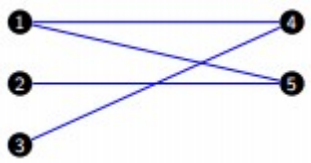
* Đồ thị có hướng - directed graph:
- cạnh khuyên: loop edge
- hai cạnh //: multile edges

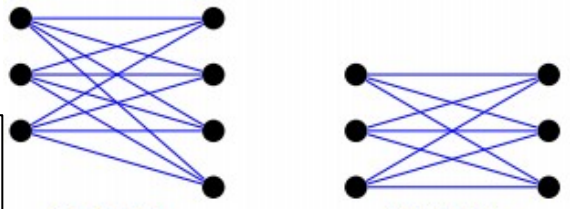
* Đồ thị vô hướng - undirected graph:
- đồ thị nền – underlying undirected graph
- đồ thị hỗn hợp - mixed graph

* Quan hệ trên đồ thị
- đồ thị con – subgraph
- đồ thị bộ phận - spanning subgraph
- đồ thị sinh - induced subgraph
- đồ thị đẳng cấu – isomorphic
- đồ thị bù nhau - complement

* Đồ thị đơn - simple graph (đồ thị ko có cạnh khuyên và cạnh //, số cạnh tối đa là n*(n - 1)/2)
- đa đồ thị - multigraph (có thể có cạnh khuyên và cạnh //)
- đồ thị rỗng - null graph (có tập cạnh rỗng)
- đồ thị đều - regular grapg (tất cả các đỉnh có cùng số bậc, gọi k là bậc các đỉnh thì đồ thị dc gọi là k-đều, với số cạnh là n*k/2 vs n là số đỉnh)
- đồ thị đầy đủ - complete graph (giữa 2 đỉnh bất kì luôn có cạnh nối, đồ thị đầy đủ có n đỉnh dc kí hiêu là Kn, số canh là n*(n – 1)/2,)
- đồ thị phân đôi
- bipartite graph



- đồ thị phân đôi đủ



- đồ thị vòng - circular graph
- đồ thị sao - star graph
- đồ thị bánh xe - wheel graph
- đồ thị n lập phương - n-cube graph

* Bậc của đỉnh - degree
- bậc cực đại - maximum degree
- bậc cực tiểu - minimum degree
- nửa bậc ngoài – out-degree
- nửa bậc trong – in-degree
- đỉnh cô lập - isolated vertex (bậc bằng 0)
- đỉnh treo – pendant vertex (bậc bằng 1)
- cạnh treo - pendant edge (cạnh kề vs đỉnh treo)

- ma trận kề - adjacency matrix
- đồ thị thưa - sparse graph
- đồ thị dày - dense graph
- chu trình - mạch – cycle
- dây chuyền - path

Algorithm 3 Tìm đường đi P từ đỉnh v_s đến v_e

```
1: function DFS_FIND_PATH(v_s, v_e)
2:     Duyệt v_s
3:     if v_s = v_e then
4:         return true
5:     for mỗi đỉnh v kề với đỉnh v_s do
6:         if v chưa được duyệt then
7:             pre[v] ← v_s
8:             if DFS_FIND_PATH(v, v_e) then
9:                 return true
10:    return false
```

Algorithm 2 Tìm kiếm theo chiều rộng

```
1: procedure BFS(v)
2:     queue ← v
3:     while queue ≠ ∅ do
4:         x ← queue
5:         Duyệt đỉnh x
6:         for mỗi đỉnh u kề với đỉnh x do
7:             if đỉnh u chưa duyệt và không có trong queue then
8:                 queue ← u
```

* Cây khung - spanning tree

* Đồ thị liên thông - connected graph - luôn tồn tại đường đi giữa mọi cặp đỉnh.
- đỉnh cắt - cut vertex, cầu – bridge
- bậc liên thông đỉnh - vertex connectivity
- bậc liên thông cạnh - edge connectivity
- thành phần liên thông - connected component
- đồ thị con liên thông tối đại - maximal connected subgraph
- đồ thị liên thông yếu - weakly connected graph
- đồ thị liên thông mạnh - strongly connected graph
- đồ thị có trọng số - weighted graph

* DFS tìm cây khung
Khởi tạo cây khung T vs đỉnh v bất kì
def dfs(v):
  for u kề v:
    if u not in T:
      thêm đỉnh u và cạnh (v,u) vào T
      dfs(u)

Kruskal

Thêm vào V_T	Thêm vào E_T
$\{c,d\}$	(c,d)
$\{d,g\}$	(d,g)
$\{a,d\}$	(a,d)
$\{b,d\}$	(b,d)
$\{e,g\}$	(e,g)
$\{f,g\}$	(f,g)

Prim

Thêm vào V_T	Thêm vào E_T
a	
b	(a,b)
d	(a,d)
g	(a,g)
c	(b,c)
e	(c,e)
f	(e,f)

* Prim
thêm đỉnh bất kì vào pqueue
while queue != []:
  v = queue.pop()
  đánh dấu thăm v
  for u kề v:
    if dist[v] > weight[u, v]
      và chưa thăm u:
        cập nhật dist[v] =
            weight[u, v]
        thêm u vào pqueue

