Trường Đại học Kinh tế - Đại học Huế

Khoa Hệ thống thông tin Kinh tế

Lập trình hướng đối tượng

Bài tập 2 : Ngôn ngữ lập trình C/C++

Hạn cuối nộp bài : ***12h00 thứ bảy, ngày 05 tháng 05 năm 2012***
Nộp bằng cách gởi email về địa chỉ ***lvman@hce.edu.vn***
Hãy đọc ***Quy định nộp báo cáo bài tập***

Giới thiệu

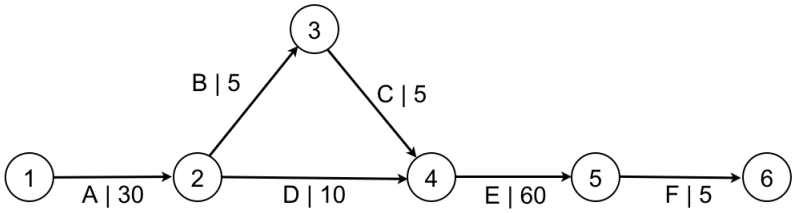
Trong bài tập này, các bạn sẽ vận dụng tất cả các khái niệm đã học của ngôn ngữ lập trình C/C++ để thiết kế, xây dựng một chương trình quản lý dự án sử dụng phương pháp PERT.

Quản lý dự án với phương pháp PERT

Các dự án lớn thông thường bao gồm nhiều công việc khác nhau. Trong mối quan hệ thời gian, những công việc này sẽ có công việc trước và công việc sau, tức là một số công việc phải hoàn thành để khởi động một công việc khác. Khi số lượng công việc cần quản lý lớn, việc quản lý, theo dõi, đánh giá, sửa đổi dự án là rất khó khăn. Do đó, phương pháp PERT (Program Evaluation Research Task) được đề ra năm 1957 với mục đích giúp cho người quản lý trả lời được các câu hỏi như :

- Dự án sẽ hoàn thành khi nào ?
- Mỗi hoạt động của dự án nên bắt đầu vào thời điểm nào và kết thúc vào thời điểm nào ?
- Những hoạt động nào của dự án phải kết thúc đúng thời hạn để tránh cho toàn bộ dự án bị kết thúc chậm hơn so với kế hoạch ?

Nguyên lý của phương pháp PERT là khá đơn giản. Nó sử dụng đồ thị có hướng để sắp xếp có thứ tự tất cả các công việc của dự án. Trong đó, mỗi cạnh có hướng sẽ là một công việc và các công việc được nối với nhau bằng các nút, còn gọi là sự kiện. Công việc A là công việc trước của công việc B thì cạnh A và cạnh B sẽ nối với nhau bằng một nút và cạnh A sẽ có hướng hướng vào nút đó, cạnh B có hướng ra khỏi nút đó.



Hình trên là sơ đồ mạng lưới PERT cho việc chuẩn bị món súp rau củ. Với A là mua rau củ hết 30 phút, B là rửa và gọt vỏ hết 5 phút, C là cắt nhỏ hết 5 phút, D là đun sôi nước muối hết 10 phút, E là luộc rau củ hết 60 phút và F là trộn chung tất cả hết 5 phút.

Phương pháp PERT bao gồm 5 bước sau :

1. **Xác định danh sách công việc**

   Lập bảng danh sách các công việc của dự án và thời gian thực hiện của mỗi công việc.

   Ví dụ : giả sử một dự án xây nhà kho có các công việc sau :

Công việc	**Thời gian (ngày)**
A. Lập kế hoạch	4
B. San lấp mặt bằng	2
C. Mua vật liệu	1
D. Đào móng	1
E. Mua cửa, cửa sổ	2
F. Vận chuyển vật liệu	2
G. Đổ móng	2
H. Vận chuyển cửa, cửa sổ	10
I. Xây tường, lợp mái	4
J. Lắp cửa, cửa sổ	1

2. **Xác định các tiền điều kiện**

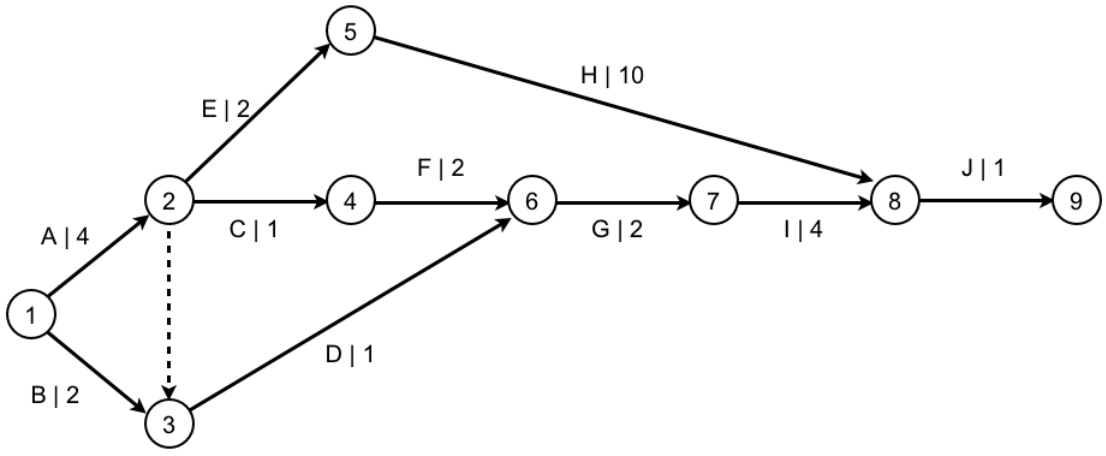
   Xác định công việc nào trước, công việc nào sau với mỗi công việc trong dự án. Từ đó, lập được bảng 3 cột, với cột giữa là công việc, cột bên trái là công việc trước, cột bên phải là công việc sau.

