Quy hoạch động (Dynamic Programming)

Khoa Công Nghệ Thông Tin,
Trường Đại Học Thủy Lợi.

Ngày 7 tháng 12 năm 2017

Sơ đồ chung

- Tương tự như thuật toán Chia để trị
- Các bài toán con không độc lập
- Giải quyết từng bài toán con và lưu trữ lời giải trong một bảng để sử dụng về sau
- Chia để trị giải quyết một bài toán theo cách tiếp cận từ trên xuống dưới trong khi quy hoạch động giải quyết bài toán theo cách từ dưới lên trên
- Quy hoạch động được dành riêng cho bài toán tối ưu

Sơ đồ chung - 3 bước

- Chia thành bài toán con: xác định cấu trúc của các bài toán con
  - Những bài toán con nhỏ nhất có thể được giải quyết một cách trực tiếp
  - Dễ dàng kết hợp các giải pháp cho các bài toán con
- Lưu trữ lời giải cho các bài toán con: tránh phải giải lại nhiều lần cùng một bài toán con
- Kết hợp
  - Từ dưới lên trên
  - Thiết lập giải pháp cho một bài toán từ các giải pháp của những bài toán con của nó

Sơ đồ chung

- Để đạt được hiệu quả
  - Số lượng bài toán con phải được giới hạn bởi một đa thức của kích thước đầu vào
  - Các bài toán con phải được giải quyết tối ưu

Chuỗi con lớn nhất

- Cho một chuỗi các số: $A = \langle a_1, \ldots, a_n \rangle$
- Một chuỗi con của A là $A[i,j] = \langle a_i, \ldots, a_j \rangle$ có trọng số $w(A[i,j]) = \sum_{k=i}^{j} a_k$
- Tìm chuỗi con của A có trọng số lớn nhất

Example

- Chuỗi: -2, 11, -4, 13, -5, 2
- Chuỗi con trọng số lớn nhất là 11, -4, 13 có trọng số là 20

Chuỗi con lớn nhất

- Gọi S_i là trọng số của chuỗi con lớn nhất kết thúc tại vị trí a_i (phần tử cuối cùng của mảng con là a_i)
- $S_1 = a_1$
- Với mỗi $i > 1$:

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

- Giá trị mục tiêu tối ưu là $max_{i \in \{1,\ldots,n\}}\{S_i\}$

Chuỗi chung dài nhất

- Đặt $X = \langle x_1, \ldots, x_n \rangle$ là một chuỗi, một chuỗi con của X được tạo nên bằng cách loại bỏ một vài phần tử từ X
- Chiều dài của một chuỗi là số lượng phần tử của chuỗi đó
- Bài toán: cho hai chuỗi $X = \langle x_1, \ldots, x_n \rangle$ và $Y = \langle y_1, \ldots, y_m \rangle$, tìm chuỗi con chung dài nhất của X và Y

Chuỗi chung dài nhất

- $S(i,j)$ là chuỗi con dài nhất của $\langle x_1, \ldots x_i \rangle$ và $\langle y_1, \ldots, y_j \rangle$, $\forall 0 \leq i \leq n, 0 \leq j \leq m$
- $S(0,j) = 0, \forall 0 \leq j \leq m$
- $S(i,0) = 0, \forall 0 \leq i \leq n$
- với mỗi $i > 0, j > 0$:

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

- Giá trị mục tiêu tối ưu là $S(n,m)$

Chuỗi chung dài nhất

Algorithm 1: LCS(X, Y)

Input: Hai chuỗi $X = \langle x_1, \ldots, x_n \rangle$ và $Y = \langle y_1, \ldots, y_m \rangle$
Output: Độ dài chuỗi con chung dài nhất của x và y

foreach $j = 0, \ldots, m$ **do**
    $S(0, j) \leftarrow 0$;
foreach $i = 0, \ldots, n$ **do**
    $S(i, 0) \leftarrow 0$;
foreach $i = 1, \ldots, n$ **do**
    foreach $j = 1, \ldots, m$ **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 $S(n, m)$;

Chuỗi con chung dài nhất - Cài đặt

```java
package week7;

import java.util.Scanner;
import java.io.File;
import java.util.ArrayList;

public class LongestCommonSubsequence {
  private int[] a;
  private int[] b;

  public void readData(String fn) {
    try {
      Scanner in = new Scanner(new File(fn));
      ArrayList<Integer> ta = new ArrayList<Integer>();
      while (true) {
        int x = in.nextInt();
        if (x == -1)
          break;
        ta.add(x);
      }
      a = new int[ta.size() + 1];
      for (int i = 0; i < ta.size(); i++)
        a[i + 1] = ta.get(i);
      ta.clear();
}
```

