TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

Báo cáo môn: TÍNH TOÁN KHOA HỌC (IT4110)

ĐỀ TÀI: Sử dụng quy hoạch tuyến tính giải bài toán Codeforces 605C

NHÓM 8

Nguyễn Huy Vũ	20235879
Phan Huy Hoàng	20235729
Phạm Thanh Hải	20235701
Phan Công Minh	20235785
Phạm Lâm Tùng	20235867
Vũ Huy Hoàng	20235732

Giảng viên hướng dẫn: TS. Vũ Văn Thiệu

Khoa: Công nghệ thông tin Việt-Nhật

Trường: Công nghệ Thông tin và Truyền thông

HÀ NỘI, 12/2024

LỜI MỞ ĐẦU

Bản báo cáo đưa ra ứng dụng của các kỹ thuật Lập trình tuyến tính (Linear Programming) để giải quyết một vấn đề đầy thách thức được trình bày trên nền tảng lập trình cạnh tranh Codeforces. Lập trình tuyến tính, một phương pháp tối ưu hóa toán học, được sử dụng để mô hình hóa và giải quyết một loạt các vấn đề liên quan đến việc tối ưu hóa hàm mục tiêu theo một tập hợp các ràng buộc tuyến tính.

Bài toán được chọn từ Codeforces chứng minh cách các kỹ thuật tính toán trong quy hoạch tuyến tính có thể giải quyết các tình huống tối ưu hóa trong thế giới thực, bao gồm phân bổ tài nguyên, lập lịch và giảm thiểu chi phí. Báo cáo trình bày chi tiết quy trình xây dựng vấn đề, bao gồm chuyển đổi vấn đề thành mô hình lập trình tuyến tính, xác định hàm mục tiêu và xác định các ràng buộc có liên quan. Báo cáo cũng thảo luận về việc sử dụng các trình giải quy hoạch tuyến tính tiêu chuẩn, chẳng hạn như phương pháp đơn hình, đơn hình 2 pha hay lý thuyết đối ngẫu, để tính toán hiệu quả các giải pháp.

Hơn nữa, chúng em phân tích tính phức tạp và hiệu suất của phương pháp quy hoạch tuyến tính trong bối cảnh lập trình cạnh tranh, so sánh nó với các kỹ thuật thay thế và đánh giá tính thực tiễn của nó trong các môi trường bị hạn chế về thời gian. Các kết quả nhấn mạnh sức mạnh và tính linh hoạt của lập trình tuyến tính trong việc cung cấp các giải pháp tối ưu hoặc gần tối ưu cho các vấn đề phức tạp, làm nổi bật giá trị của nó trong cả nghiên cứu học thuật và các ứng dụng thực tế.

MỤC LỤC

CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI (TỔNG QUAN VỀ QUY HOẠCH TUYẾN TÍNH)

1. Đặt vấn đề

- Bài toán lợi nhuận: Một công ty muốn sản xuất hai loại xe đạp (L_1 và L_2). Xe L_1, L_2 sản xuất số lượng lần lượt là x và y. Cần 3 giờ để hoàn thành một chiếc L_1, mỗi chiếc cho lợi nhuận 100$, 4h để hoàn thành một chiếc L_2 và cho lợi nhuận 200$. Công ty muốn đặt ra yêu cầu: tổng số giờ sản xuất cho cả hai loại xe không vượt quá 60 giờ; và số lượng xe L_1 gấp ít nhất hai lần số lượng xe L_2. Biết hàm lợi nhuận có dạng P = 100x + 200y. Hãy tìm x, y để P đạt GTLN.

2. Các khái niệm cơ bản

a. Quy hoạch tuyến tính

Quy hoạch tuyến tính (QHTT) – linear programming là một phương pháp tối ưu hoá trong toán học, nhằm tìm ra giá trị lớn nhất hoặc nhỏ nhất của một hàm tuyến tính khi các biến số trong hàm này bị ràng buộc bởi một số điều kiện tuyến tính.

