DOI: 10.36243/fmtu-2014.093

FIATAL MŰSZAKIAK
TUDOMÁNYOS ÜLÉSSZAKA XIX.

Kolozsvár, 2014. március 20–21.

KLASSZIKUS ALGORITMUSOK LEGRÖVIDEBB ÚT KERESÉSHEZ PROGRAMOZÁS VERSENYEKEN

ANALYSIS OF CLASSIC SHORTEST PATH ALGORITHMS USED DURING PROGRAMMING CONTESTS

Tânia Dias RIBEIRO(1), **Paulo José Lage ALVARENGA**(2)

(1) CAPES Foundation (Ministry of Education of Brazil), Address: Brasilia, DF, Zip Code 70.040-020 (Proc. Nº 88888.035405/2013-00).

(1) Óbuda University, John von Neumann Faculty of Informatics, Address: 1034 Budapest, Bécsi way 96/b. email: taniadiasr@gmail.com

(2) Federal University of Itajubá, Address: Rua Irmã Ivone Drummond, 200, Distrito Industrial II, Itabira/MG, CEP 31.160-110. pauloalvarenga@unifei.edu.br

Abstract

Programming contests occur all around the world, and the competitors need to be able to solve problems of different categories as fast as possible. Graph problems appear in several of this problems and the aim of this paper is to explain the Shortest Path Problem and show strategies to decide between three most used algorithms to deal with this problems: Dijkstra's, Bellman Ford's and Floyd Warshall's, as well as show vantages and disadvantages in the implementation of each one. Each of these algorithms has different approaches but can be used to solve several problems. It is shown how Dijkstra’s can be implemented easily with use of Standard Template Library of C++ language, and the differences between implementation of Bellman Ford’s and Floyd Warshall’s. The last one dedicated mostly to All-Pair Shortest Path problems, but can be extended to other problems. The measurement considered to influence the choice between these classical algorithms were: the code size, easiness to implement and differences in complexity – which is needed due to the required time limit of the problems.

Keywords: programming contest, graphs, shortest path.

Összefoglaló

A világszerte előforduló programozási versenyeken a versenyzők feladata külömböző problémák megoldása. Ezeken a megmérettetéseken gyakran előfordulnak gráf problémák. Ezen tanulmány célja, hogy ismertesse a legrövidebb út problémáját, és bemutassa a három legismertebb algoritmus (Dijkstra, Bellman Ford és a Floyd Warshall) alkalmazását, előnyeit, és hátrányait. Mindegyik algoritmusnak más a megközelítése, ezért különböző problémákra alkalmasak. Ez a tanulmány elemzi a kódokat mind méret, implementáció, komplexitás, melyek hatással vannak a futási időre.

Kulcsszavak: programozási verseny, gráfok, legrövidebb út.

1. Introduction

Programming contests are not just about solving problems, but solve it efficiently and in the lesser than established thresholds. It's usual in contests teams that solve the same number of problems, but amongst the tiebreaker criteria is the speed of problem submissions without errors. In this paper we discuss classical algorithms used to solve common problems involving shortest path in a graph. We will use classical problems of finding the shortest path, available in one of several online judges system, like UVA Online Judge (Available at: http://uva.onlinejudge.org/) and Sphere Online Judge (Available at: http://www.spoj.com/). The focus of this manuscript will concern about search in graph, question that appears inside many of the classical graph problems. The TSP between a vertex s to vertex t is guaranteed when it is a direct path from s to t that no other possible path has a lower weight [1][2].

2. Classical Algorithms: Dijkstra’s, Bellman Ford’s and Floyd Warshall’s

For each kind of problem mentioned above, there are different classical algorithms, like Dijkstra's, Bellman-Ford (BF), A* search, Floyd-Warshall (FW), Johnson's and Viterbi. Of those, the most frequently used are Dijkstra's, Floyd-Warshall and Bellman-Ford, reason why they'll be discussed. The problem starts with the question: "Given a weighted graph G and a starting source vertex s, what are the shortest paths from s to the other vertices of G?"[2] and is called Single-Source Shortest Path on a weighted graph, and can be solved using any of the three algorithms if certain conditions about the edges weight are satisfied, which leads to think about what will be the better one.

