SOLUTION – BUỔI 5

1. CLIMB

Bài toán này có một giải pháp tham lam, trong đó những con bò đi theo cùng một thứ tự lên núi rồi quay xuống: chúng ta bắt đầu với tất cả những con bò có U < D, được sắp xếp theo thứ tự tăng dần của U, tiếp theo là tất cả những con bò có U >= D, được sắp xếp theo thứ tự giảm dần của D. Để thực hiện thuật toán này, tất cả những gì chúng ta cần làm về cơ bản là sắp xếp, vì vậy việc mã hóa và phân tích thời gian chạy khá đơn giản. Khía cạnh thách thức của giải pháp này, giống như hầu hết các thuật toán tham lam, là thuyết phục bản thân rằng phương pháp tham lam đặc biệt này trên thực tế luôn luôn tối ưu.

Đầu tiên, chúng ta hãy lập luận rằng một giải pháp tối ưu luôn có thể sử dụng cùng một thứ tự sắp xếp các con bò trên đường xuống núi cũng như trên đường lên. Để thấy điều này, giả sử rằng có một giải pháp tối ưu nào đó trong đó thứ tự tăng và giảm là khác nhau. Đặt 1, 2, 3, ..., n biểu thị thứ tự đi lên và giả sử vị trí đầu tiên khác nhau giữa hai thứ tự là tại chỉ số x, trong đó con bò y > x xuất hiện theo thứ tự đi xuống. Bằng cách di chuyển con bò x sớm hơn theo thứ tự đi xuống để nó ngay trước con bò y, người ta có thể thấy rằng điều này chỉ có thể cải thiện lịch trình tổng thể; nó không bao giờ có thể làm cho lịch trình leo núi trở nên tồi tệ hơn (cùng với phần còn lại của các lập luận mà chúng tôi đưa ra, điều này khá rõ ràng nếu bạn vẽ những con bò được đề cập trên một dòng thời gian). Những bước di chuyển lặp đi lặp lại kiểu này có thể biến lịch trình tối ưu thành một lịch trình tốt như nhau trong đó thứ tự tăng và giảm là như nhau. Do đó, luôn tồn tại một lịch trình tối ưu với thứ tự tăng và giảm giống hệt nhau.

Tiếp theo, chúng ta thực hiện một loạt các đối số "trao đổi" khác. Giả sử rằng một giải pháp tối ưu bao gồm một thứ tự trong đó có một cặp bò liên tiếp (x,y) mà U(x) >= D(x) nhưng U(y) < D(y). Trong trường hợp này, hoán đổi thứ tự của x và y chỉ có thể giúp giải quyết tổng thể. Việc hoán đổi lặp đi lặp lại kiểu này cho chúng ta thấy rằng luôn có một giải pháp tối ưu trong đó những con bò có U < D đứng trước những con có U >= D. Cuối cùng, trong số tất cả những con bò có U < D, việc hoán đổi hai con bò liền kề (x, y) có U(x) > U(y) chỉ có thể giúp ích cho giải pháp. Tương tự như vậy, giữa các con bò theo thứ tự với U >= D, việc hoán đổi hai con bò liền kề (x,y) với D(x) < D(y) chỉ có thể hữu ích. Điều này cho chúng ta biết rằng luôn có một giải pháp tối ưu có dạng do thuật toán tham lam của chúng ta tạo ra.

#include <bits/stdc++.h>

using namespace std;

int N, end1[25000], end2[25000];

pair<int, int> cows[25000];

bool comp(pair<int, int> a, pair<int, int> b)

{

if(a.first < a.second)

{

if(b.first < b.second)

return a.first < b.first;

else

return true;

}

else

{

if(b.first > b.second)

return a.second > b.second;

else

return false;

}

}

int main()

{

FILE \* w = fopen("climb.inp", "r");

FILE \* o = fopen("climb.out", "w");

fscanf(w, "%d", &N);

for(int i = 0; i < N; i++)

fscanf(w, "%d %d", &cows[i].first, &cows[i].second);

sort(cows, cows + N, comp);

for(int i = 0; i < N; i++)

end1[i] = (i > 0 ? end1[i - 1] : 0) + cows[i].first;

for(int i = 0; i < N; i++)

end2[i] = max((i > 0 ? end2[i - 1] : 0), end1[i]) + cows[i].second;

fprintf(o, "%d\n", end2[N - 1]);

printf("%d\n", end2[N - 1]);

return 0;

}

1. ROUTE

Chúng ta có thể tạo biểu đồ biểu diễn các trang trại bằng cách tạo 5 nút cho mỗi trang trại: một cho chính trang trại và một cho mỗi tọa độ (x, y) liền kề. Đặt đường góc vuông là đường thay đổi hướng nhiều nhất một lần và chỉ bao gồm chuyển động lên, xuống, trái hoặc phải. Chúng tôi thêm một cạnh giữa mỗi cặp nút (x1, y1) và (x2, y2) nếu có thể đi từ (x1, y1) đến (x2, y2) bằng đường dẫn góc vuông không cắt bất kỳ trang trại nào, ngoại trừ tại các điểm cuối của nó. Độ dài của cạnh này là khoảng cách "Manhattan" (|x1-x2|+|y1-y2|).

