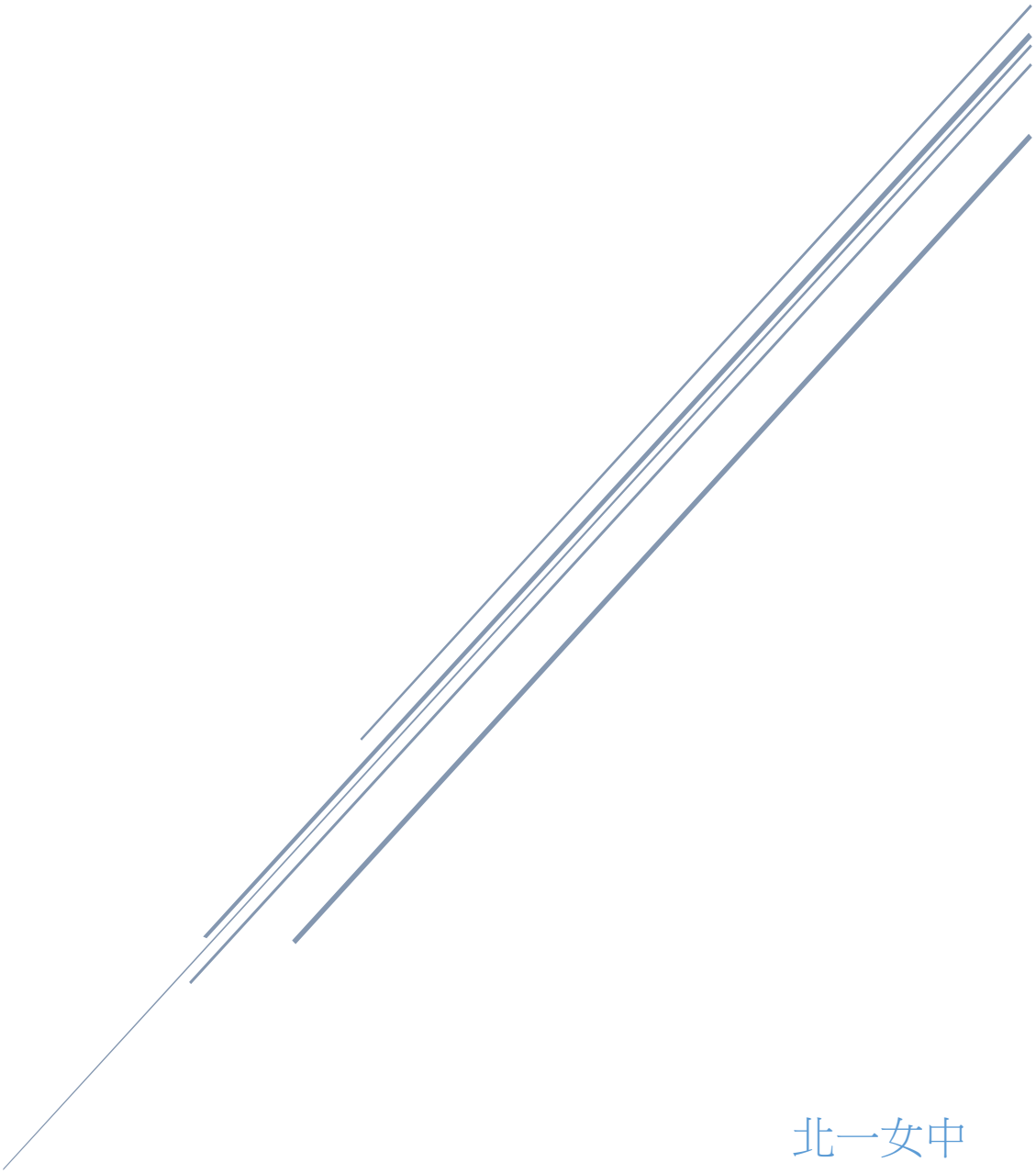
104 學年度資訊能力競賽暑期培訓講義

動態規劃篇(Dynamic Programming)



