Algorithms by Jeff E: Dynamic Programming

Siddharth Bhat

Monsoon, second year of the plague

Question: 2a.

```
// cc = closed-closed interval.
int q2a(string s) {
  const int n = s.size();
  vector<int> dp(n+1, 0);
  // dp[i] = number of words in s[0:i)
  for(int i = 1; i <= n+1 ++i) {
      for(int j = 1; j <= i - 1; ++j) {
          if (!isword(s[cc(j, i-1)]) { continue; }
          nwords[i] = max(nwords[i], 1 + nwords[j-1]);
      }
  }
  return nwords[n];
}
```

Question: 2b.

```
// cc = closed-closed interval.
int q2b(string s, string t) {
  const int n = s.size();
  vector<int> dp(n+1, 0);
  // dp[i] = number of words in s[0:i)
  for(int i = 1; i <= n+1 ++i) {
      for(int j = 1; j <= i - 1; ++j) {
          if (!isword(s[cc(j, i-1)])  || !isword(t[cc(j, i-1)])) { continue; }
          nwords[i] = max(nwords[i], 1 + nwords[j-1]);
      }
  }
  return nwords[n] > 0;
}
```

Question: 2c.

```
// cc = closed-closed interval.
int q2c(string s, string t) {
  const int n = s.size();
  vector<int> dp(n+1, 0);
```

```
  // dp[i] = number of words in s[0:i)
  for(int i = 1; i <= n+1 ++i) {
      for(int j = 1; j <= i - 1; ++j) {
          if (!isword(s[cc(j, i-1)])  || !isword(t[cc(j, i-1)])) { continue; }
          nwords[i] = max(nwords[i], 1 + nwords[j-1]);
      }
  }
  return nwords[n];
}
```

Question: 3a: largest sum subarray.

Standard solution: kendane's algorithm

```
int q3a(const vector<int> &xs) {
    const int n = xs.size();
    int best = 0;
    int cur = 0;
    for(int i = 0; i < n; ++i) {
        if (cur + xs[i] < 0) { continue; }
        cur += xs[i];
        best = max<int>(best, cur);
    }
    return best;
}
```

$\Theta(n^2)$ solution using

Question: 3b: largest product subarray.

```
int best = 1, pos = 1, neg = 1;
int q3b(const vector<int> &xs) {
    const int n = xs.size();
    for(int i = 0; i < xs.size(); ++i) {
      if (xs[i] == 0) { pos = neg = 1; }
      if (xs[i] < 0) {
           const int prevpos = pos;
           pos = neg  * xs[i];
           neg = prevpos * xs[i];
       }
      if (xs[i] > 0) {
        pos *= xs[i];
        neg *= xs[i];
```

```
      }
      best = max<int>(best, pos);
    }
    return best;
}
```