   Với ví dụ xây nhà kho trên, ta sẽ có bảng sau :

Công việc trước	Công việc	Công việc sau
	A	C, D, E
	B	D
A	C	F
A, B	D	G
A	E	H
C	F	G
D, F	G	I
E	H	J
G	I	J
H, I	J	

3. **Vẽ sơ đồ mạng PERT**

Với ví dụ xây nhà kho trên, ta có thể vẽ được sơ đồ mạng PERT như sau :



4. **Tính toán các giá trị thời gian và xác định đường găng**

Cách 1 - tính toán trên công việc

(a) Tính thời gian bắt đầu sớm nhất và thời gian kết thúc sớm nhất (EST - Earliest start time và EFT - Earliest finish time) cho từng hoạt động

Thời gian bắt đầu sớm nhất của một hoạt động *rời một nút nào đó* là thời gian muộn nhất trong các thời gian kết thúc sớm nhất đối của các hoạt động *đi vào nút đó.*

Thời gian kết thúc sớm nhất được tính theo công thức :

EFT = EST + thời gian thực hiện công việc

Thời gian bắt đầu sớm nhất của các công việc không có công việc trước là 0.

(b) Tính thời gian bắt đầu muộn nhất và thời điểm kết thúc muộn nhất (LST - Latest start time và LFT - Latest finish time) cho từng hoạt động

Thời gian kết thúc muộn nhất của một hoạt động *đi vào một nút nào đó* là thời gian sớm nhất trong các thời gian bắt đầu muộn nhất của các hoạt động *rời nút đó*.

Thời gian bắt đầu muộn nhất được tính theo công thức :

LST = LFT - thời gian thực hiện công việc

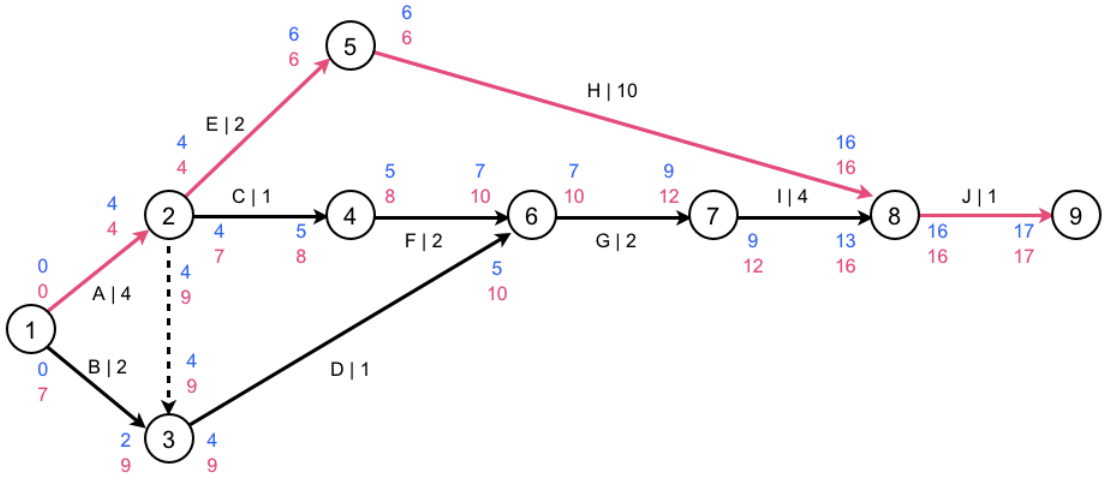
Thời gian kết thúc muộn nhất của công việc không có công việc sau chính là thời gian kết thúc sớm nhất

(c) Xác định công việc găng, đường găng

Công việc găng là công việc mà LST - EST = LFT - EFT = 0, hay nói cách khác là độ trễ cho phép bằng 0.

Một đường đi qua các cung (công việc) mà độ trễ bằng 0 là đường găng.

Với ví dụ xây nhà kho, ta có thể tính được các giá trị thời gian và tìm được đường găng như sau :



Thời gian bắt đầu được ghi ở gốc của cung, còn thời gian kết thúc ghi ở mũi tên của cung. Thời gian sớm có màu xanh dương và ghi bên trên thời gian muộn mùa đỏ. Đường găng là đường có màu đỏ.

Cách 2 - tính toán trên sự kiện Cách tính trên đây là cách tính thời gian trên các công việc. Tồn tại một cách tính đơn giản hơn dựa trên các sự kiện.

