307-01-树形背包$O(n^2)$算法

P2014选课

题目描述

在大学里每个学生，为了达到一定的学分，必须从很多课程里选择一些课程来学习，在课程里有些课程必须在某些课程之前学习，如高等数学总是在其它课程之前学习。现在有N门功课，每门课有个学分，每门课有一门或没有直接先修课（若课程a是课程b的先修课即只有学完了课程a，才能学习课程b）。一个学生要从这些课程里选择M门课程学习，问他能获得的最大学分是多少?

输入输出格式

输入格式：

第一行有两个整数N,M用空格隔开。$(1 \leq N \leq 300, 1 \leq M \leq 300)$

接下来的N行,第I+1行包含两个整数ki和si, ki表示第I门课的直接先修课，si表示第I门课的学分。若ki=0表示没有直接先修课（1<=ki<=N, 1<=si<=20）。

输出格式：

只有一行，选M门课程的最大得分。

题解

建图

兄弟儿子表示法建图：

```
for (int i = 1; i <= n; i ++) {
    cin >> p[i] >> a[i];  // a[i]: 值
    bro[i] = son[p[i]];   // i.bro = i.father.son
    son[p[i]] = i;        // i.p.son = i
    // 由于如果没有前继，p[i]=0，所以此时其前继为虚拟根：0
}
```

三次方$O(n^3)$算法：

首先分析只是二叉树的情况：

设对于一个节点u，设它的左儿子和右儿子分别为：L, R。设以左儿子为根的子树选α门课，以右儿子为根的子树选β门课，会有以下的递推公式：

$$
\begin{aligned}
f(u,0) &= 0 && (1)\\
f(u,1) &= A[u] && (2)\\
f(u,k) &= \max_{\alpha+\beta=k-1}\{f(L,\alpha)+f(R,\beta)+A[u]\} && (3)
\end{aligned}
$$

其中，$f(u,k)$表示以u为根节点的子树选k门课。

但是其并非二叉树，所以可以考虑两种方式：

（一）逐个添加儿子：

$$
\begin{aligned}
&0 \text{ son: } f^{(0)}(u,k)\\
&\downarrow\\
&1 \text{ son: } f^{(1)}(u,k)\\
&\downarrow\\
&\cdots\\
&\downarrow\\
&i-1 \text{ son: } f^{(i-1)}(u,k)\\
&\downarrow\\
&\text{i son: } f^{(i)}(u,k)
\end{aligned}
$$

由上面的想法，我们可以比较容易地得到以下的递推公式：

$$
\begin{aligned}
f^0(u,k) &= \begin{cases} 0 & k=0 \\ A[u] & k\geq 1 \end{cases} && (4)\\
f^{(i)}(u,k) &= \max_{0\leq\alpha<k}\{f(v_i,\alpha)+f^{(i-1)}(u,k-\alpha)\} && (5)\\
f(u,k) &= f^{(j)}(u,k) && (6)
\end{aligned}
$$

其中j为节点u的儿子的数量。

代码：

```
dfs(int u) {
  f[u,0] = 0; f[u,1] = a[u];
    for (int i = son[u]; i != 0; i = bro[i]) {
        dfs(i);
        // 背包，故从大到小(考虑更新顺序)
        for (int k = m + 1; k >= 1; k --) {// m + 1: 学习0(虚拟根),
多学一门
            f(i)[u,k] = f(i - 1)[u,k];
            for (α = 0; α < k; α ++) {
                                    {        f(i)[u,k];      }
                f(i)[u,k] =   max {                          }
                                    { f[i,α] + f(i-1)[u,k-α];}
            }
        }
    }
}
```

压缩i（背包问题）

```
dfs(int u) {
    f[u,0] = 0; f[u,1] = a[u]; // 初始化
    for (int i = son[u]; i != 0; i = bro[i]) {
        dfs(i);
        // 背包，故从大到小(考虑更新顺序)
        for (int k = m + 1; k >= 1; k --) {// m + 1: 学习0(虚拟根),
多学一门
            for (α = 0; α < k; α ++) {
                                {        f[u,k];      }
                f[u,k] =  max {                       }
                                {  f[i,α] + f[u,k-α]; }
            }
        }
    }
}
```

```
int main() {
    // 构图
    dfs(0);
    cout << f[0,m+1]; // 虚拟根，所以m+1
    return 0;
}
```

