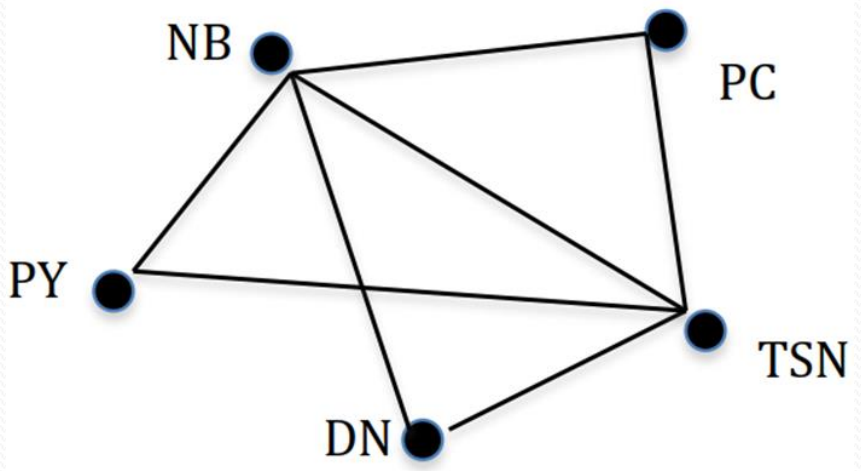
Đồ thị

Nội dung

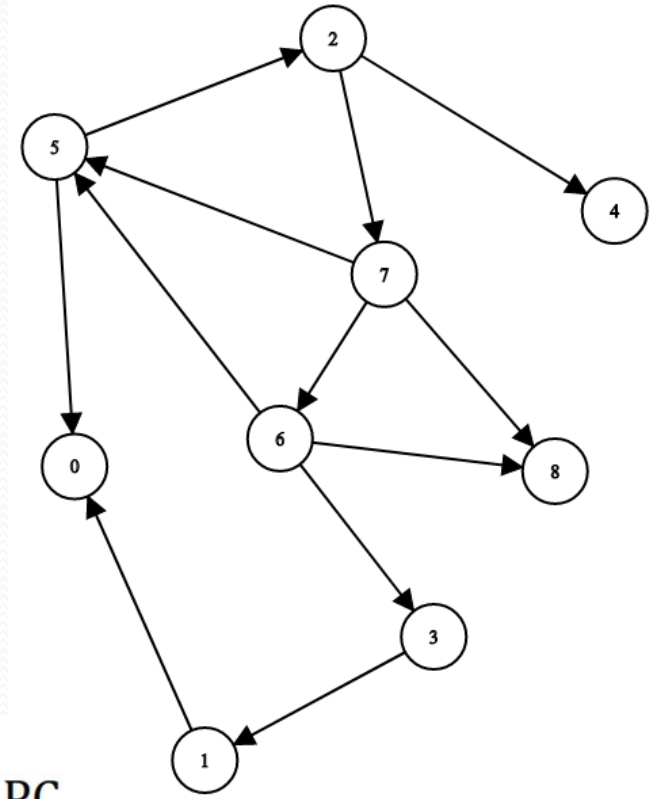
- Khái niệm
- Tổ chức dữ liệu
  - Ma trận kề
  - Danh sách kề
  - Danh sách cạnh
- Duyệt đồ thị
  - Duyệt theo chiều sâu
  - Duyệt theo chiều rộng
- Bài tập

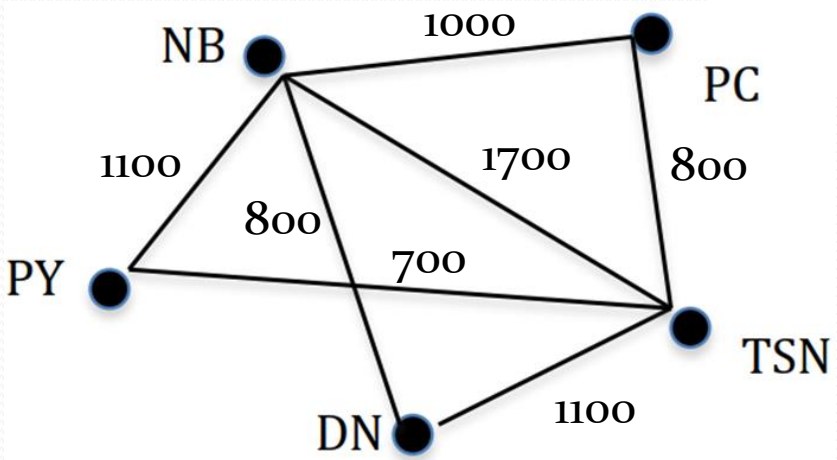
Khái niệm

- Đồ thị G là 1 cặp gồm 2 tập hợp <V, E>. Trong đó:
  - V: tập hữu hạn các phần tử, mỗi phần tử gọi là một đỉnh.
  - E: tập các cặp phần tử của V, gọi là các cạnh.
- Ví dụ: cần thể hiện thông tin về các đường bay đang hoạt động của các sân bay: Nội Bài, Tân Sơn Nhất, Đà Nẵng, Phú Yên, Phù Cát.
  - Tập đỉnh là các sân bay V= {NB, TSN, DN, PY, PC}
  - Tập cạnh các đường bay giữa các sân bay trong V đang hoạt động E = {(NB, TSN), (NB, DN), (NB,PY), (NB, PC), (TSN, DN), (TSN, PY), (TSN, PC)}

