Thuật toán Bellman-Ford

TÀI LIỆU

SOCIAL SCIENCES

Thuật toán Bellman-Ford là một thuật toán tính các đường đi ngắn nhất nguồn đơn trong một đồ thị có hướng có trọng số (trong đó một số cung có thể có trọng số âm). Thuật toán Dijkstra giải cùng bài toán này với thời gian chạy thấp hơn, nhưng lại đòi hỏi trọng số của các cung phải có giá trị không âm. Do đó, thuật toán Bellman-Ford thường chỉ được dùng khi có các cung với trọng số âm.

Thuật toán Bellman Ford chạy trong thời gian O(V·E), trong đó V là số đỉnh và E là số cung của đồ thị.

Nội dung thuật toán

```
function BellmanFord(danh_sách_đỉnh, danh_sách_cung, nguồn)
    // hàm yêu cầu đồ thị đưa vào dưới dạng một danh sách đỉnh, một danh sách cung
    // hàm tính các giá trị khoảng_cách và đỉnh_liền_trước của các đỉnh,
    // sao cho các giá trị đỉnh_liền_trước sẽ lưu lại các đường đi ngắn nhất.

    // bước 1: khởi tạo đồ thị
    for each v in danh_sách_đỉnh:
    if v is nguồn then khoảng_cách(v) := 0
    else khoảng_cách(v) := vô cùng
    đỉnh_liền_trước(v) := null
```

```
// bước 2: kết nạp cạnh
    for i from 1 to size(danh_sách_đỉnh):
    for each (u,v) in danh_sách_cung:
    if khoảng_cách(v) > khoảng_cách(u) + trọng_số(u,v) :
    khoảng_cách(v) := khoảng_cách(u) + trọng_số(u,v)
    đỉnh_liền_trước(v) := u

    // bước 3: kiểm tra chu trình âm
    for each (u,v) in danh_sách_cung:
    if khoảng_cách(v) > khoảng_cách(u) + trọng_số(u,v) :
    error "Đồ thị chứa chu trình âm"
```

Chứng minh tính đúng đắn

Tính đúng đắn của thuật toán có thể được chứng minh bằng quy nạp. Thuật toán có thể được phát biểu chính xác theo kiểu quy nạp như sau:

Bổ đề. Sau i lần lặp vòng for:

Nếu Khoảng_cách(u) không có giá trị vô cùng lớn, thì nó bằng độ dài của một đường đi nào đó từ s tới u;
Nếu có một đường đi từ s tới u qua nhiều nhất i cung, thì Khoảng_cách(u) có giá trị không vượt quá độ dài của đường đi ngắn nhất từ s tới u qua tối đa i cung.

Chứng minh.

Trường hợp cơ bản: Xét i=0 và thời điểm trước khi vòng for được chạy lần đầu tiên. Khi đó, với đỉnh nguồn khoảng_cách(nguồn) = 0, điều này đúng. Đối với các đỉnh u khác, khoảng_cách(u) = vô cùng, điều này cũng đúng vì không có đường đi nào từ nguồn đến u qua 0 cung.

Trường hợp quy nạp:

Chứng minh câu 1. Xét thời điểm khi khoảng cách tới một đỉnh được cập nhật bởi công thức khoảng_cách(v) := khoảng_cách(u) + trọng_số(u,v). Theo giả thiết quy nạp, khoảng_cách(u) là độ dài của một đường đi nào đó từ nguồn tới u. Do đó, khoảng_cách(u) + trọng_số(u,v) là độ dài của đường đi từ nguồn tới u rồi tới v.

Chứng minh câu 2: Xét đường đi ngắn nhất từ nguồn tới u qua tối đa i cung. Giả sử v là đỉnh liền ngay trước u trên đường đi này. Khi đó, phần đường đi từ nguồn tới v là đường đi ngắn nhất từ nguồn tới v qua tối đa i-1 cung. Theo giả thuyết quy nạp, khoảng_cách(v) sau i-1 vòng lặp không vượt quá độ dài đường đi này. Do đó, trọng_số(v,u) + khoảng_cách(v) có giá trị không vượt quá độ dài của đường đi từ s tới u. Trong lần lặp thứ i, khoảng_cách(u) được lấy giá trị nhỏ nhất của khoảng_cách(v) + trọng_số(v,u) với mọi v có thể. Do đó, sau i lần lặp, khoảng_cách(u) có giá trị không vượt quá độ dài đường đi ngắn nhất từ nguồn tới u qua tối đa i cung.

Khi i bằng số đỉnh của đồ thị, mỗi đường đi tìm được sẽ là đường đi ngắn nhất toàn cục, trừ khi đồ thị có chu trình âm. Nếu tồn tại chu trình âm mà từ đỉnh nguồn có thể đi đến được thì sẽ không tồn tại đường đi nhỏ nhất (vì mỗi lần đi quanh chu trình âm là một lần giảm trọng số của đường).

Ứng dụng trong định tuyến

Một biến thể phân tán của thuật toán Bellman-Ford được dùng trong các giao thức định tuyến vector khoảng cách, chẳng hạn giao thức RIP (Routing Information Protocol). Đây là biến thể phân tán vì nó liên quan đến các nút mạng (các thiết bị định tuyến) trong một hệ thống tự chủ (autonomous system), ví dụ một tập các mạng IP thuộc sở hữu của một nhà cung cấp dịch vụ Internet (ISP).

