BUỔI 4. THỨ TỰ TOPO & ỨNG DỤNG

Mục đích

- *Thực hành cài đặt các thuật toán: sắp xếp topo, xếp hạng các đỉnh, quản lý dự án*
- *Củng cố lý thuyết, rèn luyện kỹ năng lập trình*

Nội dung

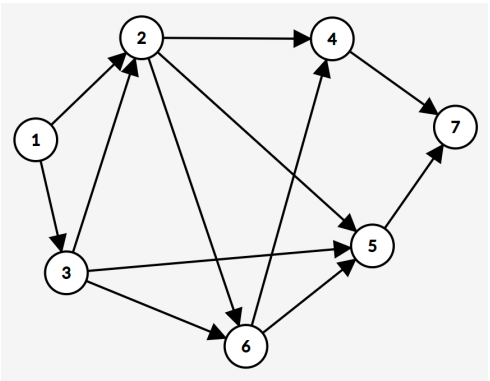
- Sắp xếp topo (topo sort) duyệt theo chiều rộng
- Cài đặt thuật toán xếp hạng đồ thị
- Cài đặt bài toán quản lý dự án

Yêu cầu

- Biểu diễn đồ thị
- Các phép toán cơ bản trên đồ thị
- Duyệt đồ thị (theo chiều rộng)
- Biết sử dụng cấu trúc dữ liệu ngăn xếp + hàng đợi

4.1 Sắp xếp topo

Xét đồ thị `G` là một đồ thị có hướng không chứa chu trình (Directected Acyclic Graph – DAG). Sắp xếp các đỉnh của `G` theo *thứ tự topo* (topological order) hay còn gọi là *sắp xếp topo* (topological sorting) là sắp xếp các đỉnh của đồ thị `G` 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.

4.1.1 Thuật toán sắp xếp topo

Có hai tiếp cận để cài đặt thuật toán sắp xếp topo: duyệt theo chiều sâu và duyệt theo chiều rộng. Trong tài liệu này ta sử dụng phương pháp duyệt theo chiều rộng.

Ý tưởng

- Đỉnh có bậc vào = 0 (không có đỉnh nào trước nó) sẽ đứng đầu 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 u, 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.

Thuật toán: topo_sort(Graph *pG, List *pL)

- Tính d[u] cho tất cả các đỉnh
- Làm hàng đợi Q rỗng
- Thêm các đỉnh có d[u] = 0 vào Q
- Tạo danh sách L rỗng
- `while` (Q không 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;
    }
```

Cài đặt thuật toán

Biểu diễn đồ thị: sử dụng phương pháp ma trận kề

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

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

`List` (danh sách chứa các số nguyên) gồm các hàm và trường sau:

- `make_null_list(List *pL)`: làm rỗng danh sách
- `push_back(List *pL, int x)`: thêm phần tử `x` vào cuối danh sách
- `size`: lưu số phần tử trong danh sách
- `element_at(List *pL, int i)`: trả về phần tử thứ `i` trong danh sách (thứ tự tính từ 1 đến `size`)

`Queue` (hàng đợi chứa các số nguyên) gồm các hàm

- `make_null_queue(Queue *pQ)`: làm rỗng hàng đợi
- `enqueue(Queue *pQ, int x)`: thêm phần tử `x` vào cuối hàng đợi
- `dequeue(Queue *pQ)`: xóa phần tử đầu hàng đợi
- `front(Queue *pQ)`: trả về phần tử đầu hàng đợi
- `empty_queue(Queue *pG)`: trả về 1 nếu hàng đợi rỗng, ngược lại trả về 0

Thuật toán xắp xếp topo

```
void topo_sort(Graph *pG, List *pL) {
	int d[MAX_N];

	//Tính bậc vào của u
	for (int u = 1; u <= pG->n; u++) {
		d[u] = 0;
		for (int x = 1; x <= pG->n; x++)
			if (pG->A[x][u] != 0)
				d[u]++;
	}

	Queue Q;

	//Làm rỗng hàng đợi Q
	make_null_queue(&Q);

	//Đưa các đỉnh có d[u] = 0 vào hàng đợi
	for (int u = 1; u <= pG->n; u++)
		if (d[u] == 0)
			enqueue(&Q, u);

	//Làm rỗng danh sách pL
	make_null_list(pL);

	//Vòng lặp chính, lặp đến khi Q rỗng thì dừng
	while (!empty_queue(&Q)) {
		int u = front(&Q);
		dequeue(&Q);
		push_back(pL, u);