Ví dụ: Giả sử muốn tối đa hoá lợi nhuận cho một doanh nghiệp sản xuất, trong đó có hai sản phẩm A và B. Lợi nhuận từ mỗi sản phẩm là khác nhau và công ty bị ràng buộc bởi các yếu tố như nguyên liệu, thời gian sản xuất có hạn. QHTT sẽ giúp ta xác định nên sản xuất bao nhiêu sản phẩm A và B để lợi nhuận đạt mức cao nhất trong phạm vi các điều kiện có sẵn.

Ứng dụng của quy hoạch tuyến tính rất rộng, từ kinh tế, kỹ thuật đến logistics và nghiên cứu khoa học, …. Nó giúp giải quyết các vấn đề liên quan đến tối ưu hóa nguồn lực và ra quyết định một cách hiệu quả.

b. Các yếu tố trong QHTT

- Hàm mục tiêu: Đây là hàm số tuyến tính muốn tối ưu hóa, có thể là để tối đa hóa hoặc tối thiểu hóa. Ví dụ, trong bài toán kinh doanh, hàm mục tiêu có thể là tổng lợi nhuận.

- Các biến quyết định: Đây là những biến số cần xác định giá trị để đạt được mục tiêu tối ưu hóa. Trong bài toán sản xuất, chúng có thể đại diện cho số lượng các sản phẩm cần sản xuất.

- Các ràng buộc tuyến tính: Đây là các điều kiện tuyến tính áp đặt lên các biến quyết định. Các ràng buộc này có thể bao gồm các giới hạn về tài nguyên (nguyên liệu, thời gian, lao động, v.v.) hoặc các yêu cầu khác. Ví dụ, mỗi sản phẩm có thể cần một lượng nguyên liệu nhất định và thời gian sản xuất.

- Điều kiện không âm: Thông thường, các biến quyết định phải không âm (tức là không nhỏ hơn 0), bởi không thể sản xuất một số lượng sản phẩm âm. Điều này đảm bảo tính thực tế của bài toán.

- Tập nghiệm khả thi: Đây là tập hợp tất cả các giá trị của các biến quyết định thỏa mãn các ràng buộc tuyến tính. Tập nghiệm khả thi phải được xác định rõ ràng để tìm được nghiệm tối ưu.

- Nghiệm tối ưu: Đây là giá trị của các biến quyết định mà tại đó hàm mục tiêu đạt giá trị tối ưu nhất (lớn nhất hoặc nhỏ nhất) trong phạm vi các ràng buộc đã đặt ra.

CHƯƠNG 2. BÀI TOÁN VÀ NỀN TẢNG LÝ THUYẾT

1. Bài toán và mô hình

a. Bài toán

Mikhail muốn trở thành một lập trình viên giỏi và có được một căn hộ ở Moscow. Để trở thành lập trình viên giỏi, anh cần ít nhất p điểm kinh nghiệm. Một căn hộ ở Moscow mất q đô la. Anh ấy đã đăng ký làm việc tại một trang web freelance. Có n dự án khác nhau, tham gia dự án thứ i sẽ tăng kinh nghiệm a_i mỗi ngày và kiếm b_i đô la mỗi ngày. Mikhail có thể dừng làm một dự án bất kỳ lúc nào và chuyển sang làm dự án khác.

Tìm số ngày tối thiểu anh ấy cần để thực hiện ước mơ của mình?

* Dữ kiện và ví dụ:

Dữ kiện: $1 \le n \le 10^5$ $1 \le p, q \le 10^6$ $1 \le a_i , b_i \le 10^6$

Ví dụ:

Input	Output
3 20 20 6 2 1 3 2 6	5.000000000000000

b. Mô hình

Với $a_1, a_2, \ldots, a_n$ và $b_1, b_2, \ldots, b_n$ cho trước, tìm cực tiểu của hàm số:

$$f(x) = \sum_{i=1}^{n} t_1 + t_2 + \ldots + t_n$$

$$s.t. \sum_{i=1}^{n} a_i t_i \geq p$$

$$\sum_{i=1}^{n} b_i t_i \geq q$$

$$t_i \geq 0, \forall i = 1, 2, \ldots, n$$

2. Nền tảng lý thuyết

a. Phương pháp đơn hình 2 pha

- Tổng quan: Thuật toán đơn hình 2 pha là phương pháp giải quyết bài toán quy hoạch tuyến tính khi bài toán không có giải pháp khả thi ban đầu.

