Thuật toán ứng dụng
Bài thực hành số 3: Chia để trị

TS. Bùi Quốc Trung, TA. Đặng Xuân Vương

Trường Đại học Bách khoa Hà Nội
Viện Công nghệ thông tin và Truyền thông

Ngày 26 tháng 4 năm 2021

Mục lục

Mục lục

04. AGGRCOW

- Có N chuồng bò và C con bò.

04. AGGRCOW

- Có N chuồng bò và C con bò.
- Chuồng bò thứ i nằm ở tọa độ x_i.

04. AGGRCOW

- Có N chuồng bò và C con bò.
- Chuồng bò thứ i nằm ở tọa độ x_i.
- **Yêu cầu:** Sắp xếp bò vào các chuồng sao cho khoảng cách giữa 2 con bò gần nhau nhất là lớn nhất.

04. AGGRCOW

- Có N chuồng bò và C con bò.
- Chuồng bò thứ i nằm ở tọa độ x_i.
- **Yêu cầu:** Sắp xếp bò vào các chuồng sao cho khoảng cách giữa 2 con bò gần nhau nhất là lớn nhất.
- **Giới hạn:**
  - $2 \leq C \leq N \leq 10^5$.
  - $0 \leq x_i \leq 10^9$.

Thuật toán

- **Thuật toán vét cạn:**

Thuật toán

- **Thuật toán vét cạn:**
  - Chọn ra C trong số N chuồng để xếp bò.

Thuật toán

- **Thuật toán vét cạn:**
  - Chọn ra C trong số N chuồng để xếp bò.
  - Độ phức tạp: $\mathcal{O}(2^N)$.

Thuật toán

- **Thuật toán vét cạn:**
  - Chọn ra C trong số N chuồng để xếp bò.
  - Độ phức tạp: $\mathcal{O}(2^N)$.
- **Lưu ý:** Còn nhiều cách vét cạn khác.

Thuật toán

- **Ý tưởng:**

Thuật toán

- **Ý tưởng:**
  - Gọi D_d là bài toán xếp bò vào chuồng sao cho khoảng cách tối thiểu giữa 2 con bò cạnh nhau là d.

Thuật toán

- **Ý tưởng:**
  - Gọi D_d là bài toán xếp bò vào chuồng sao cho khoảng cách tối thiểu giữa 2 con bò cạnh nhau là d.
  - Thử từng giá trị d, kiểm tra D_d có lời giải hay không.

Thuật toán

- **Vấn đề:**

Thuật toán

- **Vấn đề:**
  - Cách kiểm tra D_d có lời giải hay không?

Thuật toán

- **Vấn đề:**
  - Cách kiểm tra D_d có lời giải hay không?
  - Hạn chế số giá trị d phải thử?

Thuật toán

- **Vấn đề:**
  - Cách kiểm tra D_d có lời giải hay không?
  - Hạn chế số giá trị d phải thử?

Thuật toán

- **Kiểm tra:**

- **Kiểm tra:**
  - Sắp xếp các chuồng theo tọa độ tăng dần.

Thuật toán

- **Kiểm tra:**
  - Sắp xếp các chuồng theo tọa độ tăng dần.
  - Gọi s_i là chuồng chứa con bò thứ i.

Thuật toán

- **Kiểm tra:**
  - Sắp xếp các chuồng theo tọa độ tăng dần.
  - Gọi s_i là chuồng chứa con bò thứ i.
  - Xếp con bò đầu tiên vào chuồng 1 ($s_1 = 1$).

Thuật toán

- **Kiểm tra:**
  - Sắp xếp các chuồng theo tọa độ tăng dần.
  - Gọi s_i là chuồng chứa con bò thứ i.
  - Xếp con bò đầu tiên vào chuồng 1 ($s_1 = 1$).
  - Nếu con bò thứ $i - 1$ đã được xếp vào chuồng s_{i-1}, tìm chuồng $s_i > s_{i-1}$ sao cho $x[s_i] - x[s_{i-1}] \geq d$.

Thuật toán

- **Kiểm tra:**
  - Sắp xếp các chuồng theo tọa độ tăng dần.
  - Gọi s_i là chuồng chứa con bò thứ i.
  - Xếp con bò đầu tiên vào chuồng 1 ($s_1 = 1$).
  - Nếu con bò thứ $i - 1$ đã được xếp vào chuồng s_{i-1}, tìm chuồng $s_i > s_{i-1}$ sao cho $x[s_i] - x[s_{i-1}] \geq d$.
  - Nếu C con bò đều được xếp vào chuồng, bài toán có lời giải.

