ITCS-6114 Project
Kiran Gaitonde
800819341
kgaitond@uncc.edu

File Server Allocation

Data representations

- File bandwidths are considered to be integer values
- Servers are represented a using 1D Array or ArrayList (Servers), where each index of an Array/ArrayList represents a server. Length of the Array/ArrayList represents number of servers.
- Bandwidth requirements are stored in a 1D array (Bandwidths). Each index represents bandwidth requirement of a file. The length of the array is the number of files.

Algorithms

Brute Force:

- Generate all possible permutations of the Bandwidths (assuming all bandwidths are distinct).
- Then for each permutation of Bandwidths, arrange the bandwidths on Servers in all possible ways.
- At each stage store the maxServerBandwidth only if it is optimized (lesser than the previously stored max server bandwidth).
- After all permutations, maxServerBandwidth will have the optimum value.

Each permutation is arranged on servers as follows:
Suppose we have 3 servers (s1, s2, s3) and 5 files (f1, f2, f3, f4, f5) and we get a permutation (f2, f4, f1, f5, f3) the below arrangements are tried for that permutation

	s1	s2	s3
1	f2, f4, f1, f5, f3	0	0
2	f2, f4, f1, f5	f3	0
3	f2,f4,f1	f5	f3
4	f2,f4	f1,f3	f5
5	f2	f4,f5	f1,f3

Starting with all the files on server 1, go on reducing the files, arrange the remaining files on remaining servers in round robin order.

Algorithm:

```
opMaxServerBw  = 0; // optimal max server bandwidth

s[];  // servers
bw[]; // bandwidths

generateAllPermutations(s[]);

arrangeFilesForEachPermutation();
 {
        mBw = getMaxBandwidthForEachArrangement
        if(opMaxServerBw==0)
           set mBw;
       else
           if(mB < opMaxServerBw)
              opMaxServerBw = mB;
}
```

Time Complexity:

Generating all permutations takes O(n^2).
There will be n! Permutations.
Each permutation will be arranged in n ways
So total complexity will be O(n*n! + n^2) for sufficient large n it will be O(n*n!)

Space Complexity:

I'm using recursion method for generating the permutations. This will create recursion tree of height n. Which will have a space complexity of O(n).
Apart for that the arranging is done in place, which will take the space equivalent to the length of the arrays.

Dynamic Programming:

- I have implemented the black box and decision version as mentioned in the assignment.
- Set initial value for X (the initial value I use is sum of all bandwidths divided by the number of servers)
- Check if solution is possible for X, if the solution is possible then reduce the value of X and check if solution is possible for the reduced X. Repeat until optimal X is found.
- If solution is not possible for initial X, then increase the value of X and check if solution is possible. Repeat until we get an X which has a solution.
- To check if solution is possible try to arrange the files on servers so that they will not cross the max capacity bandwidth for each server. If all the files can be arranged satisfying the condition then the solution is possible, otherwise there is no solution for given capacity.
- Before checking if solution is possible, arrange the files in descending order of bandwidths, which apparently will help in checking correctly.

Algorithm:

```
s[];  // servers
bw[]; // bandwidths

Dynamic
    Initialize X
    If solutionPossible(X)
        X - -
        Check is solutionPossible(X)
        repeat till no solution for X
        return X + +
    else
        X + +
        Check is solutionPossible(X)
        repeat till solution possible for particular X
        return X

solutionPossible
    sortDescending(bw);
    arrangeFileWithGivenCapacity
    if
        all files arranged
        return true
    else
        return false
```

- Using dynamic programming we make sure that each server satisfies the capacity constraints.
- So the optimal substructure would be to find optimal solution for each server and extend it to all servers.
- The dynamic programming formula for each server using suffixes can be written as follows :

  DPforFSA (i,X, n) = max (DPforFSA (i+1, X, n)
  (DPforFSA(i+1, X-BW[i], n-1) if X> BW[i]

Time Complexity:

Checking if a solution is present can have a worst complexity of O(n*m).
Number of times we check if solution is possible depends on how far is the initial X is set from the optimal X. Suppose we check for solution k times then the total complexity will be O(k*n*m)

Space Complexity:

Space complexity will be equivalent to the size of the arrays.

Heuristic:

I have implemented two approaches for this.

Approach1:

- In this approach each file is assigned to the server having the least sum of bandwidths of already assigned files.
- Before assigning the files to the server I arrange the files in descending order of their bandwidth requirements. This apparently leads to more optimal solution.

