Interval Scheduling

Input → A list of n activities (Jobs), each specified with start and end time, which requires the use of some resource.

→ only one activity can be scheduled on the resource at a time.

→ once the activity is started, it must be run to completion.

Goal :- To find a largest compatible subset.

A subset of requests is compatible if no two activities are overlapped.

Requests 1, 2, -- n, single resource

→ s(i) start time, f(i) finish time

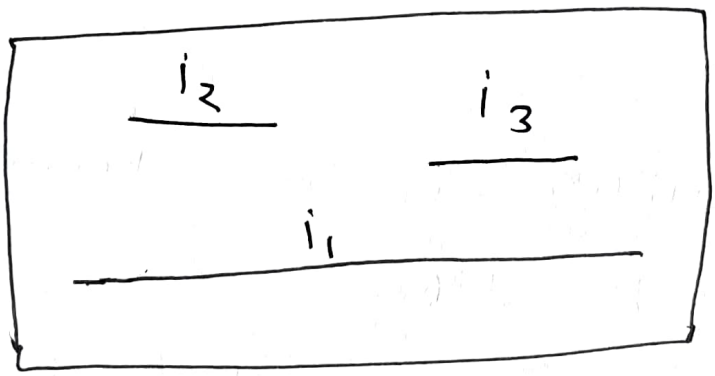
→ s(i) < f(i)

→ Two requests i and j are compatible if they don't overlap

$$f(i) \leq s(j)$$

or

$$f(j) \leq s(i)$$



Possible solution strategies:

→ select a request that starts earliest. one with the least start time.

Output :- ϕ i_1

Optimum :- $\{i_2, i_3\}$

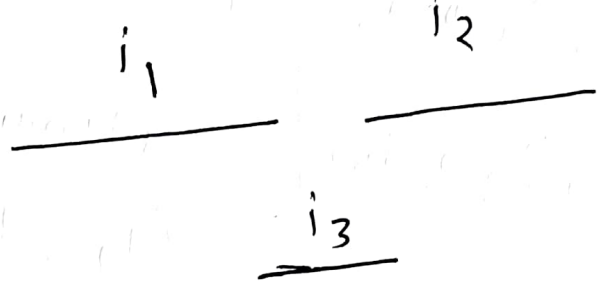
Reason :- If the earliest request i is a very long interval, we may have to reject both of other requests.

5.2 :- Accept the request that has the smallest interval time.

Minimum (f(i) − S(i))

output − {i_3}

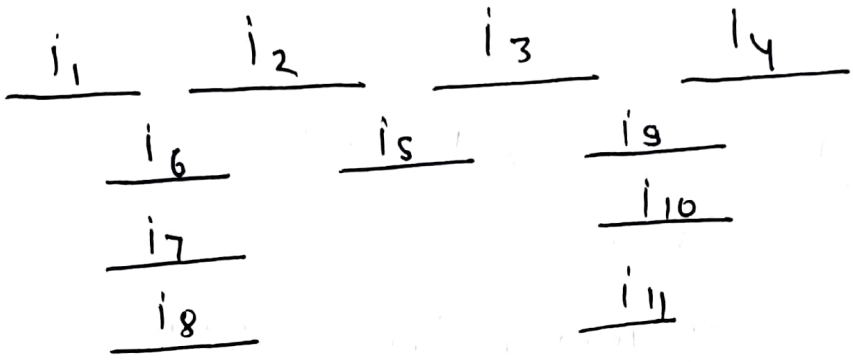
optimum − {i_1, i_2}



Reason :- Accepting a short interval would force to not pick intervals that overlap with the short interval.

5.3 :- For each request, count the number of other requests that are not ~~compatible~~ compatible, and accept the request that has the fewest number of ~~both~~ non-compatible request.

→ select the interval with fewest 'conflicts'.



i_1 — $\{i_6, i_7, i_8\}$

i_2 — $\{i_6, i_5\}$ (D, i_5)

i_3 — $\{i_5, i_9\}$ (D, i_5)

i_4 — $\{i_9\}$.

i_6 - $\{i_1, i_2\}$

i_5 - $\{i_2, i_3\}$

[redacted]

i_7 — $\{i_1, i_2\}$

i_8 - $\{i_1, i_2\}$

i_9 - $\{i_3, i_4\}$ (Due to i_4)

i_{10} - $\{i_3, i_4\}$ " i_4

i_{11} - $\{i_3, i_4\}$ " i_4

Result

$\{i_4, i_5, i_1\}$

optimum

$\{i_1, i_2, i_3, i_4\}$

1- What if we focus on finish time (f(i)) and non-compatible requests? it gives the optimal solution in $\Omega(2^n n^2)$ it?

