307-13-14-子集枚举DP

P3959 宝藏

题目描述

参与考古挖掘的小明得到了一份藏宝图，藏宝图上标出了n个深埋在地下的宝藏屋， 也给出了这n个宝藏屋之间可供开发的m条道路和它们的长度。

小明决心亲自前往挖掘所有宝藏屋中的宝藏。但是，每个宝藏屋距离地面都很远， 也就是说，从地面打通一条到某个宝藏屋的道路是很困难的，而开发宝藏屋之间的道路 则相对容易很多。

小明的决心感动了考古挖掘的赞助商，赞助商决定免费赞助他打通一条从地面到某 个宝藏屋的通道，通往哪个宝藏屋则由小明来决定。

在此基础上，小明还需要考虑如何开凿宝藏屋之间的道路。已经开凿出的道路可以 任意通行不消耗代价。每开凿出一条新道路，小明就会与考古队一起挖掘出由该条道路 所能到达的宝藏屋的宝藏。另外，小明不想开发无用道路，即两个已经被挖掘过的宝藏 屋之间的道路无需再开发。

新开发一条道路的代价是：$L \times K$

L代表这条道路的长度，K代表从赞助商帮你打通的宝藏屋到这条道路起点的宝藏屋所经过的 宝藏屋的数量（包括赞助商帮你打通的宝藏屋和这条道路起点的宝藏屋）。

请你编写程序为小明选定由赞助商打通的宝藏屋和之后开凿的道路，使得工程总代 价最小，并输出这个最小值。

输入格式

第一行两个用空格分离的正整数n, m，代表宝藏屋的个数和道路数。

接下来m行，每行三个用空格分离的正整数，分别是由一条道路连接的两个宝藏 屋的编号（编号为$1 \sim n$），和这条道路的长度 v。

输出格式

一个正整数，表示最小的总代价。

输入 #1 复制

```
4 5
```

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

输出 #1 复制

```
4
```

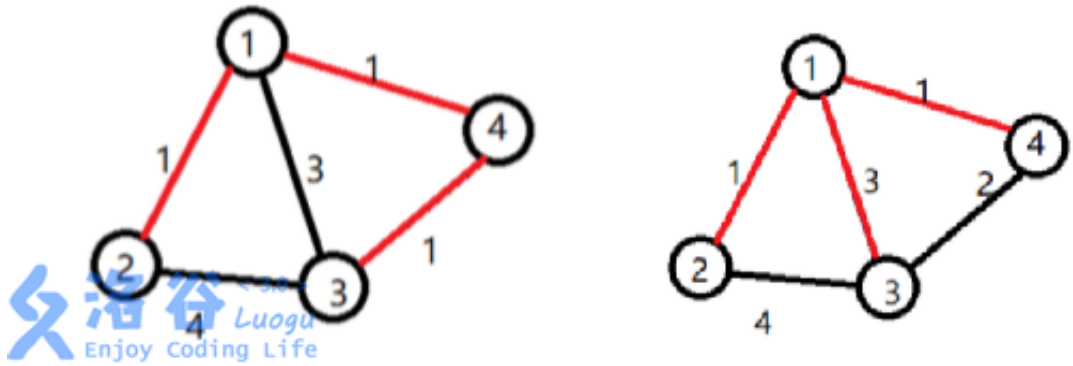
输入 #2 复制

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

输出 #2 复制

```
5
```

说明/提示



【样例解释1】

小明选定让赞助商打通了1 11 号宝藏屋。小明开发了道路$1 \to 2$，挖掘了2号宝 藏。开发了道路$1 \to 4$，挖掘了4号宝藏。还开发了道路$4 \to 3$，挖掘了3号宝 藏。工程总代价为：
$1 \times 1 + 1 \times 1 + 1 \times 2 = 4$

【样例解释2】

小明选定让赞助商打通了1号宝藏屋。小明开发了道路$1 \to 2$，挖掘了2号宝 藏。开发了道路$1 \to 3$，挖掘了3号宝藏。还开发了道路$1 \to 4$，挖掘了4号宝 藏。工程总代价为：
$1 \times 1 + 3 \times 1 + 1 \times 1 = 5$

