Người viết: Lê Đôn Khuê

21/04/2008

Giải Bài Khu vườn - GARDEN

Gọi Ax,y là số cây hoa trong ô (x, y).

Gọi Rx,y là số tổng số cây hoa nằm trong khoảng hình chữ nhật ô trái dưới là (1, 1) ô phải trên là (x, y). Ta sẽ tính được Rxy theo công thức:

- Rx,y = 0 với (x = 0 hoặc y = 0)
- Rx,y = Rx-1,y-1 – Rx,y-1 – Rx-1,y + Ax,y

Như vậy ta có thể tính số hoa nằm trong khoảng hình chữ nhật với ô trái dưới là (x1, y1), ô phải trên là (x2, y2) trong O(1) theo công thức:

Rx2,y2 – Rx2,y1-1 – Rx1-1,y2 + Rx1-1,y1-1

Để qua được 50% số test, bạn chỉ cần duyệt các cạnh của 2 hình chữ nhật (độ phức tạp là O(w4l4). Nhưng để qua được hết tất cả các test, bạn cần phải có một thuật toán tốt hơn như sau:

Ta có nhận xét là 2 hình chữ nhật không giao nhau thì sẽ tồn tại ít nhất một đường thẳng (ngang hoặc dọc) chia mảnh vườn thành 2 phần trong đó mỗi phần có một hình chữ nhật được chọn.

Như vậy, ta sẽ duyệt mọi hình chữ nhật thỏa mãn trong đó có k cây hoa. Nhưng với thuật toán duyệt O(w2l2) sẽ không đáp ứng được thời gian, vì vậy ta có một cách duyệt tốt hơn như sau:

Duyệt 2 cạnh trên (y2) và dưới (y1) của hình chữ nhật. Chúng ta sẽ xét các hình chữ nhật có 2 cạnh như thỏa mãn điều kiện đó.

Với những hình chữ nhật có cạnh trái x1, cạnh phải x2 ta sẽ xét các trường hợp sau:

- Nếu số lượng hoa = k thì ghi nhận hình chữ nhật (x1, y1, x2, y2)
- Nếu số lượng hoa < k thi` ta(ng x2.
- Nếu số lượng hoa > k thì tăng x1.

Như vậy thuật toán duyệt sẽ có độ phức tạp (w2l). Các phần còn lại chỉ là các bước đơn giản.

Dưới đây là chương trình đã được 100 điểm của bài toán này trong online contest.

```
const
max = 251;
INF = $3F3F3F3F;

var
fi, fo: text;
m, n, k: longint;
cot, a: array [0..max, 0..max] of longint;
dai, left, right, up, down: array [0..max] of longint;

procedure read_data;
var p, x, y, i: longint;
begin
readln(n, m);
readln(p, k);
for i := 1 to p do
begin
readln(y, x);
inc(a[x, y]);
end;
end;

procedure init;
var i, j, u, v: longint;
begin
fillchar(left,sizeof(left), $3F);
fillchar(right, sizeof(right), $3F);
fillchar(up,sizeof(up), $3F);
fillchar(down,sizeof(down), $3F);
for i := 1 to m do
for j := 1 to n do
cot[i, j] := cot[i - 1, j] + a[i, j];
```

```
end;

procedure update(x, y, i, j: longint);
var p: longint;
begin
p := 2 * (y - x + 1) + 2 * (j - i + 1);
if down[x] > p then down[x] := p;
if up[y] > p then up[y] := p;
if right[i] > p then right[i] := p;
if left[j] > p then left[j] := p;
end;

procedure process;
var s, i, j, x, y: longint;
begin
for x := 1 to m do
for y := x to m do
begin
for i := 1 to n do dai[i] := cot[y, i] - cot[x - 1, i];
i := 1;
j := i;
s := dai[1];
while i <= n do
begin
while (s < k) and (j < n) do
begin
inc(j);
s := s + dai[j];
end;
if s = k then update(x, y, i, j);
s := s - dai[i];
inc(i);
end;
end;

procedure write_result;
var i, best: longint;
begin
for i := 2 to n do
if left[i - 1] < left[i] then left[i] := left[i - 1];
for i := n - 1 downto 1 do
if right[i] > right[i + 1] then right[i] := right[i + 1];
for i := 2 to m do
if up[i] > up[i - 1] then up[i] := up[i - 1];
for i := m - 1 downto 1 do
if down[i] > down[i + 1] then down[i] := down[i + 1];

best := INF + INF - 1;
for i := 1 to n - 1 do
if left[i] + right[i + 1] < best then
best := left[i] + right[i + 1];
for i := 1 to m - 1 do
if up[i] + down[i + 1] < best then
best := up[i] + down[i + 1];
if best < INF + INF - 1 then writeln(best) else writeln('NO');
end;

begin
read_data;
```

```
init;
process;
write_result;
end.
```