Để kiểm tra xem một cạnh có hợp lệ hay không, ta kiểm tra đường đi (x1, y1) -> (x1, y2) -> (x2, y2) và đường đi (x1, y1) -> (x2, y1) -> (x2, y2). Chúng ta có thể lặp qua tất cả N trang trại và xem liệu một trong hai đường góc vuông có cắt bất kỳ trang trại nào không. Có nhiều cách hiệu quả hơn để kiểm tra, nhưng vì N <= 100 nên thuật toán này đủ nhanh.

Sử dụng biểu đồ mới này, chứa tối đa 5N nút và (5N)^2 cạnh, chúng ta có thể sử dụng thuật toán đường đi ngắn nhất để tìm độ dài đường đi ngắn nhất giữa mỗi nhóm. Giải pháp này hoạt động vì có thể đạt được mỗi đường dẫn ngắn nhất giữa các trang trại chỉ bằng cách sử dụng các nút 5N và đường dẫn góc vuông. Với thuật toán Dijkstra, chúng ta có thể tìm mỗi đường đi ngắn nhất trong O(N^2). Bằng cách tìm các đường đi ngắn nhất cho từng trang trại trong số N trang trại, toàn bộ vấn đề có thể được giải quyết trong O(N^3).

#include <iostream>

#include <cstdio>

#include <cstdlib>

#include <vector>

#include <set>

#include <queue>

using namespace std;

struct pt

{

int x, y;

pt(int a, int b)

{ x = a, y = b; }

} ;

int N, x[100], y[100], indx[100];

set<pair<int, int> > points; // quản lí tọa độ trang trại

vector<pt> nodes;

vector<int> adj[500]; // danh sách kề

// trả về khoảng cách nodes[a] and nodes[b].

int length(int a, int b)

{

return abs(nodes[a].x - nodes[b].x) + abs(nodes[a].y - nodes[b].y);

}

// trả về điểm đàu tiên (x1, y1) có nằm trên đoạn (x2, y2) -> (x3, y3) hay không.

bool in\_segment(int x1, int y1, int x2, int y2, int x3, int y3)

{

if(x2 == x3)

return x1 == x2 && y1 > min(y2, y3) && y1 < max(y2, y3);

else if(y2 == y3)

return y1 == y2 && x1 > min(x2, x3) && x1 < max(x2, x3);

else // invalid

return true;

}

//Returns whether a right-angle path from nodes[a] to nodes[b] is possible (does not intersect any of the N original points, except potentially at endpoints).

bool possible(int a, int b)

{

// Method 1: travel vertical first, then horizontal

bool good1 = nodes[a].x == nodes[b].x || nodes[a].y == nodes[b].y || points.find(make\_pair(nodes[a].x, nodes[b].y)) == points.end();

for(int i = 0; i < N; i++)

if(in\_segment(x[i], y[i], nodes[a].x, nodes[a].y, nodes[a].x, nodes[b].y) || in\_segment(x[i], y[i], nodes[a].x, nodes[b].y, nodes[b].x, nodes[b].y))

{

good1 = false;

break;

}

if(good1)

return true;

//Method 2: travel horizontal first, then vertical

bool good2 = nodes[a].x == nodes[b].x || nodes[a].y == nodes[b].y || points.find(make\_pair(nodes[b].x, nodes[a].y)) == points.end();

for(int i = 0; i < N; i++)

if(in\_segment(x[i], y[i], nodes[a].x, nodes[a].y, nodes[b].x, nodes[a].y) || in\_segment(x[i], y[i], nodes[b].x, nodes[a].y, nodes[b].x, nodes[b].y))

{

good2 = false;

break;

}

if(good2)

return true;

return false;

}

// trả về đường đi ngắn nhất từ nodes[a] tới nodes[b].

bool vis[500];

int dist[500], infinity = 1023456789;

int dijkstra(int a, int b)

{

for(int i = 0; i < nodes.size(); i++)

