



My TopCoder

Competitions

[Overview](#)[Copilot Opportunities](#)[Design](#)[Development](#)[UI Development](#)[QA and Maintenance](#)[CloudSpokes](#)[Algorithm](#)[High School](#)[The Digital Run](#)[Submit & Review](#)

TopCoder Networks

[Events](#)[Statistics](#)[Tutorials](#)[Forums](#)[Surveys](#)[My TopCoder](#)[Help Center](#)[About TopCoder](#)

Member Search:

Handle: [Go](#)[Advanced Search](#)
[Dashboard](#) > [TopCoder Competitions](#) > ... > [Algorithm](#)
[Problem Set Analysis](#) > [SRM 626](#)
 [Search](#)

TopCoder Competitions



SRM 626

[View](#)[Attachments \(10\)](#)[Info](#)[Browse Space](#)Added by [\[\[rng_58\]\]](#), last edited by [vexorian](#) on Jul 05, 2014 ([view change](#))Labels: (None) [EDIT](#)

Single Round Match 626

Saturday, June 28th, 2014

[Archive](#)[Match Overview](#)[Discuss this match](#)

Match summary

The Problems

[SumOfPower](#) | [FixedDiceGameDiv2](#) | [NegativeGraphDiv2](#) | [FixedDiceGameDiv1](#) | [NegativeGraphDiv1](#) | [ReflectiveRectangle](#)

NegativeGraphDiv2

[Rate It](#)[Discuss it](#)

Used as: Division Two - Level Three:

Value	1000
Submission Rate	117 / 1096 (10.68%)
Success Rate	14 / 117 (11.97%)
High Score	postan for 638.17 points (24 mins 32 secs)
Average Score	514.48 (for 14 correct submissions)

A new graph

There is a graph in this problem, but it's minimum path has strange properties. You can sometimes multiply an edge weight by -1 and there is a limit.

The idea is to make a new special graph. Imagine you cared about the state of the game. Every time you use a negative-weight edge, the number of extra times you can use that trick decreases, so this number of times you can do the trick should be part of the state. Another part of the state should be the current vertex (in the original graph) you are in. So we will describe a vertex in the new graph by pair (k, u) where k is the remaining number of charges and u the current vertex.

Starting at each (k, u) , there will be some edges that connect (k, u) with other vertices (k, v) (same number of charges). Using any edge without multiplying its weight by -1. Using a negative edge, however, would decrement k by 1 (and you cannot do it if $k = 0$). There will be negative edges between (k, u) and $(k - 1, v)$ for every edge between u and v in the original graph.

This logic creates a new graph and the shortest path between vertex (charges, 1) and any node (k, N) will be equal to the shortest path between 1 and N using at most "charges" negative edges.

Having to deal with the "at most" aspect complicates our logic a bit, because there are up to charges different goal vertices. We can make this all easier by adding a 0-cost "negative" edge between (k, u) and $(k - 1, u)$, so now the shortest path between (charges, 1) and $(0, N)$ is all we need to solve the problem.

Shortest path with negative edges

We can now try to solve the shortest path. The graph contains negative edges, so we cannot use a simple Dijkstra algorithm. Usually, the idea is to use [Bellman-Ford](#). But the total number of vertices and edges in the new graph can be quite taxing. While it is possible to optimize Bellman-Ford in the low level to pass system tests in this problem and many did during the match, it turns out that is because of lack of specific system tests against Bellman-Ford. So we should look for something faster.

Specific properties

The graph we generated between vertices (k, u) is not a general graph. It has very specific properties. Imagine that for each k , all vertices (k, u) make a sub-graph that is identical to the original graph. We can call this sub-graph a level, "level k ". Vertices in level k are connected to vertices in level $k - 1$ through negative edges (and some 0-weight edges connecting (k, u) and $(k - 1, u)$).

The idea is that we can know the minimum cost of moving between a (k, u) and (k, v) by just solving the costs in the *original graph* (without using negative edges). We can find the minimum distances for each pair (u, v) using Dijkstra or [Floyd-Warshall](#).

The levels aspect is acyclic. Once you go down a level, you cannot return back to the previous level. This acyclic property implies we can build the minimum distance from each (k, u) to $(0, N)$ in decreasing order of k .

A base case: It is easy to find all minimum distances from $(0, u)$ to $(0, N)$: Just use the distances in the original graph (Since $k = 0$, We cannot use negative edges anymore).

