# Contents

**6.0 Introduction**

**6.1 0/1-Knapsack**

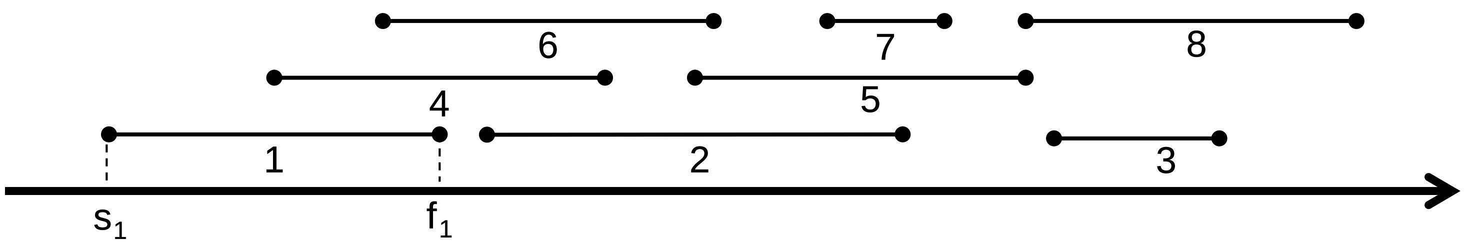
**6.2 Weighted interval scheduling**

6.3 Multistage graph

6.4 All pairs shortest path

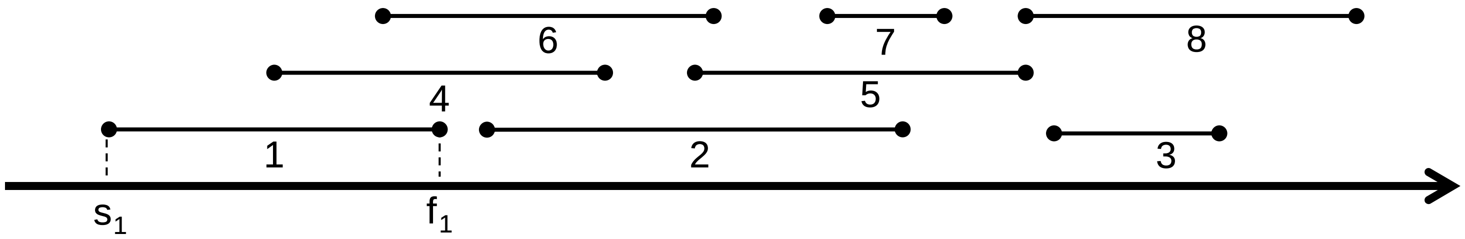
# 6.2 (Weighted) interval scheduling

- Interval scheduling
  - A set of n request: {1, ..., n}
  - $i^{th}$ request
    - {start time ($s_i$), finish time ($f_i$)}
  - Compatible
    - A subset of requests is **compatible**, if no two of them overlap in time.
  - Goal
    - Find a set of maximum compatible subsets



# 6.2 (Weighted) interval scheduling

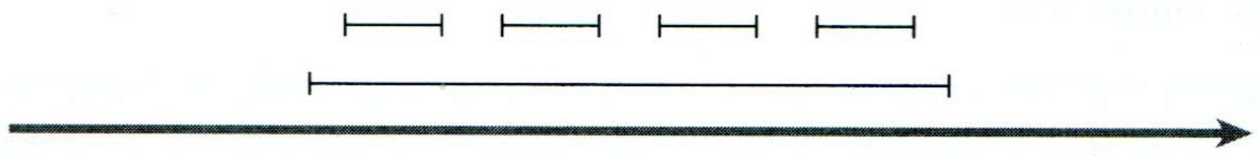
- Designing a greedy algorithm
  - Basic idea
    - Select a first request $i_1$.
    - Reject all the requests that are not compatible with $i_1$.
    - Select the next request $i_2$.
    - Reject all the requests that are not compatible with $i_2$.
    - Repeat this process  until we run out of requests.
  - Greedy algorithm
    - Which rule to select the requests.



# 6.2 (Weighted) interval scheduling

- Designing a greedy algorithm
  - Rule 1.
    - Choose the earliest starting interval.
    - Choose an interval i such that $s_i$ is minimum.