Trong cách tính này, người ta sẽ tính thời gian sớm và muộn cho mỗi sự kiện theo công thức sau :

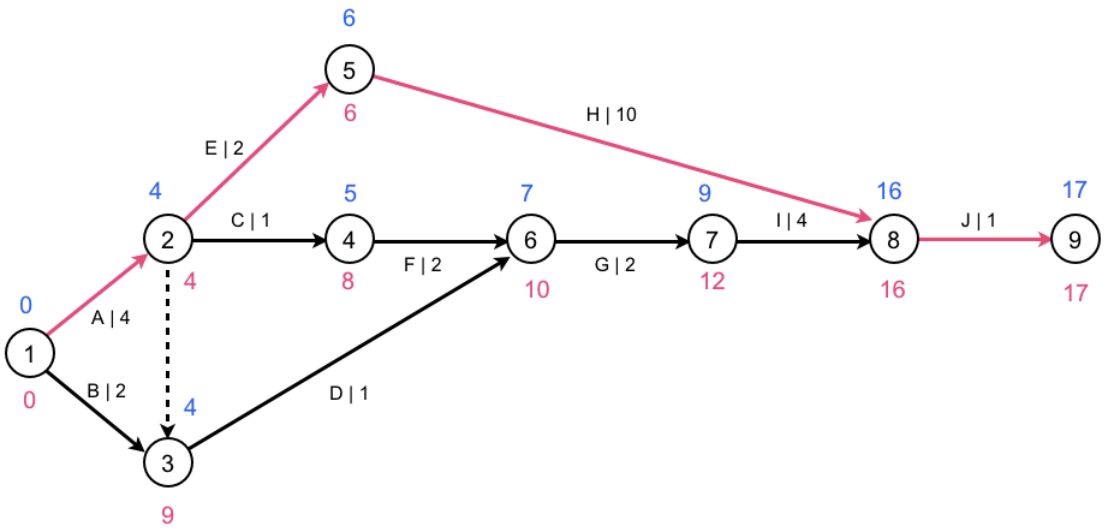
Thời gian sớm = max {Thời gian sớm của sự kiện trước công việc i + Thời gian thực hiện công việc i}, i = 1..n với n là số lượng công việc trước sự kiện cần tính

Thời gian muộn = min {Thời gian muộn của sự kiện sau công việc i - Thời gian thực hiện công việc i}, i = 1..n với n là số lượng công việc sau sự kiện cần tính

Khi đó, công việc găng sẽ được tính theo điều kiện :

Thời gian sớm của sự kiện sau - Thời gian thực hiện công việc - Thời gian sớm của sự kiện trước = 0

Với ví dụ xây dựng nhà kho, ta sẽ có kết quả như sau :



Thời gian sớm màu xanh dương và thời gian muộn màu đỏ.

5. **Tính thời gian dự trữ**

Với cách 1, thời gian dự trữ cho mỗi công việc được tính theo công thức :

Thời gian dự trữ = Thời gian kết thúc muộn nhất - thời gian bắt đầu sớm nhất - thời gian thực hiện công việc

Với cách 2, sẽ là :

Thời gian dự trữ = Thời gian muộn của sự kiện trước - thời gian sớm của sự kiện sau - thời gian thực hiện công việc

Với ví dụ xây nhà kho, ta sẽ có thời gian dữ trữ như sau :

Công việc	Thời gian dự trữ
A	4-0-4=0
B	9-0-2=7
C	8-4-1=3
D	10-4-1=5
E	0
F	10-5-2=3
G	3
H	0
I	3
J	0

Tham khảo :

1. *Toán ứng dụng* (PGS. TS. Nguyễn Hải Thanh), Mục 2.1 : Các khái niệm cơ bản về PERT - Chương 2 : Các mô hình mạng
2. *Slide Toán kinh tế* (PGS. TS. Nguyễn Thống), Chương 7 : Sơ đồ mạng (CPM & PERT)

Yêu cầu của bài tập

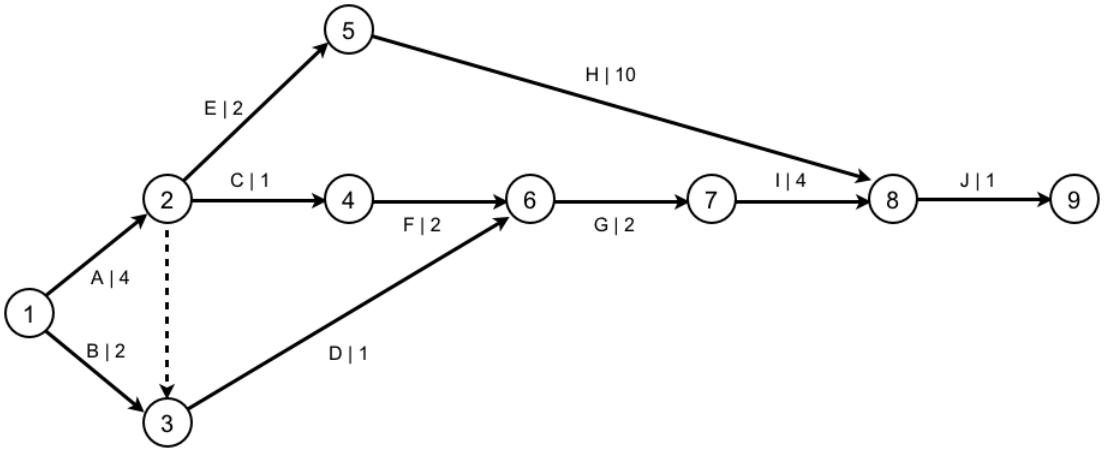
Hãy sử dụng tất cả các kiến thức về ngôn ngữ lập trình C/C++ để lập trình tạo ra một ứng dụng quản lý dự án sử dụng phương pháp PERT. Chương trình này cho phép người sử dụng nhập vào một sơ đồ mạng PERT, rồi tính toán và in ra màn hình những giá trị thời gian, đường găng và thời gian dự trữ tương ứng với sơ đồ mạng PERT đó. Các yêu cầu cụ thể của chương trình như sau :