* Đồ thị phẳng - planar graph
- Biểu diễn phẳng – planar representation

Algorithm 5 Thuật toán xây dựng cây DFS

```
1: Khởi tạo cây T với đỉnh v bất kỳ
2: DFS_TREE(u)
3: procedure DFS_TREE(v)
4:     for mỗi đỉnh u kề với v do
5:         Thêm đỉnh u và cạnh (v, u) vào cây T
6:         DFS_TREE(u)
```

Algorithm 6 Thuật toán sắp xếp thứ tự

```
1: L ← ∅
2: S ← Các đỉnh không có cạnh vào
3: while S ≠ ∅ do
4:     lấy một đỉnh v từ S
5:     thêm đỉnh v vào L
6:     for đỉnh u kề với v do loại bỏ cạnh (u, v) khỏi đồ thị
7:         if u không có cạnh vào then
8:             thêm u vào S
9: if đồ thị vẫn còn cạnh then
10:    xuất thông báo không thể sắp xếp thứ tự
11: else
12:    xuất danh sách L chứa các đỉnh theo thứ tự
```

Algorithm 2 Tìm tất cả đường đi từ đỉnh v_s đến v_e

```
1: procedure DFS_FIND_ALL_PATHS(v_s, v_e)
2:     Duyệt v_s
3:     if v_s == v_e then
4:         In ra đường đi
5:     for mỗi đỉnh v kề với đỉnh v_s do
6:         if v chưa được duyệt then
7:             PUSH(pre[v])
8:             pre[v] = v_s
9:             DFS_FIND_ALL_PATHS(v, v_e)
10:            POP(pre[v])
11:    v_s trở lại trạng thái chưa duyệt
```

* Welsh powell
- degs = sắp xếp giảm dần theo bậc
- color = 0
- while degs != []:
  v = degs.pop(0)
  tô màu v = color
  u = lấy tất cả các đỉnh ko kề vs v
  xét các đỉnh kề vs u, xem có tồn tại đỉnh nào cùng màu vs v ko, nếu ko thì tô đỉnh u này bằng color

  color += 1

Algorithm 3 Tìm đường đi từ đỉnh v_s đến v_e

```
procedure BFS_FIND_PATH(v_s, v_e)
    queue ← v_s
    while queue ≠ ∅ do
        v ← queue
        Duyệt đỉnh v
        if v == v_e then
            In ra đường và kết thúc
        for mỗi đỉnh u kề với đỉnh v do
            if đỉnh u chưa duyệt và không có trong queue then
                pre[u] = v
                queue ← u
```

Algorithm 1 Tìm đường đi P từ đỉnh v_s đến v_e

```
1: function DFS_FIND_PATH(v_s, v_e)
2:     Duyệt v_s
3:     if v_s == v_e then
4:         return true
5:     for mỗi đỉnh v kề với đỉnh v_s do
6:         if v chưa được duyệt then
7:             pre[v] = v_s
8:             if DFS_FIND_PATH(v, v_e) then
9:                 return true
10:    return false
```

Algorithm 7 Thuật toán Dijsktra

```
1: procedure DIJKSTRA_FIND_PATH(v_s, v_e)
2:     priority_queue ← v_s (với v_s.d = 0 và v_s.prev = null)
3:     while priority_queue ≠ ∅ do
4:         v ← priority_queue
5:         Duyệt đỉnh v
6:         if v = v_e then
7:             In ra đường và kết thúc
8:         for mỗi đỉnh u kề với đỉnh v do
9:             if đỉnh u chưa duyệt then
10:                if u ∈ priority_queue then
11:                    cập nhật u.d và u.pre nếu tốt hơn
12:                else
13:                    u.pre ← v và u.d ← v.d + l(v, u)
14:                    priority_queue ← u
```

* Hueristic 2
- v = node có bậc cao nhất
- while còn có đỉnh chưa dc tô màu:
  đánh dấu v bằng mã màu nhỏ nhất và mã màu này ko bị cấm
  hạ bậc v thành 0
  giảm bậc các node kề v đi 1 đồng thời cấm các node này tô màu của node v
  v = tìm lại node có bậc cao nhất

Algorithm 8 Tìm tất cả đường đi ngắn nhất

```
1: procedure DIJKSTRAFINDALLPATH(v_s, v_e)
2:     priority_queue ← v_s (với v_s.d ← 0 và v_s.prev_list ← null)
3:     while priority_queue ≠ ∅ do
4:         v ← priority_queue
5:         Duyệt đỉnh v
6:         if v = v_e then
7:             In ra đường và kết thúc
8:         for mỗi đỉnh u kề với đỉnh v do
9:             if đỉnh u chưa duyệt then
10:                if u ∈ priority_queue then
11:                    if u.d > v.d + l(v, u) then
12:                        u.d ← v.d + l(v, u)
13:                        u.pre_list ← v
14:                    if u.d = v.d + l(v, u) then
15:                        ADD(u.pre_list,v)
16:                else
17:                    u.pre_list ← v và u.d ← v.d + l(v, u)
18:                    priority_queue ← u
```

* Hierholzer
Def
  - Nếu tất cả các đỉnh đều có bậc chẵn thì chạy dc
  - res = []
  - bỏ node start nào stack
  - while stack != empty:
    - top = stack.top
    - if bậc của top == 0:
      - stack.pop()
      - thêm đỉnh này vào res
    - else:
      - xóa cạh đ.tiên kề vs top, tức cạnh (top, u)
      - thêm u vào stack