- Ứng dụng: Tối ưu hóa các hàm mục tiêu trong doanh nghiệp, lập kế hoạch tài nguyên; giải bài toán có rằng buộc phức tạp.

- Các bước tổng quát:

+ Pha 1: Xác định phương án cơ sở chấp nhận được:

- Xây dựng bài toán phụ.
- Thiết lập bảng đơn hình.
- Sử dụng thuật toán đơn hình đến khi đạt được phương án cơ sở chấp nhận được.

+ Pha 2: Tính toán giải pháp tối ưu.

- Bắt đầu từ kết quả thu được từ pha 1 và loại bỏ các biến được thêm vào hàm.
- Tiếp tục thực hiện thuật toán cho tới khi đạt được kết quả tối ưu.

b. Lý thuyết đối ngẫu

- Xây dựng bài toán đối ngẫu:

- Bài toán gốc:

$$f(x_1, x_2, \dots, x_n) = \sum_{i=1}^{n} x_i \rightarrow min$$

Điều kiện: $\Sigma_{i=1}^{n}(a_i \cdot x_i) \geq p$; $\Sigma_{i=1}^{n}(b_i \cdot x_i) \geq q$; $x_i \geq 0$;

- Bài toán đối ngẫu:

$$g(y_1, y_2) = py_1 + qy_2 \rightarrow \max$$

Điều kiện: $a_i \cdot y_1 + b_i \cdot y_2 \leq 1 \; \forall i \in [1, n]$; $y_1, y_2 \geq 0$;

- Định lý đối ngẫu:

+ Nếu bài toán quy hoạch tuyến tính có phương án tối ưu thì bài toán đối ngẫu của nó cũng có phương án tối ưu và giá trị tối ưu của chúng là bằng nhau.

+ Đối với cặp bài toán gốc đối ngẫu của quy hoạch tuyến tính chỉ có thể xảy ra một trong 3 tình huống sau đây:

Cả hai đều có phương án tối ưu và giá trị tối ưu của chúng là bằng nhau.

Cả hai đều không có phương án chấp nhận được.

Bài toán này có hàm mục tiêu không bị chặn còn bài toán kia không có phương án chấp nhận được.

CHƯƠNG 3: PHƯƠNG ÁN THỰC HIỆN

1. Sử dụng thư viện Python

a. Giới thiệu thư viện PuLP

- PuLP là một thư viện Python để giải các bài toán quy hoạch tuyến tính.

- Cấu trúc thường thấy của một chương trình giải bài toán QHTT dùng PuLP:

```python
from pulp import LpProblem, LpVariable, LpMaximize, value

model = LpProblem(name="Example", sense=LpMaximize)

x = LpVariable(name="x", lowBound=0, cat="Integer")
y = LpVariable(name="y", lowBound=0, cat="Integer")

model += 3 * x + 2 * y, "Objective_Function"

model += 2 * x + y <= 8, "Constraint_1"
model += x + 2 * y <= 6, "Constraint_2"

model.solve()

print("Trạng thái lời giải:", model.status)
print(f"Giá trị tối ưu của x: {x.varValue}")
print(f"Giá trị tối ưu của y: {y.varValue}")
print(f"Giá trị hàm mục tiêu: {value(model.objective)}")
```

