HW2 Solutions

tags: `HW2`

Overview

題目	出題者
pA: Weird Coin	張櫳齡
pB: BST	施育衡
pC: Flight Planning	李聖澄
pD: Speaker Diarization	李聖澄

Non-Programming

1. Warm up

Knapsack problem

第一種方法是上課教的

Sol A. 令 $dp(i,j)$ 為前 i 項物品中重量為 j 時的最大價值

$$dp(i,j)=\begin{cases} max\{dp(i-1,j), dp(i-1,j-w_i)+v_i\}, & \text{if } j\geq w_i \\ dp(i-1,j), & \text{if } j<w_i \end{cases}$$

weight

i \ j (item)	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	4	4	4	4	4
2	0	0	4	4	4	6	6
3	0	2	4	6	6	6	8
4	0	2	4	6	6	6	8

第二種方法轉移式不太一樣

Sol. B 令 $dp(i,j)$ 為前 i 項物品，價值為 j 的最小重量

$$dp(i,j)=\begin{cases} min\{dp(i-1,j), dp(i-1,j-v_i)+w_i\}, & \text{if } j\geq v_i \\ dp(i-1,j), & \text{if } j<v_i \end{cases}$$

item, value

i \ j	0	1	2	3	4	5	6	7	8	9	10	11
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	∞	∞	∞	∞	2	∞	∞	∞	∞	∞	∞	∞
2	∞	∞	3	∞	∞	∞	5	∞	∞	∞	∞	∞
3	∞	∞	1	∞	4	∞	∞	∞	6	∞	∞	∞
4	∞	∞	∞	4	∞	5	∞	8	∞	∞	∞	10

Game of Stone

$$dp(i,j)=P_i+\_\_\_\_(A)\_\_\_\_$$

$$(A): \sum_{k=i+1}^{j} P_k - dp(i+1,j) \text{ 或 } min\{dp(i+2,j), dp(i+1,j-1)\}$$

$$dp(i,j)=\_\_\_\_(B)\_\_\_\_+P_j$$

$$(B): \sum_{k=i}^{j-1} P_k - dp(i,j-1) \text{ 或 } min\{dp(i+1,j-1), dp(i,j-2)\}$$

$$dp(i,j)=max\{P_i+(\sum_{k=i+1}^{j} P_k - dp(i+1,j)), (\sum_{k=i}^{j-1} P_k - dp(i,j-1))+P_j\}$$

或 $dp(i,j)=max\{P_i+min\{dp(i+2,j), dp(i+1,j-1)\}, min\{dp(i+1,j-1), dp(i,j-2)\}+P_j\}$

表格：

	1	2	3	4	5	6
1	2	8	5	15	10	18
2		8	8	11	15	16
3			3	7	8	10
4				7	7	10
5					5	5
6						3

2. LIS

(1).

窮舉所有可能的子序列，時間複雜度為$O(2^n)$

(2).

$O(n)$往舊數列尋找比新增數字還小的數字，並取得「以此數作為LIS數列結尾」的LIS長度，其值+1即是新的LIS長度。

(3).

Yes.

轉移式如下:
$dp[i]=max(dp[j]+1),\ j<i\ and\ s[i]>s[j]$

維護dp數列，其$dp[i]$代表以第i個數字為結尾的最長LIS數列長度。求$dp[i]$時，對於所有的j滿足$j<i$且$s[j]<s[i]$，取$dp[j]+1$的最大值，即 `dp[i] = max(dp[i], dp[j]+1)` 。

(4).

見此處

3. LCS and LIS

(1).