The question is: Assume that we know all the minimum distances from each $(k - 1, u)$ to $(0, N)$, how can we use this information to build the minimum distances from (k, u) to $(0, N)$? We start at vertex (k, u) and we should first move to some (k, v) then we use a negative edge from v to a z . Here we should consider all possible edges $(v \rightarrow z)$ with weight w in the original graph. There is a cost to move from (k, u) to (k, v) then use the $-w$ edge to move from (k, v) to $(k - 1, z)$ and finally move from $(k - 1, z)$ to $(0, N)$ using the *already-known* minimum distance to $(0, N)$. We can repeat for each vertex u and all the edges.

A complexity analysis: We need to repeat the process for $O(\text{charges})$ levels. For each $O(N)$ vertex (k, u) we try all $O(M)$ edges in the original graph. That is $O(\text{charges}NM)$. We also need to precalculate the distances in the original graph, that would be $O(N^3)$ if we use Floyd-Warshall. In total this approach is: $O(\text{charges}NM + N^3)$

Code

```

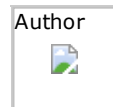
// vertices (without using negative edges)
int dist[N][N];
for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
        dist[i][j] = INF;
    }
    dist[i][i] = 0; //important because we'll need correct distance
                    // between x and x in a later step
}
for (int i = 0; i < weight.size(); i++) { //minimum edge for each pair
    int u = s[i] - 1, v = t[i] - 1, w = weight[i];
    dist[u][v] = std::min(dist[u][v], w);
}
// Floyd:
for (int k = 0; k < N; k++) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            dist[i][j] = std::min( dist[i][j], dist[i][k] + dist[k][j]);
        }
    }
}
// xdist[k][i] : Distance from vertex i to N-1 using at most k neg edges
int xdist[charges+1][N];
for (int i = 0; i < N; i++) {
    xdist[0][i] = dist[i][N-1];
    //with no negative edges, use the Floyd results
}
for (int k = 1; k <= charges; k++) { // O(charges)
    for (int i = 0; i < N; i++) { // O(charges * N )
        xdist[k][i] = xdist[k-1][i]; // "at most" k charges
        // (*This is equivalent to a 0-weight "negative" loop i->i)
    }
}
for (int j = 0; j < weight.size(); j++) { // O(charges * N * M )

```

Alternative solutions and additional comments.

<Place your comments here>

Next problem: [FixedDiceGameDiv1](#)



By **vexorian**

TopCoder Member

Editorial feedback	Choose
I liked it.	<input checked="" type="checkbox"/>
I didn't like it.	<input checked="" type="checkbox"/>

Comments ([Hide Comments](#))


As it's in the Wiki, there's a possibility to improve it. It can be language correction, wording improvement or additional explanation in some parts, your additional comments, description of alternative solutions, etc. If you want to improve the wording of editorial writer or correct some language error, please feel free to put your change over the original text. And if you wish to add a comment or describe another approach, there's a section for this at the bottom of each problem.

Before editing, please be sure to check the [guidelines](#).

You can also add a comment in this comment section. When posting a comment thread, make sure to specify the problem you are talking about.

The editorial uses [mathjax](#) to render formulas and math symbols, if you have issues with the

formatting of math contents or they don't show up, try right clicking the space that contains them and change the renderer mode in the popup menu.


 Posted by vexorian at [Jul 02, 2014 11:48](#) | [Reply To This](#)

As it's in the Wiki, there's a possibility to improve it. It can be language correction, wording improvement or additional explanation in some parts, your additional comments, description of alternative solutions, etc. If you want to improve the wording of editorial writer or correct some language error, please feel free to put your change over the original text. And if you wish to add a comment or describe another approach, there's a section for this at the bottom of each problem.

Before editing, please be sure to check the [guidelines](#).

You can also add a comment in this comment section. When posting a comment thread, make sure to specify the problem you are talking about.

The editorial uses [mathjax](#) to render formulas and math symbols, if you have issues with the formatting of math contents or they don't show up, try right clicking the space that contains them and change the renderer mode in the popup menu.


 Posted by vexorian at [Jul 02, 2014 11:48](#) | [Reply To This](#)

n


 Posted by flyman3046 at Jul 02, 2014 20:49 Updated by flyman3046 | [Reply To This](#)

Complexity of Div 2 1000 problem theoretically is still pretty high. $charges * N * M > 1e8$. How can we be sure that it won't time out?