b. Giải bài toán

```python
from pulp import *

# Read input
n, p, q = map(int, input().split())  # Number of projects, required experience, required money

a = []  # Experience gained per day
b = []  # Money earned per day

for _ in range(n):
    ai, bi = map(int, input().split())
    a.append(ai)
    b.append(bi)

# Define the Linear Programming problem
lp = LpProblem("Freelancer_Dreams", LpMinimize)

# Define variables (days worked on each project)
days = LpVariable.dicts("Days", range(n), lowBound=0, cat="Continuous")

# Objective function: Minimize total days worked
lp += lpSum(days[i] for i in range(n))

# Constraint 1: Total experience requirement
lp += lpSum(a[i] * days[i] for i in range(n)) >= p

# Constraint 2: Total money requirement
lp += lpSum(b[i] * days[i] for i in range(n)) >= q

# Solve the LP
status = lp.solve(PULP_CBC_CMD(msg=0))

# Output the result
if status == LpStatusOptimal:
    print(value(lp.objective))
else:
    print("No solution")
```

2. Sử dụng thuật toán đơn hình 2 pha

- Trong code giả C dưới đây sẽ chỉ viết hàm main() làm wrapper cho phần code LPSolver đã có sẵn

```cpp
// LPSolver: Two-phase simplex algorithm for solving linear programs of the form
// maximize c^T x
// subject to Ax <= b
// x >= 0
//
// INPUT:
// A -- an m x n matrix
// b -- an m-dimensional vector
// c -- an n-dimensional vector
// x -- a vector where the optimal solution will be stored
//
// OUTPUT:
// Value of the optimal solution (infinity if unbounded above, NaN if infeasible)

int main() {
    int n, p, q;
    scanf("%d %d %d", &n, &p, &q);

    double A[2][n] = { -1.0 };
    double b[2] = { -p, -q };
    double c[n] = { -1.0 };

    for (int i = 0; i < n; i++) {
        scanf("%lf %lf", &A[0][i], &A[1][i]);
        A[0][i] = -A[0][i];
        A[1][i] = -A[1][i];
    }

    LPSolver solver(A, b, c);
    double x[n];
    double ans = solver.solve(x);

    printf("%.15lf\n", ans);
}
```

3. Sử dụng lý thuyết đối ngẫu

- Bài toán trở thành: Tìm cực đại của hàm số:

$$g(y_1, y_2) = py_1 + qy_2$$
$$s.t.\ y_1 a_i + y_2 b_i \leq 1, \forall i = 1, 2, \ldots, n$$
$$y_1, y_2 \geq 0$$

- Mã giả giải bài toán:

Algorithm Giải bài toán 605C

Require: n, p, q, $a[1..n]$, $b[1..n]$

1: $trai \leftarrow 0, phai \leftarrow min(1, \frac{1}{\max(a)})$ {Khởi tạo biên của bài toán}
2: **while** $phai - trai > \epsilon$ **do** {Tìm kiếm tam phân}
3: $mid1 \leftarrow trai + \frac{phai-trai}{3}, mid2 \leftarrow trai + \frac{2(phai-trai)}{3}$
4: **if** $f(mid1) < f(mid2)$ **then** {f(x) tìm y lớn nhất sao cho các bđt thỏa mãn}
5: $phai \leftarrow mid2$
6: **else**
7: $trai \leftarrow mid1$
8: **end if**
9: **end while**
10: **return** $f(trai)$

- Chương trình C hoàn chỉnh:

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define N 100005
int n, a[N], b[N], p, q;
double left_bound = 0, right_bound = 1, y, mid1, mid2, delta;
// This function calculates the maximum possible value of the objective function for a fixed $y_1=x$
double F(double x) {
    y = 1;
    for (int i = 1; i <= n; i++)
        y = fmin(y, (1 - a[i] * x) / b[i]); // Original function is ax + by <= 1
        // As the (total) graph is convex, we can use the minimum value of y to calculate the maximum value of ax + by, so that all inequalities are satisfied
    return x * p + q * y;
}

