

Dynamic Programming DP

Pham Quang Dung and Do Phan Thuan

Computer Science Department, SoICT,
Hanoi University of Science and Technology.

August 15, 2016

- Similar to Divide-and-Conquer
- Subproblems are not independent
- Solve each subproblem once and store solution in to a table for further use
- Divide-and-Conquer solves a problem in a top-down fashion while DP does it in a bottom-up fashion
- DP is dedicated for optimization

Generic Schema - 3 steps

- Subproblem division: identify the structures of subproblems
 - ▶ Smallest subproblems can be solved in a direct way
 - ▶ Easy to combine solutions to subproblems
- Storing solutions to subproblems: avoid repeating the resolution of the same subproblems
- Combination
 - ▶ Bottom-up
 - ▶ Establish the solution to a problem from solutions to its subproblems

- For achieving the efficiency
 - ▶ Number of subproblems must be bounded by a polynomial of the size of the input
 - ▶ Subproblems must be solved to optimality

Largest SubArray

- Given an array of numbers: $A = \langle a_1, \dots, a_n \rangle$
- A subarray of is $A[i, j] = \langle a_i, \dots, a_j \rangle$ with weight $w(A[i, j]) = \sum_{k=i}^j a_k$
- Find the subarray of A having largest weight

Example

- sequence: -2, 11, -4, 13, -5, 2
- The largest weight subsequence is 11, -4, 13 having weight 20

Largest SubArray

- S_i is the weight of the largest subarray terminating at a_i (the last element of the subarray is a_i)
- $S_1 = a_1$
- For each $i > 1$:

$$S_i = \begin{cases} a_i & , \text{ if } S_{i-1} < 0 \\ S_{i-1} + a_i & , \text{ otherwise} \end{cases}$$

- Optimal objective value is $\max_{i \in \{1, \dots, n\}} \{S_i\}$

Maximum Weight Independent Set in a Tree

- Given a rooted tree $T = (V, E)$
 - ▶ r is the root
 - ▶ each node $v \in V$
 - ★ $w(v)$: weight of v
 - ★ $f(v)$: father of v , $f(r) = \text{null}$ by convention
 - ★ $T(v)$: subtree of T rooted at v
 - ★ $Children(v)$: set of children of v
- An independent set of T is a set $S \subseteq V$ such that v and $f(v)$ cannot be both in S , $\forall v \in V \setminus \{r\}$
- Find an independent set of T having the largest total weight

Maximum Weight Independent Set in a Tree

- Let $S(v)$ be the weight of the biggest independent set of $T(v)$, $\forall v \in V$
- Let $\bar{S}(v)$ be the weight of the biggest independent set of $T(v) \setminus \{v\}$ (donot consider v)
- $\bar{S}(v) = \sum_{x \in \text{Children}(v)} S(x)$, $\forall v \in V$
- $S(v) = \max\{\bar{S}(v), w(v) + \sum_{x \in \text{Children}(v)} \bar{S}(x)\}$, $\forall v \in V$
- If v is a leaf, then $S(v) = w(v)$ and $\bar{S}(v) = 0$

Maximum Weight Independent Set in a Tree

Algorithm 1: MaxIndependentSetOnTree($T = (V, E)$)

```
Q  $\leftarrow$   $\emptyset$ ;  
foreach  $v \in V$  do  
     $\text{deg}(v) \leftarrow \# \text{Children}(v)$ ;  
    if  $\text{deg}(v) = 0$  then  
        Enqueue( $v, Q$ );  
         $S(v) \leftarrow w(v)$ ;  
         $\bar{S}(v) \leftarrow 0$ ;  
while Q  $\neq \emptyset$  do  
     $v \leftarrow \text{Dequeue}(Q)$ ;  
     $T \leftarrow w(v) + \sum_{x \in \text{Children}(v)} \bar{S}(x)$ ;  
     $\bar{T} \leftarrow \sum_{x \in \text{Children}(v)} S(x)$ ;  
    if  $T > \bar{T}$  then  
         $S(v) \leftarrow T$ ;  
         $\text{sel}(v) \leftarrow \text{true}$ ;  
    else  
         $S(v) \leftarrow \bar{T}$ ;  
         $\text{sel}(v) \leftarrow \text{false}$ ;  
     $\bar{S}(v) \leftarrow \bar{T}$ ;  
     $u \leftarrow \text{parent}(v)$ ;  
     $\text{deg}(u) \leftarrow \text{deg}(u) - 1$ ;  
    if  $\text{deg}(u) = 0$  then  
        Enqueue( $u, Q$ );
```