$dp[i][j]=s_1$的前i個數字和s_2的前j個數字的LCS

$$dp[i][j]=\begin{cases} dp[i-1][j-1]+1, & \text{if } i>0\ and\ j>0\ and\ s_1[i]=s_2[j] \\ max(dp[i][j-1], dp[i-1][j]), & \text{else if } i>0\ and\ j>0\ and\ s_1[i]\neq s_2[j] \\ 0, & \text{else if } i=0\ or\ j=0 \end{cases}$$

i^{th} row / j^{th} column	j=0	s[j=1]=1	s[j=2]=3	s[j=3]=2	s[j=4]=4
i=0	0	0	0	0	0
s[i=1]=1	0	1	1	1	1
s[i=2]=2	0	1	1	2	2
s[i=3]=3	0	1	2	2	2
s[i=4]=4	0	1	2	2	3

(2).

拿(1).的表格舉例，假設我們是以Row-Major(將一個row填滿才會去填下一個row)，可以發現其實我們**只需要當前的row和上一個row而已**。至於再前面的row則不需要。所以只需要$2\times N$個空間，時間複雜度為$O(N)$。
利用位元運算的技巧稍微修改一下dp式，使兩個row可以交替使用(**請注意，AND為邏輯運算)：

$$dp[i\ AND\ 1][j]=\begin{cases} dp[(i-1)\ AND\ 1][j-1]+1, & \text{if } i>0\ and\ j>0\ and\ s_1[i]=s_2[j] \\ max(dp[i\ AND\ 1][j-1], dp[(i-1)\ AND\ 1][j]), & \text{else if } i>0\ and\ j>0\ and\ s_1[i]\neq s_2[j] \\ 0, & \text{else if } i=0\ or\ j=0 \end{cases}$$

Programming

pA: Weird Coin

我們可以將n用二進制表示來想：

```
n = 1，只有1種表示方法。
n = 2，10 (2進制)，二進制表示下，可以分成{1,1}，{10}有兩種表示方法
n = 3，11 (2進制)，可以分成{1,1,1}，{10,1}
n = 4，100 (2進制)，{1,1,1,1}，{10,1,1}，{10,10}，{100}
```

我們可以知道如果所求的n為奇數，那麼所求的分解結果中必含有1，因此，直接將 n-1 分出來的結果中加1即可，為s[n-1]。
如果所求的n為偶數，那麼n的分解結果分成兩種情況：

1. 含有1，這種情況可以直接在 n-1 的分解結果中加1即可 s[n-1]
2. 不含有1，那麼，分解因子的都是偶數，將每個分解的因子都除以2，剛好是n/2的分解結果，並且可以與之一一對應，這種情況有 s[n/2]

▼ 參考程式碼

```cpp
#include <iostream>
#include <vector>
using namespace std;
#define mod 1000000007

int main(){
    vector<int> num(10000015, 0);

    int index = 3;
    int n = 0;
    num[1] = 1;
    num[2] = 2;
    while(index <= 10000000){
        num[index] = num[index-1];
        index++;
        num[index] = (num[index-1] + num[index / 2]) % mod;
        index++;
    }

    while(cin >> n){
        cout<<num[n]<<endl;
    }

    return 0;
}
```

pB: BST

那個先從最簡單的暴力法開始思考，不外乎就是窮舉每一棵二分樹，並找出最小的最大路徑，這個大家應該都想的到。dp 說白了就是一種空間去換時間的做法，那是不是可以用局部的最優解去推出完整的最優解呢？

在暴力的過程中有一步應該是要窮舉每個數字當父節點的情況，那在求 `2 6 7 15 16 18 23` 的問題中可以把 `15` 當作父節點去求剩下數字的最佳二元樹，因為我們題目的樹是二分搜尋樹，對於每個父節點，它的左子節點都會小於該父節點，右子節點也是，所以問題會變成父節點的左右區間的最佳解取較大值加上父節點

$dp(i,\ j)=min(k+max(dp(i,\ k-1),\ dp(k+1,\ j)))$

pC: Flight Planning

先從給定的規則解析本題的限制：

1. 只能從給定的航線中安排
   這是最基本的規則，不然測資就沒意義了
2. 所有航班必須同時起飛
   此條件僅為簡化題目背景，因此不會出現輪流起飛的情況
3. 一個機場一次只能起飛一架班機
4. 一個機場不能同時降落兩台以上的班機
   所有機場的起飛降落關係為一對一
5. 航線間不能有交錯

