Cleanup 2 (Pochistvane 2)

Correctness of the author solution

25 November 2026

Abstract

We prove that the author solution `author/author.cpp` returns the optimal amount of plums for every input. The proof has four parts: a formula for the value of a walk, soundness of every value the code computes, a structural lemma saying that every walk is beaten by a plan of a small family, and a proof that the dynamic programme with its convex hulls finds every plan of that family.

1 Setting

Cells are numbered $1, \dots, N$, as in the code (cell i of the statement is cell $i+1$ of the code). Cell i holds $A_i \geq 0$ kilograms of trash and its merchant pays $B_i \geq 0$ plums per kilogram. Sashka starts at S and may make at most E moves. We write

$$P[i] = A_1 + \cdots + A_i, \qquad B[a..b] = \max_{a \leq x \leq b} B_x \quad (a \leq b).$$

Definition 1. A *walk* is a sequence $W = (w_0, w_1, \dots, w_T)$ with $w_0 = S$ and $|w_{t+1} - w_t| = 1$; its *cost* is T. The visited cells form an interval $[l, r]$. For a visited cell i let t_i be the time of its first visit and

$$J_i = \{\, w_t : t \geq t_i \,\}.$$

Consecutive positions differ by one, so J_i is an interval. The *price of i in W* is $\pi_i = \max_{x \in J_i} B_x$.

Theorem 1. *For every input, `solve(S, E, A, B)` returns* OPT*, the largest amount of plums Sashka can obtain with at most E moves.*

2 The value of a walk

Lemma 1. $\mathrm{OPT} = \max_{\mathrm{cost}(W) \leq E} v(W)$, *where*

$$v(W) = \sum_{i=l}^{r} A_i \, \pi_i.$$

Proof. The trash of cell i can be picked up no earlier than t_i. After that it can be sold only at cells w_t with $t \geq t_i$, that is, at cells of J_i, so it brings at most $A_i \pi_i$. Conversely, Sashka can pick up everything on first visits and carry it. For each cell i she sells its trash at a moment $t \geq t_i$ when she stands on a cell of J_i with the largest B. Selling is free and may be split into parts, so all cells reach their bound at the same time. □

3 Plans and soundness

A *plan* is a walk together with a selling cell for every collected cell; the selling cell must be visited after the first visit of the collected cell. The value of a plan is $\sum_i A_i B_{\mathrm{sell}(i)}$. The code evaluates plans assembled from the pieces in table 1. All costs are exact walk lengths.

Piece	Walk	Cost	Sold	Restriction
$\mathrm{Open}_1(l,b)$	$S \to b \to l \to b$	$(b-S)+2(b-l)$	$[l,b]$ at $B[l..b]$	—
$\mathrm{Open}_2(l,b)$	$S \to l \to b$	$2(S-l)+(b-S)$	$[l,b]$ at B_b	—
$\mathrm{Fwd}(k,b)$	$k \to b$	$b-k$	$(k,b]$ at B_b	$B_b \geq B[k+1..b-1]$
$\mathrm{Det}(k,b)$	$k \to b \to k \to b$	$3(b-k)$	$(k,b]$ at B_k	$B_k \geq B[k+1..b]$
$\mathrm{Ret}(b,r)$	$b \to r \to b$	$2(r-b)$	$(b,r]$ at B_b	—
$\mathrm{FamA}(l,r,q)$	$S \to l \to r \to q$	$(S-l)+(r-l)+(r-q)$	$[l,S]$ at $B[l..r]$; $(S,r]$ at $B[q..r]$	—

Table 1: The pieces evaluated by the code. Always $l \leq S \leq b$, $k < b \leq r$ and $l \leq q \leq S \leq r$.

Definition 2. A *DP plan* is $\mathrm{Open}_1(l,b_0)$ or $\mathrm{Open}_2(l,b_0)$, followed by pieces Fwd or Det on boundaries $b_0 < b_1 < \cdots < b_j$, followed by $\mathrm{Ret}(b_j,r)$. A *Family-A plan* is a single FamA. A plan is *allowed* if each of its Fwd and Det pieces satisfies its restriction. *Mirrored* plans are the same objects on the reversed street; the code evaluates them in its second call of `run()`, on the reversed arrays.