Dijkstra’s is a greedy algorithm, it can run at a good complexity of O((E+V)log V). One of the most important factors in its implementation is the use of a structure to choose what will be the next vertex visited, usually a priority queue. First of all occurs the common process of initialization, in which the vectors who save the data (dist[] and p[], for distance and previous respectively) are initialized with proper values (dist[source] = 0 and dist[n-source] = infinity), running at O(V). Later, technique of relaxation is used, which will start with the source (dist = 0) for first case. Then, relax(u,v) sets dist[v] = min(dist[v], dist[u] + weight(u,v)), what can be seen as the first construction, since in this part dist[v] = infinity and dist[u] + weight(u,v) < infinity. Each vertex will not be visited again, and after putting his neighbors at the queue, the procedure will replace u with the vertex x of the queue who has the smallest dist[x], as this is a greedy strategy [2][3].

The pseudocode below, from [3] shows an implementation of the Dijkstra’s, considering a graph G with edges with weight w, and S is the set of vertices with final Shortest-Paths already determined, and Q is a min-priority queue of vertices keyed by their distances values.

Table 1. *Dijkstra’s and Bellman Ford’s Pseudocode*

Dijkstra (G, w, s)	Bellman-Ford (G, w, s)
Initialize(G, s)	Initialize(G, s)
S = ∅; Q = V //all vertex	for i = 1 to |V| - 1
while (Q != ∅)	for each edge (u,v) ∈ G.E
u = Extract-Min(Q); S = S U {u}	Relax(u,v,w)
for each vertex v ∈ G.Adj[u]	for each edge (u,v) ∈ E // negative cycles
Relax(u,v,w)	if (d[v] > d[u] + w (u,v)) return false
	return true

The aim of Bellman Ford’s algorithm is make possible to find shortest-path even if are negative edge-weighted. Its implementation is simple, and if been necessary to find negative cycles, this is possible using BF. The main differences between this one and Dijkstra concern about the how it does the relaxation of the edges. At the end a distance vectors is generated with the distances between the source vertex to all reachable others.

In Floyd-Warshall's algorithm the graph representation used is an adjacency matrix. This algorithm is mostly used when the problem is the classical All-Pair Shortest Paths (APSP), and it is applicable to positive or negative weight edges, but no negative cycles. The procedure consists in a 3 nested loops that make which calculates the distance between all pairs of vertices in a graph, leading to a O(V3) complexity. Considering the how slow it can be, the applicability of the Floyd Warshall's is just when the condition V <= 100 is satisfied. For a bigger amount, it can be solved using the two algorithms mentioned above, running each one V times (one for each vertex), but if the possibility of the edges be negative, so is necessary to use Bellman Ford's, what can generate a even slower procedure [2][3]. Calling M the V*V adjacency matrix, a simple way to illustrate the procedure follows below:

```
1. for k = 1 to |V|
2.   for i = 1 to |V|
3.     for j = 1 to |V|
4.          M[i][j] = min(M[i][j], M[i][k] + M[k][i])
```

3. Major differences in complexity and implementation

When programming for contests, after the correctness of the algorithm, the main concern is about time. The programmer must choose the simplest algorithm that works, what means satisfy the time limit and produce correct answers [2]. Your code need to be able to satisfy the worst case (the input bound), usually given in the problem. In The Shortest Path (SPOJ), it's necessary give as answer the minimum transportation cost from city to city required. The problem statement says that the maximum number of vertices (cities) can be 10000.

We can express E=V-1 for the worst case. So, for the Big-Oh Notation we could express these problems as O(V*(V-1)) = O(V^2) for Bellman-Ford's and O((V+V)*log_2(V)) = O (V*log_2(V)), for Dijkstra's. For illustration, whereas the constants ignored by the Big-Oh notation to represent the same weight in both analyzed algorithms, the worst case could be obtained by applying the formula. Calling Bellman-Ford's to solve it, we will have:

$O(V \cdot E)=O(10000 \cdot 9999)=99990000$, because we can assume that a city can have V-1 neighbors. The runtime using this algorithm is unfeasible to satisfy the 5s time limit. On the other hand, using Dijkstra's, with V = 10000 and E = 9999: $O((V+E) \cdot \log_2 V) \cong 265741$, something about 376× faster. Although the problem is to determine the distance between two given vertex, using the Floyd Warshall's is clearly impracticable. But in Trip Routing UVA's problem, the problem states clearly that won't be more than 100 cities, leading to a small graph, and the weight of the edges is positive distances. A easier one is Risk UVA's problem, where each edge has weight = 1, a fact that makes it possible to solve using BFS, however the easy implementation of Floyd Warshall's code, its small code size and the small input size makes their use more attractive.

