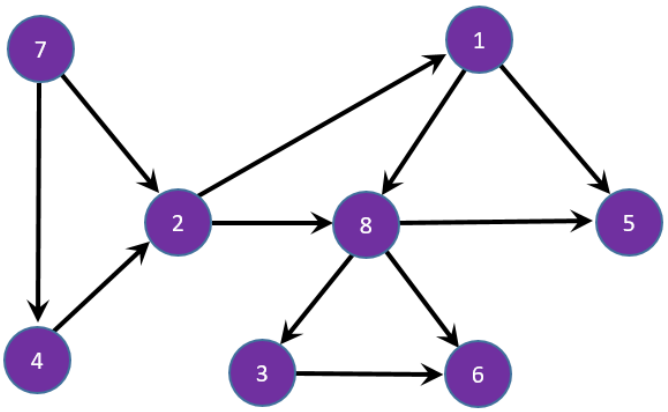
# Sortowanie topologiczne

#graf_acykliczny #dag
#krawędź powrotna
#topologia

W algorytmice ważną rolę odgrywają skierowanie grafy acykliczne (ang. **Directed Acyclic Graphs**, wymawiaj: *dajrekted esajklik graθs*, w drugim słowie akcentując *a*). Co znaczy acykliczny, chyba już rozumiemy, prawda? Z angielskiej nazwy utworzono powszechnie używany skrót **DAG** (*dag*).

Weźmy sobie taki przykładowy *dag*:




Dane wejściowe opisujące ten graf są następujące (pamiętamy, aby do ich wczytania użyć funkcji `read_directed_graph()`):

```
8 11
2 8
3 6
8 5
7 2
1 5
4 2
1 8
8 3
2 1
8 6
7 4
```

Funkcja `print_graph()` wypisze nam listy sąsiedztwa:

```
1: 5 8
2: 8 1
```

```
3: 6
4: 2
5:
6:
7: 2 4
8: 5 3 6
```

Przygotujemy teraz do naszych celów specjalną wersję funkcji realizujących algorytm DFS. Zacznijmy od `DFS_topological()`:

```
void DFS_topological(vector<vector<int>> &G, int V,
         vector<bool> &visited, vector<int> &parent,
         int &t, vector<int> &start, vector<int> &finish, stack<int> &S)
{
  for(int v = 1; v <= V; v++)
    if(!visited[v])
      DFS_visit_topological(G, V, v, visited, parent, t, start, finish, S);
}
```

Ni z gruchy, ni z pietruchy pojawił nam się tutaj jakiś stos ($S$) – prosimy o uzbrojenie się w cierpliwość, wszystko za niedługo się wyjaśni.*

Teraz funkcja `DFS_visit_topological()`:

```
void DFS_visit_topological(vector<vector<int>> &G, int V, int s,
         vector<bool> &visited, vector<int> &parent,
         int &t, vector<int> &start, vector<int> &finish, stack<int> &S)
{
  visited[s] = true;
  start[s] = ++t;
  for(int x : G[s])
    if(!visited[x])
    {
      parent[x] = s;
      DFS_visit_topological(G, V, x, visited, parent, t, start, finish, S);
    }
  finish[s] = ++t;
  . . .
}
```

Miejsce w kodzie oznaczone trzema kropkami jest szczególne – tutaj celebrowane jest zakończenie przetwarzania wierzchołka $s$. Ten fragment kodu (zaraz coś tam konkretnego wstawimy) jest wykonywany dla wszystkich wierzchołków grafu w kolejności takiej, jak z nimi kończymy. Gdybyśmy wstawili tam wypisywanie:

```
cout << s << endl;
```

*Nie wszystkie parametry oryginalnych funkcji będą nam potrzebne, ale nie pomijamy ich, aby nie tworzyć niepotrzebnego zamieszania.

wtedy na ekranie pojawiłaby się lista wszystkich wierzchołków w kolejności ich przetworzenia. Nam jednak zależy na otrzymaniu (i przechowaniu) tej listy w odwrotnej kolejności – stąd użycie stosu $S$. Jednym słowem, zamiast trzech kropek wpiszemy wstawianie $s$ na stos $S$:

```
S.push(s);
```

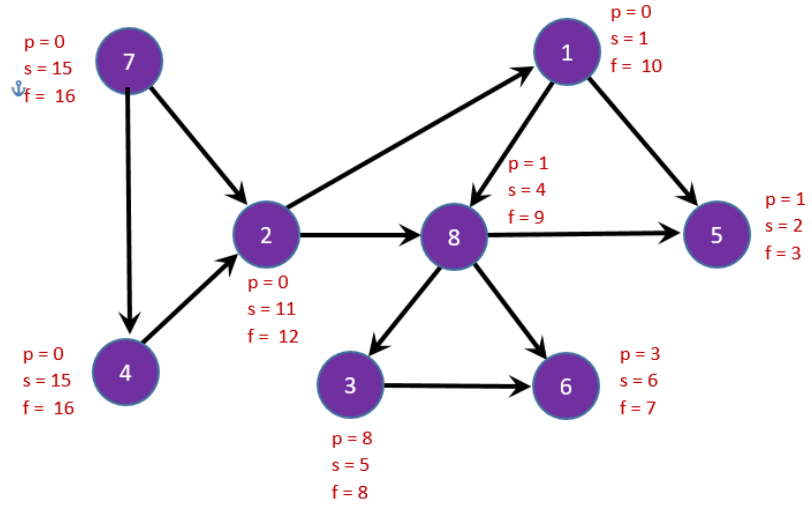
Zobaczmy, jaki będzie skutek wywołania poniższej funkcji `main()` i wpisania danych naszego grafu (na końcu daliśmy wypisanie zawartości stosu $S$):

```
int main()
{
  int V, E;
  vector<vector<int>> G;
  read_directed_graph(G, V, E);
  print_graph();
  vector<bool> visited(V + 1);
  vector<int> parent(V + 1), start(V + 1), finish(V + 1);
  stack<int> S;
  int t = 0;
  DFS_topological(G, V, visited, parent, t, start, finish, S);
  print_DFS(G, V, parent, start, finish);
  while(!S.empty())
  {
    cout << S.top() << " ";
    S.pop();
  }
  cout << endl;
}
```