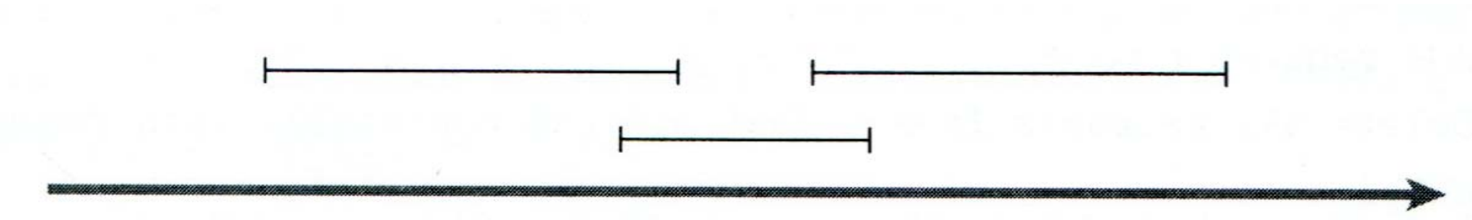
    - Counterexample



# 6.2 (Weighted) interval scheduling

- Designing a greedy algorithm
  - Rule 2.
    - Choose the shortest interval.
    - Choose an interval i such that $f_i - s_i$ is minimum.

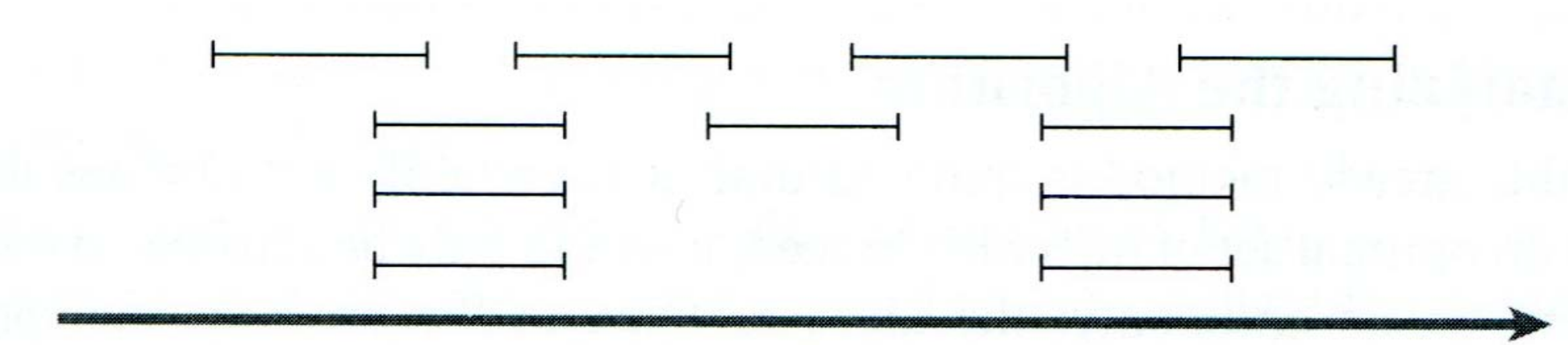
    - Counterexample



# 6.2 (Weighted) interval scheduling

- Designing a greedy algorithm
  - Rule 3.
    - Choose the interval with least conflicts.
    - Choose an interval i that overlaps least intervals.