Algorithm:

```
s[];  // servers
bw[]; // bandwidths

sortDescending(bw)

for  i = 1 to n
    getServerWithMinBW;
    assignFileToServer;
```

Time Complexity:

Sorting the array in descending order will have a complexity of O(nlogn).

After adding each file we sort the server array to get min value, complexity for that will be O(n*mlogm), where m is the no of servers.

Total complexity O(n*mlogm +nlogn)

Space Complexity:

If sorting is done using merge sort then each sort will have a space complexity of O(n)

Assigning of files is done in place which will take space according to the size of the array.

Approach2:

- This approach works best when all the files have distinct bandwidths. This approach distributes the files evenly on all the servers.
- First sort the files in descending order. Then arrange the files in Zig Zag order, i.e. If the servers are numbered s1,s2…sm, then first arrange the files from s1 to sm, then arrange from sm to s1, repeat until all the files are assigned.

Algorithm:

```
s[];  // servers
bw[]; // bandwidths

sortDescending(bw)
for i = 1 to n
  assignFilesZigZag
```

Time Complexity:

Sorting the array in descending order will have a complexity of O(nlogn).

Assigning files will have complexity O(m).

Total complexity O(nlogn +m)

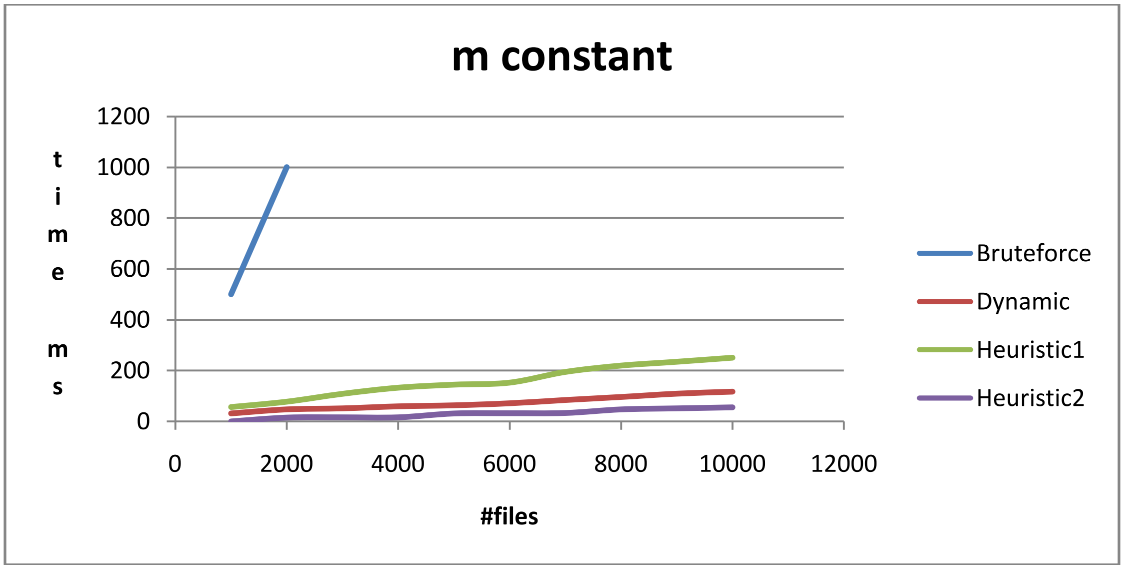
Space complexity:

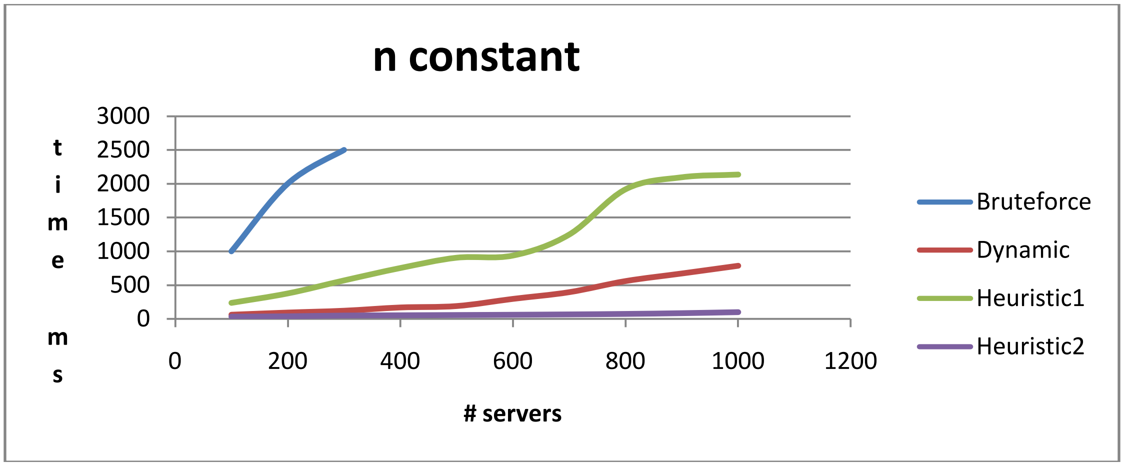
Sorting using merge sort will have a space complexity of O(n)
Assigning of files is done in place which will take space according to the size of the array.

