Multiprocessing

Dr. Raul Ramirez-Velarde

Student: Gerardo Mauricio Gutiérrez Quintana A00815174

Implement in the programing environment in which you have install MPI the following sequential algorithms:

1. **Gauss Reduction**

Gauss reduction is carried out in the following manner:

Normalization

Reduction

Normalization

Reduction

Which gives

The algorithm follows (matrix A holds the coefficients, vector b the rhs):

for (k = 1; k <= N; k++)

//Normalize pivoting row

for (j = k+1; j <= N; j++)

A[k,j] = A[k,j] / A[k,k]

y[k] = b[k] / A[k,k]

A[k,k] = 1

//Reduce rows below pivot

for (i = k+1; i <= N; i++)

//Reduce columns after pivot

for (j = k+1; j <= N; j++)

A[i,j] = A[i,j] - A[i,k] \* A[k,j]

b[i] = b[i] - A[i,k] \* y[k]

A[i,k] = 0

Test your code with the example provided

**Problem #1 demonstration**

**Code**

// Gerardo Mauricio Gutierrez Quintana

// Matricula A00815174

// Homework #1

// Gauss Reduction

#include <iostream>

using namespace std;

double A[3][3] = {{3,1,-9},{3,2,10},{2,1,7}};

double b[3] = {10,13,9};

int i, j, k;

int main(){

int N = 2;

for (k = 0; k < N; k++){

//Normalize pivoting row

for (j = k+1; j <= N; j++){

A[k][j] = A[k][j] / A[k][k];

}

b[k] = b[k] / A[k][k];

A[k][k] = 1;

//Reduce rows below pivot

for (i = k+1; i <= N; i++){

//Reduce columns after pivot

for (j = k+1; j <= N; j++){

A[i][j] = A[i][j] - A[i][k] \* A[k][j];

}

b[i] = b[i] - A[i][k] \* b[k];

A[i][k] = 0;

}

}

for (i = 0; i <= N; i++){

cout << b[i] << " " << endl;

}

return 0;

}



1. **Floyd’s Shortest Route Algorithm**

Floy’s algorithm is used to determine the shortest path for two pairs of nodes in a graph. Dijstra’s algorithm could be used, but since it creates a spanning tree from a single source, we would have to run it for every origin, that is N times. Floy’s algorithm is run only once and it will calculate al shortest paths.

The algorithm in Scilab follows:

C=x\_matrix("Direct path Matrix",[0 8 3 5 10000

8 0 2 10000 5

10000 1 0 3 4

6 10000 10000 0 7

10000 5 10000 10000 0])

N=size(C,1)

for k=1:N

for i=1:N

for j=1:N

C(i,j)=min(C(i,j),C(i,k)+C(k,j))

end

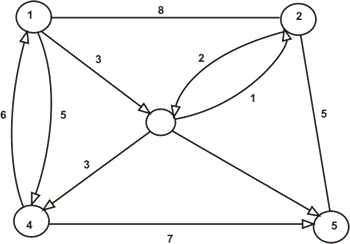
end

disp(C)

input("Press <enter>")

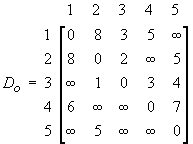
end

It solves the following example:

****

**Figure Z1. Transportation network for studying Floyd's algorithm.**

Starting matrix *Do* (or C) is as follows:

****

We now go to the first algorithmic step. Let k = 1. As an illustration of Step 2 we calculate the elements of the first three rows of matrix D1. Calculations for other rows are left as an exercise.

# 1

# Matrix D1 is as follows:

# 1

And we get the following result:

C =

0. 4. 3. 5. 7.

8. 0. 2. 5. 5.

9. 1. 0. 3. 4.

6. 10. 9. 0. 7.

13. 5. 7. 10. 0.

**Execution**

Comparing going through k=1

0. 8. 3. 5. 10000.

8. 0. 2. 13. 5.

10000. 1. 0. 3. 4.

6. 14. 9. 0. 7.

10000. 5. 10000. 10000. 0.

Press <enter>

Comparing going through k=2

0. 8. 3. 5. 13.

8. 0. 2. 13. 5.

9. 1. 0. 3. 4.

6. 14. 9. 0. 7.

13. 5. 7. 18. 0.

Press <enter>

Comparing going through k=3

0. 4. 3. 5. 7.

8. 0. 2. 5. 5.

9. 1. 0. 3. 4.

6. 10. 9. 0. 7.

13. 5. 7. 10. 0.

Press <enter>

Comparing going through k=4

0. 4. 3. 5. 7.

8. 0. 2. 5. 5.

9. 1. 0. 3. 4.

6. 10. 9. 0. 7.

13. 5. 7. 10. 0.