    - Counterexample



# 6.2 (Weighted) interval scheduling

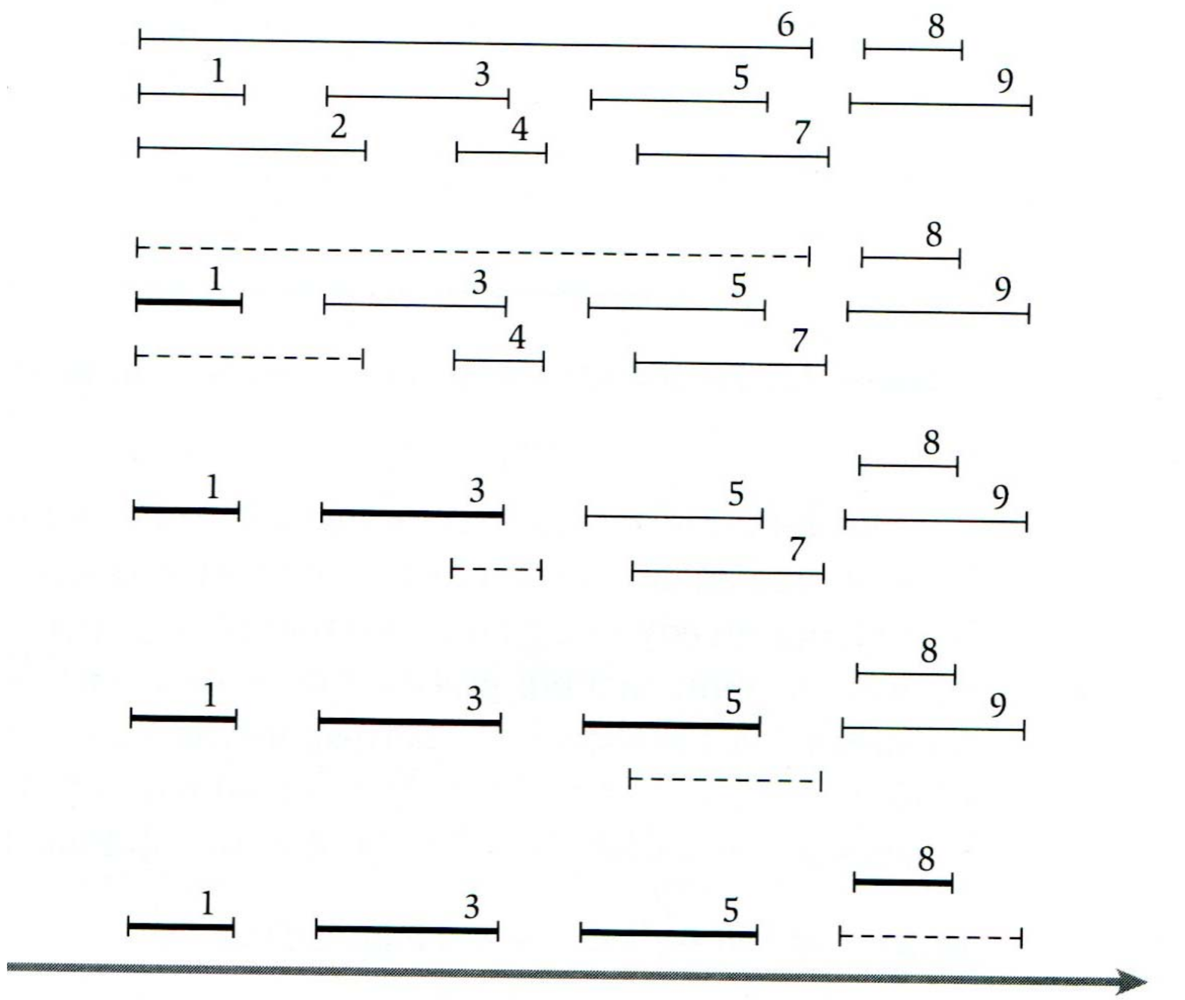
- Designing a greedy algorithm
  - Rule 4.
    - Choose the earliest finishing interval.
    - Choose an interval i such that $f_i$ is minimum.

```
Interval Scheduling( int n, request R )
{
    request A; //      solution
    A ← Φ;
    while ( R is not Φ )
        choose a request i from R whose finish time is minimum;
        add i to A;
        delete all the requests from R that are not compatible with i;

    return A;
}
```