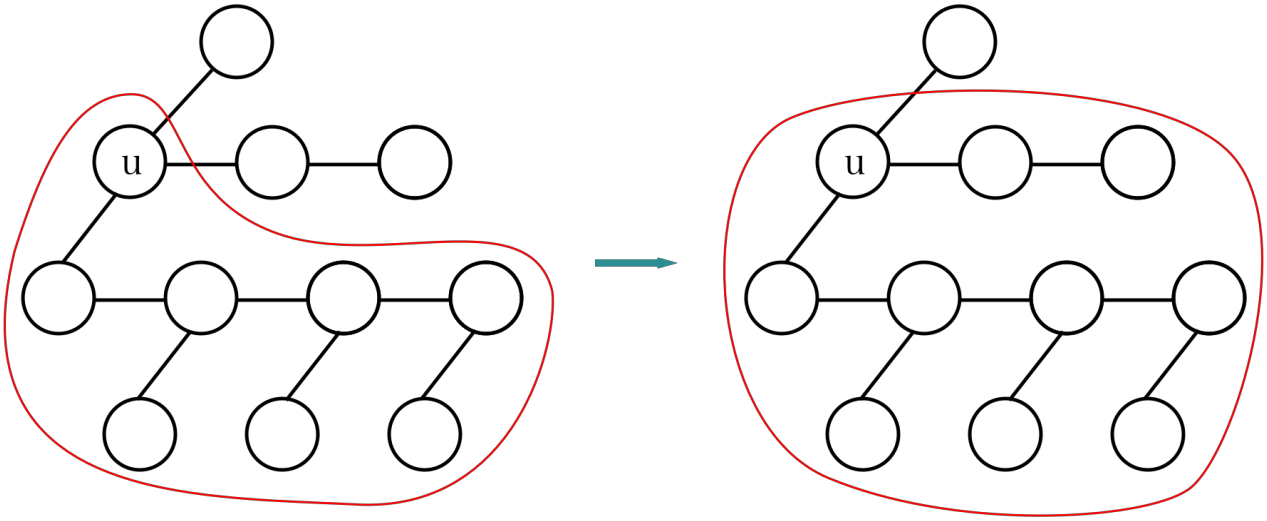
时间复杂度分析：

二次方$O(n^2)$算法

计算一个单元需要$O(n)$，计算n^2单元—>$O(n^3)$算法。

但是想要其变为平方算法$O(n^2)$：所以在此使用第二种方法来解决多叉树的问题：

现在定义$f(u,k)$表示的以这个节点为根的一座包括自己所有儿孙和弟弟们的一片森林：



这样的好处就是可以把其分为互不相干的两个部分：儿孙们和弟弟们。

递推公式：

$$f(u,k)=\max\begin{cases}f(b,k)\\ \max_{\alpha+\beta=k-1}\{A[u]+f(s,\alpha)+f(b,\beta)\}\end{cases}\tag{7}$$

代码：

```
dfs(int u) {
    int b = bro[u], s = son[u];
    if (s != 0) dfs(s);
    if (b != 0) dfs(b);
    for (k = 1; k <= m; k ++) f[u][k] = f[b][k];
    for (α = 0; α <= m; α ++) {
        for (β = 0;  β <= m; β ++) {
            k = α + β + 1;
            f[u][k] = max {
                f[s][α] + f[b][β] + A[u];
                f[u][k];
            };
        }
    }
}
```

```
dfs(son[0]);
cout << f[son[0]][m];
```