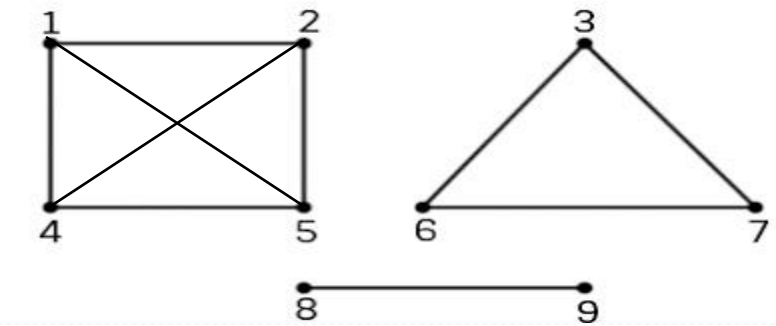


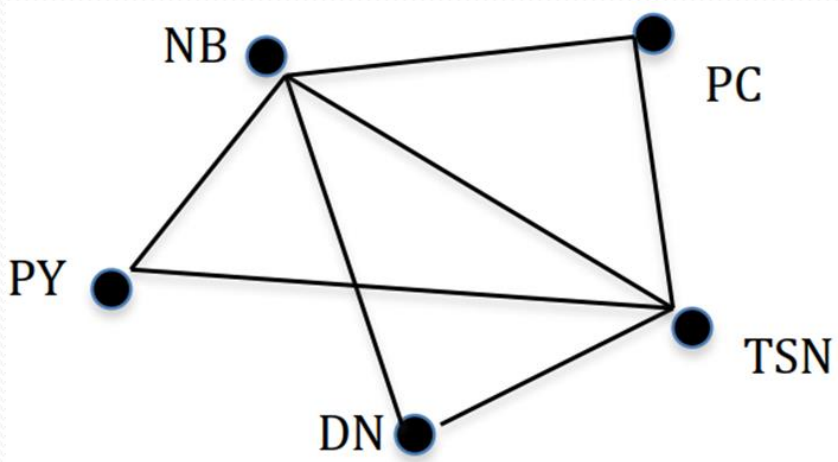
- Ví dụ: đồ thị "biết nhà" của 9 người.
- Đồ thị có hướng: mỗi cạnh xác định hướng giữa hai đỉnh. Khi đó cạnh (a, b) thuộc đồ thị thì a gọi là đỉnh đầu, b gọi là đỉnh cuối.
- Đồ thi vô hướng: các cạnh không xác định hướng.
- Đồ thị có trọng số: mỗi cạnh được gắn với 1 số, gọi là trọng số của cạnh.
- Đỉnh kề: đỉnh y gọi là đỉnh kề của đỉnh x trong đồ thị nếu (x, y) là một cạnh của đồ thị.





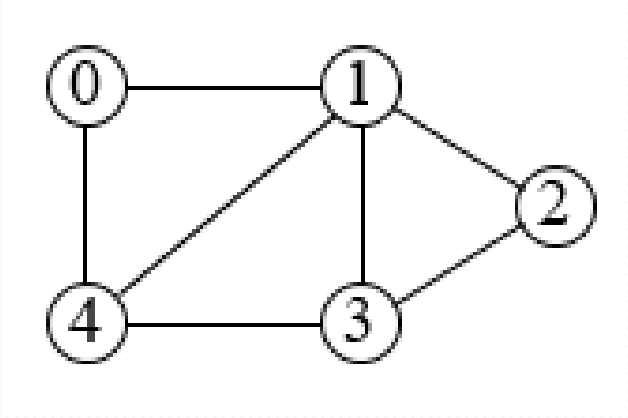
- Đường đi: đường đi trong đồ thị là một dãy các đỉnh v1, v2, …, vk. Trong đó (vi, vi+1) là cạnh với i=1, 2, .., k-1.
- Ví dụ đồ thị biểu diễn các đường bay: NB, TSN, PY, NB, DN là một đường đi.
- Đường đi đơn: là đường đi mà một cạnh không đi qua nhiều hơn 1 lần.
- Chu trình: là đường đi mà đỉnh xuất phát trùng với đỉnh kết thúc.
- Liên thông (đồ thị vô hướng):
  - Hai đỉnh của đồ thị gọi là liên thông nếu có đường đi giữa hai đỉnh.
  - Đồ thị liên thông nếu hai đỉnh bất kỳ liên thông.