		//Xoá đỉnh u <=> giảm bậc vào của các đỉnh kề v của u
		for (int v = 1; v <= pG->n; v++)
			if (pG->A[u][v] != 0) {
				d[v]--;
				if (d[v] == 0)
					enqueue(&Q, v);
			}
	}
}
```

Bài tập 1. Viết chương trình đọc một *đồ thị có hướng không chu trình* G, áp dụng thuật toán sắp xếp theo phương pháp duyệt theo chiều rộng để xắp xếp các đỉnh của G. In các đỉnh ra màn hình theo thứ tự topo. In các đỉnh trên một dòng, cách nhau bằng 1 khoảng trắng.

4.1.2 Ứng dụng sắp xếp topo

Xếp đá

Peter rất thích chơi đá. Anh ta thường dùng đá để trang trí sân nhà của mình. Hiện tại, Peter có n hòn đá. Dĩ nhiên mỗi hòn đá có một khối lượng nào đó. Peter muốn đặt các hòn đá này dọc theo lối đi từ cổng vào nhà của mình. Peter lại muốn sắp xếp như thế này: hòn đá nặng nhất sẽ đặt ở cạnh cổng rào, kế tiếp là hòn đá nặng thứ 2, ... hòn đá nhẹ nhất sẽ được đặt cạnh nhà. Như vậy nếu đi từ trong nhà ra cổng, ta sẽ gặp các hòn đá có khối lượng tăng dần.

Tuy nhiên, điều khó khăn đối với Peter là anh chỉ có một cây cân đĩa mà không có quả cân nào. Nói cách khác, mỗi lần cân Peter chỉ có thể biết được hòn đá nào nhẹ hơn hòn đá nào chứ không biết nó nặng bao nhiêu.

Sau `m` lần cân, Peter biết được sự khác nhau về cân nặng của `m` cặp. Với các thông tin này, hãy giúp Peter sắp xếp các viên đá theo thứ tự anh mong muốn.

Đầu vào (Input)

Dữ liệu đầu vào được nhập từ bàn phím với định dạng:

- Dòng đầu tiên chứa 2 số nguyên n và m, tương ứng là số hòn đó và số lần cân
- m dòng tiếp theo mỗi dòng chứa 2 số nguyên u v nói rằng hòn đá u nhẹ hơn hòn đá v. Giả sử không có cặp (u, v) nào được lặp lại.

Đầu ra (Output)

- In ra màn hình thứ tự của các hòn đá theo khối lượng tăng dần, mỗi số trên một dòng.

Bạn có thể yên tâm là dữ liệu đầu được giả sử rằng chỉ có một kết quả quả duy nhất.

Ví dụ:

Đầu vào	Đầu ra
3 2 1 3 3 2	1 3 2

Trong ví dụ này, ta có: hòn đá 1 nhẹ nhất, kế đến là hòn đá 3 và sau cùng là hòn đá 2.

Để giải bài toán này, ta mô hình hoá về đồ thị có hướng

- Hòn đá ⇔ Đỉnh
- Hòn đá u nhẹ hơn hòn đá v ⇔ cung (u, v). u là đỉnh gốc, v là đỉnh ngọn.

Bài toán sắp xếp thứ tự các hòn đá sao cho hòn đá nhẹ đứng trước trở thành bài toán xếp thứ tự các đỉnh sao cho đỉnh gốc đứng trước đỉnh ngọn. Đây chính là thứ tự topo.

Bài tập 2. Hãy lập trình giải bài toán này và nộp bài lên hệ thống.

4.2 Xếp hạng các đỉnh đồ thị

Cho đồ thị có hướng không chu trình (DAG) G. Do G không có chu trình nên G có ít nhất 1 đỉnh s có bậc vào = 0 gọi là gốc (root) 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. Nếu đồ thị có nhiều gốc thì chọn gốc có đường đi dài nhất.

4.2.1 Thuật toán 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.
- r[u]: lưu hạng của đỉnh u
- S1: danh sách lưu các đỉnh đang xác định hạng (các gốc cũ)
- S2: lưu các đỉnh sắp sửa xem xét (có d[u] == 0, các gốc mới)

Thuật toán

- Tính d[u] cho tất cả các đỉnh
- Đưa các đỉnh u có d[u] = 0 vào S1
- k = 0 (*hoặc 1 tuỳ theo ta muốn gốc có hạng mấy*)
- `while` (S1 không rỗng)