Evaluations

- I have used my own instances for testing.
- Brute force and dynamic programming returned optimal solutions for all the instances used, but due to use of recurrences brute force takes huge amount of time compared to dynamic programming.
- Heuristic first approach returned optimal value for almost all of the instances, but due to sorting after each allocation the time taken increases with the increase in number of servers.
- Heuristic second approach, is the fastest of all the algorithms implemented, but it returns optimal solution only if all the bandwidth requirements are distinct.
- Most of the algorithm runtimes were as expected. I think the performance of the brute force can be improved by a better algorithm.

Below are the time plots for various instances used:





References

- Introduction to Algorithms 3rd edition, by Cormen, Leiserson, Rivest & Stein
- **Code References:** https://docs.oracle.com/javase/8/

Code Listing

Brute Force approach:

```
public class BruteForce {

    private static int noOfServers;
    private static int opMaxServerBw = 0;

// arrange files on server for each permutation in all possible ways
    private static void arrangeFilesForEachPermutation(int[] b){
        int x = b.length;
        while(x>0){
            int [] servers = new int [noOfServers];
            for (int j = 0; j < x; j++){
                servers[0] = servers[0]+b[j];
            }

            int[] bwSuff = new int [b.length-x];
            int l=0;
            for (int k = x; k < b.length; k++){
                bwSuff[l] = b[k];
                l++;
            }
            int[] remServers = arrangeFilesRndRbnForRemServers(bwSuff);
            int p=0;
            for (int j = 1; j < noOfServers; j++){
                servers[j]=remServers[p];
                p++;
            }

            Arrays.sort(servers);
            int mB = servers[servers.length-1];
            if(opMaxServerBw==0) {
                opMaxServerBw = mB;
            }
            else{
                if(mB < opMaxServerBw){
                    opMaxServerBw = mB;
                }
            }
            x--;
        }

    }

//arrange files on remaining servers in round robin order, and return server array
        private static int[] arrangeFilesRndRbnForRemServers(int[] b){

            int m = noOfServers-1;
            int [] remservers = new int [m];
            int n = b.length;

            if(n>m){
                int x = n % m;
                for (int i = 0; i < m; i++){
                    remservers[i] = remservers[i]+b[i];

                }
                for (int k = m ; k < n-x; k=k+m){
                    for (int j = 0; j < m; j++){
                        remservers[j] = remservers[j]+b[k+j];

                    }
                }
                int l = 0;
                for (int i = n-x; i < n; i++){
                    remservers[l] = remservers[l]+b[i];
                    l++;
                }
            }
            else{
                int l = 0;
                for (int i = 0; i < n; i++){
                    remservers[l] = remservers[l]+b[i];

                    l++;
                }
            }
            return remservers;
        }



/*generate all possible permutations of bandwidths;
 then arrange each permutation on servers in all possible ways.
 reference:this function is inspired by algorithm for permutations of string
 which can found commonly*/

    private static void tryAllPermutations(int[] bwPreList, int[] bwList) {
      int listLen = bwList.length;
      if (bwList.length == 0) {
        arrangeFilesForEachPermutation(bwPreList);
      }
      else {
          for (int i = 0; i < listLen; i++)
            tryAllPermutations(concatArrayInteger(bwPreList, bwList[i]),
concatArrays(getSubArray(bwList,0,i),getSubArray(bwList,i+1,listLen)));
      }
    }

//concatenates an array of integers and an integer
    private static int[] concatArrayInteger (int[] a , int b){
        int [] c = new int[a.length+1];
        for (int i = 0; i < a.length; i++){
            c[i]=a[i];
        }
        c[a.length]=b;
        return c;
    }
```

```
//concatenates two arrays of integers
    private static int[] concatArrays (int[] a , int[] b){
        int [] c = new int[a.length+b.length];
        for (int i = 0; i < a.length; i++){
            c[i]=a[i];
        }
        int j=0;
        for (int i = a.length; i < (a.length+b.length); i++){
            c[i]=b[j];
            j++;
        }
        return c;

    }

//returns an sub array with given start and end index
    private static int[] getSubArray (int[] a, int start, int end){
        int [] sub = new int[end-start];
        int j=0;
        for (int i = start; i < end; i++) {
            sub[j]=a[i];
            j++;
        }
        return sub;
    }

//runs brute force algorithm
    public void runBruteForce(int[] b){
        int [] emp = new int[0];
        tryAllPermutations(emp,b);
    }

//main function for testing
    public static void main(String []args) {
        GetInput gi = new GetInput();
        gi.getInput();
        int n = gi.getNoOfFiles();
        noOfServers = gi.getNoOfServers();
        int[] bws = new int[n];
        bws = gi.readFile();
        BruteForce bf = new BruteForce();
        long startTime = System.currentTimeMillis();
        bf.runBruteForce(bws);
        long stopTime = System.currentTimeMillis();
        long executionTime = stopTime - startTime;
        System.out.println("Optimal Max Bandwidth : "+ opMaxServerBw);
        System.out.println("Time taken : "+ executionTime);
     }
}
```

