5
Polynomial Method for APSP

Version: 190

In this and the next chapter we will study the All-Pairs-Shortest-Paths problem. We will see the fastest known algorithm for APSP and prove several problems to be equivalent to it.

Problem 5.1.
All-Pairs-Shortest-Paths (APSP)
Given: *A directed weighted graph G on n nodes.*
Determine: *The distance between any pair of nodes u, v in G.*
Conjecture: *No $\mathcal{O}(n^{3-\varepsilon})$-time algorithm exists.*

We start by formally defining all notions used in this problem definition. Recall that a directed graph G consists of a set of vertices $V(G)$ and a set of edges $E(G) \subseteq V(G) \times V(G)$. An edge $(u, v) \in E(G)$ points from u to v. We typically assume that $V(G) = \{1, \ldots, n\}$. In a **weighted** directed graph, each edge $e \in E(G)$ is associated with an edge weight $w(e)$. A **path** from u to v in G is a sequence of distinct vertices $v_1, \ldots, v_k \in V(G)$ such that $u = v_1, v = v_k$, and $(v_i, v_{i+1}) \in E(G)$ for all $1 \leq i < k$. The number of **hops** of this path is $k - 1$. The **length** of this path is $\sum_{i=1}^{k-1} w(v_i, v_{i+1})$. The **distance** from u to v, denoted by $d(u, v) = d_G(u, v)$, is the minimal length of any path from u to v. A path attaining this minimal length is called a **shortest path** from u to v. In APSP, our task is to compute the distance $d(u, v)$ for each pair of nodes $u, v \in V(G)$.

We will assume that all edge weights are integral and, in particular, come from the set $\{1, \ldots, W\}$. Moreover, we assume that $W \leq n^c$ for some (possibly very large) constant $c > 0$. Thus, each weight fits into a constant number of machine cells and can be accessed in constant time.

As is well known, APSP can be solved in time[1] $\mathcal{O}(n^3)$. A sequence of results improved this running time by logarithmic factors (and even

[1] Robert W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5(6):345, 1962; and Stephen Warshall. A theorem on boolean matrices. *J. ACM*, 9(1):11–12, 1962

by superpolylogarithmic factors, as we will see below). However, no $\mathcal{O}(n^{3-\varepsilon})$-time algorithm is known. Since APSP is an extremely fundamental problem and the $\mathcal{O}(n^3)$-time algorithm is widely taught in undergraduate courses, an improvement to time $\mathcal{O}(n^{3-\varepsilon})$ would indeed be very surprising. This gives rise to the APSP Hypothesis, stating that APSP has no $\mathcal{O}(n^{3-\varepsilon})$-time algorithm for any $\varepsilon > 0$.

In this chapter we will see the fastest known algorithm for APSP, which uses the polynomial method. (In the next chapter we will then explore conditional lower bounds based on the APSP Hypothesis.)

Theorem 5.2. *(Polynomial Method for APSP[2]) APSP has a randomized algorithm that runs in time*

$$n^3/2^{\Omega(\sqrt{\log n})}.$$

We remark that $2^{\sqrt{\log n}}$ grows faster than $(\log n)^c$ and slower than n^ε, for any $c, \varepsilon > 0$. In particular, the theorem yields strong lower order improvements, but does not falsify the APSP Hypothesis.

[2] Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *Proc. 46th Annual ACM Symposium on Theory of Computing (STOC'14)*, pages 664–673, 2014

5.1 *Equivalence with Min-Plus Product*

Our algorithm for APSP is based on a classic equivalence of APSP and the so-called Min-Plus Product.

Problem 5.3.

Min-Plus Product	
Given:	*Two $n \times n$-matrices A, B*
Determine:	*The $n \times n$-matrix $C = A \otimes B$ with $C[i,j] = \min_{1\le k\le n} A[i,k] + B[k,j]$.*

Note that in standard matrix multiplication we would instead want to compute the matrix with entries $C[i,j] = \sum_{1\le k\le n} A[i,k] \cdot B[k,j]$. Since "$\sum$" and "$\cdot$" are replaced by "min" and "$+$", the above problem is called Min-Plus Product (or $(\min,+)$-Product).

The naive algorithm that simply evaluates the definition of C solves Min-Plus Product in time $\mathcal{O}(n^3)$. In contrast to standard matrix multiplication, no fast matrix multiplication à la Strassen is known for Min-Plus Product, in particular, no $\mathcal{O}(n^{3-\varepsilon})$-time algorithm is known.

