WINTER CAMP VOI2020

Đỗ Phan Thuận

Email/Facebook: thuandp.sinhvien@gmail.com
Bộ môn Khoa Học Máy Tính, Viện CNTT & TT,
Trường Đại Học Bách Khoa Hà Nội.

Ngày 11 tháng 11 năm 2020

Outline

CNTDIV

Cho số nguyên dương N, xét số nguyên dương $T = N \times (N+1) \times (N+2)$. Yêu cầu hãy đếm số ước của T^2 mà nhỏ hơn T và không phải ước của T.

Thuật toán 25 điểm

Đúng cho các test giới hạn số N và số lượng bộ test Q bé, khi đó làm trực tiếp, tính từng ước của T và T^2 trong mỗi test.

Thuật toán 100 điểm

- Sàng nguyên tố đến 10^6 để tính các ước nguyên tố $T = 2^{p_2} * 3^{p_3} * \ldots * k^{p_k}$
- Gọi $cntT$ là số ước của T mà $< T$, gọi $cntTT$ là số ước của T^2 mà $< T$ -> $res = cntTT - cntT$
- Lưu ý 1: số ước của T^2 mà $\leq T$ bằng số ước của T^2 mà $> T$
- Lưu ý 2: số ước của T^2 gấp đôi số ước của T

STRALT

1. Chuỗi S có dạng $n(C)$, trong đó n là số tự nhiên nằm ngay trước dấu ngoặc tròn, được biến đổi thành chuỗi D thu được bằng cách lặp liên tiếp n lần chuỗi C. Ví dụ, với chuỗi $5(ab)$ ta có $n = 5$ và thu được dãy $D = ababababab$.
2. Chuỗi S có dạng $[*C]$ được biến đổi thành một chuỗi palindrom (nghĩa là chuỗi đối xứng) có độ dài chẵn, thu được bằng cách ghép chuỗi C với chuỗi ngược của C. Ví dụ, với chuỗi $[*abc]$, chuỗi palindrom thu được có độ dài chẵn là $abccba$.
3. Chuỗi S có dạng $[C*]$ được biến đổi thành một chuỗi palindrom có độ dài lẻ, thu được bằng cách ghép dãy C với chuỗi ngược của C mà bỏ đi ký tự đầu tiên. Ví dụ, với chuỗi $[abc*]$, chuỗi palindrom thu được có độ dài lẻ là $abcba$.

Một chuỗi được coi là đã được giải nén nếu nó chỉ bao gồm các chữ cái viết thường của bảng chữ cái tiếng Anh.

Yêu cầu: Cho chuỗi đã nén S, hãy giúp Bob xác định số lần biến đổi thuộc 3 kiểu trên, cùng với chuỗi T ban đầu trước khi nén của chuỗi S.

Thuật toán

Sử dụng Stack và xử lý xâu để tách từng dạng ra xử lý lần lượt theo độ ưu tiên từ trong ra ngoài đúng với nguyên lý LIFO của Stack: vào sau ra trước.

TELMOV

Cô kỹ sư Alice đang sống ở trong thiên hà VNOI2020. Trong thiên hà này có N hành tinh khác nhau và M kênh vận chuyển hai chiều dạng (x, y, t) cho phép bạn di chuyển từ hành tinh x đến hành tinh y (hoặc ngược lại) trong t giây.

Nhưng Alice nhận thấy phương pháp vận chuyển này rất kém hiệu quả nên đã phát triển một thiết bị cho phép bạn dịch chuyển từ hành tinh x đến bất kỳ hành tinh y nào khác trong P giây với điều kiện bạn có thể đến hành tinh y đó từ hành tinh x chỉ sử dụng tối đa L kênh vận chuyển.

Thiết bị này hiện mới là bản thử nghiệm nên không thể được sử dụng quá K lần. Alice đang ở hành tinh 1 và muốn biết thời gian tối thiểu để đến hành tinh N.

Yêu cầu: Viết chương trình tính thời gian tối thiểu cần thiết để đến được hành tinh N bắt đầu từ hành tinh 1.

Thuật toán 24 điểm

Đúng với các test thoả mãn điều kiện không được sử dụng phép dịch chuyển và tất cả các kênh vận chuyển đều có thời gian $= 1$. Do đó ta chỉ cần sử dụng thuật toán loang BFS đơn giản để giải quyết.

