M1721 Bus Fare

Problem by: Charlie Li

Problem Statement

- There are N bus stops, given the number of passengers get on and get off at each stop, and the distance between the stops.
- Find the minimum and maximum total bus fare received by the bus company
- The bus fare of each trip is equal to the square root of total distance travelled

Subtasks

- $2 \leq N \leq 10^6$, $1 \leq d_i \leq 10000$, $0 \leq a_i, b_i \leq 10000$
- Subtask 1 (20 points): sum of a_i <= 10
- Subtask 2 (40 points): N <= 1000
- Subtask 3 (30 points): N <= 10^5
- Subtask 4 (10 points): No additional constraint

Subtask 1

- Subtask 1 (20 points): sum of a_i <= 10
- We can exhaust all the possible configuration for all passengers
- Time complexity: O((sum of a_i)!)

Observation

- It is important to see that sqrt(d) is a strictly increasing function
- Then 1/sqrt(d) will be a strictly decreasing function
- Where 1/sqrt(d) = sqrt(d)/d = average cost
- This means the average cost will be lower if one travel for a longer distance

- So, you may think that when the passengers travel as short as possible, then the bus company will received the maximum total fare
- When the passengers travel as long as possible, the bus company will received the minimum total fare

Observation

- We can do this using greedy!

- In order to let the passengers travel as short as possible, we should let those who get on first to get off first
- In order to let the passengers travel as long as possible, we should let those who get on last to get off first

Subtask 2

- Subtask 2 (40 points): N <= 1000
- For every bus stop, we can find those passengers who get on the bus first and still on bus and let them get off the bus to calculate the maximum bas fare
- Also, we can find those passengers who get on the bus last and still on bus and let them get off the bus to calculate the minimum bas fare

Subtask 2

- Code:

(a2 and b2 are copies of a and b)

- Time complexity: $O(N^2)$

```
for (int i = 1; i <= n; i++) {
    for (int j = 1; b[i]; j++) {
        if (a[j] <= b[i]) {
            b[i] -= a[j];
            ans2 += a[j] * sqrt(d[i - 1] - d[j - 1]);
            a[j] = 0;
        }
        else {
            a[j] -= b[i];
            ans2 += b[i] * sqrt(d[i - 1] - d[j - 1]);
            b[i] = 0;
        }
    }
    for (int j = i - 1; b2[i]; j--) {
        if (a2[j] <= b2[i]) {
            b2[i] -= a2[j];
            ans1 += a2[j] * sqrt(d[i - 1] - d[j - 1]);
            a2[j] = 0;
        }
        else {
            a2[j] -= b2[i];
            ans1 += b2[i] * sqrt(d[i - 1] - d[j - 1]);
            b2[i] = 0;
        }
    }
}
```

Subtask 3

- Subtask 3 (30 points): N <= 10^5
- In stead of using a for loop to check every stop, we can use a heap(aka. priority queue) to store the smallest stop or the biggest stop.
- Time complexity: O(N log N)
- There is an O(N) solution, but an O(N log N) is enough for 100 points

Subtask 4

- Subtask 4 (10 points): No additional constraint
- Here, we are going to see an O(N) solution

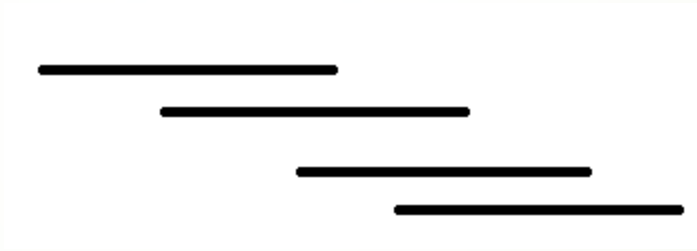
- Go back to the observation part, we have known that:
  - In order to let the passengers travel as short as possible, we should let those who get on first to get off first
  - In order to let the passengers travel as long as possible, we should let those who get on last to get off first
- Doesn't they looks familiar?
  - First in first out
  - Last in first out

Subtask 4

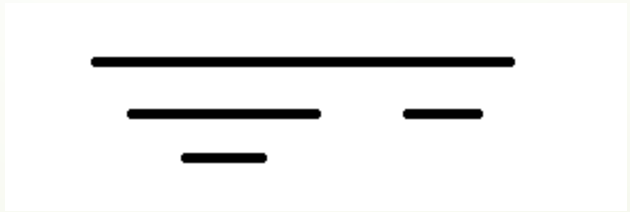
- And yes, we can use a queue to find the maximum fare and a stack to find the minimum fare!
- Time complexity: O(N)

Why greedy works?