Thuật toán gồm các bước sau:

Mỗi nút tính khoảng cách giữa nó và tất cả các nút khác trong hệ thống tự chủ và lưu trữ thông tin này trong một bảng.
Mỗi nút gửi bảng thông tin của mình cho tất cả các nút lân cận.
Khi một nút nhận được các bảng thông tin từ các nút lân cận, nó tính các tuyến đường ngắn nhất tới tất cả các nút khác và cập nhật bảng thông tin của chính mình.

Nhược điểm chính của thuật toán Bellman-Ford trong cấu hình này là

Không nhân rộng tốt
Các thay đổi của tô-pô mạng không được ghi nhận nhanh do các cập nhật được lan truyền theo từng nút một.
Đếm dần đến vô cùng (nếu liên kết hỏng hoặc nút mạng hỏng làm cho một nút bị tách khỏi một tập các nút khác, các nút này vẫn sẽ tiếp tục ước tính khoảng cách tới nút đó và tăng dần giá trị tính được, trong khi đó còn có thể xảy ra việc định tuyến thành vòng tròn)

Cài đặt

C

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

    /* Let INFINITY be an integer value not likely to be
    confused with a real weight, even a negative one. */
    #define INFINITY ((1 << 14)-1)

    typedef struct {
    int source;
    int dest;
    int weight;
    } Edge;

    void BellmanFord(Edge edges[], int edgecount, int nodecount, int source)
    {
    int *distance = malloc(nodecount * sizeof *distance);
    int i, j;
    for (i=0; i < nodecount; ++i)
    distance[i] = INFINITY;
    distance[source] = 0;

    for (i=0; i < nodecount; ++i) {
    for (j=0; j < edgecount; ++j) {
    if (distance[edges[j].source] != INFINITY) {
    int new_distance = distance[edges[j].source] + edges[j].weight;
    if (new_distance < distance[edges[j].dest])
    distance[edges[j].dest] = new_distance;
    }
    }
    }

    for (i=0; i < edgecount; ++i) {
    if (distance[edges[i].dest] > distance[edges[i].source] + edges[i].weight) {
    puts("Negative edge weight cycles detected!");
    free(distance);
    return;
    }
    }

    for (i=0; i < nodecount; ++i) {
    printf("The shortest distance between nodes %d and %d is %d\n",
    source, i, distance[i]);
    }
    free(distance);
    return;
    }

    int main(void)
    {
    /* This test case should produce the distances 2, 4, 7, -2, and 0. */
    Edge edges[10] = {{0,1, 5}, {0,2, 8}, {0,3, -4}, {1,0, -2},
    {2,1, -3}, {2,3, 9}, {3,1, 7}, {3,4, 2},
    {4,0, 6}, {4,2, 7}};
    BellmanFord(edges, 10, 5, 4);
    return 0;
    }
```

0 bình luận Sắp xếp theo Cũ nhất

Thêm bình luận...

Plugin bình luận của Facebook

TẢI VỀ TÁI SỬ DỤNG(/user/reuse/m/e2b821d1/1)

Wikipedia (/profile/8)

0 GIÁO TRÌNH (/PROFILE/8?TYPES=2) | 5771 TÀI LIỆU (/PROFILE/8?TYPES=1)

(/profile/8)

ĐÁNH GIÁ:

0 dựa trên 0 đánh giá

NỘI DUNG CÙNG TÁC GIẢ

Quảng ninh (/m/quang-ninh/990954be)
Tim người (/m/tim-nguoi/c9dd5369)
WTO (/m/wto/22570df1)
Các kiểu bộ nhớ (/m/cac-kieu-bo-nho/f4953d18)
phong trào thơ mới (/m/phong-trao-tho-moi/6a4f7b3d)
Đá (/m/da/c01eb5f4)
Alexandre Yersin (/m/alexandre-yersin/775eda83)
Họ Dầu (/m/ho-dau/201a4385)
Ngôi sao điện ảnh (/m/ngoi-sao-dien-anh/a4c2c341)
Cá da phiến (/m/ca-da-phien/4c075293)

TRƯỚC | TIẾP

NỘI DUNG TƯƠNG TỰ

Bài toán luồng cực đại trong mạng (/m/bai-toan-luong-cuc-dai-trong-mang/4452c6b2)
Các hàm trực tuyến (inline) (/m/cac-ham-truc-tuyen-inline/90ef4eb3)
Lớp và đối tượng trong C# (/m/lop-va-doi-tuong-trong-c/7ae4309c)
Hàm trong C++ (/m/ham-trong-c/a256cd4c)
Nạp chồng toán tử (/m/nap-chong-toan-tu/48a58069)
Mẫu thuẫn vợ chồng trong gia đình và những yếu tố ảnh hưởng (/m/mau-thuan-vo-chong-trong-gia-dinh-va-nhung-yeu-to-anh-huong/2fc1d308)
Kiểu và khai báo biến trong C (/m/kieu-va-khai-bao-bien-trong-c/3c02db4b)
Biểu thức và khoảng trắng (/m/bieu-thuc-va-khoang-trang/53c9b9c8)
Truyền tham số (/m/truyen-tham-so/3624d9fb)
Luồng (Thread) trong Java (/m/luong-thread-trong-java/cc1dbe94)

TRƯỚC | TIẾP