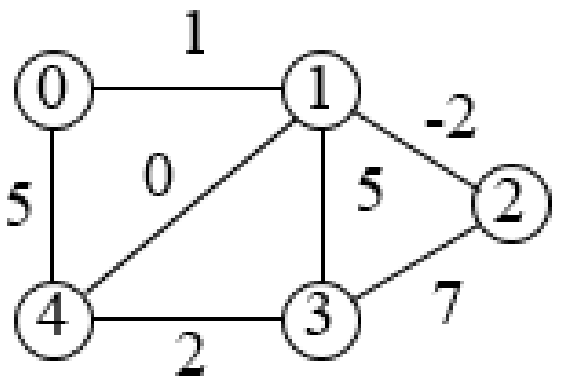
Tổ chức dữ liệu

- Ma trận kề
- Đánh số các đỉnh của đồ thị bằng các số: 0, 1, 2, ..., n-1.
- Dùng 1 mảng 2 chiều a_{nxn} để biểu diễn các cạnh:
  - a_{ij} = 1 nếu có cạnh từ i đến j
  - a_{ij} = 0 nếu không có cạnh từ i đến j
- Ví dụ:



	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	1	1
2	0	1	0	1	0
3	0	1	1	0	1
4	1	1	0	1	0

- Nếu đồ thị có trọng số thì:
  - a_{ij} = trọng số của cạnh (i,j) nếu có cạnh từ i đến j
  - a_{ij} = vô cùng nếu ngược lại



Khai báo tổ chức dữ liệu:

```
#define MAX_VERTICES 100
struct AdjMatrixGraph
{
   int numVertices;
   int adjMatrix[MAX_VERTICES][MAX_VERTICES];
};
```

	0	1	2	3	4
0	∞	1	∞	∞	5
1	1	∞	-2	5	0
2	∞	-2	∞	7	∞
3	∞	5	7	∞	2
4	5	0	∞	2	∞

Nhận xét

- Ma trận kề thuận tiện kiểm tra đỉnh kề adjMatrix[i][j]=1 thì j kề i.
- Ma trận kề lãng phí bộ nhớ với đồ thị vô hướng và đồ thị thưa.

Đỉnh 2 có kề với đỉnh 4 không?

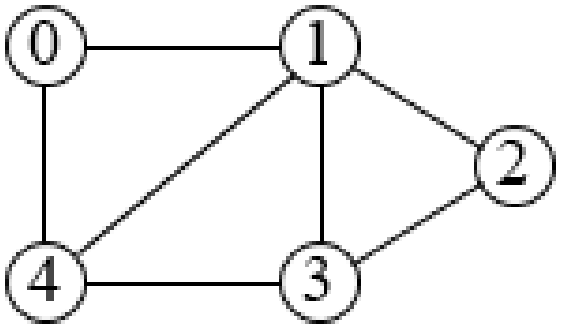
Những đỉnh kề với đỉnh 3?

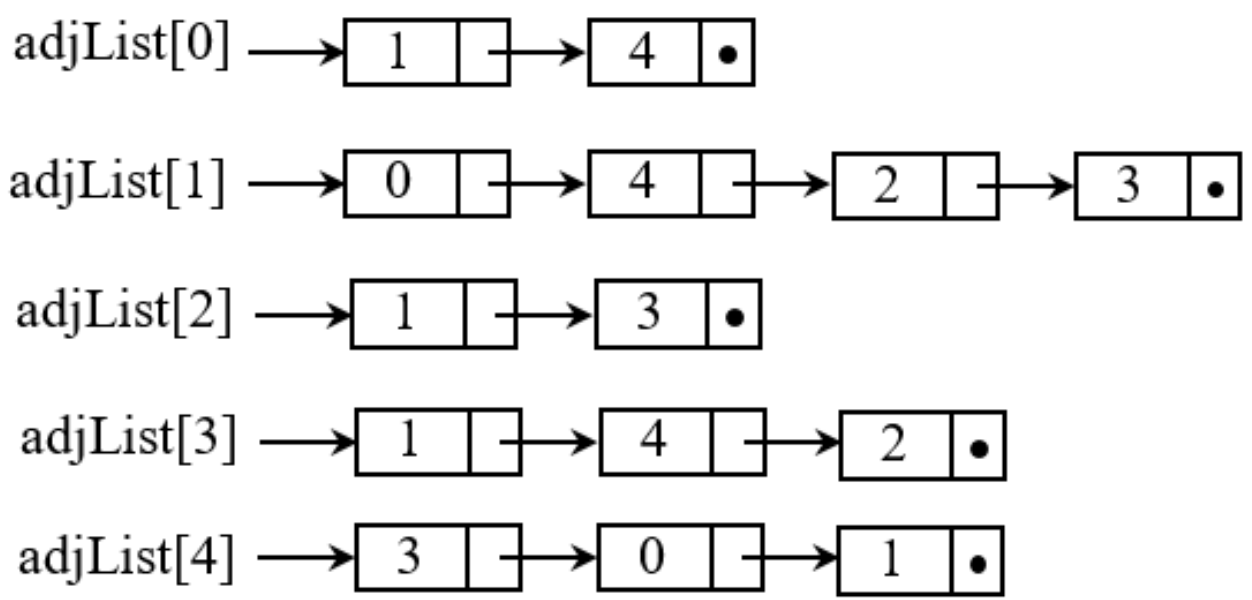
	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	1	1
2	0	1	0	1	0
3	0	1	1	0	1
4	1	1	0	1	0

