BUỔI 999. CÂY KHUNG CÓ HƯỚNG CÓ TRỌNG LƯỢNG NHỎ NHẤT

Mục đích:

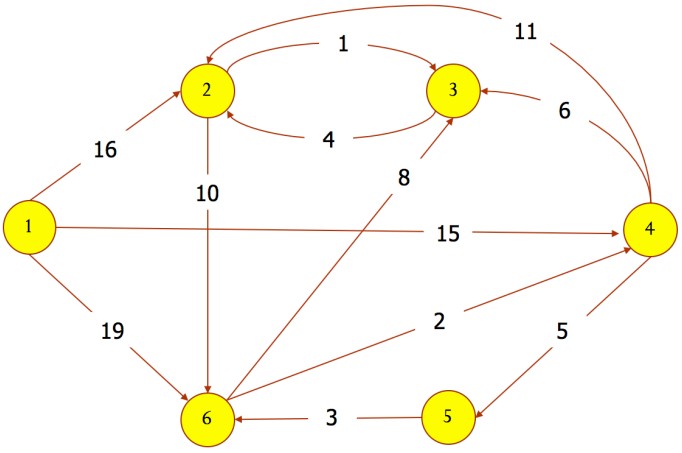
- Củng cố lý thuyết về cây khung có hướng
- Cài đặt giải thuật Chu-Liu/Edmonds

Yêu cầu:

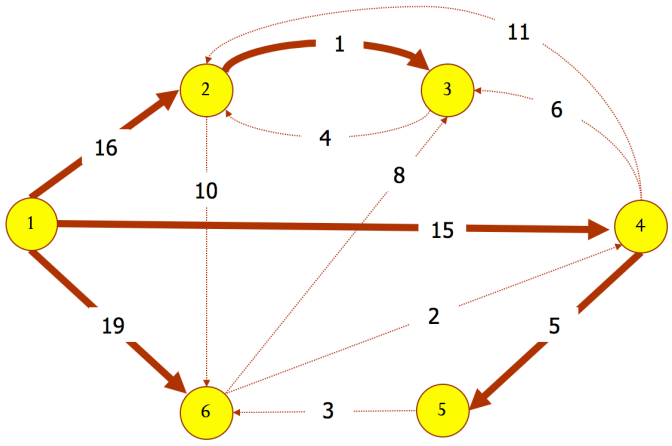
- Biết sử dụng ngôn ngữ lập trình C
- Biết cài đặt các cấu trúc dữ liệu cơ bản
- Biết biểu diễn đồ thị trên máy tính

999.1 Cây khung có hướng trọng lượng nhỏ nhất

Cho đồ thị có hướng G, gốc r. Luôn có đường đi từ r đến tất các đỉnh khác. Tìm cây T từ G sao cho tổng trọng số của các cung trong T nhỏ nhất.

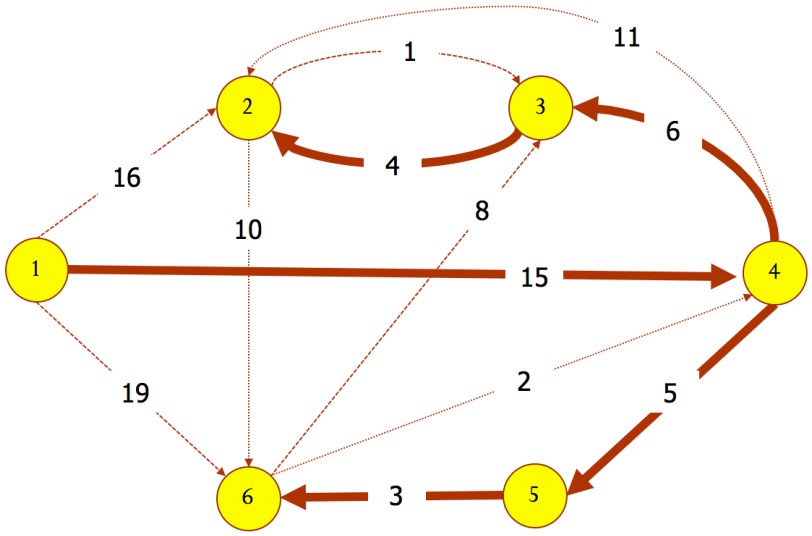


Một cây khung T của G là:



Trọng lượng cuả T = 16 + 1 + 15 + 19 + 5 = 56

Cây khung có trọng lượng nhỏ nhất: 15 + 5 + 6 + 4 + 3 = 33



999.2 Giải thuật Chu-Liu/Edmonds:

Ý tưởng: gồm hai pha

- Pha co
  - Xây dựng đồ thị xấp xỉ H_t từ G_t.
    - Với mỗi đỉnh của G_t chọn cung đi đến nó có trọng số bé nhất => thêm nó vào H_t.
  - Nếu H_t không chứa chu trình => chuyển sang pha giãn
    - Cần kiểm tra các chu trình có trong H_t
  - Co G_t thành đồ thị G_{t+1} dựa trên các chu trình có trong H_t.
    - Mỗi đỉnh trong chu trình H_t là tương ứng với các đỉnh mới
    - Với mỗi cung (u, v, w) trong G_t, thêm cung (id[u], id[v], w') vào Gt+1 với id[u] là đỉnh mới của u, đại diện cho chu trình chứa u; w' = w – w(cung đi đến v).
    - Để lưu vết, cần cho cung mới này (của G_{t+1})chỉ vào cung cũ (của G_t)
  - Tăng t và lặp lại
- Pha giãn
  - H_t là cây khung của đồ thị G_t.
  - Mở các nút trong cây H_t để thu được cây H_{t-1}.
    - Thực chất quá trình này là điều chỉnh lại các cung của H_{t-1}.
    - Với mỗi cung của H_t:
      - Thêm nó vào H_{t-1} và xoá bớt 1 cung tương ứng trong chu trình.
      - Điều chỉnh lại trọng số cho cung vừa thêm = trọng số cũ + trọng số cung bị xoá.
  - Lặp lại quá trình này cho đến khi thu được H_0.

999.3 Cài đặt giải thuật Chu-Liu/Edmonds:

999.3.1 Cấu trúc dữ liệu

Biểu diễn G:

Ta định nghĩa cấu trúc dữ liệu Graph để lưu trữ các đồ thị G. Ta sử dụng cách biểu diễn danh sách cung để biểu diễn đồ thị.

```
#define MAXN 100
#define MAXM 500
#define INF 9999999

typedef struct {
  int u, v; //đỉnh đầu, đỉnh cuối
  int w;    //trọng số
  int link; //chỉ đến cung trước đó trong đồ thị G_{t-1}
} Edge;

typedef struct {
  int n, m;
  Edge edges[MAXM];
} Graph;
```

Để có thể truy vết trong pha giãn, với mỗi cung của G_t, ta cần biết nó trước đây tương ứng với cung nào trong đồ thị G_{t-1}. Điều này có thể dễ dàng bằng cách lưu lại chỉ số cung (link) tương ứng của cung này trong đồ thị G_{t-1}.

Biểu diễn H:

Do đồ thị xấp xỉ H là 1 đồ thị đặc biệt: mỗi đỉnh chỉ cần lưu 1 cung đi đến nó (hay đỉnh cha của nó), nên H tương đương với 1 cây. Ta cần 1 CTDL riêng để lưu trữ các H. Với mỗi đỉnh ta cần lưu trữ: đỉnh cha, trọng số cung đi đến và link chỉ vào cung trước đó của cung này trong đồ thị G_{t-1}. Cũng vì lý do truy vết, ta cũng lưu luôn ***link***.

```
typedef struct {
  int n;
  int parent[MAXN]; //đỉnh cha của u
  int weight[MAXN]; //trọng số của cung đi đến u
  int link[MAXN];   //chỉ đến cung trước đó trong G_{t-1}
} Tree;
```

999.3.2 Các bước chính của giải thuật

a. Khởi tạo H và T:

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

void init_tree(Tree *T, int n) {
  T->n = n;
  int i;
  for (i = 1; i <= n; i++) {
    T->parent[i] = -1;
    T->weight[i] = INF;
    T->link[i]   = -1;
  }
}