Thuật toán

- **Kiểm tra:**
  - Sắp xếp các chuồng theo tọa độ tăng dần.
  - Gọi s_i là chuồng chứa con bò thứ i.
  - Xếp con bò đầu tiên vào chuồng 1 ($s_1 = 1$).
  - Nếu con bò thứ $i - 1$ đã được xếp vào chuồng s_{i-1}, tìm chuồng $s_i > s_{i-1}$ sao cho $x[s_i] - x[s_{i-1}] \geq d$.
  - Nếu C con bò đều được xếp vào chuồng, bài toán có lời giải.
  - Độ phức tạp: $\mathcal{O}(N + C)$.

Thuật toán

- **Quan sát:**

Thuật toán

- **Quan sát:**
  - Nếu D_d có lời giải thì $D_x, \forall x \leq d$ cũng có lời giải.

Thuật toán

- **Quan sát:**
  - Nếu D_d có lời giải thì $D_x, \forall x \leq d$ cũng có lời giải.
  - Nếu D_d không có lời giải thì $D_x, \forall x \geq d$ cũng không có lời giải.

Thuật toán

- **Cải tiến:**
  - Tìm d bằng thuật toán tìm kiếm nhị phân.

Thuật toán

- **Mở rộng:**
  - Bài toán tương tự: PIE.

Code

```
int solution() {
    int minD = 0;
    int maxD = x[n] - x[1];
    int res = 0;
    while (minD <= maxD) {
        int d = (minD + maxD) / 2;
        if (check(d)) {
            res = d;
            minD = d + 1;
        } else {
            maxD = d - 1;
        }
    }
    return res;
}
```

Code

```
bool check(int d) {
    s[1] = 1;
    for (int i = 2; i <= c; i++) {
        s[i] = -1;
        int k = s[i-1];
        for (int j = k; j <= n; j++) {
            if (x[j] - x[k] >= d) {
                s[i] = j;
                break;
            }
        }
        if (s[i] == -1) {
            return false;
        }
    }
    return true;
}
```

Code

```
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    int t;
    cin >> t;
    while (t--) {
        cin >> n >> c;
        for (int i = 1; i <= n; i++) {
            cin >> x[i];
        }
        sort(x+1, x+n+1);
        cout << solution() << endl;
    }
    return 0;
}
```

Mục lục

04. FIBWORDS

- Dãy Fibonacci Words của xâu nhị phân được định nghĩa như sau:

$$F(n) = \begin{cases} 0, & \text{if } n = 0 \\ 1, & \text{if } n = 1 \\ F(n-1) + F(n-2), & \text{if } n \geq 2 \end{cases}$$

04. FIBWORDS

- Dãy Fibonacci Words của xâu nhị phân được định nghĩa như sau:

$$F(n) = \begin{cases} 0, & \text{if } n = 0 \\ 1, & \text{if } n = 1 \\ F(n-1) + F(n-2), & \text{if } n \geq 2 \end{cases}$$

- **Yêu cầu:** Cho n và một xâu nhị phân p. Đếm số lần p xuất hiện trong $F(n)$ (các lần xuất hiện này có thể chồng lên nhau).

04. FIBWORDS

- Dãy Fibonacci Words của xâu nhị phân được định nghĩa như sau:

$$F(n) = \begin{cases} 0, & \text{if } n = 0 \\ 1, & \text{if } n = 1 \\ F(n-1) + F(n-2), & \text{if } n \geq 2 \end{cases}$$

- **Yêu cầu:** Cho n và một xâu nhị phân p. Đếm số lần p xuất hiện trong $F(n)$ (các lần xuất hiện này có thể chồng lên nhau).
- **Giới hạn:** $0 \leq n \leq 100$, p có không quá 100000 ký tự, kết quả không vượt quá 2^{63}.

Thuật toán I

- **Thuật toán 1 - Vét cạn:** So sánh xâu p với mọi xâu $f(n)[i..(i+lp)]$ (lp là độ dài xâu p).
- **Thuật toán 2 - Chia để trị:** Xâu $f(n)$ gồm 2 xâu con là $f(n-1)$ và $f(n-2)$.
    - Đếm số lần p xuất hiện trong $f(n-1)$, $f(n-2)$.
    - Đếm số lần p xuất hiện ở đoạn giữa của xâu $f(n)$ (đoạn đầu của p là đoạn cuối của $f(n-1)$, đoạn cuối của p là đoạn đầu của $f(n-2)$).