北一女中

2015/08/21

Zerojudge 題庫分類～動態規劃(Dynamic Programming)篇

內容

a252: Another LCS

內容：

一般 LCS 問題(Longest Common Subsequence, 最長共同子字串)就是給定兩個字串，求出他們的 LCS。為了理解這裡子字串定義，舉例來說，對於字串

abccdda

abc、adda、aca、bda 等都是它的子字串，但 adc、bdde、bddd 等不是他的子字串。

對於兩個字串 accbbeffg、fcebg，他們的 LCS 長度為 3，而 LCS 為 cbg 或 ceg。

現在我們把問題弄得難一點，給三個字串，請求出他們的 LCS 長度為多少？

輸入說明：

每個測資檔僅包含一筆測資，每筆測資有三個字串。測資保證三個字串的長度都不超過 100，而且字串皆由小寫字母組成。

輸出說明：

對每筆測資，輸出 LCS 的長度。

範例輸入：

```
abe
acb
babcd
```

範例輸出：

```
2
```

參考解答

```
//
// Zerojudge: a252 - Another LCS
//
// Accepted (4ms, 4.1MB), 2015/07/24
//
#include <iostream>

using namespace std;

#define N   101
#define max3(a, b, c)   max(max(a, b), c)

int dp[N][N][N] = {0};

int main(void)
{
    string s1, s2, s3;
    cin >> s1 >> s2 >> s3;

    s1 = "0" + s1;
    s2 = "0" + s2;
    s3 = "0" + s3;

    int n1 = s1.length();
    int n2 = s2.length();
    int n3 = s3.length();

    for (int i = 1; i < n1; i++)
        for (int j = 1; j < n2; j++)
            for (int k = 1; k < n3; k++)
                if ((s1[i] == s2[j]) && (s2[j] == s3[k]))
                    dp[i][j][k] = dp[i-1][j-1][k-1] + 1;
                else
                    dp[i][j][k] = max3(dp[i-1][j][k],
                                       dp[i][j-1][k],
                                       dp[i][j][k-1]);

    cout << dp[n1-1][n2-1][n3-1] << endl;
}
```

演算法筆記上的 LCS 一般解

```
//
// 這是一個 LCS (Longest Common Sequence) 的標準範例,
// 內容包含求出 LCS 的長度,
// 以及求出 LCS 的內容。
//
// 本程式以 "演算法筆記" 的內容為範本,
// 但修正了 "演算法筆記" 的些微錯誤。
//
// 寫作日期: 2015/07/24
//
#include <iostream>
#include <string>

using namespace std;

#define N  2000

string s1, s2;
int dp[N][N];   // dynamic programming array
int pa[N][N];   // parents for backtracking

void backtracking(int m, int n)
{
    if (m == 0 || n == 0)
        return;
    else if (pa[m][n] == 1)     // west
        backtracking(m, n-1);
    else if (pa[m][n] == 2)     // north-west
    {
        backtracking(m-1, n-1);
        cout << s1[m];
    }
    else if (pa[m][n] == 3)     // north
        backtracking(m-1, n);
}

int main(void)
{
    cin >> s1 >> s2;

    s1 = "0" + s1;
    s2 = "0" + s2;

    int m = s1.size();
    int n = s2.size();

    memset(dp, 0, sizeof(dp));
    memset(pa, 0, sizeof(pa));

    for (int i = 1; i < m; i++)
    {
        for (int j = 1; j < n; j++)
        {
            if (s1[i] == s2[j]) // common sequence found
            {
                dp[i][j] = dp[i-1][j-1] + 1;
                pa[i][j] = 2;   // north-west
            }
            else
```

```
            {
                if (dp[i-1][j] < dp[i][j-1])   // north path is longer
                {
                    dp[i][j] = dp[i][j-1];
                    pa[i][j] = 1;   // north
                }
                else    // west path is longer
                {
                    dp[i][j] = dp[i-1][j];
                    pa[i][j] = 3;   // west
                }
            }
        }
    }

    cout << "Longest common sequence = ";
    backtracking(m-1, n-1);
    cout << "\nLength = " << dp[m-1][n-1] << endl;
}
```

b123: 最大矩形

內容：

在一個M × N的區域內，散落了許多不同的障礙物，我們想要知道的是，在這個M × N的區域內，最大的矩形空地面積是多少？倘若我們用 0 與 1 表示這個區域內的空地狀況：0 代表這個子區域已被障礙物覆蓋，1 代表這個子區域仍為空地，我們假設每一個 0 或 1 所代表的子區域面積為 1，那麼在下面這個例子中(M=4, N=5)，最大的矩形空地為陰影所覆蓋的區域，其面積為 8。

0	0	1	1	0
0	1	1	1	1
0	1	1	1	1
0	0	1	0	0

在本題中，請依據輸入輸出的規定，針對輸入的地圖，輸出其最大的矩形空地面積。

輸入說明：

輸入資料第一行有兩個整數，依序為 M 和 N, $M \leq 200$, $N \leq 200$；接下來的 M 行中，每一行有 N 個 0 或 1 的數字。每個數字之間用一個空白隔開。