Giải Bài MEAN SEQUENCE - Dãy trung bình

Gọi min[i] và max[i] là miền giá trị của số S[i] để thỏa mãn dãy m[1] đến m[i – 1]. Ta có m[i – 1] ≤ min[i] ≤ max[i] ≤ m[i] với i > 1. (miền giá trị bao gồm cả 2 đầu mút) Ta khởi tạo min[1] = -∞, max[1] = m[1].

Khi đã biết min[i - 1], max[i – 1] ta sẽ tính được min[i], max[i] bằng cách lấy đối xứng đoạn min[i – 1], max[i – 1] qua m[i – 1], rồi lấy giao của đoạn min[i], max[i] với đoạn [m[i], +∞). Nếu miền giá trị của S[i] là rỗng thì kết luận luôn đáp số bằng 0. Cuối cùng đáp số sẽ là max[n + 1] - min[n + 1] + 1.

Tuy nhiên vấn đề là chúng ta không làm được mảng 5 000 000 phần tử longint. Chúng ta có thể nhận thấy, min[i], max[i] đều được tính từ min[i – 1], max[i – 1] nên chúng ta có thể dùng các biến đơn luân phiên nhau. Như vậy độ phức tạp là O(n) bộ nhớ O(1). Dưới đây là chương trình minh họa:

```
const
maxN = 5000005;
INF = 1111111111;
var n, res: longint;
procedure process;
var i: longint;
t, bi, bi1, mini, maxi, mini1, maxi1: int64;
x: extended;
begin
res := maxlongint;
readln(n);
readln(x);
bi1 := trunc(x * 2);
mini1 := -INF;
maxi1 := bi1 shr 1;
for i := 2 to n do
begin
readln(x);
bi := trunc(x * 2);
mini := bi1 - maxi1;
maxi := bi1 - mini1;
if 2 * mini - bi > 0 then
begin
writeln('0');
exit;
end;
t := bi shr 1;
if maxi > t then maxi := t;
maxi1 := maxi;
mini1 := mini;
bi1 := bi;
end;
res = maxi - mini + 1;
writeln(res);
end;

begin
process;
end.
```

Giải Bài MOUTAINS - Hành trình qua núi

Đây là bài toán thiên về cấu trúc dữ liệu, cụ thể là interval tree. Mỗi nút trên cây sẽ lưu thông tin về một đoạn đường ray liên tiếp. Với nút P lưu thông tin về đoạn [L, R], nút P sẽ lưu SP là chênh lệch giữa điểm R và L. HP là độ cao lớn nhất trong đoạn [L, R]. Nếu tính gốc là điểm L.