Similarly as for APSP, we will assume that all entries of the matrices A, B lie in the set $\{1, \dots, W, \infty\}$, where again $W \le n^c$. Here, ∞-entries correspond to non-edges in the APSP problem.

We show that APSP and Min-Plus Product are tightly connected. This is somewhat surprising, since Min-Plus Product has a naive $\mathcal{O}(n^3)$-time algorithm, while for APSP this is non-trivial.

Lemma 5.4. *If APSP is in time $T(n)$, then Min-Plus Product is in time $\mathcal{O}(T(n))$. If Min-Plus Product is in time $T(n)$, then APSP is in time $\mathcal{O}(T(n) \log n)$.*

In fact, this statement even holds without the factor $\log n$, but then requires a more complicated proof.

We will prove this lemma in the next chapter, where we explore additional equivalences with APSP. We will first use it in order to design an algorithm for APSP.

5.2 *Polynomial Method for Min-Plus Product*

Since APSP is equivalent to Min-Plus Product, in order to prove Theorem 5.2 it suffices[3] to design an algorithm for Min-Plus Product that runs in time $n^3/2^{\Omega(\sqrt{\log n})}$.

[3] Indeed, the additional $\log(n)$-factor from Lemma 5.4 is dominated by the factor $1/2^{\Omega(\sqrt{\log n})}$.

The algorithm has similarities to the algorithm for OV presented in the last chapter: they both use the polynomial method in order to translate some subproblem into a polynomial evaluation problem. However, apart from this general approach the algorithms differ significantly.

We denote by A, B the given $n \times n$-matrices. We again have a parameter s and a number of groups $g = \frac{n}{s}$. This time we will choose $s = 2^{\varepsilon\sqrt{\log n}}$ for some small constant $\varepsilon > 0$.

For simplicity, assume that n and s are powers of 2, so s divides n.

1. Reduction to rectangular case: We split matrix A into g many $n \times s$-submatrices $A_1, \dots, A_g$. Similarly, we split matrix B into g many $s \times n$-submatrices $B_1, \dots, B_g$.

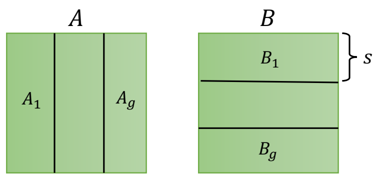

Figure 5.1: Illustration of matrix splitting.

We want to compute $C_x := A_x \otimes B_x$ for each $1 \leq x \leq g$. Note that from these matrices we can compute the output $C = A \otimes B$ by $C[i,j] = \min_{1 \leq x \leq g} C_x[i,j]$.

Each subproblem $A_x \otimes B_x$ is the Min-Plus Product of an $n \times s$-matrix with an $s \times n$-matrix[4]. We will show how to solve each such subproblem in time $\widetilde{\mathcal{O}}(n^2)$. Over all $g = \frac{n}{s}$ subproblems this will yield total time $\widetilde{\mathcal{O}}(\frac{n^3}{s}) = n^3/2^{\Omega(\sqrt{\log n})}$.

[4] This is reminiscent of rectangular matrix multiplication from the last chapter.

2. Witness problem: From now on we denote by A, B one of the subproblems from above, i.e., A is an $n \times s$-matrix and B is an $s \times n$-matrix and our goal is to compute $A \otimes B$. Consider the *witness problem*: Given A, B, compute an $n \times n$-matrix W where:

$$W[i,j] \text{ is some } k \in [s] \text{ minimizing } A[i,k] + B[k,j].$$

From these witnesses, we can reconstruct $C = A \otimes B$ by $C[i,j] = A[i, W[i,j]] + B[W[i,j], j]$. We can thus focus on the witness problem.

3. Make witnesses unique: Suppose that we replace A, B by A', B' with

$$A'[i,k] := n \cdot A[i,k] + k,$$
$$B'[k,j] := n \cdot B[k,j].$$

Their product $C' = A' \otimes B'$ satisfies

$$C'[i,j] = \min_k A'[i,k] + B'[k,j] = \min_k \big(n \cdot (A[i,k] + B[k,j]) + k\big).$$

Note that due to the large factor n, the primary objective of this minimization problem is to minimize the sum $A[i,k] + B[k,j]$, and among all k minimizing the primary objective, the secondary objective is to minimize k. In particular, there is a *unique* k minimizing $A'[i,k] + B'[k,j]$, because among all k minimizing $A[i,k] + B[k,j]$ there is a unique minimal k.