【数据规模与约定】

对于20%的数据： 保证输入是一棵树，$1 \leq n \leq 8$，$v \leq 5000$且所有的v都相等。

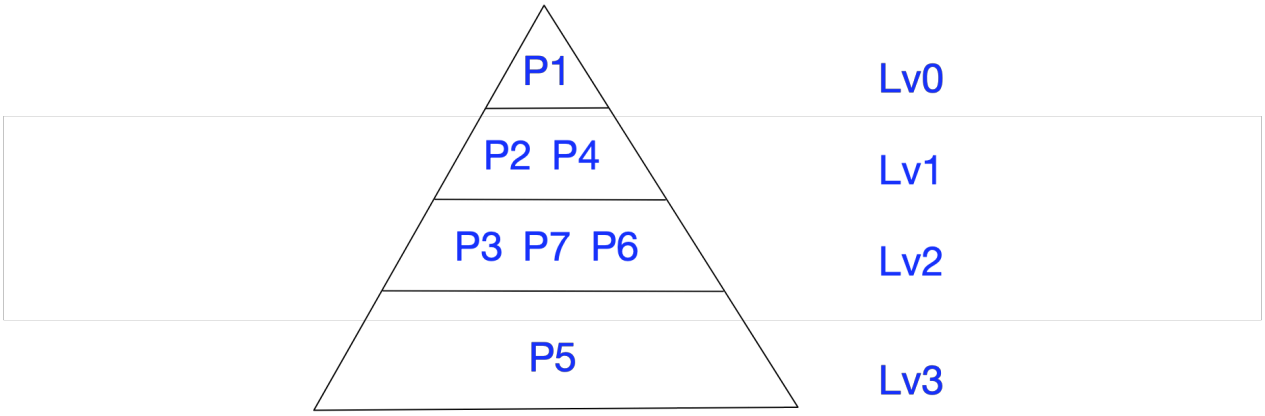
对于40%的数据：$1 \leq n \leq 81$，$0 \leq m \leq 10000$，$v \leq 5000$且所有的v都相等。

对于70%的数据：$1 \leq n \leq 8$，$0 \leq m \leq 1000$，$v \leq 5000$

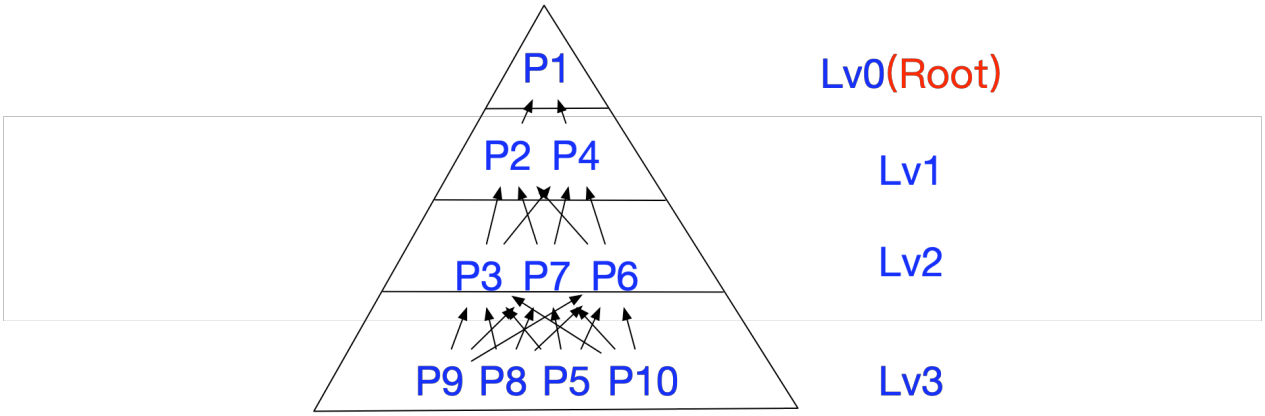
对于100%的数据：$1 \leq n \leq 12$，$0 \leq m \leq 1000$，$v \leq 500000$

