**Sound Packing Algorithm Analysis**

**Documentation**

Dina Essam Mohamed Omran

Esraa Mohamed Mohamed Ibrahrim

Aya Soliman Mahmoud Hegazy

Omar Hussien Saleh Mahmoud

Mohab Mohamed Mohamed Ali

**1-Worst fit algorithm using linear search**

public static LinkedList<Folder> worstFitLS(List<AudioFile> input, int maxcap) //O(NxM)

{

LinkedList<Folder> myFolders = new LinkedList<Folder>();

Folder firstFolder = new Folder(maxcap);

myFolders.AddLast(firstFolder);

for (int i = 0; i < input.Count; i++) //O(N)

{

int max\_remain\_cap = 0;

Folder max\_remain\_folder = null;

LinkedListNode<Folder> current = myFolders.First;

while (current!=null) //O(M)

{

if (current.Value.remaincap > max\_remain\_cap)

{

max\_remain\_cap = current.Value.remaincap;

max\_remain\_folder = current.Value;

}

current = current.Next;

}

if ((max\_remain\_folder != null) &&

(max\_remain\_folder.remaincap >= (int)input[i].Duration.TotalSeconds))

{

max\_remain\_folder.addFile(input[i]);

}

else

{

Folder folder = new Folder(maxcap);

folder.addFile(input[i]);

myFolders.AddLast(folder);

}

}

return myFolders;

}

Overall complexity of the function is **O(NxM)** where N is the number of Audio files, M is the number of Folders.

**2-Worst fit decreasing linear search**

public static LinkedList<Folder> worstFitDecreasingLS(List<AudioFile> input, int maxcap)

{

// convert the input list to an array

AudioFile[] inputArray = input.ToArray();

MinHeap.HeapSort(inputArray); //O(Nlog(N))

LinkedList<Folder> myFolders = new LinkedList<Folder>();

Folder firstFolder = new Folder(maxcap);

myFolders.AddLast(firstFolder);

for (int i = 0; i < inputArray.Length; i++)//O(N)

{

int max\_remain\_cap = 0;

Folder max\_remain\_folder = null;

LinkedListNode<Folder> current = myFolders.First;

for (int j = 0; j < myFolders.Count; j++)//O(M)

{

if (current.Value.remaincap > max\_remain\_cap)

{

max\_remain\_cap = current.Value.remaincap;

max\_remain\_folder = current.Value;

}

current = current.Next;

}

if ((max\_remain\_folder != null) &&

(max\_remain\_folder.remaincap >= (int)inputArray[i].Duration.TotalSeconds))

{

max\_remain\_folder.addFile(inputArray[i]);

}

else

{

Folder folder = new Folder(maxcap);

folder.addFile(inputArray[i]);

myFolders.AddLast(folder);

}

}

return myFolders;

}

Overall complexity is bounded by **O( max( N log N),(N,M) )** where N is the number of Audio files, M is the number of Folders.

**3-Worst fit decreasing priority queue**

public static List<Folder> worstFitDecreasingHEAP(List<AudioFile> input, int maxcap)

{

MaxHeap<AudioFile> Audios = new MaxHeap<AudioFile>(input);

MaxHeap<Folder> myFolders = new MaxHeap<Folder>();

Folder firstFolder = new Folder(maxcap);

Folder temp;

myFolders.PUSH(firstFolder);

while (input.Count > 0) **//O(N)**

{

if(Audios.Top().Duration.TotalSeconds <= myFolders.Top().remaincap)

{ //O(log N)

myFolders.Top().addFile(Audios.Top()); //O(1)

temp = myFolders.Top(); //O(1)

myFolders.POP(); //O(LogM)

myFolders.PUSH(temp); //O(LogM)

}

else

{

temp = new Folder(maxcap); //O(1)

temp.addFile(Audios.Top()); //O(1)

myFolders.PUSH(temp); //O(LogM)

}

Audios.POP(); //O(logN)

}

return myFolders.GETLIST();

}

Overall complexity is bounded by **O(N log N)** where N is the number of Audio files.

4-worst fit using priority queue

public static List<Folder> worstFitHEAP(List<AudioFile> input, int maxcap)

{

MaxHeap<Folder> myFolders = new MaxHeap<Folder>();

Folder firstFolder = new Folder(maxcap); //O(1)

Folder temp; //O(1)

myFolders.PUSH(firstFolder);

for(int i=0;i<input.Count;i++) //O(N)

{

if (input[i].Duration.TotalSeconds <= myFolders.Top().remaincap)

{

myFolders.Top().addFile(input[i]);

temp = myFolders.Top();

myFolders.POP(); //O(log M)

myFolders.PUSH(temp); //O(log M)

}

else

{

temp = new Folder(maxcap); //O(1)

temp.addFile(input[i]); //O(1)

myFolders.PUSH(temp); //O(log M)

}

}

return myFolders.GETLIST();

}

}

}

Overall complexity is bounded by **O(N log M)** where N is the number of Audio files, M is the number of Folders.

