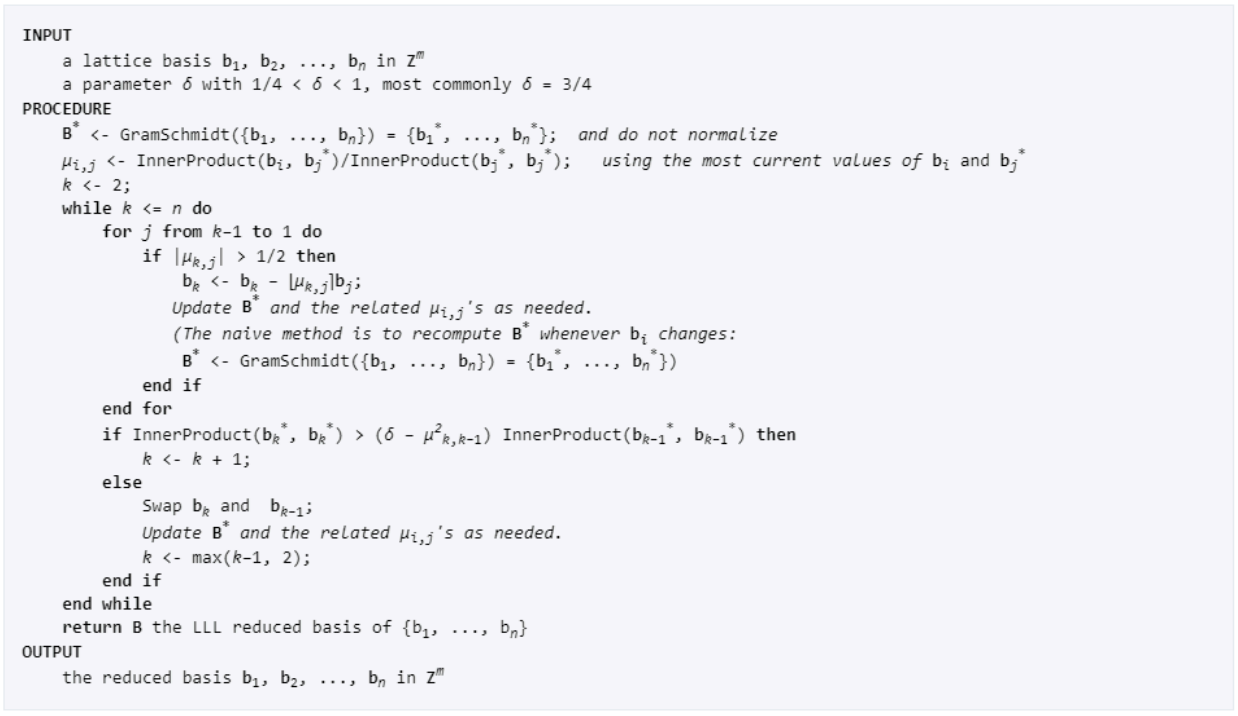
Word Count: 691
Discussion of Solution Design and Choices Made:

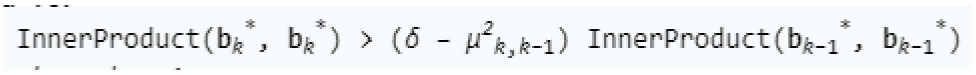
For my solution, I used the basic Schnorr–Euchner enumeration method. However, in order to increase the efficiency and speed of my solution, I also implemented the LLL lattice basis reduction algorithm. This made it so that my Schnorr-Euchner enumeration had a simpler basis as an input and was quicker while calculating the shortest vector.



```
INPUT
    a lattice basis $b_1, b_2, ..., b_n$ in $Z^m$
    a parameter $\delta$ with $1/4 < \delta < 1$, most commonly $\delta = 3/4$
PROCEDURE
    $B^*$ <- GramSchmidt({$b_1, ..., b_n$}) = {$b_1^*, ..., b_n^*$};  and do not normalize
    $\mu_{i,j}$ <- InnerProduct($b_i, b_j^*$)/InnerProduct($b_j^*, b_j^*$);   using the most current values of $b_i$ and $b_j^*$
    $k$ <- 2;
    while $k <= n$ do
        for $j$ from $k$-1 to 1 do
            if $|\mu_{k,j}| > 1/2$ then
                $b_k$ <- $b_k - \lfloor\mu_{k,j}\rceil b_j$;
               Update $B^*$ and the related $\mu_{i,j}$'s as needed.
               (The naive method is to recompute $B^*$ whenever $b_i$ changes:
                $B^*$ <- GramSchmidt({$b_1, ..., b_n$}) = {$b_1^*, ..., b_n^*$})
            end if
        end for
        if InnerProduct($b_k^*, b_k^*$) > ($\delta - \mu^2_{k,k-1}$) InnerProduct($b_{k-1}^*, b_{k-1}^*$) then
            $k$ <- $k$ + 1;
        else
            Swap $b_k$ and $b_{k-1}$;
            Update $B^*$ and the related $\mu_{i,j}$'s as needed.
            $k$ <- max($k$-1, 2);
        end if
    end while
    return B the LLL reduced basis of {$b_1, ..., b_n$}
OUTPUT
    the reduced basis $b_1, b_2, ..., b_n$ in $Z^m$
```