Press <enter>

Comparing going through k=5

0. 4. 3. 5. 7.

8. 0. 2. 5. 5.

9. 1. 0. 3. 4.

6. 10. 9. 0. 7.

13. 5. 7. 10. 0.

Press <enter>

In reality, this algorithm uses two matrices. The second matrix stablishes the immediate predecessor of node j on the shortest path leading from node i to node j is actually node i. See this reference:

<https://nptel.ac.in/courses/105104098/TransportationII/mod13/16slide.htm>

Modify the code to implement the algorithm using both matrices: matrix C for the shortest distance and matrix Q for the shortest path.

**Problem #2 demonstration**

**Code**

// Gerardo Mauricio Gutierrez Quintana

// Matricula A00815174

// Homework #1

// Floyd Algorithm

#include <iostream>

using namespace std;

#define INF 99999

#define N 5

int A[N][N] = {{0, 8, 3, 5, INF}, {8, 0, 2, 13, 5}, {INF, 1, 0, 3, 4}, {6, 14, 9, 0, 7}, {INF, 5, INF, INF, 0}};

void floyd(){

for (int k = 0; k < N; k++){

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

for (int j = 0; j < N; j++){

if (A[i][k] + A[k][j] < A[i][j]){

A[i][j] = A[i][k] + A[k][j];

}

}

}

}

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

for (int j = 0; j < N; j++){

if (A[i][j] < 10){

cout << A[i][j] << " ";

}

else

cout << A[i][j] << " ";

}

cout << endl;

}

}

int main(){

floyd();

return 0;

}



1. **Bucket and Bubble Sort**

**Bubble sort**

We take an unsorted array for our example. Bubble sort takes Ο(n2) time so we're keeping it short and precise.

Bubble Sort

Bubble sort starts with very first two elements, comparing them to check which one is greater.

Bubble Sort

In this case, value 33 is greater than 14, so it is already in sorted locations. Next, we compare 33 with 27.

Bubble Sort

We find that 27 is smaller than 33 and these two values must be swapped.

Bubble Sort

The new array should look like this –

Bubble Sort

Next we compare 33 and 35. We find that both are in already sorted positions.

Bubble Sort

Then we move to the next two values, 35 and 10.

Bubble Sort

We know then that 10 is smaller 35. Hence they are not sorted.

Bubble Sort

We swap these values. We find that we have reached the end of the array. After one iteration, the array should look like this –

Bubble Sort

To be precise, we are now showing how an array should look like after each iteration. After the second iteration, it should look like this –

Bubble Sort

Notice that after each iteration, at least one value moves at the end.

Bubble Sort

And when there's no swap required, bubble sorts learns that an array is completely sorted.

Bubble Sort

Now we should look into some practical aspects of bubble sort.

## Pseudocode

We observe in algorithm that Bubble Sort compares each pair of array element unless the whole array is completely sorted in an ascending order. This may cause a few complexity issues like what if the array needs no more swapping as all the elements are already ascending.

To ease-out the issue, we use one flag variable **swapped** which will help us see if any swap has happened or not. If no swap has occurred, i.e. the array requires no more processing to be sorted, it will come out of the loop.

Pseudocode of BubbleSort algorithm can be written as follows −

procedure bubbleSort( list : array of items )

loop = list.count;

for i = 0 to loop-1 do:

swapped = false

for j = 0 to loop-1 do:

/\* compare the adjacent elements \*/

if list[j] > list[j+1] then

/\* swap them \*/

swap( list[j], list[j+1] )

swapped = true

end if

end for

/\*if no number was swapped that means

array is sorted now, break the loop.\*/

if(not swapped) then

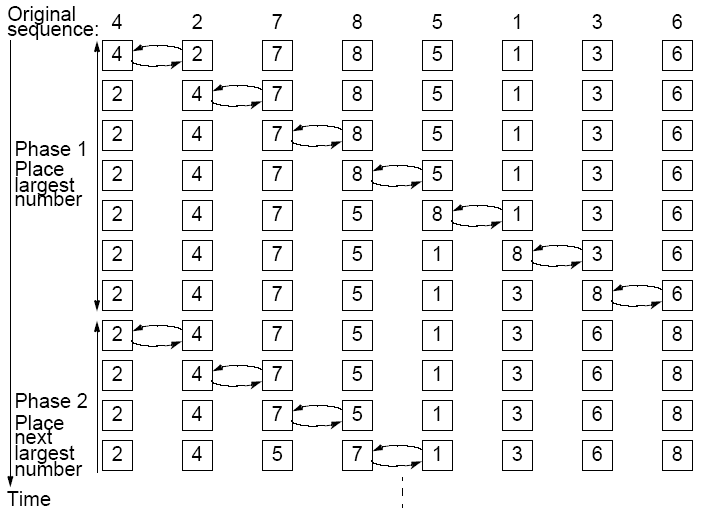
break

end if

end for

end procedure return list

This is another example:

****

**Bucket Sort**

Bucket Sort, also known as Bin Sort, is a distributed sorting algorithm, which sort elements from an array by performing these steps:

1. Distribute the elements into buckets or bins.
2. Sort each bucket individually.
3. Merge the buckets in order to produce a sorted array as the result.