- Consider that every passenger is a horizontal line of which the left is the stop he get on the bus and the right is the stop he get off the bus
- If we let those who get on first to get off first, the lines look like this:

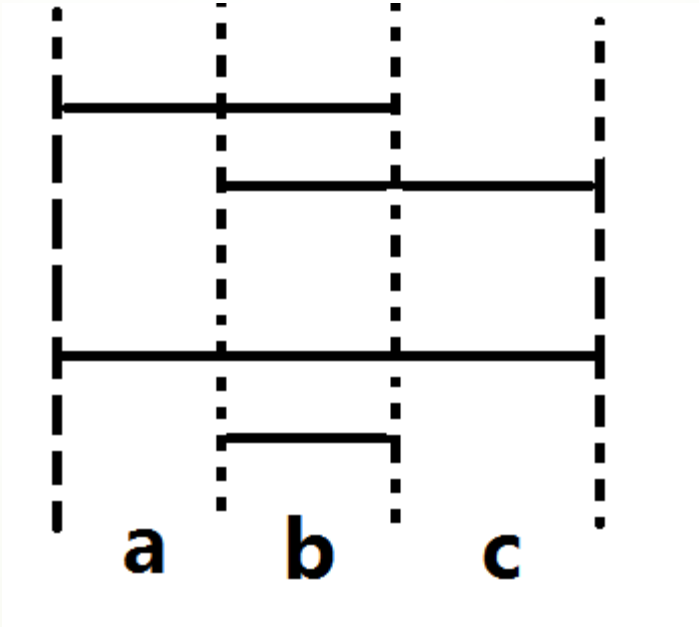


- If we let those who get on last to get off first, the lines look like this:



Why greedy works?

- Consider the case with only two passengers:



- The upper two is first in first out, the lower two is last in first out

Why greedy works?

- Total bus received from the upper two = $\sqrt{a+b}+\sqrt{b+c}$
- Total bus received from the lower two = $\sqrt{a+b+c}+\sqrt{b}$

$$ac \geq 0$$
$$ab+ac+b^2+bc \geq ab+b^2+bc$$
$$\sqrt{(a+b)(b+c)} \geq \sqrt{(a+b+c)(b)}$$
$$a+b+2\sqrt{(a+b)(b+c)}+b+c \geq a+b+c+2\sqrt{(a+b+c)(b)}+b$$
$$\left(\sqrt{a+b}+\sqrt{b+c}\right)^2 \geq \left(\sqrt{a+b+c}+\sqrt{b}\right)^2$$
$$\sqrt{a+b}+\sqrt{b+c} \geq \sqrt{a+b+c}+\sqrt{b}$$

- So, if all pair of passengers look like the upper two, we have maximum bus fare
- It all pair of passengers look like the lower two, we have minimum bus fare

M1722 Least Digit Sum Multiple

Problem by: Charlie Li

Problem Statement

- Given an integer N, find the minimum multiple of N with least possible digit sum

Subtasks

- $1 <= N <= 10^5$
- Subtask 1 (16 points): $1 <= N <= 20$
- Subtask 2 (8 points): $N = 2^p5^q$ for some non-negative integer p, q
- Subtask 3 (22 points): $N = k\ 2^p5^q$ for some non-negative integer k, p and q where $k <= 20$
- Subtask 4 (28 points): $N <= 1000$
- Subtask 5 (26 points): No additional constraint

Subtask 1

- Subtask 1 (16 points): 1 <= N <= 20
- This subtask is for hardcode

- 1 2 3 4 5 6 7 8 9 10
- 1 10 3 100 10 12 1001 1000 9 10
- 11 12 13 14 15 16 17 18 19 20
- 11 12 1001 10010 30 10000 100000001 18 1000000001 100