Figure 1: LLL Algorithm Pseudocode

"LLL" is an algorithm that calculates the "LLL-reduced" (almost orthogonal) version of a given basis $B = \{b_1, b_2, b_3, ..., b_n\}$ of a lattice L. It uses the orthogonal basis (calculated using the Gram-Schmidt process), the Gram-Schmidt coefficient matrix μ $= \frac{<b_i, b^*_j>}{<b^*_j, b^*_j>}$ and a special δ parameter, which is used to check if the Lovász condition is satisfied. δ is most commonly assumed to be 0.75.



InnerProduct(b_k^*, b_k^*) > ($\delta - \mu^2_{k,k-1}$) InnerProduct(b_{k-1}^*, b_{k-1}^*)

Figure 2: The Lovász Condition

The LLL algorithm is widely considered to be one of the most efficient ways of calculating shortened basis vectors. It runs in $O(n^5 * a * log^3 B)$ where n is the number of basis vectors b, a is the lattice definition $L = R^a$ and B is the length of the largest basis vector b.

The usage of LLL makes it so that my program is very efficient especially in lattices that are smaller or equal to 10x10 as running the LLL algorithm alone will provide a basis that is shortened enough that the shortest vector has either been found or will be found very quickly by using the Schnorr-Euchner enumeration algorithm.

The Schnorr-Euchner enumeration algorithm calculates the shortest vector in a lattice L with basis $B = \{b_1, b_2, b_3, ..., b_n\}$. It uses the orthogonal basis B^* from the Gram-Schmidt process in order to calculate the $\mu = \frac{<b_i, b^*_j>}{<b^*_j, b^*_j>}$ matrix. Different from LLL, however, it also computes an array of the length of each vector in B^*. It also requires the radius of the lattice as its second parameter.

Algorithm: The basic Schnorr–Euchner enumeration Schnorr and Euchner (1994)

```
Input: A basis $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ of a lattice $L$ and a radius $R$ with $\lambda_1(L) \le R$
Output: The shortest non-zero vector $\mathbf{s} = \sum_{i=1}^{n} v_i \mathbf{b}_i$ in $L$
 1: Compute Gram–Schmidt information $\mu_{i,j}$ and $\|\mathbf{b}_i^*\|^2$ of $\mathbf{B}$
 2: $(\rho_1, \ldots, \rho_{n+1}) = \mathbf{0}$, $(v_1, \ldots, v_n) = (1, 0, \ldots, 0)$, $(c_1, \ldots, c_n) = \mathbf{0}$, $(w_1, \ldots, w_n) = \mathbf{0}$
 3: $k = 1$, last_nonzero $= 1$ // largest $i$ for which $v_i \ne 0$
 4: while true do
 5:   $\rho_k \leftarrow \rho_{k+1} + (v_k - c_k)^2 \cdot \|\mathbf{b}_k^*\|^2$ // $\rho_k = \|\pi_k(\mathbf{s})\|^2$
 6:   if $\rho_k \le R^2$ then
 7:     if $k = 1$ then $R^2 \leftarrow \rho_k$, $\mathbf{s} \leftarrow \sum_{i=1}^{n} v_i \mathbf{b}_i$; // update the squared radius
 8:     else $k \leftarrow k - 1$, $c_k \leftarrow -\sum_{i=k+1}^{n} \mu_{i,k} v_i$, $v_k \leftarrow \lfloor c_k \rceil$, $w_k \leftarrow 1$;
 9:   else
10:     $k \leftarrow k + 1$ // going up the tree
11:     if $k = n + 1$ then return $\mathbf{s}$;
12:     if $k \ge$ last_nonzero then last_nonzero $\leftarrow k$, $v_k \leftarrow v_k + 1$;
13:     else
14:       if $v_k > c_k$ then $v_k \leftarrow v_k - w_k$; else $v_k \leftarrow v_k + w_k$; // zig-zag search
15:       $w_k \leftarrow w_k + 1$
16:     end if
17:   end if
18: end while
```

