VE281

Data Structures and Algorithms

Dynamic Programming

Announcement

- Written assignment 6 released
  - On graph algorithms and dynamic programming
  - Due by 5:40 pm on Dec. 8th

Outline

- Summary of Dynamic Programming

- Another Example: Longest Common Subsequence (LCS)

Dynamic Programming for Optimization

- There are two key ingredients that an optimization problem must have in order for dynamic programming to apply:
  - Optimal substructure;
  - Overlapping subproblems.

Optimal Substructure

- An optimal solution to the problem contains **within** it **optimal solutions to subproblems**.
  - In matrix-chain multiplication, the optimal order on calculating $A_i \times \cdots \times A_j$ that splits the product between A_k and A_{k+1} contains within it optimal solutions to the problem of ordering $A_i \times \cdots \times A_k$ and $A_{k+1} \times \cdots \times A_j$.

- You can show optimal substructure property by supposing that each of the subproblem solutions is not optimal and then deriving a contradiction.

Overlapping Subproblems

- A recursive algorithm for the problem solves the same subproblems **over and over**, rather than always generating new subproblems.
  - E.g., subproblems of matrix-chain multiplication overlap.
  - In contrast, a problem for which a divide-and-conquer approach is suitable usually generates **brand-new** problems at each step of the recursion.
- Dynamic-programming algorithms take advantage of overlapping subproblems by
  - solving each subproblem once …
  - … and then storing the solution in a table where it can be looked up when needed.

Designing a Dynamic-Programming Algorithm

1. Characterize **the structure** of an optimal solution.
   - Usually, we need to define a **general** problem.
2. **Recursively** define the value of an optimal solution.
3. Compute the value of an optimal solution, typically in a **bottom-up** fashion.
4. Construct an optimal solution from computed information.

Memoization

- In dynamic programming, solutions to subproblems are pre-computed and stored in a table.
  - A **bottom-up** approach.
- An alternative approach is to "**memoize**" during the recursion.
  - A **top-down** approach. Start from the largest subproblem.
  - When a subproblem is encountered **first time** during recursion, its solution is computed and then stored in a table…
  - …each subsequent time that we **encounter this subproblem again**, we simply look up the value stored in the table and return it.

Outline

- Summary of Dynamic Programming

- Another Example: Longest Common Subsequence (LCS)

Longest Common Subsequences

Terminology

- Consider sequences of symbols, such as X = <A, B, A, C>.
- **Subsequence**: derived from another sequence by **deleting** some elements **without changing** the order of the remaining elements.

- Example:
  - Sequence <A, B, D> is a subsequence of <A, C, B, C, B, D>

- Exercise:
  - Is sequence <C, B, D> a subsequence of <B, A, C, A, B, D>?
  - Is sequence <B, A, C> a subsequence of <B, D, B, C, A, B>?

Longest Common Subsequences

Problem

- Given: two sequences X and Y.
- Output: a sequence that is a **longest common subsequence (LCS)** of both X and Y.

- Example: X = <A, B, C, B, D, A, B> and Y = <B, D, C, A, B, A>
  - <B>, <B, C, B>, <B, C, A, B>, <B, D, A, B> are **common subsequences** of X and Y.
  - <B, C, A, B> is a LCS.
  - <B, D, A, B> is another LCS.

How to find LCS?

By Dynamic Programming!

Recap: Designing a Dynamic-Programming Algorithm

1. Characterize **the structure** of an optimal solution.
   - Usually, we need to define a **general** problem.
2. **Recursively** define the value of an optimal solution.
3. Compute the value of an optimal solution, typically in a **bottom-up** fashion.
4. Construct an optimal solution from computed information.

Recap: Designing a Dynamic-Programming Algorithm

1. Characterize **the structure** of an optimal solution.
   - Usually, we need to define a **general** problem.
2. **Recursively** define the value of an optimal solution.
3. Compute the value of an optimal solution, typically in a **bottom-up** fashion.
4. Construct an optimal solution from computed information.

Longest Common Subsequences

Optimal Structure

- Suppose the sequence X is $X[1..n]$ and the sequence Y is $Y[1..m]$.
- Define general problem Q_{ij}: find the longest common subsequences of $X[1..i]$ and $Y[1..j]$.
  - Define $c(i,j)$ to be the length of the LCS of $X[1..i]$ and $Y[1..j]$.
  - We ultimately want to solve Q_{nm}.

- To solve Q_{ij}, we can **recursively** solve subproblems of smaller size.