Thuật toán II

- Đếm số lần p xuất hiện trong $f(i)$ với i nhỏ: Sử dụng **thuật toán 1**.
- Đếm số lần p xuất hiện ở đoạn giữa xâu $f(n)$:
  - Giả sử 2 xâu $f(i-1)$ và $f(i)$ có độ dài lớn hơn độ dài xâu p, $f(i-1)$ có dạng $x..a$, $f(i)$ có dạng $y..b$, trong đó x, y, a, b có độ dài bằng độ dài của p (x và a hay y và b có thể chồng lên nhau).
  - **Nhận xét 1:** $x = y$.
  - **Nhận xét 2:** Nếu $n \equiv i(mod 2)$ thì đoạn giữa của $f(n)$ là $..ax..$, ngược lại, đoạn giữa của $f(n)$ là $..bx..$.

- Cài đặt:
  - **void preprocessing()**: Tính trước các xâu fibonacci word, 2 xâu cuối cùng có độ dài không nhỏ hơn 10^5.
  - **long long count(string s, string p)**: Đếm số lần p xuất hiện trong s theo thuật toán 1.
  - **long long count(int n, string p)**: Đếm số lần p xuất hiện trong $f(n)$ theo thuật toán 2.
  - **long long solve(int n, string p)**:
    - Xử lý trường hợp $f(n)$ có độ dài nhỏ hơn độ dài của p.
    - Khởi tạo mảng c: $c[i]$ là số lần xuất hiện của p trong $f(i)$.
    - Sử dụng hàm count(s, p) để đếm số lần xuất hiện của p trong $f(i)$ và $f(i-1)$ với $f(i-1)$ là fibonacci word đầu tiên có độ dài không nhỏ hơn độ dài của p rồi lưu vào mảng c.
    - Sử dụng hàm count(s, p) để đếm số lần xuất hiện của p trong ax và bx, lưu vào mảng mc.
    - Sử dụng hàm count(n, p) để đếm số lần xuất hiện của p trong $f(n)$.

Code

```
long long solve(int n, string p) {
    int lp = p.size();
    if (n < n_prepare && l[n] < lp) {return 0;}
    for (int j = 0; j <= n; j++) {c[j] = -1;}
    int i = 1;
    while (l[i - 1] < lp) {i++;}
    c[i - 1] = count(f[i - 1], p);
    c[i] = count(f[i], p);
    string x = f[i].substr(0, lp - 1);
    string a =
    f[i - 1].substr(f[i - 1].size() - (lp - 1));
    string b =
    f[i].substr(f[i].size() - (lp - 1));
    mc[i % 2] = count(a + x, p);
    mc[(i + 1) % 2] = count(b + x, p);
    return count(n, p);
}
```

Code

```
long long count(int n, string p) {
    if (c[n] < 0) {
        c[n] = count(n - 1, p)
        + count(n - 2, p)
        + mc[n % 2];
    }
    return c[n];
}
```

Mục lục

04. CLOPAIR

- Có N điểm trên mặt phẳng Oxy.

04. CLOPAIR

- Có N điểm trên mặt phẳng Oxy.
- **Yêu cầu:** Tìm 2 điểm có khoảng cách Euclid nhỏ nhất.

- Có N điểm trên mặt phẳng Oxy.
- **Yêu cầu:** Tìm 2 điểm có khoảng cách Euclid nhỏ nhất.
- **Giới hạn:**
  - $2 \leq N \leq 5 \times 10^4$.
  - $|x_i|, |y_i| \leq 10^6$.

Thuật toán

- **Thuật toán 1:** Xét mọi cặp điểm và tính khoảng cách giữa 2 điểm đó.

Thuật toán

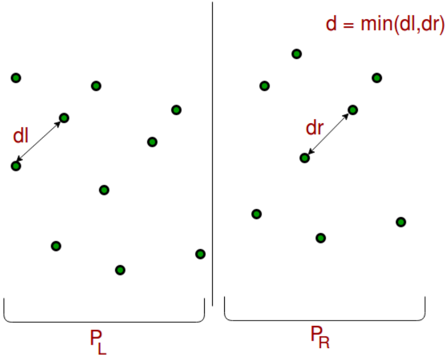
- **Thuật toán 1:** Xét mọi cặp điểm và tính khoảng cách giữa 2 điểm đó.
- **Thuật toán 2:**

Thuật toán

- **Thuật toán 1:** Xét mọi cặp điểm và tính khoảng cách giữa 2 điểm đó.
- **Thuật toán 2:**
  - Chia các điểm đã cho thành 2 tập điểm, 2 điểm cần tìm có thể cùng nằm ở tập 1, cùng nằm ở tập 2 hoặc mỗi điểm nằm ở 1 tập.