# 6.2 (Weighted) interval scheduling

- Example



# 6.2 Weighted interval scheduling

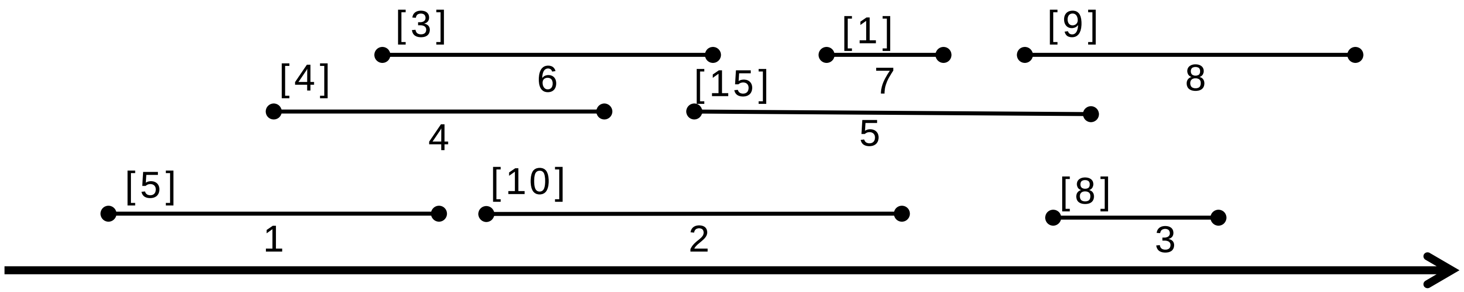
- Extended problem

  → Weighted interval scheduling

  – Interval with different weights

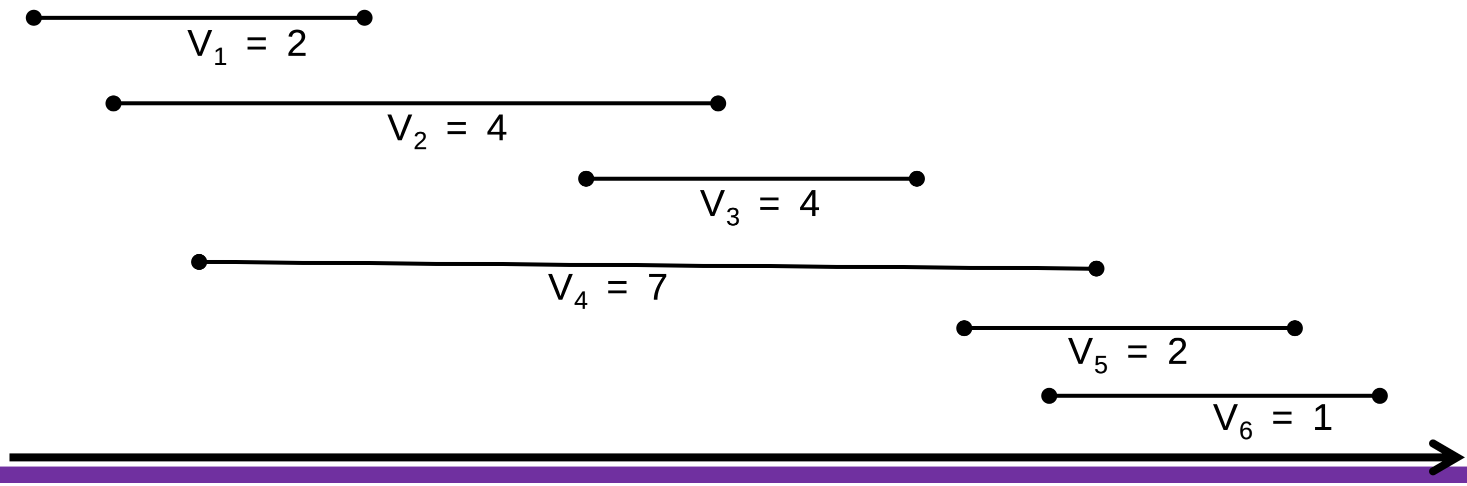
  – Complete requests with maximum weights