很不幸，它还是$O(n^3)$的。（证明略）

优化：（将k改为$sz[s], sz[b]$）

```
dfs(int u) {
    int b = bro[u], s = son[u];
    f[u,0] = 0; f[u,1] = a[u]; // 初始化
    if (s != 0) dfs(s);
    if (b != 0) dfs(b);
    sz[u] = sz[s] + sz[b] + 1; // 算子树大小
    for (k = 1; k <= m; k ++) f[u][k] = f[b][k]; // 划水行为(递推公式第一行)
    for (α = 0; α <= sz[s]; α ++) {
        for (β = 0;  β <= sz[b]; β ++) {
            k = α + β + 1;
            f[u][k] = max {
                f[s][α] + f[b][β] + A[u];
                f[u][k];
            };
        }
    }
}
```

对于时间复杂度的证明：

非常好证明以下不等式：

$$a^2 + b^2 + ab \leq (a + b + 1)^2 \tag{8}$$

在这里我们记$|s| = sz(s)$, $|b| = sz(b)$，所以时间复杂度：

$$\begin{aligned} & |s|^2 + |b|^2 + |s||b| && (9)\\ \leq\ & (|s| + |b| + 1)^2 && (10)\\ =\ & u^2 && (11) \end{aligned}$$

故时间复杂度为$O(n^2)$。

将方法（一）的$O(n^3)$算法变为$O(n^2)$算法：

```
void dfs(int u) {
    sz[u] = 1;
    for (int i = son[u]; i != 0; i = bro[i]) {
        dfs(i);
        sz[u] += sz[i];
        for (int k = sz[u]; k >= 1; k --) {
            for (α = k - 1; α >= 0; α --) {
                f[u][k] = max{
                    f[u][k];
                    f[u][k-α] + f[i][α];
```

```
                };
            }
        }
    }
}
```

注：

（一）对于第二种方法，开数组时要开两倍大小，因为$k=\alpha+\beta+1\leq 2m+1$

（二）对于第一种方法的背包问题之所以要$k:t\to 1$，是因为这样可以避免$f^{(i-1)}(u,k-\alpha)$提前被更新。

完整代码：

（一）方法一：

```cpp
#include <iostream>

using namespace std;

typedef long long ll;
const int maxn = 305;
ll n, m;
ll son[maxn], bro[maxn], a[maxn], p[maxn], f[maxn][maxn], sz[maxn];

void dfs(int u) {
  sz[u] = 1;
    f[u][0] = 0; f[u][1] = a[u];
  for (int i = son[u]; i != 0; i = bro[i]) {
    dfs(i);
    sz[u] += sz[i];
    for (int k = sz[u]; k >= 1; k --) {
      for (int alpha = 0; alpha < k; alpha ++) {
        f[u][k] = max(f[u][k], f[u][k-alpha] + f[i][alpha]);
      }
    }
  }
}

int main() {
    cin >> n >> m;
    for (int i = 1; i <= n; i ++) {
        cin >> p[i] >> a[i];
        bro[i] = son[p[i]];
        son[p[i]] = i;
    }
    dfs(0);
    cout << f[0][m + 1] << endl;
```

```
    return 0;
}
```

（二）方法二：

```
#include <iostream>

using namespace std;

typedef long long ll;
const int maxn = 3005;
ll n, m;
ll son[maxn], bro[maxn], a[maxn], p[maxn], f[maxn][maxn], sz[maxn];

void dfs(int u) {
    ll b = bro[u], s = son[u];
    f[u][0] = 0; f[u][1] = a[u];
    if (s != 0) dfs(s);
    if (b != 0) dfs(b);
    sz[u] = sz[s] + sz[b] + 1;
    for (int i = 1; i <= m; i ++) f[u][i] = f[b][i];
    for (int alpha = 0; alpha <= m; alpha ++) {
        for (int beta = 0; beta <= m; beta ++) {
            ll k = alpha + beta + 1;
            f[u][k] = max(f[u][k], f[s][alpha] + f[b][beta] +
a[u]);
        }
    }
}

int main() {
    cin >> n >> m;
    for (int i = 1; i <= n; i ++) {
        cin >> p[i] >> a[i];
        bro[i] = son[p[i]];
        son[p[i]] = i;
    }
    dfs(son[0]);
    cout << f[son[0]][m] << endl;
    return 0;
}
```

