Dynamic Programming

Rossano Venturini[1]

1 **Computer Science Department, University of Pisa, Italy**
rossano.venturini@unipi.it

Notes for the course *"Competitive Programming and Contests"* at Department of Computer Science, University of Pisa.
Web page: `https://github.com/rossanoventurini/CompetitiveProgramming`

These notes sketch the content of the 16th lecture.

1 Minimum Cost Path

▶ **Problem 1** (Minimum Cost Path). *We are given a matrix M of $n \times m$ integers. The goal is to find the minimum cost path to move from the top-left corner to the bottom-right corner by moving only down or to left.*

Consider the matrix below.

1	2	6	9
0	0	3	1
1	7	7	2

The problem is solved by building a new matrix W. Entry $W[i][j]$ is computed by taking $M[i][j] + \min(W[i-1][j], W[i][j-1])$.

Matrix W for the example above is as follows.

1	3	9	18
1	1	4	5
2	8	11	7

2 Longest Bitonic Subsequence

▶ **Problem 2** (Longest Bitonic Subsequence). *Given a sequence S, the goal is to compute the length of the longest bitonic subsequence.*

A bitonic sequence is a sequence that first increases and then decreases.

Consider for example the sequence $S = 2, -1, 4, 3, 5, -1, 3, 2$. The subsequence $-1, 4, 3, -1$ is bitonic. A longest bitonic subsequence is $2, 3, 5, 3, 2$.

The idea is to compute the longest increasing subsequence from left to right and the longest increasing subsequence from right to left by using the $\Theta(n \log n)$ time solution of the previous lecture.

Then, we combine these two solutions to find the longest bitonic subsequence. This is done by taking the values in a column, adding them and subtracting one. This is correct because $\mathsf{LIS}[i]$ computes the longest increasing subsequence of $S[1, i]$ and ends with $S[i]$.

S	2	-1	4	3	5	-1	3	2
LIS	1	1	2	2	3	1	2	2
LDS	2	1	3	2	3	1	2	1
LBS	2	1	4	3	5	1	3	2

3 Subset sum

▶ **Problem 3** (Subset sum). *Given a set S of n non-negative integers, and a value v, determine if there is a subset of the given set with sum equal to given v.*

The problem has a solution which is almost the same as 0/1 knapsack problem.

As in the 0/1 knapsack problem, we construct a matrix W with $n+1$ rows and $v+1$ columns. Here the matrix contains booleans.

Entry $W[i][j]$ is true if and only if there exists a subset of the first i items with sum j.

The entries of the first row $W[0][]$ are set to false while entries of the first column $W[][0]$ are set to true.

Entry $W[i][j]$ is true either if $W[i-1][j]$ is true or $W[i-1][j-S[i]]$ is true.

As an example, consider the set $S = \{3, 2, 5, 1\}$ and value $v = 6$. Below the matrix W for this example.

	0	1	2	3	4	5	6
0	T	F	F	F	F	F	F
3	T	F	F	T	F	F	F
2	T	F	T	T	F	T	F
5	T	F	T	T	F	T	F
1	T	T	T	T	T	T	T

4 Coin change

▶ **Problem 4** (Coin change). *We have m types of coins available in infinite quantities where the value of each coin is given in the array $C = [c_1, c_2, \ldots, c_m]$. The goal is find out how many ways we can make the change of the amount K using the coins given.*

For example, if $C = [1, 2, 3, 8]$, there are 3 ways to make $K = 3$, i.e., $1, 1, 1$, $1, 2$ and 3.

The solution is similar to the one for subset sum.

The goal is to build a $(m+1) \times (K+1)$ matrix W, i.e., we compute the number of ways to change any amount smaller than or equal to K by using coins in any prefix of C.

The easy cases are when $K = 0$. We have just one way to change this amount. Thus, the first column of W contains 1 but $W[0, 0] = 0$ which is the number of ways to change 0 with no coin.

Now, for every coin we have an option to include it in solution or exclude it.

If we decide to include the ith coin, we reduce the amount by coin value and use the sub problem solution $(K - c[i])$.

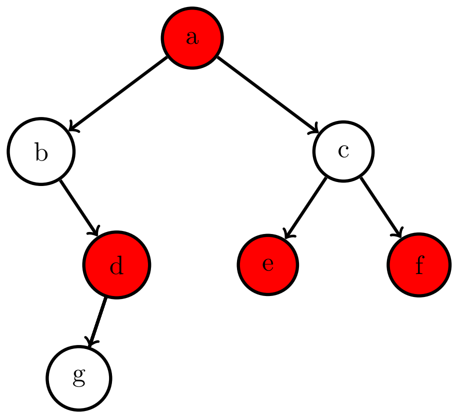
If we decide to exclude the ith coin, the solution for the same amount without considering that coin is on the entry above.

5 Largest independent set on trees

▶ **Problem 5** (Largest independent sets on trees). *Given a tree T, find one of its largest independent sets.*

An independent set is a set of nodes I such that there is no edge connecting a pair of nodes in I.

Example below shows a tree whose largest independent set consists of the red nodes a, d, e, and f.



In general the largest independent set is not unique. For example, we can obtain a different largest independent set by replacing nodes a and d with nodes b and g.

Consider a bottom up traversal of the tree T. For any node u, we have two possibilities: either add or not add the node u to the independent set.

In the former case, u's children cannot be part of the independent set but its grandchildren could. In the latter case, u's children could be part of the independent set.

Let $\mathsf{LIST}(u)$ be the size of the independent set of the subtree rooted at u and let C_u and G_u be the set of children and the set of grandchildren of u, respectively.

Thus, we have the following recurrence.

$$\mathsf{LIST}(u) = \begin{cases} 1 & \text{if u is a leaf} \\ \max(1 + \sum_{v \in G_u} \mathsf{LIST}(v), \sum_{v \in C_u} \mathsf{LIST}(v)) & \text{otherwise} \end{cases}$$

The problem is, thus, solved with a post-order visit of T in linear time.

Observe that the same problem on general graphs is NP-Hard.