Dynamic programming (1/2)

Prof. Rhadamés Carmona

Last revision: 08/01/2020

Content

- What is DP
- Algorithm design
- Fibonacci
- Combinations
- Max Sum 1D
- Max Sum 2D
- Coin change

What is DP (dynamic programming)

- Method developed by Richard Bellman in the 1950s
- It is a technique for solving optimization problems. A n-size problem is solved by combining optimal solutions for smaller-sized problems.
- Unlike divide and conquer, we have optimal and overlapping substructures ➔ The same subproblem may occur more than once.

What is DP (dynamic programming)

- Due to the overlapping of subproblems, solutions to these subproblems are stored; thus, they are not calculated more than once.
- The principle of optimality means that the optimal solution is obtained by combining some of the optimal solutions in its sub-problems.
- Top-down and bottom-up are often used.

What is DP (dynamic programming)

- Bottom-up. We start by solving trivial problems (e.g. n=0 or n=1); then, we combine these solutions to obtain larger solutions.
- Top-down. It's recursive. The problem is divided into sub-problems and then each subproblem is resolved. Optimal solutions to subproblems are memorized to avoid recalculation (due to overlapping).

Algorithm design

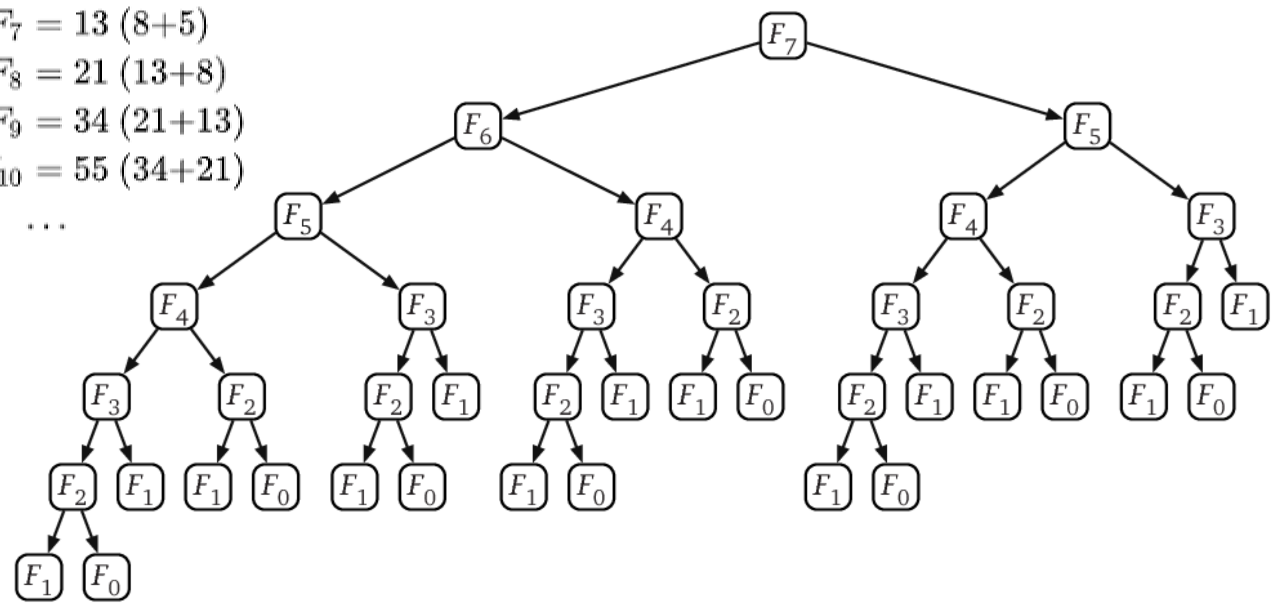
- Verify that the solution can be achieved on the basis of a succession of decisions and that it complies with the principle of optimality.
- Find a recursive expression for the solution.
- Use the recursive expression to populate the partial solutions table until you find the optimal solution to the problem.
- Rebuild the solution by finding on the table the path that has led us to the optimal solution.

