BUỔI 5. PHẦN 2 - LUỒNG CỰC ĐẠI TRÊN MẠNG

Mục đích:

- Cài đặt giải thuật đánh dấu Ford-Fullkerson tìm luồng cực đại trên mạng

Yêu cầu:

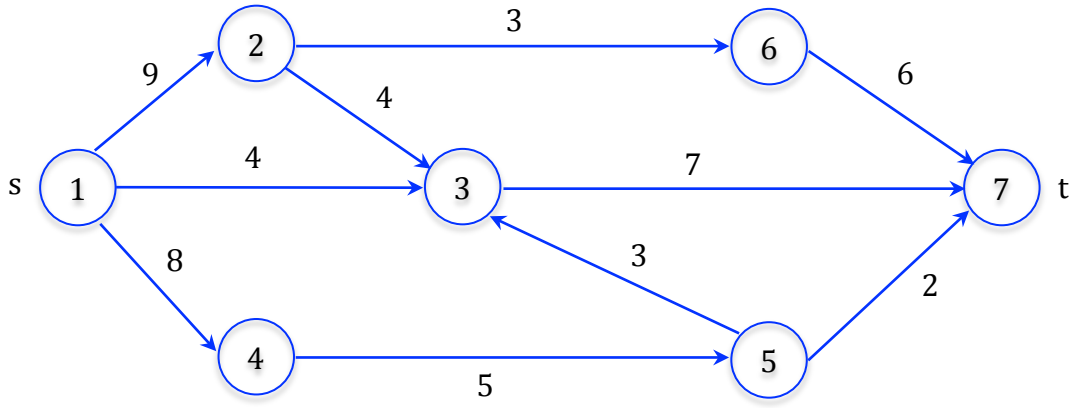
- Biết cách biểu diễn đồ thị
- Biết cách duyệt đồ thị (chiều rộng + chiều sâu)
- Biết sử dụng cấu trúc dữ liệu ngăn xếp + hàng đợi

5.1 Bài toán luồng cực đại trên mạng

Cho mạng được biểu diễn bằng đồ thị có hướng G = <V, E>. Mỗi cung e = (u, v) được gán một khả năng thông qua lớn nhất là c[u][v].

Quy ước: mạng chỉ có 01 đỉnh phát s và 01 đỉnh thu t.

Ví dụ: mạng có 7 đỉnh với đỉnh phát s = 1 và đỉnh thu t = 7. Khả năng thông qua của các cung được cho trên các cung tương ứng.



5.2 Giải thuật đánh dấu Ford – Fullkerson

Ý tưởng:

- Khởi tạo luồng f = 0: luồng chảy qua tất các cung đều = 0.
- Lặp:
  - Tìm đường tăng luồng (bằng cách đánh dấu các đỉnh)
  - Nếu tìm thấy => tăng luồng
  - Nếu không còn đường tăng luồng => dừng

Sau khi kết thúc, các đỉnh được đánh dấu thuộc về X0, các đỉnh chưa được đánh dấu thuộc về Y0, ta thu được lát cắt hẹp nhất (X0, Y0).

Cài đặt giải thuật

- Biểu diễn đồ thị: sử dụng phương pháp ma trận trọng số, để có thể lưu được luồng trên các cung ta sử dụng biến F.

```
#define MAXN 100
#define NO_EDGE 0
#define INF 9999999

typedef struct {
  int C[MAXN][MAXN]; //khả năng thông qua của cung
  int F[MAXN][MAXN]; //luồng trên cung
  int n;
} Graph;
```

- Các biến/cấu trúc dữ liệu bổ sung
    - Cấu trúc dữ liệu Label: lưu nhãn của một đỉnh, nhãn của các đỉnh: labels. Trường 'dir' cho biết nhãn là +, - hay chưa có nhãn (dir = 0).

```
typedef struct {
  int dir; // >0: +, <0: -, 0: chưa có nhãn
  int pre; //đỉnh trước
  int sigma; //lượng tăng luồng
} Label;
Label labels[MAXN]; //nhan cac dinh
```

    - Cấu trúc dữ liệu hàng đợi/ngăn xếp để lưu các đỉnh đã được đánh dấu nhưng chưa xét (xem giải thuật trong giáo trình).

Phát hoạ giải thuật (6 bước, xem giáo trình):

```
int FordFullkerson(Graph* G, int s, int t) {
  //I. Khởi tạo luồng = 0, gán F[u][v] = 0 với mọi u, v.
  init_flow(G);
  //II. Lặp
  Queue Q;
  do {
    //Bước 1 – xoá nhãn các đỉnh và gán nhãn cho s
    //Xoá tất cả các nhãn
    //Gán nhãn s: (+, s, oo)
    //Khởi tạo Q rỗng, Đưa s vào Q
    //Bước 2,3 – lặp gán nhãn cho các đỉnh
    while (Q is chưa rỗng) {
      //Lấy 1 đỉnh trong Q ra => x
      //Xét gán nhãn cho các đỉnh kề với x
      //Xét gán nhãn cho các đỉnh đi đến x
      //Nếu t được gán nhãn =>
      //  tìm được đường tăng luồng, thoát vòng lặp.
    }
    if (tìm được đường tăng luồng) {
      //Bước 4, 5, 6 - Tăng luồng
    } else
      break; //thoát vòng lặp
  } while (1);
}
```