Each piece walks every street edge at most three times, so

$$\mathrm{cost}(\mathrm{plan}) \leq 3(N-1) \quad \text{for every plan.} \tag{1}$$

We say that a plan $\mathcal{P}$ *dominates* a walk W if $\mathrm{cost}(\mathcal{P}) \leq \mathrm{cost}(W)$ and every cell visited by W is sold in $\mathcal{P}$ at a price at least its price in W. By Lemma 1 and $A \geq 0$, a dominating plan is worth at least $v(W)$.

Lemma 2 (Soundness)**.** *Every value computed by the code is the value of some plan of cost at most E. Hence the code never returns more than* OPT.

Proof. An entry `row[e]` at boundary b comes from one of three sources:

- a starting state (Open_1 or Open_2);
- an earlier entry, by a Fwd or Det move, with exactly the energy of that piece;
- the step `row[e] = max(row[e], row[e-2])`.

By induction, each entry is the value of a plan prefix that ends at b and costs at most e. Det moves read from the hulls need not satisfy the restriction, but they are still real walks with real sales. The final return uses $r = \min(N, b + \lfloor (E-e)/2 \rfloor)$, so the total cost is at most E. The Family-A loop chooses q so that its cost is at most E. If $E > 3N$, the code works with $3N \leq E$. Lemma 1 gives the bound OPT. □

4 Every walk is dominated by an allowed plan

Lemma 3 (Main lemma)**.** *For every walk W there is an allowed plan, in one of the two orientations, that dominates W.*

The proof constructs the plan explicitly in six steps.

Step 0: the trivial walk, and the orientation

If W visits only S, the plan $\mathrm{Open}_1(S,S) + \mathrm{Ret}(S,S)$ costs 0 and sells S at $B_S = \pi_S$.

Otherwise, the cell that is visited for the first time last is either l or r. If it is l, reflect the street ($i \mapsto N+1-i$); the code evaluates mirrored plans as well. From now on the last new cell is r, and $r > S$.

Step 1: the order of first visits

Let $\sigma = t_l$ (so $\sigma = 0$ if $l = S$) and let $h = \max\{w_t : t \le \sigma\}$.

Claim 1.

(a) The cells visited up to time σ are exactly $[l,h]$, and $S \le h < r$.
(b) After σ the cells $h+1, h+2, \dots, r$ are visited for the first time, in this order.
(c) For $i \in [l,h]$ we have $\pi_i = M \coloneqq B[l..r]$.
(d) For $i \in (h,r]$ we have $J_i = [\alpha_i, r]$ with $\alpha_i = \min\{w_t : t \ge t_i\}$; moreover α is non-decreasing and $l \le \alpha_i \le i$.

Proof. At any moment the visited cells form an interval containing S; at time σ its ends are l and h. Since r is the last new cell, it is first visited after σ, so $h < r$. After σ the left end stays l, so new cells appear only on the right, one at a time. This gives (a) and (b). For $i \le h$ we have $t_i \le \sigma \le t_r$, so J_i contains both l and r, which gives (c). For $i > h$ we have $t_i \le t_r$, so J_i reaches r. As t_i increases with i, the minima α_i do not decrease, which gives (d). □

Let m be the rightmost cell of $[l,r]$ with $B_m = M$.

Step 2: a lower bound on the cost of W

Split $(h,r]$ into maximal runs on which α is constant:

$$(h,r] = (x_0,x_1] \cup (x_1,x_2] \cup \cdots \cup (x_{k-1},x_k], \qquad x_0 = h,\ x_k = r,$$

with $\alpha_i = v_g$ on the g-th group. Then $v_1 < v_2 < \cdots < v_k$ and $v_g \le x_g$.