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

b. Xây dựng đồ thị xấp xỉ Ht từ Gt

Với mỗi cung (u, v) có trọng số w, ta so sánh với trọng số của cung đến v (weight[v]) để xem có cập nhật được không. Khởi tạo tất cả weight[v] = ∞. Cần phải loại bỏ cha của root (nếu có) để tránh các hiệu ứng lề.

```
void buildH(Graph* G, int root, Tree* H) {
   init_tree(H, G->n); //khởi tạo cây rỗng
   int e;
   for (e = 0; e < G->m; e++) {
      int u = G->edges[e].u;
      int v = G->edges[e].v;
      int w = G->edges[e].w;
      int link = G->edges[e].link;
      if (w < H->weight[v]) {
        H->parent[v] = u;
        H->weight[v] = w;
        H->link[v] = link; //chỉ đến cung của Gt-1
      }
    }
    H->parent[root] = -1; //loại bỏ cha của root
    H->weight[root] = 0;  //(nếu có)
}
```

c. *Kiểm tra chu trình trong H*

Do mỗi đỉnh trong H có nhiều nhất là 1 đỉnh cha, nên ta lần theo cha của các đỉnh kiểm tra xem H có chu trình hay không. Nếu có chu trình, ta gán tất cả các đỉnh trong chu trình này một tên mới (sử dụng để xây dựng G_{t+1}).

Từ 1 đỉnh i, nếu đi 1 vòng mà quay lại nó (color[u] = i) thì ta tìm được chu trình. Nếu không, ta sẽ gặp gốc (u = root).

Để tránh xử lý 1 đỉnh đã nằm trong 1 trình ta thêm điều kiện (id[u] == -1) trong vòng lặp while.

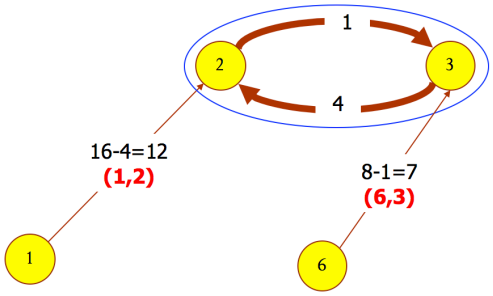
```
//Bổ sung mảng hỗ trợ id lưu tên mới cho các đỉnh
int id[MAXN];

int find_cycles(Tree* H, int root) {
  int i, u, no = 0;
  int color[MAXN];
  //Khởi tạo id, color
  for (i = 1; i <= H->n; i++) {
    id[i] = -1;
    color[i] = -1;
  }
  //Duyệt qua từng đỉnh, và lần theo parent của nó
  for (i = 1; i <= H->n; i++) {
    int u = i;
    while (u != root && id[u] == -1 && color[u] != i) {
      color[u] = i;
      u = H->parent[u];
    }
    //Nếu gặp lại i => tạo chu trình
    if (color[u] == i) {
      no++;
      int v = H->parent[u];
      while (v != u) {
        id[v] = no; //gán id mới cho v
        v = H->parent[v];
      }
      id[u] = no; //u cũng là 1 đỉnh trong chu trình
    }
  }
  return no; //trả về số chu trình tìm được
```

d. *Co đồ thị G_t thành G_{t+1}*

Giả sử sau quá trình kiểm tra chu trình đã có tên mới cho từng đỉnh trong G_t. Nếu u là 1 đỉnh trong G_t thì id[u] là tên mới của u trong G_{t+1}.

