動態規劃學習筆記

前言:

程式競賽的各個演算法個個演算法中，我認為涵蓋範圍最廣的可以說是動態規劃。動態規劃頻繁地出現於各個演算法中。除了最單純的動態規劃求費氏數列，再到RMQ的Sparse Table，樹論中的樹直徑，都可以看見動態規劃的影子存在。本筆記共分成三個部分，其一為單純的動態規劃，其二為利用動態規劃技巧求靜態的RMQ
，最後為利用動態規劃求樹直徑。
本篇筆記為我和另一位同學一起整理，主要分工為他負責背包問題，我負責稀疏表以及樹直徑。

心得:

完成這一篇筆記，感覺我的高中生活又增加了點甚麼。高中三年真的太快，要想留下一些回憶，勢必得將走過的路記錄下來。而這一篇筆記正是我們學習動態規劃所留下的足跡。另外，我們在學習時所參考的資料非常的雜，很多都是資訊大神們在學成之後所留下來的，許多資料都艱辛難懂，對於初學者而言相當不利，我們希望能夠留下一份盡可能淺顯易懂的資料，讓未來的學弟學妹們能夠更順利。也希望未來的學弟學妹們能夠一起建立起一份學習的資料庫，讓大家一起進步!

動態規劃裸題:背包問題

01背包

題目敘述

- 有N件物品和容量為V的背包，第i件物品的費用為c[i]價值為w[i]，求裝那些物品能使背包內價值總和最大?
- $1 \leq n \leq 100, 1 \leq v \leq 1000$

大致思路

- 由於每樣物品都為一件，故能以每件物品放或不放來思考。
- 定義式:$dp[i][v]$表示前i件物品在容量為v時可獲得的最大價值。
- 轉移式:$dp[i][v] = max\{dp[i-1][v], dp[i-1][v-c[i]] + w[i]\}$
- 我們能對此題進行一簡單的優化將其空間複雜度降為O(V)，觀察轉移式可知，當運算到第i項物品時，只需要知道i-1項之值即可，故只需開一維dp[]陣列，再來由於每樣物品最多一樣，所以運算到dp[v-c[i]]時要由最大值V~c[i]逆序運算，以免覆蓋掉原陣列之答案，此方法也被稱為滾動陣列。

複雜度

- $O(NV)$

練習題

- [洛谷P1048 (https://www.luogu.com.cn/problem/P1048)](https://www.luogu.com.cn/problem/P1048)

完全背包

題目敘述

- 有N件物品和容量為V的背包，每件物品都有無限件用，第i件物品的費用為c[i]價值為w[i]，求裝那些物品能使背包內價值總和最大?
- $1 \leq n \times v \leq 10^7$

大致思路

- 這題和01背包問題思路大致相同，但由於每樣物品都有無限件，故無法以每件物品放或不放來思考，但我們依然能以相同方式定義。
- 定義式:$dp[i][v]$表示前i件物品在容量為v時可獲得的最大價值。
- 轉移式:
$dp[i][v] = max\{dp[i-1][v], dp[i-1][v-k \times c[i]] + k \times w[i] | 0 \leq k \times c[i] \leq V\}$
- 完全背包問題能透過01背包問題的滾動陣列優化之概念將時間複雜度降為O(NV)，大致概念和01背包問題相同，但這次物品的數量並無上限，故不用和01背包問題相同害怕覆蓋到原陣列之答案而逆序運算，只要由c[i]~最大值V運算即可，由此可見dp中充次著許多神奇的優化方式。

複雜度

- 原$O(V \times \sum(v/c[i]))$，優化後為$O(NV)$

練習題

- [洛谷P1616 (https://www.luogu.com.cn/problem/P1616)](https://www.luogu.com.cn/problem/P1616)

多重背包

題目敘述

- 有N件物品和容量為V的背包，第i件物品的數量為n[i]費用為c[i]價值為w[i]，求裝那些物品能使背包內價值總和最大?
- $N \leq \sum n_i \leq 10^5, 0 \leq V \leq 4 \times 10^4, 1 \leq N \leq 100$