题解

我们可以想到，将整张图分为一块一块的（分层图），类似于下面这样的金字塔的形状：



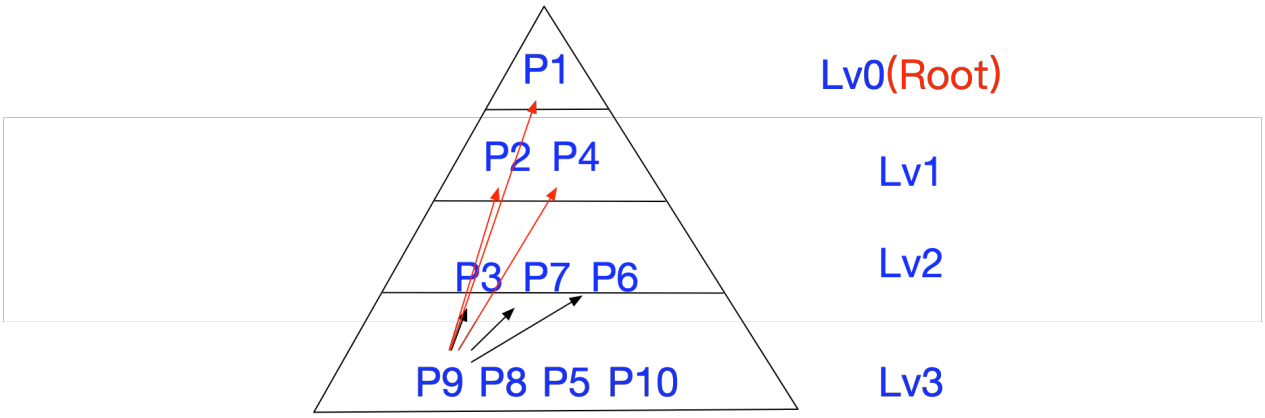
这样做的意义是我们只允许每层的元素与上一层相连，就达到了题目的用意，这样每个节点的$L \times K$就是$w(u,v) \times lv,\ u \in S_{lv-1},\ v \in S_{lv}$，其中$lv0$的只能有一个节点，就是赞助商免费送的根，总结下来可以得到下面这张构图的结论：



定义$f(lv,S)$，S为一个点集，$f(lv,S)$表示的是截至到$0 \sim lv$层中所有点分别为S的这$lv+1$层的总共的花费是多少。那么会有以下的式子：（枚举T为S的真子集）

$$f(lv,S)=\min_{T \subset S}\left\{f(lv-1,S-T)+lv \times b\sum_{u \in T}\min_{a \in S-T}w(u,a)\right\} \tag{1}$$

但是仔细观察上式，还是存在问题，就是$\sum_{u \in T}\min_{a \in S-T}w(u,a)$的部分，其会出现下图的情况（单单对于$P9$而言的图）：



换句话说，其可能不止访问上一层的，还会访问更加上面的点。但是我们分析后会发现，这无伤大雅，因为我们假设访问上面的点为u，层数为lv'，且$lv' < lv - 1$，则此时的费用为：

$$L' \times K = w(u, P_9) \times lv > w(u, P_9) \times (lv' + 1) \tag{2}$$

很好理解，这种状态不是最优的，所以在方程迭代的过程中，这种状态会被更好地状态所替代。

接下来，就是研究方程的边界问题：

$$f_{i \in V}(0, \{i\}) = 0,\ f_{S \subset V,\ |S| \geq 2}(0, S) = \infty \tag{3}$$

或者我们可以化为：

$$f(i, S) = \begin{cases} 0 & i \in V, S = \{i\} \\ \infty & \text{otherwise} \end{cases} \tag{4}$$

接下来我们可以思考一下这个方程的实现，因为表示集合与判断属于或者包含的关系并不是一件简单的事情。最终，我们想到可以使用二进制数来表示集合，例如$1,\ 2,\ 3,\ 7,\ 8$都有，$4,\ 5,\ 6$没有的集合可以表示为：

$$S = (\ 11000111\ )_2$$
$$\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow$$
$$87654321$$

用这种思想，我们可以得到以下的结论：

1. 要判断T是否为S的子集，只需要判断：
   $(S \mid T) == S$即可。
2. 如果要判断e是否在集合S当中，只需要判断
   $(S \mid (1 << e)) == S$即可。
3. 想要求$S - T$这个集合：
   $S - T = S \hat{} T$。
4. 迭代递归求子集：

$T = (T - 1)\ \&\ S$，其中T为子集S为全集，T的初始值为S，这样可以枚举S的所有子集，时间复杂度为$O(3^n)$而不是$O(4^n)$。

所以可以基于方程，给出代码：

```cpp
#include <iostream>
#define inf 1e9;
using namespace std;

typedef long long ll;

ll n, m, w[13][(1 << 12) + 5], f[13][(1 << 12) + 5];

int main() {
    cin >> n >> m;
    if (n == 1 && m == 0) {
        cout << 0 << endl;
        return 0;
    }
    for (ll i = 0; i < 13; i ++)
        for (ll s = 0; s < (1 << n); s ++)
            w[i][s] = inf;
    while (m --> 0) {
        ll u, v, len; cin >> u >> v >> len;
        u --; v --;
        w[u][1 << v] = min(w[u][1 << v], len);
        w[v][1 << u] = min(w[v][1 << u], len);
    }
    for (ll i = 0; i < n; i ++)
        for (ll s = 1; s < (1 << n); s ++) {
            ll lb = s & (-s);
            w[i][s] = min(w[i][s - lb], w[i][lb]);
        }
    for (ll i = 0; i < n; i ++) {
        for (ll s = 0; s < (1 << n); s ++)
            f[i][s] = inf;
    }
    for (ll j = 0; j < n; j ++)
        f[0][1 << j] = 0;
    for (ll lv = 1; lv < n; lv ++) {
        for (ll s = 0; s < (1 << n); s ++) {
            for (ll t = s - 1; t > 0; t = (t - 1) & s) {
                ll sum = 0;
                for (ll x = 0; x < n; x ++)
                    if ((t | (1 << x)) == t) sum += w[x][s ^ t];
                f[lv][s] = min(f[lv][s], f[lv - 1][s ^ t] + lv *
sum);
            }
        }
```

```
    }
    ll ans = inf;
    for (ll i = 1; i < n; i ++)
        ans = min(ans, f[i][(1 << n) - 1]);
    cout << ans << endl;
    return 0;
}
```