Nút P sẽ là lá nếu mọi di đều bằng nhau. Nếu trường hợp nút P không phải là lá, thì P sẽ có 2 nút con là P1 và P2. P1 lưu thông tin về đoạn [L, (L + R) div 2], P2 lưu thông tin về đoạn [(L + R) div 2 + 1, R]. Ta sẽ có SP = SP1 + SP2; HP = max(HP1, SP1 + HP2). Nếu gọi M là số yêu cầu, cách làm này có độ phức tạp là O(M*logN), tuy nhiên bộ nhớ lên đến O(N*logN). Vì giới hạn N là 1 tỷ, ta sẽ không thể lưu được. Ta sẽ phải tìm cách khác để lưu. Ta để ý thấy, M ≤ 100 000. Như vậy, sẽ có không quá 200 000 đầu mút các đoạn ray. Ta chỉ cần sửa đổi một chút, nút P lưu đoạn từ L đến R được sửa lại thành đoạn từ đầu mút thứ L đến đầu mút thứ R. Dưới đây là chương trình mẫu cho bài này. Đây là chương trình viết bằng C++ của Max Zhou (thí sinh Trung Quốc trong cuộc thi IOI online)

```
#include
#include
#include
#define INFINITY 1000000000

struct Titem {
int oper;
int L, R, v;
};

struct Tnode {
int L, R;
int fm, sum, v;
bool used;
int Lch, Rch;
};

void init();
void prefix();
void solve();
void createtree(int r, int L, int R);
void inserttree(int r, int L, int R, int v);
int findtree(int r, int limit);
inline int count(int r);

int n;

int nitem;
Titem item[110000];

int nlist, list[210000];

int nnode;
Tnode tree[410000];

int main() {
init();
prefix();
solve();
return 0;
}

void init() {
char line[1000];
int L, R, v;

scanf('%d', &n);
```

```
while ( scanf('%s', line)==1 )
if ( strcmp(line, 'I')==0 ) {
scanf('%d %d %d', &L, &R, &v);

item[nitem].oper=0; item[nitem].L=L; item[nitem].R=R; item[nitem].v=v;
++nitem;
}
else if ( strcmp(line, 'Q')==0 ) {
scanf('%d', &v);

item[nitem].oper=1; item[nitem].v=v;
++nitem;
}
else break;
}

void prefix() {
int i;

for ( i=0; i
if ( item[i].oper==0 ) {
list[nlist++]=item[i].L; list[nlist++]=item[i].R+1;
}
list[nlist++]=1; list[nlist++]=n+1;

std::sort(list, list+nlist);
nlist=std::unique(list, list+nlist)-list;

for ( i=0; i
if ( item[i].oper==0 ) {
item[i].L=std::lower_bound(list, list+nlist, item[i].L)-list;
item[i].R=std::lower_bound(list, list+nlist, item[i].R+1)-list-1;
}
}

void solve() {
int i;

tree[nnode].fm=-INFINITY; ++nnode;
createtree(1, 0, nlist-2);
for ( i=0; i
if ( item[i].oper==0 ) inserttree(1, item[i].L, item[i].R, item[i].v);
else {
int res = findtree(1, item[i].v);

printf('%dn', res);
}
}

void createtree(int r, int L, int R) {
tree[r].L=L; tree[r].R=R;
tree[r].fm=tree[r].sum=0; tree[r].used=false;
tree[r].Lch=tree[r].Rch=0;

if ( L
tree[r].Lch=++nnode; createtree(nnode, L, (L+R)/2);
tree[r].Rch=++nnode; createtree(nnode, (L+R)/2+1, R);
}
}
```

```
void update(int r) {
if ( tree[r].used ) {
tree[r].sum=count(r)*tree[r].v;
tree[r].fm=std::max(tree[r].v, tree[r].sum);
}
else {
int Lch = tree[r].Lch, Rch = tree[r].Rch;

tree[r].sum=tree[Lch].sum+tree[Rch].sum;
tree[r].fm=std::max(tree[Lch].fm, tree[Lch].sum+tree[Rch].fm);
}
}

void inserttree(int r, int L, int R, int v) {
if ( L<=tree[r].L && R>=tree[r].R ) {
tree[r].used=true; tree[r].v=v; update(r);
return;
}

int Lch = tree[r].Lch, Rch = tree[r].Rch;
if ( tree[r].used && tree[r].L
tree[Lch].used=true; tree[Lch].v=tree[r].v; update(Lch);
tree[Rch].used=true; tree[Rch].v=tree[r].v; update(Rch);
}
tree[r].used=false;

if ( L<=(tree[r].L+tree[r].R)/2 ) inserttree(Lch, L, R, v);
if ( R>(tree[r].L+tree[r].R)/2 ) inserttree(Rch, L, R, v);
update(r);
}

int findtree(int r, int limit) {
if ( tree[r].used || tree[r].L==tree[r].R )
if ( tree[r].v<=0 ) return count(r);
else return std::min(limit/tree[r].v, count(r));

int Lch = tree[r].Lch, Rch = tree[r].Rch;
if ( tree[tree[r].Lch].fm<=limit )
return count(Lch)+findtree(Rch, limit-tree[Lch].sum);
else
return findtree(Lch, limit);
}

inline int count(int r) {
int L = tree[r].L, R = tree[r].R;
return list[R+1]-list[L];
}
```