1. Xây dựng cấu trúc dữ liệu để lưu trữ sơ đồ mạng PERT
2. Xây dựng cấu trúc dữ liệu để dùng trong tính toán các giá trị thời gian
3. Cho phép nhập thông tin sơ đồ mạng PERT từ bàn phím
4. Hỗ trợ đối số dòng lệnh
5. Tính thời gian sớm
6. Tính thời gian muộn
7. Tìm tất cả các đường găng (nếu có)
8. Tính thời gian dự trữ
9. In kết quả ra màn hình
10. (Tùy chọn - cộng điểm) Cho phép nhập thông tin sơ đồ mạng PERT từ file văn bản

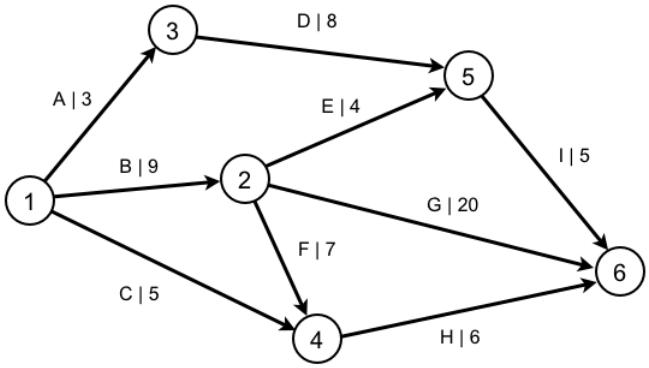
Test

Các bạn dùng 4 mạng PERT sau để test ứng dụng và copy màn hình kết quả vào báo cáo.

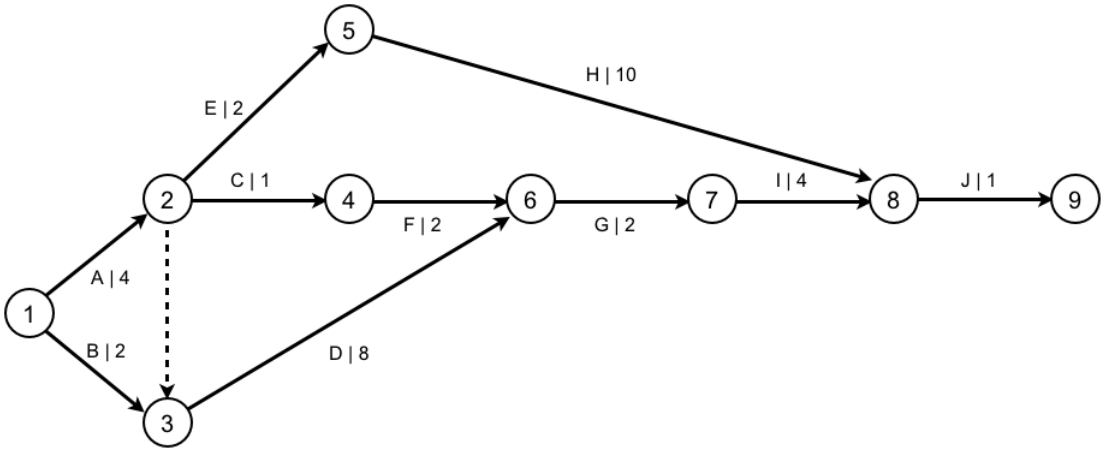
Pert1



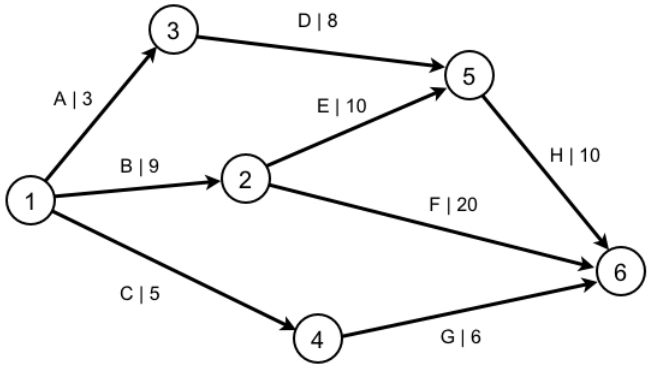
Pert2



Pert3



Pert4



Chú ý *: Trong hai mạng Pert1 và Pert3 có một công việc được thể hiện bằng đường gạch đứt nét. Khi nhập vào bạn hãy nhập tên của nó là K nhưng thời gian công việc là 0. Đây là công việc ảo.*

Nếu bạn làm chức năng 10 - nhập thông tin sơ đồ mạng PERT từ file văn bản - thì lấy 4 file pert1.txt, pert2.txt, pert3.txt, pert4.txt đính kèm theo bài thực hành để làm file đầu vào.