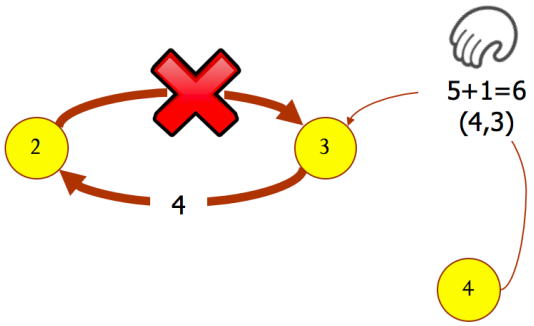
Với mỗi cung e: (u, v, w) trong G_t ta sẽ kiểm tra xem id[u] == id[v]. Nếu khác nhau ta thêm cung (id[u], id[v], w – Ht->weight[v]) vào G_{t+1} và link cung này vào cung e của G_t.




```
void contract(Graph* G, Tree* H, int no, Graph* G1) {
  init_graph(G1, no);
  int e;
  for (e = 0; e < G->m; e++) {
    int u = G->edges[e].u;
    int v = G->edges[e].v;
    int w = G->edges[e].w;
    if (id[u] != id[v])
      add_edge(G1, id[u], id[v], w - H->weight[v], e);
  }
}
```

e. Giãn H_t thành H_{t-1}

Thực chất quá trình giãn H_t thành H_{t-1} thêm các cung từ H_t vào H_{t-1} và xoá bớt 1 cung tương ứng trong chu trình của H_{t-1}. Trong ví dụ bên dưới, ta thêm cung (4, 3) vào H_{t-1} và xoá bỏ cung (2, 3) đi. Điều này tương ứng với việc điều chỉnh đỉnh cha của 3 (trước đây là 2) thành 4 và thay đổi trọng số của cung tương ứng: 5 + 1 = 6 (trọng số cung mới + trọng số cung cũ).



```
void expand(Tree* H, Graph* G1, Tree* H1) {
  int i;
  for (i = 1; i <= H->n; i++)
    if (H->parent[i] != -1) {
      //Lấy cung tương ứng trong G_{t-1}
      Edge pe = G1->edges[H->link[i]];
      //Đổi cha của pe.v thành pe.u
      H1->parent[pe.v] = pe.u;
      H1->weight[pe.v] += H->weight[i];
      H1->link[pe.v]   = pe.link;
    }
}
```

f. Giải thuật hoàn chỉnh

Giờ đây, ta đã có đủ các khối cần thiết cho giải thuật. Chỉ cần lắp ráp lại là có một giải thuật hoàn chỉnh.

```
#define MAXIT 10
void ChuLiu(Graph* G0, int s, Tree* T) {
  Graph G[MAXIT];
  Tree H[MAXIT];
  int i, e;

  int t = 0;
  int root = s;
  G[0] = *G0;

  //Pha co
  while (1) {
    //Xây dựng đồ thị xấp thị
    buildH(&G[t], root, &H[t]);
    int no = find_cycles(&H[t], root);
    if (no == 0) break;

    //Đặt tên mới cho các đỉnh không nằm trong CT
    for (i = 1; i <= H[t].n; i++) {
      if (id[i] == -1)
      id[i] = ++no;
    }
    //Co
    contract(&G[t], &H[t], no, &G[t+1]);
    root = id[root]; //gốc mới
    t++;
  }
  //Pha giãn
  int k;
  for (k = t; k > 0; k--)
    expand(&H[k], &G[k-1], &H[k-1]);

  //Kết quả là H[0]
  *T = H[0];
}
```

g. Chương trình chính

Bổ sung hàm phần đọc dữ liệu và in kết quả. Thế là xong.

```
int main() {
  Graph G;
  int n, m, i, e, u, v, w;

  scanf("%d%d", &n ,&m);
  init_graph(&G, n);
  for (e = 0; e < m; e++) {
    scanf("%d%d%d", &u, &v, &w);
    add_edge(&G, u, v, w, -1);
  }

  Tree T;
  ChuLiu(&G, 1, &T);

  for (i = 1; i <= T.n; i++)
    if (T.parent[i] != -1)
      printf("(%d, %d) %d\n", T.parent[i], i,
                              T.weight[i]);

  return 0;
}
```

999.4 Bài tập

Viết chương trình đọc đồ thị và kiểm tra xem nó có liên thông mạnh hay không. Nếu có in ra "Yes", ngược lại in ra "No".

Gợi ý: *sau khi duyệt toàn bộ đồ thị, nếu min_index của các đỉnh đều giống nhau (=1) thì đồ thị liên thông mạnh.*