**BÁO CÁO PHÂN TÍCH THIẾT KẾ VÀ GIẢI THUẬT**

**Sinh Viên : Ngô Đình Phúc**

**Lớp: KTPM**

**MSV: 15150178**

# Đề bài

**Bài 16**: Với bài toán lựa chọn các hoạt động và thuật toán **Activity Selection** hãy trình bày các nội dung sau:

1. Nêu bài toán;

2. Mô tả chi tiết thuật toán;

3. Đánh giá độ phức tạp thuật toán;

4. Tự xác định 2 bộ dữ liệu (với số phần tử N>=5), với mỗi bộ dữ liệu hãy thực hiện từng bước thuật toán đã mô tả ở mục 2 và ghi ra kết quả mỗi bước;

5. Viết chương trình trên C/C++

# Nêu bài toán:

Có n công việc cần thực hiện; ai -thời điểm bắt đầu, bi -thời điểm kết thúc công việc i (i=1..n)

**Task:**

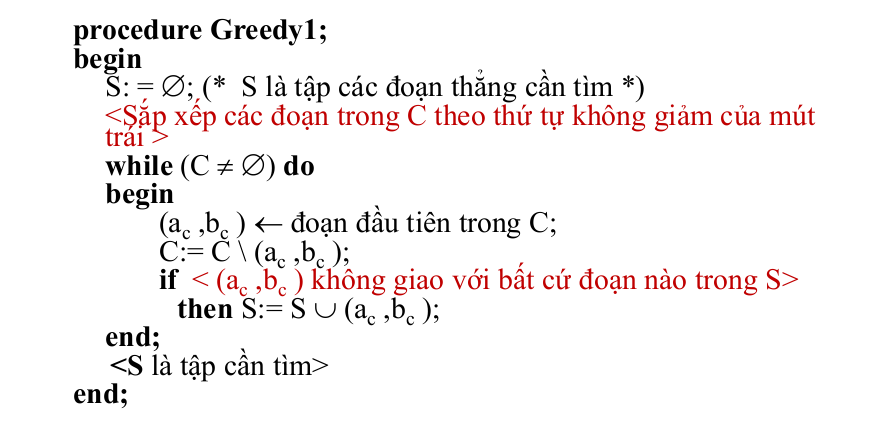
Hãy chọn ra các công việc để một người có thể thực hiện được nhiều việc nhất.

Ý tưởng (tham lam):

* Gọi C là tập các công việc ban đầu
* Gọi S là tập các công việc được lựa chọn

**Trường hợp 1**

* Sắp xếp các công việc theo thứ tự tăng dần của đầu mút trái(ai).
* Lần lượt xét các đoạn trong danh sách theo thứ tự đã sắp xếp và bổ sung đoạn thẳng đang xét vào S nếu nó không có điểm chung với bất cứ đoạn nào trong S.



Cho 8 công việc

6

2

7

1

5

3

4

8

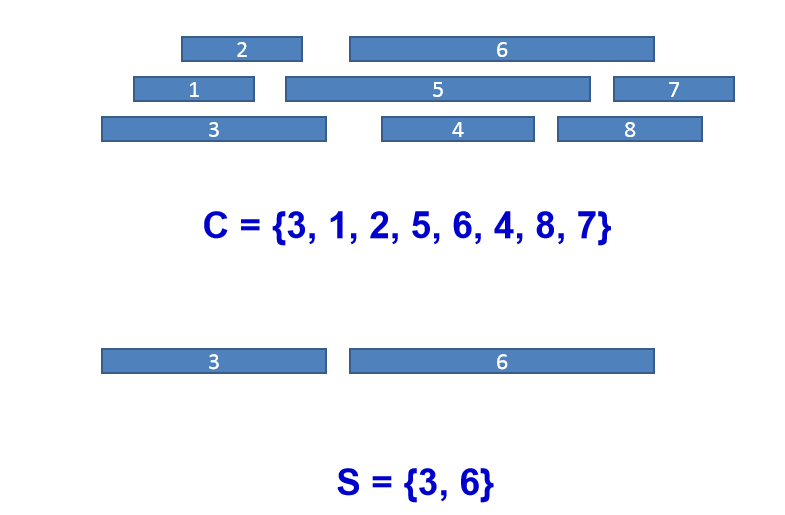
• Sắp xếp công việc theo thứ tự tăng dần của nút trái ta được thứ tự các công việc