	0	1	2	3	4
0	∞	1	∞	∞	5
1	1	∞	-2	5	0
2	∞	-2	∞	7	∞
3	∞	5	7	∞	2
4	5	0	∞	2	∞

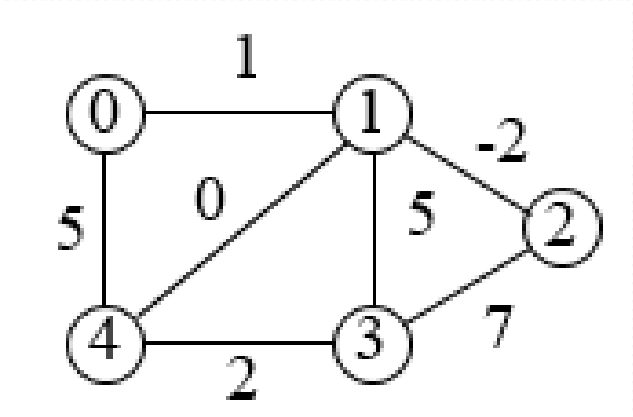
Danh sách kề

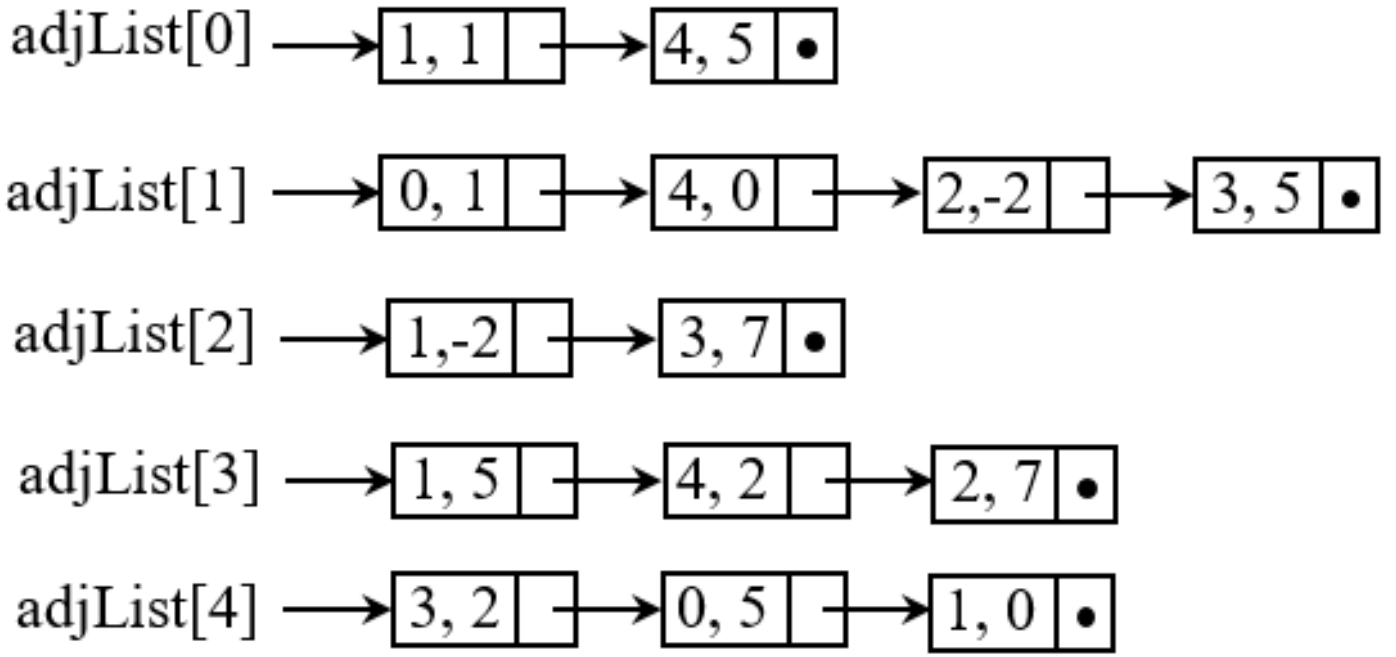
- Mỗi đỉnh lưu danh sách các đỉnh kề với nó trong một dslk đơn.
- Ví dụ:





- Với đồ thị có trọng số:






```
struct AdjList
{
   int adjVertex;
   float weight;
   AdjList* next;
};
```