P3177 [HAOI2015]树上染色

题目描述

有一棵点数为 N 的树，树边有边权。给你一个在 0~ N 之内的正整数 K ，你要在这棵树中选择 K个点，将其染成黑色，并将其他 的N-K个点染成白色 。 将所有点染色后，你会获得黑点两两之间的距离加上白点两两之间的距离的和的受益。问受益最大值是多少。

输入输出格式

输入格式：

第一行包含两个整数 N, K 。接下来 N-1 行每行三个正整数 fr, to, dis ， 表示该树中存在一条长度为 dis 的边 (fr, to) 。输入保证所有点之间是联通的。

输出格式：

输出一个正整数，表示收益的最大值。

题解

递推公式建立

定义$f(u,x)$表示在以u为根的子树当中，有x个节点被染黑，在u处对答案的贡献。有：

$$f(u,x)=\max_{0\le j\le x,\ v\in u.son}\{f(u,x),\ f(u,x-j)+f(v,j)+val\} \tag{15}$$

$$val=w[e]*\{j*(k-j)+(|v.G|-j)*[(n-k)-(|v.G|-j)]\} \tag{16}$$

$$|v.G|=size[v],\ e=(u,v) \tag{17}$$

代码：

```cpp
#include <iostream>
#include <cstring>
using namespace std;

typedef long long ll;
const int maxn = 4005;

struct edge {
    int to, next, w;
} g[maxn];

ll n, k, ecnt = 0, head[maxn], f[maxn][maxn], sz[maxn];

void addEdge(int u, int v, int w) {
    g[ecnt].to = v;
    g[ecnt].w = w;
    g[ecnt].next = head[u];
    head[u] = ecnt ++;
}

void dfs(int u, int p) {
    sz[u] = 1;
```

```
    f[u][0] = f[u][1] = 0;
    // 子树是一个一个添加（树型背包的第一种添加子树的方法）
    for (int e = head[u]; e != -1; e = g[e].next) {
        int v = g[e].to;
        if (v != p) {
            dfs(v, u);
            sz[u] += sz[v];
            for (int x = sz[u]; x >= 0; x --) {
                for (int y = sz[v]; y >= 0; y --) {
                    // 特判：这个节点可行（其子树的节点够）
                    if (f[u][x] != -1) {
/*  val表示：
（新的子树中的黑点个数×除此之外的所有黑点个数+新的子树中的白点个数×除此之外的白点个数） ==> 必将经过下面那条边的次数
                                        ×
          （这些所有次数必定将要进过的一条边：就是u，v之间的边，边权为w[e]）
                                        =
                这个新的子树添加时，与u连接的这条边对答案的贡献
*/
  ll val = g[e].w * (y * (k - y) + (sz[v] - y) * ((n - k) - (sz[v] - y)));
/* 所以下面的式子可以转换为更加易于理解的式子：
f[u][x] = max{f[u][x], f[u][x-k] + f[v][k] + val}, 其中：0 ≤ k ≤ x; v为u的一个儿子
我们不难发现，节点是在dfs的过程中，一 个 一 个 添加的，所以
1. f[u][x-k]表示的是之前（没有添加现在子树的时候），以u为根的子树有x-k个点染黑对答案的贡献
2. f[v][k]表示的是以v为根的子树（由于dfs的顺序，此时这个子树是完全的），有k个节点被染黑对答案，到v的贡献
3. val表示的是为了补全（2）当中的那些染黑的节点，在u -> v的贡献
*/
  f[u][x + y] = max(f[u][x + y], f[u][x] + f[v][y] + val);
                    }
                }
            }
        }
    }
}

int main() {
    memset(head, -1, sizeof(head));
    memset(f, -1, sizeof(f));
    cin >> n >> k;
    for (int i = 1; i < n; i ++) {
        int from, to, dis;
        cin >> from >> to >> dis;
        addEdge(from, to, dis);
        addEdge(to, from, dis);
    }
```

```
    dfs(1, 0);
    cout << f[1][k] << endl;
    return 0;
}
```