大致思路

- 這題有需多優化方式，但我們先依照和上述相同的方式定義。

- 定義式:$dp[i][v]$表示前i件物品在容量為v時可獲得的最大價值。

- 轉移式:$dp[i][v] = max\{dp[i-1][v], dp[i-1][v-k \times c[i]] + k \times w[i] | 0 \le k \le n[i]\}$

- 多重背包問題能透過二進制編碼轉為01背包問題將時間複雜度降為$O(V \times \sum logn[i])$，這裡我們要使用單調對列優化方式將複雜度降為和前面相同的O(VN)。

- 首先原轉移式為:
  $dp[i][v] = max\{dp[i-1][v], dp[i-1][v-k \times c[i]] + k \times w[i] | 0 \le k \le n[i]\}$

- 由滾動陣列優化後
  $dp[j] = max\{dp[j], dp[j-v] + w, \ldots\ldots, dp[j - s \times v] + s \times w\}$

- 由此可見原問題被轉為求$dp[j], dp[j-v] + w, \ldots\ldots, dp[j - s \times v] + s \times w$ 的最大值

- 透過觀察可知j、$j-v$、$j-2v \ldots j-sv$是同餘v的，而dp[0...V]中共有v组，由此可知這v組中的數據只會受到前一層和它同餘的狀態影響，如下:

```
dp[j]    = dp[j]
dp[j+v]  = max(dp[j]+w ,dp[j+v])
dp[j+2v] = max(dp[j]+2w,dp[j+v]+w ,dp[j+2v])
dp[j+3v] = max(dp[j]+3w,dp[j+v]+2w,dp[j+2v]+w,dp[j+3v])
.....
```

為了計算方便稍作修改如下:

```
dp[j]    = dp[j]
dp[j+v]  = max(dp[j],dp[j+v]-w)+w
dp[j+2v] = max(dp[j],dp[j+v]-w,dp[j+2v]-2w)+2w
dp[j+3v] = max(dp[j],dp[j+v]-w,dp[j+2v]-2w, dp[j+3v]-3w)+3w
......
```

- 這樣函數內比較形式统一後dp[j+kv]即代表前k項的最大值無須再全值修改後比較。

- 最後只需以單調對列之方式計算dp[j]~dp[j+sv]之最大值即可，由此便可發現動態規劃中更多優美的優化方式。

複雜度

- 原$O(V \times \sum n[i])$，優化後為$O(NV)$

練習題

- [洛谷P1776 (https://www.luogu.com.cn/problem/P1776)](https://www.luogu.com.cn/problem/P1776)

動態規劃應用:Sparse Table

前言

若要求某區間的和可以利用前綴和來處理。那麼，假使區間極值呢?是不是也可以事先將所有記錄下來，得到$O(1)$的查詢呢?答案是可以的。我們利用一種資料結構叫做稀疏表(sparse table)可以做到。雖然他看起來很像前綴和，但是本質上還是dp。嚴格來講，前綴和也可算是dp的一種應用。

想法

- 假設有一個陣列$A_1, A_2, A_3, ..., A_n$
- 令 $spt[i][j] = max/min\{A_i$ 、A_{i+1} 、$...$ 、$A_{i+2^j-1}\}$
- 如此，可以建一張$log_2 n \times n$的表格
- 填滿整張表格，複雜度為$nlog_2 n$

建立表格

```
//初始化第一排
for(int i=0;i<n;++i)spt[i][0]=arr[i];
for(int j=1;j<=log2(n)+1;++j){
    for(int i=0;i<n;++i){
        //利用位元運算加速，避免落入次方的複雜度陷阱
        spt[i][j]=max(spt[i][j-1],spt[i+(1<<(j-1))][j-1]);
    }
}

```

如何搜尋

表格倒是建好了，現在的問題是:我要如何快速的搜尋出我所要的區間極值?這個搜尋的速度是真的很快很快，因為他只是找兩個索引然後進行運算而已。
假設要求區間l, r之間的極值，我們可以換個想法，先假設$int\ x = log_2(r - l + 1)$，我們可以尋找**由$l$往右推$x$格**，**由$r$往左推$x$格**兩個區間聯集的極值。為何要這樣做?因為$l - r$不一定為2的冪次，如果單純從$l$往右或是$r$往左都有可能造成部分區間沒有搜尋到的問題，因此我們要取兩段區