Claim 2.

$$T \ \ge\ K \coloneqq (h-S) + (h-l) + (r-l) + 2\sum_{g<k}(x_g - v_g) + (r - v_k).$$

Proof. Before σ the walk goes from S to h and then to l, so $\sigma \ge (h-S)+(h-l)$. Let $g < k$. The value of α at x_g is v_g, while at x_g+1 it is $v_{g+1} > v_g$. So the walk stands on v_g at some moment ρ_g with $t_{x_g} \le \rho_g < t_{x_g+1}$. It also stands on v_k at some moment $\rho_k \ge t_r$. Hence after σ the walk visits, in this order,

$$l,\ x_1,\ v_1,\ x_2,\ v_2,\ \dots,\ x_k,\ v_k,$$

and the sum of these distances is $(r-l) + 2\sum_{g<k}(x_g - v_g) + (r - v_k)$. □

Figure 1 shows a walk of this shape; it is not drawn to scale.

We now ask the plan to give the *required prices*

$$R_i = M \quad (i \in [l,h]), \qquad R_i = B[\min(i, v_g)..r] \quad (i \in (x_{g-1}, x_g]).$$

For the groups of Step 2 we have $v_g \le i$, so $R_i = \pi_i$. A plan of cost at most K whose prices are at least R dominates W.

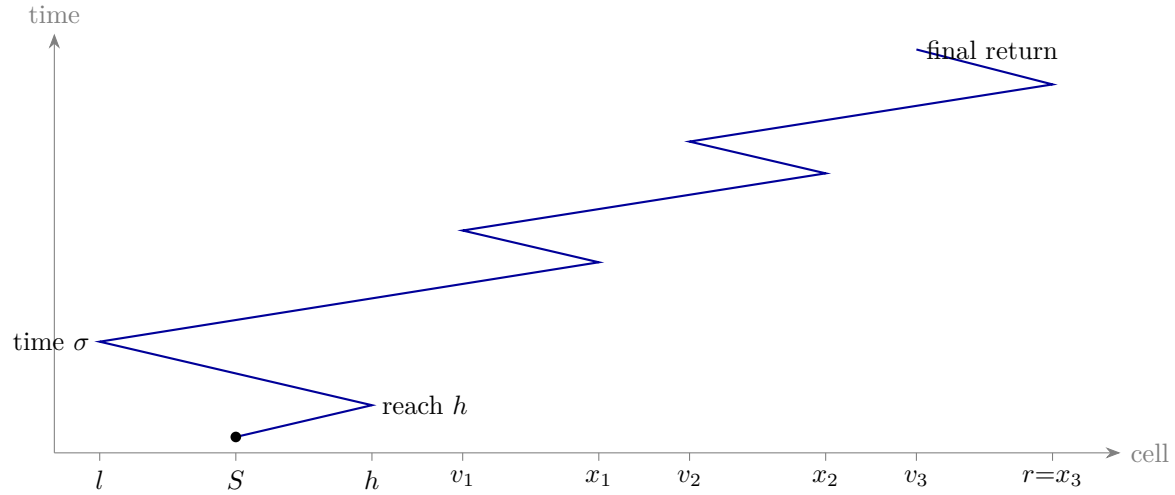

Figure 1: The shape of a walk after Steps 1–2: an opening $S \to h \to l$, a sweep to the right with dips back to $v_1, v_2, \ldots$, and a final return to v_k.

Step 3: normalisation

We modify the parameters k, x_g, v_g by three rules. No rule increases K, and no rule decreases any R_i. During the process $v_g \le x_g$ always holds, and the formulas for K and R are used as written. For $g < k$ let p_g be the rightmost cell of $[v_g, r]$ with the largest B.

E1 (overlapping dips).
If $v_{g+1} < x_g$, merge groups g and $g+1$ into $(x_{g-1}, x_{g+1}]$ with dip $v = \min(v_g, v_{g+1})$. Every R_i grows, because $\min(i, v)$ is at most both old values. If $v = v_g$, then K drops by $2(x_g - v_{g+1}) > 0$ (by $(x_g - v_g) + (x_g - v_k) > 0$ when $g + 1 = k$). If $v = v_{g+1}$, then K drops by $2(x_g - v_g) \ge 0$.