int main() {
    scanf("%d %d %d", &n, &p, &q);
    for (int i = 1; i <= n; i++) {
        scanf("%d %d", &a[i], &b[i]);
        right_bound = fmin(right_bound, 1.0 / a[i]);
        // As the graph is convex, the maximum value of x is the minimum value of 1 / a[i]
    }
    // Perform ternary search
    while (right_bound - left_bound > 1e-15) { // $\varepsilon = 1e-15$
        delta = (right_bound - left_bound) / 3;
        mid1 = left_bound + delta;
        mid2 = right_bound - delta;
        if (F(mid1) > F(mid2)) {
            right_bound = mid2;
        } else {
            left_bound = mid1;
        }
    }
    printf("%.15f\n", F(left_bound));
    return 0;
}
```

- Kết quả thu được:

Author	Problem	Lang	Verdict	Time	Memory
Manager: phanhoang1366	529572B - 48	GNU C11	Accepted	140 ms	868 KB

CHƯƠNG 4: PHÂN TÍCH CÁC PHƯƠNG ÁN

1. Nhược điểm lớn của ý tưởng dùng thư viện Python

- Hỗ trợ: Trình chấm online nổi tiếng như Codeforces đều không hỗ trợ thư viện ngoài, bao gồm SciPy và NumPy.

- Khó phân tích: việc dùng thư viện có sẵn sẽ làm việc phân tích độ phức tạp khó hơn nhiều.

- Thời gian không tối ưu: Với số dự án n = 100000 cần thời gian chạy 2-4s, không phù hợp nếu bài toán yêu cầu độ nhanh (sẽ bị Time Limit Exceeded).

=> Như vậy, chương trình này phù hợp để kiểm tra tính đúng đắn của 1 thuật toán nào đó tốt hơn.

2. So sánh 2 phương pháp: Phương pháp đơn hình 2 pha và Lý thuyết đối ngẫu

- Phương pháp đơn hình 2 pha:
  - Độ phức tạp của thuật toán: Tệ nhất là $O(2^n)$, với $n = 100000$ thì không thể giải quyết được.
  - Dễ cài đặt do tính "mì ăn liền"
- Phương pháp sử dụng tìm kiếm tam phân trong mặt phẳng:
  - Độ phức tạp của thuật toán: $O(n \log \frac{1}{\epsilon})$, với ϵ là sai số mong muốn.
  - Yêu cầu có kiến thức và nhận thấy cần sự chuyển đổi bài toán (do đôi khi bài toán đối ngẫu không đơn giản hơn bài toán gốc).

CHƯƠNG 5: ỨNG DỤNG QUY HOẠCH TUYẾN TÍNH

1. Thực phẩm và nông nghiệp

Nông dân áp dụng các kỹ thuật lập trình tuyến tính vào công việc của họ. Bằng cách xác định loại cây trồng nào họ nên trồng, số lượng cây trồng và cách sử dụng hiệu quả, nông dân có thể tăng doanh thu của mình.

Trong dinh dưỡng, lập trình tuyến tính cung cấp một công cụ mạnh mẽ để hỗ trợ lập kế hoạch cho nhu cầu ăn kiêng. Để cung cấp giỏ thực phẩm lành mạnh, giá rẻ cho các gia đình có nhu cầu, các chuyên gia dinh dưỡng có thể sử dụng lập trình tuyến tính. Các ràng buộc có thể bao gồm hướng dẫn về chế độ ăn uống, hướng dẫn về chất dinh dưỡng, khả năng chấp nhận của văn hóa hoặc một số kết hợp của chúng. Mô hình toán học cung cấp hỗ trợ để tính toán các loại thực phẩm cần thiết để cung cấp dinh dưỡng với chi phí thấp, nhằm ngăn ngừa bệnh không lây nhiễm. Dữ liệu thực phẩm chưa qua chế biến và giá cả là cần thiết cho các phép tính như vậy, đồng thời tôn trọng các khía cạnh văn hóa của các loại thực phẩm. Hàm mục tiêu là tổng chi phí của giỏ thực phẩm. Lập trình tuyến tính cũng cho phép thay đổi thời gian cho tần suất tạo ra các giỏ thực phẩm như vậy.

2. Ứng dụng trong kỹ thuật