Giải bài BIRTHDAY - Ngày sinh nhật

Đề bài yêu cầu sắp xếp lại chỗ ngồi của những đứa trẻ sao cho khoảng cách mà đứa trẻ phải di chuyển xa nhất là ngắn nhất có thể được. Chúng ta nên lưu ý có 2 cách sắp xếp cuối cùng: thuận chiều kim đồng hồ và ngược chiều kim đồng hồ. Ta có nhận xét là hai trường hợp này được giải quyết như nhau, nên ta chỉ giải quyết trường hợp thuận chiều kim đồng hồ:

Gọi a[i] là vị trí ban đầu của những đứa trẻ. Chúng ta chuyển đứa trẻ thứ i đến vị trí i. Đặt d[i] là một số thể hiện hướng và khoảng cách đứa trẻ i phải di chuyển.

- |d[i]| = khoảng cách di chuyển của đứa trẻ thứ i.
- d[i] > 0 nếu di chuyển cùng chiều kim đồng hồ.
- d[i] < 0 nếu di chuyển ngược chiều kim đồng hồ.
- Nếu khoảng cách d[i] bằng N/2 (N chẵn) thì ưu tiên di chuyển thuận chiều kim đồng hồ.

Ta có: -((N – 1) div 2) ≤ d[i] ≤ N div 2
Nếu mỗi đứa trẻ được dịch sang phải một đơn vị, thì d[i] sẽ thay đổi theo
• Nếu d[i] ≤ N div 2 thì d[i] = d[i] + 1.
• Nếu d[i] = N div 2 thì d[i] = -((N – 1) div 2)
Nhiệm vụ của chúng ta là xoay cách đứa trẻ thêm một số lần để sao cho max(|d[i]|) là min. Chúng ta nhận thấy có thể tính được giá trị này trong thời gian O(n). Xét C là đoạn liên tiếp dài nhất thuộc [-((N – 1) div 2), N div 2] mà không có d[i] nào xuất hiện. Chú ý đoạn dài nhất có thể là [a, N div 2] hợp [-((N – 1) div 2), b] với b < a. Nếu C có P phần tử thì đáp số sẽ là (N – P) div 2.
Dưới đây là bài giải đã được 100 điểm.

```
#include
#include
#include
#define MAXN 1000001
#define Swap(x, y) ((x) ^= (y), (y) ^= (x), (x) ^= (y))
int a[MAXN], N, i, j;
int u[MAXN];

void Reverse(void)
{ int i;
for (i = 0; i <= (N - 2) / 2; i ++)
Swap(a[i], a[N - i - 1]);
}
int GoodMin(void)
{ int i, j, first, last, max_gap = 0;
memset(u, 0, sizeof(u));
for (i = 0; i < N; i ++)
{ j = a[i] - i;
while (j < 0) j += N;
u[j] = 1;
}
for (i = 0; u[i] == 0; i ++);
first = last = i;
for (++ i; i < N; i ++)
if (u[i])
{
if (i - last > max_gap) max_gap = i - last;
last = i;
}
if (N - last + first > max_gap)
max_gap = N - last + first;
return (N - max_gap + 1) / 2;
}

void GoodSolve(void)
{
int Ans1, Ans2;
Ans1 = GoodMin();
Reverse();
Ans2 = GoodMin();
printf("%dn", Ans1 < Ans2? Ans1 : Ans2);
}

int main(void)
{
// freopen("bir.in", "r", stdin);
scanf("%d", &N);
for (i = 0; i < N; i ++)
{
```

```
scanf("%d", &(a[i]));
-- a[i];
}
GoodSolve();
return 0;
}
```

Giải Bài RECTANGLE GAME - Trò chơi cắt hình chữ nhật

Ta có một vài định nghĩa sau:

• Một trạng thái của bảng được thể hiện bằng cặp số (n, m).