```
Làm rỗng S2
for (các đỉnh u trong S1) {
        r[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)
```

Thuật toán này đã được cải tiến so với thuật toán gốc ở chỗ chỉ dùng 2 danh sách S1 và S2 thay vì phải dùng một mảng các danh sách như trong phần lý thuyết.

Cài đặt thuật toán

Biểu diễn đồ thị: sử dụng phương pháp ma trận kề

Các cấu trúc dữ liệu bổ sung

`List` (danh sách chứa các số nguyên) gồm các hàm và trường sau:

- `make_null_list(List *pL)`: làm rỗng danh sách
- `push_back(List *pL, int x)`: thêm phần tử `x` vào cuối danh sách
- `size`: lưu số phần tử trong danh sách
- `element_at(List *pL, int i)`: trả về phần tử thứ `i` trong danh sách (thứ tự tính từ 1 đến `size`).
- `copy_list(List *pS1, List *pS2)`: copy các phần tử của `pS2` vào `pS1`.

Thuật toán

```
//Lưu hạng của các đỉnh
int r[MAX_N];

//Hàm xếp hạng
void rank (Graph *pG) {
	int d[MAX_N]; //Lưu bậc các đỉnh

	//Tính bậc vào của các đỉnh d[u]
	for (int u = 1; u <= pG->n; u++) {
		d[u] = 0;
		for (int x = 1; x <= pG->n; x++)
			if (pG->A[x][u] != 0)
				d[u]++;
	}

	//Sử dụng 2 danh sách S1, S2
	List S1, S2;

	//Tìm gốc, đưa vào S1
	make_null_list(&S1);
	for (int u = 1; u <= pG->n; u++)
		if (d[u] == 0)
			push_back(&S1, u);


	//Vòng lặp chính, lặp đến khi S1 rỗng thì dừng
	int k = 0; //hạng tính từ 0. Tùy theo bài toán có thể cho k = 1
	while (S1.size > 0) {
		make_null_list(&S2);
		for (int i = 1; i <= S1.size; i++) {
			int u = element_at(&S1, i);   //Lấy các gốc u trong S1 ra
			r[u] = k;                     //Xếp hạng cho u

			//Xoá đỉnh u <=> giảm bậc vào của các đỉnh kề v của u
			for (int v = 1; v <= pG->n; v++)
				if (pG->A[u][v] != 0) {
					d[v]--;
					if (d[v] == 0)
						push_back(&S2, v);
				}
		}
		copy_list(&S1, &S2);    //Copy S2 vào S1
		k++;                    //Tăng hạng kế tiếp cho các gốc mới
	}
}
```

Bài tập 3a. Viết chương trình đọc một đơn đồ thị có hướng không chu trình và in ra hạng của từng đỉnh ra màn hình theo mẫu:

```
r[1] = abc
r[2] = xyz
…
```

Bài tập 3b. Tương tự bài 3a nhưng với đa đồ thị có hướng không chu trình.

Gợi ý: Gom các đa cung thành đơn cung. Có thể xử lý trong hàm `add_edge()` hoặc chỗ tính bậc vào của các đỉnh.

4.2.2 Ứng dụng

Bài tập 4a. Chia kẹo

Cô giáo Trang chuẩn bị kẹo để phát cho các bé mà cô đang giữ. Dĩ nhiên môi bé đều có một tên gọi rất dễ thương ví dụ: Mạnh Phát, Diễm Quỳnh, Đăng Khoa, ... Tuy nhiên, để đơn giản vấn đề ta có thể giả sử các em được đánh số từ 1 đến n.

Cô giáo muốn rằng tất cả các em đều phải có kẹo. Cô lại biết thêm rằng có một số bé có ý muốn hơn bạn mình một chút vì thế các em ấy muốn kẹo của mình nhiều hơn của bạn.

Hãy viết chương trình giúp cô tính xem mỗi em cần được chia ít nhất bao nhiêu kẹo và tổng số kẹo ít nhất mà cô phải chuẩn bị là bao nhiêu.

Đầu vào (Input)

Dữ liệu đầu vào được nhập từ bàn phím với định dạng:

- Dòng đầu tiên chứa 2 số nguyên n và m, tương ứng là số bé và số cặp bé mà trong đó có 1 bé muốn có kẹo hơn bạn mình.
- m dòng tiếp theo mỗi dòng chứa 2 số nguyên a, b nói rằng bé a muốn có kẹo nhiều hơn bé b.

Đầu ra (Output)

- In ra màn hình số kẹo ít nhất của từng em, mỗi em trên một dòng.
- Dòng cuối cùng in tổng số kẹo ít nhất mà cô giáo Trang cần phải chuẩn bị

Ví dụ:

Đầu vào	Đầu ra
7 10 2 1 3 1 4 1 3 2 6 2 7 3 5 4 3 5 7 5 7 6	1 2 4 2 3 3 5 20

Mô hình bài toán về đồ thị có hướng.

- Em bé <=> Đỉnh
- Em bé a muốn có nhiều kẹo hơn em bé b <=> cung (b, a). Chú ý: a là ngọn, b là gốc.

Bài toán trở thành tìm hạng của các đỉnh. Do yêu cầu của đề bài là mỗi em bé đều phải có kẹo nên em bé tương ứng với đỉnh gốc (hạng 0) sẽ có 1 viên kẹo. Em bé tương ứng với hạng 1 sẽ có 2 viên kẹo, … Hay tổng quát:

$$\text{Số kẹo} = \text{hạng} + 1$$

Hãy lập trình giải bài toán này và nộp bài lên hệ thống.

Bài tập 4b. Xếp hạng các đội bóng

Trong một giải đấu bóng đá gồm n đội bóng, đánh số từ 1 đến n. Mỗi trận đấu có hai đội thi đấu với nhau cho đến khi phân biệt thắng thua (ví dụ: hiệp phụ, đá luân lưu). Sau khi giải đấu kết thúc, ban tổ chức muốn xếp hạng các đội theo quy tắc sau:

- Hạng được tính từ 1, 2, 3, ...
- Đội không thua trận nào xếp hạng 1
- Nếu đội A đá thắng đội B thì hạng của đội A nhỏ hơn hạng của đội B.
- Nếu một đội có thể nhận nhiều hạng khác nhau thì chọn hạng nhỏ nhất.

Hoặc bạn cũng có thể sử dụng định nghĩa sau:

$$\mathbf{Hạng(v) = max\ \{Hạng(u)\ \} + 1}$$

với u là đội thắng đội v.

Hãy giúp ban tổ chức viết chương trình xếp hạng cho các đội. Giả sử không xảy ra trường hợp A thắng B, B thắng C, …, Z thắng A.

Đầu vào (Input)

- Dữ liệu đầu vào được nhập từ bàn phím với định dạng:
- Dòng đầu tiên chứa 2 số nguyên n và m, tương ứng là số đội và số trận đấu.
- m dòng tiếp theo mỗi dòng chứa 2 số nguyên **u v** mô tả kết quả trận đấu: **u** thắng, **v** thua.

Đầu ra (Output)

- In ra màn hình hạng của các đội bóng theo số thứ tự của đội trên cùng 1 dòng, mỗi đội cách nhau 1 khoảng trắng.

<Hạng đội 1> <Hạng đội 2> ... <Hạng đội n>

Trong ví dụ bên dưới ta có: hạng của 1 = 1, hạng của 2 = 3 và hạng của 3 = 2.

Đầu vào	Đầu ra
3 2 1 3 3 2	1 3 2

Hãy lập trình giải bài toán này và nộp bài lên hệ thống.

4.3 Bài toán quản lý dự án

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 trễ nhất để bắt đầu công việc mà không ảnh hưởng đến tiến độ của dự án

4.3.1 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 0
1 5 7 9 0
```

Dòng đầu tiên là số công việc (10 công việc), 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 phải làm trước 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 nó rỗng.
- Công việc 2 (công việc B) có d[2] = 3 và danh sách công việc trước nó là {1}.
- …

4.3.2 Đọc và xây dựng đồ thị

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

- Đọc số lượng công việc n.
- Khởi tạo đồ thị có (n + 2) đỉnh. Đỉnh α có số thứ tự là n + 1 và đỉnh β có thứ tự là n + 2.
- Đọc từng đỉnh u và tạo cung dựa trên các công việc trước của u.
- Thêm cung từ α vào các đỉnh có bậc vào bằng 0
- Thêm cung từ các đỉnh có bậc ra bằng 0 vào β.

```
//Lưu thời gian hoàn thành công việc
int d[MAX_N];

//Đọc danh sách các công việc và tạo đồ thị
int main() {
	Graph G;
	int n, u, x, v, j;

	//1. Đọc đồ thị
	freopen("tenfile", "r", stdin);
	scanf("%d", &n);

	//1a. Tạo đồ thị có n + 2 đỉnh (alpha = n+1 và beta = n + 2)
	init_graph(&G, n+2);
	int alpha = n + 1, beta = n + 2;
	d[alpha] = 0; //thời gian hoàn thành alpha là 0.

	//1b. Đọc danh sách các công việc
	for (u = 1; u <= n; u++) {
		scanf("%d", &d[u]);	//Thời gian hoàn thành công việc u
		do {
			scanf("%d", &x);	//Đọc công việc trước của u
			if (x > 0)
				add_edge(&G, x, u);
		} while (x > 0);		//Đọc đến khi gặp số 0 thì dừng
	}

	//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++;		//deg_neg là bậc vào của u
		if (deg_neg == 0)
			add_edge(&G, alpha, u);
	}

	//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++;		//deg_neg là bậc ra của u
		if (deg_pos == 0)
			add_edge(&G, u, beta);
	}
	...
}
```