{

dist[i] = infinity;

vis[i] = false;

}

// Don't visit farms (except for the start and end locations).

for(int i = 0; i < N; i++)

if(indx[i] != a && indx[i] != b)

vis[indx[i]] = true;

dist[a] = 0;

for(int i = 0; i < nodes.size(); i++)

{

int next = 0;

for(int j = 0; j < nodes.size(); j++)

if(!vis[j] && (dist[j] < dist[next] || vis[next]))

next = j;

if(vis[next] || dist[next] == infinity)

return -1;

if(next == b)

return dist[next];

vis[next] = true;

for(int j = 0; j < adj[next].size(); j++)

if(!vis[adj[next][j]])

dist[adj[next][j]] = min(dist[adj[next][j]], dist[next] + length(next, adj[next][j]));

}

return -1;

}

int main()

{

FILE \* w = fopen("route.inp", "r");

FILE \* o = fopen("route.out", "w");

fscanf(w, "%d", &N);

for(int i = 0; i < N; i++)

{

fscanf(w, "%d %d", &x[i], &y[i]);

points.insert(make\_pair(x[i], y[i]));

}

// Make nodes

for(int i = 0; i < N; i++)

for(int a = -1; a <= 1; a++)

for(int b = -1; b <= 1; b++)

{

if(a == 0 && b == 0)

{

nodes.push\_back(pt(x[i], y[i]));

indx[i] = nodes.size() - 1;

}

else if(a \* b == 0 && points.find(make\_pair(x[i] + a, y[i] + b)) == points.end())

nodes.push\_back(pt(x[i] + a, y[i] + b));

}

// Make edges

for(int i = 0; i < nodes.size(); i++)

for(int j = i + 1; j < nodes.size(); j++)

if(possible(i, j))

{

adj[i].push\_back(j);

adj[j].push\_back(i);

}

// Dijkstra's Algorithm

int answer = 0;

for(int i = 0; i < N; i++)

{

int next = dijkstra(indx[i], indx[(i + 1) % N]);

if(next < 0)

{

answer = -1;

break;

}

else

answer += next;

}

fprintf(o, "%d\n", answer);

printf("%d\n", answer);

return 0;

}

3. SHARE

Giải quyết bằng quy hoạch động.

Đặt good[n][B\_2][B\_3] là "true" nếu có thể đạt đến trạng thái chỉ sử dụng kiện 1..n trong đó B\_2 và B\_3 biểu thị tổng lượng cỏ khô trong chuồng 2 và 3 (giống như trong lời giải bài toán). Lưu ý rằng chúng ta không cần đưa B\_1 vào không gian trạng thái của mình, vì B\_1 được xác định bằng cách biết n, B\_2 và B\_3: B\_1 là tổng kích thước của kiện 1..n trừ B\_2 trừ B\_3.

Bây giờ, ta điền vào bảng tất cả các giá trị good[][][] và giải pháp thu được bằng cách xem good[N][][] cho mục nhập "true" có giá trị nhỏ nhất là max(B\_1,B\_2,B\_3). Để làm cho thuật toán sử dụng bộ nhớ hiệu quả nhất có thể, cần lưu ý rằng ta chỉ cần lưu trữ hai "cấp độ" của bảng (cho n và n-1) tại một thời điểm. Hơn nữa, ta biết rằng giải pháp tối ưu nhiều nhất là tổng của tất cả các kích cỡ kiện chia cho 3,

#include <iostream>

#include <fstream>

using namespace std;

const int MAXS = 700;

ifstream fin ("share.in");

ofstream fout ("share.out");

int n, bale, tsum;

bool good[2][MAXS+100][MAXS+100];

int main()

{

for (int i = 0; i < 2; i++)

for (int j = 0; j < MAXS; j++)

for (int k = 0; k < MAXS; k++)

good[i][j][k] = false;

good[0][0][0] = true;

tsum = 0;

fin >> n;

for (int i = 0; i < n; i++)

{

fin >> bale;

tsum += bale;

for (int j = 0; j < MAXS; j++)

for (int k = 0; k < MAXS; k++)

{

if (good[i%2][j][k])

{

good[(i+1)%2][j][k] = true;

good[(i+1)%2][j+bale][k] = true;

good[(i+1)%2][j][k+bale] = true;

}

}

}

int ans = MAXS;

for (int i = 0; i < MAXS; i++)

for (int j = 0; j < MAXS; j++)

if (good[n%2][i][j])

ans = min (ans, max (i, max (j, tsum - (i + j))));

fout << ans << "\n";

return 0;

}