Dynamic Programming
DP

Pham Quang Dung and Do Phan Thuan

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

November 6, 2017

Generic Schema

- 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

Generic Schema

- 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: $A = \langle a_1, \ldots, a_n \rangle$
- A subarray of is $A[i,j] = \langle a_i, \ldots, a_j \rangle$ 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 $\max_{i \in \{1,\ldots,n\}}\{S_i\}$
- Complexity: $O(n)$

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

Maximum Weight Independent Set in a Tree

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

```
Q ← ∅;
foreach v ∈ V do
    deg(v) ← ♯Children(v);
    if deg(v) = 0 then
        Enqueue(v, Q);
        S(v) ← w(v);
        S̄(v) ← 0;

while Q ≠ ∅ do
    v ← Dequeue(Q);
    T ← w(v) + ∑_{x∈Children(v)} S̄(x);
    T̄ ← ∑_{x∈Children(v)} S(x);
    if T > T̄ then
        S(v) ← T;
        sel(v) ← true;
    else
        S(v) ← T̄;
        sel(v) ← false;
    S̄(v) ← T̄;
    u ← parent(v);
    deg(u) ← deg(u) − 1;
    if deg(u) = 0 then
        Enqueue(u, Q);
```

Maximum Weight Independent Set in a Tree

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 $X = \langle x_1, \ldots, x_n \rangle$ 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 $X = \langle x_1, \ldots, x_n \rangle$ and $Y = \langle y_1, \ldots, y_m \rangle$, find the longest common subsequence of X and Y

Longest common subsequence

- $S(i,j)$ is the longest subsequence of $\langle x_1, \ldots x_i \rangle$ and $\langle y_1, \ldots, y_j \rangle$, $\forall 0 \leq i \leq n, 0 \leq j \leq m$
- $S(0,j) = 0, \forall 0 \leq j \leq m$
- $S(i,0) = 0, \forall 0 \leq i \leq n$
- 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 $S(n,m)$
- Complexity: $O(n \times m)$

Longest common subsequence

Algorithm 4: LCS(X, Y)

Input: Sequences $X = \langle x_1, \ldots, x_n \rangle$ and $Y = \langle y_1, \ldots, y_m \rangle$
Output: Length of the longest common subsequence of x and y

```
foreach j = 0, ..., m do
    S(0, j) ← 0;
foreach i = 0, ..., n do
    S(i, 0) ← 0;
foreach i = 1, ..., n do
    foreach j = 1, ..., m do
        if x_i = y_j then
            S(i, j) ← S(i − 1, j − 1) + 1;
        else
            S(i, j) ← max{S(i − 1, j), S(i, j − 1)};
return S(n, m);
```

Edit-Distance Problem

- Input: two strings $X = \langle x_1, \ldots, x_n \rangle$ and $Y = \langle y_1, \ldots, y_m \rangle$
- 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 $\langle x_1, \ldots, x_i \rangle$ and $\langle y_1, \ldots, y_j \rangle$
- $d(0,0) = 0$
- $d(0,j) = j, \forall j = 1, \ldots, m$ and $d(i,0) = i, \forall i = 1, \ldots, n$
- $d(i,j) = \min\{d(i-1,j-1) + \delta(i,j), d(i-1,j) + 1, d(i,j-1) + 1\}$ where

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

Edit-Distance

Algorithm 5: EditDistance(X, Y)

Input: Sequences $X = \langle x_1, \ldots, x_n \rangle$ and $Y = \langle y_1, \ldots, y_m \rangle$
Output: The minimal number of operations to make X become Y

```
foreach j = 1, ..., m do
    d(0, j) ← j;
foreach i = 1, ..., n do
    d(i, 0) ← i;
d(0, 0) ← 0;
foreach i = 1, ..., n do
    foreach j = 1, ..., m do
        δ ← 1;
        if x_i = y_j then
            δ ← 0;
        d(i, j) = min{d(i − 1, j − 1) + δ, d(i − 1, j) + 1, d(i, j − 1) + 1};
return d(n, m);
```

Exercises

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