Figure 3: The Basic Schnorr-Euchner Enumeration Algorithm

Similar to how the LLL method checks if the Lovász condition has been satisfied, the Schnorr-Euchner enumeration method checks to see if a condition holds true. Namely, whether radius squared is bigger than a calculated number ρ. Given that this condition holds true, the code calculates s, the shortest current vector by using a sigma statement $s = \sum_{i=1}^{n} v_i b_i$. The algorithm then adjusts the number k, used to parse the basis vectors, according to the $\rho_k < R^2$ statement. When all of the basis vectors have been parsed, the algorithm returns the shortest vector s.

$$\rho_k \leftarrow \rho_{k+1} + (v_k - c_k)^2 \cdot \|\mathbf{b}_k^*\|^2$$

Figure 4: The Schnorr-Euchner Condition

The Schnorr-Euchner algorithm is known as one of the most efficient algorithms when it comes to solving SVP. However, it is true that since its creation in 1994, exactly 30 years ago, there have been many improvements and specialisations to the method, with many similar but more refined options popping up. It also does stand, however, that the basic version of the algorithm is still quite successful, especially with bases that do not have too many dimensions. Thus, it will still prove useful to us given the task we will be utilising it for.

Analysis of Performance:

Time data:

Dimensions	**Time** (in μs)
10	30580
9	15428
8	10679
7	6999
6	4785
5	3054
4	2158
3	1968
2	1984

Figure 5: Time Data Table

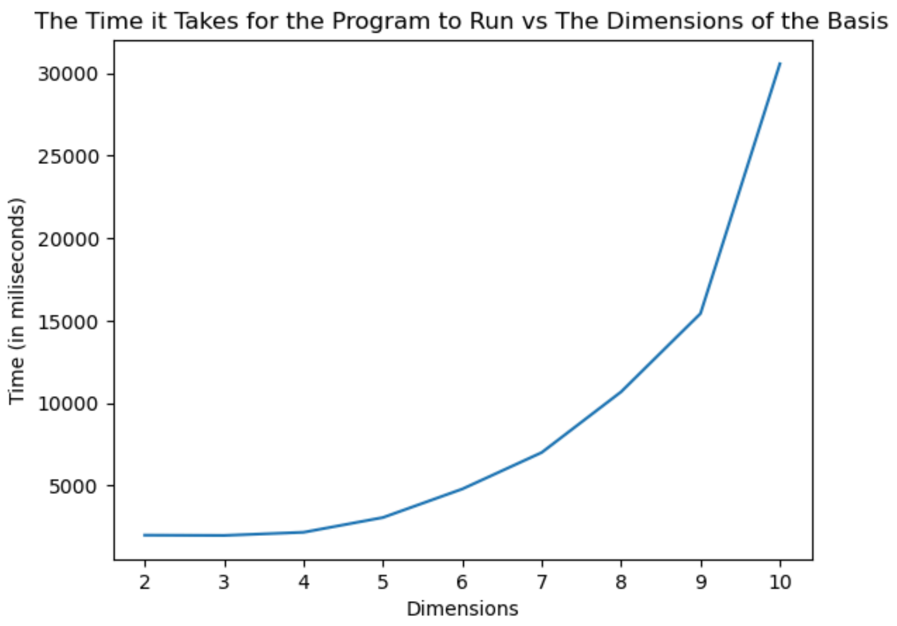


Figure 6: Time Data Graph

The data and graph above depicts the amount of time it took for the program to run for each dimension of basis up to 10 dimensions. It is apparent that the time starts increasing exponentially as dimensions start going over 8, 9, 10 and so on. Since the program will be tested predominantly with 10x10 and smaller bases, this will not be a big issue. However, it is important to note that for bigger cases the program might need more optimising.

Memory data:

Dimensions	**Bytes**
10	**8.321.184**
9	**4.074.664**
8	**1.363.400**
7	**1.175.856**
6	**630.536**
5	**233.696**
4	**137.704**
3	**95.976**
2	**86.696**