C = {3, 1, 2, 5, 6, 4, 8, 7}

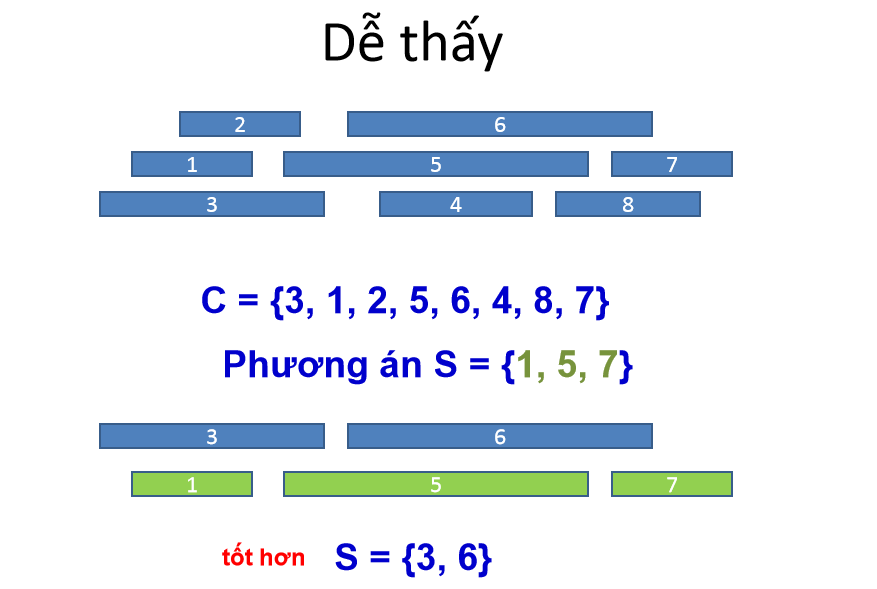
Sau khi thực hiện xong công việc 1 với Dộ dài là 3. Thì khoảng công việc có độ dài 1, và 2 bị xung đột nên bị hủy,

Công việc có độ dài bằng 5 thì có thời gian bắt đầu nằm trong thời gian thực hiện của công việc 3,

Chọn tiếp công việc 6 gần nhất với cv 3, khi chọn 6 thì các công việc có độ dài 4,8,7 bị hủy vì đều có thời gian bắt đầu nằm trong công việc 6



Dễ thấy : Phương án chọn kia chưa hề tối ưu vì ta có thể chọn ra trường hợp mà được số công việc lớn hơn phương án trước



Trường Hợp 2

Ý tưởng (tham lam):

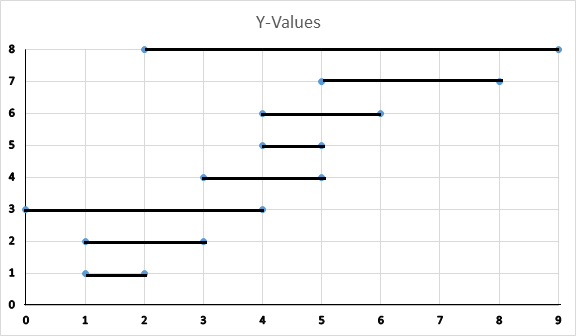
* Gọi C là tập các công việc ban đầu
* Gọi S là tập các công việc được lựa chọn
* Sắp xếp các công việc theo thứ tự tăng dần của thời gian thực hiện công việc (bi - ai).
* Lần lượt xét các đoạn trong danh sách theo thứ tự đã sắp xếp và bổ sung đoạn thẳng đang xét vào S nếu nó không có điểm chung với bất cứ đoạn nào trong S.

Trường Hợp 3

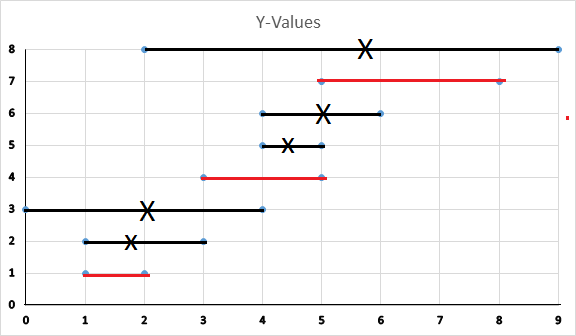
Ý tưởng (tham lam):

* Gọi C là tập các công việc ban đầu
* Gọi S là tập các công việc được lựa chọn
* Sắp xếp các công việc theo thứ tự không giảm của đầu mút phải (bi).
* Lần lượt xét các đoạn trong danh sách theo thứ tự đã sắp xếp và bổ sung đoạn thẳng đang xét vào S nếu nó không có điểm chung với bất cứ đoạn nào trong S.