第五條條件是解題的關鍵點。先看一個合理的航線安排該如何表示：

```
2 4 5
1 2 4
```

此為範例圖示的合理航線安排，下方為Alan國，上方為ROC。

可以發現，若航線不交錯，上下兩個數列都會是遞增的，那若是存在交錯的航線呢？

```
2 4 1
1 2 4
```

如此範例，第三條航線與其他兩條航線交錯，因此出現上方數列不遞增的情況。

綜合以上目標，程式的目標便是**選定一組航線，使得起點和終點的數列皆為遞增**。

首先，我們先對所有航線進行排序：先比較第一個元素，由小排到大，若第一個元素相等則排序第二個元素，由大排到小（原因後面會解釋）。

以範例圖示為例，會得到以下數列

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

如此一來，我們至少可以保證上方數列是遞增的，現在僅需在下方數列選擇遞增的子序列即可。又因我們要選取越多航線越好，因此要找下方數列的**LIS長度**。

先解釋為何第二個元素要逆序排列。當第一個元素的值一樣時，依據題目規定，我們只能從這幾個航線中挑出一個航線（同個機場只能起飛/降落一架班機），以下方例子為例：

```
1 1 1
2 3 4
```

挑選時，我們只能在 `(1,2), (1,3), (1,4)` 中挑選一條航線，作為LIS的一部分。**倘若排序第二個元素時為遞增排序，則在尋找LIS長度時，有可能選到兩條以上的航線**。上述例子中，理論上只能選擇一條，但在LIS的作法下，會選到三條航線，這顯然是不對的。

至於解決的方法，便是保證在數對第一個值相同的集合中，任兩個數對的第二個值不會遞增，因為**凡是有兩個數對的第二個值是遞增關係，便有可能選到兩個以上的航線**。讓所有數對不遞增的方法，就是**降序排列**。在第一個值一樣時，將對第二個值進行降序排列，可以保證找LIS時，這些數對中只會取得一個。

LIS的$O(N^2)$在此不贅述，僅講解$O(NlogN)$的作法。

我們維護一個數列lis，其$lis[i]$代表**LIS長度為i的數列中，結尾數字最小的值**，對於每次遍歷到的值，使用**二分搜**去尋找第一個大於目前值的位置j（我就說還會用到二分搜），並取代該位置j的數字，相當於「找到LIS長度為j的所有LIS數列中，結尾值更小的數字」。使用這樣貪婪的作法，能保證最後lis最大的index，便是數列的LIS長度。

此演算法稱作RSK algorithm，使用二分搜+貪婪對LIS進行加速。更詳細的解說與證明可參考演算法筆記。

▼ 參考程式碼

```cpp
#include <bits/stdc++.h>
using namespace std;

class Route{
public:
    int u, v;
    bool operator< (Route rhs) const {
        // 排序規則
        return u == rhs.u ? v > rhs.v : u < rhs.u;
    }
};

int n,m,r;

int lis(vector<int>& arr){
    // tmp[i]代表LIS長度為i的所有序列中，結尾數字最小的值
    // 為二分搜方便，將頭設為極小值-1，其餘為極大值0x3f3f3f3f
    vector<int> tmp(arr.size()+1, 0x3f3f3f3f);
    int ans = 0;
    tmp[0] = -1;
    for(int i : arr){
        // 尋找該數字能插入tmp的哪個位置，即該數字為結尾最長的LIS長度
        int pos = lower_bound(tmp.begin(), tmp.end(), i) - tmp.begin();
        // 更新tmp及答案
        tmp[pos] = i, ans = max(ans, pos);
    }
    return ans;
}

int main(){

    scanf("%d %d %d", &n, &m, &r);

    vector<Route> routes(r);
    for(int i = 0;i < r;++i){
        scanf("%d %d", &routes[i].u, &routes[i].v);
    }

    // 排序後將要做LIS的數列取出
    sort(routes.begin(), routes.end());
    vector<int> arr;
    for(int i = 0;i < r;++i){
        arr.push_back(routes[i].v);
    }

    printf("%d\n", lis(arr));

}
```

