Asterix and Obelix

Link submit:
https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1187

Solution:

C++	https://ideone.com/n87TyE
Java	https://ideone.com/F3SYGM
Python	https://ideone.com/WiZpnw

Tóm tắt đề:

Asterix và bạn ông ấy Obelix đang trở về nhà sau chiến thắng người Romans. Có C thành phố và R con đường nối giữa các thành phố đó. Mỗi con đường có một chi phí cần bỏ ra để đi qua. Asterix quyết định sẽ tổ chức một buổi tiệc tại một trong những thành phố trên đường về nhà. Nếu tổ chức buổi tiệc tại thành phố i thì sẽ tốn chi phí là $cost_i$. Để thể hiện tình cảm của mình dành cho Obelix, anh ta sẽ tổ chức bữa tiệc tại thành phố đắt nhất trên con đường họ đi qua.

Tính tổng chi phí nhỏ nhất mà Asterix bỏ ra là bao nhiêu, bao gồm chi phí di chuyển qua các con đường và chi phí tổ chức tiệc.

Input:

Dữ liệu có nhiều test case. Mỗi test case có cấu trúc như sau:

- Dòng đầu tiên gồm ba số nguyên C ($C \le 80$), R ($R \le 1000$), Q ($Q \le 6320$) với C là số thành phố (các thành phố được đánh dấu từ 1 đến C), R là số con đường, Q là số truy vấn.
- Dòng thứ hai chứa C số nguyên với f_i là chi phí tổ chức tiệc tại thành phố i.
- R dòng kế tiếp, mỗi dòng gồm ba số nguyên c_1, c_2, d thể hiện đường đi nối giữa hai thành phố c_1 và c_2 với chi phí là d.
- Q dòng cuối, mỗi dòng gồm hai số nguyên c_1, c_2 hỏi chi phí nhỏ nhất để đi từ c_1 đến c_2.

Dữ liệu kết thúc bằng một dòng chứa ba số 0 (ứng với dòng C R Q).

Output:

Mỗi test case, dòng đầu tiên in "Case #cs:" với cs là số thứ tự của test case (tính từ 1). Sau đó với mỗi truy vấn in trên 1 dòng, dòng thứ i là chi phí nhỏ nhất bỏ ra để đi từ c_1 đến c_2 của truy vấn thứ i. Nếu không có đường đi thì in ra -1. In một dòng trắng giữa 2 test case liên tiếp.

Ví dụ:

7 8 5 2 3 5 15 4 4 6 1 2 20 1 4 20 1 5 50 2 3 10 3 4 10 3 5 10 4 5 15 6 7 10 1 5 1 6 5 1 3 1 6 7 4 4 2 2 1 8 3 1 2 7 1 3 5 2 4 8 3 4 6 1 4 2 3 0 0 0	Case #1 45 -1 45 35 16 Case #2 18 20

Giải thích ví dụ:

Ở test case thứ nhất:

Đường đi từ 1 đến 5: đi đường 1 – 2 – 3 – 5, với chi phí các đường đi qua là 20 + 10 + 10 = 40, thành phố 3 có chi phí tổ chức sự kiện lớn nhất là 5 nên kết quả là 45. Ta không đi đường 1 – 4 – 5 vì mặc dù tổng chi phí đi qua chỉ là 20 + 15 = 35 nhưng chi phí tổ chức sự kiện tại thành phố 4 là 15 nên tổng chi phí là 50.

Đường đi từ 1 đến 6: không thể đi từ 1 đến 6 được nên kết quả là -1.

Đường đi từ 5 đến 1: tương tự truy vấn 1.

Đường đi từ 3 đến 1: Đi theo đường 1 – 2 – 3 với tổng chi phí là 20 + 10 + 5 = 35.

Đường đi từ 6 đến 7: chỉ có 1 cách đi duy nhất là 6 – 7 với chi phí là 16.

Hướng dẫn giải:

Ở bài này thì có thể giải bằng cả 3 thuật toán tìm đường đã học là Dijkstra, Bellman Ford và Floyd Warshall. Tuy nhiên với mỗi test case (1 đồ thị) thì lại có nhiều truy vấn, nếu sử dụng

Dijkstra hay Bellman thì phải chạy lại thuật toán với mỗi truy vấn. Như vậy thì FloydWarshall sẽ là thuật toán lựa chọn tối ưu.

Ta thấy rằng chi phí để đi từ u đến v lúc này không chỉ là tổng của các cạnh trên đường đi đã chọn, mà còn phụ thuộc vào chi phí của đỉnh lớn nhất trong đường đi. Như vậy, ngoài ma trận dist với dist[i][j] là chi phí nhỏ nhất để đi từ i đến j thì ta cần thêm 1 ma trận khác khác là maxCost với maxCost[i][j] là chi phí của đỉnh lớn nhất trong đường đi nhỏ nhất từ i đến j hiện tại. Ta thấy rằng nếu mình có 2 đường đi từ i đến k và từ k đến j thì dist[i][j] có thể tính theo công thức dist[i][k] + dist[k][j] – min(maxCost[i][k], maxCost[k][j]). Sở dĩ ở đây có trừ cho min của 2 maxCost là vì trong 2 đoạn đường từ i đến k và từ k đến j mình đã tính cả 2 đỉnh lớn nhất trong 2 đoạn, nhưng khi gộp lại thì mình chỉ giữ 1 đỉnh lớn hơn, vì vậy phải trừ đi chi phí của đỉnh nhỏ ra. Đồng thời sau khi tính xong thì mình cũng cập nhật lại maxCost của i và j.

Tuy nhiên, ở bài này thì cần lưu ý rằng ta có 2 đại lượng là dist và maxCost có thể ảnh hưởng qua lại lẫn nhau (dist tính theo maxCost, còn maxCost thì chỉ được cập nhật khi dist thay đổi). Nên để đảm bảo rằng cả 2 đại lượng đều tối ưu, ta phải duyệt Floyd 2 lần.

Sau khi có ma trận kết quả thì với mỗi truy vấn ta chỉ việc truy xuất giá trị dist[c1][c2] để lấy kết quả là được.

Độ phức tạp: $O(T * C^3)$ với T là số lượng bộ test, C là số thành phố trong mỗi test case.