Các kỹ sư cũng sử dụng lập trình tuyến tính để giúp giải quyết các vấn đề về thiết kế và sản xuất. Ví dụ, trong lưới cánh máy bay, các kỹ sư tìm cách tối ưu hóa hình dạng khí động học. Điều này cho phép giảm hệ số cản của cánh máy bay. Các ràng buộc có thể bao gồm hệ số nâng, độ dày tối đa tương đối, bán kính mũi và góc cạnh sau. Tối ưu hóa hình dạng tìm cách tạo ra cánh máy bay không bị sốc với hình dạng khả thi. Do đó, lập trình tuyến tính cung cấp cho các kỹ sư một công cụ thiết yếu để tối ưu hóa hình dạng.

3. Tối ưu hóa vận tải

Các hệ thống vận tải dựa vào lập trình tuyến tính để tiết kiệm chi phí và thời gian. Các tuyến xe buýt và tàu hỏa phải tính đến lịch trình, thời gian di chuyển và hành khách. Các

hãng hàng không sử dụng lập trình tuyến tính để tối ưu hóa lợi nhuận của họ theo giá ghế khác nhau và nhu cầu của khách hàng. Các hãng hàng không cũng sử dụng lập trình tuyến tính để lập lịch trình cho phi công và các tuyến đường. Tối ưu hóa thông qua lập trình tuyến tính giúp tăng hiệu quả của các hãng hàng không và giảm chi phí.

4. Sản xuất hiệu quả

Sản xuất đòi hỏi phải chuyển đổi nguyên liệu thô thành sản phẩm để tối đa hóa doanh thu của công ty. Mỗi bước của quy trình sản xuất phải hoạt động hiệu quả để đạt được mục tiêu đó. Ví dụ, nguyên liệu thô phải đi qua nhiều máy khác nhau trong một khoảng thời gian nhất định trên dây chuyền lắp ráp. Để tối đa hóa lợi nhuận, công ty có thể sử dụng biểu thức tuyến tính về lượng nguyên liệu thô cần sử dụng. Các ràng buộc bao gồm thời gian dành cho từng máy. Bất kỳ máy nào tạo ra tình trạng tắc nghẽn đều phải được giải quyết. Lượng sản phẩm được tạo ra có thể bị ảnh hưởng để tối đa hóa lợi nhuận dựa trên nguyên liệu thô và thời gian cần thiết.

5. Ngành năng lượng

Hệ thống lưới điện hiện đại không chỉ kết hợp các hệ thống điện truyền thống mà còn kết hợp cả năng lượng tái tạo như quang điện gió và mặt trời. Để tối ưu hóa nhu cầu về tải điện, cần phải tính đến máy phát điện, đường dây truyền tải và phân phối, cũng như lưu trữ. Đồng thời, chi phí phải duy trì ở mức bền vững để có lợi nhuận. Lập trình tuyến tính cung cấp một phương pháp để tối ưu hóa thiết kế hệ thống điện. Phương pháp này cho phép khớp tải điện trong khoảng cách ngắn nhất giữa thời điểm phát điện và nhu cầu theo thời gian. Lập trình tuyến tính có thể được sử dụng để tối ưu hóa việc khớp tải hoặc tối ưu hóa chi phí, cung cấp một công cụ có giá trị cho ngành năng lượng.

TÀI LIỆU THAM KHẢO

Eiselt, H. A.-L. (2007). *Linear programming and its applications.* Springer Science & Business Media.

"Linear Programming - GeeksforGeeks." 25 Jul. 2024, https://www.geeksforgeeks.org/linear-programming/.

"Graphical Solution of Linear Programming Problems." 09 Aug. 2024, https://www.geeksforgeeks.org/graphical-solution-of-linear-programming-problems/.

"Linear Programming - YouTube." https://www.youtube.com/watch?v=Bzzqx1F23a8.

"Linear Programming and Network Flows | Wiley Online Books." 16 Nov. 2009, https://onlinelibrary.wiley.com/doi/book/10.1002/9780471703778.

"Linear Programming | Brilliant Math & Science Wiki." https://brilliant.org/wiki/linear-programming/.

"Linear Programming: Theory and Applications - Whitman College." https://www.whitman.edu/Documents/Academics/Mathematics/lewis.pdf.

"Linear Programming - Thomas Jefferson High School for Science and ...." https://activities.tjhsst.edu/sct/lectures/2021/2021_03_12_Linear_Programming.pdf.