輸出說明：

請印出最大矩形空地面積。

範例輸入：

```
4 5
0 0 1 1 0
0 1 1 1 1
0 1 1 1 1
0 0 1 0 0
```

範例輸出 ：

```
8
```

參考解答

```
//
// Zerojudge: b123 - 最大矩形(Area)
//
// Accepted (0ms, 168KB), 2015/07/28
//
// 參考來源: http://hoyusun.blogspot.tw/2012/03/b123-area.html
//
#include <iostream>
#include <climits>

using namespace std;

#define M 201
#define N 201

int mp[M][N] = {0};   // map

int main(void)
{
    int m, n;

    while (cin >> m >> n)
    {
        // 讀入資料
        for (int i = 0; i < m; i++)
            for (int j = 0; j < n; j++)
                cin >> mp[i][j];

        // 逐列計算每一欄累積下來的面積(寬度)
        for (int i = 0; i < m; i++)
        {
            for (int j = 1; j < n; j++)
            {
                if (mp[i][j])
                    mp[i][j] = mp[i][j-1] + 1;
            }
        }

        int ans = 0;
        for (int i = 0; i < m; i++)
        {
            for (int j = 0; j < n; j++)
            {
                int min_width = INT_MAX;
                for (int h = 0; i-h >= 0 && mp[i-h][j]; h++)
                {
                    min_width = min(min_width, mp[i-h][j]);
                    int area = min_width * (h + 1);
                    ans = max(ans, area);
                }
            }
        }

        cout << ans << endl;
    }
}
```

b184: 裝貨櫃問題

內容：

現在一共有若干項貨品可選擇運載，每一項 k 都有一個已知的體積 v[k]，以及載運的利潤 c[k]，但是貨櫃的總容量是 100，可能無法將貨物全部裝入，希望選出其中的若干項，其體積總和不超過 100，使得利潤最大。(每一項貨物的體積為 1~100 的整數，而利潤是 1~60000 的整數) (提示：每種貨品只能選一次)

輸入說明：

第一行是貨品數量，接下來每行各有兩筆數據，第一筆代表各項貨物的體積，第二筆代表各項貨物的利潤。例如，第一筆貨物的體積為 30，利潤為 60，第二筆貨物的體積為 20，利潤為 50。

輸出說明：

輸出最大的利潤。

範例輸入：

```
4
30 60
20 50
35 40
60 70
10
80 88
33 66
13 26
77 150
95 195
45 90
8 16
20 41
40 13
68 20
```

範例輸出 ：

```
150
198
```

參考解答

```
//
// Zerojudge: b184 - 裝貨櫃問題
//
// 每種貨物只能裝一個, 求最大利潤值
//
// Accepted (0ms,80KB), 2015/07/31
//
#include <iostream>
#include <iomanip>
#include <cstring>

using namespace std;

#define N   100

int main(void)
{
    int n,          // number of sets
        v,          // volume
        c,          // cost
        dp[N+1];    // dynamic programming array

    while (cin >> n)
    {
        memset(dp, 0, sizeof(dp));
        while (n--)
        {
            cin >> v >> c;

            for (int j = N; j - v >= 0; j--)
                dp[j] = max(dp[j-v]+c, dp[j]);
        }
        cout << dp[N] << endl;
    }
}
```

d231: 基因序列密碼問題

內容：

基因序列是由四個鹼基 A、C、G、T 組合而成，例如：AGTTACGGGTTCGTAA 有可能是某個基因序列。在生物學裡常見的問題是要找出兩的基因序列的最長共同子序列 (Longest Common Subsequence)，例如：AGTTACGGGTTCGTAA 和 GTCGGAAG 的最長共同子序列是 GTCGGAA。請注意 subsequence 和 substring 不同，subsequence 的字母不需要在原來字串裡鄰近出現，只需要保持字母的順序。你的任務就是要寫一個程式找出兩個基因列序的最長共同子序列。假設每一對基因列序最多只有一個最長共同子序列。