Thuật toán 50 điểm

Đúng với các test thoả mãn điều kiện không sử dụng phép dịch chuyển. Do đó bài toán chính là bài toán tìm đường đi ngắn nhất trên đồ thị có trọng số và ta chỉ cần sử dụng thuật toán Dijkstra đơn giản để giải quyết.

Thuật toán 50 điểm

Đúng với các test thoả mãn điều kiện không sử dụng phép dịch chuyển. Do đó bài toán chính là bài toán tìm đường đi ngắn nhất trên đồ thị có trọng số và ta chỉ cần sử dụng thuật toán Dijkstra kết hợp với Hàng đợi ưu tiên PQ để giải quyết.

Thuật toán 66 điểm

Đúng với các test thoả mãn điều kiện số lượng hành tinh nhỏ hơn hoặc bằng 300. Do đó có thể chia bài toán thành hai bước

1. Xây dựng lại đồ thị bằng cách bổ sung thêm các cạnh đi được bởi cách dịch chuyển thoả mãn các ràng buộc về số lần sử dụng kênh vận chuyển.
2. Áp dụng thuật toán tìm đường đi ngắn nhất trên đồ thị mới xây dựng.

Thuật toán 100 điểm

- Quan sát thấy các điều kiện hạn chế $L, K \leq 10$ là rất nhỏ. Vì vậy ta có thể lưu lại được các trạng thái dịch chuyển trong quá trình xây dựng đường đi ngắn nhất bởi thuật toán Dijkstra vào mảng ba chiều $10000 \times 10 \times 10$.
- Sửa hàm đánh giá tối ưu của thuật toán Dijkstra thành $F(x, y, z)$, nghĩa là thời gian đến x nhỏ nhất sử dụng y lần dịch chuyển và qua z cạnh tính từ lần dịch chuyển gần nhất. Chuyển trạng thái dựa vào điều kiện không quá K lần dịch chuyển giới hạn cho tham số y và sử dụng tối đa L kênh vận chuyển giới hạn cho tham số z.

ALICEADD

- Cho hai số a và b, hãy viết chương trình bằng C/C++ tính số $c = a + b$
- Lưu ý giới hạn: $a, b < 10^{19}$ dẫn đến c có thể vượt quá khai báo `long long`

Thuật toán

- Chỉ cần khai báo a, b, c kiểu `unsigned long long`, trường hợp tràn số chỉ xảy ra khi a, b có 19 chữ số và c có 20 chữ số

1. Tách $a = a1 \times 10 + a0$
2. Tách $b = b1 \times 10 + b0$
3. Tách $a0 + b0 = c1 \times 10 + c0$
4. In ra liên tiếp $a1 + b1 + c1$ và $c0$

SQUARE

Xét dãy số sau:
$0, 0+1, 0+1+3, 0+1+3+5, \ldots, 0+1+3+\ldots+(2n-1), \ldots$.
Đây là dãy được tạo bởi tổng vài số tự nhiên lẻ đầu tiên và các số hạng của dãy đều là số chính phương (tức là bình phương của một số nguyên): $0, 1, 4, 9, \ldots, n^2, \ldots$.
Tổng quát hóa dãy này bằng cách thay số 0 ở đầu bởi một số nguyên k, như vậy ta được dãy:
$k, k+1, k+1+3, k+1+3+5, \ldots, k+1+3+\ldots+(2n-1), \ldots$.
Tuy nhiên khác với trường hợp $k = 0$ ở trên, dãy này chỉ có một vài số hạng là số chính phương.
Yêu cầu: Cho trước số nguyên k, cần tìm số nguyên không âm nhỏ nhất sao cho bình phương của nó xuất hiện trong dãy số trên.

Thuật toán

Bài toán đưa về tìm x dương nhỏ nhất mà k = (x + y)(x - y)

TAXI

Crab vừa rộng mô hình dịch vụ sang chuyển phát hàng hóa khi xe đang rảnh. Có n gói hàng, gói thứ i muốn chuyển từ vị trí i đến vị trí $i + n$. Cần lập lịch cho xe xuất phát từ vị trí 0, chuyển hết các gói hàng và quay lại vị trí xuất phát. Sức chứa của xe là đủ lớn, do đó gói hàng thứ i sẽ được chuyển nếu ít nhất một lần, lộ trình của xe có đi qua i trước khi đi qua $i + n$. Ví dụ với $n = 3$, lộ trình sau là thỏa mãn: $0 - 1 - 2 - 1 - 5 - 3 - 6 - 4 - 0$

