Chapter VI

All Pair Shortest Paths and Matrix Multiplication

VI.1 APSPs and Matrix Multiplication

There is a close similarity between the inner loop in the APSP algorithm and matrix multiplication. Recall that for $n \times n$ matrices $A = (a_{ij})$ and $B = (b_{ij})$, its product $C = (c_{ij})$ is given by

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}.$$

This product is computed by the algorithm

```
MatrixMultiply ($A, B$)
    for $i \leftarrow 1$ to $n$
        for $j \leftarrow 1$ to $n$
            $C[i][j] \leftarrow 0$
            for $k \leftarrow 1$ to $n$
                $C[i][j] \leftarrow C[i][j] + A[i][k] \cdot B[k][j]$
```

while the APSP inner loop is

```
for all vertices $u$
    for all vertices $v$
        $D'[u][v] \leftarrow \infty$
        for all vertices $x$
            $D'[u][v] \leftarrow \min(D'[u][v], D[u][x] + w[x][v])$.
```

There is a close similarity between both algorithms: in the second min substitutes + and + substitutes $\cdot$. Matrix multiplication can be performed in time $o(n^3)$ using Strassen's algorithm. and so this similarity brings the question of whether shortest paths can be computed in time $o(n^3)$. Below, we describe Strassen's algorithm and then describe how to compute all-pair shortest distances (APSD) in time $o(n^3)$ via matrix multiplication for the case of unit weights. For the all pair shortest paths problem (APSP)), clearly $o(n^3)$ time is not possible if one wants to store explicitly a shortest path for every pair. However, a compact $\Theta(n^2)$ representation is possible: for each i, j, store the first vertex after i in a shortest path from i to j. Such *successor* matrix can be found in time $o(n^3)$. For this, a solution to the problem of finding *witnesses* for boolean matrix multiplication is used. Some extensions of these results are possible for non-unit weights, but this extensions are omitted here.

VI.2 Strassen's Matrix Multiplication Algorithm

Consider the product of 2×2 matrices

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} r & s \\ t & u \end{pmatrix}$$

where

$$\begin{aligned} r &= ae + bg \\ s &= af + bh \\ t &= ce + dg \\ u &= cf + dh. \end{aligned}$$

If we are dealing with $2k \times 2k$ matrices, we can write similarly

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \cdot \begin{pmatrix} E & F \\ G & H \end{pmatrix} = \begin{pmatrix} R & S \\ T & U \end{pmatrix}$$

where $A, B, C, D, E, F, G, H, R, S, T, U$ are $k \times k$ matrices and

$$\begin{aligned} R &= AE + BG \\ S &= AF + BH \\ T &= CE + DG \\ U &= CF + DH. \end{aligned}$$

This leads to a divide-and-conquer algorithm whose running time satisfies the recurrence (assume n is a power of 2)

$$T(n) = 8T(n/2) + \Theta(n^2)$$

which has a solution $\Theta(n^3)$, no better than following the definition.

Strassen's algorithm is based on the following clever way of multiplying 2×2 matrices with only 7 multiplications instead of 8. With

$$\begin{aligned} p_1 &= a(f - h) \\ p_2 &= (a + b)h \\ p_3 &= (c + d)e \\ p_4 &= d(g - e) \\ p_5 &= (a + d)(e + h) \\ p_6 &= (b - d)(g + h) \\ p_7 &= (a - c)(e + f) \end{aligned}$$

then

$$\begin{aligned} r &= p_5 + p_6 + p_4 - p_2 \\ s &= p_1 + p_2 \\ t &= p_3 + p_4 \\ u &= p_1 - p_7 - p_3 + p_5. \end{aligned}$$

This can be easily but tediously verified. How did Strassen come up with this ? The CLRS describes an approach how one could have found this, but it is not very simple, and not clear that was the path followed by Strassen.

Now, similar equations apply to the $2k \times 2k$ matrix product above, which means that in the divide-and-conquer approach, recursion is performed on 7 pairs of matrices. This leads to the recurrence

$$T(n) = 7T(n/2) + \Theta(n^2)$$

which has solution $\Theta(n^{\log_2 7}) = \Theta(n^{2.80735})$. Improved methods have been devised since Strassen's original one; the current best running time is $O(n^{2.376})$ and is beyond this course. The best lower bound is the trivial $\Omega(n^2)$ (since there are that many input and output entries). Since the algorithms for shortest paths below depend on the running time for matrix multiplication, we will use a function $M(n)$ to indicate the time for multiplying two $n \times n$ matrices. (Note that we are assuming $O(1)$ time addition and multiplication integer operations.)