Thuật toán

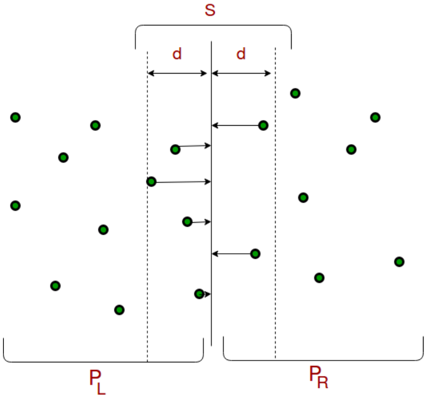
- Sắp xếp các điểm theo hoành độ tăng dần rồi chia tập điểm thành 2 tập có kích thước bằng nhau (ngăn cách bởi x_{median}), gọi là tập P_L và tập P_R.
- Tìm 2 điểm có khoảng cách nhỏ nhất trên mỗi tập điểm, khoảng cách nhỏ nhất lần lượt là dl và dr, gọi $d = min(dl, dr)$.



Hình 1:

Thuật toán

- Nếu 2 điểm có khoảng cách nhỏ nhất không thuộc cùng một tập (P_L hoặc P_R), chúng không được cách đường phân chia 2 tập quá d. Gọi những điểm này là điểm tiềm năng.



Hình 2:

Thuật toán

- Tính khoảng cách của các cặp điểm tiềm năng rồi so sánh với d. Độ phức tạp có thể lên đến $\mathcal{O}(n^2)$.

Thuật toán

- Tính khoảng cách của các cặp điểm tiềm năng rồi so sánh với d. Độ phức tạp có thể lên đến $\mathcal{O}(n^2)$.
- Liệu có thể giảm độ phức tạp của thuật toán xuống $\mathcal{O}(n)$?

Thuật toán

- Giảm độ phức tạp:
  - Sắp xếp các điểm tiềm năng theo tung độ tăng dần.
  - Với mỗi điểm tiềm năng i, chỉ xét các điểm tiềm năng j sao cho $y[j] - y[i] \leq d$.
  - Bằng cách này, số điểm j cần xét tối đa là 6. Chứng minh?

Code

```
struct Point{
    double x, y;
    int id;
    Point(double x = 0, double y = 0, int id = 0) {
        this->x = x;
        this->y = y;
        this->id = id;
    }
} p[NMAX];

struct Solution{
    int id1, id2;
    double distance;
    Solution(int id1 = -1, int id2 = -1, double distance
        this->id1 = id1;
        this->id2 = id2;
        this->distance = distance;
    }
};
```

Code

```
bool cmpX(Point p1, Point p2) {
    return p1.x < p2.x;
}

bool cmpY(Point p1, Point p2) {
    return p1.y < p2.y;
}

double getDistance(Point p1, Point p2) {
    double dx = p1.x - p2.x;
    double dy = p1.y - p2.y;
    return sqrt(dx * dx + dy * dy);
}
```

Code

```
Solution solve(int first, int last) {
    if (first >= last) {
        Solution best = Solution(-1, -1, MAX_VALUE);
        return Solution(-1, -1, MAX_VALUE);
    }
    int mid = (first + last) / 2;
    Solution sl = solve(first, mid);
    Solution sr = solve(mid+1, last);
    Solution best = (sl.distance < sr.distance) ?
        sl : sr;
    double d = best.distance;
    vector<Point> strip;
    for (int i = first; i <= last; i++) {
        if (abs(p[i].x - p[mid].x) <= d) {
            strip.push_back(p[i]);
        }
    }
```

Code

```
    sort(strip.begin(), strip.end(), cmpY);
    for (int i = 0; i < strip.size(); i++) {
        for (int j = i+1; j < strip.size(); j++) {
            if (strip[j].y - strip[i].y > d) {
                break;
            }
            double tmp = getDistance(strip[i], strip[j])
            if (tmp < best.distance) {
                int id1 = strip[i].id;
                int id2 = strip[j].id;
                if (id1 > id2) swap(id1, id2);
                best = Solution(id1, id2, tmp);
            }
        }
    }
    return best;
}
```

Code

```
int main() {
    int N;
    cin >> N;
    for (int i = 0; i < N; i++) {
        cin >> p[i].x >> p[i].y;
        p[i].id = i;
    }
    sort(p, p+N, cmpX);
    Solution sol = solve(0, N-1);
    cout << sol.id1 << ' ' << sol.id2 << ' ' << fixed <<
    return 0;
}
```

Thuật toán ứng dụng
Bài thực hành số 3: Chia để trị

TS. Bùi Quốc Trung, TA. Đặng Xuân Vương

Trường Đại học Bách khoa Hà Nội
Viện Công nghệ thông tin và Truyền thông

Ngày 26 tháng 4 năm 2021