Chuỗi con chung dài nhất - Cài đặt

```
      while (true) {
        int x = in.nextInt();
        if (x == -1)
          break;
        ta.add(x);
      }
      b = new int[ta.size() + 1];
      for (int i = 0; i < ta.size(); i++)
        b[i + 1] = ta.get(i);
      in.close();
    } catch (Exception ex) {
      ex.printStackTrace();
    }
  }
```

Chuỗi con chung dài nhất - Cài đặt

```
public void solve() {
  int[][] S = new int[a.length][b.length];
  char[][] traces = new char[a.length][b.length];
  for (int i = 0; i < a.length; i++)
    for (int j = 0; j < b.length; j++) traces[i][j] = 0;
  for (int i = 0; i < a.length; i++) S[i][0] = 0;
  for (int i = 0; i < b.length; i++)  S[0][i] = 0;

  for (int i = 1; i < a.length; i++) {
    for (int j = 1; j < b.length; j++) {
      if (a[i] == b[j]) {
        S[i][j] = S[i-1][j-1] + 1; traces[i][j] = 'D';// diagonal
      } else {
        if (S[i-1][j] > S[i][j-1]) {
          S[i][j] = S[i-1][j]; traces[i][j] = 'U';// move up
        } else {
          S[i][j] = S[i][j - 1]; traces[i][j] = 'L';// move left
        }
      }
    }
  }
```

Chuỗi con chung dài nhất - Cài đặt

```
    System.out.println("Result = " + S[a.length - 1][b.length - 1]);
    int i = a.length - 1;
    int j = b.length - 1;
    while (i >= 1 && j >= 1) {
      if (traces[i][j] == 'D') {
        System.out.println(i + "," + j + " --> " + a[i]);
        i = i-1; j = j-1;
      } else {
        if(traces[i][j] == 'U'){
          i = i-1;
        }else{
          j = j-1;
        }
      }
    }
  }

  public static void main(String[] args) {
    LongestCommonSubsequence LCSS = new LongestCommonSubsequence();
    LCSS.readData("data\\week7\\LongestCommonSubsequence\\LCSS.txt");
    LCSS.solve();
  }
}
```

Tập độc lập trọng số lớn nhất trong một Cây

- Cho một cây $T = (V, E)$
  - r là nút gốc
  - Với mỗi nút $v \in V$
    - $w(v)$: trọng số của v
    - $f(v)$: nút cha của v, theo quy ước $f(r) = \text{null}$
    - $T(v)$: cây con của T có gốc tại v
    - $Children(v)$: tập các con của v
- Một tập độc lập của T là một tập $S \subseteq V$ sao cho v và $f(v)$ không thể cùng thuộc $S, \forall v \in V \setminus \{r\}$
- Tìm một tập con độc lập của T có tổng trọng số lớn nhất

Tập độc lập trọng số lớn nhất trong một Cây

- Đặt $S(v)$ là trọng số của tập độc lập lớn nhất của $T(v), \forall v \in V$
- Đặt $\overline{S}(v)$ là trọng số của tập độc lập lớn nhất của $T(v) \setminus \{v\}$ (không xét v)
- $\overline{S}(v) = \sum_{x \in Children(v)} S(x), \forall v \in V$
- $S(v) = \max\{\overline{S}(v), w(v) + \sum_{x \in Children(v)} \overline{S}(x)\}, \forall v \in V$
- Nếu v là một lá, thì $S(v) = w(v)$ và $\overline{S}(v) = 0$

Tập độc lập trọng số lớn nhất trong một Cây

Algorithm 2: MaxIndependentSetOnTree($T = (V, E)$)

$Q \leftarrow \emptyset$;
foreach $v \in V$ **do**
    $deg(v) \leftarrow \sharp Children(v)$;
    if $deg(v) = 0$ **then**
        Enqueue(v, Q);
        $S(v) \leftarrow w(v)$;
        $\overline{S}(v) \leftarrow 0$;

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

Tập độc lập trọng số lớn nhất trong một Cây

Algorithm 3: printSol(v)

```
if sel(v) = true then
    print(v);
    foreach x ∈ Children(v) do
        printSolExclude(x);
else
    foreach x ∈ Children(v) do
        printSol(x);
```

Algorithm 4: printSolExclude(v)

```
foreach x ∈ Children(v) do
    printSol(x);
```

Tập độc lập trọng số lớn nhất trong một Cây
Cài đặt

```
package week7;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Scanner;
import java.io.File;
import java.util.HashMap;
public class MaxIndependentSetOnTrees {
  private int n;// nodes are numbered 0,1,...,n-1 (normalized nodes)
  private int[] p;// p[i] is the parent of normalized node i
  private int[] w;// w[i] is the weight of normalized node i
  private int[] id;// id[i] is the original ID of the normalized node
  public void readData(String fn){
    try{
      Scanner in = new Scanner(new File(fn));
      HashMap<Integer, Integer> mID2Index =
                new HashMap<Integer, Integer>();
      HashMap<Integer, Integer> mID2Weight =
                new HashMap<Integer, Integer>();
      HashMap<Integer, Integer> mID2Parent =
                new HashMap<Integer, Integer>();
```

