Datenstrukturen und Algorithmen

Exercise 8

FS 2021

Program of today

1. Quiz
2. Feedback of last exercise

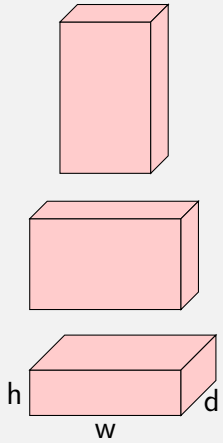
1. Quiz

Quiz: Stacking Boxes

- Given: n boxes with sizes $w_i \times d_i \times h_i$
- Wanted: maximal height of a permitted stack
- Permitted stack: the base area of stacked boxes must become strictly smaller in both directions (width and depth)

Boxen Stapeln

We assume that there are enough boxes of a kind such that each box is available in all orientations (right hand side of the figure below).

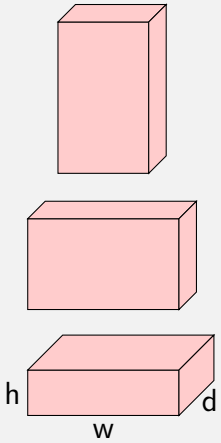


Box	1	2	3	4	5	6
$[w \times d \times h]$	$[1 \times 2 \times 3]$	$[1 \times 3 \times 2]$	$[2 \times 3 \times 1]$	$[3 \times 4 \times 5]$	$[3 \times 5 \times 4]$	$[4 \times 5 \times 3]$

Boxen Stapeln

We assume that there are enough boxes of a kind such that each box is available in all orientations (right hand side of the figure below).

Design a DP Algorithm to find the maximum height of a permitted stack



Box	1	2	3	4	5	6
$[w \times d \times h]$	$[1 \times 2 \times 3]$	$[1 \times 3 \times 2]$	$[2 \times 3 \times 1]$	$[3 \times 4 \times 5]$	$[3 \times 5 \times 4]$	$[4 \times 5 \times 3]$

Solution Idea

- $n \times n$ Table
- Entry at row i and column j: height of highest possible stack formed from maximally i boxes and basement box j.

$[w \times d]$	$[1 \times 2]$	$[1 \times 3]$	$[2 \times 3]$	$[3 \times 4]$	$[3 \times 5]$	$[4 \times 5]$
h	3	2	1	5	4	3
1	<u>3</u>	2	1	5	4	3
2	3	2	<u>4</u>	8	8	8
3	3	2	4	<u>9</u>	8	11
4	3	2	4	9	8	<u>12</u>

Determination of the table: $\Theta(n^3)$, for each entry all entries in the row above must be considered. Computation of the optimal solution by traversing back, worst case $\Theta(n^2)$

Alternative Solution Idea

- $1 \times n$ Table, topologically sorted[1] according to half-order stackability
- Entry at index j: height of highest possible stack with basement box j.

$[w \times d]$	$[1 \times 2]$	$[1 \times 3]$	$[2 \times 3]$	$[3 \times 4]$	$[3 \times 5]$	$[4 \times 5]$
h	3	2	1	5	4	3
	3	2	4	9	8	12

Topological sort in $\Theta(n^2)$. Traverse from left to right in $\Theta(n)$, overal $\Theta(n^2)$. Traversing back also $\Theta(n^2)$

[1] explanation soon

2. Feedback of last exercise

Piecewise Constant Approximation

$$H_{\gamma,y} : \mathcal{P} \mapsto \gamma|\mathcal{P}| + \sum_{I \in \mathcal{P}} \sum_{i \in I} (y_i - \mu_I)^2$$

Piecewise Constant Approximation

$$H_{\gamma,y} : \mathcal{P} \mapsto \gamma|\mathcal{P}| + \sum_{I \in \mathcal{P}} \sum_{i \in I} (y_i - \mu_I)^2$$

- As in exercise 1 efficient computation of mean: $\mu_I = \frac{1}{|I|} \sum_{i \in I} y_i$

Piecewise Constant Approximation

$$H_{\gamma,y} : \mathcal{P} \mapsto \gamma|\mathcal{P}| + \sum_{I \in \mathcal{P}} \sum_{i \in I} (y_i - \mu_I)^2$$

- As in exercise 1 efficient computation of mean: $\mu_I = \frac{1}{|I|} \sum_{i \in I} y_i \Rightarrow$ prefixsum ✓

Piecewise Constant Approximation

$$H_{\gamma,y} : \mathcal{P} \mapsto \gamma|\mathcal{P}| + \sum_{I \in \mathcal{P}} \sum_{i \in I} (y_i - \mu_I)^2$$

- As in exercise 1 efficient computation of mean: $\mu_I = \frac{1}{|I|} \sum_{i \in I} y_i \Rightarrow$ prefixsum ✓
- Efficient computing $e_{[l,r)} = \sum_{i=l}^{r-1} (y_i - \mu_{[l,r)})^2$

Piecewise Constant Approximation

$$H_{\gamma,y} : \mathcal{P} \mapsto \gamma|\mathcal{P}| + \sum_{I \in \mathcal{P}} \sum_{i \in I} (y_i - \mu_I)^2$$

- As in exercise 1 efficient computation of mean: $\mu_I = \frac{1}{|I|} \sum_{i \in I} y_i \Rightarrow$ prefixsum ✓
- Efficient computing $e_{[l,r)} = \sum_{i=l}^{r-1} (y_i - \mu_{[l,r)})^2$
  $\Rightarrow e_{[l,r)} = \sum_{i=l}^{r-1} y_i^2 - \frac{1}{r-l} \left(\sum_{i=l}^{r-1} y_i \right)^2$ ✓