Funkcja `print_DFS()` wypisze nam informacje o rezultatach przeszukiwania grafu:

```
1: p=0 s=1 f=10
2: p=0 s=11 f=12
3: p=8 s=5 f=8
4: p=0 s=13 f=14
5: p=1 s=2 f=3
6: p=3 s=6 f=7
7: p=0 s=15 f=16
8: p=1 s=4 f=9
```

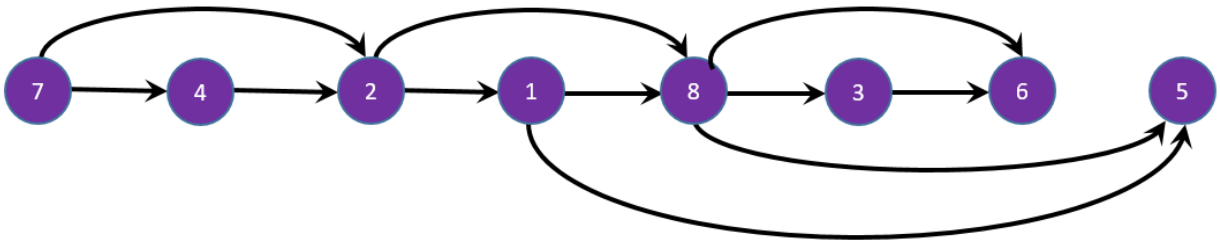
co przekłada się na rozpiskę na grafie:




Na ekranie na samym końcu wydruku pojawi się zawartość stosu (lista wierzcołków posortowana malejąco według czasów $finish$):

```
7 4 2 1 8 3 6 5
```

Ułóżmy te wierzchołki wzdłuż jednej linii według powyższej kolejności i dorysujmy krawędzie (wszystkie) naszego grafu:




Hmmm... Co ja pacze? Ja pacze, że wszystkie strzałki pokazują w prawą stronę! Można powiedzieć, że graf został na swój sposób „uczesany". Doprowadzenie grafu do takiego stanu nosi nazwę *sortowania topologicznego* (ang. **topological sort**, wymawiaj: *topolodżikal sort* z akcentem na trzecią sylabę). Dzięki tej operacji graf staje się strukturą w zasadzie liniową, z określonym porządkiem wierzchołków.

W fundamentalnym podręczniku algorytmiki Cormena i spółki zamieszczono przykład sortowania topologicznego grafu, którego wierzchołkami są elementy ubrania, a zwroty krawędzi określają, co przed czym należy ubierać. Pozwolimy sobie na małą przeróbkę, przypisując wierzchołkom naszego grafu części staropolskiej garderoby:[†]

1. giezło,
2. hajdawery,
3. onuce,
4. ineksprymable,

[†]W tamtych czasach nie było Supermana, który – jak wiadomo – ubiera femurały na hajdawery.

5. pas słucki,
6. baczmagi,
7. femurały.
8. żupan.

Teraz kolejność ubierania (co przed czym, według krawędzi w naszym grafie):

- hajdawery przed żupanem,
- onuce przed baczmagami,
- żupan przed pasem słuckim,
- femurały przed hajdawerami,
- giezło przed pasem słuckim,
- inekspymable przed hajdawerami,
- giezło przed żupanem,
- żupan przed onucami,
- hajdawery przed giezłem,
- żupan przed baczmagami,
- femurały przed inekspymablami.

W wyniku sortowania topologicznego uzyskujemy taką oto kolejność ubierania się:

**femurały, inekspymable, hajdawery, giezło, żupan, onuce, baczmagi, pas słucki.**

Czy zawsze sortowanie topologiczne grafu skierowanego jest możliwe? Oczywiście, że nie – graf musi być acykliczny, czyli musi być *dagiem*. Jeśli w grafie występuje cykl, wtedy pojawia się w nim przynajmniej jedna *krawędź powrotna*, czyli wskazująca w przeciwnym kierunku do generalnego kierunku „uczesania" i nie da się jej nijak obrócić we właściwym kierunku.

Sortowanie topologiczne nie musi dawać jednoznacznej kolejności wierzchołków, na przykład w naszym grafie na ostatnim rysunku można zamienić wierzchołki 6 oraz 5 i graf nadal pozostanie prawidłowo posortowany. Niech Czytelnik zastanowi się, kiedy sortowanie topologiczne daje absolutnie jednoznaczny wynik.