Fibonacci

This expression is top-down. If we write it algorithmically without memorizing, this may take too long for small values of n (e.g. 50)

$$Fib(n) = \begin{cases} Fib(n-1) + Fib(n-2) & \text{si } n > 1 \\ n & \text{si } n = 0 \text{ o } n = 1 \end{cases}$$

$$\begin{aligned}
F_2 &= F_1 + F_0 = 1 + 0 = 1 \\
F_3 &= F_2 + F_1 = 1 + 1 = 2 \\
F_4 &= F_3 + F_2 = 2 + 1 = 3 \\
F_5 &= F_4 + F_3 = 3 + 2 = 5 \\
F_6 &= F_5 + F_4 = 5 + 3 = 8 \\
F_7 &= 13\ (8{+}5) \\
F_8 &= 21\ (13{+}8) \\
F_9 &= 34\ (21{+}13) \\
F_{10} &= 55\ (34{+}21) \\
&\ldots
\end{aligned}$$



Fibonacci

```
#define N 50
int f[N];

int fibTD(int n)                  // top-down versión; it is constant time if computed before
{                                 // O(n) in storage
        if (f[n] != -1)           // already computed?
                return f[n];
        f[n] = fibTD(n - 1) + fibTD(n - 2);
        return f[n];
}

int main() {
        memset(f, 0xff, sizeof(f) );      // set all cells to -1 (very fast way…)
        f[0] = 0; f[1] = 1;               // base cases
        while (true) {
                int n;  cin >> n;
                cout << fibTD(i) << endl;
        }
        return 0;
}
```

Fibonacci

```
#define N 50
int f[N];
int fibBU1(int n) {    // buttom up version: O(n) time and O(n) space
        if (n<=1)
                return n;
        for (int i=2; i<=n; i++)
                f[i] = f[i-2] + f[i-1];
        return f[n];
}

int fibBU2(int n) {    // buttom up optimized version: O(n) time and O(1) space
        if (n<=1)
                return n;
        int  a = 0, b = 1;              // fi-2 and fi-1 (mem for 2 values only)
        for (int i=2; i<=n; i++) {
                int fi = a+b;
                a = b; b = fi;          // updating fi-2 and fi-1 for next iteration
        }
        return b;
}
```

Combinations C(n,m)

$$C(n,m) = \binom{n}{m} = \frac{n!}{m!(n-m)!}$$

Numerically unstable

Base cases

$$\binom{n}{m} = \begin{cases} 1 & m = 0 \\ 1 & n = m \\ \binom{n-1}{m-1} + \binom{n-1}{m} & 1 < m < n \end{cases}$$

Combinations C(n,m)

Bottom-up Solution - Pascal's Triangle
Newton's binomial coefficients $(a+b)^n$.
Note that $C(n,m)=C(n-1,m-1)+C(n-1,m)$

m	0	1	2	3	4	5	6
n=0	1						
n=1	1	1					
n=2	1	2	1				
n=3	1			1			
n=4	1				1		
n=5	1					1	
n=6	1						1

m	0	1	2	3	4	5	6
n=0	1						
n=1	1	1					
n=2	1	2	1				
n=3	1	3	3	1			
n=4	1				1		
n=5	1					1	
n=6	1						1

Combinations C(n,m)

Bottom-up Solution - Pascal's Triangle
Newton's binomial coefficients $(a+b)^n$.
Note that C(n,m)=C(n-1,m-1)+C(n-1,m)

```
int comb(int n, int m){
  int A[n+1][m+1];
  A[0][0] = 1;
  for (int i = 1; i <= n; i++){
     A[i][0] = A[i][i] = 1;
     for (int j = 1; j < i; j++)
        A[i][j] = A[i-1][j-1] + A[i-1][j];
  }
  return A[n][m];
}
```

m	0	1	2	3	4	5	6
n=0	1						
n=1	1	1					
n=2	1	2	1				
n=3	1	3	3	1			
n=4	1				1		
n=5	1					1	
n=6	1						1