**Q1. Is there a greedy algorithm that solves this problem?**



# 6.2 Weighted interval scheduling

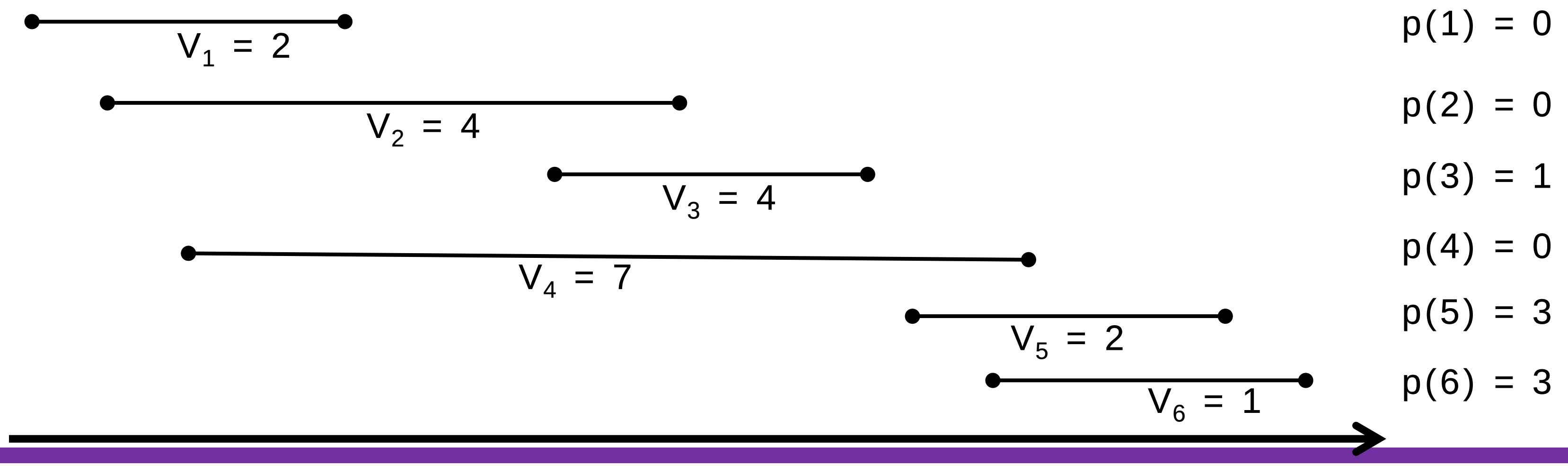
- Problem
  - Each interval is defined by three values: {start time ($s_i$), finish time ($f_i$), weight ($v_i$)}
  - Goal
    - Find a set of maximum compatible subsets (Greedy)
      ➔ **Find a set of compatible intervals such that the sum of the weights is maximum**



# 6.2 Weighted interval scheduling

- Strategy
  - Sort intervals according to the decreasing order of finish time

    {6, 5, 4, 3, 2, 1}
  - Define p(j) for an interval j
    - p(j) = the largest index i < j such that intervals i & j are disjoint