Hướng dẫn lập trình

Những thuật toán, cấu trúc dữ liệu và cách xử lý được đưa ra trong các phần sau đây chỉ là đề nghị. Các bạn có thể đưa ra thuật toán, cấu trúc dữ liệu và cách xử lý riêng và hãy chỉ ra trong báo cáo của bạn. Thuật toán, cấu trúc dữ liệu và cách xử lý của bạn là tốt hơn sẽ được đánh giá cao hơn và ngược lại, điểm của bạn sẽ thấp nếu chương trình của bạn hỗ trợ ít chức năng hơn hoặc cách xử lý ít phức tạp hơn.

Trong mỗi chức năng các bạn nên xây dựng thành các hàm chức năng nhỏ hơn. Như vậy, chương trình sẽ không bị rối và dễ gỡ lỗi. Đồng thời các chức năng khác có thể sử dụng lại các hàm đó.

Xử lý đối số dòng lệnh

Với chương trình này, các bạn có thể hỗ trợ 4 dạng đối số dòng lệnh sau :

```
1 pert.exe
2 pert.exe <num>
3 pert.exe <path>
4 pert.exe -h
```

Với `pert.exe` là tên chương trình.

Dạng thứ nhất chỉ gọi chương trình mà không đưa vào đối số nào cả. Như vậy, trong chương trình ta sẽ cho phép người sử dụng nhập vào các công việc cho đến khi nào người sử dụng không muốn nhập nữa thì thôi.

Dạng thứ hai gọi chương trình kèm với một số (`<num>`). Số này thể hiện số công việc. Như vậy, chương trình sẽ chỉ cho người sử dụng nhận số công việc cụ thể đó thôi.

Dạng thứ ba là gọi chương trình kèm theo đường dẫn (`<path>`) đến tập tin văn bản chứa mạng PERT. (Đọc thêm phần Nhập thông tin sơ đồ mạng PERT từ file văn bản bên dưới)

Dạng thứ tư kèm theo đối số `-h` để yêu cầu in ra hướng dẫn sử dụng của chương trình.

Thuật toán của phần xử lý đối số dòng lệnh như sau :

```
Kiểm tra số lượng đối số dòng lệnh
    Nếu lớn hơn 2 thì thông báo lỗi và in ra hướng dẫn sử dụng chương trình
    Nếu chỉ có 1 đối số thì chuyển đến việc nhập dữ liệu kiểu 1
Kiểm tra đối số thứ hai
    Nếu là -h (so sánh chuỗi ký tự trong đối số thứ hai với chuỗi ‘‘-h’’)
    thì in ra hướng dẫn sử dụng
    Nếu thuộc dạng 2 (dùng hàm kiểm tra xem đối số thứ hai có phải là số
    hay không) thì chuyển đến việc nhập dữ liệu kiểu 2
    Nếu thuộc dạng 3 thì chuyển đến việc nhập dữ liệu từ file văn bản
```

Như vậy, bạn nên viết 4 hàm để xử lý cho 4 chức năng :

- Hàm in ra hướng dẫn sử dụng - tức là in ra các khuôn dạng lời gọi chương trình với các đối số
- Nhập dữ liệu kiểu 1
- Nhập dữ liệu kiểu 2

  Xem phần sau để biết về nhập dữ liệu hai kiểu này
- Nhập dữ liệu từ file văn bản

  Xem phần nhập dữ liệu từ file văn bản bên dưới để biết chi tiết

Chú ý *: Nếu bạn không làm chức năng thứ 10 thì có thể bỏ qua việc xử lý dạng thứ 3 của đối số dòng lệnh*

Nhập thông tin sơ đồ mạng PERT từ bàn phím

Việc nhập dữ liệu nên được tách ra làm hai ra phần :

1. Nhập danh sách công việc (không quan tâm đến công việc nào trước, công việc nào sau)
2. Nhập tiền điều kiện (chính là xác định công việc nào trước và công việc nào sau)

Nhập danh sách công việc kiểu 1 Với kiểu này, số lượng công việc là không được biết trước, nên sau khi người sử dụng nhập xong một công việc thì chương trình sẽ hỏi xem người sử dụng có muốn nhập tiếp hay không. Tùy vào câu trả lời của người sử dụng (có thể mã hóa thành số 1-yes và số 0-no, người sử dụng nhập vào số 1 hoặc 0) mà chương trình cho phép người sử dụng tiếp tục nhập hoặc thoát ra khỏi tiến trình nhập này.

Hoặc một cách khác, các bạn có thể dừng việc nhập khi người sử dụng nhập vào ký tự kết thúc file (end of file - `Ctrl+Z`)

Nhập danh sách công việc kiểu 2 Số lượng công việc là đã được xác định trước tại đối số dòng lệnh, nên ta chỉ cần lặp lại việc nhập thông tin công việc với số lần đã cho đó.

Chú ý : Thông thường, trong quản lý người ta thường gán cho mỗi công việc một số id. Với nhập dữ liệu kiểu 1 và 2, chương trình nên sinh ra tự động số id này cho các công việc. Việc sinh id tự động nên theo thứ tự được nhập vào của công việc (Công việc đầu tiên sẽ có id là 0, công việc thứ hai id là 1,...).

Sau khi đã nhập xong danh sách công việc, chương trình nên in ra lại danh sách công việc đã được nhập, nhưng kèm theo mỗi công việc là một số id đã được sinh ra. Dựa vào danh sách công việc với id này. Người sử dụng có thể nhập tiền điều kiện (xác định công việc trước và sau của một công việc thông qua `id`).