間，剛剛好重疊一點點或是完全沒有重疊，又必須要是二的冪次才可以發揮稀疏表的功能，因此，我們便選擇了這兩段區間。
以下為程式碼:

```
cin >>l>>r;
int x=log2(r-l+1);
cout<<max(spt[l][x],spt[r-(1<<x)+1][x])<<'\n';
```

致命缺點

稀疏表仍是有幾個較為致命的缺點。其一為記憶體的使用。我在解靜態RMQ裸題的時候使用了稀疏表，使用的記憶體量竟然高達32MB，這是第一個致命缺點，如果題目再複雜一些或許就會MLE，第二個缺點就較為致命了，就是他不支援單點修改值。
我認為，稀疏表與BIT(binary index tree,二元索引樹)有些相似，然而，若是在BIT中單點修改，只需要修改log_2n個點就可以了，但是在稀疏表中進行單點修改，卻是整張表都要重新建立，若是有m個數字，進行n次操作，每次操作會修改並且查詢，重新建表的複雜度為$O(mlog_2m)$，查詢的複雜度為$O(1)$(還是很快)，單點修改的複雜度等同於建表，進行n次，這樣整體複雜度會高達$O(nmlog_2m)$顯然很容易造成TLE。再者，稀疏表在建表、查詢時需要極謹慎，一不小心差一格都可能查詢錯誤，錯誤率高，這是稀疏表的幾個缺點。相較而言其他的資料結構，如選手自己發明的線段樹，就好寫許多。

動態規劃應用:樹直徑

前言:

要找尋一棵樹的直徑，我們有兩中方法。其一實作上較簡單，操作兩次dfs尋找最遠距離。第一次隨機找一個點p，找他的最遠距離，假使與p距離最遠的點叫做k，此時，我們只需要再對k做一次dfs找到距離k最遠的點q，$distance(k,q)$就是樹的直徑。
然而，這個看似相當好操作的方法其實有個缺點，但凡有某條路徑的長度為負數，找到的樹直徑就會有問題。再者，其時間複雜度不佳，因此，我們可以再換個方式，利用樹狀DP的技巧來找到樹直徑。

想法:

首先，先開兩個陣列$d1[N], d2[N]$，$d1[i]$為以第i個節點當作子樹樹根往下搜尋的最大深度，$d2[i]$為以第i個節點當作子樹樹根往下搜尋的次大深度。如此，樹直徑就是所有$d1[i]+d2[i]$中的最大值。

程式實作

第一行輸入一個n,$n \leq 10^5$，代表共有n個點
接下來$n-1$行每行三個整數a, b, x，代表b為a的兒子且$dis(a,b)=x, -10^4 \leq x \leq 10^4$

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

//d1[i]:以i為根的子樹之最大/次大深度 rtd:判斷樹根用
//len[i]:第i個節點到他父親之間的距離
int n,rt,d1[100005],d2[100005],rtd[100005],len[100005];
vector<int> v[100005];

inline int dfs(int root){
    if(v[root].empty()){
        d1[root]=0,d2[root]=0;
        return len[root];
    }
    int mx1=-0x3f3f3f3f,mx2=-0x3f3f3f3f;
    for(auto i:v[root]){
        if((dfs(i)+len[i])>mx1){
            mx2=mx1;mx1=dfs(i)+len[i];
        }
    }
    d1[root]=mx1,d2[root]=mx2;
}

int main(){
    cin >>n;
    for(int i=0;i<n-1;++i){
        int a,b,x;
        cin >>a>>b>>x;
        v[a].eb(b);
        len[b]=x;
        rtd[b]++;
    }
    for(int i=1;i<=n;++i){
        if(rtd[i]==0){
            rt=i;
            break;
        }
    }

    dfs(rt);
    int mx=-0x3f3f3f3f;
    for(int i=1;i<=n;++i){
        if((d1[i]+d2[i])>mx)mx=d1[i]+d2[i];
    }
    cout<<mx<<'\n';
}
```