Dynamic Programming Part 3

Arnav Gupta

April 9, 2024

Contents

1 Edit Distance

Input: arrays of characters A and B of length n and m respectively

Output:minimum number of add, delete, or change operations that turn A into B

1.1 Recurrence

Definition: Let $D[i, j]$ be the edit distance between $A[1..i]$ and $B[1..j]$

- $D[0, j] = j$ for all j (add j characters)
- $D[i, 0] = i$ for all i (delete i characters)
- $D[i, j]$ is the min of:
  - $D[i-1, j-1]$ if $A[i] = B[j]$ or $D[i-1, j-1]$ otherwise
  - $D[i-1, j] + 1$ (delete $A[i]$ and match $A[1..i-1]$ with $B[1..j]$)

– $D[i, j-1] + 1$ (add $B[j]$ and match $A[1..i]$ with $B[1..j-1]$)

The algorithm computes all $D[i, j]$ using 2 nested loops so runtime $\Theta(mn)$

2 Optimal Binary Search Trees

Input: integers from 1 to n, probabilities of access $p_1, \ldots, p_n$ that add to 1

Output: an optimal BST with keys from 1 to n, where optimal minimizes

$$\sum_{i=1}^{n} p_i \cdot (\text{depth}(i) + 1)$$

which is the expected number of tests for a search

Optimal static ordering for linked lists and Huffman trees are both built using greedy algorithms.

Greedy algorithm doesn't work for optimal binary search trees.

2.1 Recurrence

Definition: define $M[i, j]$ with $M[i, j]$ being the minimal cost for items $\{i, \ldots, j\}$ and $M[i, j] = 0$ for $j < i$

Recurrence

$$M[i, j] = \min_{i \le k \le j} \left(M[i, k-1] + M[k+1, j]\right) + \sum_{\ell=i}^{j} p_\ell$$

This gives $M[i, i] = p_i$.

2.2 Algorithm

Remark: to get $\sum_{\ell=i}^{j} p_\ell$

- compute $S[\ell] = p_1 + \cdots + p_\ell$ for $\ell = 1, \ldots, n$
- then $p_i + \cdots + p_j = S[j] - S[i-1]$ with $S[0] = 0$

```
OptimalBST(p1, ..., pn, S0, ..., Sn):
  for i in range(1, n+1):
      M[i, i-1] = 0
for d in range(0, n-1):
```

```
for i in range(1, n - d):
    j = d - 1
    M[i,j] = min for i <= k <= j of (M[i, k-1] + M[k+1, j]) + S[j] - S[i-1]
```

3 Independent Sets in Trees

An independent set of a graph $G = (V, E)$ is $S \subseteq V$ if there are no edges between elements of S.

The maximum independent set problem (for a general graph) has:

- **input**: $G(V, E)$
- **output**: an independent set of maximum cardinality

3.1 Algorithm

$I(v)$ is the size of the largest independent set of the subtree rooted at v

$$I(v) = \max \left\{ 1 + \sum_{\text{grandchildren } u \text{ of } v} I(u), \sum_{\text{children } u \text{ of } v} I(u) \right\}$$