輸入說明：

條件限制

基因序列長度為整數，1≤基因序列長度≤50

輸入格式

第一行是第一個基因序列，1≤基因序列長度≤50。

第二行是第二個基因序列，1≤基因序列長度≤50。

輸出說明：

輸出格式

請由螢幕印出第一個和第二個基因序列的最長共同子序列，如果沒有最長共同子序列就輸出字母 E。

範例輸入：

```
AAAG
GAG
```

範例輸出：

```
AG
```

參考解答

```
// Zerojudge: d231 - 基因序列密碼問題
// Accepted (0ms,168KB), 2015/07/31
#include <iostream>

using namespace std;

#define N   51

string s1, s2;
int dp[N][N] = {0}; // dynamic programming array for max length
int pr[N][N] = {0}; // prev array for backtracking

void back_track(int i, int j)
{
    if (i == 0 || j == 0) return;

    if (pr[i][j] == 2) {
        back_track(i-1, j-1);
        cout << s1[i];
    }
    else if (pr[i][j] == 1)
        back_track(i-1, j);
    else
        back_track(i, j-1);
}

int main(void)
{
    cin >> s1 >> s2;
    s1 = "0" + s1;
    s2 = "0" + s2;

    int sz1 = s1.size();
    int sz2 = s2.size();

    for (int i = 1; i <= sz1; i++)
    {
        for (int j = 1; j <= sz2; j++)
        {
            if (s1[i] == s2[j]) {
                dp[i][j] = dp[i-1][j-1] + 1;
                pr[i][j] = 2;
            }
            else if (dp[i-1][j] >= dp[i][j-1]) {
                dp[i][j] = dp[i-1][j];
                pr[i][j] = 1;
            }
            else {
                dp[i][j] = dp[i][j-1];
                pr[i][j] = 3;
            }
        }
    }

    if (dp[sz1-1][sz2-1] == 0)
        cout << "E";
    else
        back_track(sz1-1, sz2-1);
}
```

d289: 多元一次方程式

內容：

本題非常簡單？！可就不一定了，那就得看你如何看本題了。試求：

$$1 \times A + 13 \times B + 33 \times C + 43 \times D + 139 \times E + 169 \times F + 1309 \times G + 2597 \times H = I$$

的非負整數解有多少。

輸入說明：

每一筆測試資料會有一個數 I。$(1 \leq I \leq 8000)$

輸出說明：

輸出有多少組解答。

範例輸入：

```
1
2
3
4
5
```

範例輸出：

```
1
1
1
1
1
```

參考解答

```
//
// Zerojudge: d289 - 多元一次方程式
//
// Accepted (12ms, 160KB), 2015/08/06
//
// 參考出處: http://hoyusun.blogspot.tw/2012/03/d289.html
//
#include <iostream>

using namespace std;

int coins[8] = {1, 13, 33, 43, 139, 169, 1309, 2597};

int main(void)
{
   int dp[8001] = {1};

   for (int i = 0; i < 8; i++)
      for (int j = coins[i]; j < 8001; j++)
         dp[j] += dp[j-coins[i]];

   int n;
   while (cin >> n)
      cout << dp[n] << endl;
}
```

d784: 連續元素的和

內容：

已知一 n 個元素的整數數列，找出該數列連續元素的和的最大值。

輸入說明：

第一行的數字，代表有幾組測試資料，第二行開始的每一行即為一筆測試資料。
每一筆測試資料以空格分開數字：
第一個整數為數列長度 n，其範圍為[1,100]的整數；
接下來的 n 個整數，其範圍為[-10000, 10000]的整數。

輸出說明：

對每一筆測試資料，以一行輸出最大連續元素的和。

範例輸入：

```
3
5 1 2 -3 4 5
5 1 2 3 4 5
6 10 -5 7 6 -1 -3
```

範例輸出：

```
9
15
18
```

解法：(出處: http://aikosenoo.pixnet.net/blog)

[動態規劃]最大連續元素和(Maximum Consecutive Sum)

基本定義：輸入一串長度為 n 的整數序列，求最大連續和的值。

輸入

第一行有一個數字 n (1<=n<=100)
第二行有 n 個數字

輸出

一行，代表最大連續和的值

範例輸入

```
5
2 -1 5 -4 3
```

範例輸出

```
6
```

畫一個表格來看...
例一:

輸入	2	-1	5	-4	3
Sum	2	1	6	2	5
Max	2	2	6	6	6

例二:

輸入	-3	1	4	3	-1
Sum	-3	1	5	8	7
Max	-3	3	5	8	8

等於是列出一個表，然後計算 "連續元素和 sum" 和 "最大連續元素和 max"