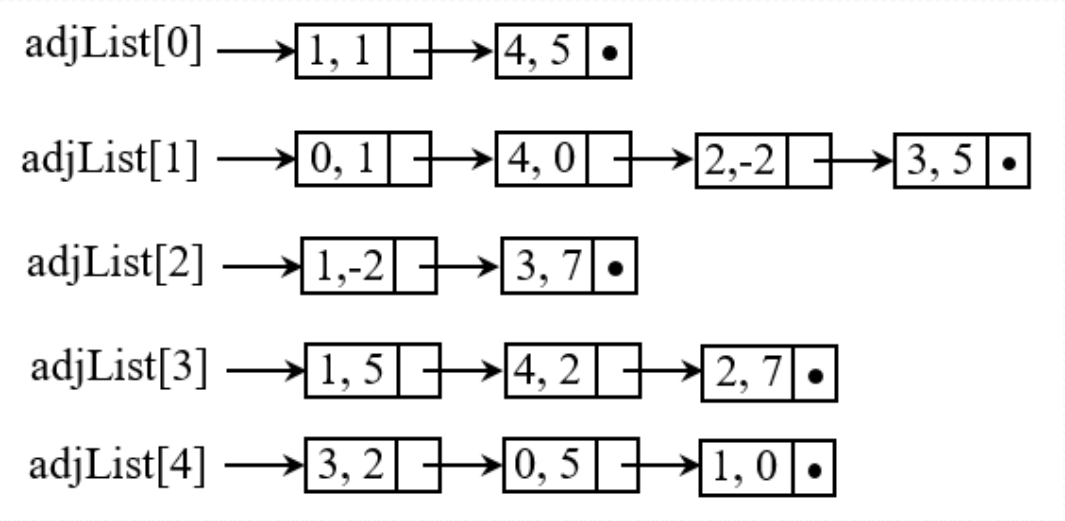
```
struct AdjListGraph
{
   int numVertices;
   AdjList* adjList[MAX_VERTICES];
};
```

Nhận xét

- Danh sách kề tiết kiệm bộ nhớ hơn ma trận kề.
- Để kiểm tra đỉnh y kề đỉnh x không phải tìm đỉnh y trong danh sách kề đỉnh x.
- Danh sách kề thuận lợi thao tác duyệt các đỉnh kề của một đỉnh.

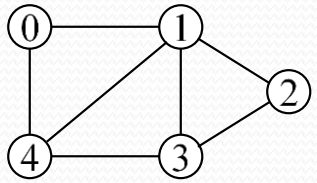
Đỉnh 2 có kề với đỉnh 4 không?

Những đỉnh kề với đỉnh 3?

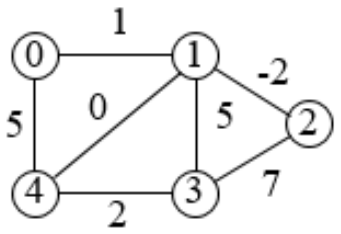


Danh sách cạnh

- Đánh số các cạnh của đồ thị 0, 1,...M-1
- Lưu danh sách các cạnh của đồ thị, mỗi cạnh gồm đỉnh đầu, đỉnh cuối và trọng số (nếu có).



	startVertex	destVertex
0	0	1
1	0	4
2	1	2
3	1	3
4	1	4
5	2	3
6	3	4



startVertex	destVertex	weight
0	1	1
0	4	5
1	2	-2
1	3	5
1	4	0
2	3	7
3	4	2

```
struct Edge
{
   int startVertex, destVertex;
   float weight;
};
```

```
struct EdgeListGraph
{
   int numVertices, numEdges;
   Edge edgeList[MAX_EDGES];
};
```

Nhận xét

- Danh sách cạnh không thuận lợi cho việc kiểm tra đỉnh kề và duyệt đỉnh kề.
- Danh sách cạnh thuận lợi cho các thao tác với cạnh như tìm cây khung.

Đỉnh 2 có kề với đỉnh 4 không?

Những đỉnh kề với đỉnh 3?

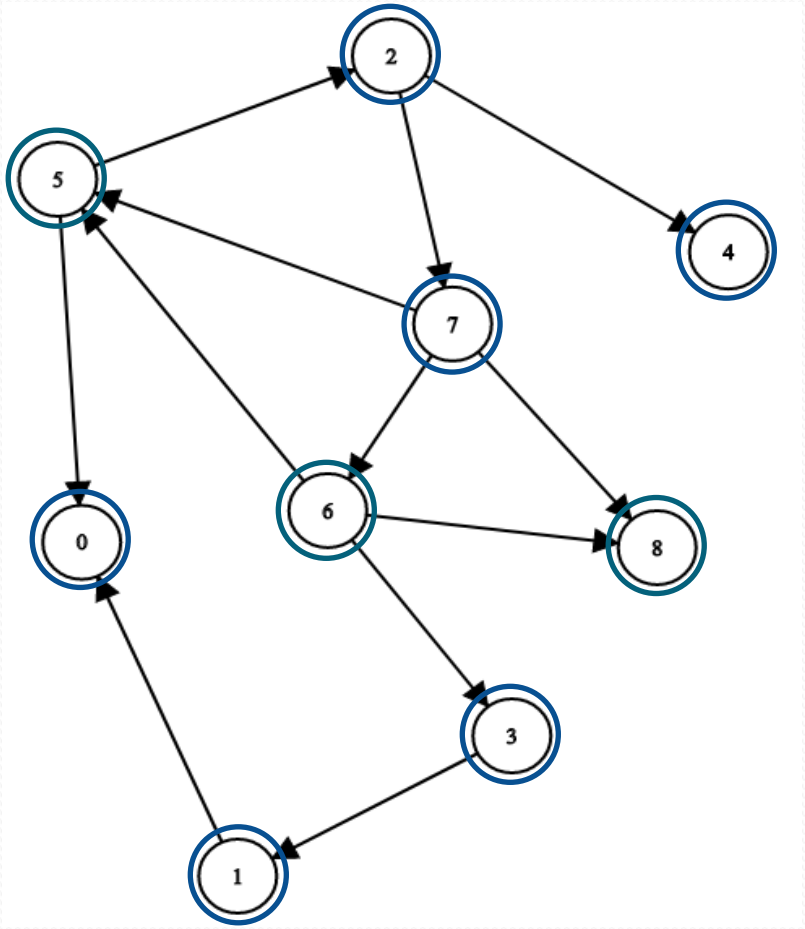
startVertex	destVertex	weight
0	1	1
0	4	5
1	2	-2
1	3	5
1	4	0
2	3	7
3	4	2