Subtask 2

- Subtask 2 (8 points): $N = 2^p5^q$ for some non-negative integer p, q
- For this subtask, we know that the least possible digit sum is 1 since $10^{max(p,q)}$ is a multiple or N
- In fact, $10^{max(p,q)}$ is the smallest one

Subtask 3

- Subtask 3 (22 points): N = $k2^p5^q$ for some non-negative integer k, p and q where k <= 20
  - Here, we may assume that k is neither a multiple of 2 nor 5
- It is easy to show that the least possible digit sum of N is equal to the least possible digit sum of k.
- It is also easy to show that when least possible digit sum of k < 3, the answer is just ans(k) × $10^{\max(p, q)}$
- When K = 3 it is also not complicated, the answer will be in the form 3 × 10^n or 12× 10^n
  - If q > p then answer = 3 × 10^q
  - Otherwise, answer = 12× $10^{\max(p-2, q)}$
- When K = 9, the only thing we know is the least possible digit sum is equal to 9, in worse case the number of checking is $\frac{N^{\log_2 5}}{9^n}$ where n is the largest power of 9 less than N such that n has a digit sum of 9
  - Total numbers of checking is around 10^6 for N <= 10^5, so we can use brute force

Subtask 4

- Subtask 4 (28 points): N <= 1000
- It is good to observe that ans(N) has at most N digits (proof at the back)
- We will separate the question into two parts
  - We will find the least possible digit sum first
  - Then, we will find the smallest multiple
- For the first part, we can use dp to do it

Subtask 4

- Define dp[i][j] to be the least possible digit sum of an integer with i digits such that it mod N = j
- The least possible digit sum of multiple of N is $\min_{1 \le i \le N} dp[i][0]$
- Transition formula: dp[i][j] = min(dp[i-1][a] + b)
  - where 0 <= b <= 9 and (a * 10 + b) % n = j
- Base case: dp[1][b % n] = min(b) where 1 <= b <= 9
- It is more convenient to write code like:
  - for k = 0 to 9, dp[i+1][(j * 10 + k) % n] = min(dp[i+1][(j * 10 + k) % n], dp[i][j] + k)
  - for k = 1 to 9, dp[1][k % n] = min(dp[1][k % n], k)

Subtask 4

- After finding the least possible digit sum, we should then find the minimum no. of digits needed
  - that is find the smallest d such that $dp[d][0] = \min_{1 \le i \le N} dp[i][0]$
- Then we can trace the path which lead us to dp[d][0]
  - tracing from the d-1 down to 1, mark down all the (i, j) which lead us to (d, 0)
  - Then we walk from the first digit to the d^{th} digit using the smallest digit every step to find the ultimate answer
- Overall time complexity: $O(n^2)$

Subtask 4

- Code for DP part:

```
for (int i = 1; i <= n; i++) for (int j = 0; j < n; j++) for (int k = 0; k < 10; k++)
    dp[i + 1][(j * 10 + k) % n] = min(dp[i + 1][(j * 10 + k) % n], dp[i][j] + k);
```

- Code for tracing path part (l is initialized to 1):

```
void trace(int i, int j) {
    for (int k = 0; k < 10; k++) {
        if (u[i + 1][(j * 10 + k) % n] && dp[i + 1][(j * 10 + k) % n] == dp[i][j] + k) {
            printf("%d", k);
            trace(i + 1, (j * 10 + k) % n);
            break;
        }
    }
}
```

```
for (int i = 1; i <= n; i++) if (dp[i][0] < dp[l][0]) l = i;
u[l][0] = true;
for (int i = n - 1; i; i--) for (int j = 0; j < n; j++) for (int k = 0; k < 10; k++)
    if (u[i + 1][(j * 10 + k) % n] && dp[i + 1][(j * 10 + k) % n] == dp[i][j] + k) {
        u[i][j] = true;
    }
for (int i = 1; i <= 9; i++) {
    if (u[l][i%n]) {
        printf("%d", i);
        trace(l, i%n);
        printf("\n");
        break;
    }
}
```