Piecewise Constant Approximation

$$H_{\gamma,y} : \mathcal{P} \mapsto \gamma|\mathcal{P}| + \sum_{I \in \mathcal{P}} \sum_{i \in I} (y_i - \mu_I)^2$$

- As in exercise 1 efficient computation of mean: $\mu_I = \frac{1}{|I|} \sum_{i \in I} y_i \Rightarrow$ prefixsum ✓
- Efficient computing $e_{[l,r)} = \sum_{i=l}^{r-1} (y_i - \mu_{[l,r)})^2$
  $\Rightarrow e_{[l,r)} = \sum_{i=l}^{r-1} y_i^2 - \frac{1}{r-l} \left(\sum_{i=l}^{r-1} y_i \right)^2$ ✓
- **Dynamic programming**: definition of the table, computation of an entry, calculation order, extracting solution

Piecewise Constant Approximation

$$H_{\gamma,y} : \mathcal{P} \mapsto \gamma|\mathcal{P}| + \sum_{I \in \mathcal{P}} \sum_{i \in I} (y_i - \mu_I)^2$$

- As in exercise 1 efficient computation of mean: $\mu_I = \frac{1}{|I|} \sum_{i \in I} y_i \Rightarrow$ prefixsum ✓
- Efficient computing $e_{[l,r)} = \sum_{i=l}^{r-1} (y_i - \mu_{[l,r)})^2$
  $\Rightarrow e_{[l,r)} = \sum_{i=l}^{r-1} y_i^2 - \frac{1}{r-l} \left(\sum_{i=l}^{r-1} y_i \right)^2$ ✓
- **Dynamic programming**: definition of the table, computation of an entry, calculation order, extracting solution $\Rightarrow$ **?**

Dynamic programming

- **Definition of the DP table**: two tables: B and V with each $n+1 \times 1$ entries, $B[k]$ contains the pointer to the end of the best previous interval, $V[k]$ contains the corresponding attainable minimum of H_γ.
- **Computation of an entry**: for computing new entry in $B[k+1]$ compute H for all partitions from 0 to $k+1$.
- **Calculation order**: from left to right
- **Extracting the solution**: construct intervals with $B[n]$ going from right to left, Minimum is given by $V[n]$

Sums

Given a data vector of length $n \in \mathbb{N}$: $(y_i)_{i=1\ldots n} \in \mathbb{R}^n$

Sum $m_n := \sum_{i=1}^{n} y_i \Rightarrow \mu_n = m_n/n$

Sum of Squares $s_n := \sum_{i=1}^{n} y_i^2$

$$
\begin{aligned}
e_n &:= \sum_{i=1}^{n} (y_i - \mu_n)^2 = \sum_{i=1}^{n} y_i^2 - 2\mu_n y_i + \mu_n^2 \\
&= s_n - 2\mu_n \left(\sum_{i=1}^{n} y_i \right) + n \cdot \mu_n^2 = s_n - 2\mu_n \cdot n\mu_n + n \cdot \mu_n^2 \\
&= s_n - n \cdot \mu_n^2 = s_n - m_n^2/n
\end{aligned}
$$

Statistics

```
// post: return mean of data[from,to)
double mean(unsigned int from, unsigned int to) const{
    assert(from < to && to <= n);
    return getsum(vsum,from,to) / (to-from);
}

// post: return err of constant approximation in interval [from,to)
double err(unsigned int from, unsigned int to) const{
    assert(from < to && to <= n);
    double m = getsum(vsum,from,to);
    return getsum(vssq,from,to) - m*m / (to-from);
}
```

DP – Setup and Base Case

```
double MinimizeH(double gamma,const Statistics& s,
                 std::vector<double>& result){
    int n = s.size();
    // B[k] contains the pointer to the end of the best previous interval
    // i.e. best possible approximation is given by
    // best possible approximation of [0,B[k]), [B[k],k)
    std::vector<int> B(n+1);
    // V(k) contains the corresponsing attainable minimum of H_gamma
    std::vector<double> V(n+1);
    // base case: empty interval
    B[0] = 0;
    V[0] = 0;
```

DP – Construct Table

```
// now consider all combinations of Partition([0, left)) + [left, right)
for (int right=1; right <= n; ++right){
    // interval [0, right)
    int best = 0;
    double min = gamma + s.err(0,right);
    // intervals [left, right), left > 0
    for (int left = 1; left < right; ++left){
        double h = V[left] + gamma + s.err(left,right);
        if (h < min){
            min = h; best = left;
        }
    }
    B[right] = best;
    V[right] = min;
}
```

DP – Reconstruct Solution

```
    // reconstruct solution
    unsigned int right=n;
    while (right != 0){
        unsigned int left  = B[right];
         fill (result,s.mean(left,right), left ,right);
        right  = left;
    }
    return V[n];

}
```

Levenshtein Distance

```
// D[n,m] = distance between x and y
// D[i,j] = distance between strings x[1..i] and y[1..j]
vector<vector<unsigned>> D(n+1,vector<unsigned>(m+1,0));
for (unsigned j = 0; j <=m; ++j)
  D[0][j] = j;
for (unsigned i = 1; i <= n; ++i){
  D[i][0] = i;
  for (unsigned j = 1; j <=m; ++j){
    unsigned q = D[i-1][j-1] + (x[i-1]!=y[j-1]);
    q = std::min(q,D[i][j-1]+1);
    q = std::min(q,D[i-1][j]+1);
    D[i][j] = q;
  }
}
return D[n][m];
```

Questions?