E3 (a useless dip).
If $g < k$, $v_g < x_g$ and $p_g > x_g$, set $v_g \coloneqq x_g$. Then K drops by $2(x_g - v_g) > 0$. For $i > v_g$ the old requirement was B_{p_g}, and $p_g \in [i, r]$, so the new requirement $B[i..r]$ is at least as large.

E4 (no dip).
If $g < k$ and $v_g = x_g$, merge group g into group $g+1$, keeping the dip v_{g+1}. K does not change, and $B[\min(i, v_{g+1})..r] \ge B[i..r]$.

Each rule decreases k or K, so the process stops. At the end, for every $g < k$:

$$\text{(N1)} \quad v_{g+1} \ge x_g, \tag{2}$$

$$\text{(N2)} \quad v_g < x_g \ \text{ and } \ p_g \le x_g. \tag{3}$$

Step 4: the group lemma

Lemma 4 (Group lemma)**.** *Let $S \le a \le v \le x \le r$, and let p be the rightmost cell of $[v, r]$ with the largest B; assume $p \le x$. Suppose every cell $i \in (a, x]$ requires $B[\min(i, v)..r]$. Then allowed pieces lead from boundary a to boundary x, meet all these requirements, and cost at most*

$$(x - a) + 2(x - v).$$

If $x = r$, the pieces may instead finish with a Ret*, at cost at most $(r - a) + (r - v)$.*

Proof. Case $p = a$. Then $v = a$, and every requirement equals $B_a = B[a..r]$. Use $\mathrm{Det}(a, x)$, of cost $3(x - a)$, or $\mathrm{Ret}(a, r)$, of cost $2(r - a)$. $\mathrm{Det}(a, x)$ is allowed, because B_a is the largest value on $[a, r]$.

Case $p > a$. For $i \in (a, p]$ the requirement equals $B[i..p]$. Indeed, for $i \le v$ it is $B[i..r]$ and $B_p \ge B[v..r]$; for $i > v$ it is $B_p = B[i..p]$. Define $c_0 = a$ and let c_{t+1} be the rightmost cell of $(c_t, p]$ with the largest B, until $c_t = p$. Every $\mathrm{Fwd}(c_t, c_{t+1})$ is allowed, sells each of its cells i at $B_{c_{t+1}} = B[i..p]$, and the chain costs $p - a$. The cells of $(p, x]$ require B_p. If $x > p$, use $\mathrm{Det}(p, x)$; it is allowed because B_p is the largest value on $[v, r]$. The total cost is

$$(p - a) + 3(x - p) = (x - a) + 2(x - p) \le (x - a) + 2(x - v),$$

since $p \ge v$. When $x = r$, use $\mathrm{Ret}(p, r)$ instead: $(p - a) + 2(r - p) = (r - a) + (r - p) \le (r - a) + (r - v)$. □

Step 5: the groups after the first

The bound splits as $K = K_1 + c_2 + \cdots + c_k$, where $c_g = (x_g - x_{g-1}) + 2(x_g - v_g)$ for $2 \le g < k$, $c_k = (r - x_{k-1}) + (r - v_k)$, and

$$K_1 = (h - S) + (h - l) + (x_1 - l) + \begin{cases} 2(x_1 - v_1), & k > 1, \\ r - v_1, & k = 1. \end{cases}$$

For $g \ge 2$, Lemma 4 applies with $a = x_{g-1}$, $v = v_g$, $x = x_g$: (N1) gives $a \le v$, and (N2) (or $x_k = r$) gives $p_g \le x_g$. It covers group g at cost at most c_g. It remains to cover $[l, h]$ and group 1 at cost at most K_1, ending at boundary x_1 (or ending the plan if $k = 1$).

