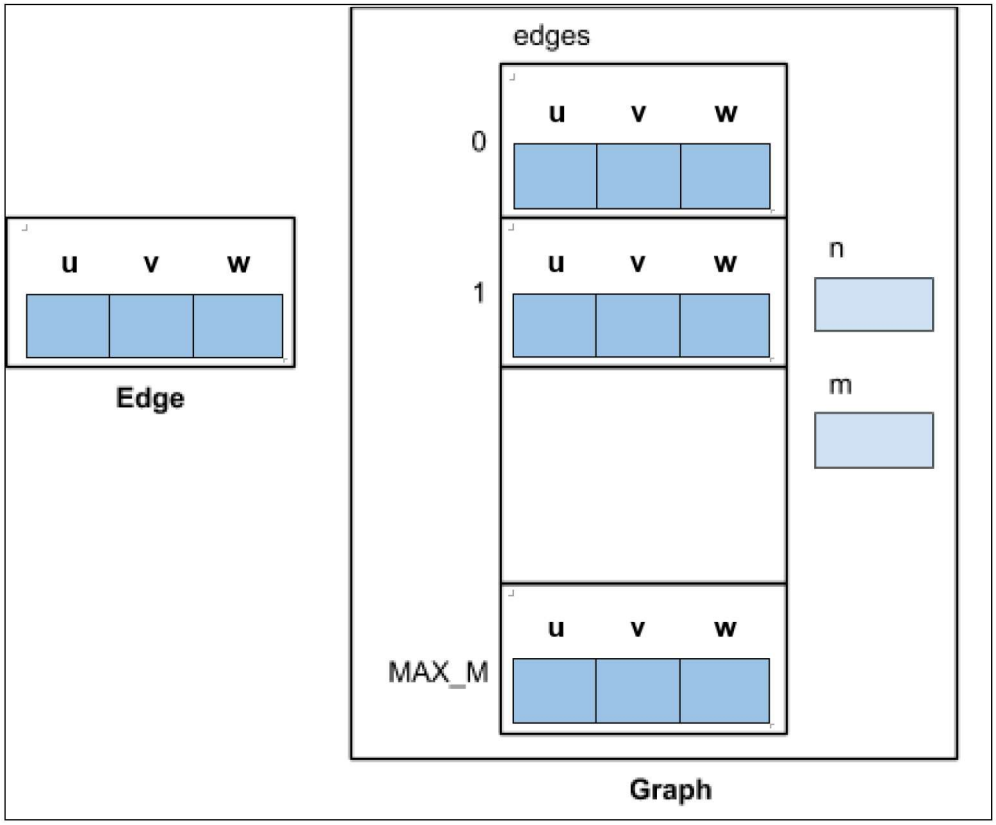
Biểu diễn bằng danh sách cung trọng số

- **Ý tưởng:**
  - Mỗi cung lưu 2 đỉnh đầu mút (endpoint) và trọng số (w) của nó
  - Lưu tất cả các cung của đồ thị vào một danh sách (*danh sách đặc* hoặc *danh sách liên kết*)

- **Sơ đồ tổ chức dữ liệu biểu diễn đồ thị bằng danh sách cung trọng số:**



Cài đặt biểu diễn đồ thị bằng danh sách cung có trọng số:

```
#define M 1000

typedef struct {
  int u, v; // đỉnh đầu v, đỉnh cuối v
  int w;    // trọng số w
} Edge;

typedef struct {
  int n, m;          // n: đỉnh, m: cung
  Edge edges[MAX_M]; // lưu các cung của đồ thị
} Graph;
```

Khởi tạo đồ thị:

```
void init_graph(Graph* G, int n) { G->n = n;
  G->m = 0;
}
```

Thêm 1 cung vào đồ thị:

```
void add_edge(Graph* G, int u, int v, int w) {
  G->edges[G->m].u = u;
  G->edges[G->m].v = v;
  G->edges[G->m].w = w;
  G->m++;
}
```

Giải thuật Bellman – Ford

Tương tự giải thuật Dijkstra, giải thuật Bellman – Ford cho phép tìm đường đi ngắn nhất từ 1 đỉnh s đến các đỉnh khác.

Ý tưởng:

- Khởi tạo: giống Dijkstra
  - pi[u]=∞ vớI mọi u != s;
  - pi[s] = 0, p[s] = -1;
- Lặp n–1 lần
  - Duyệt qua tất cả các cung (u, v) và cập nhật pi[v] và p[v] nếu thoả điều kiện

```
if (pi[u] + L[u][v] < pi[v]) {
    pi[v] = pi[u] + L[u][v];
    p[v] = u;
}
```

Về cơ bản, giải thuật Bellman – Ford có cùng nguyên lý như giải thuật Dijkstra. Điểm khác biệt là ở mỗi lần lặp: giải thuật Dijkstra chọn ra đỉnh có pi[u] bé nhất và cập nhật các đỉnh kề của u; trong khi đó giải thuật Bellman – Ford duyệt qua kết tất cả các cung (u, v) và cập nhật các đỉnh v (gần như cập nhật tất cả các đỉnh của đồ thị).

Vì thế để thuận tiện cho giải thuật Bellman – Ford ta phải biểu diễn đồ thị sao cho duyệt qua các cung của nó dễ dàng. Cách đơn giản nhất là lưu **danh sách các cung** của đồ thị.