Calculates some combinations that will not be used. But it's iterative. Trade-off...
The other detail is symmetry. So you could calculate only half (m- 0... n/2)

Combinations C(n,m)

Bottom-up Solution - Pascal's Triangle
Newton's binomial coefficients $(a+b)^n$.
Note that $C(n,m)=C(n-1,m-1)+C(n-1,m)$

```
int comb(int n, int m) {
   int A[n];// 1D array, O(n) storage
   for (int i = 0; i <= n; i++)    {
         A[0] = A[i] = 1; // base cases
         int prev = A[0];
         for (int j = 1; j < i; j++) {
                  int prevAj= A[j];
                  A[j] = prev + prevAj;
                  prev = prevAj;
         }
   }
   return A[m];
}
```

m	0	1	2	3	4	5	6
n=0	1						
n=1	1	1					
n=2	1	2	1				
n=3	1	3	3	1			
n=4	1				1		
n=5	1					1	
n=6	1						1

Combinations C(n,m)

Top-down solution

C(5,2) = C(4,1) + C(4,2)
= C(3,0) + **C(3,1) + C(3,1)** + C(3,2)
= C(3,0) + C(2,0) **+ C(2,1)** + C(2,0) + **C(2,1) + C(2,1)** + C(2,2)
= C(3,0) + C(2,0) + C(1,0) + C(1,1) + C(2,0) + C(1,0) + C(1,1) + C(1,0) + C(1,1) + C(2,2)
= 10

Notice that C(3,1) is calculated 2 times (overlapping)
C(2,1) is calculated 3 times...
We need a buffer (the same array as for buttom-up)
to store "undiscovered" combinations
Thus, they are not calculated more than once

Combinations C(n,m)

```
                    //top-down solution
int memo[30][30];
int combTD(int n, int m) {
  if (memo[n][m] > 0)
      return memo[n][m];
  if (n == m || m == 0) {
      memo[n][m] = 1;
      return 1;
  }
  memo[n][m] = combTD(n - 1, m - 1) + combTD(n - 1, m);
  return memo[n][m];
}
…
memset(memo, 0, sizeof(memo));
printf("C(6,3) = %d\n", combTD(6, 3));
// if you ask again for combTD(6, 3), or combTD(5,2),
// it is just an access to memo ➔ constant time, O(1)
```

Maximum sum

- Given a sequence of integers a_1,... a_n, what is the subsequence that adds up the most?
- The "naive" solution is to do 3 loops ($O(n^3)$); and for every possible pair (i, j), find the sum $a_i + a_{i+1} + \ldots + a_j$, and compare against the maximum found
- If all values are positive, the solution is trivial. Just add all of them
- This problem is 1D. For each i check if it is better to add a_i to the sequence, or start another one. Memo(i) = max(a_i + Memo(i-1), a_i). It is 1D **Kadane's algorithm**

Maximum sum

- Bottom-up solution (start from base case(s))
- Base case: Memo[0] = A[0]
- Common case: Memo[i] = max(A[i] + Memo[i-1], A[i])

	0	1	2	3	4	5	6	7	8
A	1	-2	4	-3	2	1	5	-2	-1
Memo	1								

|---------------------------------Fill the table---------------------------------------|

Maximum sum

- Memo[0] = A[0]
- Memo[i] = max(A[i] + Memo[i-1], A[i])
- The maximum sum can be found in any position
- How to rebuild the winning sequence?
- If memo[k] is the maximum. Add all elements A[k]+A[k-1]+... until they add up memo[k].

	0	1	2	3	4	5	6	7	8
A	1	-2	4	-3	2	1	5	-2	-1
Memo	1								

|----------------------------------Fill the table--|

Maximum sum

- Memo[0] = A[0]
- Memo[i] = max(A[i] + Memo[i-1], A[i])
- The maximum sum can be found in any position
- How to rebuild the winning sequence?
- If memo[k] is the maximum. Add all elements A[k]+A[k-1]+... until they add up memo[k].