而當 sum<0 時，sum 不累加

這樣就可以得到最後的 max 了。

參考解答一：窮舉法(不建議)

```
//
// Zerojudge, d784: 連續元素的和
// http://zerojudge.tw/ShowProblem?problemid=d784
//
#include <iostream>
#include <vector>

using namespace std;

int main(void)
{
    int lines;
    cin >> lines;
    while (lines--)
    {
        int n;
        cin >> n;
        vector<int> a(n), b(n);
        for (int i = 0; i < n; i++)
            cin >> a[i];

        int mx = 0;
        for (int i = 0; i < n; i++)
            if (a[i] < 0) mx += a[i];

        for (int i = 0; i < n; i++)
        {
            for (int j = i; j >= 0; j--)
            {
                int sum = 0;
                for (int k = i; k >= j; k--)
                    sum += a[k];
                if (mx < sum) mx = sum;
            }
        }
        cout << mx << endl;
    }
}
```

參考解答二：動態規劃法

```
//
// Zerojudge, d784: 連續元素的和
// http://zerojudge.tw/ShowProblem?problemid=d784
//
#include <iostream>
#include <vector>

using namespace std;

int main(void)
{
    int lines;
    cin >> lines;
    while (lines--)
    {
        int n;
        cin >> n;

        vector<int> a(n), sum(n);
        for (int i = 0; i < n; i++)
            cin >> a[i];

        // dp for sum
        sum[0] = a[0];
        for (int i = 1; i < n; i++)
            sum[i] = (sum[i-1] > 0)? sum[i-1] + a[i] : a[i];

        int mx = sum[0];
        for (int i = 1; i < n; i++)
            if (mx < sum[i]) mx = sum[i];

        cout << mx << endl;
    }
}
```

a164: 區間最大連續和

內容：

給定一個整數序列 A_i，其中 $1 \le i \le N$。

有 Q 筆詢問，每次詢問一個正整數數對 L, R，問閉區間 $[A_L, A_R]$ 的最大連續和。

什麼叫做 "最大連續和" 呢？請參見 d784: 連續元素的和

也就是說要找到兩個正整數 i, j 滿足 $L \le i \le j \le R$，且極大化 $A_i + A_{i+1} + \cdots + A_j$ 的值。

當 i, j 有多組解時，輸出 i 最小的。再有多組解時，輸出 j 最小的。

輸入說明：

輸入含多組測試資料，請用 EOF 判斷結束。

每組資料：

第一行有兩個正整數 N, Q。

再來一行有以空白分隔的 N 整數，依序表示 A_1 到 A_N。

再來會有 Q 行，每行兩個正整數 L, R 表示一個詢問。

保證：

單一檔案不超過十筆數據。
$1 \le N \le 200{,}000$
$1 \le Q \le 100{,}000$
$1 \le L \le R \le N$
輸入所有數字都是整數，且絕對值小於 1,000,000,000

輸出說明 ：

每一組測資的輸出第一行請輸出"Case x:"表示為第 x 組測資，從 1 開始編號。

接下來輸出 Q 行，每行三個數字依序表示該筆詢問的 i, j 以及 $A_i + A_{i+1} + \cdots + A_j$ 的值。

範例輸入：

```
10 3
0 0 0 1 0 0 0 -8 -3 5
1 7
8 9
8 10
```

範例輸出：

```
Case 1:
1 4 1
9 9 -3
10 10 5
```

提示：

本題須使用線段樹(segment tree)結構才能在時間限制下得解。

a639: DNA Density

內容：

生物的 DNA 係由 A, T, C, G 四個字元所構成的序列。因為 DNA 的序列太長了，我們希望找出比較重要的區段，以方便研究。

生物學家對於某些含有高密度 C 或 G 的區段，特別有興趣。因此，請你撰寫程式，尋找具有最高密度 C 或 G 的區段。

假設 DNA 序列的最大長度為 120 個字元，由大、小寫英文字母 ATCG 組成。生物學家有興趣的 DNA 區段，其最小長度為 L，$5 \le L \le 40$，該區段具有 C 或 G 的個數為 w，則 CG 密度為 w/L。