pD: Speaker Diarization

本題的要求為在給定的區間集合中，選擇最少的區間，使得區間的聯集時間為最大，換句話說，就是**選最少的區間，覆蓋到最多的時間**。

這題是使用到**區間覆蓋演算法**，也是一個證明過正確性的貪婪演算法，在課堂中有提到過並給出證明。其選擇的策略非常簡單：

1. 用左邊界的大小升序排序所有區間，以保證開始的時間會一直增加
2. 以第一個區間作為第一個選擇的區間
3. 對於目前選擇的區間，尋找「左邊界位於此區間中」的所有區間中，右邊界最大的區間
   > 白話點就是找與目前區間有交集的所有區間裡，右邊界伸最遠的
4. 選擇此區間，並重複第三步，直到選完為止。

在此策略中，每個區間都只會被看過一次，所以可以宣告一個 `idx` 一直遞增下去。舉個例子，若編號1-5的區間中，要找右邊界伸最遠的區間，即使選定的區間編號為2，也會因為在這次選取中，剩下編號3-5的區間永遠不會伸得比編號2還遠，下一輪中絕對選不到這些區間，因此不用從3開始再看一次。

這個演算法原先的作法，若無法完全覆蓋整段時間的測資，會直接回傳-1，但只要在做法上做一些微調，便可輕鬆求得未覆蓋到的時間總和最小值。

對於一個選定的區間，**若尋找右邊界伸最遠的區間時沒有找到，則直接選擇下一個區間，並計算未覆蓋的時間長度。**

策略對了，剩下就是實作的小細節了，包括維護現在覆蓋到的最右邊界，以及目前找到伸最遠的最右邊界等等。參見以下程式碼：

▼ 參考程式碼

```cpp
#include <bits/stdc++.h>
using namespace std;

class Segment {
public:
    int a, b;
    bool operator< (Segment rhs) const{
        // 以左邊界排序數對，左邊界相同則排序右邊界
        return a == rhs.a ? b < rhs.b : a < rhs.a;
    }
};

int main(){
    int n,l;
    scanf("%d %d", &n, &l);
    vector&lt;Segment&gt; segs(n);
    for(int i = 0;i < n;++i){
        scanf("%d %d", &segs[i].a, &segs[i].b);
    }

    int now_r = 0, max_r = 0, cnt = 0, idle = 0, idx = 0;
    sort(segs.begin(), segs.end());

    // 先算第一個區間開始前的空白片段
    idle += segs[0].a;
    // 重新設定起始的已覆蓋的右邊界值
    now_r = max_r = segs[0].a;
    //當區間沒完全遍歷且右邊界還沒到底時，繼續執行區間覆蓋
    while(idx < n && now_r < l){
        if(segs[idx].a > now_r){
            // 若現在的左邊界大於目前覆蓋到的右邊界，代表中間有空白片段
            idle += segs[idx].a - now_r;
            now_r = max_r = segs[idx].a;
        }
        // 策略核心：尋找接下來「左邊界位於目前區間中」的所有區間，並找出其中右邊界最大的區
        while(idx < n && segs[idx].a <= now_r && max_r < l){
            max_r = max(max_r, segs[idx].b);
            ++idx;
        }
        // 更新目前覆蓋到的右邊界
        if(now_r != max_r) now_r = max_r, ++cnt;
    }
    // 結束前的空白片段
    idle += l - now_r;
    printf("%d %d\n", cnt, idle);

}
```