	0	1	2	3	4	5	6	7	8
A	1	-2	4	-3	2	1	5	-2	-1
Memo	1	-1	4	1	3	4	**9**	7	6
							K		

Maximum sum

- Memo[0] = A[0]
- Memo[i] = max(A[i] + Memo[i-1], A[i])
- The maximum sum can be found in any position
- How to rebuild the winning sequence?
- If memo[k] is the maximum. Add all elements A[k]+A[k-1]+... until they add up memo[k].

	0	1	2	3	4	5	6	7	8
A	1	-2	4	-3	2	1	5	-2	-1
Memo	1	-1	4	1	3	4	**9**	7	6
							K		

Maximum sum

- Complexity:
  - Extra storage of O(n) for memo
  - O(n) to fill memo
  - O(n) to get the winning sequence
  - Total O(n), much better than O(n^3) of naïve approach

	0	1	2	3	4	5	6	7	8
A	1	-2	4	-3	2	1	5	-2	-1
Memo	1	-1	4	1	3	4	**9**	7	6
							K		

Maximum 2D sum

- The "naive" version varies all pairs (i,j) (k,l), $O(n^4)$, and for each pair, do the sum between them, $O(n^2)$, which solves to $O(n^6)$.
- A better version (but not the best) first finds the accumulated function in $O(n^2)$:

$$acum[i, j] = \sum_{I=0}^{i} \sum_{J=0}^{j} a[I, J]$$

- Then, $O(n^4)$: the sum of pairs between(i,j) and (k,l) can be obtained in constant time with a simple formula

$$suma(i, j, k, l) = acum[k, l] - acum[i-1, l] - acum[k, j-1] + acum[i-1, j-1]$$

Maximum 2D sum

- Another DP version uses 2D **Kadane's algorithm**, $O(n^3)$.
- First, create **acum** matrix in $O(n^2)$.
- Then, for each pair j_0,j_1 ($j_0<=j_1$) of columns (in total $O(n^2)$ pairs), obtain a 1D vector B, such that
  $B[i] = A[i, j_0]+ \ldots + A[i, j_1]$.
- B can be filled by using acum matrix in $O(n)$:
  $B[i] = acum[i,j_1]-acum[i-1,j_1]-acum[i,j_0-1]+acum[i-1,j_0-1]$.
- Perform $O(n)$ 1D max sum on B, obtaining the rows sub range $i_0..i_1$ of maximum sum for a given column pair j_0,j_1.
- Keep track of the best indexes combination found.

Coin change

- This problem can be solved with backtracking (slow), with a greedy algorithm (fast but not always optimal), and finally with DP (fast and optimal)
- Given a monetary cone, return the smallest amount of coins for a given change of j $
- Suppose memo[i,j] contains the number of coins for a change of j $'s
- Suppose that the values of the monetary cone are value[0], value[1], ... , value[i]. For example, value = {1,2,5,10,20,50,100, …)

Coin change

- Let’s define **int memo[m][v+1];** m represents the range of different coins used for the solution (with values value[0], … value[m-1]), and v is the change value (an integer number, for example 27). This, memo[i,j] is the minimum amount of coins, for a change of j dollars, using the first i+1 types of coins.
- Let’s define **int value[m]**, value[i] stores the value of the i-th coin type . For example, if the monetary cone is 1 5 10 20 50 100, then m=6, and value[2]=10.
- Notice that memo[*,0]=0 (base case), also, memo[0,j] = j **div** value[0]. If the division is inexact, the change is not posible with coins with the value of **value[0]** dollars (set memo[0,j]=INFINITE if division is inexact). But it only happens in very weird countries…lol.