Step 6: the opening

In each case we give the plan, the difference "K_1 minus its cost" (or the inequality it reduces to), and the reason the prices suffice.

Case I: $v_1 \ge h$**.**

- **I.1:** $m \le h$**.** Take $\mathrm{Open}_1(l, h)$ and then Lemma 4 with $a = h$, $v = v_1$, $x = x_1$. The cost is $(h - S) + 2(h - l)$ plus the bound of the lemma, which is exactly K_1. The cells of $[l, h]$ are sold at $B[l..h] = M$.
- **I.2:** $m > h$**.** Let g^* be the group with $x_{g^*-1} < m \le x_{g^*} =: x^*$, and $v^* = v_{g^*}$. Take $\mathrm{Open}_2(l, m)$. It sells $[l, m]$ at M, which meets every requirement. For the cells of $(m, x^*]$:
  - if $v^* \le m$, they require M, because $m \in [v^*, r]$. Use $\mathrm{Det}(m, x^*)$ (allowed, as $B_m = M$ is the largest value on $[l, r]$), nothing if $x^* = m$, or $\mathrm{Ret}(m, r)$ if $g^* = k$. Comparing with K up to x^* reduces to $S + v^* \le h + m$, or to $2S + v_k \le 2h + m$ when $g^* = k$. Both hold since $S \le h$ and $v^* \le m$;
  - if $v^* > m$, apply Lemma 4 with $a = m$, $v = v^*$, $x = x^*$ ((N2) gives $p_{g^*} \le x^*$). The comparison reduces to $S \le h$.

  The dips of groups $1, \ldots, g^* - 1$ only make K larger, so leaving them out of the comparison is safe. The groups after g^* are covered by Step 5.

Case II: $v_1 < h$**.** Every cell of group 1 requires $B[v_1..r] = B_{p_1}$.

- **II.a:** $k > 1$**.** First, $m \le x_1$: if $m \ge v_1$ then $p_1 = m$ and (N2) gives $p_1 \le x_1$; if $m < v_1$ then $m < h$. Take $\mathrm{Open}_1(l, x_1)$, which sells $[l, x_1]$ at $B[l..x_1] = M$. Then

$$K_1 - \text{cost} = 2(h - v_1) > 0.$$

- **II.b:** $k = 1$, so $x_1 = r$ and $K_1 = (h - S) + (h - l) + (r - l) + (r - v_1)$. Every plan below sells all requested cells at M.

Condition	Plan	K_1 − cost
$v_1 \le m$, $S \le m \le h$	$\mathrm{Open}_1(l,m) + \mathrm{Ret}(m,r)$	$(h-m)+(h-v_1) \ge 0$
$v_1 \le m$, $m > h$	$\mathrm{Open}_2(l,m) + \mathrm{Ret}(m,r)$	$2(h-S)+(m-v_1) \ge 0$
$v_1 \le m < S$	$\mathrm{FamA}(l,r,m)$	$2(h-S)+(m-v_1) \ge 0$
$m < v_1$, $m \ge S$	$\mathrm{Open}_2(l,m) + \mathrm{Ret}(m,r)$	$(h-S)+(m-S)+(h-v_1) > 0$
$m < v_1$, $m < S$	mirrored: $S \to r \to m \to l \to m$	$(h-v_1)+(h-m) > 0$

For $\mathrm{FamA}(l,r,m)$: $[l,S]$ is sold at $B[l..r] = M$ and $(S,r]$ at $B[m..r] = M$. The mirrored plan is Open_2 followed by Ret on the reversed street. It costs $2(r-S)+(S-m)+2(m-l)$ and sells all of $[l,r]$ at B_m. This case really needs the mirrored orientation, and the third row really needs FamA.