VI.3 All Pair Shortest Path Distances – Unit Weights

Given a graph G with unit weights, we want to compute for each pair of vertices u, v in $V(G)$ the shortest distance between them. We describe an algorithm based on matrix multiplication that runs in time $o(n^3)$.

The idea is to first to solve recursively the problem for the graph G', which has an edge (u, v) iff u and v are at distance 1 or 2 in G. Let $\mathbf{A}, \mathbf{D}$ and $\mathbf{A}', \mathbf{D}'$ be the adjacency and distance matrices for G and G' respectively. (We will be using the convention of writing the entries of a matrix with the corresponding lower case letter, and the row and column numbers as subindices. If convenient, we may also write $[\mathbf{A}]_{ij}$, instead of a_{ij}.)

Claim 1. *Let $\mathbf{Z} = \mathbf{A}^2$. Then there is a path of length 2 in G between vertices i and j iff $z_{ij} > 0$ (z_{ij} is actually the number of such paths).*

$\mathbf{Z}$ is used to compute $\mathbf{A}'$: $a'_{ij} = 1$ iff $i \neq j$ and $a_{ij} = 1$ or $z_{ij} > 0$. The bottom of the recursion happens when G' is the complete graph. This is the case iff G has diameter 2 (maximum shortest path length over all pairs of vertices) and then $\mathbf{D} = 2\mathbf{A}' - \mathbf{A}$.

How to obtain d_{ij} from d'_{ij} ? Roughly $d_{ij} = 2d'_{ij}$ but a correction needs to made depending on the parity of d_{ij}:

Observation 2. *For any i, j:*

(i) If d_{ij} is even then $d_{ij} = 2d'_{ij}$.

(ii) If d_{ij} is odd then $d_{ij} = 2d'_{ij} - 1$.

This does not seem very helpful since d_{ij} is what we are trying to compute. Fortunately, there is a fix. The following obsercation is a simple exercise:

Observation 3. *For $i \neq j$:*

(i) For any neighbor k of i, $d_{ij} - 1 \leq d_{kj} \leq d_{ij} + 1$.

(ii) There is a neighbor k of i such that $d_{kj} = d_{ij} - 1$.

The previous two observations can be used to prove the following:

Lemma 4. *For any $i \neq j$:*

(i) If d_{ij} is even, then $d'_{kj} \geq d'_{ij}$ for every neighbor k of i.

(ii) If d_{ij} is odd, then $d'_{kj} \leq d'_{ij}$ for every neighbor k of i. Moreover, there exists a neighbor k of i with $d'_{kj} < d'_{ij}$.

We leave the proof of this lemma as an exercise. Now, let $\deg(i)$ be the degree of i in G. Summing the inequalities in the previous lemma, we obtain:

Lemma 5. *For any $i \neq j$:*

$$d_{ij} \text{ is even iff } \sum_{\{k,i\} \in E(G)} d'_{kj} \geq d'_{ij} \cdot \deg(i).$$

Note z_{ii} (defined above) is equal to $\deg(i)$, and that

$$\sum_{\{k,i\} \in E(G)} d'_{kj} = \sum_k a_{ik} d'_{kj} = s_{ij}$$

where $\mathbf{S} = \mathbf{A}\mathbf{D}'$.

In summary, the algorithm works as follows:

```
MM-APSD ($\mathbf{A}$)
/* $\mathbf{A} = [a_{ij}]$ is the adjacency matrix of $G$ */
1.  $\mathbf{Z} \leftarrow \mathbf{A}^2$
2.  for all $i, j$ /* define $\mathbf{A}' = [a'_{ij}]$ */
            $a'_{ij} \leftarrow [i \neq j$ and ( $a_{ij} = 1$ or $z_{ij} > 0)]$
3.  if (for all $i \neq j$, $a'_{ij} = 1$) return $(2\mathbf{A}' - \mathbf{A})$
4.  $\mathbf{D}' \leftarrow$ MM-APSD($\mathbf{A}'$)
5.  $\mathbf{S} \leftarrow \mathbf{A}\mathbf{D}'$
6.  for all $i, j$ /* define $\mathbf{D} = [d_{ij}]$ */
            $d_{ij} \leftarrow 2d'_{ij} - [s_{ij} < d'_{ij} z_{ii}]$
7.  return ($\mathbf{D}$).
```

The running time is described by the recurrence, where n is the longest distance:

$$T(n, \delta) = 2M(n) + T(n, \lceil \delta/2 \rceil) + O(n^2),$$

which implies that $T(n, n) = O(M(n) \log n)$.