Therefore, without loss of generality we can phrase the witness problems as computing the $n \times n$-matrix W where:

$$W[i,j] \text{ is the unique } k \in [s] \text{ minimizing } A[i,k] + B[k,j].$$

4. Fredman's trick: We rewrite the definition of the witness matrix as:

$$\begin{aligned} W[i,j] = \text{unique } k \in [s] \text{ s.t. } \forall \ell \in [s]\colon & A[i,k] + B[k,j] \le A[i,\ell] + B[\ell,j] \\ \Leftrightarrow & A[i,k] - A[i,\ell] \le B[\ell,j] - B[k,j] \\ \Leftrightarrow & A[i,k,\ell] \le B[j,k,\ell], \end{aligned}$$

for $A[i,k,\ell] := A[i,k] - A[i,\ell]$ and $B[j,k,\ell] := B[\ell,j] - B[k,j]$. This rearrangement is known as Fredman's trick[5]. It makes the left hand side ($A[i,k,\ell]$) independent of j and the right hand side ($B[j,k,\ell]$) independent of i. Moreover, there are only ns^2 different values $A[i,k,\ell]$, which is much less compared to the $n^2 s$ different sums $A[i,k] + B[k,j]$. This allows us to precompute all numbers $A[i,k,\ell], B[j,k,\ell]$ in time much less than $\mathcal{O}(n^2)$.

More precisely, we should add W to $A[i,k,\ell]$ and $B[j,k,\ell]$ to ensure positivity.

[5] Michael L. Fredman. New bounds on the complexity of the shortest path problem. *SIAM J. Comput.*, 5(1):83–89, 1976

We arrived at the following formulation of the witness problem:

$$W[i,j] = \text{unique } k \in [s]\colon \bigwedge_{\ell \in [s]} \big[A[i,k,\ell] \le B[j,k,\ell]\big].$$

Here $[x \le y]$ evaluates to 1 (or true) if $x \le y$ holds, and to 0 (or false) otherwise.

5. Bits of witnesses: Analogously to the polynomial method for OV, our goal is to convert the witness problem first into a Boolean circuit and then into a probabilistic polynomial. However, a Boolean circuit computes one bit, but a witness $W[i,j]$ consists of $\log s$ bits, as $W[i,j]$ can take any value in $[s]$. For this reason, we consider the binary representation of witnesses:

$$W[i,j] = \sum_{b=0}^{\log s} 2^b \cdot W_b[i,j],$$

where $W_b[i,j] \in \{0,1\}$ is the b-th bit of $W[i,j]$. We will construct a circuit/probabilistic polynomial for each $W_b[i,j]$.

Claim 5.5. *We have*

$$W_b[i,j] = \bigvee_{\substack{k\in[s] \\ b\text{-th bit} \\ \text{of } k \text{ is } 1}} \bigwedge_{\ell\in[s]} \big[A[i,k,\ell] \le B[j,k,\ell]\big].$$

Proof. Note that the expression $\bigwedge_{\ell\in[s]} \big[A[i,k,\ell] \le B[j,k,\ell]\big]$ is true if and only if $k = W[i,j]$. If the b-th bit of $W[i,j]$ is 1, then in the outer $\bigvee$ we enumerate $k = W[i,j]$, and for this k the inner $\bigwedge$ evaluates to true, so we have $W_b[i,j] = 1$. Similarly, if the b-th bit of $W[i,j]$ is 0, then in the outer $\bigvee$ we always have $k \neq W[i,j]$, and the inner $\bigwedge$ always evaluates to false. □

Here we make use of the uniqueness of the witnesses.

Before we show how to express the inner part $\big[A[i,k,\ell] \le B[j,k,\ell]\big]$ as a Boolean circuit, let us discuss how we convert the outer $\bigvee$ and $\bigwedge$ from a Boolean circuit into a probabilistic polynomial. For the outer $\bigvee$, the next claim shows that we can simply replace it by a $\bigoplus$. For the inner $\bigwedge$, we will then later apply Razborov-Smolensky.