Cho biết độ dài tuyến đường đi lại giữa mọi cặp vị trí, hãy tìm lộ trình của taxi có tổng độ dài các tuyến đường đi qua là nhỏ nhất. Lưu ý, các tuyến đường trong thành phố là đường một chiều nên khoảng cách từ x đến y có thể khác với khoảng cách từ y đến x, và có thể đường đi ngắn nhất x và y không phải là đường đi trực tiếp giữa chúng. Nếu có nhiều lộ trình thỏa mãn có cùng độ dài nhỏ nhất, in ra một lộ trình bất kỳ

Thuật toán

1. Floyd để đường đi thoả mãn bất đẳng thức tam giác
2. QHĐ bitmask hoặc duyệt nhánh cận giống bài toán người du lịch

Bài toán người du lịch (hay người bán hàng)

Applying the Traveling Salesman Problem to Business Analytics

Published on April 2, 2016

Bài toán người du lịch là bài toán NP-khó kinh điển nhưng có rất nhiều ứng dụng trong thực tế, đặc biệt ngày nay với ứng dụng của khoa học dữ liệu và phân tích tài chính.

Mỗi khi một hành trình của người bán hàng kết thúc, các dữ liệu sẽ được phân tích bởi các thuật toán trong khoa học dữ liệu, áp dụng vào ngành phân tích tài chính, từ đó có thể 'hoc máy' các kết quả lịch sử để áp dụng vào kế hoạch phát triển tiếp theo.

Bài toán người du lịch

- Cho một đồ thi n đỉnh và giá trị trọng số $c_{i,j}$ trên mỗi cặp đỉnh i, j. Hãy tìm một chu trình đi qua tất cả các đỉnh của đồ thị, mỗi đỉnh đúng một lần sao cho tổng các trọng số trên chu trình đó là nhỏ nhất
- Đây là bài toán NP-khó, vì vậy không tồn tại thuật toán tất định thời gian đa thức nào hiện biết để giải bài toán này
- Thuật toán duyệt toàn bộ đơn giản duyệt qua toàn bộ các hoán vị các đỉnh cho độ phức tạp là $O(n!)$, nhưng chỉ có thể chạy được đến $n \leq 11$
- Liệu có thể làm tốt hơn với phương pháp qui hoạch động?

Bài toán người du lịch

- Không mất tính tổng quát giả sử chu trình bắt đầu và kết thúc tại đỉnh 0
- Gọi $\mathrm{tsp}(i, S)$ cách sử dụng ít chi phí nhất để đi qua toàn bộ các đỉnh và quay trở lại đỉnh 0, nếu như hiện tại hành trình đang ở tại đỉnh i và người du lịch đã thăm tất cả các đỉnh trong tập S
- Bước cơ sở: $\mathrm{tsp}(i, \text{tập mọi đỉnh}) = c_{i,0}$
- Bước chuyển đệ qui:
  $\mathrm{tsp}(i, S) = \min_{j \notin S} \{ c_{i,j} + \mathrm{tsp}(j, S \cup \{j\}) \}$

Biểu diễn tập hợp bằng Bitmask

- Cho một số lượng nhỏ ($n \leq 30$) phần tử
- Gán nhãn bởi các số nguyên $0, 1, \ldots, n-1$
- Biểu diễn tập hợp các phần tử này bởi một biến nguyên 32-bit
- Phần thử thứ i trong tập được biểu diễn bởi số nguyên x nếu bit thứ i của x là 1
- Ví dụ:
  - Cho tập hợp $\{0, 3, 4\}$
  - `int x = (1<<0) | (1<<3) | (1<<4);`

Biểu diễn tập hợp

- Tập rỗng:

```
0
```

- Tập có một phần tử:

```
1<<i
```

- Tập vũ trụ (nghĩa là tất cả các phần tử):

```
(1<<n)-1
```

- Hợp hai tập:

```
x|y
```

- Giao hai tập:

```
x&y
```

- Phần bù một tập:

```
~x & ((1<<n)-1)
```

Biểu diễn tập hợp

- Kiểm tra một phần tử xuất hiện trong tập hợp:

```
if (x & (1<<i)) {
    // yes
} else {
    // no
}
```