Algorithm 2: printSol(v)

```
if  $sel(v) = true$  then
    print( $v$ );
    foreach  $x \in Children(v)$  do
        | printSolExclude( $x$ );
else
    foreach  $x \in Children(v)$  do
        | printSol( $x$ );
```

Algorithm 3: printSolExclude(v)

```
foreach  $x \in Children(v)$  do
    | printSol( $x$ );
```

Longest Common Sequence



- Let $X = \langle x_1, \dots, x_n \rangle$ be a sequence, a subsequence of X is generated by removing some elements from X
- The length of a sequence is the number of elements
- Problem: Given two sequence $X = \langle x_1, \dots, x_n \rangle$ and $Y = \langle y_1, \dots, y_m \rangle$, find the longest common subsequence of X and Y

Longest common subsequence

- $S(i, j)$ is the longest subsequence of $\langle x_1, \dots, x_i \rangle$ and $\langle y_1, \dots, y_j \rangle$,
 $\forall 0 \leq i \leq n, 0 \leq j \leq m$
- $S(0, j) = 0, \forall 0 \leq j \leq m$
- $S(i, 0) = 0, \forall 0 \leq i \leq n$
- for each $i > 0, j > 0$:

$$S(i, j) = \begin{cases} S(i-1, j-1) + 1, & \text{if } x_i = y_j \\ \max\{S(i-1, j), S(i, j-1)\}, & \text{otherwise} \end{cases}$$

- Optimal objective value is $S(n, m)$

Longest common subsequence

Algorithm 4: LCS(X, Y)

Input: Sequences $X = \langle x_1, \dots, x_n \rangle$ and $Y = \langle y_1, \dots, y_m \rangle$

Output: Length of the longest common subsequence of x and y

foreach $j = 0, \dots, m$ **do**

$S(0, j) \leftarrow 0$;

foreach $i = 0, \dots, n$ **do**

$S(i, 0) \leftarrow 0$;

foreach $i = 1, \dots, n$ **do**

foreach $j = 1, \dots, m$ **do**

if $x_i = y_j$ **then**

$S(i, j) \leftarrow S(i - 1, j - 1) + 1$;

else

$S(i, j) \leftarrow \max\{S(i - 1, j), S(i, j - 1)\}$;

return $S(n, m)$;

Edit-Distance Problem

- Input: two strings $X = \langle x_1, \dots, x_n \rangle$ and $Y = \langle y_1, \dots, y_m \rangle$
- 3 operations on X
 - ▶ Insert a character after the position i
 - ▶ Delete a character at position i
 - ▶ Replace a character by another
- Find a sequence of operations of smallest length that make X become Y (distance of X and Y)

Edit-Distance Problem

- For each $0 \leq i \leq n$ and $0 \leq j \leq m$, $d(i, j)$ is the distance of string $\langle x_1, \dots, x_i \rangle$ and $\langle y_1, \dots, y_j \rangle$
- $d(0, 0) = 0$
- $d(0, j) = j, \forall j = 1, \dots, m$ and $d(i, 0) = i, \forall i = 1, \dots, n$
- $d(i, j) = \min\{d(i-1, j-1) + \delta(i, j), d(i-1, j) + 1, d(i, j-1) + 1\}$
where

$$\delta(i, j) = \begin{cases} 0 & , \text{ if } x_i = y_j \\ 1 & , \text{ otherwise} \end{cases}$$

Algorithm 5: EditDistance(X, Y)

Input: Sequences $X = \langle x_1, \dots, x_n \rangle$ and $Y = \langle y_1, \dots, y_m \rangle$

Output: The minimal number of operations to make X become Y

foreach $j = 1, \dots, m$ **do**

$d(0, j) \leftarrow j$;

foreach $i = 1, \dots, n$ **do**

$d(i, 0) \leftarrow i$;

$d(0, 0) \leftarrow 0$;

foreach $i = 1, \dots, n$ **do**

foreach $j = 1, \dots, m$ **do**

$\delta \leftarrow 1$;

if $x_i = y_j$ **then**

$\delta \leftarrow 0$;

$d(i, j) = \max\{d(i-1, j-1) + \delta, d(i-1, j) + 1, d(i, j-1) + 1\}$;

return $d(n, m)$;

- Gold
- Nurses
- Maximum Subsequence
- The Tower of Babylon
- Marble Cut
- Communication networks