Thực hành Buổi 4

Xếp hạng đồ thị và bài toán tổ chức thi công

Mục đích:

- Cài đặt giải thuật xếp hạng đồ thị
- Sắp xếp topo (topo sort) : dựa trên giải thuật xếp hạng
- Cài đặt giải bài toán thi công

1. Bài toán xếp hạng đồ thị

Cho đồ thị có hướng không chu trình (DAG) G = <V, E>. G có 1 đỉnh s có bậc vào = 0 gọi là gốc của đồ thị.

Ta định nghĩa hạng (rank) của một đỉnh u là chiều dài (số cung) của đường đi dài nhất từ s đến u, ký hiệu rank[u].

Rõ ràng rank[s] = 0.

2. Giải thuật xếp hạng đồ thị

Cho đồ thị có hướng không chu trình G, ta cần phải xác định hạng (rank) của các đỉnh của G.

Ý tưởng:

- Đỉnh có bậc vào = 0 (gốc cũ) sẽ có hạng = 0
- Gỡ bỏ các cung nối các đỉnh hạng 0 với các đỉnh kề của nó
- Sau khi gỡ bỏ các cung, các đỉnh có bậc vào = 0 (gốc mới) sẽ có hạng = 1
- Gỡ bỏ các cung nối các đỉnh có hạng bằng 1 với các đỉnh kề của nó
- …

Các biến hỗ trợ

- d[u]: bậc vào của đỉnh u, mỗi khi gỡ bỏ cung nối đến u, ta giảm d[u] đi 1.
- rank[u]: lưu hạng của đỉnh u
- S1: danh sách lưu các đỉnh đang xác định hạng (gốc cũ)
- S2: lưu các đỉnh sắp sửa xem xét (có d[u] == 0, gốc mới)

Giải thuật:

- Tính d[u] cho tất cả các đỉnh
- Đưa các đỉnh có d[i] = 0 vào S1
- k = 1
- Lặp (cho đến khi S1 rỗng)

```
for (các đỉnh u trong S1) {
        rank[u] = k
        for (các đỉnh kề v của u) {
                d[v]--;
                if (d[v] == 0)
                        Đưa v vào S2
        }
}
k ++;
copy S2 vào S1 (gán S1 = S2)
```

Cài đặt giải thuật

Sử dụng phương pháp ma trận kề để biểu diễn đồ thị

```
#define MAX_VERTICES 100
typedef struct {
    int n;
    int A[MAX_VERTICES][MAX_VERTICES];
} Graph;
```

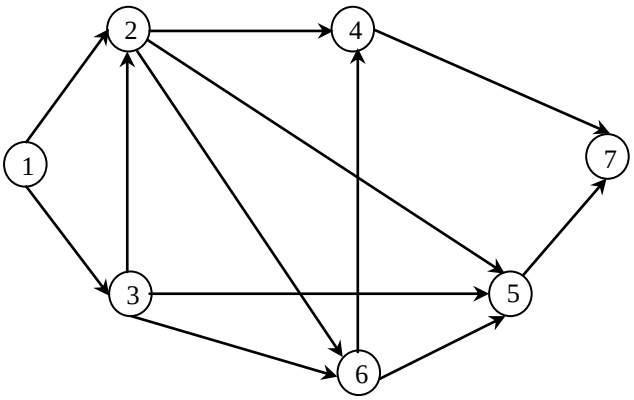
Giải thuật xếp hạng:

```
int rank[MAX_VERTICES];
void ranking(Graph* G) {
    int d[MAX_VERTICES];
    int x, u;
    for (u = 1; u <= G->n; u++)
        d[u] = 0;
    for (x = 1; x <= G->n; x++)
        for (u = 1; u <= G->n; u++)
            if (G->A[x][u] != 0)
                d[u]++;
    List S1, S2;
    make_null_list(&S1);

    for (u = 1; u <= G->n; u++)
        if (d[u] == 0)
            push_back(&S1, u);

    int k = 1, i;
    while (S1.size > 0) {
        make_null_list(&S2);
        for (i = 1; i <= S1.size; i++) {
            int u = element_at(&S1, i);
            rank[u] = k;
            int v;
            for (v = 1; v <= G->n; v++)
                if (G->A[u][v] != 0) {
                    d[v]--;
                    if (d[v] == 0)
                        push_back(&S2, v);
                }
        }
        copy_list(&S1, &S2); //S1 = S2
        k++;
    }
}
```