Duyệt đồ thị

- Duyệt đồ thị là xuất phát từ 1 đỉnh, dựa vào các cạnh lần lượt thăm các đỉnh, mỗi đỉnh 1 lần.
- Từ đỉnh u thăm được đỉnh v nếu (u, v) là một cạnh của đồ thị
- Thuật toán duyệt đồ thị theo chiều sâu xuất phát từ đỉnh v:
- Input: Đồ thị g, đỉnh xuất phát v
- Output: thứ tự các đỉnh được thăm
- Action:
  - Thăm đỉnh v
  - Với mỗi đỉnh w kề v, nếu w chưa thăm thì duyệt đồ thị theo chiều sâu xuất phát từ w.

Ví dụ

- Duyệt theo chiều sâu xuất phát từ 7:
- 7, 5, 0, 2, 4, 6, 3, 1, 8
- Duyệt theo chiều sâu xuất phát từ một đỉnh có thăm hết các đỉnh của đồ thị không?



Cài đặt

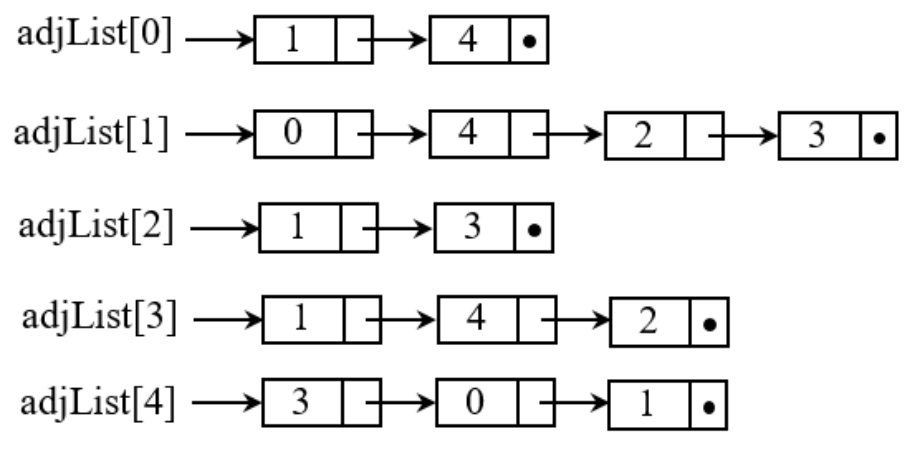
+ Dùng mảng visited[] để đánh dấu các đỉnh đã thăm.
+ Đồ thị biểu diễn bằng ma trận kề:

```
void DFS(AdjMatrixGraph g, int v, bool visited[])
{
    visited[v] = true;
    for (int w = 0; w < g.numVerties; w++)
    {
        if (g.adjMatrix[v][w] == 1 && !visited[w])
            DFS(g, w, visited);
    }
}
```

	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	1	1
2	0	1	0	1	0
3	0	1	1	0	1
4	1	1	0	1	0

+ Đồ thị biểu diễn bằng danh sách kề:

```
void DFS(AdjListGraph g, int v, bool visited[])
{
    visited[v] = true;
    AdjList* p = g.adjList[v];
    while (p != nullptr)
    {
        if (!visited[p->adjVertex])
            DFS(g, p->adjVertex, visited);
        p = p->next
    }
}
```



Đồ thị biểu diễn bằng ds cạnh

```
void DFS(EdgeListGraph g, int v, bool visited[]){
      visited[v] = true;
      for(int i = 0; i < g.numEdges; i++)
      {
            if(g.edgeList[i].startVertex == v)
            {
                  w =g.edgeList[i].destVertex;
                  if(!visited[w])
                        DFS(g, w, visited);
            }
      }
}
```