Subtask 5

- Subtask 5 (26 points): No additional constraint
- Noted that in the solution of subtask 4, the answer will not pass through two point with same value mod n (proof at the back)
- So the DP part can be replaced using some other graph
- We can use a graph with n nodes and 10×n edges
  - Node i is corresponding to the values mod n = i
  - The edges are corresponding to adding a digit to the values (represented by a node), so the weight of an edge is equal to the digit it is adding

Subtask 5

- We can run shortest path on the graph once (starting points are 1%n, 2%n, … and 9%n, ending point is 0) to find the least possible digit sum and also the number of digits of the answer.
- Then we can use the similar strategy for tracing the path as stated in previous page but this time requires a dfs to mark the nodes
- By using Dijkstra, we can have an O(n log n) solution.

Subtask 5

- Code for the shortest path part:

```
for (int i = 0; i < 10; i++) {
    // ds for digit sum, d for digits, x is current node
    y = (x * 10 + i) % n;
    if (node[x].ds + i < node[y].ds || node[x].ds + i == node[y].ds && node[x].d + 1 < node[y].d) {
        node[y].ds = node[x].ds + i;
        node[y].d = node[x].d + i;
        q.push(node[y]);
    }
}
```

Answer have at most N digits

- Proof:
  - Suppose the ans(N) has M digits where M > N
  - Let f(x) = floor(ans(N) / 10^x) and g(x) = ans(N) % 10^x where 1 <= x <= M
    - f(x) is the first M - x digits of the answer and g(x) is the last x digits
  - f(x) is strictly decreasing for 1 <= x <= M
  - f(x) % N is not distinct for 1 <= x <= M (pigeonhole principle)
  - So we can find M >= x > y >= 0 such that f(x) % N = f(y) % N
  - Since ans(N) % N = 0
  - So (f(y) * 10^y + g(y)) % N = 0 and (f(x) * 10^y + g(y)) % N is also 0
  - And digit sum of f(x) <= digit sum of f(y)
  - However, f(x) * 10^y + g(y) < f(y) * 10^y + g(y) = ans(N)
  - f(x) * 10^y + g(y) is a better answer
  - Contradiction arises, QED

Path don't pass node with same mod value twice

- Proof:
  - Suppose the ans(N) has M digits
  - Let f(x) = floor(ans(N) / 10^x) and g(x) = ans(N) % 10^x where 1 <= x <= M
    - f(x) is the first M - x digits of the answer and g(x) is the last x digits
  - f(x) is strictly decreasing for 1 <= x <= M
  - Suppose the path passes through the node with same mod value twice
  - Then, we can find M >= x > y >= 0 such that f(x) % N = f(y) % N
  - Since ans(N) % N = 0
  - So (f(y) * 10^y + g(y)) % N = 0 and (f(x) * 10^y + g(y)) % N is also 0
  - And digit sum of f(x) <= digit sum of f(y)
  - However, f(x) * 10^y + g(y) < f(y) * 10^y + g(y) = ans(N)
  - f(x) * 10^y + g(y) is a better answer
  - Contradiction arises, QED

M1723 Debug or rewrite?

Theo

Description

Given N pairs of integers

For each pair we can either “buy” the first one or the second one, and we want to minimize the cost

When you buy a item of specific type, the cost will be modified depending on how many items of that type you bought before

Notice: the answer may (will) exceed 2^31 - 1, use int64 or long long!

Subtask 4

No additional cost!