Đề giải thuật này chạy được, bạn cần bổ sung cấu trúc dữ liệu List (xem các bài trong buổi thực hành trước) với các phép toán:

- Hàm make_null_list(List* L): khởi tạo danh sách rỗng
- Trường size: lưu số phần tử trong danh sách
- Hàm element_at(List* L, int i): trả về phần tử thứ i trong danh sách (thứ tự tính từ 1 đến size).
- Hàm copy(List* S1, List* S2): tạo rỗng S1 và copy các phần tử của S2 vào S1

Bài tập 1. Viết chương trình đọc một đồ thị từ tập tin và in ra hạng của từng đỉnh.




```
7 12
1 2
1 3
…
```

3. Sắp xếp topo

Sắp xếp topo là sắp xếp các đỉnh của đồ thị theo thứ tự sao cho nếu G chứa cung (u, v) thì đỉnh u nằm trước đỉnh v.

Trong đồ thị trên, một thứ tự topo có thể là: 1 3 2 6 4 5 7. Một thứ tự khác cũng hợp lệ là 1 3 2 5 4 7.

Ta có thể cải tiến giải thuật xếp hạng một chút để có được danh sách các đỉnh được sắp xếp.

Ý tưởng:

- Đỉnh có bậc vào = 0 (không có đỉnh nào trước nó) sẽ đứng trước danh sách
- Gỡ bỏ các cung nối các đỉnh có bậc vào = 0 với các đỉnh kề của nó. Một số đỉnh kề này sẽ có bậc vào = 0, ta sẽ thêm nó vào danh sách
- Và cứ tiếp tục như thế

Các biến hỗ trợ

- d[u]: lưu bậc vào của đỉnh i, mỗi khi gỡ bỏ cung nối đến u, ta giảm d[u] đi 1.
- Q: hàng đợi lưu các các đỉnh sẽ xét
- L: danh sách các đỉnh được sắp xếp

Giải thuật: **topo_sort(Graph* G, List* L)**

- Tính d[u]
- Thêm các đỉnh có d[u] = 0 vào Q
- Tạo danh sách L rỗng
- Lặp (cho đến khi Q rỗng)
    - Lấy đỉnh đầu tiên trong Q ra => gọi nó là đỉnh u
    - Đưa u vào cuối danh sách L
    - for (các đỉnh kề v của u) {
        - d[v]--;
        - if (d[v] == 0)
            - Đưa v vào Q;
    - }

Để chạy được giải thuật này, bạn cần cài đặt cấu trúc dữ liệu danh sách và cấu trúc dữ hàng đợi (xem các bài thực hành trước).

Bài tập 2. Viết chương trình đọc đồ thị từ tập tin, sắp xếp các đỉnh theo thứ tự topo và in các đỉnh ra màn hình theo thứ tự này.

4. Bài toán tổ chức thi công

Lý thuyết: xem bài học trên lớp

Vi dụ: Có một dự án xây nhà với 10 công việc như sau:

Công việc	Nội dung công việc	Thời gian thực hiện d(i) tính theo tuần	Công việc trước đó
A	Các công việc nề	7	∅
B	Dựng khung cho mái	3	A
C	Lợp mái	1	B
D	Lắp đặt hệ thống vệ sinh, chiếu sáng	8	A
E	Trang trí mặt tiền	2	C,D
F	Ráp cửa sổ	1	C,D
G	Trang hoàng vườn	1	C,D
H	Làm trần	2	F
J	Sơn phết	2	H
K	Chuyển nhà	1	E,G,J

Ta cần:

- Xác định thời điểm sớm nhất để hoàn thành dự án
- Xác định thời điểm sớm nhất và trể nhất để bắt đầu công việc mà không ảnh hưởng đến tiến độ của dự án

Biểu diễn đầu vào của bài toán

Để đơn giản trong cài đặt, ta đánh số lại các công việc theo thứ tự 1, 2, 3 thay vì A, B, C và lưu vào tập tin theo định dạng như sau:

A	B	C	D	E	F	G	H	J	K	α	β		
1	2	3	4	5	6	7	8	9	10	11	12		

```
10
7 0
3 1 0
1 2 0
8 1 0
2 3 4 0
1 3 4 0
1 3 4 0
2 6 0
2 8
1 5 7 9 0
```

Dòng đầu tiên là số công việc (10), các dòng tiếp theo mỗi dòng mô tả một công việc bao gồm d[u]: thời gian hoàn thành công việc u và danh sách các công việc trước đó của u. Danh sách được kết thúc bằng số 0. Ví dụ: công việc 1 (công việc A) có d[1] = 7 và danh sách các công việc trước đó rỗng.