"Duality in linear programming. Part 1 - Codeforces." https://codeforces.com/topic/105669/en9.

"Duality in linear programming. Part 2 — in competitive programming ...." https://codeforces.net/blog/entry/105789.

"Duality in Linear Programming 4 - MIT - Massachusetts Institute of ...." https://web.mit.edu/15.053/www/AMP-Chapter-04.pdf.

"jaehyunp/stanfordacm: Stanford ACM-ICPC related materials - GitHub." https://github.com/jaehyunp/stanfordacm.

Phụ lục: Mã C giải bằng phương pháp đơn hình 2 pha

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <float.h>

// typedef long double double;

#define EPS 1e-9L

typedef struct {
    int m, n;
    int *B, *N;
    double **D;
} LPSolver;

void swap_int(int *a, int *b) {
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

void LPSolver_init(LPSolver *solver, int m, int n, double **A, double *b, double *c) {
    int i, j;
    solver->m = m;
    solver->n = n;
    solver->B = (int *)malloc(m * sizeof(int));
    solver->N = (int *)malloc((n + 1) * sizeof(int));
    solver->D = (double **)malloc((m + 2) * sizeof(double *));
    for (i = 0; i < m + 2; i++) {
        solver->D[i] = (double *)calloc(n + 2, sizeof(double));
    }
    for (i = 0; i < m; i++)
        for (j = 0; j < n; j++)
            solver->D[i][j] = A[i][j];
    for (i = 0; i < m; i++) {
        solver->B[i] = n + i;
        solver->D[i][n] = -1;
        solver->D[i][n + 1] = b[i];
    }
    for (j = 0; j < n; j++) {
        solver->N[j] = j;
        solver->D[m][j] = -c[j];
    }
```

```
    solver->N[n] = -1;
    solver->D[m + 1][n] = 1;
}

void Pivot(LPSolver *solver, int r, int s) {
    int i, j;
    double inv = 1.0 / solver->D[r][s];
    for (i = 0; i < solver->m + 2; i++) {
        if (i != r) {
            for (j = 0; j < solver->n + 2; j++) {
                if (j != s) {
                    solver->D[i][j] -= solver->D[r][j] * solver->D[i][s] * inv;
                }
            }
        }
    }
    for (j = 0; j < solver->n + 2; j++) {
        if (j != s) {
            solver->D[r][j] *= inv;
        }
    }
    for (i = 0; i < solver->m + 2; i++) {
        if (i != r) {
            solver->D[i][s] *= -inv;
        }
    }
    solver->D[r][s] = inv;
    swap_int(&(solver->B[r]), &(solver->N[s]));
}

int Simplex(LPSolver *solver, int phase) {
    int x = (phase == 1) ? solver->m + 1 : solver->m;
    while (1) {
        int s = -1;
        int j;
        for (j = 0; j <= solver->n; j++) {
            if (phase == 2 && solver->N[j] == -1) continue;
            if (s == -1 || solver->D[x][j] < solver->D[x][s] ||
                (solver->D[x][j] == solver->D[x][s] && solver->N[j] < solver->N[s]))
                s = j;
        }
        if (solver->D[x][s] > -EPS) return 1;
        int r = -1;
        int i;
        for (i = 0; i < solver->m; i++) {
```

```
            if (solver->D[i][s] < EPS) continue;
            double lhs = solver->D[i][solver->n + 1] / solver->D[i][s];
            double rhs = (r == -1) ? 0 : (solver->D[r][solver->n + 1] / solver->D[r][s]);
            if (r == -1 || lhs < rhs ||
              (lhs == rhs && solver->B[i] < solver->B[r]))
               r = i;
        }
        if (r == -1) return 0;
        Pivot(solver, r, s);
    }
}

double Solve(LPSolver *solver, double *x) {