VI.4 Witnesses for Boolean Matrix Multiplication

Let **A** and **B** be $n \times n$ boolean (0/1) matrices. The *boolean product* of **A** and **B** is the matrix **P** with entries:

$$p_{ij} = \bigvee_{k=1}^{n} (a_{ik} \wedge b_{kj})$$

where $\vee$ and $\wedge$ are the boolean operators OR and AND. Thus, $p_{ij} = 1$ iff for some k, $a_{ik} = b_{kj} = 1$. Clearly, $\mathbf{P} = \mathbf{AB}$ can be computed in time $O(M(n))$, since we can handle the 0/1 entries as integers and obtain an integer product matrix **M**: $p_{ij} = 1$ iff $m_{ij} > 0$. In some applications though, it does not suffice to know that $p_{ij} = 1$, one also wants an index k with $a_{ik} = b_{kj} = 1$, which is called a *witness.* Thus, a *witness matrix* for $\mathbf{P} = \mathbf{AB}$ is a matrix **W** with

$$w_{ij} = \begin{cases} 0 & \text{if } p_{ij} = 0 \\ k \text{ with } a_{ik} = b_{kj} = 1 & \text{if } p_{ij} = 1 \end{cases}$$

It seems difficult to do better than checking each k foor each pair i, j. Since we can compute **P** in time $O(M(n)) = o(n^3)$, we would like to be able to compute a witness matrix also in subcubic time.
Suppose that for i, j with $p_{ij} = 1$, there is a unique witness k_{ij}. Note that then $\sum_{k=1}^{n} (ka_{ik}) b_{kj} = k_{ij}$. If we define $\hat{\mathbf{A}}$ by $\hat{a}_{ik} = ka_{ik}$ then the i, j entry of $\hat{\mathbf{A}}\mathbf{B}$ is a correct witness for $p_{ij} = 1$. On the other hand, if the witness for i, j is not unique then the i, j entry is "garbage", we cannot in general identify an index based on its value.[1]

Using this very specific solution, a general solution can be obtained with the help of randomization. Let w_{ij} be the number of witnesses for $p_{ij} = 1$. Suppose we let

$$\hat{a}^R_{ik} = r^{ij}_k k a_{ik}$$

where r^{ij}_k is a random variable

$$r^{ij}_k = \begin{cases} 1 & \text{with probability } \pi_{ij} \\ 0 & \text{with probability } 1 - \pi_{ij} \end{cases}$$

with

$$\frac{1}{2w_{ij}} \leq \pi_{ij} < \frac{1}{w_{ij}}.$$

Claim 6. *The probability that $\sum_{k=1}^{n} \hat{a}^R_{ik} b_{kj}$ is a witness for $p_{ij} = 1$ is at least $1/2e$ (where e is Euler's number, the base of natural log).*

Proof. To simplify notation, let $w = w_{ij}$, $\pi = \pi_{ij}$ with $1/2w \leq \pi < 1/w$. The question can be restated as follows: Suppose that there are $w \geq 1$ white balls and $n - w$ black balls. If we pick each of the n balls independently with probability π, what is the probability ρ that exactly one white ball is chosen ? This is easily computed:

$$\rho = w \cdot \pi \cdot (1 - \pi)^{w-1}.$$

Using the bounds for π, for $w > 1$, we get $\rho > (1/2)(1 - 1/w)^{w-1} \geq 1/2e$, where we have used $(1 - 1/w)^{w-1} \geq 1/e$, for $w > 1$. $\rho \geq 1/2e$ also applies for $w = 1$. This completes the proof. $\square$

[1] As it was suggested during the lecture, if we used $\hat{a}_{ik} = 2^k a_{ik}$ then we could identify all the witnesses. This however would use n-bit numbers and we can't reasonably continue assuming that integer operations take time $O(1)$.

We want a unique witness. Multiplying by r_k^{ij} gives this with probability at least $1/2e$. This is not sufficient. However, by repeating the "experiment" many times, the probability of getting exactly one witness becomes greater: if N trials are made, then the probability of failure in all of them is less than (using $1 - x \leq e^{-x}$)

$$(1 - 1/2e)^N \leq e^{-N/2e}.$$

Chosing $N = 2ec \log n$, the probability of failure is less than $1/n^c$. This is better because it means that it would fail only for a few entries i, j.