Coin change

- To update memo[i,j] there are 2 possibilities:
  - j < value[i]: the value of the i-th coin type exceeds the change. In this case memo[i,j] = memo[i-1,j], which means, the same change with coins of smaller values
  - j >= value[i]: Take the minimum between not using the coin with value value[i] (memo[i-1,j]) or using it (1+memo[i,j-value[i]])

Coin change

$$memo(i,j) = \begin{cases} 0\ if\ j = 0 \\ j\ div\ value[0] or\ \infty\ if\ i = 0 \\ memo(i-1,j)\ if\ j < value[i] \\ \min(memo(i-1,j), 1 + memo(i, j - value[i])\ if\ j \geq value[i] \end{cases}$$

Example, try a return of 11 in a currency cone 2,3,7

		0	1	2	3	4	5	6	7	8	9	10	11
2	0	0	999	1	999	**2**	999	3	999	4	999	5	999
3	1	0	999	1	1	2	2	2	3	3	3	4	**4**
7	2	0	999	1	1	2	2	2	1	3	2	2	**3**

3 coins: one of 7, two out of 2

Coin change

$$memo(i,j) = \begin{cases} 0 \; if \; j = 0 \\ j \; div \; value[0] or \; \infty \; if \; i = 0 \\ memo(i-1,j) \; if \; j < value[i] \\ \min(memo(i-1,j), 1 + memo(i, j - value[i]) \; if \; j \geq value[i] \end{cases}$$

Example, try a return of 18 in a currency cone 1,2,5

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	0	0	**1**	2	**3**	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
2	1	0	**1**	1	**2**	2	3	3	4	**4**	5	5	6	6	**7**	7	8	8	9	**9**
5	2	0	1	1	**2**	2	1	2	2	**3**	3	2	3	3	**4**	4	3	4	4	**5**

5 coins: three out of 5, one of 2, one of 1

Coin change

```
int buildTable(int change, int value[], int n)
{
        for (int i = 0; i < n; i++)
                memo[i][0] = 0;
        for (int j = 0; j <= change; j++)
                memo[0][j] = (j % value[0]) ? INFINITE: j / value[0];
                //memo[0][j] = j / value[0]; for serious countries
        for (int j = 1; j <= change; j++) for (int i = 1; i <n; i++)
          if (j < value[i])  // change < coin; do not use this coin
            memo[i][j] = memo[i - 1][j];
          else   // use the coin, or might be not
            memo[i][j] = std::min(memo[i - 1][j], memo[i][j-value[i]] + 1);
        return memo[n-1][change];    // #coins used
}
```

Coin change

- To rebuild the solution, we start with memo[m][c], with m=size(value)-1, c=change.
- The m-th coin value[m] is selected if memo[m][c]!=memo[m-1][c]. We then continue with c=c-value[m].
- It is not selected if they are the same, in which case we follow after making m=m-1.
- We do this until m=0, in which case we take those lower denomination coins.
- If the exact change is not possible, the result is INFINITE.

Coin change

```
vector<int> generateChange(int change, int value[], int n) {
        int m = n - 1;
        int c = change;
        vector<int> coinsToPay(n);
        for (int i = 0; i < n; i++)
                coinsToPay[i] = 0;
        while (m != 0) {
          if (memo[m][c] == memo[m - 1][c])     // m-th coin type was not used
                m--;                            // check lower values
          else  {
                coinsToPay[m]++;                // m-th coin type was used
                c = c - value[m];               // substract the coin value
          }
        }
        coinsToPay[0] = memo[0][c];   // monedas de menor den. que cubren c
        return coinsToPay;
}
```

Problems (all easy, do all if possible)

- COINS - Bytelandian gold coins
- https://www.spoj.com/problems/COINS/
- NOCHANGE - No Change
- https://www.spoj.com/problems/NOCHANGE/
- MAXSUMSU - Maximum Subset Sum
- https://www.spoj.com/problems/MAXSUMSU/
- MAXSUMSQ - Maximum Sum Sequences
- https://www.spoj.com/problems/MAXSUMSQ/

Problems (average ones)

- ACTIV - Activities
- https://www.spoj.com/problems/ACTIV/
- BORW - Black or White
- https://www.spoj.com/problems/BORW/
- ANARC09A - Seinfeld
- https://www.spoj.com/problems/ANARC09A/

Problems (harder)

- Next week….