For each pair buy the cheapest of the two

O(N)

Subtask 1

Simulation of decision

Try all possible buying sequence

There are a total of 2^n of them (For each pair the sequence diverse by 2)

O(2^n)

Subtask 2

Notice that solution for subtask 1 can be easily transformed into a dp solution

dp(cur, acnt, bcnt): the minimal cost to buy item cur..n if we bought **acnt** items of first type and **bcnt** items of second type

dp(cur, acnt, bcnt) = min(dp(cur + 1, acnt + 1, bcnt) + (a[i] - fA[acnt]),

dp(cur + 1, acnt, bcnt + 1) + (b[i] - fB[cnt]))

O(N^3) state, O(1) state per transition, therefore O(N^3)

Subtask 3

Actually, bcnt = (cur - 1 - acnt), since you bought (cur - 1) items already, and all of the items there are not of first type is second type

So, the number of dp state is actually O(N^2)

Full solution

Think in other way

If we know that we will buy k items of first type and n-k items of second type

Total cost = (cost of the k items of first type) - **[sum(i from 0 to k - 1) (fA[i])]**

(cost of the n-k items of second type) -

[sum(i from 0 to (n-k)-1) (fB[i])]

These two terms are constant if we fix k!

(Also notice that the order of buying does not matter actually!!!)

Full solution

We need to compute the minimal of the two non-constant terms

Assume that we initially buy all items of second type.

We now want to buy exactly k items of first type instead of second type

The cost reduced = b[i] - a[i] for each i

Sort the array by decreasing (b[i] - a[i]) and take the first k!

Full solution

Number of k to be considered: O(N)

-> O(N^2 lg N)

However, notice that to transist from k=c to k=c+1, we just need to buy the (c+1)-th item of first type instead of second type

-> O(1) per k after O(N) precomputation, O(N lg N) (sorting)

HKOI 2016/17
Mini-Comp II Editorial

M1624 Guess the Number

Alex Tung

18 March 2017

Problem Description

- Guess a number X in [1, N]

- Call guess(K):
  - returns 0 if X <= K -> Bob pays A coins
  - returns 1 if X > K -> Bob pays B coins
- Call answer(V) if X can be determined

- Target: minimize cost **in the worst case**

Subtask 1: A = B

- One optimal strategy is, of course, **binary search**

- Number of steps required = $\lceil \log_2 N \rceil$
- Cost = A * $\lceil \log_2 N \rceil$

Subtask 2: N <= 5000

Use the following DP formulation (A, B fixed):

- dp[m]: minimum cost for guessing in range [1, m]

- Say you call guess(K)
  - guess(K) = 0: pay A coins, range becomes [1, K]
  - guess(K) = 1: pay B coins, range becomes [K + 1, m]

Therefore, for a specific K,

dp[m] = max(dp[K] + A, dp[m – K] + B)

- dp[1] = 0
- Transition formula:

$$\mathrm{dp}[\mathrm{m}] = \min_{1 \le K < m} \{\max(\mathrm{dp}[\mathrm{K}] + \mathrm{A}, \mathrm{dp}[\mathrm{m} - \mathrm{K}] + \mathrm{B})\}$$

