CS 341: Algorithms
Lec 13: Dynamic Programming- Part 3

Armin Jamshidpey Collin Roberts

Based on lecture notes by Éric Schost

David R. Cheriton School of Computer Science, University of Waterloo

Winter 2025

Edit Distance

Input: arrays $A[1..n]$ and $B[1..m]$ of characters

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

Example: $A =$**snowy**, $B =$**sunny**

s	**n**	**o**	**w**	**y**
s	**u**	**n**	**n**	**y**

3C

s	_	**n**	**o**	**w**	**y**
s	**u**	**n**	**n**	_	**y**

1A, 1C, 1D

_	**s**	**n**	**o**	**w**	_	**y**
s	**u**	**n**	_	_	**n**	**y**

2A, 1C, 2D

Examples: DNA sequences made of **a**, **c**, **g**, **t**

The recurrence

Definition: let $\boldsymbol{D[i,j]}$ be the edit distance between $\boldsymbol{A[1..i]}$ and $\boldsymbol{B[1..j]}$

- $D[0, j] = j$ for all j (add j characters)
- $D[i, 0] = i$ for all i (delete i characters)
- $D[i, j]$ is the min of **three** values
  - $D[i-1, j-1]$ (if $A[i] = B[j]$) or $D[i-1, j-1] + 1$ (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 two nested loops, so runtime $\boldsymbol{\Theta(mn)}$

Optimal binary search trees

Input:

- integers (or something else) $1, \ldots, n$
- probabilities of access $p_1, \ldots, p_n$, with $p_1 + \cdots + p_n = 1$

Output:

- an **optimal** BST with keys $1, \ldots, n$
- **optimal:** minimizes $\sum_{i=1}^{n} p_i \cdot (\text{depth}(i) + 1)$ = expected number of tests for a search

Optimal binary search trees

Input:

- integers (or something else) $1, \ldots, n$
- probabilities of access $p_1, \ldots, p_n$, with $p_1 + \cdots + p_n = 1$

Output:

- an **optimal** BST with keys $1, \ldots, n$
- **optimal:** minimizes $\sum_{i=1}^{n} p_i \cdot (\text{depth}(i) + 1)$ = expected number of tests for a search

Example: $p_1 = p_2 = p_3 = p_4 = p_5 = 1/5$: ?

Optimal binary search trees

Input:

- integers (or something else) $1, \ldots, n$
- probabilities of access $p_1, \ldots, p_n$, with $p_1 + \cdots + p_n = 1$

Output:

- an **optimal** BST with keys $1, \ldots, n$
- **optimal:** minimizes $\sum_{i=1}^{n} p_i \cdot (\text{depth}(i) + 1)$ = expected number of tests for a search

Example: $p_1 = p_2 = p_3 = p_4 = p_5 = 1/5$: ?

See also

- optimal static ordering for **linked lists**
- **Huffman trees**

both built using greedy algorithms

Setting up the recurrence

Definition define $\boldsymbol{M[i,j]}$ by

- $M[i,j] =$ the minimal cost for items $\{i,\ldots,j\}$, $1 \le i \le j \le n$
- $M[i,j] = 0$ for $j < i$

Recurrence

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

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

check: gives $M[i,i] = p_i$

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(p_1, ..., p_n, S_0, ..., S_n)
1.  for i = 1, ..., n+1
2.      M[i, i-1] ← 0
3.  for d = 0, ..., n-1                d = j - i
4.      for i = 1, ..., n-d
5.          j ← d + i
6.          M[i, j] ← min_{i≤k≤j}(M[i, k-1] + M[k+1, j]) + S[j] - S[i-1]
```

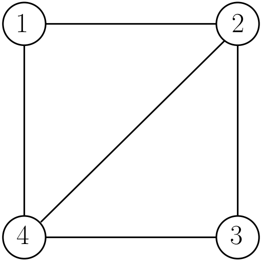
Runtime $O(n^3)$

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):
input: $G(V, E)$
Output: An independent set of maximum cardinality.

Example (not a tree):



$S = \{1, 3\}$.

Algorithm (sketch)

$I(v) :=$ size of largest independent set of subtree rooted at v

Algorithm (sketch)

$I(v) :=$ size of largest independent set of subtree rooted at v

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