CPSC 320: Intermediate Algorithm Design 2019/20 Term 2

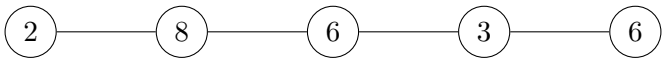
Tutorial 7

Prof. Nick Harvey *University of British Columbia*

1 Independent Set on a Path

Let $G = (V, E)$ be an undirected graph with n nodes. A subset of the nodes is called an *independent set* if no two of them are joined by an edge. Finding large independent sets is difficult in general, but it can be done efficiently if the graph is simple enough.

Let us call the graph G a *path* if its nodes can be written as $v_1, v_2, \ldots, v_n$ where all edges are of the form $v_i v_{i+1}$. Suppose that each node v_i has a positive integer *weight* w_i. For example, in the following path, the weights are the numbers drawn inside the nodes.



The weight of an independent set is defined to be the *sum* of the weights of the nodes in the independent set.

1. Give a succinct English description for what an independent set in this graph must look like.
2. Define recurrence relations for
   - $With[i]$: the maximum sum we can obtain using non-consecutive elements from $v_1, \ldots, v_i$, where we *definitely* include element v_i in the sum.
   - $Without[i]$: the maximum sum we can obtain using non-consecutive elements from $v_1, \ldots, v_i$, where we *do not* include the element v_i in the sum.

3. Solving those recurrences directly would require exponential time. Use the "memoization" technique to give a polynomial-time algorithm that computes the weight of the maximum independent set for the whole path. What is the actual runtime of your algorithm?

```
1: global w[1..n] (the weights of the nodes)
2: global With[0..n], Without[0..n] (the memoization arrays)
3: function MAIN
4:     Initialize all _____
5:     return _____
6: end function
7: function COMPUTEWITH(i)
8:     if With[i]≠null then
9:         return _____
10:    end if
11:    if i=0 then
12:        With[i] ← _____
13:    else
14:        With[i] ← _____
15:    end if
16:    return _____
17: end function
18: function COMPUTEWITHOUT(i)
```

```
19: end function
```

This algorithm runs in _____

4. It is quite clear that these recurrences can be solved "from left to right", by starting with small values of i and building up to larger values. This approach is called "dynamic programming" rather than "memoization". Using this approach give pseudocode that returns the *maximum weight* of an independent set.

```
1: global w[1..n] (the weights of the nodes)
2: global With[1..n], Without[1..n]
3: function MaxWtIndepSetOnPath
```

```
4: end function
```

5. How can we find the independent set itself? It turns out that that information is also encoded in the arrays $With[1..n]$ and $Without[1..n]$? Give pseudocode to extract it.

```
1: global w[1..n] (the weights of the nodes)
2: global With[1..n], Without[1..n]
3: function FindIndepSetByBacktracking
4:     S ← ∅
5:     i ← n + 1
6:     while i ≥ 2 do
```

```
7:     end while
8: end function
```