Time complexity: $O(N^2)$

Subtask 3: N <= 10^7

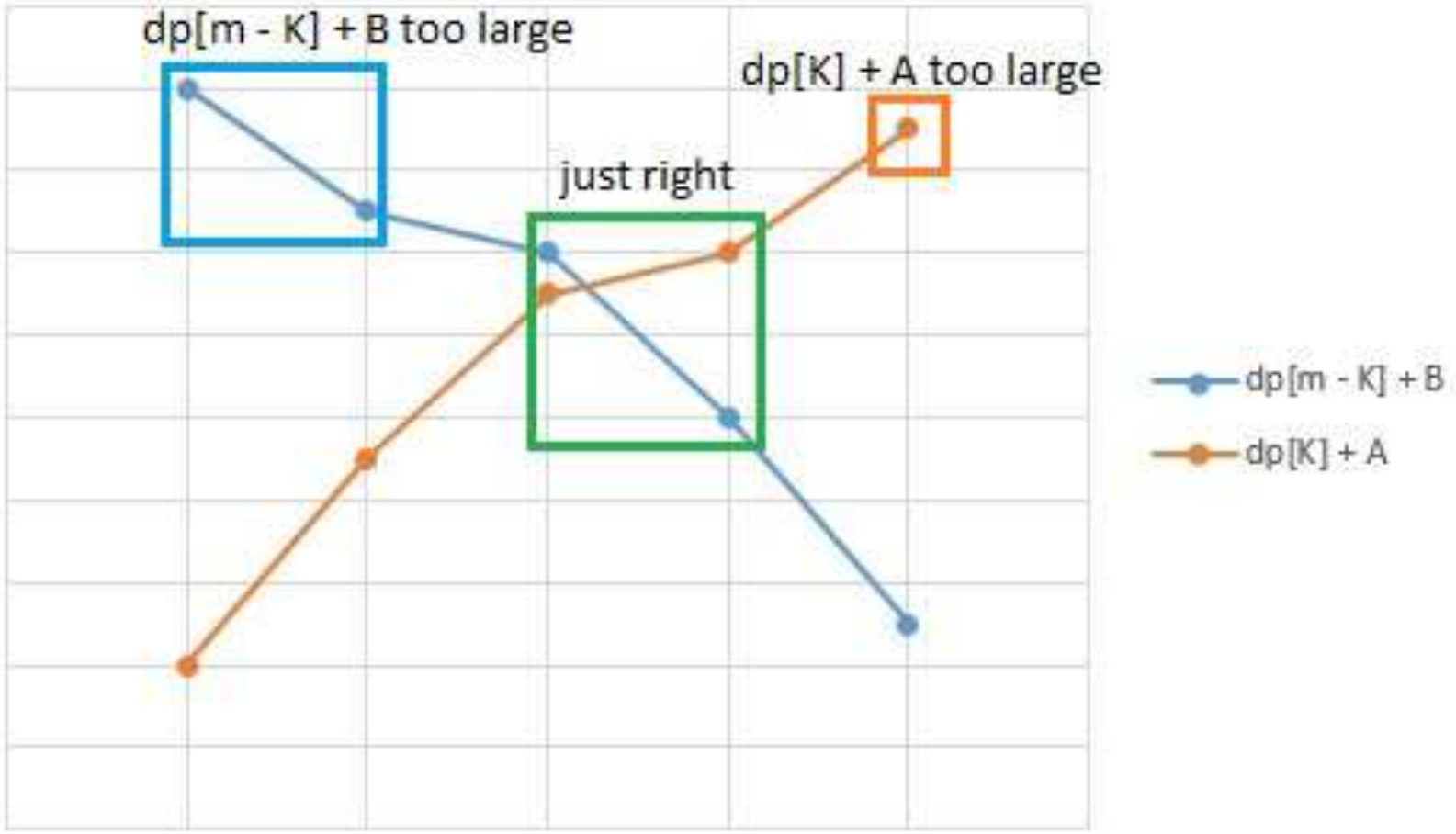
- Recall the transition formula

$$dp[m] = \min_{1 \le K < m} \{\max(dp[K] + A, dp[m - K] + B)\}$$

- Observe that dp[] is increasing

- So, as K ↑, (dp[K] + A) ↑ and (dp[m – K] + B) ↓

The optimal K will be near the location where the graphs of (dp[K] + A) and (dp[m – K] + B) intersect!



- Concretely, we want to find an index I_m := opt[m] such that:
  - $dp[I_m] + A < dp[m - I_m] + B$
  - $dp[I_m + 1] + A >= dp[m - I_m - 1] + B$

Then we only need to consider $K = I_m$ and $K = I_m + 1$

- Fortunately, $I_m <= I_{m+1}$
- Use a pointer to maintain I_m

Time complexity: O(N)

Subtask 4: N <= 10^{18}

- From sample 4, you may observe that the answer will not grow very large
- In fact, it is upper-bounded by max(A, B) * $\lceil \log_2 N \rceil$ (WHY?)

We need to use another DP formulation :P

Let dp2[C]: maximum range of numbers one can distinguish using at most C coins

Transition formula:

$$dp2[C] = \begin{cases} 1, & \text{if } C < \max(A, B) \\ = dp2[C - A] + dp2[C - B], & \text{otherwise} \end{cases}$$

Reason:

- Pay A coins -> distinguish dp2[C – A] numbers
- Pay B coins -> distinguish dp2[C – B] numbers

Answer equals the first C such that dp[C] >= N

Time complexity: O(max(A, B) * $\lceil \log_2 N \rceil$)

The End