Proof of Lemma 3. In every case the plan is allowed: its Fwd and Det pieces come from Lemma 4, or are $\mathrm{Det}(m,x^*)$ with $B_m = M$. Its cost is at most $K \le T$ by Claim 2 and Step 3. Every cell of W is sold at least at $R_i \ge \pi_i$. □

5 The dynamic programme finds every allowed plan

Lemma 5. *Let $E' = \min(E, 3N)$. For every allowed plan of cost at most E' in the orientation of one call of `run()`, that call returns at least the value of the plan.*

Throughout, $j = b - S$ for a boundary $b \ge S$, and $\mathrm{row}_b[e]$ is the code's table entry.

Ranges and parity. A plan of cost at most E' pays $2(S-l)$ to return from l, so $l \ge S - E'/2$; all its boundaries are at most $S + E'$. Hence the loop limits `L` and `R` lose nothing. Every plan prefix ending at b has cost $\equiv j \pmod 2$:

$$\mathrm{cost}\,\mathrm{Open}_1(l,b) = 3j + 2(S-l), \qquad \mathrm{cost}\,\mathrm{Open}_2(l,b) = 2(S-l) + j,$$

and Fwd, Det add $b-k$ and $3(b-k)$, both congruent to the increase of j. After the step `row[e] = max(row[e], row[e-2])`, $\mathrm{row}_b[e]$ means "cost at most e".

Lines. A Fwd move from (k,e') to (b,e) satisfies $e - j = e' - j'$, where $j' = k - S$. A Det move satisfies $e - 3j = e' - 3j'$. So moves stay inside one forward group $f = (e-j)/2$ or one detour group $g = (e-3j+2E)/2$, and inside a group the target energy is determined by b. The value of a move is a line evaluated at one point:

$$\text{Fwd}: \quad \mathrm{row}_k[e'] + B_b\,(P[b] - P[k]) = B_b P[b] + \underbrace{(-P[k])}_{\text{slope}}\, B_b + \underbrace{\mathrm{row}_k[e']}_{\text{intercept}}, \quad x = B_b;$$

$$\text{Det}: \quad \mathrm{row}_k[e'] + B_k\,(P[b] - P[k]) = \underbrace{B_k}_{\text{slope}}\, P[b] + \underbrace{\mathrm{row}_k[e'] - B_k P[k]}_{\text{intercept}}, \quad x = P[b].$$

Lines enter a group's vector in increasing k. Forward slopes $-P[k]$ never increase because $A \ge 0$. Detour slopes of alive cells never increase, because the monotone stack keeps B non-increasing.

How lines leave a vector.

(a) *At a push.* An equal-slope line with the smaller intercept is dropped. Otherwise the last line b before the new line c is dropped when `useless(a,b,c)` holds, i.e.

$$(b.c - a.c)(m_a - m_c) \le (c.c - a.c)(m_a - m_b), \qquad m_a > m_b > m_c.$$

This says exactly that at the point x_{ac} where a and c meet, b is not above them. Then $b(x) \le \max(a(x), c(x))$ for every x: a line dropped at a push is everywhere below one of two lines present at that moment.

(b) *At a query* at point x, the last line is dropped while the line before it is at least as large at x. That line has a larger or equal slope, so it stays at least as large for every $x' \ge x$.

(c) *Dead cells* (detour groups only). A cell k is *dead* at time b if $B_t > B_k$ for some $t \in (k, b]$. The cells killed by a new b are exactly the alive cells with the largest indices, so dead lines always form a suffix of the vector. After removing them nothing dead remains; every query and push sees only alive lines.

After (a), every vector is an upper hull; removing lines from its back leaves an upper hull. At a fixed x the values along an upper hull are unimodal, so popping from the back while the previous line is not smaller returns the maximum of the vector.