Thuật toán để nhập tiền điều kiện như sau :

```
Nhập tiền điều kiện cho công việc i
    Nhập các id công việc trước của công việc i
    Nhập các id công việc sau của công việc i
    Lặp tiếp với công việc i + 1
```

Việc nhập các `id` có thể là nhập trên một dòng với các `id` cách nhau bằng một khoảng trống. Dùng `cin.getline()` để lấy dòng các id đó. Sau đó, dùng hàm `strtok` để tách lấy các số `id`. (Tham khảo về hàm `strtok` tại http://cplusplus.com/reference/clibrary/cstring/strtok/)

Hoặc dùng cách khác đơn giản hơn. Ví dụ với nhập các `id` công việc trước, đầu tiên ta yêu cầu nhập vào số lượng công việc trước của công việc `i`, sau đó ta dùng vòng lặp for để nhập các `id` của các công việc trước.

Cấu trúc dữ liệu để lưu mạng PERT

Ở phần giới thiệu về phương pháp PERT, tôi đã đưa ra hai cách tính khác nhau cho các giá trị trên mạng PERT. Tương ứng với hai cách tính này là hai cách tổ chức mạng PERT khác nhau. Phụ thuộc vào việc bạn lựa chọn cách tổ chức mạng PERT nào, chúng ta sẽ có cách tổ chức cấu trúc dữ liệu khác nhau và dẫn đến thuật toán là khác nhau. Phần tiếp theo tôi sẽ trình bày cả hai cách tổ chức cấu trúc dữ liệu cho cả hai cách tính giá trị trên mạng PERT.

Cách 1 Với cách này, ta chỉ cần quản lý các cung (công việc). Tôi gọi là cấu trúc dữ liệu `Task`. `Task` bên cạnh việc lưu trữ id, tên công việc, thời gian thực hiện, các thời gian sớm muộn, thời gian dự trữ, đường găng thì còn phải lưu danh sách các công việc trước và danh sách các công việc sau. Hai danh sách này các bạn có thể sử dụng các con trỏ để trỏ về chính các đối tượng `Task` là công việc trước hoặc sau. Để lưu danh sách công việc trước và sau, bạn có thể sử dụng mảng động, hoặc danh sách liên kết, hoặc `vector`.

Lời khuyên của tôi là bạn nên dùng ***`vector`****.*

Như vậy, tạm thời cấu trúc `Task` của tôi có thể viết như sau :

```
1  struct Task
2  {
3      int id; // id cong viec
4      string name; // ten cong viec
5      int duration; // thoi gian thuc hien
6      int est; // thoi gian bat dau som
7      int eft; // thoi gian bat dau muon
8      int lst; // thoi gian ket thuc som
9      int lft; // thoi gian ket thuc muon
10     int slack; // thoi gian du tru
11     bool isCritical; // la duong gang hay khong ?
12
13     vector<Task*> priorList; // ds cong viec truoc
14     vector<Task*> laterList; // ds cong viec sau
15 };
```

Trong môn Cấu trúc dữ liệu và giải thuật, để lưu trữ một danh sách liên kết thông thường người ta dùng con trỏ head trỏ vào đầu của danh sách liên kết đó. Ở đây ta không thể dùng cách này vì mạng PERT có thể có nhiều công việc cùng bắt đầu tại thời điểm 0 (công việc không có công việc trước). Do đó, ta cần có giải pháp khác để lưu trữ toàn bộ danh sách công việc mà lại thuận tiện cho việc duyệt qua danh sách đó một cách dễ dàng. Giải pháp của tôi là dùng một `vector` (tôi đặt tên là `listTask`) và lưu trữ toàn bộ các con trỏ đến công việc được nhập vào trong vector đó. Như vậy, tôi không phân biệt đâu là head đâu là last.

Trong quá trình nhập dữ liệu từ bàn phím, ở bước thứ nhất - nhập danh sách công việc, chúng ta sẽ tạo ra một `Task` mới cho mỗi công việc được nhập vào. Rồi gán `Task` mới đó vào `listTask` (có thể gán vào cuối `listTask`) (Chú ý : lúc đầu ta chỉ có dữ liệu cho `id`, `name` và `duration`). Ở bước thứ hai - nhập tiền điều kiện, thì `id` mà người sử dụng nhập vào chính là vị trí của `Task` trong `listTask`. Dựa vào đó ta có thể lấy ra con trỏ đến công việc trước (hoặc sau) từ `listTask` rồi gán vào `priorList` (hoặc `laterList`).

Cách 2 Với cách này ta cần quản lý cả các cung (công việc) và các nút (sự kiện). Ở đây tôi sẽ có cấu trúc `Task` cho công việc và cấu trúc `Event` cho sự kiện. `Task` lúc này sẽ chỉ lưu trữ id, tên công việc, thời gian thực hiện, thời gian dự trữ, đường găng, và hai sự kiện trước và sau.

Còn `Event` sẽ lưu các thời gian sớm muộn và hai danh sách công việc trước và sau. Các danh sách này bạn cũng có thể sử dụng `vector` để lưu trữ.

Như vậy, cấu trúc `Task` và `Event` của tôi có thể viết như sau :

```
struct Task
{
    int id; // id cong viec
    string name; // ten cong viec
    int duration; // thoi gian thuc hien

    int slack; // thoi gian du tru
    bool isCritical; // la duong gang hay khong ?

    Event* priorEvent; // su kien truoc
    Event* laterEvent; // su kien sau
};
```

```
struct Event
{
    int earliestTime; // thoi gian som
    int latestTime; // thoi gian muon

    vector<Task*> priorTaskList; // ds cong viec truoc
    vector<Task*> laterTaskList; // ds cong viec sau
};
```

Tương tự với cách làm ở cách 1, ta có thể dùng hai `vector` để lưu trữ danh sách các `Task` (`listTask`) và danh sách các `Event` (`listEvent`).