Tập độc lập trọng số lớn nhất trong một Cây
Cài đặt

```
        n = 0;
        while(true){
          int u = in.nextInt();
          if(u == -2) break;
          n++;
          mID2Index.put(u, n-1);

          int wu = in.nextInt();
          int pu = in.nextInt();

          mID2Weight.put(u, wu);
          mID2Parent.put(u, pu);
        }
        p = new int[n];
        w = new int[n];
        id = new int[n];
```

Tập độc lập trọng số lớn nhất trong một Cây
Cài đặt

```
      for(int k: mID2Index.keySet()){
        int u = mID2Index.get(k);
        id[u] = k;
        int wu = mID2Weight.get(k);
        int pu = mID2Parent.get(k);
        if(pu >= 0) pu = mID2Index.get(pu);
        p[u] = pu;
        w[u] = wu;
      }
      in.close();
    }catch(Exception ex){
      ex.printStackTrace();
    }
  }
```

Tập độc lập trọng số lớn nhất trong một Cây
Cài đặt

```
public void solve(){
  int[] m = new int[n];// m[i] is the MIDS on subtree rooted at i
  int[] me = new int[n];// me[i] is the MIDS on subtree rooted at
                        // i excluding node i
  int[] d = new int[n];// d[i] is the degree of node i
                       // (number of children of node i)
  for(int i = 0; i < n; i++) d[i] = 0;
  for(int i = 0; i < n; i++){
    int pi = p[i];
    if(pi >= 0)
      d[pi]++;
  }

  Queue Q = new LinkedList();
  for(int i = 0; i < n; i++){
    m[i] = w[i]; me[i] = 0;
    if(d[i] == 0)
    Q.add(i);
  }
```

Tập độc lập trọng số lớn nhất trong một Cây
Cài đặt

```
    while(Q.size() > 0){
      int j = (int)Q.remove();
      int i = p[j];
      if(i >= 0){
        me[i] += m[j];
        m[i] += me[j];
        d[i]--;
        if(d[i] == 0){
          m[i] = me[i] > m[i] ? me[i] : m[i];
          Q.add(i);
        }
      }
    }
    int max = -1;
    for(int i = 0; i < n; i++)  max = max > m[i] ? max : m[i];
    System.out.println("result = " + max);
  }
  public static void main(String[] args) {
    MaxIndependentSetOnTrees MIST = new MaxIndependentSetOnTrees();
    MIST.readData("data\\week7\\MaxIndependentSetOnTrees\\MIST.txt");
    MIST.solve();
  }
}
```

Bài toán Khoảng-Cách-Biến-Đổi

- Đầu vào: 2 chuỗi $X = \langle x_1, \ldots, x_n \rangle$ và $Y = \langle y_1, \ldots, y_m \rangle$
- 3 thao tác trên X
  - Chèn một ký tự vào sau vị trí i
  - Xóa một ký tự tại vị trí i
  - Thay thế một ký tự bởi ký tự khác
- Tìm một chuỗi các thao tác với chiều dài nhỏ nhất để biến X thành Y (khoảng cách giữa X và Y)

Bài toán Khoảng-Cách-Biến-Đổi

- Với mỗi $0 \leq i \leq n$ và $0 \leq j \leq m$, $d(i,j)$ là khoảng cách của hai chuỗi $\langle x_1, \ldots, x_i \rangle$ và $\langle y_1, \ldots, y_j \rangle$
- $d(0,0) = 0$
- $d(0,j) = j, \forall j = 1, \ldots, m$ và $d(i,0) = i, \forall i = 1, \ldots, n$
- $d(i,j) = \min\{d(i-1,j-1) + \delta(i,j), d(i-1,j) + 1, d(i,j-1) + 1\}$ với

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

Khoảng-Cách-Biến-Đổi

Algorithm 5: EditDistance(X, Y)

Input: Hai chuỗi $X = \langle x_1, \ldots, x_n \rangle$ và $Y = \langle y_1, \ldots, y_m \rangle$
Output: Số lượng thao tác nhỏ nhất để biến X thành Y

foreach $j = 1, \ldots, m$ **do**
  $d(0, j) \leftarrow j$;

foreach $i = 1, \ldots, n$ **do**
  $d(i, 0) \leftarrow i$;

$d(0, 0) \leftarrow 0$;
foreach $i = 1, \ldots, n$ **do**
  foreach $j = 1, \ldots, m$ **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 $d(n, m)$;

Bài tập

- Gold
- Nurses
- Chuỗi con lớn nhất
- Tháp Babylon
- Marble Cut
- Mạng giao tiếp