Figure 7: Memory Data Table

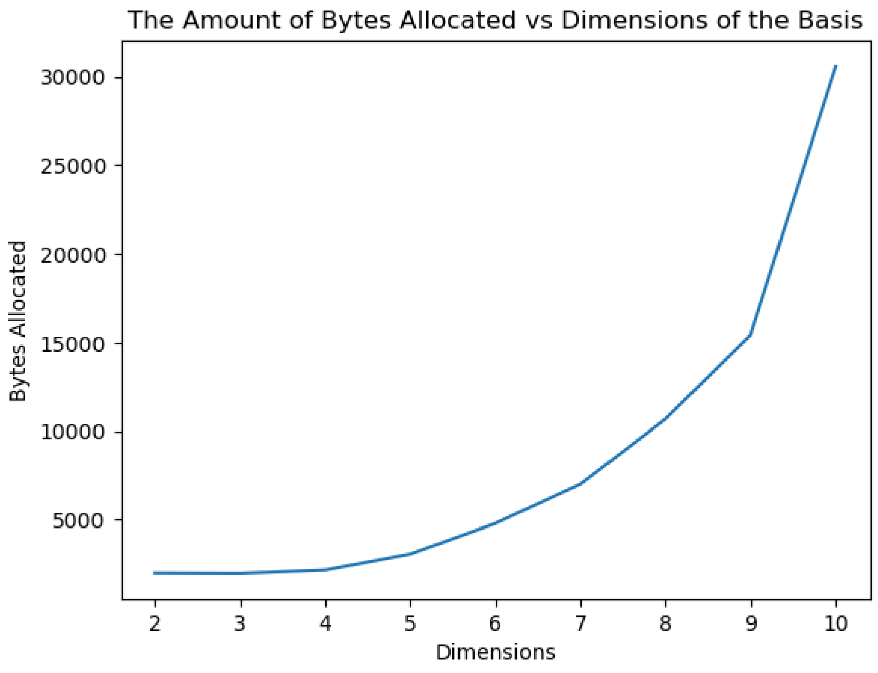


Figure 8: Memory Data Graph

The data and graph above depict the amount of bytes allocated for the program. It's obvious to see a trend that is very similar to the speed data; namely that the allocated bytes start increasing dramatically as dimensions increase. Here, using the LLL reduction algorithm adds to the memory usage, but also reduces the workload of the main enumeration algorithm as the LLL-reduced matrix is easier to enumerate and also has the payoff of higher speed. However, it is also important to note that manual memory allocation using malloc and free could be used to achieve a more optimised result.

Article References:

Correia, F., Mariano, A., Proença, A., Bischof, C., & Agrell, E. (2016). Parallel improved Schnorr-Euchner enumeration SE++ for the CVP and SVP. https://ieeexplore.ieee.org/document/7445396/

Hoffstein, J., Pipher, J., & Silverman, J. H. (2008). *An introduction to mathematical cryptography*. Springer.

Lenstra, A. K., Lenstra, H. W., & Lovász, L. (1982). *Factoring polynomials with rational coefficients*. Mathematisch Annalen.

Micciancio, D. (2012). CSE206A :Lattice algorithms and applications. https://cseweb.ucsd.edu/classes/wi12/cse206A-a/LecEnum.pdf

Micciancio, D. (n.d.). *Lattice cryptography*. Enum. https://cseweb.ucsd.edu/~daniele/LatticeLinks/Enum.html

Takagi, T., Wakayama, M., Tanaka, K., Kunihiro, N., Kimoto, K., & Ikematsu, Y. (2021). *International Symposium on Mathematics, quantum theory, and cryptography proceedings of MQC 2019*. Springer Singapore.

Code References:

Dagan, O. (2015, June 11). *If conditions in a makefile, inside a target*. Stack Overflow. https://stackoverflow.com/questions/15977796/if-conditions-in-a-makefile-inside-a-target

GeeksforGeeks. (2020, February 14). *Vector of vectors in C++ STL with examples*. GeeksforGeeks. https://www.geeksforgeeks.org/vector-of-vectors-in-c-stl-with-examples/

GeeksforGeeks. (2023, May 3). *Measure execution time of a function in C++*. GeeksforGeeks. https://www.geeksforgeeks.org/measure-execution-time-function-cpp/

GeeksforGeeks. (2024, January 8). *Exception handling in C++*. GeeksforGeeks. https://www.geeksforgeeks.org/exception-handling-c/

leemes, & Petzke, K. (2021, October 13). *How to remove all the occurrences of a char in C++ string*. Stack Overflow. https://stackoverflow.com/questions/20326356/how-to-remove-all-the-occurrences-of-a-char-in-c-string ronag. (2012, October 28). *Initialize a vector to Zeros C++/C++11*. Stack Overflow. https://stackoverflow.com/questions/13110130/initialize-a-vector-to-zeros-c-c11

TechWithTim. (2021, May 9). *Learn C++ with me #18 - vectors*. YouTube. https://www.youtube.com/watch?v=RXzzE2wnnlo&ab_channel=TechWithTim

Valgrind Home. (n.d.). https://valgrind.org/