Các công việc cần thực hiện:

1. Viết hàm init_flow(Graph* G): gán luồng F[u][v] = 0.

```
void init_flow(G) {
  //Gán g->F[u][v] = 0 với u, v = 1..n.
  …
}
```

2. Cài đặt cấu trúc dữ liệu Queue với các phép toán:
   - make_null_queue(Queue* Q): tạo hàng đợi rỗng
   - enqueue(Queue* Q, int x): thêm phần tử x vào cuối hàng đợi.
   - top(Queue* Q): trả về phần tử đầu hàng đợi.
   - dequeue(Queue* Q): xoá phần tử đầu hàng đợi.
   - empty(Queue* Q): kiểm tra hàng đợi rỗng hay không.

3. Chi tiết hoá bước 1 – xoá nhãn các đỉnh và gán nhãn cho s.

```
…
  do {
    //Bước 1 – gán nhãn s
    //1.1 Xoá tất cả các nhãn
    for (u = 1; u <= G->n; u++)
      labels[u[.dir = 0;

    //1.2 Gán nhãn s: (+, s, oo)
    labels[u].dir = +1;
    labels[u].pre = s;
    labels[u].sigma = INF; //sigma(s) = oo.
    //1.3 Khởi tạo Q rỗng, Đưa s vào Q
    make_null_queue(&Q);
    enqueue(&Q, s);

    …

  } while (1);
}
```

4. Chi tiết hoá bước 2, 3 – gán nhãn cho các đỉnh

Sử dụng kỹ thuật duyệt đồ thị để gán nhãn cho các đỉnh. Ta có thể sử dụng hàng đợi (queue) hoặc ngăn xếp (stack) hoặc hàng đợi ưu tiên (priority queue) để đánh dấu các đỉnh. Trong phần cài đặt này ta sử dụng hàng đợi để lưu các đỉnh đã được đánh dấu nhưng chưa xét. Sinh viên có thể cải tiến phần này để cài đặt bước đánh dấu bằng Stack hoặc hàng đợi ưu tiên theo sigma(x), ưu tiên tìm đường tăng luồng lớn nhất có thể.

```
…
  do {

    //Bước 2,3 – lặp gán nhãn cho các đỉnh
    int found = 0;
    while (!empty(&Q)) {
      //Lấy 1 đỉnh trong Q ra => x
      int x = top(&Q); dequeue(&Q);
      for (v = 1; v <= G->n; v++) {
        //Xét gán nhãn cho các đỉnh kề với x, cung thuận
        if (labels[v].dir == 0 && G->C[u][v] != NO_EDGE
                              && G->F[u][v] < G->C[u][v]) {
          labels[v].dir = +1; //cung thuận
          labels[v].pre = u;
          labels[v].sigma = min(labels[u].sigma,
                                G->C[u][v] - G->F[u][v]);
          enqueue(&Q, v);
        }
        //Xét gán nhãn cho các đỉnh đi đến x, cung nghịch
        if (labels[v].dir == 0 && G->C[v][u] != NO_EDGE
                               && G->F[v][u] > 0) {
          labels[v].dir = -1; //cung nghịch
          labels[v].pre = u;
          labels[v].sigma = min(labels[u].sigma,
                                        G->F[u][v]);
          enqueue(&Q, v);
        }
      }

      //Nếu t được gán nhãn =>
      //     tìm được đường tăng luồng, thoát vòng lặp.
      if (labels[t].dir != 0) {
        found = 1;
        break;
      }
    }
    …
  } while (1);
```

5. Chi tiết hoá bước 4, 5, 6 – tăng luồng

Sau khi thoát khỏi vòng while(!empty(&Q), có hai trường hợp xảy ra:

- found = 0: không gán nhãn được t. Điều này đồng nghĩa với việc không thể tìm được đường tăng luồng => giải thuật kết thúc.
- found = 1: đỉnh t được gán nhãn, dựa vào nhãn của t ta lần ngược theo trường pre của các nhãn để tăng/giảm luồng.

```
…
  do {
    …
    if (found == 1) {
      //Bước 4, 5, 6 - Tăng luồng
      int x = t;
      int sigma = labels[t].sigma;
      sum_flow += sigma; //luồng tăng thêm.

      while (x != s) {
        int u = labels[x].pre;
        if (labels[x].dir > 0)//tăng luồng
          G->F[u][x] += sigma;
        else //giảm luồng
          G->F[x][u] -= sigma;
        x = u;
      }

    } else
      break; //thoát vòng lặp
  } while (1);

  return sum_flow; //trả về luồng cực đại của mạng
}
```