Claim 5.6. *We have*

$$W_b[i,j] = \bigoplus_{\substack{k\in[s] \\ b\text{-th bit} \\ \text{of } k \text{ is } 1}} \bigwedge_{\ell\in[s]} \big[A[i,k,\ell] \le B[j,k,\ell]\big].$$

Proof. Note that if at most one x_i is true then we have $x_1 \vee \ldots \vee x_n = x_1 \oplus \ldots \oplus x_n$. Moreover, $\bigwedge_{\ell\in[s]} \big[A[i,k,\ell] \le B[j,k,\ell]\big]$ is true if and only if $k = W[i,j]$, so it is true for at most one enumerated k. Combining these facts yields the claim. □

6. LEQ circuit: It remains to to convert the expression $[a \le b]$ into a Boolean circuit (and then further to a probabilistic polynomial). For numbers $x, y \in \{1,\ldots,W\}$ denote the binary representation by $x = \sum_{b=0}^{B} 2^b \cdot x_b$, and similarly for y. Note that we have $x \le y$ if and only if $x = y$ or there exists an index i such that the i-th bit of x is 0, the i-th bit of y is 1, and all higher bits are equal. This yields a Boolean circuit that checks whether $x \le y$:

Here $B = \lceil \log W \rceil$.

$$LEQ(x,y) = \left(\bigwedge_{j=0}^{B} (x_j = y_j) \right) \vee \bigvee_{i=0}^{B} \left((x_i = 0) \wedge (y_i = 1) \wedge \bigwedge_{j=i+1}^{B} (x_j = y_j) \right)$$

We slightly rewrite this expression to get it closer to a polynomial over $\mathbb{F}_2$, as follows.

Claim 5.7. *There are values $\alpha_{h,j}, \beta_{h,j}, \gamma_{h,j} \in \{0,1\}$ and $i_j \in \{0,\dots,B\}$ for $h, j \in [B+2]$ such that*

$$LEQ(x,y) = \bigoplus_{h=1}^{B+2} \bigwedge_{j=1}^{B+2} \left(\gamma_{h,j} \oplus x_{i_j}^{\alpha_{h,j}} \oplus y_{i_j}^{\beta_{h,j}}\right).$$

Proof. First note that $LEQ(x,y)$ considers $B+2$ cases (either $x = y$ or for some $0 \le i \le B$ the i-th bit of x is smaller than in y but all higher bits are equal). Since at most one of these cases occurs for any x, y, we can replace the $\bigvee$ over these $B+2$ cases by a $\bigoplus$. Further, note that each case considers a $\bigwedge$ over at most $B+2$ constraints of the form $x_i = y_i$ or $x_i = 0$ or $y_i = 1$. We rewrite $x_i = y_i$ as $1 \oplus x_i \oplus y_i$. We rewrite $x_i = 0$ as $0 \oplus x_i \oplus y_i^0$. We rewrite $y_i = 1$ as $1 \oplus x_i^0 \oplus y_i$. By possibly repeating constraints to fill up to exactly $B+2$ constraints, we have written $LEQ(x,y)$ in the claimed form. □

Note that $y_i^0 = x_i^0 = 1$.

6. Probabilistic polynomial for LEQ: We apply Razborov-Smolensky to the expression $E_h := \bigwedge_{j=1}^{B+2} \left(\gamma_{h,j} \oplus x_{i_j}^{\alpha_{h,j}} \oplus y_{i_j}^{\beta_{h,j}}\right)$ to obtain

$$\begin{aligned} E_h \approx_{2^{-t}} E_{h,r} :=& RS_t\left(\left\{\gamma_{h,j} \oplus x_{i_j}^{\alpha_{h,j}} \oplus y_{i_j}^{\beta_{h,j}} \mid 1 \le j \le B+2\right\}\right) \\ =& \prod_{q=1}^{t}\left(1 \oplus \bigoplus_{j=1}^{B+2} r_{h,q,j} \cdot \left(1 \oplus \gamma_{h,j} \oplus x_{i_j}^{\alpha_{h,j}} \oplus y_{i_j}^{\beta_{h,j}}\right)\right), \end{aligned}$$

where each $r_{h,q,j}$ is a random bit and r denotes the sequence of all these random bits. Note that the right hand side is a product of t *linear functions* in the variables x_i, y_i (since the exponents $\alpha_{h,j}, \beta_{h,j}$ are in $\{0,1\}$). We split these linear functions into a sum of two expressions, one depending only on x and one depending only on y.[6] In other words, we write

