Tutorial 5: Dynamic programming

1 Maximum subarray

In the *maximum subarray* problem, we are given as input an array $A[1], \ldots, A[n]$ of n integers. Our goal is to find the maximum value of any subarray of A. I.e., the valid solution is the value

$$\max_{i,j:1\le i\le j\le n} \sum_{\ell=i}^{j} A[\ell].$$

Design a dynamic programming algorithm that solves the maximum subarray problem.

1.1 Hint

Consider computing the values $B[1], \ldots, B[n]$ where $B[j]$ is the maximum subarray that ends at $A[j]$—i.e., $B[j] = \max_{i:1\le i\le j} \sum_{\ell=i}^{j} A[\ell]$.

1.2 Solution

The dynamic programming algorithm is known as *Kadane's algorithm.* The key observation is that there are only two possibilities to consider for $B[j]$: either it includes the maximum subarray that ends at $A[j-1]$ (along with $A[j]$), or it does not (in which case $B[j] = A[j]$). This observation immediately yields the following algorithm.

Algorithm 1: Kadane($A[1..n]$)

```
1 B[1] ← A[1];
2 msub ← B[1];
3 for j = 2, ..., n do
4     B[j] ← max{A[j], B[j − 1] + A[j]};
5     msub ← max{msub, B[j]};
6 return msub;
```

Theorem 1. *The* Kadane *algorithm solves the maximum subarray problem.*

Proof. We first need to show that the values $B[1], \ldots, B[n]$ satisfy

$$B[j] = \max_{i:1\le i\le j} \sum_{\ell=i}^{j} A[\ell]$$

for every $j \in \{1, 2, \ldots, n\}$. We do so by induction on j. In the base case, when $j = 1$ the only subarray that ends at $A[1]$ is the subarray that includes that value, so $B[1]$ satisfies the condition. For the induction step, assume that $B[j-1] = \max_{i:1\le i\le j-1} \sum_{\ell=i}^{j-1} A[\ell]$. Then

$$\begin{aligned} B[j] &= \max\left\{B[j-1] + A[j], A[j]\right\} \\ &= \max\left\{\max_{i:1\le i\le j-1} \sum_{\ell=i}^{j-1} A[\ell] + A[j], A[j]\right\} \\ &= \max\left\{\max_{i:1\le i\le j-1} \sum_{\ell=i}^{j} A[\ell], \sum_{\ell=j}^{j} A[\ell]\right\} \\ &= \max_{i:1\le i\le j} \sum_{\ell=i}^{j} A[\ell], \end{aligned}$$

as we wanted to show.

Finally, since the maximum subarray of A must terminate at $A[j]$ for some value of $j \in \{1, 2, \ldots, n\}$, then it equals $B[j]$ for some value of j and `msub` $\leftarrow B[j]$ at that iteration of the for loop. □

Theorem 2. *The time complexity of the* Kadane *algorithm is* $\Theta(n)$.

Proof. The operations in lines 1,2, 4, and 5 all take constant time, and the for loop is run $n - 1$ times. □

2 Longest increasing subsequence

Given a sequence of non-negative integers $a_1, \ldots, a_n$, design an algorithm that computes the length of the longest subsequence of increasing numbers within the original sequence. For example, on instance

$$0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15$$

one of the longest increasing subsequences is $0, 4, 6, 9, 11, 15$ so the valid solution for this instance is 6. (Note that there are multiple increasing subsequences of length 6 for this example.)

Design an algorithm that solves the longest increasing subsequence in time $O(n^2)$.

2.1 Solution

One slick solution to this problem is to reduce it to the longest common subsequence, by sorting the list of integers and finding the length of the longest common subsequence between the original sequence of integers and its sorted version.

We can also design an algorithm for this problem directly using the dynamic programming technique. For $i = 1, 2, \ldots, n$, let $M[i]$ denote the length of the longest increasing subsequence of the sequence $a_1, \ldots, a_i$. Then $M[1] = 1$. For larger values of k, the longest increasing subsequence of $a_1, \ldots, a_k$ either includes a_k or it does not. Therefore,

$$M[k] = \max\{M[j] + 1, M[k-1]\}$$

when $j < k$ is the largest index for which $a_j < a_k$. The algorithm we obtain by implementing this approach is as follows.

Algorithm 2: LIS$(a_1, \ldots, a_n)$

```
1 M[0] ← 0;
2 a_0 ← −∞;
3 for k = 1, ..., n do
4     j ← max{0 ≤ ℓ < k : a_ℓ < a_k};
5     M[k] ← max{M[j] + 1, M[k − 1]};
6 return M[n];
```

Theorem 3. *The* LIS *algorithm solves the longest increasing subsequence problem.*

Proof. We prove the correctness of the algorithm by showing that $M[k]$ is equal to the length of the longest increasing subsequence of $a_1, \ldots, a_k$ by induction on k. For the base case, when $k = 0$ the longest increasing subsequence of an empty sequence has length 0, as defined in line 1 of the algorithm.

For the induction step, assume that for every $\ell < k$, $M[\ell]$ corresponds to the length of the longest increasing subsequence of the sequence $a_1, \ldots, a_\ell$. We now consider the two possible cases regarding the longest common subsequence of $a_1, \ldots, a_k$.

Case 1. If $a_1, \ldots, a_k$ has a longest increasing subsequence that does not include a_k, then the subsequence is in fact a subsequence of $a_1, \ldots, a_{k-1}$ so $M[k] = M[k-1]$.

Case 2. If $a_1, \ldots, a_k$ has a longest increasing subsequence that does include a_k, then that subsequence does not include any element a_ℓ, $\ell < k$, that satisfies $a_\ell > a_k$. In particular, if $j < k$ is the largest index for which $a_j < a_k$, that longest increasing subsequence does not include any of the elements $a_{j+1}, a_{j+2}, \ldots, a_{k-1}$ and so the subsequence with a_k removed is a subsequence of $a_1, \ldots, a_j$; the length of the LIS in this case is therefore $M[j] + 1$.

From the above argument, and since either a_k is part of the longest increasing subsequence of $a_1, \ldots, a_k$ or not, we have that $M[k] = \max\{M[k-1], M[j]+1\}$ with $j < k$ being the largest index for which $a_j < a_k$, as we wanted to show. □

Theorem 4. *When step 4 is implemented with a naïve linear search through all the indices* $0 \leq \ell < k$*, the* LIS *algorithm has time complexity* $\Theta(n^2)$*.*

Proof. The main for loop is run n times and the total time complexity of the operations inside this loop is $\sum_{k=1}^{n} \Theta(k) = \Theta(n^2)$. □