6. Hàm main()

Trong hàm main(), ta đọc dữ liệu từ tập tin và gọi hàm FordFullkerson(&G, 1, n) với quy ước: đỉnh phát – 1 và đỉnh thu n. Sinh viên có thể sửa chương trình để cho phép chọn đỉnh phát và đỉnh thu bất kỳ.

```
int main() {
  Graph G;
  int n, m, u, v, e, c;
  //Dòng này dùng để đọc dữ liệu từ file bằng hàm scanf.
  freopen ("Ten file", "r", stdin);
  scanf("%d%d", &n, &m);
  init_graph(&G, n);

  for (e = 0; e < m; e++) {
    scanf("%d%d%d", &u, &v, &c);
    G.C[u][v] = c;
  }
  int max_flow = FordFullkerson(&G, 1, n);
  printf("Max flow: %d\n", max_flow);

  return 0;
}
```

Trong hàm main(), ta có sử dụng lệnh:

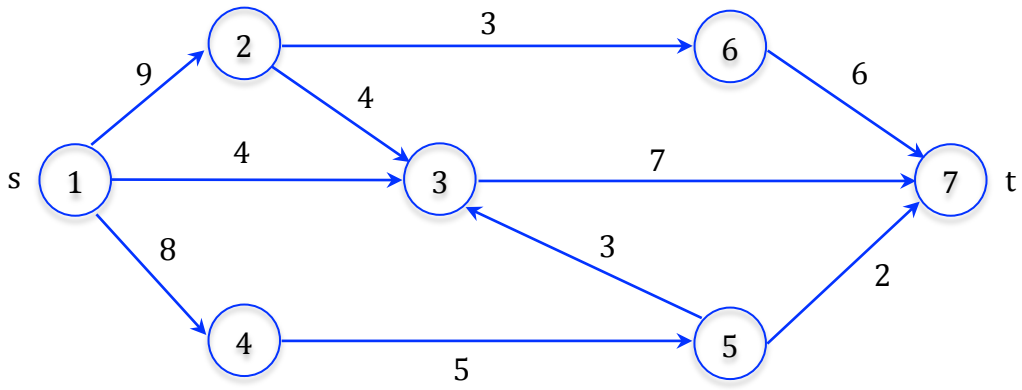
```
freopen ("Ten file", "r", stdin);
```

Lệnh này cho phép đọc dữ liệu từ tập tin giống như đọc dữ liệu từ bàn phím. Sau lệnh này ta có thể dùng scanf để đọc dữ liệu giống như đọc dữ liệu từ bàn phím. Nếu muốn đọc trực tiếp dữ liệu từ bàn phím, ta chỉ cần bỏ dòng này đi là xong !

Tập tin dữ liệu đầu vào giống như các đồ thị có trọng số:

```
n m
u1 v1 c1
u2 v2 c2
…
um vm cm
```

Bài tập 1. Tích hợp các phần trên thành chương trình hoàn chỉnh cho phép đọc một mạng từ tập tin và in ra màn hình luồng cực đại của mạng. Kiểm thử với mạng sau đây



Một phần của tập tin dữ liệu:

```
7 10
1 2 9
1 3 4
1 4 8
…
```

Bài tập 2. Cải tiến chương trình để in ra lát cắt hẹp nhất của mạng theo dạng:

(x1, x2, …/y1, y2, …)

với x1, x2, … là các đỉnh thuộc tập X0, y1, y2, … là các đỉnh thuộc tập Y0.

Gợi ý: sau khi vòng lặp do … while (1) kết thúc, sẽ có một số đỉnh được gán nhãn và một số đỉnh không được gán nhãn. Nhãn các đỉnh được lưu trong mảng labels.

Trong hàm main() ta có thể in các đỉnh đã được gán nhãn và các đỉnh chưa gán nhãn.

```
int main() {
  …
  int max_flow = FordFullkerson(&G, 1, n);
  printf("Max flow: %d\n", max_flow);
  …
  //In lát cắt (X0, Y0)
  //In X0, đỉnh đã có nhãn
  for (u = 1; u <= n; u++)
    if (labels[u].dir ! = 0) in u ra màn hình.

  //In Y0, đỉnh chưa có nhãn
  for (u = 1; u <= n; u++)
    if (labels[u].dir == 0) in u ra màn hình.


  return 0;
}
```

Bài tập 3. Cải tiến bài tập 1 bằng cách sử dụng ngăn xếp (stack) thay vì hàng đợi.

Bài tập 4. Cải tiến bài tập 1 bằng cách sử dụng chiến lược đánh dấu như sau: mỗi lần lấy 1 đỉnh từ trong Q, thay vì chọn theo chiến lược vào trước ra trước (FIFO – queue) hoặc vào trước ra sau (FILO – stack) ta sẽ chọn đỉnh có nhãn (+/-, pre, sigma) với **sigma lớn nhất** ra xét trước. Với chiến lược, mỗi lần tìm đường tăng luồng ta luôn có đường tăng luồng lớn nhất.