Ta có dãy công việc Đã được Sắp xếptheo thứ tự không giảm của đầu mút phải (**EndTime**).   
 **C[StartTime,EndTime] ={{1,2},{1,3},{0,4},{3,5},{4,5},{4,6},{5,8},{2,9}}**



Kết Quả lựa chọn công việc



**S[StartTime,EndTime] ={{1,2},{3,5},{5,8}}**

# Mô tả chi tiết thuật toán:

**Input**:

Đầu vào là 1 tập hợp các công việc Gồm có thời gian bắt đầu và thời gian kết thúc (sao cho thời gian bắt đầu nhỏ hơn thời gian kết thúc)

**Output**:

Đầu ra là các công việc được lựa chọn sao cho thỏa mãn yêu cầu (task) của bài toán ( ở đây là các công việc mà 1 người có thể thực hiện nhiều việc nhất)

-------- Thuật Toán -------

Đầu tiên ta có 1 struct biểu diễn công việc gồm StartTime và EndTime

1. **struct** Activity
2. {
3. **int** StartTime, EndTime;
4. }

Tiếp đó là 1 hàm để sắp xếp công việc (radix Sort) theo EndTime

1. **void** ActivitySort(Activity \*a, **int** n)
2. {
3. **int** b[100];
4. **int** i, m = a[0].EndTime, exp = 1;
5. **for** (i = 0; i < n; i++)
6. {
7. **if** (a[i].EndTime > m)
8. m = a[i].EndTime;
9. }
10. **while** (m / exp > 0)
11. {
12. **int** bucket[10] = { 0 };
13. **for** (i = 0; i < n; i++)
14. bucket[a[i].EndTime / exp % 10]++;
15. **for** (i = 1; i < 10; i++)
16. bucket[i] += bucket[i - 1];
17. **for** (i = n - 1; i >= 0; i--)
18. b[--bucket[a[i].EndTime / exp % 10]] = a[i].EndTime;
19. **for** (i = 0; i < n; i++)
20. a[i].EndTime = b[i];
21. exp \*= 10;
22. }
23. }

Chúng thực hiện ActivitySelection

1. **void** ActivitySelection(Activity act[],**int** n)
2. {
3. ActivitySort(act,n);
4. **int** i = 0;
5. printf("Activity %d : (%d-%d)\n", i + 1, act[i].StartTime, act[i].EndTime);
6. **for** (**int** j = 1; j < n; j++)
7. {
8. **if** (act[j].StartTime >= act[i].EndTime)
9. {
10. printf("Activity %d : (%d%d)\n", j + 1, act[j].StartTime, act[j].EndTime);
11. i = j;
12. }
13. }
14. }
15. Sort các hoạt động theo End time thời gian kết thúc
16. Sau khi sort ta lấy công việc thứ 0 ra
17. Lặp từ 1 đến n công vệc  
    nếu hoạt động thứ j=1 có startTime lớn hơn hoặc bằng End Time của công việc trước công việc thứ i=0 thì ta thực hiện  
    - In ra hoạt động thỏa mãn  
    - gán i bằng j và thực hiện công việc tiếp theo thứ j++

# Đánh giá độ phức tạp của thuật toán:

# Tự xác định 2 bộ dữ liệu (với số phần tử N>=5), với mỗi bộ dữ liệu hãy thực hiện từng bước thuật toán đã mô tả ở mục 2 và ghi ra kết quả mỗi bước;

Bộ dữ liệu 1

**C[StartTime,EndTime] ={{1,2},{1,3},{0,4},{3,5},{4,5},{4,6},{5,8},{2,9}}**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | i=0,j=1 | i=0,j=2 | i=0,j=3 | i=3,j=4 | i=3,j=5 |
| act[i].StartTime | 1 | 1 | 1 | 3 | 3 |
| act[i].EndTime | 2 | 2 | 2 | 5 | 5 |
| act[j].StartTime | 1 | 0 | 3 | 4 | 4 |
| act[j].EndTime | 3 | 4 | 5 | 5 | 6 |
|  |  |  | i=j=3 |  |  |
| Lấy (1,2) |  |  | Lấy(3,5) |  |  |
|  |  |  |  |  |  |

|  |  |  |
| --- | --- | --- |
|  | i=3,j=6 | i=6,j=7 |
| act[i].StartTime | 3 | 5 |
| act[i].EndTime | 5 | 8 |
| act[j].StartTime | 5 | 2 |
| act[j].EndTime | 8 | 9 |
|  | i=j=6 |  |
|  | Lấy(5,8) |  |
|  |  |  |