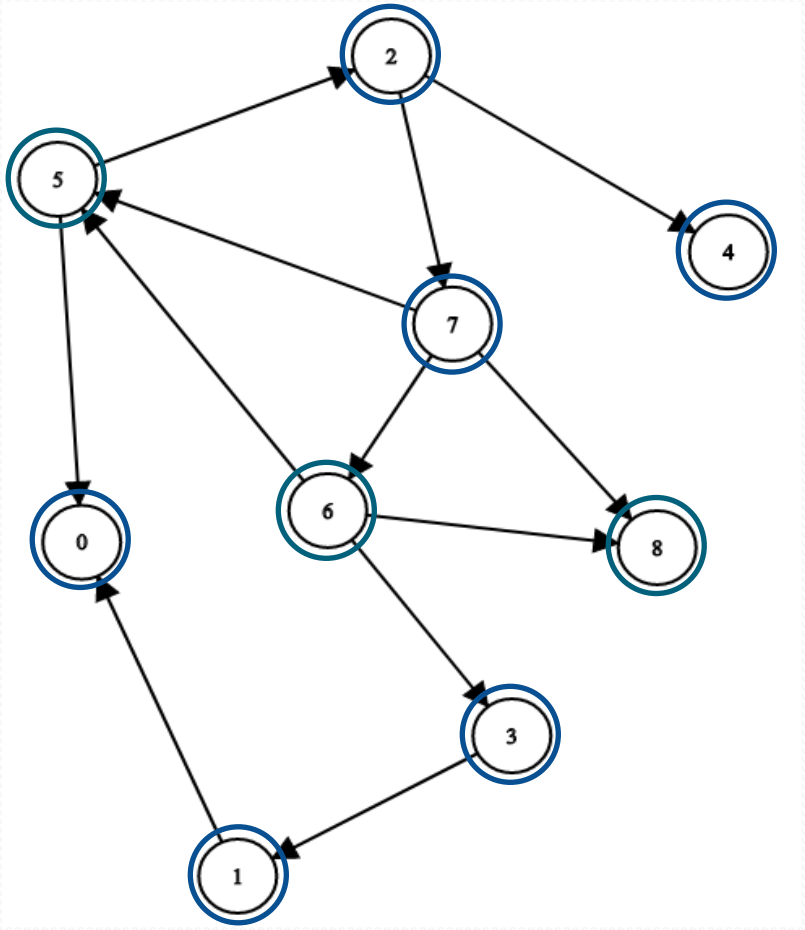
startVertex	destVertex	weight
0	1	1
0	4	5
1	2	-2
1	3	5
1	4	0
2	3	7
3	4	2

Duyệt theo chiều rộng

- Thuật toán duyệt đồ thị theo chiều rộng xuất phát từ đỉnh v:
- Input: đồ thị g, đỉnh xuất phát v
- Output: thứ tự thăm các đỉnh xuất phát từ v
- Action:
- Thăm đỉnh v
- Thăm lần lượt những đỉnh kề của v mà chưa thăm
- Với mỗi đỉnh vừa thăm, thăm lần lượt các đỉnh kề của nó mà chưa thăm
- Lặp lại bước trên cho đến khi không thăm thêm được đỉnh nào

Ví dụ

- Duyệt theo chiều rộng xuất phát từ 7:
- 7, 5, 6, 8, 0, 2, 3, 4, 1
- Dùng hàng đợi q lưu các đỉnh trong quá trình duyệt.
  - Đầu tiên đưa đỉnh xuất phát vào q
  - Lặp khi q khác rỗng
    - Lấy 1 đỉnh u từ q ra thăm
    - Đưa những đỉnh kề của u chưa thăm vào q



7	5	6	8	0	2	3	4	1

- Cài đặt: dùng một hàng đợi lưu các đỉnh trong quá trình duyệt.
- Đồ thị biểu diễn bằng ma trận kề:

```
void BFS(AdjMatrixGraph g, int v){
 queue<int> q; int u;
 bool visted[g.numVertices] = {false};
 q.push(v);
 vistied[v] = true ;
 while (!q.empty()) {
  u = q.front(); q.pop();
  visit(u);
  for(int w = 0; w < g.numVertices; w++)
       if(g.adjMatrix[u][w] == 1 && !visited[w]){
         q.push(w); visited[w] = true;
       }
  }
}
```

- Duyệt theo chiều rộng trên đồ thị biểu diễn bằng danh sách kề:

```
void BFS(AdjListGraph g, int v){
 queue<int> q; int u;
 bool visted[g.numVertices] = {false};
 q.push(v);
 vistied[v] = true ;
 while (!q.empty()) {
  u = q.front(); q.pop();
  visit(u);
  AdjList* p = g.adjList[u];
  while (p != nullptr){
    int w = p->adjVertex;
    if(!visited[w]) {
       q.push(w); visited[w] = true;
    }
    p = p->next;
  }
}
```

Bài tập

Bài 9.1 Cho đồ thị có hướng, biểu diễn bằng danh sách kề.

Trình bày thuật toán và cài đặt các thao tác:

a) Đếm số cạnh xuất phát từ v.

b) Kiểm tra đỉnh y có kề đỉnh x không?

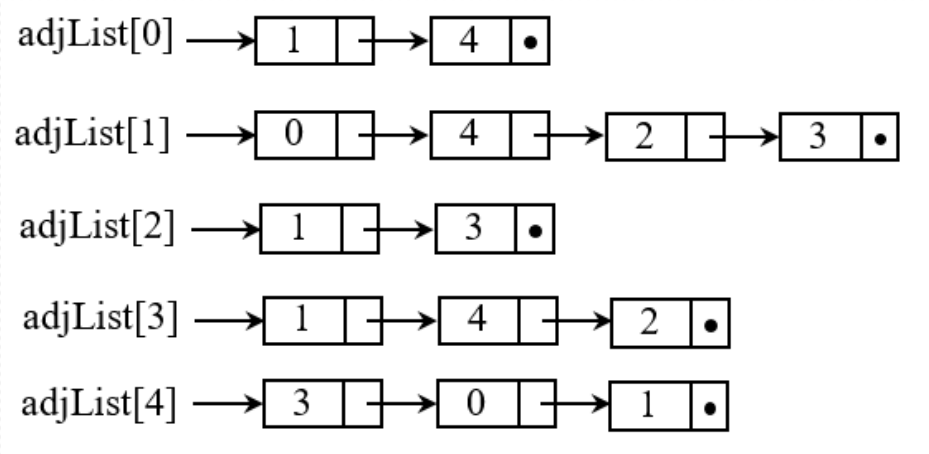
c) Cho biết từ x có thể thăm những đỉnh nào của đồ thị?

d) Cho biết từ x không thể thăm những đỉnh nào của đồ thị?