• Một trạng thái P được gọi là trạng thái thắng nếu ta luôn có cách thắng mặc cho bước tiếp theo đối thủ đi thế nào.

• Một trạng thái P được gọi là trạng thái thua nếu đối thủ luôn có cách thắng mặc cho các bước tiếp theo ta đi thế nào.

Theo định nghĩa trên:

• (1, 1) là trạng thái thua.

• Một trạng thái thắng nếu có tồn tại ít nhất một nước đi dẫn đến trạng thái thua.

• Một trạng thái thua mọi nước đi tiếp theo đều hướng đến một trạng thái thắng.

Chúng ta có thể lập một bảng như sau, vị trí i j là dấu * thì trạng thái (i, j) là trạng thái thua và ngược lại.

Sau khi phân tích, chúng ta có thể rút ra một nhận xét như sau:

Trạng thái (n, m) là trạng thái thua khi và chỉ khi $(m + 1) = 2^k (n + 1)$. Ta sẽ chứng minh nhận xét này:

Trường hợp $k = 0 \leftrightarrow m = n$.

• m = n = 1: (m, n) là trạng thái thua theo định nghĩa.

• m > 1: Nếu đối thủ đưa ta vào trạng thái (n, p) thì $m/2 \leq p < m$. Vậy ta có thể đi từ trạng thái (n, p) đến trạng thái (p, p) một trạng thái thua.

Trường hợp $k \neq 0$. Không mất tính tổng quát giả sử k > 0. Vì $(m + 1) = 2^k (n + 1) \leftrightarrow (n + 1) = 2^{-k}(m + 1)$. Khi đang ở trạng thái (n, m) ta sẽ có hai hướng: (p, m) hoặc (n, q):

• Nếu sau khi cắt, ta đến trạng thái (p, m). Ta có $n/2 \leq p < n$. Mà $n = 2^k (m + 1) – 1$ nên ta có bất đẳng thức sau: $2^{k-1}(m + 1) – 1 < p < 2^k(m + 1) – 1$. Như vậy, người tiếp theo có thể thực hiện nước đi từ trạng thái (p, m) đến trạng thái $(2^{k-1}(m+1) – 1, m)$.

• Nếu sau khi cắt, ta đến trạng thái (n, q) trong đó $m/2 \leq q < m$. Ta sẽ chứng minh không tồn tại số nguyên thỏa mãn $n = 2^i(q + 1) – 1$. Ta sẽ chứng minh bằng phản chứng. Giả sử tồn tại số nguyên i thỏa mãn đẳng thức $n = 2^i(q + 1) – 1$.

- Ta có $n = 2^k(m + 1) – 1$ và m > q nên i > k.

- Mặt khác, $m/2 \leq q$, ta có:

- $2^k(m + 1) – 1 = n = 2^i(q+1) – 1 \geq$

$2^i(m/2 + 1) – 1 = 2^{i -1}(m + 2) > 2i-1 (m + 1) – 1$

- $\leftrightarrow k > i – 1 \leftrightarrow i < k + 1$

i là số nguyên và k < i < k + 1 (vô lý).

Vì vậy, nếu $m + 1 = 2^k(n + 1)$ thì trạng thái này là trạng thái thua. Nhiệm vụ của chúng ta trong mỗi lần thực hiện một nước đi từ (n, m) đến (n’, m’) sao cho $m’ + 1 = 2^i (n’ + 1)$. Không mất tính tổng quát, ta giả sử $n \geq m$. Ta sẽ có chọn m’, n’ sao cho:

• m’ = m

• $n’ = 2^k (m’ + 1) – 1$, lớn nhất và nhỏ hơn n.