Where **n** = the number of elements and **k** is the number of buckets.

In the best case, the algorithm distributes the elements uniformly between buckets, a few elements are placed on each bucket and sorting the buckets is **O(1)**. Rearranging the elements is one more run through the initial list.

In the worst case, the elements are sent all to the same bucket, making the process take **O(n2)**.

## Pseudocode

function bucketSort(array, n) is

buckets ← new array of n empty lists

for i = 0 to (length(array)-1) do

insert array[i] into buckets[msbits(array[i], k)]

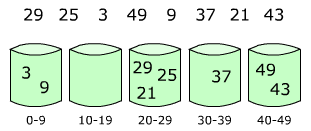
for i = 0 to n - 1 do

nextSort(buckets[i]);

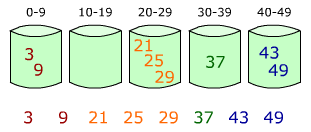
return the concatenation of buckets[0], ...., buckets[n-1]

## Graphically explained

1. Distribute elements in buckets:

[](https://camo.githubusercontent.com/10e73aca86e6abd85f32f63a270126a19f00414c/68747470733a2f2f75706c6f61642e77696b696d656469612e6f72672f77696b6970656469612f636f6d6d6f6e732f362f36312f4275636b65745f736f72745f312e706e67)

1. Sorting inside every bucket and merging:

[](https://camo.githubusercontent.com/19ff8f6edf48140cbf28fa9907b42bcb19fa14e7/68747470733a2f2f75706c6f61642e77696b696d656469612e6f72672f77696b6970656469612f636f6d6d6f6e732f332f33392f4275636b65745f736f72745f322e706e67)

## An example

### Input

Suppose we have the following list of elements: [2, 56, 4, 77, 26, 98, 55]. Let's use 10 buckets. To determine the capacity of each bucket we need to know the maximum element value, in this case 98.

So the buckets are:

* bucket 1: from 0 to 9
* bucket 2: from 10 to 19
* bucket 3: from 20 to 29
* and so on.

### Distribution

Now we need to choose a distribution function.

bucketNumber = (elementValue / totalNumberOfBuckets) + 1

Such that by applying that function we distribute all the elements in the buckets.

In our example it is like the following:

1. Apply the distribution function to 2. bucketNumber = (2 / 10) + 1 = 1
2. Apply the distribution function to 56. bucketNumber = (56 / 10) + 1 = 6
3. Apply the distribution function to 4. bucketNumber = (4 / 10) + 1 = 1
4. Apply the distribution function to 77. bucketNumber = (77 / 10) + 1 = 8
5. Apply the distribution function to 26. bucketNumber = (26 / 10) + 1 = 3
6. Apply the distribution function to 98. bucketNumber = (98 / 10) + 1 = 10
7. Apply the distribution function to 55. bucketNumber = (55 / 10) + 1 = 6

Our buckets will be filled now:

