Lindsey Bieda and Joe Frambach
Dynamic Programming Problems
9.25.2011

8. The input to this problem is a sequence $S$ of integers (not necessarily positive). The problem is to find the consecutive subsequence of $S$ with maximum sum. "Consecutive" means that you are not allowed to skip numbers. For example if the input was

$$12, -14, 1, 23, -6, 22, -34, 13$$

the output would be 1, 23, -6, 22. Give a linear time algorithm for this problem.

An inefficient iterative and naïve algorithm is as follows: Blindly iterating over all possible sets and calculating their sums and recording the maximum summation

```
gmax = $-\infty$, gstart, glength
For start in [0,n]:
    For length in [1, n-start]:
        sum = $\sum_{i=start}^{start+length} S[i]$
        if $sum > gmax$ :
            gmax = sum
            gstart = start
            glength = length
```

For this we can derive a more elegant, informed, and efficiencisized solution by keeping track of a maximum sum for a start point and iterating over all of the possible starting indices

```
For start in [0,n]:
    startmax = $-\infty$, startmaxlength
    sum = S[start]
    For length in [1,n-start]:
        If sum + S[start+length] > startmax:
            sum = sum + s[start+length]
            startmaxlength = length
            startmax = sum
        Else:
            sum = S[start+length]
    If startmax > gmax:
        gmax = startmax
        gstart = start
        glength = startmaxlength
```

From this we can derive the following, where we keep the maximum sum for the start point globally and update it when looking at what the value of adding the next number in the list will be:

```
gmax = S[0], gstartmax = S[0], gend = 0
For i in [1,n]:
    if $gstartmax + S[i] > S[i]$:
        gstartmax = gstartmax + S[i]
    else: gstartmax = S[i]

    if $gstartmax > gmax$:
        gmax = gstartmax
        gend = i
```

9. The input to this problem is a tree $T$ with integer weights on the edges. The weights may be negative, zero, or positive. Give a linear time algorithm to find the shortest simple path in $T$. The length of a path is the sum of the weights of the edges in the path. A path is simple if no vertex is repeated. Note that the endpoints of the path are unconstrained.

Here is an incorrect inefficient recursive algorithm.

```
**minpath**($node$):
    $min = \infty$
    For all $child$ of $node$:
        $childmin$ = weight($node$,$child$) + **minpath**($child$)
        if $childmin <$ weight($node$,$child$):
            $min = childmin$
        else
            $min$ =weight($node$,$child$)
    return $min$
```

This algorithm is incorrect because it does not consider paths between children of the same node.