Now, since each i, j has possibly a different w_{ij} and we don't know them anyway, to handle all the range 1 to n, we try all the probabilities $\pi_s = 1/2^s$ with $s = 0, \ldots, \lceil \log n \rceil$. A way to achieve these probabilities is to start with a set $R = \{1, \ldots, n\}$ and iteratively take a sample from R with probability 1/2: each element of R is taken independently with probability 1/2 – we call this a (1/2)-sample from R. We make this the new set R. Thus, in the i-th iteration, k is in R with probability $1/2^i$. Now, we can write the algorithm

```
BMM-Witness ($\mathbf{A}, \mathbf{B}$)
1.  $\mathbf{W} \leftarrow -\mathbf{AB}$
2.  repeat $2ec \log n$ times
3.      $R \leftarrow \{1, \ldots, n\}$
4.      repeat $\lceil \log n \rceil$ times
5.          compute $\mathbf{A}^R$: $a_{ik}^R \leftarrow [k \in R] k a_{ik}$
6.          $\mathbf{Z} \leftarrow \mathbf{A}^R \mathbf{B}$
7.          for all $i, j$
8.              if $w_{ij} < 0$ and $z_{ij}$ is a witness
9.                  then $w_{ij} \leftarrow z_{ij}$
10.         $R \leftarrow$(1/2)-sample from $R$
11. for all $i, j$
12.     if $w_{ij} < 0$
13.         then find witness for $i, j$ by brute force
```

The minus sign for $\mathbf{W}$ (set in line 1) is just for book-keeping purposes – a negative entry means that we still need to find a witness for that entry (hence the check in lines 8 and 12). In line 5, $[k \in R]$ is an indicator (note that it is 1 with probability $1/2^i$ in the i-th iteration). In line 8, checking whether z_{ij} is a witness is done simply by checking that $a_{iz_{ij}} = b_{z_{ij}j} = 1$. In line 13, all the entries a_{ik}, b_{kj} need to be checked and so each check needs time $O(n)$. The result is correct because in lines 9 and 13, we set a witness only if we have directly checked it.

Running Time. For every i, j, the loop of lines 4-10 guarantees that we try a probability that is close to $1/w_{ij}$ as it is needed, and this is tried $2ec \log n$ times, so the probability of failing to find a witness for i, j in that loop is at most $1/n^c$. Since there are n^2 pairs i, j, the expected number of failed pairs is at most $1/n^{c-2}$. With $c = 1$, this is at most n and then the expected time of the loop 11-13 is at most $O(n^2)$. In the loop 4-10, the running time is dominated by the matrix multiplication in line 6. So the expected total running time is $O(M(n) \log^2 n)$.

VI.5 Successors for Shortest Paths

We now use boolean multiplication witnesses to obtain a successor matrix $\mathbf{S}$ for the APSP problem. Consider a pair i, j. Clearly, s is a successor of i on a shortest path from i to j iff $d_{sj} = d_{ij} - 1$ ((recall $\mathbf{D} = [d_{ij}]$ is the matrix of shortest distances). Suppose we define a boolean matrix $\mathbf{F}$ by $f_{sj} = 1$ if $d_{sj} = d_{ij} - 1$. Then a successor for i, j could be found as the i, j entry in a witness matrix for the boolean product $\mathbf{AF}$. This is not useful though because since the setting of d_{sj} depends on i, it would mean that we need to try all the n possibilities.

Fortunately, observation 3(i) comes to our rescue: we only need to distinguish distances module 3 ! More precisley, we would like to define $\mathbf{F}$ by $f_{sj} = 1$ if $d_{sj} = d_{ij} - 1 \bmod 3$. Note that though this definition still depends on i, only 3 different cases are possible. So, it is sufficient to define 3 different matrices $\mathbf{F}^{(c)}$, $c = 0, 1, 2$, by

$$f_{sj}^{(c)} = 1 \text{ if } d_{sj} = c - 1 \bmod 3.$$

The algorithm outline is as follows:

```
MM-APSP ($\mathbf{A}$)
1.  $\mathbf{D} \leftarrow$ MM-APSD($\mathbf{A}$)
2.  for $c = 0, 1, 2$
3.      compute $\mathbf{F}^{(c)}$: $f_{sj}^{(c)} = 1$ iff $d_{sj} = c - 1 \bmod 3$
4.      $\mathbf{W}^{(c)} = \text{BMM-W}\textsc{itness}(\mathbf{A}, \mathbf{F}^{(c)})$
5.  compute $\mathbf{S}$: $s_{ij} = w_{ij}^{(d_{ij} \bmod 3)}$
6.  return $\mathbf{S}$
```

The expected total running time is $O(M(n) \log^2 n)$.