BÁO CÁO GIẢI THUẬT

Bài toán: Fucs và gánh nặng mưu sinh

1 Subtask 1: $N \leq 20$

1.1 1. Phương pháp thiết kế thuật toán

- **Phương pháp:** Vét cạn (Brute Force) sử dụng kỹ thuật **Bitmask** (Duyệt toàn bộ tập con).
- **Nguyên lý áp dụng:**
  1. Biểu diễn mỗi tập hợp con các công việc được chọn bằng một số nguyên (mask) từ 0 đến $2^N - 1$. Nếu bit thứ i của mask là 1, công việc thứ i được chọn.
  2. Với mỗi tập con được chọn, ta sắp xếp các công việc theo thứ tự hạn chót tăng dần (EDF - Earliest Deadline First). Đây là chiến thuật tối ưu để kiểm tra xem một tập công việc có thể hoàn thành đúng hạn hay không.
  3. Kiểm tra tính hợp lệ: Duyệt qua danh sách đã sắp xếp, cộng dồn thời gian thực hiện. Nếu tại bất kỳ thời điểm nào tổng thời gian vượt quá hạn chót của công việc hiện tại, tập con đó không hợp lệ.
  4. Cập nhật kết quả: Nếu tập con hợp lệ, so sánh tổng lợi nhuận với lợi nhuận lớn nhất hiện có để cập nhật.

1.2 2. Tính phù hợp của phương pháp

- Giới hạn dữ liệu của Subtask 1 là $N \leq 20$.
- Số lượng tập con tối đa là $2^{20} \approx 1.048.576$.
- Với mỗi tập con, chi phí kiểm tra (bao gồm sắp xếp) là nhỏ. Do đó, máy tính có thể xử lý toàn bộ không gian nghiệm trong thời gian dưới 1 giây.
- Phương pháp này đảm bảo tìm ra nghiệm **tối ưu toàn cục** (Global Optimum) chính xác tuyệt đối.

1.3 3. Phân tích độ phức tạp

- **Độ phức tạp thời gian:** $O(2^N \cdot N \log N)$.
  - Có 2^N trạng thái mask.
  - Với mỗi trạng thái, ta mất $O(N \log N)$ để sắp xếp theo Deadline và $O(N)$ để kiểm tra tính hợp lệ.
- **Độ phức tạp không gian:** $O(N)$ để lưu trữ vector các công việc trong tập con hiện tại.

---

2 Subtask 2: N lớn (Giải thuật Heuristic)

2.1 1. Phương pháp thiết kế thuật toán

- **Phương pháp:** Metaheuristic - **Iterated Greedy (IG)** kết hợp với cơ chế chấp nhận của **Simulated Annealing (SA)**.
- **Nguyên lý áp dụng:**
  1. **Khởi tạo đa điểm (Multi-start):** Tạo lời giải ban đầu bằng 3 chiến thuật tham lam khác nhau (theo Profit, Density, Tightness) và chọn cái tốt nhất.
  2. **Vòng lặp Iterated Greedy:**
     - *Destruction (Phá hủy):* Loại bỏ một phần các công việc khỏi lời giải hiện tại. Sử dụng hai chế độ: xóa ngẫu nhiên rải rác và xóa theo khối (Block Removal) để tạo khoảng trống thời gian lớn.
     - *Construction (Xây dựng):* Thử chèn lại các công việc chưa được chọn vào lịch trình. Sử dụng chiến thuật hỗn hợp: 30% ưu tiên Density, 30% ưu tiên Profit, 40% Random Shuffle để tăng tính khám phá.
  3. **Cơ chế chấp nhận (Acceptance):** Sử dụng công thức của Simulated Annealing. Luôn chấp nhận nghiệm tốt hơn. Nghiệm xấu hơn được chấp nhận với xác suất $P = e^{\Delta/T}$, trong đó T giảm dần theo thời gian.

