Báo cáo:

Thuật toán tìm đường đi ngắn nhất - Dijkstra

B1: Từ đình gốc, khởi tạo bảng khoảng cách đến các đỉnh, đỉnh chưa xét đặt là 0, các đỉnh còn lại có giá trị chính bằng trọng số.

B2: Từ bảng đã tạo chọn đỉnh a có trọng số bé nhất và các lần sau không xét đỉnh này nữa.

B3: Lần lượt xét các đỉnh kề b với đỉnh a vừa tìm được ở B2, nếu khoảng cách từ đỉnh gốc tới đỉnh b nhỏ hơn khoảng cách hiện tại đang được ghi nhận thì cập nhật lại giá trị và đỉnh kề a vào khoảng cách hiện tại của b.

B4: Sau khi xét tất cả các đỉnh kề b của a, ta được bảng cập nhật mới rồi quay lại B2, thuật toán dừng lại khi đỉnh ở B2 chính là đỉnh đích hoặc khi duyệt hết các đỉnh.

Mã giả:

void LineXMin(start, end, \*\*a, n)

begin

// B1

for i = 1 to n - 1 do

begin

value[i] = a[start][i];

if value[i] != 0 then

dad[i] = 0;

end

value[start] = -1;

while true do

begin

// B2:

min = -1;

index = -1;

for i = 0 to n - 1 do

if value[i] > 0 and (value[i] < min or min == -1) then

begin

min = value[i];

index = i;

end

if index == x or index == -1 then // tìm được đường ngắn nhất hoặc không tìm được

break;

value[index] = -1; // đánh dấu đã duyệt

// B3:

for i = 0 to n - 1 do

if value[i] != -1 and a[index][i] != 0 then

if value[i] > min + a[index][i] or value[i] == 0 then

begin

dad[i] = index;

value[i] = min + a[index][i];

end

end

end

Ở B1 thực hiện vòng lặp n lần => O(n)

Ở B2 thực hiện vòng lặp gán, lệnh rẽ nhánh n lần => O(n)

Ở B3 thực hiện vòng lặp và gán, lệnh rẽ nhanh n lần => O(n)

Ở B4 thực hiện tốt nhất là 1 lần => B2, 3, 4 = O(1)\*[O(n) + O(n)] = O(n)

Xấu nhất là n - 1 đỉnh tức n - 1 lần => B2, 3, 4 = O(n) \*[O(n) + O(n)] = O(n^2)

* Tổng độ phức tạp: O(n) + O(n^2) = O(n^2)

Thuật toán tìm đường đi dài nhất - áp dụng tìm kiếm theo chiều sau DFS.

Ý tưởng: Từ đỉnh gốc tìm tất cả các đường đi từ đỉnh gốc tới đỉnh đích rồi lưu đường đi dài nhất vào 1 mảng sau mỗi lần tìm được 1 đường đi mới.

B1: Lưu các đỉnh đã kiểm tra vào mảng, cập nhật số lượng đỉnh trong mảng và đánh dấu đã kiểm tra

B2: Kiểm tra đỉnh đang xét có phải đỉnh đích không

B3a: Nếu phải tiến hành tính tổng các trọng số và so sánh với độ dài lớn nhất, nếu lớn hơn tiến hành lưu các đỉnh vào 1 mảng, và kết thúc

B3b: Nếu không phải tiến hành duyệt các đỉnh còn lại và gọi đệ quy tại đỉnh mới đó

void LineXMax(start, end, \*\*a, n, dadMax[2][100], \*nDad, &lengthMax, \*check)

begin

// B1

DadMax[1][nDad[1] = start;

nDad[1] = nDad[1] + 1;

check[start] = 1;

// B2

if start == end then

begin

// B3a

sum = 0;

for i = 0 to nDad[1] - 2 do

sum = sum + a[DadMax[1][i]][DadMax[1][i+1]]; // trọng số từ đỉnh cha tới đỉnh con

if lengthMax < sum then

begin

lengthMax = sum;

nDad[0] = nDad[1];

for i = 0 to nDad[1] - 1 do

DadMax[0][i] = DadMax[1][i]

end

return;

end

// B3b

for i = 0 to n - 1 do

if a[start][i] != 0 && check[i] == 0 && check[end] == 0 then

begin

LineXMax(i, end, a, n, DadMax, nDad, lengthMax, check);

check[i] = 0;

nDad[1] = nDad[1] - 1;

end

end

Ở B1: O(1)