Độ phức tạp của thuật toán trong trường hợp xấu nhất là O(n log n)

```
uses preclib;
var
x, y, m: longint;
a: array [1..100] of longint;

procedure make(u, v: longint);
begin
if ođ(v) then
v := v div 2 + 1
else
v := v div 2;
m := 1;
a[1] := u;
while a[m] < v do
begin
inc(m);
```

```
a[m] := a[m - 1] * 2 + 1;
end;
end;

begin
while true do
begin
x := dimension_x;
y := dimension_y;
if (x > y) and (2 * y >= x) then
begin
cut(vertical, y);
continue;
end;
if (y > x) and (2 * x >= y) then
begin
cut(horizontal, x);
continue;
end;
if x > y then make(y, x) else make(x, y);
if x > y then
cut(vertical, a[m])
else
cut(horizontal, a[m]);
end;
end.
```

Giải Bài RIVER - Các dòng sông

Đây là một bài toán quy hoạch động trên cây khá hay. Trước tiên, chúng ta cần xây dựng một cây mà mỗi nút trên cây, chỉ có duy nhất một đường đến gốc (Byteland town). v là cha của u nếu v là ngôi làng đầu tiên trên đường đi từ u đến gốc.

Gọi r là gốc của cây (Byteland town). Đặt depth(u) là số cạnh trên đường đi từ u đến r. Ta có thể tính được depth(u) trong O(n). Đặt deg(u) là số con của u và số cây gỗ bị cắt ở nút u là trees(u).

Hàm quy hoạch động A[v, t, p] với ý nghĩa chi phí ít nhất của việc xây t nhà máy trong cây con có gốc là v và những cây gỗ mà không được xử lý ở trong cây con v, thì sẽ trôi đến nút p (p nằm trên đường từ v đến r). Chúng ta sẽ tính A[v, t, p] với mỗi v và $0 \le t \le k$. Ta sẽ tính được A[v, t, p] theo công thức sau:

• 0 nếu cây con gốc v có đúng t nút

• min(A1[v, t, p], A2[v, t]) trong trường hợp còn lại

trong đó

• A1[v, t, p] là chi phí ít nhất của việc xây dựng t nhà máy trong cây con có gốc là v, không xây nhà máy tại nút v và những cây gỗ mà không được xử lý ở trong cây con v, thì sẽ trôi đến nút p (p nằm trên đường từ v đến r).

• A2[v, t] là chi phí ít nhất của việc xây dựng t nhà máy trong cây con có gốc là v và tại v có xây một nhà máy.

Gọi d = deg(v) và vi là các con của v. Ta sẽ tính A1[v, t, p] và A2[v, t] theo công thức sau:

• A1[v, t, p] = trees(v) * khoảng cách(v, p) + min($\sum$i=1..d A[v_i, t_i, p]) thỏa mãn $\sum t_i = t$.

• A2[v, t] = min ($\sum$i=1..d A[v_i, t_i, depth(v)]) thỏa mãn $\sum t_i = t - 1$.

Nhìn vào 2 công thức trên, ta sẽ thấy ngay rằng, chúng ta cần phải xét mọi cách chia t nhà máy vào các cây con của v. Trong trường hợp nút v có nhiều nút con, chúng ta cần phải có một thuật toán tốt hơn là duyệt. Chúng ta sẽ cần đến một hàm quy hoạch động tiếp theo:

• B[i, s, v, p] là chi phí ít nhất để xây s nhà máy trong i cây con gốc v_1, v_2, v_3, …, v_i của v. Những cây gỗ không được xử lý ở trong cây con v, sẽ trôi đến nút p (p nằm trên đường từ v đến r).

• C[i, s, v] với i, s, v có ý nghĩa tương tự như trên nhưng điểm khác là những cây gỗ chưa được xử lý sẽ được chuyển đến một nhà máy đặt tại v.

