CS580 Algorithm Design, Analysis, and Implementation

Lecture notes, Feb 01, 2022
Wufei Ma

1 Dynamic Programming

Knapsack problem. Given n objects and a knapsack. Item i weighs $w_i > 0$ kilograms and has value $v_i > 0$. The knapsack has a capacity of W kilograms. The goal is to fill the knapsack so as to maximize total value.

Let $\mathrm{OPT}(i, w)$ be the max profit subset of items $1, \dots, i$ with weight limit w. We have

$$\mathrm{OPT}(i,w) = \begin{cases} 0 & \text{if } i = 0 \\ \mathrm{OPT}(i-1,w) & \text{if } w_i > w \\ \max\{\mathrm{OPT}(i-1,w), v_i + \mathrm{OPT}(i-1,w-w_i)\} & \text{otherwise} \end{cases}$$

This algorithm can be implemented by filling up an $n \times W$ array. The running time is $\Theta(nW)$, which is pseudo-polynomial with respect to the input size.

The decision version of the knapsack problem is NP-complete. Given w_i's and W, is there a combination of w_i's such that the sum of the weights is exactly W?

Matrix chain multiplication. Given matrices $\mathbf{A}_1, \dots, \mathbf{A}_n$ of size $p_0 \times p_1, \dots, p_{n-1} \times p_n$, find a parenthesization that minimizes the number of multiplications.

For instance, let $\mathbf{A}_1$ be a 3×2 matrix, $\mathbf{A}_2$ be a 2×4 matrix, and $\mathbf{A}_3$ be a 4×1 matrix. In this case, $(\mathbf{A}_1\mathbf{A}_2)\mathbf{A}_3$ takes $3 \times 2 \times 4 + 3 \times 4 \times 1 = 36$ mutliplications. However, $\mathbf{A}_1(\mathbf{A}_2\mathbf{A}_3)$ takes only $2 \times 4 \times 1 + 3 \times 2 \times 1 = 14$ multiplications.

Claim. There are $\frac{1}{n}\binom{2n-2}{n-1}$ different parenthesization of n matrices.

A recursive algorithm. Let m_{ij} be the minimal number of multiplications necessary to compute $\mathbf{A}_i \dots \mathbf{A}_j$.

```
1     Function M(i,j):
2         if i == j:
3             return 0
4         else:
5             min = inf
6             for k = i to j-1 do:
7                 multi = M(i,k) + M(k+1,j) + p[i-1]*p[k]*p[j]
8                 if multi < min:
9                     min = multi
10                endif
```

```
11            endfor
12        endif
```

The time complexity of the algorithm is

$$
\begin{aligned}
T(n) &= \sum_{k=1}^{n-1} (T(k) + T(n-k)) + O(1) \\
&= 2 \sum_{k=1}^{n-1} T(k) + O(1) \\
&\geq 2T(n-1) \\
&\geq \sum_{i=1}^{n-1} 2^i \\
&= 2^n - 1
\end{aligned}
$$

The problem of this recursive algorithm is that the same subproblem are computed many times. The solution is to cache the results to the subproblems.

Dynamic prorgramming algorithm. We use a table $\mathbf{M}_{n\times n}$ and $\mathbf{S}_{n\times n}$ to store the optimal solution to subproblems.

```
1    Function MatrixChain(p):
2        for i = 1 to n do:
3            M[i, i] = 0
4        endfor
5        for l = 2 to n do:
6            for i = 1 to n-l+1 do:
7                j = i+l-1
8                M[i, j] = inf
9                for k = i to j-1 do:
10                   multi = M[i,k] + M[k+1,j] + p[i-1]*p[k]*p[j]
11                   if multi < M[i,j]:
12                       M[i,j] = multi
13                       S[i,j] = k
14                   endif
15               endfor
16           endfor
17       endfor
```

To output the optimal solution, we use

```
1    Function PrintParent(S, i, j):
2        if i == j:
3            print(Ai)
4        else:
5            print '('
6            PrintParent(S, i, S[i,j])
```

```
7            PrintParent(S, S[i,j]+1, j)
8            print ')'
9        endif
```

The time complexity is $O(n^3)$ and the space complexity is $O(n^2)$.

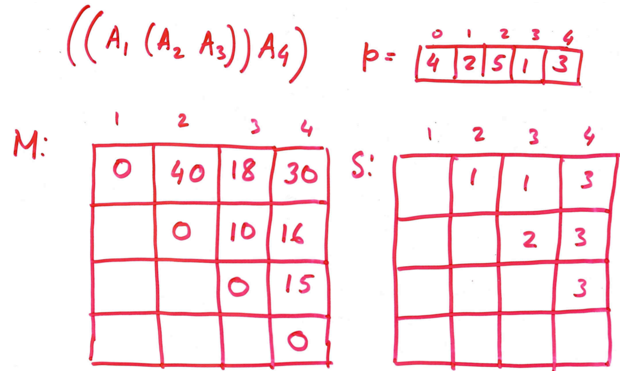

Figure 1: An example.

Polygon triangulation. A polygon can be represented with the counter-clockwise sequence of its vertices.

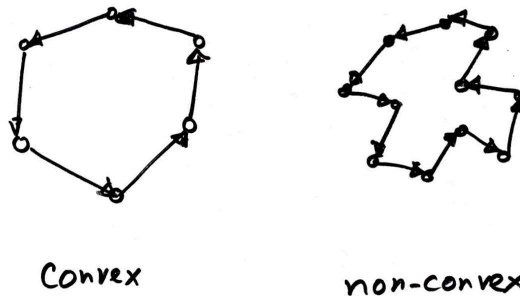

Figure 2: Convex and non-convex polygons.

Left turns. A polygon is convex if and only if any three consecutive points form a left turn. The three points a, b, c is a left turn if and only if

$$\det \begin{bmatrix} ax & ay & a \\ bx & by & 1 \\ cx & cy & 1 \end{bmatrix} > 0$$

Existence of a polygon triangulation. A triangulation is a decomposition of the polygon's interior into triangles whose vertices are the vertices of the polygon. Every polygon can be triangulated, which can be proved by induction. In addition, the number of triangles is $n-2$ and the number of chords is $n-3$.

Constructing the optimal triangulation. Let w_{ij} be a non-negative weight of the a chord connecting node i and j. The goal is to find the triangulation of the polygon with the maximal sum of weights.

Again, we use two arrays $\mathbf{T}_{n-1 \times n-1}$ and $\mathbf{V}_{n-1 \times n-1}$ where $T(i,l)$ stores weights of the best triangulation $P(i, i+l)$ and $V(i,l)$ stores the vertex $i+k$ so that $p_i p_{i+k} p_{i+l}$ is a triangle in the optimal solution.