2.2 2. Tính phù hợp của phương pháp

- **Tại sao chọn Iterated Greedy (IG)?**
  - Bài toán lập lịch thường có cấu trúc mà việc "phá đi xây lại"một phần nhỏ (Destruction/Construc giúp thoát khỏi bẫy tối ưu cục bộ rất hiệu quả mà vẫn giữ được các đặc tính tốt của lời giải cũ.
  - IG đơn giản, cài đặt nhanh và tốn ít bộ nhớ, cho phép thực hiện hàng triệu vòng lặp trong giới hạn 5 giây.
- **Tại sao kết hợp Simulated Annealing?**
  - Nếu chỉ dùng Greedy đơn thuần, thuật toán sẽ kẹt ngay ở đỉnh đồi đầu tiên. Cơ chế xác suất của SA cho phép thuật toán "chấp nhận lùi bước"để tìm ra hướng đi mới tốt hơn về lâu dài.
- **Cấu hình tham số:**
  - *Time Limit:* 4.8 giây (An toàn cho giới hạn 5s của đề bài).
  - *Temperature (T):* Khởi tạo $T = 500$, $T_{min} = 0.05$.
  - *Cooling rate (α):* 0.9995. Lý do chọn α rất gần 1 là vì thời gian cho phép rất dài (5s), ta muốn quá trình giảm nhiệt diễn ra cực chậm để thuật toán dò tìm kỹ lưỡng không gian nghiệm (Deep Exploration).

2.3 3. Phân tích độ phức tạp

- **Độ phức tạp thời gian:** Phụ thuộc vào số vòng lặp K thực hiện được trong 5 giây.

$$O(K \cdot N)$$

  Trong đó mỗi bước chèn/xóa mất khoảng $O(N)$ (do sử dụng vector insert/erase và linear scan kiểm tra). Với $N = 500$ và thời gian 5s, K có thể lên tới hàng trăm nghìn hoặc triệu, đảm bảo độ hội tụ.

- **Độ phức tạp không gian:** $O(N)$ để lưu trữ danh sách công việc và lời giải hiện tại.

Phụ lục: Mã nguồn cài đặt (Python)

Dưới đây là mã nguồn Python cài đặt cho cả hai Subtask dựa trên phương pháp đã phân tích.

```python
import sys
import math
import time
import random
from bisect import bisect_right

# Tang gioi han de quy neu can
sys.setrecursionlimit(2000)

class Job:
    def __init__(self, id, r, t, p):
        self.id = id
        self.r = r
        self.t = t
        self.p = p
        # Tranh chia cho 0
        self.density = p / r if r > 0 else 1e18

    # Dinh nghia key so sanh cho ham sort/bisect
    def sort_key(self):
        return (self.t, self.r)

def solve():
    # --- Doc Input ---
    input_data = sys.stdin.read().split()
    if not input_data:
        return

    iterator = iter(input_data)
    try:
        N = int(next(iterator))
    except StopIteration:
        return

    jobs = []
    for i in range(N):
        r = int(next(iterator))
        t = int(next(iterator))
        p = int(next(iterator))
        jobs.append(Job(i + 1, r, t, p))

    # --- Cac ham bo tro ---
    def is_valid_schedule(schedule):
        """Kiem tra tinh hop le cua lich trinh (O(N))"""
        current_time = 0
        for job in schedule:
            current_time += job.r
            if current_time > job.t:
                return False
        return True

    def try_insert_sorted(schedule, candidate, current_profit):
        """Thu chen job vao vi tri dung theo deadline."""
        idx = 0
        n_sched = len(schedule)

```

```
        insert_pos = n_sched
        cand_key = candidate.sort_key()

        # Tim vi tri chen de giu thu tu deadline
        for i in range(n_sched):
            if schedule[i].sort_key() >= cand_key:
                insert_pos = i
                break

        schedule.insert(insert_pos, candidate)

        if is_valid_schedule(schedule):
            return True, current_profit + candidate.p
        else:
            schedule.pop(insert_pos) # Hoan tac
            return False, current_profit

    # --- SUBTASK 1: EXACT (N <= 20) ---
    def solve_exact():
        max_profit = 0
        best_order = []
        limit = 1 << N

        for mask in range(limit):
            current_subset = []
            current_profit = 0
            for i in range(N):
                if (mask >> i) & 1:
                    current_subset.append(jobs[i])
                    current_profit += jobs[i].p

            if current_profit <= max_profit:
                continue

            # Sort theo Deadline (EDF rule)
            current_subset.sort(key=lambda x: x.sort_key())

            if is_valid_schedule(current_subset):
                max_profit = current_profit
                best_order = [job.id for job in current_subset]

        print(max_profit)
        print(*(best_order))

    # --- SUBTASK 2: HEURISTIC (ITERATED GREEDY + SA) ---
    def solve_heuristic():
        start_time = time.time()

        # 1. Da khoi tao (Multi-start Greedy)
        best_sol = []
        best_profit = -1

        # Cac chien thuat sort: Density, Profit, Tightness
        strategies = [
            lambda x: x.density,        # Cao -> Thap (reverse=True)
            lambda x: x.p,              # Cao -> Thap (reverse=True)
            lambda x: x.t               # Thap -> Cao
        ]
        reverses = [True, True, False]