Ta sẽ tính được mảng B theo công thức sau:

• B[0, s, v, p] = 0

• B[i, s, v, p] = min (B[i – 1, s – j, v, p] + A[vi, j, p]) với $0 \le j \le s$.

Còn mảng C sẽ được tính theo công thức sau:

• C[0, s, v] = 0

• C[i, s, v] = min (C[i – 1, s – j, v] + A[v_i, j, depth(v)]) với $0 \le j \le s$.

Dựa vào ý nghĩa A1, A2, B, C ta có thể nhận thấy:

• A1[v, t, p] = B[deg(v), t, v, p]

• A2[v, t] = C[deg(v), t – 1, v]
Để tính mọi giá trị A1[v, t, p] (cũng như A2[v, t]) với p thuộc{0..depth(v)}, chúng ta tính B[i, s, v, p] (cũng như C[i, s, v]) với i = 0,…,deg(v) và s = 0,…,k. Như vậy với mỗi cặp v, p ta tính được trong $O(k^2(deg(v) + 1))$. Để tính toàn bộ A, B, C ta mất $O(n\sum k^2(deg(v) + 1)) = O(k^2n^2)$ vì $\sum deg(v) = n – 1$.
Đáp số cuối cùng là A1[r, k, 0].
Dưới đây là toàn bộ chương trình:

```
program riv;
const
maxn = 100;
maxk = 50;
var
lch, fch, next, w, d : array [0..maxn] of integer;
f1, f2 : array [1..maxn, 0..maxn, -1..maxk] of longword;
dis : array [0..maxn, 0..maxn] of longword;
maxcost, maxDword : longword;
n, k : integer;

procedure readata;
var
i, fa : integer;
begin
readln(n, k);
for i := 1 to n do begin
readln(w[i], fa, d[i]);
if fch[fa] = 0 then begin
fch[fa] := i;
lch[fa] := i;
end
else begin
next[lch[fa]] := i;
lch[fa] := i;
end;
end;
end;
function min(x, y : double) : double;
begin
if x < y then min := x
else min := y;
if min > maxDword then
min := maxDword;
end;
procedure dp2(i, grand, k : integer); forward;
procedure dp1(i, grand, k : integer);
begin
if f1[i, grand, k] <> maxDword then exit;
if fch[i] <> 0 then begin
dp2(fch[i], grand, k);
dp2(fch[i], i, k-1);
f1[i, grand, k] := round(min(f2[fch[i], grand, k] + dis[i, grand] * w[i], f2[fch[i], i, k-1]));
end
else
if k = 1 then
f1[i, grand, k] := 0
else if k = 0 then
f1[i, grand, k] := round(min(dis[i, grand] * w[i], maxcost))
else
f1[i, grand, k] := maxcost;
end;
```

```
procedure dp2(i, grand, k : integer);
var
k1, k2 : integer;
begin
if k = -1 then begin
f2[i, grand, k] := maxcost;
exit;
end;
if f2[i, grand, k] <> maxDword then exit;
if next[i] <> 0 then begin
k2:=k;
for k1 := 0 to k do begin
dp1(i, grand, k1);
dp2(next[i], grand, k2);
f2[i, grand, k] := round(min(f2[i, grand, k], f1[i, grand, k1] + f2[next[i], grand, k2]));
dec(k2);
end
end
else begin
dp1(i, grand, k);
f2[i, grand, k] := f1[i, grand, k];
end;
end;

procedure dfs(root, i : integer; cost : longword);
var
ch : integer;
begin
dis[i, root] := cost;
ch := fch[i];
while ch <> 0 do begin
dfs(root, ch, cost + d[ch]);
ch := next[ch];
end;
end;

procedure init;
var
i, j : integer;
begin
for i := 0 to n do dfs(i, i, 0);
maxDword := not maxDword;
fillDword(f1,sizeof(f1) shr 2,maxDword);
fillDword(f2,sizeof(f2) shr 2,maxDword);
maxcost := 2000000001;
end;

begin
reađata;
init;
dp2(fch[0], 0, k);
writeln(f2[fch[0], 0, k]);
end.
```