Gần tương tự với cách 1 ở trên, trong quá trình nhập dữ liệu từ bàn phím, ở bước nhập danh sách công việc, chúng ta sẽ tạo ra một `Task` mới cho mỗi công việc được nhập vào. Rồi gán `Task` mới đó vào `listTask` (có thể ở vị trí cuối `listTask`). Ở bước nhập tiền điều kiện, thì dựa trên `id` mà người sử dụng nhập vào, ta xác định con trỏ đến `Task` trước (hoặc sau). Giả sử như đó là `Task` trước. Khi đó, ta sẽ kiểm tra xem `priorEvent` đã có trỏ vào `Event` nào hay chưa. Nếu chưa (`priorEvent == NULL` - tức là phải khởi tạo cho nó bằng `NULL`) thì ta sẽ tạo ra một `Event` mới, gán `Task` đang được nhập công việc trước và sau vào `laterTaskList` và gán `Task` tương ứng với `id` đang xét vào `priorTaskList` và cuối cùng là gán `Event` mới tạo ra vào cuối `listEvent`. Nếu đã có `Event` (`priorEvent != NULL`) thì ta chỉ cần gán `Task` tương ứng với `id` đang xét vào `priorTaskList` của `Event` đó. Với trường hợp `Task` sau, ta cũng làm tương tự.

Cấu trúc dữ liệu dùng trong tính toán các giá trị thời gian

Phương pháp tính toán thời gian sớm và muộn của tôi không dùng các thuật toán đường đi ngắn nhất, cây khung,... Tôi chỉ sử dụng một ý tưởng đơn giản như sau : đối với cây hay đồ

thị, người ta hay dùng một `queue` hoặc `stack` để lưu các cạnh hay nút cần xét tiếp theo. Với `queue`, thì tôi chèn các cạnh (hoặc nút) vào sau và xét (hay tính toán) cho cạnh (hay nút) nằm đầu `queue`. Với `stack`, thì tôi chèn vào sau và xét ở sau. Hay nói cách khác, cách dùng `queue` chính là duyệt cây theo bề rộng và `stack` là duyệt cây theo chiều sâu.

Đối với mạng PERT, ta cần duyệt theo chiều sâu. Do đó, các bạn có thể sử dụng một `stack` cho cấu trúc dữ liệu này hoặc có thể dùng `list` trong bộ thư viện chuẩn của C/C++.

Tham khảo :

1. stack - http://cplusplus.com/reference/stl/stack/
2. list - http://cplusplus.com/reference/stl/list/

Tính thời gian sớm

Trong các phần tính toán thời gian sớm, muộn, thời gian dự trữ, đường găng tôi sẽ chỉ trình bày với cách cấu trúc dữ liệu mạng PERT thứ nhất.

Thuật toán tính thời gian sớm như sau :

```
stack<Task*> stackTask;
Khởi tạo thời gian bắt đầu và kết thúc sớm của tất cả các Task về 0
Tìm các Task trong listTask không có Task trước (priorList là rỗng) và gán
vào stackTask
Trong khi stackTask còn có phần tử để xét
    Lấy phần tử ở cuối stack ra (công việc t)
    Tính thời gian kết thúc sớm theo công thức
    Lấy danh sách công việc sau của công việc t
        Nếu thời gian kết thúc sớm của công việc t > thời gian bắt đầu
        sớm của một công việc sau t thì
            gán thời gian bắt đầu sớm của công việc sau t bằng thời gian
            kết thúc sớm của t
            và gán công việc sau t vào stackTask (push_back)
```

Mô phỏng quá trình tính toán thời gian sớm với stackTask

Ví dụ với mạng pert2, dữ liệu trong stackTask sẽ thay đổi khi tính thời gian sớm như sau :

Bước 1 : Gán các task không có task trước vào stackTask

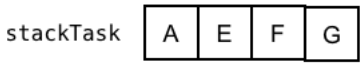
Bước 2 : Lấy phần tử cuối stackTask ra (C) và tính thời gian kết thúc sớm

Bước 3 : Lấy task sau của C là H gán vào stackTask

Bước 4 : Lấy phần tử cuối stackTask ra (H) và tính thời gian kết thúc sớm

Bước 5 : Lấy phần tử cuối stackTask ra (B) và tính thời gian kết thúc sớm

Bước 6 : Lấy task sau của B là E, F, G gán vào stackTask



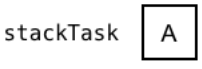
Bước 7 : Lấy phần tử cuối stackTask ra (G) và tính thời gian kết thúc sớm