I mostly see that if the number of operations $\leq 1e8$ then it would not time out.

 Posted by nilmish.iit at [Jul 02, 2014 23:08](#) | [Reply To This](#)

In topcoder $\leq 1e9$ tends to be fine actually, specially in case of this problem with simple operations. C++ takes 0.5 seconds in worst case and Java is usually 2x slower so it should be okay too.

 Posted by vexorian at [Jul 03, 2014 00:43](#) | [Reply To This](#)

About FixedDiceGameDiv1, another idea to calculate number of every possible score and then calculate probability finally. The codes like this:

```
int flag_a = 0;
for (int i=1; i<=b; i++) an[flag_a]i = 1;
for (int i=1; i<a; i++) {
for (int j=1; j<=b; j++)
```

Unknown macro: { for (int k=0; k<2501; k++) an[1-flag_a][j+k] += an[flag_a][k]; }

```
flag_a = 1 - flag_a;
for (int j=0; j<2501; j++) an[ 1-flag_a ][j] = 0;
}
```


```
double total_case_a = 0;
for (int i=0; i<2501; i++)
```

Unknown macro: { total_case_a += an[flag_a][i]; }

```
for (int i=0; i<2501; i++)
```

Unknown macro: { an[flag_a][i] = an[flag_a][i] / total_case_a; }

The result can be quite huge and seems only double can store it. The answer is not same with the standard solution when the parameters are big. It should due to accuracy lost of double type. My question is how can we be sure there is no accuracy lost in the standard solution?

 Posted by houxiang at [Jul 03, 2014 02:48](#) | [Reply To This](#)

In the problem ReflectiveRectangle I didn't get the line "We can find the point with x1=1 that has b bounces: y1=1+b". What are x1 and y1 here?

I get a feeling I'm missing something trivial, if it is so, please forgive me.

 Posted by adityavikram at [Jul 03, 2014 06:44](#) | [Reply To This](#)

They are just the names of the point that has b bounces and its x coordinate is 1. I should have probably added numbers to the grids so that it is easy to see that the grid makes a coordinate system though.

 Posted by vexorian at [Jul 03, 2014 10:42](#) | [Reply To This](#)

In the editorial of NegativeGraphDiv2 I haven't got this portion...

```
for (int k = 1; k <= charges; k++) { // O(charges)
for (int i = 0; i < N; i++) { // O(charges * N )
xdist[k][i] = xdist[k-1][i]; //"at most" k charges
// (*This is equivalent to a 0-weight "negative" loop i->i)
```

How is distance from node 'i' to node 'N-1' using at most K charges equivalent to distance from node 'i' to node 'N-1' using at most K-1 charges ??? Would anybody give me the answer please ???


 Posted by mukitmks25 at [Jul 05, 2014 10:52](#) | [Reply To This](#)

It isn't, but the distance using at most K charges shall be at most equal to the distance with K-1 charges. Because if there is a path using K-1 charges and it is very small, you can still use that same path when the condition is "at most K".


 Posted by vexorian at [Jul 05, 2014 11:58](#) | [Reply To This](#)

In NegativeGraphDiv1:

(Note that A[u][u]=0, for every u). Is it correct? Isn't A[u][u] negative when there is self-loop in u?

 Posted by sluki at [Jul 05, 2014 13:58](#) | [Reply To This](#)

Changed that part.

 Posted by vexorian at [Jul 05, 2014 16:44](#) | [Reply To This](#)

In NegativeGraphDiv2 why we can not use Dijkstra after making new graph ?

I think negative edges can not be problem in our Dijkstra, because whenever we use a negative edge, we can not update any previous node. I mean, the reason that make Dijkstra wrong on a graph with negative edge, isn't true here.

 Posted by arabpour.m at [Jul 25, 2014 14:43](#) | [Reply To This](#)

Can someone please explain the $O(N^2)$ solution for SumOfPower div 2 250 pt. problem?

```
int ans = 0;
```

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

```
Unknown macro: {   ans += (i+1)*(N - i)*array[i]; }
```

Didn't understand that at all.

Thanks.

 Posted by blizzard_one at [Aug 26, 2014 02:59](#) | [Reply To This](#)

 [Add Comment](#)

[Home](#) | [About TopCoder](#) | [Press Room](#) | [Contact Us](#) | [Privacy](#) | [Terms](#)
[Developer Center](#) | [Corporate Services](#)

Copyright TopCoder, Inc. 2001-2014