Min Cost Flow Contest brief editorial

Maxim Akhmedov
Yandex

September 24th, 2019

1 Problem A. Max Flow Min Cost

Just implement iterative min cost flow approach using Ford-Bellman. Yeah, the constraints seem to be high for this approach to work, but it is a flow problem, so everything is possible (and might work a lot faster than you expect).

2 Problem B. Assignments

Introduce standard two-layer network for representing matchings and find minimum cost n-flow.

3 Problem C. Beer

Introduce a network that contains all edges of the graph with their capacities and costs. Make vertex 1 to be source. Add an auxillary vertex for sink, to which all vertices (except source) will be connected with infinite capacity edges of cost $-profit[i]$, where $profit[i]$ is the profit from selling one barrel of beer in city i. Find minimum cost flow mcf, and the answer will be $-mcf$.

4 Problem D. Domino in Casino

Note that rectangular field domino coverings may be treated as matchings in bipartite graph: consider the table chess coloring, now any possible domino is an edge between some two vertices of different parts.

Now use the similar approach as in problem B.

5 Problem E. Automata Programming

Let's use the power of imagination. Imagine a timeline, and consider all tasks to be segments on that line. Now each executor may be seen as somebody, who walks from left to right across that timeline, visiting some of the task segments. Let us reduce a problem to a maximum flow k-cost problem.

Consider a graph with all interesting time moments (segment endoints) as vertices. Each task may be implemented as an edge of capacity 1 and cost c_i. In order to allow moving from one task to another, we have to put edges from any right endpoint to any later left endpoint. In worst case it results in $O(n^2)$ edges, which is too much, so we may introduce special auxillary

vertices, which will work as "suffix sums": from $suff_i$ there will be an infinite capacity zero cost edge to $suff_{i+1}$ and to i-th left endpoint (in sort order). Such transformation will reduce size of the graph to linear, which is OK for running k Ford-Bellmans.

6 Problem F. Inspection

There are several approaches for this problem, the easiest one is to "invert" min cost k-flow algorithm in some sense. Namely, we need to make it profitable to pass through an unvisited ski resort. Represent each ski resort as a pair of edges with first having unit capacity and unit cost, and second having infinite capacity and zero cost. Make all the remaining edges of the graph be of zero cost and infinite capacity. Now we have to find minimum k such that the maximum cost of the k-flow is at least (in fact, exactly) n. Increase k until the required condition is held.

7 Problem G. Binary Tree on Plane

Let us formulate some conditions on vertices and edges of the desired graph:

- Each vertex has in-degree of at most 1;
- Each vertex has out-degree of at most 2;
- Each edge goes only in descending direction of y-coordinate;
- There are exactly n edges;
- Total length of the edges should be minimum possible.

We will treat the fourth condition as the condition for the value of the flow, i.e. we will seek some flow of value k. Total length of the edges will be the cost of the flow. For each point, introduce two vertices: in_i and out_i. Draw edges of capacity 1 and cost $dist(i, j)$ from in_i to out_j (if it goes downwards), also draw a zero cost edge from s to in_i of capacity 2 from out_i to t of capacity 1. If there is an $(n-1)$-flow, the minimum cost such flow will be an answer.

8 Problem H. Aarelia Mountains

Consider adjacent differences in our array. Note that "range increase/descrease" procedure affects them as changing two integers by the same value x simultaneously: one is increased by x and one is decreased by x. This is pretty similar to what happens with vertex divegences when we increase flow value through some edge.

We won if all adjacent differences are positive. Consider the total absolute value of negative elements in adjacent difference sequence. We need to find flow of at least such value from negative values to positive of minimum possible cost.

9 Problem I. Equipment Assembling

Consider the following graph: $n+1$ vertices, for each i there is an edge e_i from i to $i+1$, and also for each Zealot kind (i, j) there is an edge from $j+1$ to i of cost c_{ij}. We want to find a minimum cost circulation (zero value flow), which satisfies the property that value of the flow

through each e_i is at least b_i. To make that happen, split e_i into two edges, one of which is very profitable (i.e. has cost $-inf$ where inf is large enough integer) of capacity b_i, and second one has infinite capacity and cost 0.

In such case minimum cost flow has to satisfy all $-inf$'s, and among all solutions with the desired value of $-inf$'s taken, it will choose the minimum cost assignment.

10 Problem J. Longest Shortest Path

This problem requires understanding of Linear Programming Duality.

Let's solve the following problem: given the distance d, find the minimum cost of making distance from s to t be at least d. The original problem may be reduced to this using the binary search.

The original problem may be formulated as follows: choose non-negative x_e for each edge e such that for any path P total x_e over all $e \in P$ is at least $def(P) = \max\{0, d - len(P)\}$. Under such constraints, minimize sum of $x_e \cdot c_e$. Note that there is an exponential number of constraints in this problem.

This problem is dual to choosing non-negative y_P for each path P in graph, such that for each edge e total y_P over all P containing e is at most c_e. Under such constratints, maximize sum of $y_P \cdot def(P)$.

Note that dual conditions define a (decomposed) $s-t$ flow in graph with capacities c_e, where y_P defines a flow through path P. Sum of $y_p \cdot def(P)$ may be rewritten as $d \cdot val(f) - cost(f)$ if we consider the cost of edge e to be l_e. So, we have to find the flow that maximizes $d \cdot val(f) - cost(f)$. This may be done by considering the fact that $cost(val)$ is a convex function.