Bước 8 : Lấy phần tử cuối stackTask ra (F) và tính thời gian kết thúc sớm

Bước 9 : Lấy task sau của F là H gán vào stackTask

Bước 10 : Lấy phần tử cuối stackTask ra (H) và tính thời gian kết thúc sớm

Bước 11 : Lấy phần tử cuối stackTask ra (E) và tính thời gian kết thúc sớm



Bước 12 : Lấy task sau của E là I gán vào stackTask

Bước 13 : Lấy phần tử cuối stackTask ra (I) và tính thời gian kết thúc sớm

Bước 14 : Lấy phần tử cuối stackTask ra (A) và tính thời gian kết thúc sớm

Bước 15 : Lấy task sau của A là D gán vào stackTask

Bước 16 : Lấy phần tử cuối stackTask ra (D) và tính thời gian kết thúc sớm

Tính thời gian muộn

Tương tự như tính thời gian sớm, thuật toán tính thời gian muộn như sau :

```
stack<Task*> stackTask;
Khởi tạo thời gian kết thúc muộn của tất cả các Task về giá trị lớn nhất
của kiểu int
Tìm các Task trong listTask không có Task sau (laterList là rỗng) và gán
vào stackTask
Đồng thời trong quá trình đó, xác định giá trị kết thúc sớm lớn nhất của
các Task không có Task sau để gán làm giá trị kết thúc muộn của các Task
đó
Trong khi stackTask còn có phần tử để xét
    Lấy phần tử ở cuối stack ra (công việc t)
    Tính thời gian bắt đầu muộn theo công thức
    Lấy danh sách công việc trước của công việc t
        Nếu thời gian bắt đầu muộn của công việc t < thời gian kết thúc
        muộn của một công việc trước t thì
            gán thời gian kết thúc muộn của công việc trước t bằng thời
            gian bắt đầu muộn của t
            và gán công việc trước t vào stackTask (push_back)
```

Xác định công việc găng

Duyệt qua danh sách công việc `listTask` và kiểm tra điều kiện trên từng công việc. Nếu đó là công việc găng thì gán `isCritical` của công việc đó là `true`.

Xác định đường găng

Các bạn có thể sử dụng đệ qui để in danh sách các đường găng. Bắt đầu từ các công việc không có công việc trước mà `isCritical` là `true`, ta sẽ đệ qui tiếp cho các công việc sau.

Cách thứ hai là các bạn có thể sử dụng ngay chính `stackTask` để lưu các `Task` găng cần duyệt tiếp theo. Nhưng với cách này, bạn cần lưu lại phần đường găng đã duyệt qua cho các trường hợp có rẽ nhánh.

Tính thời gian dự trữ

Tương tự xác định công việc găng, chỉ cần lặp qua danh sách công việc `listTask` và áp dụng công thức.

Nhập mạng PERT từ file văn bản

Về cách thao tác với tập tin văn bản, các bạn có thể tham khảo :

1. *Giáo trình Kỹ thuật lập trình nâng cao* (Đặng Quế Vinh) : Ch. 7 : Tệp tin
2. *Input/Output with files* - http://cplusplus.com/doc/tutorial/files/

Nội dung của tập tin chứa mạng PERT được tôi quy định như sau :

- Dòng đầu tiên trong tập tin chứa 1 con số - là số công việc (gọi là `n`)
- Từ dòng thứ hai đến hết file, được chia ra thành `n` nhóm, mỗi nhóm 3 dòng
  - Dòng đầu tiên lưu tên công việc và thời gian thực hiện được phân cách bằng 1 khoảng trống
  - Dòng thứ hai là các id của các công việc trước cách nhau bằng khoảng trống
  - Dòn thứ ba là các id của các công việc sau cách nhau bằng khoảng trống

Ví dụ sau đây là nội dung của tập tin pert1.txt :

```
1  10
2  A  4
3
4  4  2
5  B  2
6
7  3
8  C  1
9  0
10 5
11 D  1
```

```
1  10
6
E  2
0
7
F  2
2
6
G  2
5  3
8
H  10
4
9
I  4
6
9
J  1
7  8

K  0
0
3
```

Thuật toán đọc và tạo `listTask` như sau (với cách 1) :

```
Đọc dòng đầu tiên trong file để lấy số lượng công việc (n)
Tạo ra n công việc với name là rỗng và duration là 0 và gán vào listTask
Duyệt qua các công việc trong listTask
    Đọc dòng tiếp theo trong file -> đó là dòng lưu tên công việc
        Tách tên gán vào name
        Tách thời gian công việc gán vào duration
    Đọc dòng tiếp theo -> dòng id của các công việc trước
        Chuyển dòng id này về vector các chỉ số (vector kiểu int)
        Duyệt qua vector này, dựa vào các id để lấy con trỏ tương ứng với
        vị trí thứ id của công việc để gán vào priorList
    Đọc dòng tiếp theo -> dòng id của các công việc sau
        Chuyển dòng id này về vector các chỉ số (vector kiểu int)
        Duyệt qua vector này, dựa vào các id để lấy con trỏ tương ứng với
        vị trí thứ id của công việc để gán vào priorList
```

Một số hàm tiện ích cần dùng

Hàm này nhận một chuỗi các `id` rồi trả ra một `vector` các số nguyên :

```
vector<int> lineToVector(char* str)
{
    vector<int> temp;
    char *pch;
    int v;
    pch = strtok(str, " ,");
    while (pch != NULL)
    {
        v = atoi(pch);
        temp.push_back(v);
        pch = strtok(NULL, " ,");
    }
    return temp;
}
```

Hàm này đổi một chuỗi kiểu `string` sang kiểu `char*` :

```
char* stringToChar(string s)
{
    char *a = new char[s.size() + 1];
    a[s.size()] = 0;
    memcpy(a, s.c_str(), s.size());
    return a;
}
```