Longest Common Subsequences

Recursion

- If the **last** symbols are same, i.e., $X[i] = Y[j]$, then …
  - the **last** symbol of LCS of $X[1..i]$ and $Y[1..j]$ is $X[i]$.
  - the LCS of $X[1..i]$ and $Y[1..j]$ is
    the LCS of $X[1..(i-1)]$ and $Y[1..(j-1)] + X[i]$

- Example: X = <A, B, A, **C**>, Y = <B, C, D, **C**>
  - LCS(X, Y) = LCS(<A, B, A>, <B, C, D>) + **C**

Longest Common Subsequences

Recursion

- If the **last** symbols are not same, i.e., $X[i] \neq Y[j]$, then LCS of $X[1..i]$ and $Y[1..j]$ is …
  - **either** LCS of $X[1..(i-1)]$ and $Y[1..j]$,
  - **or** LCS of $X[1..i]$ and $Y[1..(j-1)]$.
  - … depending on which one of the above two is **longer**!

- Example: X = <A, B, D, **C**>, Y = <B, C, D, **D**>
  - LCS(X,Y) is the **longer** one of
    LCS(<A, B, D, **C**>, <B, C, D>) and
    LCS(<A, B, D>. <B, C, D, **D**>).

Longest Common Subsequences
Recursion

- In summary, we have:

$$c(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ c(i-1, j-1) + 1 & \text{if } i,j > 0 \text{ and } X[i] = Y[j] \\ \max\{c(i-1,j), c(i,j-1)\} & \text{if } i,j > 0 \text{ and } X[i] \neq Y[j] \end{cases}$$

- The straightforward recursive algorithm has exponential time complexity. However, the total number of different subproblems is not exponential.
  - They are Q_{ij}, for $0 \leq i \leq n, 0 \leq j \leq m$.
  - The total number is $(n+1)(m+1)$.
- We use a tabular, bottom-up approach.

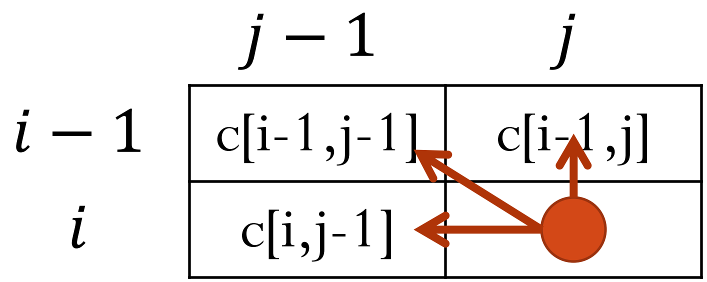
Recap: Designing a Dynamic-Programming Algorithm

1. Characterize **the structure** of an optimal solution.
   - Usually, we need to define a **general** problem.
2. **Recursively** define the value of an optimal solution.
3. Compute the value of an optimal solution, typically in a **bottom-up** fashion.
4. Construct an optimal solution from computed information.

Longest Common Subsequences

Algorithm

```
int LCS(X[1..n], Y[1..m]) {
   for i=0 to n
      c(i,0)=0;
   for j=1 to m
      c(0,j)=0;
   for i=1 to n
      for j=1 to m
         if X[i]==Y[j]
            c[i,j]=c[i-1,j-1]+1;
         else
            c[i,j]=max(c[i-1,j],c[i,j-1]);
   return c[n,m];
}
```



Longest Common Subsequences

Example

- X = <A, B, C, B> and Y = <B, D, C, A, B, A>

		Y:	B	D	C	A	B	A
		0	1	2	3	4	5	6
X:	0	0	0	0	0	0	0	0
A	1	0						
B	2	0						
C	3	0						
B	4	0						

Longest Common Subsequences
Example

- X = <A, B, C, B> and Y = <B, D, C, A, B, A>

	Y:		B	D	C	A	B	A
		0	1	2	3	4	5	6
X:	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0						
C	3	0						
B	4	0						

Longest Common Subsequences

Example

- X = <A, B, C, B> and Y = <B, D, C, A, B, A>

		Y:	B	D	C	A	B	A
		0	1	2	3	4	5	6
X:	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0						
B	4	0						

Longest Common Subsequences
Example

- X = <A, B, C, B> and Y = <B, D, C, A, B, A>

		Y:	B	D	C	A	B	A
		0	1	2	3	4	5	6
X:	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2
B	4	0						

Longest Common Subsequences

Example

- X = <A, B, C, B> and Y = <B, D, C, A, B, A>

		Y:	B	D	C	A	B	A
		0	1	2	3	4	5	6
X:	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2
B	4	0	1	1	2	2	3	3

Recap: Designing a Dynamic-Programming Algorithm

1. Characterize **the structure** of an optimal solution.
   - Usually, we need to define a **general** problem.
2. **Recursively** define the value of an optimal solution.
3. Compute the value of an optimal solution, typically in a **bottom-up** fashion.
4. Construct an optimal solution from computed information.

Question: how to obtain a LCS for X and Y, **not just** $c[n, m]$?

Longest Common Subsequences

Example

- X = <A, B, C, B> and Y = <B, D, C, A, B, A>

		Y:	B	D	C	A	B	A
		0	1	2	3	4	5	6
X:	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2
B	4	0	1	1	2	2	3	3

Question: how to obtain a LCS for X and Y?
Hint: from c[i,j] table.