Dynamic approach:

```
public class Dynamic {

//returns an average of bandwidths rounded to next integer
    private static int getAverageBWPerServer(int [] bws, int m){
        double x = 0.0;
        for (int i = 0; i < bws.length; i++){
             x = x + bws[i];
        }
        return((int) Math.ceil(x/m));
    }

//returns an array sorted in descending order
    private static int[] arrangeBWSDescending(int[] bws){
        int d[] = new int[bws.length];
        Arrays.sort(bws);
        int j = bws.length;
        for (int i = 0; i < bws.length; i++){
            d[j-1] = bws[i];
            j--;
        }
        return d;
    }

//checks if a solution is possible with given files, noOfServes and budget
    private static boolean isSolutionPossible(int [] bws, int m , int
budgetPerServer){
        int serversBWs [] = new int [m];

        for (int i = 0; i < bws.length; i++){
            int flag = 1;
            for (int j = 0; j < m; j++){
                if((budgetPerServer-serversBWs[j]) >= bws[i]){

                    serversBWs[j] = serversBWs[j] + bws[i];

                    break;
                }
                else{
                    flag = flag+1;
                }
            }
            if(flag>m){
                return false;
            }
        }
        return true;
    }


//runs Dynamic bottom up algorithm
    public int runDynamic(int [] bws, int m)
    {
        //set initial budget
        int budgetPerServer = getAverageBWPerServer(bws,m);
        System.out.println("inttial budget : "+ budgetPerServer);
        int d[] = arrangeBWSDescending(bws);
        /* if solution present for initial budget; then check if solution is
present
        for lower budget*/
        if(isSolutionPossible(d, m, budgetPerServer)){
            while(true){
                budgetPerServer = budgetPerServer-1;
                if (isSolutionPossible(d,m,budgetPerServer))
                {
                    continue;
                }
                else{
                    break;
                }
            }
            return budgetPerServer+1;
        }
        /* if solution not present for initial budget; then check if solution is
present
        for next higher budget*/
        else{
            while(true){
                budgetPerServer = budgetPerServer + 1;
                if (isSolutionPossible(d,m,budgetPerServer)){
                    break;
                }
                else{
                    continue;
                }
            }
            return budgetPerServer;
        }
    }

//main function for testing
    public static void main(String []args) {
        GetInput gi = new GetInput();
        gi.getInput();
        int n = gi.getNoOfFiles();
        int noOfServers = gi.getNoOfServers();
        int[] bws = new int[n];
        bws = gi.readFile();
        Dynamic dbu = new Dynamic();
        long startTime = System.currentTimeMillis();
        int maxOptBw =      dbu.runDynamic(bws,noOfServers);
        long stopTime = System.currentTimeMillis();
        long executionTime = stopTime - startTime;
        System.out.println("Optimal Max Bandwidth : "+ maxOptBw);
        System.out.println("Time taken : "+ executionTime);
     }
}
```