    int i, j, r = 0;
    for (i = 1; i < solver->m; i++)
        if (solver->D[i][solver->n + 1] < solver->D[r][solver->n + 1])
            r = i;
    if (solver->D[r][solver->n + 1] < -EPS) {
        Pivot(solver, r, solver->n);
        if (!Simplex(solver, 1) || solver->D[solver->m + 1][solver->n + 1] < -EPS)
            return -INFINITY;
        for (i = 0; i < solver->m; i++)
            if (solver->B[i] == -1) {
                int s = -1;
                for (j = 0; j <= solver->n; j++)
                    if (s == -1 || solver->D[i][j] < solver->D[i][s] ||
                        (solver->D[i][j] == solver->D[i][s] && solver->N[j] < solver->N[s]))
                        s = j;
                Pivot(solver, i, s);
            }
    }
    if (!Simplex(solver, 2))
        return INFINITY;
    for (j = 0; j < solver->n; j++)
        x[j] = 0.0;
    for (i = 0; i < solver->m; i++)
        if (solver->B[i] < solver->n)
            x[solver->B[i]] = solver->D[i][solver->n + 1];
    return solver->D[solver->m][solver->n + 1];
}

int main() {
    int n, p, q;
    // Read the values of n, p, and q from input
    if (scanf("%d %d %d", &n, &p, &q) != 3) {
```

```
    fprintf(stderr, "Error reading n, p, q.\n");
    return 1;
}

// Allocate memory for the matrix A (2 x n) and initialize it to -1
double **A = (double **)malloc(2 * sizeof(double *));
if (A == NULL) {
    fprintf(stderr, "Memory allocation failed for A.\n");
    return 1;
}
for (int i = 0; i < 2; i++) {
    A[i] = (double *)malloc(n * sizeof(double));
    if (A[i] == NULL) {
        fprintf(stderr, "Memory allocation failed for A[%d].\n", i);
        return 1;
    }
    for (int j = 0; j < n; j++) {
        A[i][j] = -1.0L; // Initialize to -1.0
    }
}

// Allocate and initialize vector b with -p and -q
double b[2];
b[0] = (double)(-p);
b[1] = (double)(-q);

// Allocate and initialize vector c of length n with -1.0
double *c = (double *)malloc(n * sizeof(double));
if (c == NULL) {
    fprintf(stderr, "Memory allocation failed for c.\n");
    return 1;
}
for (int i = 0; i < n; i++) {
    c[i] = -1.0L;
}

// Read the values for A[0][i] and A[1][i], multiply them by -1.0
for (int i = 0; i < n; i++) {
    double a0, a1;
    if (scanf("%lf %lf", &a0, &a1) != 2) {
        fprintf(stderr, "Error reading A[0][%d] and A[1][%d].\n", i, i);
        return 1;
    }
    A[0][i] = -1.0L * a0;
    A[1][i] = -1.0L * a1;
```

```
    }

    // Initialize the LPSolver with the provided A, b, and c
    LPSolver solver;
    LPSolver_init(&solver, 2, n, A, b, c);

    // Allocate memory for the solution vector x
    double *x = (double *)malloc(n * sizeof(double));
    if (x == NULL) {
        fprintf(stderr, "Memory allocation failed for x.\n");
        return 1;
    }

    // Solve the linear program
    double value = Solve(&solver, x);

    // Output the negative of the optimal value with 15 decimal places
    printf("%.15f\n", -value);

    // Free allocated memory
    free(x);
    for (int i = 0; i < 2; i++) {
        free(A[i]);
    }
    free(A);
    free(c);
    free(solver.B);
    free(solver.N);
    for (int i = 0; i < solver.m + 2; i++) {
        free(solver.D[i]);
    }
    free(solver.D);

    return 0;
}
```