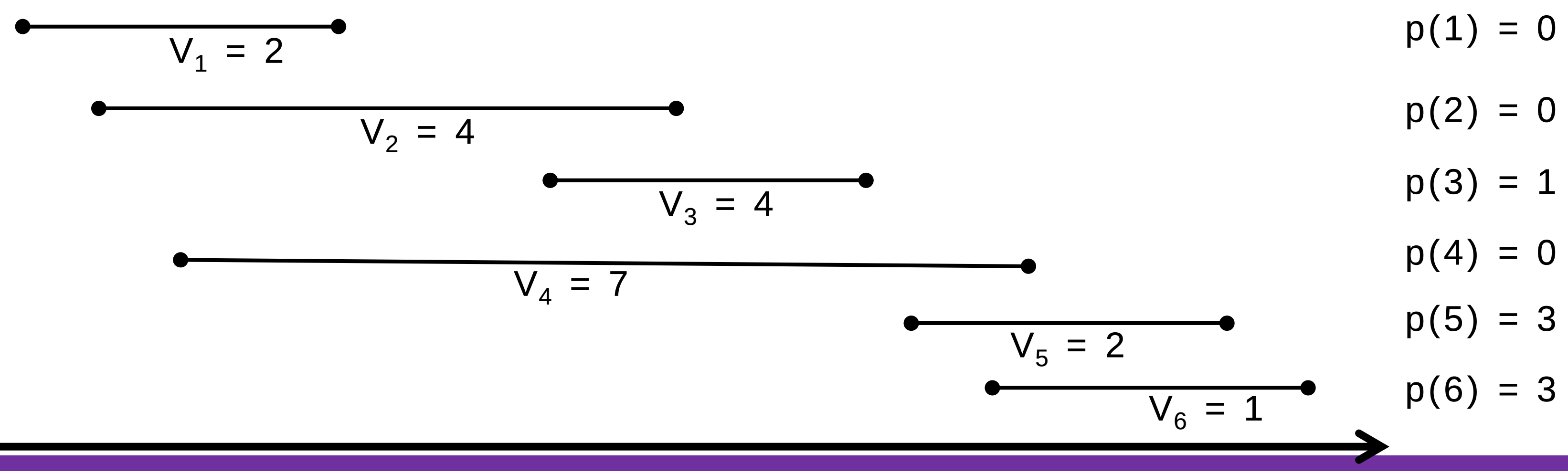
# 6.2 Weighted interval scheduling

- Solution
  - OPT(j)
    - The optimal solution of j intervals

$$\mathrm{OPT}(j) = \max\{v_j + \mathrm{OPT}(p(j)), \mathrm{OPT}(j-1)\}$$

Solution with selecting j-th interval

Solution without selecting j-th interval

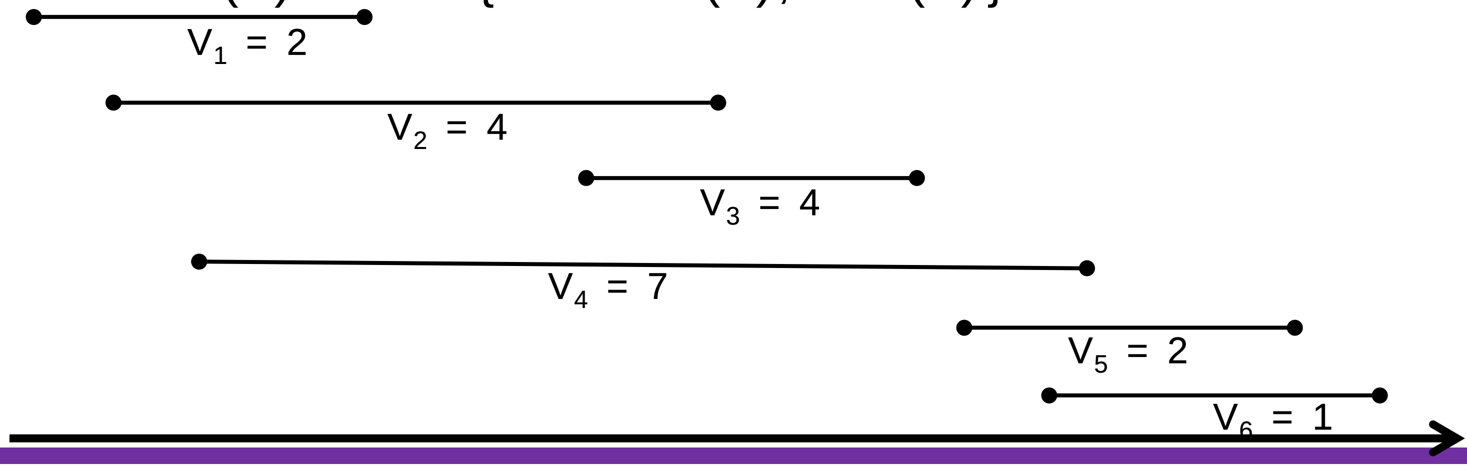


# 6.2 Weighted interval scheduling

- Example

**$OPT(j) = \max\{v_j + OPT(p(j)), OPT(j-1)\}$**

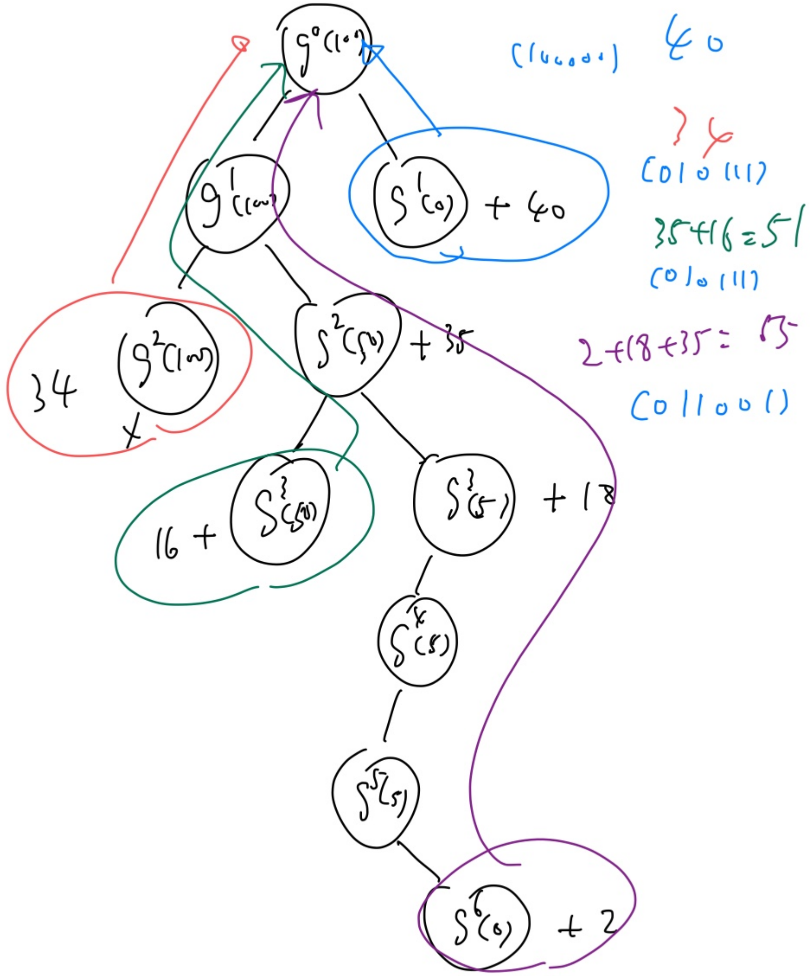
  - 6 intervals
    - $OPT(6) = \max\{v_6 + OPT(p(6)), OPT(5)\}$
      $= \max\{1 + OPT(3), OPT(5)\}$
    - $OPT(5) = \max\{2 + OPT(3), OPT(4)\}$
    - $OPT(4) = \max\{7 + OPT(0), OPT(3)\}$
    - $OPT(3) = \max\{4 + OPT(1), OPT(2)\}$
    - $OPT(2) = \max\{4 + OPT(0), OPT(1)\}$
    - $OPT(1) = \max\{2 + OPT(0), OPT(0)\}$




$p(1) = 0$
$p(2) = 0$
$p(3) = 1$
$p(4) = 0$
$p(5) = 3$
$p(6) = 3$

# 6.2 Weighted interval scheduling

- Comparison



# 6.2 Weighted interval scheduling

- Algorithm (recursive)

```
int OPT ( int j )
{
    if ( j == 0 )
        return 0;

    return max( v[j] + OPT(P[j]), OPT(j-1) );
}
```

  – Problem?
    - Redundantly calling OPT(x)
    - Use an extra memory to void redundant recursive call

# 6.2 Weighted interval scheduling

- Algorithm (memoization & recursive)