[6] The constant term of the linear function can be arbitrarily assigned to the left or right expression

$$E_{h,r} = \prod_{q=1}^{t}\left(L_{h,q,r}(x) \oplus R_{h,q,r}(y)\right).$$

Here each $L_{h,q,r}(x)$ depends only on the input x, the random bits r, and the indices h, q. It turns out that for all our applications of the LEQ circuit we can precompute the values $L_{h,q,r}(x)$ and $R_{h,q,r}(y)$. Therefore, we may treat the symbols $L_{h,q,r}(x), R_{h,q,r}(y)$ as new variables that are the actual inputs to the LEQ circuit. This helps, because in terms of these new variables it is straightforward to expand $E_{h,r}$ into a sum of monomials:

$$E_{h,r} = \bigoplus_{z \in \{0,1\}^t} \prod_{q=1}^{t} L_{h,q,r}(x)^{z_q} \cdot R_{h,q,r}(y)^{1-z_q}.$$

We have thus written $E_{h,r}$ as a sum of 2^t monomials.

Recalling Claim 5.7, we now have

$$LEQ(x,y) = \bigoplus_{h=1}^{B+2} E_h \approx_{(B+2)2^{-t}} \bigoplus_{h=1}^{B+2} E_{h,r},$$

where we used the union bound to bound the error of $B+2$ replacements of the form $E_h \approx_{2^{-t}} E_{h,r}$ by at most $(B+2)\cdot 2^{-t}$. We write $LEQ_r(x,y)$ for the final term on the right hand side. Since each $E_{h,r}$ is a sum of 2^t monomials, $LEQ_r(x,y)$ is a sum of $(B+2)\cdot 2^t$ monomials (in terms of the new variables $L_{h,q,r}(x), R_{h,q,r}(y)$).

8. *Final probabilistic polynomial:* We now use the expression from Claim 5.6 for the witness bit $W_b[i,j]$, replace its inner $\bigwedge$ by an application of Razborov-Smolensky, and replace the $\leq$ by the probabilistic polynomial LEQ_r. This yields the probabilistic polynomial

$$W_{b,r}[i,j] := \bigoplus_{\substack{k\in[s] \\ b\text{-th bit} \\ \text{of } k \text{ is } 1}} RS_{\log(8s)}\big(\{LEQ_r(A[i,k,\ell],B[j,k,\ell]) \mid 1\leq \ell \leq s\}\big).$$

In the following we analyze the error probability and the number of monomials of this probabilistic polynomial.

Claim 5.8. *We have*

$$W_b[i,j] \approx_{\frac{1}{4}} W_{b,r}[i,j].$$

Proof. There are two sources of errors: (1) $RS_{\log(8s)}(.)$ errs with probability at most $2^{-\log(8s)} = \frac{1}{8s}$. Since we use this transformation for at most s values of k, we obtain total error probability at most $\frac{1}{8}$ from this term. (2) LEQ_r errs with probability at most $(B+2)\cdot 2^{-t}$. Since we apply LEQ_r exactly s^2 times (over s values of k and s values of ℓ), we obtain total error probability $s^2(B+2)2^{-t}$ from this term. We set

$$t := \log(8s^2(B+2)),$$

to make this error probability at most $\frac{1}{8}$. In total, this yields error probability at most $\frac{1}{4}$. □

Claim 5.9. *Set $s = 2^{\varepsilon\sqrt{\log n}}$, where $\varepsilon > 0$ is a sufficiently small constant and n is at least a sufficiently large constant. Then the probabilistic polynomial $W_{b,r}[i,j]$ consists of at most $n^{0.1}$ monomials.*

This bound enables the fast polynomial evaluation from the polynomial method.

Proof. Observe that if $p_1,\dots,p_s$ are polynomials consisting of m' monomials, then $RS_t(p_1,\dots,p_s)$ consists of at most $(s\cdot m'+1)^t$ monomials (by simply counting all possible terms in the expansion). Using this bound on $W_{b,r}[i,j]$ shows that its number of monomials is at most

$$m \le s \cdot \left(s \cdot (B+2) \cdot 2^t + 1\right)^{\log(8s)} = s \cdot \left(8s^3(B+2)^2\right)^{\log(8s)},$$

where we used our choice of $t = \log(8s^2(B+2))$.