Biểu diễn tập hợp

- Tại sao nên làm như vậy mà không dùng set<int>?
- Biểu diễn đỡ tốn khá nhiều bộ nhớ (nén 32,64,128 lần)
- Tất cả các tập con của tập n phần tử này có thể biểu diễn bởi các số nguyên trong khoảng $0 \dots 2^n - 1$
- Dễ dàng lặp qua tất cả các tập con
- Dễ dàng sử dụng một tập hợp như một chỉ số của một mảng

Bài toán người du lịch

```
const int N = 20;
const int INF = 100000000;
int c[N][N];
int mem[N][1<<N];
memset(mem, -1, sizeof(mem));

int tsp(int i, int S) {
    if (S == ((1 << N) - 1))  return c[i][0];

    if (mem[i][S] != -1) {
        return mem[i][S];
    }
    int res = INF;
    for (int j = 0; j < N; j++) {
        if (S & (1 << j))
            continue;
        res = min(res, c[i][j] + tsp(j, S | (1 << j)));
    }
    mem[i][S] = res;
    return res;
}
```

Bài toán người du lịch

- Như vậy lời giải tối ưu có thể được đưa ra như sau:

- ```
  printf("%d\n", tsp(0, 1<<0));
  ```
  ```

Bài toán người du lịch

- Độ phức tạp tính toán?

Bài toán người du lịch

- Độ phức tạp tính toán?

- Có $n \times 2^n$ khả năng input
- Mỗi input được tính trong thời gian $O(n)$, giả thiết mỗi lời gọi đệ qui chỉ mất $O(1)$
- Thời gian tính tổng cộng là $O(n^2 \times 2^n)$
- Như vậy có thể tính nhanh được với n lên đến 20

Bài toán người du lịch

- Độ phức tạp tính toán?

- Có $n \times 2^n$ khả năng input
- Mỗi input được tính trong thời gian $O(n)$, giả thiết mỗi lời gọi đệ qui chỉ mất $O(1)$
- Thời gian tính tổng cộng là $O(n^2 \times 2^n)$
- Như vậy có thể tính nhanh được với n lên đến 20

- Làm thế nào để đưa ra được chính xác hành trình của người du lịch?

Bài toán người du lịch - Truy vết bằng đệ qui

```
void trace_tsp(int i, int S) {
    printf("%d ", i);
    if (S == ((1 << N) - 1))  return;

    int res = mem[i][S];
    for (int j = 0; j < N; j++) {
        if (S & (1 << j))
            continue;