P3052 [USACO12MAR]摩天大楼里的奶牛Cows in a Skyscraper

题目描述

A little known fact about Bessie and friends is that they love stair climbing races. A better known fact is that cows really don't like going down stairs. So after the cows finish racing to the top of their favorite skyscraper, they had a problem. Refusing to climb back down using the stairs, the cows are forced to use the elevator in order to get back to the ground floor.

The elevator has a maximum weight capacity of W (1 <= W <= 100,000,000) pounds and cow i weighs C_i (1 <= C_i <= W) pounds. Please help Bessie figure out how to get all the N (1 <= N <= 18) of the cows to the ground floor using the least number of elevator rides. The sum of the weights of the cows on each elevator ride must be no larger than W.

给出n个物品，体积为w[i]，现把其分成若干组，要求每组总体积<=W，问最小分组。(n<=18)

输入输出格式

输入格式：

* Line 1: N and W separated by a space.

* Lines 2..1+N: Line i+1 contains the integer C_i, giving the weight of one of the cows.

输出格式：

* A single integer, R, indicating the minimum number of elevator rides needed.

one of the R trips down the elevator.

输入输出样例

输入样例#1： 复制

```
4 10
5
6
3
7
```

输出样例#1： 复制

```
3
```

说明

There are four cows weighing 5, 6, 3, and 7 pounds. The elevator has a maximum weight capacity of 10 pounds.

We can put the cow weighing 3 on the same elevator as any other cow but the other three cows are too heavy to be combined. For the solution above, elevator ride 1 involves cow #1 and #3, elevator ride 2 involves cow #2, and elevator ride 3 involves cow #4. Several other solutions are possible for this input.

题解

这道题使用一种另类的贪心想法，这种贪心的转移状态比较特殊，在这里设了两个转移方程，分别为：

$$\begin{bmatrix} f(S) \\ g(S) \end{bmatrix} = \min_{u \in S} \begin{cases} \begin{bmatrix} f(\complement_S\{u\}) \\ g(\complement_S\{u\}) + w(u) \end{bmatrix} & g(\complement_S\{u\}) + w(u) \le C \\ \begin{bmatrix} f(\complement_S\{u\}) + 1 \\ w(u) \end{bmatrix} & \text{otherwise} \end{cases} \tag{5}$$

其中是将$\begin{bmatrix} f(S) \\ g(S) \end{bmatrix}$定义为一个关于$S$矩阵。$f(S)$代表以$S$为集合的牛已经关闭的电梯的数量，即电梯不再对牛开放，已经完成运送了，$g(S)$表示这集合S当中，除了$f(S)$中已经被送走的牛，剩下的牛的重量。注：其中，一定存在$g(S) < C$（没有一头的重量超过C）。

有了上述定义，这个方程非常容易理解，所以只需要进行子集枚举的背包就可以了，故时间复杂度为$O(n3^n)$。

AC代码：

```cpp
#include <iostream>
#include <algorithm>

using namespace std;
pair<int, int> f[1 << 23];
int n, c, w[1 << 23];

```

```cpp
int main() {
    cin >> n >> c;
    int all = (1 << n) - 1;
    for (int i = 0; i < n; i ++)
        cin >> w[i];
    for (int s = 1; s <= all; s ++) {
        f[s] = make_pair(n, 0);
        for (int u = 0; u < n; u ++) {
            if (s & (1 << u)) {
                int x = f[s ^ (1 << u)].first;
                int y = f[s ^ (1 << u)].second;
                if (y + w[u] <= c) {
                    f[s] = min(f[s], make_pair(x, y + w[u]));
                } else {
                    f[s] = min(f[s], make_pair(x + 1, w[u]));
                }
            }
        }
    }
    cout << f[all].first + (f[all].second ? 1 : 0) << endl;
    return 0;
}
```