Khai báo

```
struct AdjList
{
   int adjVertex;
   float weight;
   AdjList* next;
};
```

```
struct AdjListGraph
{
   int numVertices;
   AdjList* adjList[MAX_VERTICES];
};
```



Câu a)

- Đếm số cạnh xuất phát từ v.
- Thuật toán:
- Input: đồ thị g, đỉnh v
- Output: số cạnh xuất phát từ v
- Action
- dem=0
- Duyệt danh sách kề đỉnh v
  - Tăng biến dem lên 1
- Trả về biến dem


```
int countOutEdges(AdjListGraph g, int v)
{
    int count = 0;
    AdjList *p = g.adjList[v];
    while (p != nullptr){
      count++;
      p = p->next;
    }
    return count;
}
```

Câu b)

- Kiểm tra đỉnh y có kề đỉnh x không?
- Thuật toán:
- Input: đồ thị g, 2 đỉnh x, y
- Output: True nếu y kề x, False ngược lại
- Action:
- Duyệt danh sách kề đỉnh x
  - Nếu đỉnh đang xét là y thì trả về True
- Trả về False

```
int isAdjacent(AdjListGraph g, int x, int y)
{
   AdjList *p = g.adjList[x];
   while (p != nullptr){
     if (p->adjVertex == y)
       return true;
     p = p->next;
   }
   return false;
}
```

Câu c)

- Cho biết từ x có thể thăm những đỉnh nào của đồ thị?
- Thuật toán:
- Input: đồ thị g, đỉnh x
- Output: các đỉnh thăm được từ x
- Action:
- Duyệt đồ thị xuất phát từ x
  - Mỗi khi thăm 1 đỉnh thì đánh dấu đỉnh đó đã thăm
- Duyệt các đỉnh, in các đỉnh đã thăm

```
void DFS(AdjListGraph g, int v, bool
visited[]){
 visited[v] = true;
 AdjList* p = g.adjList[v];
 while (p != nullptr){
  if (!visited[p->adjVertex])
     DFS(g, p->adjVertex, visited);
  p = p->next
 }
}

void Visit(AdjListGraph g, int x){
 bool visited[g.numVertices] = {false};
 DFS(g, x, visited);
 for (int i = 0; i < g.numVertices; i++)
   if (visted[i])
    cout << i << " “;
}
```

Câu d)

- Cho biết từ x không thể thăm những đỉnh nào của đồ thị?
- Hướng dẫn:
  - Dùng thuật toán duyệt
  - Kiểm tra mảng visited[] đỉnh nào chưa thăm là không thăm được từ x.

Bài tập

Bài 9.2 Cho một nhóm gồm N người đánh số từ 0 đến N-1. Cho biết người v là F0 và thông tin tiếp xúc gần của những người trong N người. Hãy trình bày thuật toán và viết các hàm trên đồ thị biểu diễn bằng danh sách cạnh:

a) Cho biết những người là F1
b) Cho biết những người đã tiếp xúc với người v trực tiếp hay qua trung gian
c) Cho biết F_i của từng người.

Bài tập

Bài 9.3 Trong một lớp học có N sinh viên được đánh số từ 0 đến N-1, trong đó mỗi sinh viên biết số điện thoại của một số sinh viên trong lớp (gọi là biết trực tiếp). Sinh viên a không biết trực tiếp số điện thoại của sinh viên c nhưng biết số điện thoại của sinh viên b và sinh viên b biết số điện thoại của sinh viên c thì sinh viên a gọi là biết gián tiếp số điện thoại của sinh viên c. Cho trước thông tin biết số điện thoại của N sinh viên. Hãy viết các hàm thực hiện:

a) Cho biết sinh viên x biết trực tiếp số điện thoại sinh viên y không?

b) Cho biết sinh viên x biết trực tiếp hoặc gián tiếp số điện thoại của sinh viên z không?

c) Cho biết sinh viên x có thể biết được số điện thoại hết cả lớp không?

d) Cho biết những sinh viên mà không ai biết số điện thoại.

e) Cho biết có sinh viên nào biết số điện thoại hết cả lớp không?

f) Tìm tập hợp ít sinh viên nhất mà những sinh viên này biết hết số điện thoại của cả lớp.

Tổng kết

- Đồ thị là cấu trúc dữ liệu cho phép mô tả các mối quan hệ theo từng cặp
- 3 cách biểu diễn đồ thị: ma trận kề, danh sách kề, danh sách cạnh.
- Duyệt đồ thị: theo chiều sâu, theo chiều rộng.