Recall that we assume all our weights to lie in $\{1, \dots, W\}$, where $W \le n^c$. In particular, each weight consists of $B = \mathcal{O}(\log n)$ bits. By our choice of $s = 2^{\varepsilon\sqrt{\log n}}$ we have $s \gg B$, so B is a lower order term in our bound on m. This allows us to bound

$$m \le s^{\mathcal{O}(\log s)}.$$

More precisely, there are constants $C, s_0 > 0$ such that for any $s \ge s_0$ we have $m \le s^{C \cdot \log s}$. This yields the upper bound

$$m \le s^{C \cdot \log s} = \left(2^{\varepsilon\sqrt{\log n}}\right)^{C\varepsilon\sqrt{\log n}} = 2^{C\varepsilon^2 \log n} = n^{C\varepsilon^2}.$$

For sufficiently small constant $\varepsilon > 0$, we thus ensure $m \le n^{0.1}$. $\square$

Final Algorithm

Recall that we split our original instance A, B into rectangular subproblems. On each subproblem $A_x \otimes B_x$, in order to compute witnesses we consider the probabilistic polynomial $W_{b,r}$ for the b-th witness bit. We boost its constant success probability by drawing $200 \ln n$ samples; this yields polynomials $p_{1,b}, \dots, p_{200 \ln n, b}$. For each polynomial $p_{t,b}$ we precompute the values of the new variables: $L_{h,q,r}(A_x[i,k] - A_x[i,\ell])$ and $R_{h,q,r}(B_x[\ell,j] - B_x[k,j])$. This allows us to evaluate $p_{t,b}$ on all pairs i, j. Each evaluation gives a guess for the bit $W_b[i,j]$ that is correct with probability at least $\frac{3}{4}$. The majority of these guesses is correct with high probability. Combining the computed witnesses for all subproblems yields the correct result $C = A \otimes B$ with high probability. This algorithm is described by the following peudocode.

Final Algorithm On input A, B we do the following.

```
1. Split $A$ into $n \times s$-submatrices $A_1, \dots, A_g$ and $B$ into $s \times n$-submatrices
   $B_1, \dots, B_g$.
2. For each $1 \le x \le g$ do:
   3. For each $t = 1, \dots, 200 \ln n$ and $b = 0, \dots, \log s$ do:
      4. Sample a polynomial $p_{t,b}$ from the distribution $W_{b,r}$ by sam-
         pling the random bits $r$.
```

```
        5. Compute the values of the new variables:
               $(\tilde{A}_i)_{h,q,k,\ell} := L_{h,q,r}(A_x[i,k] - A_x[i,\ell])$
               $(\tilde{B}_j)_{h,q,k,\ell} := R_{h,q,r}(B_x[\ell,j] - B_x[k,j])$
        6. Evaluate $p_{t,b}(\tilde{A}_i, \tilde{B}_j)$ for all pairs $1 \le i,j \le n$
    7. Let $W^x_{b,i,j}$ be the majority among $p_{1,b}(\tilde{A}_i, \tilde{B}_j), \dots, p_{200\ln n,b}(\tilde{A}_i, \tilde{B}_j)$.
    8. Let $W^x_{i,j} := \sum_{b=0}^{\log s} 2^b \cdot W^x_{b,i,j}$.
9. Let $C[i,j] := \min_{1 \le x \le g} A[i, W^x_{i,j}] + B[W^x_{i,j}, j]$.
```

The use of boosting immediately implies a success probability of at least $1 - \frac{1}{n}$. We checked above that the number of monomials satisfies $m \le n^{0.1}$ and thus fast polynomial evaluation in time $\mathcal{O}(n^2 \log^2 n)$ can be applied. In total over $\mathcal{O}(g \log^2 n) = \mathcal{O}(\frac{n}{s} \log^2 n)$ iterations, this step takes time

$$\mathcal{O}\left(\frac{n^3}{s} \log^4 n\right) = \frac{n^3}{2^{\Omega(\sqrt{\log n})}}.$$

It can be checked that all other steps of this algorithm are dominated by this running time.

Bibliography

Version: 190

Robert W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5(6): 345, 1962.

Michael L. Fredman. New bounds on the complexity of the shortest path problem. *SIAM J. Comput.*, 5(1):83–89, 1976.

Stephen Warshall. A theorem on boolean matrices. *J. ACM*, 9(1):11–12, 1962.

Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *Proc. 46th Annual ACM Symposium on Theory of Computing (STOC'14)*, pages 664–673, 2014.