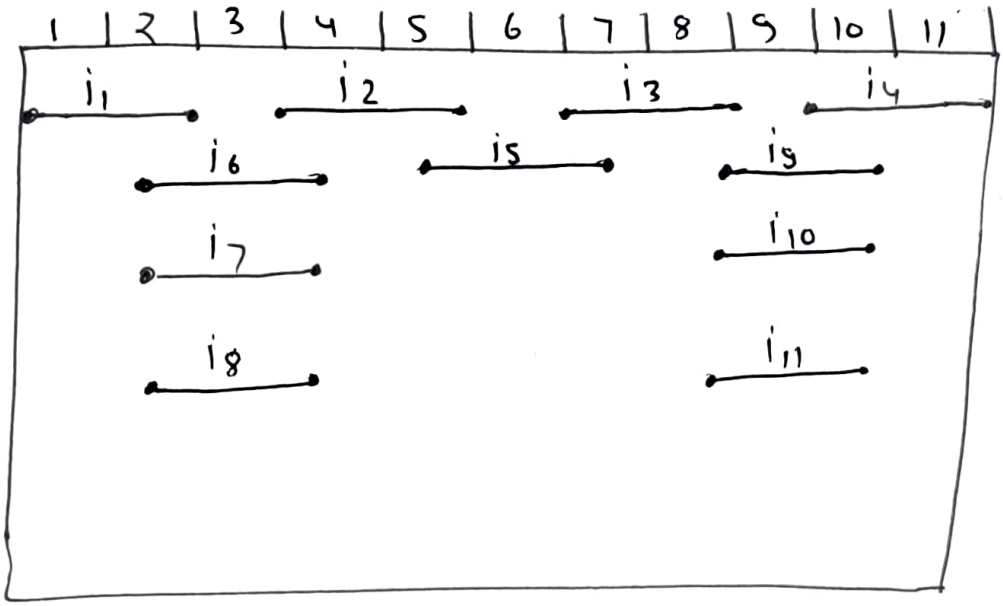
2- What if we use Brute force strategy?

→ 2^n is the time to list all the possible subsets of intervals.

→ $O(n^2)$ to determine compatibility of generated subset.

5.4!-

Accept first the request that finishes first, the request "i" for which f(i) is as small as possible.



Q- is it optimal? (Assignment)

Algorithm:

```
A = {1};
J = 1
for (i=2 to n)
  if s(i) ≥ f(J)
    A = A + {i};
    J = i;
return A;
```

Running time.

sorting the jobs in increasing order
+
scan the list.

$= O(n \log n) + O(n) = O(n \log n)$

Data Compression

An alphabet Σ is a finite non-empty set of symbols.

$$\Sigma = \{A, B, C, D\}$$

fixed-length encoding scheme is

Symbol	Encoding
A	00
B	01
C	10
D	11

Data encoding is used in digital communication and storage.

Now Let suppose we have to store a data file compared of 100 character.

Total length of the file would be

$$100 \times 2 = 200 \text{ bit} + \text{encoding scheme.}$$

Variable length encoding

When some symbols of the alphabet occur much more frequently than other, V.L.E is more efficient than F.L.E.

$\Sigma = \{A, B, C, d\}$

A>=

Now suppose encoding is as follows.

S	E
A	0
B	01
C	10
D	1

freq. A = 40

B = 30

C = 20

d = 10

in a 100 character file. In this case total bits required to store this file is

$40 \times 1 + 30 \times 2 + 20 \times 2 + 10 \times 1$

$= 40 + 60 + 40 + 10 = 150 +$ Encoding.

What is the string "001" ?

So in V.L.E it is unclear where one symbol ~~stops~~ and next one begins. This problem does not arise with fixed length code. If every symbol is encoded using 2 bit, the second symbol always starts with 3rd bit and so on.

Prefix - Free Code

We can eliminate all the ambiguity by insisting that a code be prefix free. Means that each pair of distinct symbols $a, b \in \Sigma$ the encoding of a is not a ~~for~~ prefix of that of b.

Symbol	Encoding
A	0
B	10
C	110
D	111

0 is used to encod A, encoding of other must start with 1

B is encoded as 10, the encoding C, D must begain with 11.

Benefits of Prefix-Free code.

S	Freq
A	60%
B	25
C	10
D	5

S.	F.L.E	V.L.P.F.C
A	00	0
B	01	10
C	10	110
d	11	111

Average per symbol length in F.L.E is 2 bit.

V. L. P. F. E $1*.6 + 2*.25 + 3*(.1 + .05)$

$= 1.55$

Problem - Optimal Prefix-Free Codes

Input : A non-negative frequency P_a for each symbol a of an alphabet Σ of size $n \geq 2$

Output : The prefix-free binary code with minimum possible average encoding length

$\sum_{a \in \Sigma} P_a \cdot$ (number of bits used to encode a)