```
int OPT ( int j )
{
    if ( j == 0 )
        return 0;

    if ( M[j] is not empty )
        return M[j];

    M[j] = max( v[j] + OPT(P[j]), OPT(j-1) );
    return M[j];
}
```

  – Time complexity?

# 6.2 Weighted interval scheduling

- Algorithm (memoization & non-recursive)

```
int OPT ( int j )
{
    int M[];

    M[0] = 0;

    for ( i = 1; i <= n; i++ )
        M[i] = max( v[i] + M[P[i]], M[i-1] );
}
```

# All about Dynamic Programming

| Type | Stepwise approach | Recursive structure | Approach | Time | Specialty |
|---|---|---|---|---|---|
| 0/1 KNAP | Choosing objects | KNAP(1, n, M) = max {KNAP(2, n, M), KNAP(2, n, M-$w_1$)+$p_1$} | Forward | O($2^n$) | Smart Degenerate Case |
| Weighted Interval Scheduling | Selecting intervals | OPT(j) = max {OPT(p(j)) + $v_j$, OPT(j-1)} | Backward | O(n) | Auxiliary memory |
| Multistage Graph | | | | | |
| All Pairs Shortest Path | | | | | |

# 6.2 Weighted interval scheduling

- 다음 설명 중 옳지 않은 것을 모두 고르시오.

(a) weighted interval scheduling 문제에서 각 interval은 (period, weight)로 표현된다.

(b) weighted interval scheduling 문제는 backward dynamic programming으로 해결할 수 있다.

(c) weighted interval scheduling 문제는 greedy algorithm으로도 해결할 수 있다.

(d) memoization을 이용할 경우 weighted interval scheduling 문제는 재귀적인 방법으로 해결하지 않을 수 있다.