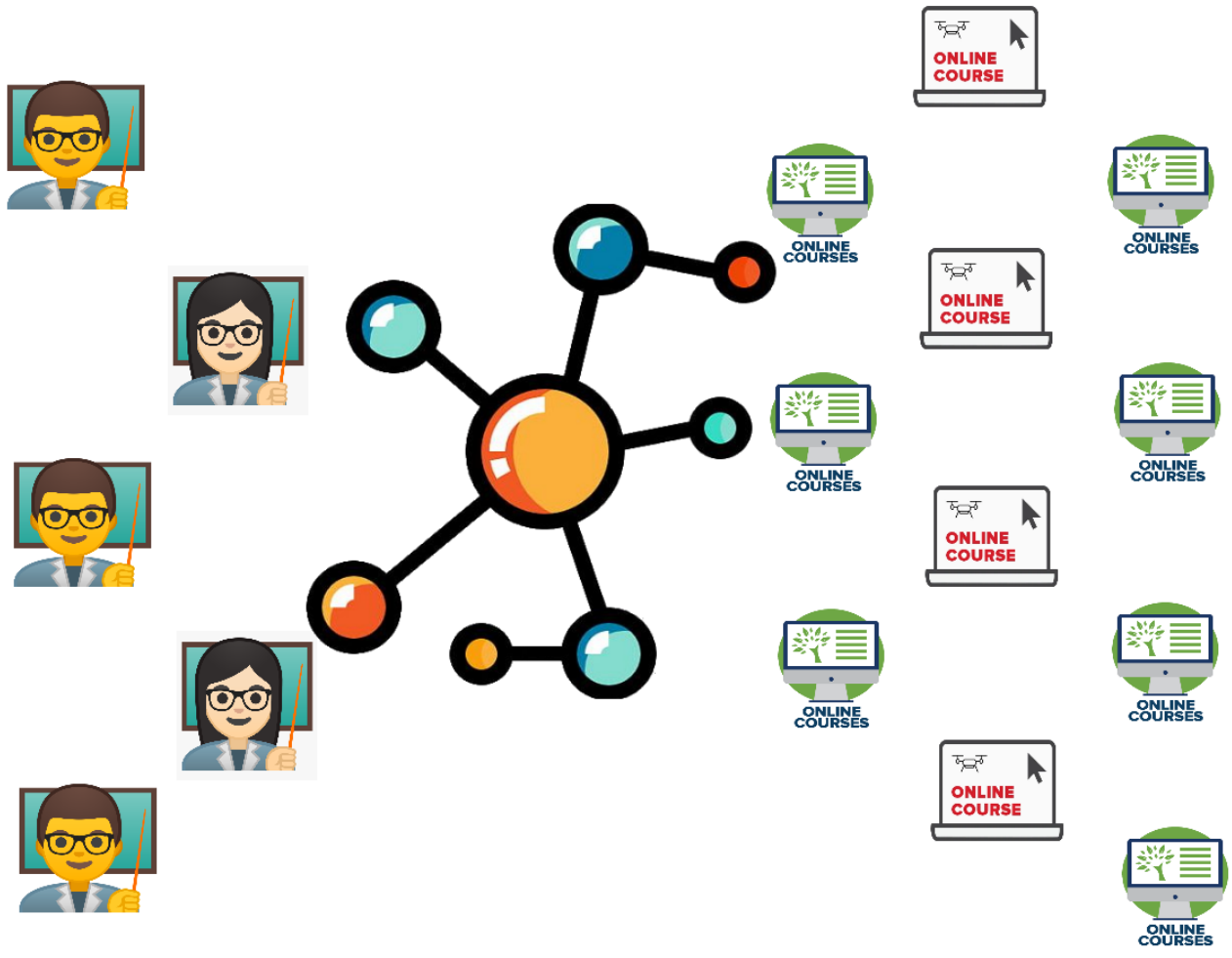
        if (res == c[i][j] + mem[j][S | (1 << j)]){
            trace_tsp(j, S | (j << j));
            break;
        }
    }
}
```

Bài toán người du lịch - Truy vết bằng vòng lặp

```
int answer = tsp(0, 1);
printf("%d\n", answer);
stack<int> s;
s.push(0);
for (int i = 0, S = 1, k = 0; k < n-1; ++k) {
    for (int j = 0; j < n; ++j){
      if ((S & (1 << j))
          && (mem[i][S] == c[i][j] + mem[j][S | (1 << j)])
              s.push(j);
              i = j;
              S = S | (1 << j);
      }
    }
}
while (!s.empty()) {
     printf("%d ", s.back());
     s.pop();
}
```

BCADIF

- Có n môn học và m giáo viên, mỗi giáo viên có danh sách các môn có thể dạy.
- Có danh sách các môn học không thể để cùng một giáo viên dạy do trùng giờ.
- Load của một giáo viên là số môn phải dạy của giáo viên đó.
- **Yêu cầu:** Tìm cách xếp lịch cho giáo viên sao cho Load tối đa của các giáo viên là tối thiểu.


ONLINE COURSE
ONLINE COURSES

Thuật toán

- Sử dụng thuật toán vét cạn, duyệt toàn bộ môn học, xếp giáo viên dạy môn học đó.
- Sử dụng thuật toán nhánh cận:
  - Chọn môn học chưa có người dạy có số giáo viên dạy ít nhất để phân công trước.
  - Nếu phân công cho giáo viên A môn X, mọi môn học trùng lịch với môn X không thể được dạy bởi giáo viên A sau này.
  - Nếu maxCurrentLoad hiện tại lớn hơn minLoad tối ưu thu được trước đó thì không duyệt nữa.

Nhánh cận tốt hơn

- Sử dụng thuật toán nhánh cận:
  - Chọn môn học chưa có người dạy có số giáo viên dạy ít nhất để phân công trước. Sử dụng Hàng đợi ưu tiên **PQ** để lưu tải của các giáo viên, mỗi bước chọn sẽ duyệt nhanh được giáo viên từ tải thấp đến tải cao.
  - Nếu phân công cho giáo viên A môn X, mọi môn học trùng lịch với môn X không thể được dạy bởi giáo viên A sau này. Lưu ý ở bước phân công xong và bước quay lui cần phải cập nhật lại PQ.
  - Nếu maxCurrentLoad hiện tại lớn hơn minLoad tối ưu thu được trước đó thì không duyệt nữa.