Công việc 2 (công việc B) có d[2] = 3 và danh sách công việc trước đó là {1}.

Đọc và xây dựng đồ thị: sử dụng đoạn lệnh bên dưới để đọc dữ liệu đầu vào và xây dựng đồ thị.

```
int d[MAX_VERTICES];
…

int main() {
   Graph G;
   int n, u, x, v, j;
   //1. Đọc đồ thị
   FILE* file = fopen("tenfile", "r");
   fscanf("%d", &n);
   init_graph(&G, n+2); //them 2 dinh gia alpha va beta
   d[n+1] = 0;

   for (u = 1; u <= n; u++) {
      scanf("%d", &d[u]); //thoi gian hoan thanh CV u
      do {
         fscanf("%d", &x);
         if (x > 0) add_edge(&G, x, u);
      } while (x > 0);
   }
   …
}
```

Thêm các cung nối alpha đến các đỉnh có bậc vào bằng 0

```
int main() {
   //1. Đọc đồ thị
   //2. Thêm cung nối alpha với các đỉnh có bậc vào = 0
   for (u = 1; u <= n; u++) {
      int deg_neg = 0;
      for (x = 1; x <= n; x++)
         if (G.A[x][u] > 0)
            deg_neg++;
      if (deg_neg == 0)
         add_edge(&G, n+1, u); //đỉnh alpha là đỉnh n+1
   }
   …
}
```

Thêm các cung nối các đỉnh có bậc ra bằng 0 vào beta

```
int main() {
   //1. Đọc đồ thị
   //2. Thêm cung nối alpha với các đỉnh có bậc vào = 0
   //3. Thêm cung nối các đỉnh có bậc ra = 0 vào beta
   for (u = 1; u <= n; u++) {
      int deg_pos = 0;
      for (v = 1; v <= n; v++)
         if (G.A[u][v] > 0)
            deg_pos++;
      if (deg_pos == 0)
         add_edge(&G, u, n+2); //đỉnh beta là đỉnh n+2
   }
   …
}
```

Ta cũng có thể tính bậc vào và bậc ra của các đỉnh trong quá trình đọc danh sách công việc và xây dựng đồ thị (xem như bài tập).

Các bước còn lại:

- Xếp thứ tự topo
- Tính t[u]
- Tính T[u]

```
int main() {
   //1. Đọc đồ thị
   //2. Thêm cung nối alpha với các đỉnh có bậc vào = 0
   //3. Thêm cung nối các đỉnh có bậc ra = 0 vào beta
   //4. Xếp thứ tự topo và lưu vào trong L
   topo_sort(&G, &L)

   //5. Tính t[u]
   int t[MAX_VERTICES];
   t[n+1] = 0; //t[alpha] = 0
   for (j = 2; j <= L.size; j++) {
      int u = element_at(&L, j);
      t[u] = +oo;// vô cùng, ví dụ: 999999
      for (x = 1; x <= G.n; x++)
         if (G.A[x][u] > 0)
            t[u] = min(t[u], t[x] + d[x]);
   }
   //6. tính T[u]
   T[n+2] = t[n+2];
   for (j = L.size - 1; j >= 1; j--) {
      int u = element_at(&L, j);
      t[u] = -1;
      for (v = 1; v <= G.n; v++)
         if (G.A[u][v] > 0)
            T[u] = max(T[u], T[v] - d[u]);
   }
   //7. In kết quả: in t[u] và T[u] ra màn hình


   return 0;
}
```

Bài tập 3. Tổng hợp các đoạn chương trình trên, viết chương trình đọc bài toán thi công được biểu diễn trong tập tin như trong phần 4 và in ra thời điểm sớm nhất và trễ nhất để bắt đầu của các công việc.