Heuristic approach:

```
public class Heuristic {

//returns array sorted in descending order
    private static int[] arrangeBWSDescending(int[] bws){
        int d[] = new int[bws.length];
        Arrays.sort(bws);
        int j = bws.length;
        for (int i = 0; i < bws.length; i++){
            d[j-1] = bws[i];
            j--;
        }
        return d;
    }

/*arrange files in following way : assign each file to a server having minimum
sum of bandwidths of files already assigned */
    private static int arrangeFilesMinBWFirst(int[] b, int m)
    {
        ArrayList<Integer> serverBw = new ArrayList<Integer>(m);
        int n = b.length;

        for (int i = 0; i < m; i++){
            serverBw.add(b[i]);
        }

        for (int i = m; i < n; i++){
            int x = b[i]+ Collections.min(serverBw);
         int index  = serverBw.indexOf(Collections.min(serverBw));
         serverBw.remove(index);
         serverBw.add(index,x);
        }
        return Collections.max(serverBw);

    }

/*arrange files in zig zag order, i.e, assuming servers are numbered as S1,S2...Sm;
first arrange files from S1 to Sm, then Sm to S1, repeat until all files are
assigned*/
    private static int arrangeFilesZigZag(int[] bws, int m){
        int serversBWs [] = new int [m];
        int k =0;
        int l =m-1;
        for (int i = 0; i < bws.length; i++){
            if(k < m){
                serversBWs[k] = serversBWs[k] + bws[i];
                if(k == m-1){
                    l = m-1;
                }
                k++;
                continue;
            }
            if(l>-1){
                serversBWs[l] = serversBWs[l] + bws[i];
                if(l == 0){
                    k = 0;
                }
                l--;
                continue;
            }
        }
        Arrays.sort(serversBWs);
        return serversBWs[m-1];
    }

//sorts the bandwidths in descending order and run's minimum server bandwidth first
approach
    public int runHeuristic1(int[] bws, int m){
        int d[] = new int[bws.length];
        d = arrangeBWSDescending(bws);
        return(arrangeFilesMinBWFirst(d,m));
    }

//sorts the bandwidths in descending order and run's zig zag appproach
    public int runHeuristic2(int[] bws, int m){
        int d[] = new int[bws.length];
        d = arrangeBWSDescending(bws);
        return (arrangeFilesZigZag(d,m));
    }

// main function for testing
    public static void main(String []args) {
        GetInput gi = new GetInput();
        gi.getInput();
        int n = gi.getNoOfFiles();
        int m = gi.getNoOfServers();
        int[] bws = new int[n];
        bws = gi.readFile();
        int optMaxB;
        Heuristic h = new Heuristic();
        long startTime = System.currentTimeMillis();
        optMaxB = h.runHeuristic1(bws,m);
        //optMaxB = h.runHeuristic2(bws,m);
        long stopTime = System.currentTimeMillis();
        long executionTime = stopTime - startTime;
        System.out.println("Maximum Bandwidth : "+ optMaxB);
        System.out.println("Time taken : "+ executionTime);
    }
}
```

Getting input:

```
public class GetInput {

          private String filePath; // path of file containing bandwidth requirements;
                                                     // one bandwidth per line
          private int noOfServers; // number of servers
          private int noOfFiles;   // number of files

          public void setFilePath(String path) {
              this.filePath = path;
           }
          public String getFilePath() {
              return filePath;
          }
          public void setNoOfServers(int m) {
              this.noOfServers = m;
           }
          public int getNoOfServers() {
              return noOfServers;
           }
          public void setNoOfFiles(int n) {
              this.noOfFiles = n;
           }
          public int getNoOfFiles() {
              return noOfFiles;
           }

// gets inputs from user
          public void getInput(){
                BufferedReader b = new BufferedReader(new
InputStreamReader(System.in));
                try{
                      System.out.print("Enter file path having bandwith requirements:
");
                      setFilePath(b.readLine());
                      System.out.print("Enter #servers: ");
                      setNoOfServers(Integer.parseInt(b.readLine()));
                      System.out.print("Enter #files: ");
                      setNoOfFiles(Integer.parseInt(b.readLine()));
                 }
             catch (IOException e) {
                System.out.println(e);
                System.exit(1);
             }
          }

// returns an array containing bandwidths from the given file path
          public int[] readFile() {
                  int n = getNoOfFiles();
                     int[] bws = new int[n];
                     String path = getFilePath();
                     try{
                            FileReader f = new FileReader(path);
                            BufferedReader b = new BufferedReader(f);
```

```
                          String bw;
                          for (int i = 0; i < n; i++){
                                 bw = b.readLine();
                                 bws[i] = Integer.parseInt(bw);

                          }
                          b.close();
                    }
                    catch (IOException e) {
                             System.out.println(e);
                             System.exit(1);
                          }
                    return bws;
          }

// main function for testing
          public static void main(String []args) {
               GetInput gi = new GetInput();
        gi.getInput();
        gi.readFile();
        System.out.println(gi.getFilePath());
        System.out.println(gi.getNoOfServers());
        System.out.println(gi.getNoOfFiles());
        System.out.println(gi.readFile());
           }
}
```