Ở B2, B3a: Xấu nhất khi gặp được đường đi dài hơn: O(1)\*[O(n) + O(n)] = O(n)

Tốt nhất: Khi đường đi mới ngắn hơn độ dài nhất: O(1)\*O(n) = O(n)

Ở B3b: Duyệt lần lượt n đỉnh => O(n)

* BigO = O(n)\*O(n) = O(n^2)
* Ảnh có chứa văn bản

  Mô tả được tạo tự động

Tiếng anh:

Report:

The shortest way to find the way - Dijkstra

B1: From the original communal house, initialize the distance to the vertices, the peak is not considered as 0, the remaining vertices have the main value equal to the weight.

B2: From the table, selected vertex A with the smallest weight and the next time no longer considered this peak.

B3: In turn, review the vertices B with vertex A just found at B2, if the distance from the base to the base B is smaller than the current distance is being recorded, then update the value and the vertex A at the distance The present of b.

B4: After reviewing all the vertices B of A, we get a new update table and then return to B2, the algorithm stops when the peak at B2 is the target peak or when the vertex is approved.

Pseudocode:void LineXMin(start, end, \*\*a, n)

begin

// B1

for i = 1 to n - 1 do

begin

value[i] = a[start][i];

if value[i] != 0 then

dad[i] = 0;

end

value[start] = -1;

while true do

begin

// B2:

min = -1;

index = -1;

for i = 0 to n - 1 do

if value[i] > 0 and (value[i] < min or min == -1) then

begin

min = value[i];

index = i;

end

if index == x or index == -1 then // Find the shortest path or not find it

break;

value[index] = -1; // Approved

// B3:

for i = 0 to n - 1 do

if value[i] != -1 and a[index][i] != 0 then

if value[i] > min + a[index][i] or value[i] == 0 then

begin

dad[i] = index;

value[i] = min + a[index][i];

end

end

end

In B1 performing n times => O(n)

In B2, the assignment loop, branching command n times => O(n)

In B3, the assignment loop, branching command n times => O(n)

In B4 the best performance is 1 time => B2, 3, 4 = O(1)\*[O(n) + O(n)] = O(n)

The worst is n - 1 vertex n - 1 time => B2, 3, 4 = O(n) \*[O(n) + O(n)] = O(n^2)

* BigO: O(n) + O(n^2) = O(n^2)

The longest wayfinding algorithm - applies dimensional search after DFS.

Idea: From the root peak find all the paths from the root peak to the destination peak and then save the longest path into 1 array after each new path is found.

Step 1: Save the checked vertices to the array, update the number of vertices in the array, and mark checked

Step 2: Check if the peak in question is the target peak

Step3a: If you have to sum the weights and compare them to the greatest length, if you are larger, save the vertices to 1 array, and finish

Step3b: If you don't have to go through the remaining vertices and call recursively at that new vertex

void LineXMax(start, end, \*\*a, n, dadMax[2][100], \*nDad, &lengthMax, \*check)

begin

// B1

DadMax[1][nDad[1] = start;

nDad[1] = nDad[1] + 1;

check[start] = 1;

// B2

if start == end then

begin

// B3a

sum = 0;

for i = 0 to nDad[1] - 2 do

sum = sum + a[DadMax[1][i]][DadMax[1][i+1]]; // weighting from parent vertex to child vertex

if lengthMax < sum then

begin

lengthMax = sum;

nDad[0] = nDad[1];

for i = 0 to nDad[1] - 1 do

DadMax[0][i] = DadMax[1][i]

end

return;

end

// B3b

for i = 0 to n - 1 do

if a[start][i] != 0 && check[i] == 0 && check[end] == 0 then

begin

LineXMax(i, end, a, n, DadMax, nDad, lengthMax, check);

check[i] = 0;

nDad[1] = nDad[1] - 1;

end

end

Ở B1: O(1)

Ở B2, B3a: Worst when encountering a longer route: O(1)\*[O(n) + O(n)] = O(n)

Best: When the new path is shorter than the longest: O(1)\*O(n) = O(n)

Ở B3b: Browse in turn n vertices => O(n)

* BigO = O(n)\*O(n) = O(n^2)

Ảnh có chứa văn bản

Mô tả được tạo tự động