輸入說明：

有多筆測試資料，每筆測試資料一行，包含一個整數 L 及 DNA 序列，以空白隔開。

輸出說明：

每筆測試資料，請以一行輸出其最大 CG 密度，並精確至小數以下第二位。

範例輸入：

```
5 agGCTGCAatGACAGTTGGG
20 AaggctatacagtactaatCtTtTgcatggc
```

範例輸出：

```
0.83
0.41
```

提示：

輸入範例說明：

一、以第一筆測資為例

L=5，表示計算 CG 密度的最小長度為 5 ，但最大 CG 密度可能出現在 L = 6, 7, 8, ...。

(1) DNA 區段長度為 5，則 gGCTG、GCTGC 的 CG 密度皆為 4/5=0.80。

(2) DNA 區段長度為 6，則 gGCTGC 的 CG 密度為 5/6=0.83。

(3) DNA 區段長度 7 以上，CG 密度皆小於 0.83，所以最大 CG 密度 0.83。

二、第二筆測資：ggctatacagtactaatCtTtTgcatggc 有最大 CG 密度為 12/29=0.41。(此時區段長度為 29)

附錄：UVa 10534 – Wavio Sequence

Wavio 數列由一連串的整數構成的。他有一些有趣的特性。

- 他的長度一定是奇數，也就是 L= 2*n+1
- 在 Wavio 數列中的前 n+1 個整數為嚴格遞增。
- 在 Wavio 數列中的後 n+1 個整數為嚴格遞減。
- 在 Wavio 數列中沒有相鄰的 2 個數是一樣的。

例如：1, 2, 3, 4, 5, 4, 3, 2, 0 是一個長度為 9 的 Wavio 數列。但是 1, 2, 3, 4, 5, 4, 3, 2, 2 不是一個 Wavio 數列。在這個問題中，給你一連串的整數，請你找出在這些整數中你可以找到的一個子字串為 Wavio 數列的最大長度是多少？例如在以下一連串的整數：

1 2 3 2 1 2 3 4 3 2 1 5 4 1 2 3 2 2 1

最長的 Wavio 數列是：1 2 3 4 5 4 3 2 1，所以應該輸出 9。

輸入條件

每次輸入的資料均含有多組數列（最多不會超過 75 組）。

每組測試資料以 1 個正整數 N（$1 \le N \le 10000$）開始，代表給你數列的長度。下一列則是由 N 個整數所組成的數列（請參考 ”輸入/輸出範例”）。

輸出條件

對每組測試資料請輸出一列。在輸入的一連串的整數，請你找出在這些整數中你可以找到的 Wavio 數列最大的長度是多少。

輸入範例	輸出範例
10 1 2 3 4 5 4 3 2 1 10 19 1 2 3 2 1 2 3 4 3 2 1 5 4 1 2 3 2 2 1 5 1 2 3 4 5	9 9 1

參考解答

```
//////////////////////////////////////////////////////////////
// UVa: 10534 - Wavio Sequence
//
// Accepted: 2015/05/02
//
// 參考資料:
// 1. 演算法筆記 - Longest Increasing Subsequence
// 2. CSDN - http://blog.csdn.net/sdjzping/article/details/8758552
//
// 本題必須使用時間複雜度為 nlog(n) 的演算法來解 LIS 問題，
// 才能避色 TLE.
//////////////////////////////////////////////////////////////
#include <iostream>
#include <vector>

using namespace std;

int LIS(vector<int>& s, vector<int>& t);

int main(void)
{
    int n;

    while (cin >> n)
    {
        // read in the input data
        vector<int> s(n);
        for (int i = 0; i < n; i++)
            cin >> s[i];

        // find forward LIS length
        vector<int> a(s.size(), 1);
        LIS(s, a);

        // find backward LIS length
        vector<int> rs(s.rbegin(), s.rend());
        vector<int> b(rs.size(), 1);
        LIS(rs, b);

        vector<int> c(b.rbegin(), b.rend());

        int ans = 0, mx = -1;
        for (int i = 0; i < n; i++)
        {
            ans = 2 * min(a[i], c[i]) - 1;
            mx = (mx < ans)? ans : mx;
        }

        cout << mx << endl;
    }
}

// s 是輸入的序列, 而 t 則儲存 LIS 長度
int LIS(vector<int>& s, vector<int>& t)
{
    // 不得不判斷的特例
    if (s.size() == 0) return 0;
```

```
    // 先放入一個數字，免得稍後 v.back() 出錯。
    vector<int> v;
    v.push_back(s[0]);

    for (int i = 1; i < s.size(); ++i)
    {
        if (s[i] > v.back())
        {
            v.push_back(s[i]);
            t[i] = v.end() - v.begin();
        }
        else
        {
            vector<int>::iterator it;
            it = lower_bound(v.begin(), v.end(), s[i]);
            *it = s[i];
            t[i] = (it - v.begin() + 1);
        }
    }

    return v.size();
}
```