Comparing these factors, it is possible to see that Floyd Warshall is faster to implement but is not appropriate to all cases, as it depends on the maximum number of vertices that the problem may have. To do a good implementation of Dijkstra's using C++, a language widely used at the contests, is indicated using the containers of the Standard Template Library (STL), so the required task of implementing the priority queue becomes easy, because it's already done and ready for use.

The code below is the Dijkstra's main procedure, and has 9 lines of code, which should be added with previous reading procedures to fill the adjacency list and after the procedure, the methods to print the output. To understand the code, its uses this structure to the Adjacency list representation, where vector and pair are containers available in the STL: vector< vector< pair<int, int > > > AdjList; Then reading the edges, initialize de distance vector (also using vector container and setting all positions (V+1) to infinity), we can do:

priority_queue<pair<int,int>,vector<pair<int,int>>,greater<pair<int,int>>>pq;

Then setting the first edge with the source s: pq.push(pair<int,int>(0, s)); Using these implementations, the main procedure of Dijkstra states as follow:

```
while (!pq.empty()){
  pair<int,int> front = pq.top();  pq.pop();
  int d = front.first, u = front.second;
  if (d > dist[u]) continue;
     for (int j = 0; j < AdjList[u].size(); j++){
        pair<int,int> v = AdjList[u][j];
        if (dist[u] + v.second < dist[v.first]){
           dist[v.first] = dist[u] + v.second;
           pq.push(pair<int,int>(dist[v.first], v.first)); }   }   }
```

As in programming contests the teams are able to bring books and material to help during the competition, it’s not necessary to memorize how to implement these structures, but it’s necessary to know how they work, and be aware that it will make everything easier.

To use Bellman Ford’s algorithm faster as possible, the programmer can use the same structures to allocate the edges and the memory limit goes down. After allocate the edges and initialize the distance vector properly, the main procedure of Bellman Ford’s can be done easily with less than 10 lines of code. The faster to code, as demonstrated, is Floyd Warshall’s, since it only needs a matrix with V*V size, with the weight of the conections propperly defined (or infinity) and the straightfoward main 3 nested loops to solve the problem.

4. Conclusion

These observations lead us to conclude that each of these 3 algorithms has specific applications, but they can be suitable for other approaches as well, since the problem parameters may be adjusted on the procedures of each algorithm (like the use of Dijkstra’s to solve problems of All-Pair Shortest Path, in which it have to be adapted to make a call of Dijkstra's for each vertex instead of only once). Despite being the fastest, Dijkstra isn’t able to handle negative weight. For this kind of problems, Bellman Ford’s and Floyd Warshall’s are the perfect choices; furthermore only Bellman Ford’s is suitable when the problem may have negative cycles. The programming contestant needs to have in mind that devote part of his time on better understanding of the language resources, like C++ STL, will lead to better implementations of these algorithms, saving development time and granting algorithm efficiency, as well as avoiding unnecessary complications during the contest.

Since these concepts are dominated, it's time to move forward and understand others graph problems, learning how to detect it and which are the techniques that will lead to the correct answers.

References

[1] R. Sedgewick and K. Wayne, Kevin:. *Algorithms*. Pearson Eductation, Boston, MA (2011), p. 515-683.

[2] S. Halim: *Competitive Programming*. Online Version (2011), Available at: https://sites.google.com/site/stevenhalim/ accessed: January 2014.

[3] T.H. Cormen, C. E. Leiserso, R. L. Rivest and C. Stein: *Introduction to Algorithms*, Massachusetts, USA (2009), p. 643-695.