**1** [](https://camo.githubusercontent.com/a419b8438fd7c693c39b20ed2eae0d893bd5a08b/68747470733a2f2f706978616261792e636f6d2f7374617469632f75706c6f6164732f70686f746f2f323031342f30332f32342f31372f32312f7061696c2d3239353439315f3936305f3732302e706e67): [2, 4]  
**2** [](https://camo.githubusercontent.com/a419b8438fd7c693c39b20ed2eae0d893bd5a08b/68747470733a2f2f706978616261792e636f6d2f7374617469632f75706c6f6164732f70686f746f2f323031342f30332f32342f31372f32312f7061696c2d3239353439315f3936305f3732302e706e67): []  
**3** [](https://camo.githubusercontent.com/a419b8438fd7c693c39b20ed2eae0d893bd5a08b/68747470733a2f2f706978616261792e636f6d2f7374617469632f75706c6f6164732f70686f746f2f323031342f30332f32342f31372f32312f7061696c2d3239353439315f3936305f3732302e706e67): [26]  
**4** [](https://camo.githubusercontent.com/a419b8438fd7c693c39b20ed2eae0d893bd5a08b/68747470733a2f2f706978616261792e636f6d2f7374617469632f75706c6f6164732f70686f746f2f323031342f30332f32342f31372f32312f7061696c2d3239353439315f3936305f3732302e706e67): []  
**5** [](https://camo.githubusercontent.com/a419b8438fd7c693c39b20ed2eae0d893bd5a08b/68747470733a2f2f706978616261792e636f6d2f7374617469632f75706c6f6164732f70686f746f2f323031342f30332f32342f31372f32312f7061696c2d3239353439315f3936305f3732302e706e67): []  
**6** [](https://camo.githubusercontent.com/a419b8438fd7c693c39b20ed2eae0d893bd5a08b/68747470733a2f2f706978616261792e636f6d2f7374617469632f75706c6f6164732f70686f746f2f323031342f30332f32342f31372f32312f7061696c2d3239353439315f3936305f3732302e706e67): [55, 56]  
**7** [](https://camo.githubusercontent.com/a419b8438fd7c693c39b20ed2eae0d893bd5a08b/68747470733a2f2f706978616261792e636f6d2f7374617469632f75706c6f6164732f70686f746f2f323031342f30332f32342f31372f32312f7061696c2d3239353439315f3936305f3732302e706e67): []  
**8** [](https://camo.githubusercontent.com/a419b8438fd7c693c39b20ed2eae0d893bd5a08b/68747470733a2f2f706978616261792e636f6d2f7374617469632f75706c6f6164732f70686f746f2f323031342f30332f32342f31372f32312f7061696c2d3239353439315f3936305f3732302e706e67): [77]  
**9** [](https://camo.githubusercontent.com/a419b8438fd7c693c39b20ed2eae0d893bd5a08b/68747470733a2f2f706978616261792e636f6d2f7374617469632f75706c6f6164732f70686f746f2f323031342f30332f32342f31372f32312f7061696c2d3239353439315f3936305f3732302e706e67): []  
**10** [](https://camo.githubusercontent.com/a419b8438fd7c693c39b20ed2eae0d893bd5a08b/68747470733a2f2f706978616261792e636f6d2f7374617469632f75706c6f6164732f70686f746f2f323031342f30332f32342f31372f32312f7061696c2d3239353439315f3936305f3732302e706e67): [98]

We can choose to insert the elements in every bucket in order, or sort every bucket after distributing all the elements.

### Put the elements back in the list

Finally we go through all the buckets and put the elements back in the list:

[2, 4, 26, 55, 56, 77, 98]

**Bubble sort demonstration**

**Code**

// Gerardo Mauricio Gutierrez Quintana

// Matricula A00815174

// Homework #1

// Bubble sort

#include <iostream>

using namespace std;

#define N 5

int arr[N] = {14, 33, 27, 35, 10};

void bubbleSort(){

bool swapped = true;

int j = 0;

int tmp;

while (swapped){

swapped = false;

j++;

for (int i = 0; i < N - j; i++){

if (arr[i] > arr[i + 1]) {

tmp = arr[i];

arr[i] = arr[i + 1];

arr[i + 1] = tmp;

swapped = true;

}

}

}

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

cout << arr[i] << " ";

}

cout << endl;

}

int main(){

bubbleSort();

return 0;

}



**Bucket sort demonstration**

**Code**

// Gerardo Mauricio Gutierrez Quintana

// Matricula A00815174

// Homework #1

// Bucket sort

#include <iostream>

#include <algorithm>

#include <vector>

#include <cmath>

using namespace std;

int getMax(int arr[], int N) {

int i, m = arr[0];

for(i = 1; i < N; i++) {

if(arr[i] > m) {

m = arr[i];

}

}

return m;

}

void bucketSort(int arr[], int N){

int bucket = 10;

vector<int> b[bucket];

int max = getMax(arr, N);

int divider = ceil(float(max + 1) / bucket);

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

int bi = floor( arr[i] / divider );

b[bi].push\_back(arr[i]);

}

for (int i = 0; i < bucket; i++)

sort(b[i].begin(), b[i].end());

int index = 0;

for (int i = 0; i < bucket; i++)

for (int j = 0; j < b[i].size(); j++)

arr[index++] = b[i][j];

}

int main(){

int N = 7;

int arr[] = {29, 25, 3, 49, 9, 37, 21};

bucketSort(arr, N);

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

cout << arr[i] << " ";

cout << endl;

return 0;

}



1. **Search Algorithm**

Search algorithm is quite simple:

int present = false;

for (i = 1; !present && i <= N; i++)

if ( A [i] == x) present = true;

**Problem #4 demonstration**

**Code**

// Gerardo Mauricio Gutierrez Quintana

// Matricula A00815174

// Homework #1

// Search

#include <iostream>

using namespace std;

int main(){

int N = 5;

int A[5] = {2, 4, 6, 8, 10};

int x = 10;

int present = false;

for (int i = 1; !present && i <= N; i++)

if ( A [i] == x){

present = true;

cout << "It is present" << endl;

}

return 0;

}