```

```
        for s in range(3):
            candidate_list = jobs[:]
            candidate_list.sort(key=strategies[s], reverse=reverses[s])

            current_sol = []
            current_profit = 0
            for job in candidate_list:
                success, current_profit = try_insert_sorted(current_sol, job, current_profit)

            if current_profit > best_profit:
                best_profit = current_profit
                best_sol = list(current_sol)

        # 2. Simulated Annealing Loop
        current_sol = list(best_sol)
        current_profit = best_profit

        jobs_by_density = sorted(jobs, key=lambda x: x.density, reverse=True)
        jobs_by_profit = sorted(jobs, key=lambda x: x.p, reverse=True)

        T = 500.0
        alpha = 0.9995 # Giam nhiet cham
        min_T = 0.05
        iter_count = 0

        while True:
            # Check time moi 100 vong lap
            if (iter_count % 100) == 0:
                if time.time() - start_time > 4.8: # Dung o 4.8s
                    break
            iter_count += 1

            neighbor_sol = list(current_sol)
            neighbor_profit = current_profit

            # --- PHASE 1: DESTRUCTION ---
            if neighbor_sol:
                mode = random.randint(0, 1)
                limit_remove = max(1, len(neighbor_sol) // 4)
                remove_count = random.randint(1, limit_remove)

                if mode == 0 or len(neighbor_sol) < 3:
                    # Random Removal
                    for _ in range(remove_count):
                        if not neighbor_sol: break
                        idx = random.randint(0, len(neighbor_sol) - 1)
                        neighbor_profit -= neighbor_sol[idx].p
                        neighbor_sol.pop(idx)
                else:
                    # Block Removal (Xoa chuoi lien nhau)
                    start_idx = random.randint(0, len(neighbor_sol) - 2)
                    end_idx = min(len(neighbor_sol), start_idx + remove_count)
                    for k in range(start_idx, end_idx):
                        neighbor_profit -= neighbor_sol[k].p
                    del neighbor_sol[start_idx:end_idx]

            # --- PHASE 2: CONSTRUCTION ---
```

```
            used_ids = {job.id for job in neighbor_sol}
            strategy = random.randint(0, 99)

            if strategy < 30: # 30%: Priority Density
                for job in jobs_by_density:
                    if job.id not in used_ids:
                        _, neighbor_profit = try_insert_sorted(neighbor_sol, job, neighbor_profit)

            elif strategy < 60: # 30%: Priority Profit
                for job in jobs_by_profit:
                    if job.id not in used_ids:
                        _, neighbor_profit = try_insert_sorted(neighbor_sol, job, neighbor_profit)

            else: # 40%: Random Shuffle (Exploration)
                candidates = [j for j in jobs if j.id not in used_ids]
                random.shuffle(candidates)
                for job in candidates:
                    _, neighbor_profit = try_insert_sorted(neighbor_sol, job, neighbor_profit)

            # --- PHASE 3: ACCEPTANCE ---
            delta = neighbor_profit - current_profit
            accept = False

            if delta >= 0:
                accept = True
            else:
                probability = math.exp(delta / T)
                if random.random() < probability:
                    accept = True

            if accept:
                current_sol = neighbor_sol
                current_profit = neighbor_profit
                if current_profit > best_profit:
                    best_profit = current_profit
                    best_sol = list(current_sol)

            T *= alpha
            if T < min_T: T = min_T

        # Output result
        print(best_profit)
        print(*(job.id for job in best_sol))

    if N <= 20:
        solve_exact()
    else:
        solve_heuristic()

if __name__ == "__main__":
    solve()
```

Listing 1: Mã nguồn giải thuật đầy đủ