Cài đặt Giải thuật Bellman – Ford:

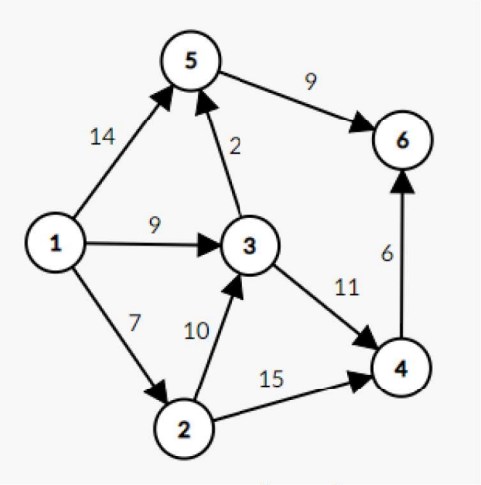
```
#define INFINITY 9999999

int pi[MAXN];
int p[MAXN];

void BellmanFord(Graph* pG, int s) {
  int u, v, w, it, k;
  for (u = 1; u <= pG->n; u++) {
    pi[u] = INFINITY;
  }
  pi[s] = 0;
  p[s] = -1; //trước đỉnh s không có đỉnh nào cả

  // lặp n-1 lần
  for (it = 1; it < G->n; it++) {
    // Duyệt qua các cung và cập nhật (nếu thoả)
    for (k = 0; k < G->m; k++) {
      u = G->edges[k].u;
      v = G->edges[k].v;
      w = G->edges[k].w;
      if (pi[u] + w < pi[v]) {
        pi[v] = pi[u] + w;
        p[v] = u;
      }
    }
  }
}
```

Kết quả giải thuật Bellman – Ford: cách sử dụng kết quả giống giải thuật Moore – Dijkstra.

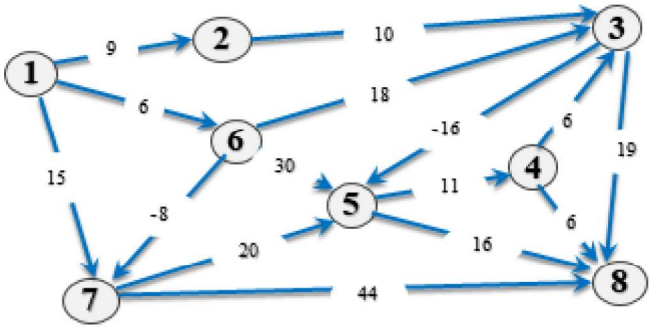
Phát hiện chu trình âm:

Giải thuật Bellman – Ford hơn giải thuật More – Dijkstra ở chỗ có thể chạy được với đồ thị có trọng số âm. Sau khi chạy xong giải thuật, ta có thể phát hiện được chu trình âm bằng cách duyệt qua các cung một lần nữa nếu tiếp tục cải tiến được pi[v] thì có nghĩa là đồ thị *có chu trình âm*.

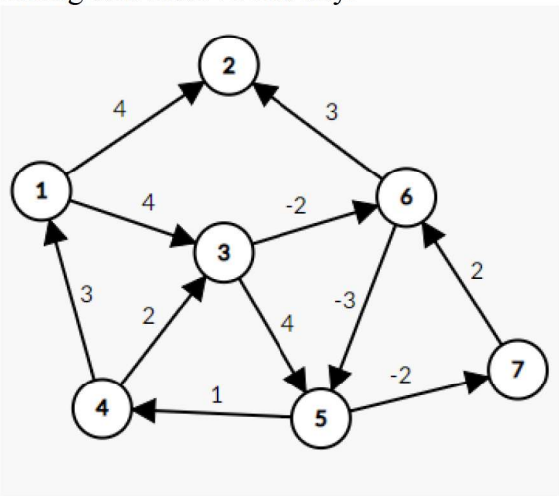
Bài tập 1: Hãy vẽ sơ đồ tổ chức dữ liệu biểu diễn đồ thị bằng danh sách cung có trọng số cho đồ thị sau đây:



Bài tập 2: Xem slide chương Tìm đường đi ngắn nhất sử dụng thuật toán Bellman-Ford, áp dụng cho đồ thị sau đây (Bài tập 2 không cần viết tay nộp cho Thầy, chủ yếu là xem nội dung trong slide và cố gắng hiểu thuật toán nhe em):



Bài tập 3: Cho đồ thị có hướng như hình vẽ sau đây:



a. Hãy vẽ sơ đồ tổ chức dữ liệu biểu diễn đồ thị bằng danh sách cung có trọng số cho đồ thị.

b. Hãy chạy thủ công giải thuật Bellman-Ford để tìm đường đi ngắn nhất từ đỉnh 1 đến tất cả các đỉnh còn lại.

c. Đồ thị trên có chứa chu trình âm. Em hãy chỉ ra một chu trình âm của đồ thị trên. (Biết rằng chu trình là chu trình có tổng trọng số trên chu trình đó mang giá trị âm)

d. Tiếp tục chạy thủ công đoạn code sau và kiểm tra điều `pi[u] + w < pi[v],` nếu điều kiện này thỏa thì đồ thị có chu trình âm.

```
// Kiểm tra chu trình âm bằng cách
// Duyệt qua các cung một lần nữa
for (k = 0; k < G->m; k++) {//đọc lại từng cung của đồ thị
  int u = G->edges[k].u;
  int v = G->edges[k].v;
  int w = G->edges[k].w;
  if (pi[u] + w < pi[v]) {//điều kiện tồn tại chu trình âm
    // Có chu trình âm
    …
    break;
  }
}
```