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. Hãy tự do sáng tạo, đừng tự trói buộc vào chương trình mẫu!

4.3.3 Tính toán thời điểm sớm nhất và trễ nhất để bắt đầu công việc

Các bước còn lại trong thuật toán:

- Xếp thứ tự topo
- Tính t[u]: thời điểm sớm nhất
- Tính T[u]: thời điểm trễ nhất

```
//Lưu thời gian hoàn thành công việc
int d[MAX_N];

//Đọc danh sách các công việc và tạo đồ thị
int main() {
	Graph G;
	int n, u, x, v, j;

	//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 các đỉnh theo giải thuật sắp xếp topo.
	List L;
	topo_sort(&G, &L);

	//5. Tính t[u]
	int t[MAX_N];
	t[alpha] = 0;

	//alpha chắc chắn nằm đầu danh sách, các đỉnh còn lại đi từ 2 đến L.size
	for (j = 2; j <= L.size; j++) {
		int u = element_at(&L, j);
		t[u] = -oo; //vô cùng bé, ví dụ: -999999
		for (x = 1; x <= G.n; x++)
			if (G.A[x][u] > 0)
				t[u] = max(t[u], t[x] + d[x]);
	}

	//6. tính T[u]
	int T[MAX_N];
	T[beta] = t[beta];

	//beta chắc chắn nằm cuối danh sách, đi ngược lại từ L.size - 1 về 1
	for (j = L.size - 1; j >= 1; j--) {
		int u = element_at(&L, j);
		T[u] = +oo; //vô cùng lớn, ví dụ: 999999
		for (v = 1; v <= G.n; v++)
			if (G.A[u][v] > 0)
				T[u] = min(T[u], T[v] - d[u]);
	}

	//7. In kết quả: in t[u] và T[u] ra màn hình
	...

	return 0;
}
```

Bài tập 5. Tổng hợp các đoạn chương trình trên, viết chương trình đọc bài toán thi công được biểu diễn trong tập tin như trong phần 4.3.1 và in ra thời điểm sớm nhất và trễ nhất để bắt đầu của các công việc.

Bài tập 6. Ước lượng thời gian hoàn thành dự án phần mềm

Khôi là người quản lý dự án của công ty phần mềm Grafity Ltd. Nhiệm vụ của anh là nhận dự án, phân tích và chia phân công các lập trình viên thực hiện nó. Khi nhận được dự án, việc đầu tiên của Khôi là phân chia dự án thành các công việc và ước lượng thời gian hoàn thành từng công việc. Ngoài ra, tùy theo dự án, các công việc này có thể phụ thuộc nhau ví dụ: công việc A cần phải làm xong rồi thì mới có thể thực hiện công việc B.

Sau khi có được danh sách các công việc, thời gian hoàn thành các công việc và sự phụ thuộc giữa chúng. Khôi sẽ ước lượng được cần ít nhất bao nhiêu thời gian để hoàn thành dự án. Làm thủ công khá mất thời gian nên Khôi mới nhờ đến bạn. Hãy lập trình để giúp anh ấy.

Đầu vào (Input)

Dữ liệu đầu vào được nhập từ bàn phím với định dạng:

- Dòng đầu tiên chứa 1 số nguyên n là số công việc.
- Dòng thứ 2 chứa n số nguyên tương ứng với thời gian hoàn thành của n công việc
- Dòng thứ 3 chứa 1 số nguyên m cho biết số cặp công việc phụ thuộc nhau
- m dòng tiếp theo, mỗi dòng chứa 2 số nguyên A B nói rằng phải hoàn thành công việc A xong thì mới có thể bắt đầu B..

Đầu ra (Output)

- In ra màn hình một số nguyên duy nhất cho biết thời gian ít nhất để hoàn thành dự án.