Claim 3 (Forward groups)**.** *Let ℓ be the line of a state (k, e') in its forward group, and let* $\mathrm{Fwd}(k, b)$ *be allowed. Then the query at b returns at least $\ell(B_b)$.*

Proof. All queries to this group at boundaries $t \in (k, b)$ use $x_t = B_t \le B_b$, because $\mathrm{Fwd}(k, b)$ is allowed. Follow the chain of removals that starts at ℓ. Each removed line is below its dominator either everywhere (a), or for all $x \ge x_t$ with $x_t \le B_b$ (b). Every removal in the chain happens at a time in (k, b), so each step holds at $x = B_b$. The chain ends at a line still present at b, and the query returns at least its value. □

Claim 4 (Detour groups)**.** *For every b, every code state (k, e') with $k < b$, and every* $\mathrm{Det}(k, b)$*, allowed or not,*

$$\mathrm{row}_b[e' + 3(b-k)] \;\ge\; \mathrm{row}_k[e'] + B_k\,(P[b] - P[k]).$$

Proof. Strong induction on b; for a fixed b, induction on k from $b-1$ downwards. Let ℓ be the line of (k, e').

k is dead at b. Let b^* be the rightmost cell of $(k, b]$ with the largest B; then $B_{b^*} > B_k$ and $\mathrm{Fwd}(k, b^*)$ is allowed. Use the following route:

1. $\mathrm{Fwd}(k, b^*)$: by Claim 3 at b^*, the state at b^* is worth at least $\mathrm{row}_k[e'] + B_{b^*}(P[b^*] - P[k])$.
2. If $b^* < b$, then $\mathrm{Det}(b^*, b)$: it has a nearer source, so it is covered by the inner induction.

The energy $e' + (b^* - k) + 3(b - b^*)$ is at most $e' + 3(b-k)$ and has the same parity. The value is at least

$$\mathrm{row}_k[e'] + B_{b^*}(P[b] - P[k]) \;\ge\; \ell(P[b]).$$

k is alive at b. Detour queries use $x = P[b]$, which never decreases, so (b) gives domination for all later queries. Follow the chain of removals that starts at ℓ.

- A dominator from an *earlier* source k' is alive at b. It was alive at the moment of the removal, and $B_{k'} \ge B_k \ge B[k+1..b]$. So the chain continues through it.
- A dominator from a *later* source $k'' \in (k, b)$ is either alive at b, and the chain continues; or dead at b. In the second case its value is covered by the inner induction, since it lies in the same group and so has the same target energy.

So $\ell(P[b])$ is at most the query result or a value already bounded by row_b. □

Proof of Lemma 5. Induction along the boundaries of the plan. The starting states are written directly; allowed Fwd moves are covered by Claim 3; Det moves by Claim 4. The final Ret is evaluated for every state with the largest affordable r, and selling more cells never lowers the value. Family-A plans are evaluated directly. The code uses the smallest affordable q, and a smaller q only widens the range of $B[q..r]$. □

6 Proof of the theorem

Proof of Theorem 1. Let W be an optimal walk of cost at most E. By Lemma 3, some allowed plan $\mathcal{P}$, in some orientation, dominates W. By (1), $\mathrm{cost}(\mathcal{P}) \le \min(T, 3(N-1)) \le E'$. Its value is at least $v(W) = \mathrm{OPT}$, since $A \ge 0$. By Lemma 5, the call of `run()` in that orientation returns at least $\mathrm{val}(\mathcal{P}) \ge \mathrm{OPT}$. By Lemma 2, neither call returns more than OPT. Hence `solve` returns exactly OPT. □

Machine checks

These do not replace the proof; they test that every step says what the code does.

Check	Scope	Result
Lemma 3 with exactly the allowed plans and cost $\le \min(T, 3N)$	every walk, every S, every ordering of B, $N \le 8$	0 failures in about $4 \cdot 10^9$ cases
Steps 0–6 executed as a program, every inequality asserted	300 000 random walks with $N \le 40$; all walks of length ≤ 10 for $N \le 5$ with every B	0 failures; every case of Step 6 occurs
The same program with one step broken	four broken variants, 20 000 walks each	189 to 1008 failures each
`author.cpp` against the exhaustive optimum	every S, $E \le 3N+1$, every ordering of B, $A \in \{0,1,3\}^N$, $N \le 6$	0 mismatches in $4.7 \cdot 10^7$ calls; three wrong solutions are caught