# Contents

# 1 ZJNU - 2268 0/1 分数规划

## 1.1 题意

**Figure 1.1:** 图片测试

给 $n$ 个对象的 $w_i$ 值和 $t_i$ 值，和重量下界 $W$ ，取其中任意个对象，求：

$$Max(\frac{\sum t_k \times 1000}{\sum w_k})(\sum w_k > W, k \in \{1 \dots N\})$$

## 1.2 思路

0/1 分数规划 预备知识

1 把题目公式化以后，我们可以做以下变形：

$$V = Max(\frac{\sum t_k \times 1000}{\sum w_k})$$

$$V_{Max} = \frac{\sum t_k \times 1000}{\sum w_k}$$

$$V_{Max} \times \sum w_k = \sum t_k \times 1000$$

$$\sum t_k \times 1000 - V_{Max} \times \sum w_k = 0$$

本题的目标是求 $V_{Max}$。假如有一个可能的答案 $V_t$，

$$\sum t_k \times 1000 - V_t \times \sum w_k = \Delta x$$

先不提如何选出对象，假如最后 $\Delta x \geq 0$ 说明此解可行，或存在更优解；反之，此解不可行。我们可以通过二分的方法来实现这一过程。

对于选择的过程，可以通过背包来检验答案。设 $dp[i]$ 表示重量为 $i$ 的对象组合，所具有的 $\Delta x$ 的最大值。但这里我们**只需要** 知道有没有一种可能，使得当前的解可行，所以计算最大值即可。

注意：$\Delta x$ 可能有负值，所以初始化要是 $-\infty$。

$$dp[k] = Max(dp[j] + t[i] \times 1000 + V_t \times w[i], dp[k]), (j \in \{W \dots 0\})$$

注意到重量的数据范围有 1000000，但是在这个问题中，限制我们的只有 $W$（范围只到 1000）。

对于小于 $W$ 的组合，我们需要通过动态规划递推出一些组合，直到它们的重量超过 $W$；对于超过 $W$ 的那些组合，都放在 $dp[W]$ 更新也可以，因为我们我们只是在找能不能用当前 $V_t$ 得到非负的 $\Delta x$。

综上，**写一个二分，在 *check* 函数里背包一下就可以了。（一句话题解）**

> 比如，$V_t \times w_i$ 的乘法可能会爆 int 之类的。

复杂度：$NW\log_2(K)$

## 1.3 代码

```
// 看看中文注释的情况
#include <iostream>
#include <algorithm>
using namespace std;
typedef long long ll;
const int MAXN = 257;
const ll INF = 0x3f3f3f3f;
int n,W;
int ans;
int w[MAXN],t[MAXN];
ll dp[1007];

bool check(ll v){
    dp[0]=0;
    for(int i=1;i<=W;i++)
        dp[i]=-INF;
    for(int i=1;i<=n;i++)
        for(int k,j=W;j>=0;j--)
            k=min(j+w[i],W),
            dp[k]=max(dp[j]+t[i]-v*w[i],dp[k]);
    if(dp[W]>=0)return true;
    return false;
}
int main(){
    ios::sync_with_stdio(0);
    cout.tie(0);cin.tie(0);
    cin>>n>>W;
    for(int i=1;i<=n;i++){
        cin>>w[i]>>t[i];
        t[i]*=1000;
    }
```

```
    ll mid,l=0,r=250005;
    while(l<r){
        mid=(l+r+1)>>1;
        if(check(mid)){
            l=mid;
        }else{
            r=mid-1;
        }
    }
    cout<<l<<endl;
    return 0;
}


/* Author: bnfcc -> tc2000731 -> tieway59
 * Description:
 *      维护下凸包，对于每个 x 维护 f(x)=k*x+b 的最大值。
 *      query max value within all f(x) functions.
 *      c++11 features included.
 * Problems:
 *      https://nanti.jisuanke.com/t/41306
 *      https://nanti.jisuanke.com/t/41097
 */
template<typename var=long long, const int SIZE = 1000005, typename
 ↪  ldb=long double>
struct Hull {
    struct fx {
        var k, b;

        fx() {}

        fx(var k, var b) : k(k), b(b) {}

        var f(var x) { return k * x + b; }
    };

    int cnt;
    fx arr[SIZE];
```

```
bool empty() {
    return cnt == 0;
}

void init() {
    cnt = 0;
}

void add(const fx &p) {
    arr[cnt++] = p;
}

void pop() {
    cnt--;
}

bool chek(const fx &a, const fx &b, const fx &c) {
    ldb ab, ak, bb, bk, cb, ck;
    tie(ab, ak, bb, bk, cb, ck) =
            tie(a.b, a.k, b.b, b.k, c.b, c.k);
    return (ab - bb) / (bk - ak) > (ab - cb) / (ck - ak);
}

void insert(const fx &p) {///k 从小到大插入
    if (cnt && arr[cnt - 1].k == p.k) {
        if (p.b <= arr[cnt - 1].b)return;
        else pop();
    }
    while (cnt >= 2 && chek(arr[cnt - 2], arr[cnt - 1], p))pop();
    add(p);
}

/*var query(var x) {///x 从大到小查询          从小到大用队列
   while (cnt > 1 && arr[cnt - 2].f(x) > arr[cnt - 1].f(x))pop();;
    return arr[cnt - 1].f(x);
}*/
```

```
    var query(var x) {///二分查询, x 顺序任意
        int l = 0, r = cnt - 1;
        while (l < r) {
            int mid = (l + r) >> 1;
            if (arr[mid].f(x) >= arr[mid + 1].f(x))r = mid;
            else l = mid + 1;
        }
        return arr[l].f(x);
    }
};

// vector stack
template<typename var=long long, const int SIZE = 1000005, typename
 ↪ ldb=long double>
struct Hull {
    struct Line {
        var k, b;

        Line() {}

        Line(var k, var b) : k(k), b(b) {}

        var f(var x) { return k * x + b; }
    };

    int cnt;
    vector <Line> con;//

    bool empty() {
        return cnt == 0;
    }

    void init(const int &n) {
        con.clear();
        if (n > con.capacity())con.reserve(n);
        cnt = 0;
    }
```

```
    void add(const Line &p) {
        con.emplace_back(p);
        cnt++;
    }

    void pop() {
        cnt--;
        con.pop_back();
    }

    bool chek(const Line &a, const Line &b, const Line &c) {
        ldb ab, ak, bb, bk, cb, ck;
        tie(ab, ak, bb, bk, cb, ck) =
                tie(a.b, a.k, b.b, b.k, c.b, c.k);
        return (ab - bb) / (bk - ak) > (ab - cb) / (ck - ak);
    }

    void insert(const Line &p) {///k 从小到大插入
        if (cnt && con[cnt - 1].k == p.k) {
            if (p.b <= con[cnt - 1].b)return;
            else pop();
        }
        while (cnt >= 2 && chek(con[cnt - 2], con[cnt - 1], p))pop();
        add(p);
    }

    var query(var x) {///二分查询, x 顺序任意
        int l = 0, r = cnt - 1;
        while (l < r) {
            int mid = (l + r) >> 1;
            if (con[mid].f(x) >= con[mid + 1].f(x))r = mid;
            else l = mid + 1;
        }
        return con[l].f(x);
    }
};

Hull<> hull;
```