchołkami:

  0, 1, 15, 6, 5, 9, 8.

Przeszukiwanie grafu w głąb

```
dfs(v)
begin
    L[v] ← true;
    for u ∈N(v) do
    if not (L(u)) then
    begin
        F (u) ← v;
        dfs(u);
    end;
end;
```

```
Przeszukiwanie w głąb dowolnego grafu,
    niekoniecznie spójnego
begin
    for v ∈ V do
    begin
        L[v] ← false;
        F[v] ← 0;
    end;
    for v ∈V do if not (L[v]) then dfs(u);
end.
```

W programie głównym liczba kroków w pętlach wynosi n, nie licząc kroków wykonywanych przez procedurę dfs. Procedura ta jest wywoływana co najwyżej n razy w drugiej pętli.

Całkowita liczba kroków pętli for w dfs dla wszystkich wywołań tej jest rzędu m. Całkowita złożoność tego algorytmu wynosi **O(n+m)**.

dfs a bfs

1. W algorytmie bfs kolejka jest jawną strukturą, założoną przez programistę, natomiast stos w dfs jest ukryty, gdyż każda procedura rekurencyjna jest związana z założeniem stosu przez kompilator.
2. Obydwa typy przeszukiwania mogą być użyte do badania składowych spójności.
3. Oba typy przeszukiwania mogą być użyte do znajdowania drogi między ustalonymi wierzchołkami v i u. Wystarczy przeszukać graf z wierzchołka v, aż do momentu odwiedzenia wierzchołka u. Przeszukiwanie bfs znajduje drogę najkrótszą, dfs znajduje dowolną drogę.
4. Przeszukiwanie wszerz poprowadzone aż kolejka będzie pusta, daje w wyniku drzewo najkrótszych dróg.
5. Zaletą przeszukiwania w głąb jest to, że w chwili odwiedzenia wierzchołka u stos zawiera ciąg wierzchołków określający drogę z v do u.
6. Wprowadzenie tablicy F daje możliwość łatwego odtworzenia drogi z v do u w obydwu przeszukiwaniach.
7. Obydwa przeszukiwania można wykonywać na grafach jak i digrafach.
8. Wiele algorytmów grafowych rozpoczyna się od przeszukiwania grafu. Rodzaj przeszukiwania dobieramy w zależności od tego, jaki własności grafu badamy.

Zliczanie składowych – różne algorytmy

Przeszukiwaniu grafu (w głąb lub wszerz) - wierzchołki, do których można dotrzeć z jednego wierzchołka, poruszając się po krawędziach w dowolnym kierunku (także „pod prąd"), tworzą jedną składową

Algorytm find - union, polegający na sukcesywnym łączeniu
w coraz większe zbiory wierzchołków połączonych krawędziami i należących do jednej składowej.

Algorytm polegający na usuwaniu z digrafu kolejno wierzchołków, do których nie wchodzi żadna krawędź; w ten sposób każda składowa digrafu zostaje zredukowana do cyklu, a cykle możemy już łatwo zliczyć.

Zadanie Skarbonki

XII OI, Etap I

Skarbonki

Smok Bajtazar ma n skarbonek. Każda skarbonkę można otworzyć jej kluczem lub rozbić młotkiem. Bajtazar powrzucał klucze do pewnych skarbonek, pamięta przy tym który do której.
Bajtazar zamierza kupić samochód i musi dostać się do wszystkich skarbonek. Chce jednak zniszczyć jak najmniej z nich.
Pomóż Bajtazarowi ustalić, ile skarbonek musi rozbić.

Zadanie Skarbonki - rozwiązanie

Utwórzmy graf skierowany G mający n wierzchołków ponumerowanych liczbami od 1 do n. Wierzchołki o numerach i i j są połączone krawędzią (biegnącą od i do j), jeśli klucz do skarbonki i znajduje się w skarbonce j.

Przykład Popatrzmy na przykład takiego grafu. Przypuśćmy, że $n = 30$ oraz w pliku wejściowym (po liczbie 30) znajdują się następujące liczby:

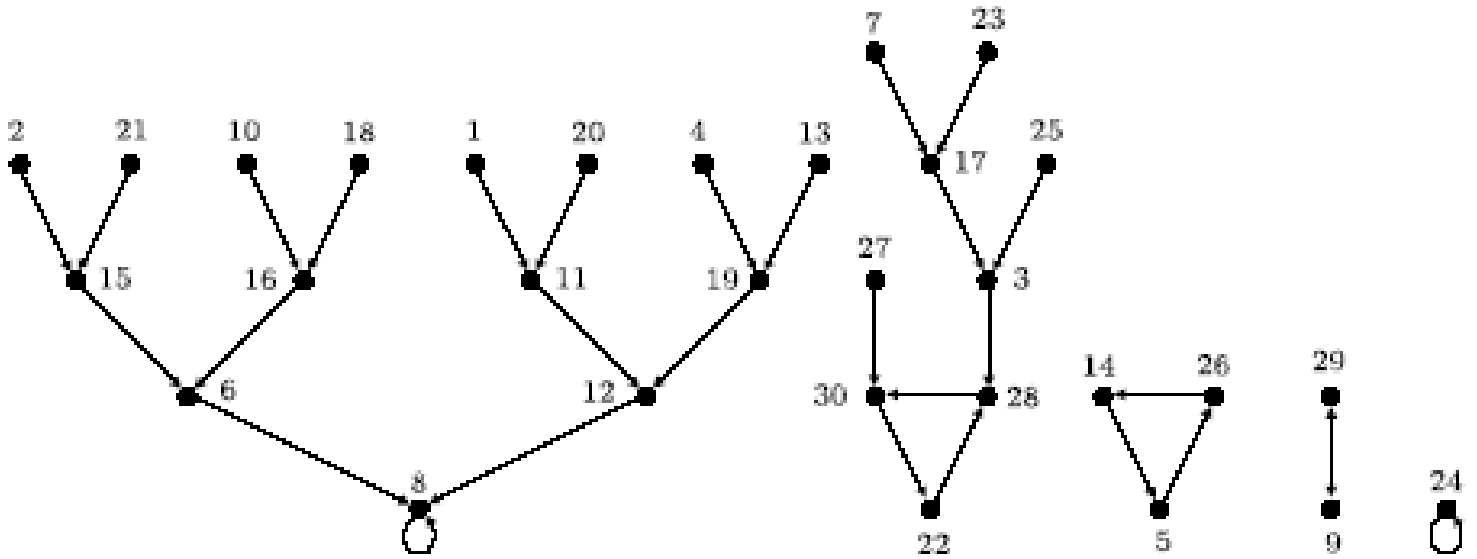
11, 15, 28, 19, 26, 8, 17, 8, 29, 16, 12, 8, 19, 5, 6, 6, 3, 16, 12, 11, 15, 28, 17, 24, 3, 14, 30, 30, 9, 22.

Inaczej mówiąc, w grafie G znajdują się następujące krawędzie:

1	→	11	2	→	15	3	→	28	4	→	19	5	→	26
6	→	8	7	→	17	8	→	8	9	→	29	10	→	16
11	→	12	12	→	8	13	→	19	14	→	5	15	→	6
16	→	6	17	→	3	18	→	16	19	→	12	20	→	11
21	→	15	22	→	28	23	→	17	24	→	24	25	→	3
26	→	14	27	→	30	28	→	30	29	→	9	30	→	22

Zadanie Skarbonki - rozwiązanie

A oto rysunek tego grafu:



Zauważmy, że przedstawiony graf ma 5 składowych (czyli, mówiąc językiem potocznym, składa się z pięciu oddzielnych kawałków) i każda z tych składowych ma postać cyklu (być może długości 1), do którego dochodzą skierowane drzewa (czyli grafy bez cykli).

Oczywiście rozbicie którejkolwiek skarbonki odpowiadającej wierzchołkowi w cyklu pozwala otworzyć wszystkie skarbonki odpowiadające wierzchołkom składowej, w której ten cykl się znajduje. Oznacza to, że zadanie sprowadza się do policzenia składowych grafu.

Zadanie Prostokąty

V OI, Etap III

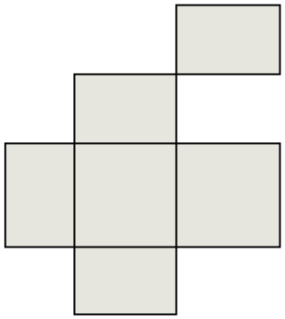
Prostokąty

Na płaszczyźnie narysowano n prostokątów, których boki są równoległe do osi współrzędnych i wierzchołki mają obie współrzędne całkowite.
Przyjmujemy że:
• każdy prostokąt jest blokiem,
• jeśli dwa różne bloki maja wspólny odcinek to tworzą one nowy blok, w przeciwnym przypadku mówimy, że są rozłączne.

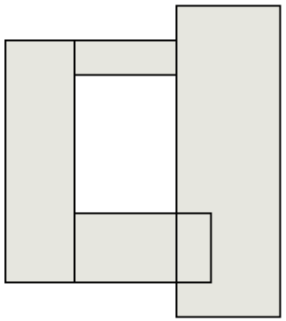
Znajdź liczbę bloków na płaszczyźnie.

Zadanie Prostokąty - przykłady

2 bloki



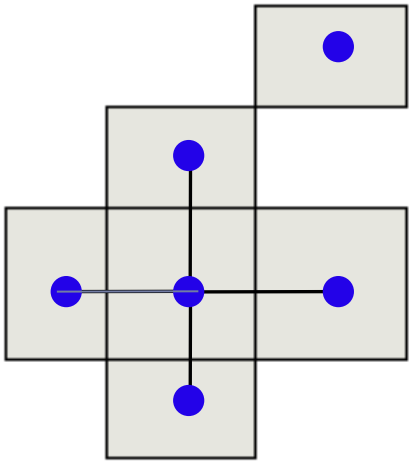
1 blok



Zadanie Prostokąty - rozwiązanie

Rozpatrujemy graf: wierzchołki to podane prostokąty, dwa prostokąty są połączone krawędzią, jeśli maja wspólny odcinek (tzn. gdy tworzą blok).

Dwa prostokąty wchodzą w skład tego samego bloku, jeśli w tak skonstruowanym grafie istnieje między nimi ścieżka. Zatem szukana liczba bloków jest równa liczbie składowych skonstruowanego przez nas grafu.



Dziękuję z uwagę

dr Anna Beata Kwiatkowska

aba@mat.umk.pl

tel. 602 184 813