**5-First Fit Decreasing using Linear search**

public static LinkedList<Folder> firstFitDecreasingLS( List<AudioFile> input, int maxcap)

{

LinkedList<Folder> myFolders = new LinkedList<Folder>();

AudioFile[] inputArray = input.ToArray();

MinHeap.HeapSort(inputArray); //O(Nlog(N))

for (int i = 0; i < inputArray.Length; i++)//O(N)

{

Folder remain\_folder = null;

LinkedListNode<Folder> current = myFolders.First;

for (int j = 0; j < myFolders.Count; j++)//O(M)

{

if (current.Value.remaincap >= (int)inputArray[i].Duration.TotalSeconds)

{

remain\_folder = current.Value;

break;

}

current = current.Next;

}

if ((remain\_folder != null))

{

remain\_folder.addFile(inputArray[i]);

}

else

{

Folder folder = new Folder(maxcap);

folder.addFile(inputArray[i]);

myFolders.AddLast(folder);

}

}

return myFolders;

}

Overall complexity is equal to **O( max( N log(N), NxM ) ) )** where N is the number of Audio files, M is the number of Folders.

**6-Best Fit Strategy**

public static LinkedList<Folder> bestFitLS(List<AudioFile> input, int maxcap)

{

LinkedList<Folder> myFolders = new LinkedList<Folder>();

Folder firstFolder = new Folder(maxcap);

myFolders.AddLast(firstFolder);

for (int i = 0; i < input.Count; i++) //O(N)

{

int min\_remain\_cap = maxcap;

Folder min\_remain\_folder = null;

LinkedListNode<Folder> current = myFolders.First;

for (int j = 0; j < myFolders.Count; j++) //O(M)

{

if ((current.Value.remaincap <= min\_remain\_cap) &&

(current.Value.remaincap >= (int)input[i].Duration.TotalSeconds))

{

min\_remain\_cap = current.Value.remaincap;

min\_remain\_folder = current.Value;

}

current = current.Next;

}

if ((min\_remain\_folder != null))

{

min\_remain\_folder.addFile(input[i]);

}

else

{

Folder folder = new Folder(maxcap);

folder.addFile(input[i]);

myFolders.AddLast(folder);

}

}

return myFolders;

}

Overall complexity of the function is **O(NxM)** where N is the number of Audio files, M is the number of Folders

**7-Best Fit Decreasing using Linear Search**

public static LinkedList<Folder> bestFitDecreasingLS(List<AudioFile> input, int maxcap)

{

AudioFile[] inputArray = input.ToArray();

MinHeap.HeapSort(inputArray); //O(Nlog(N))

return bestFitLS(inputArray.ToList<AudioFile>(), maxcap);

}

Overall complexity is equal to **O( max( N log(N), NxM ) ) )** where N is the number of Audio files, M is the number of Folders.

**8-First Fit using Linear Search**

public static LinkedList<Folder> firstFitLS(List<AudioFile> input, int maxcap)

{

LinkedList<Folder> myFolders = new LinkedList<Folder>();

AudioFile[] inputArray = input.ToArray();

for (int i = 0; i < inputArray.Length; i++)//O(N)

{

Folder remain\_folder = null;

LinkedListNode<Folder> current = myFolders.First;

for (int j = 0; j < myFolders.Count; j++)//O(M)

{

if (current.Value.remaincap >= (int)inputArray[i].Duration.TotalSeconds)

{

remain\_folder = current.Value;

break;

}

current = current.Next;

}

if ((remain\_folder != null))

{

remain\_folder.addFile(inputArray[i]);

}

else

{

Folder folder = new Folder(maxcap);

folder.addFile(inputArray[i]);

myFolders.AddLast(folder);

}

}

return myFolders;

}

Overall complexity of the function is **O(NxM)** where N is the number of Audio files, M is the number of Folders.

**9-Next Fit using Linear Search**

public static LinkedList<Folder> NextFitLS(List<AudioFile> input, int maxcap)

{

LinkedList<Folder> myFolders = new LinkedList<Folder>();

Folder firstFolder = new Folder(maxcap);

myFolders.AddLast(firstFolder);

AudioFile[] inputArray = input.ToArray();

Folder temp;

for (int i = 0; i < inputArray.Length; i++)//O(N)

{

if(inputArray[i].Duration.TotalSeconds<=myFolders.Last().remaincap)

{

myFolders.Last().addFile(inputArray[i]);

}

else

{

temp = new Folder(maxcap);

temp.addFile(input[i]);

myFolders.AddLast(temp);

}

}

return myFolders;

}

Overall complexity of the function is **O(N)** where N is the number of Audio files, M is the number of Folders.

**10-Next Fit Decreasing using Linear Search**

public static LinkedList<Folder> NextFitDecreasingLS(List<AudioFile> input, int maxcap)

{

AudioFile[] inputArray = input.ToArray();

MinHeap.HeapSort(inputArray); //O(Nlog(N))

return NextFitLS(inputArray.ToList<AudioFile>(), maxcap);

}

Overall complexity of the function is **O(N log(N))** where N is the number of Audio files, M is the number of Folders.

**11-Folder Filling**

public static LinkedList<Folder> folderFilling(LinkedList<AudioFile> input, int maxcap)

{

LinkedList<Folder> myFolders = new LinkedList<Folder>();

while (input.Count != 0) //O(N)

{

int N = input.Count;

bool[] taken = new bool[N+1];

Folder[,] Timeline = new Folder[N+1, maxcap + 1];

LinkedListNode<AudioFile> current = input.First;

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

{

for (int w = 0; w <= maxcap; w++) //O(D)

{

if (i == 0 || w == 0)

Timeline[i, w] = new Folder(maxcap);

else if (current.Value.Duration.TotalSeconds <= w)

{

Folder folder2 = Timeline[i - 1, w];

Folder folder1 = Timeline[i - 1, w - (int)current.Value.Duration.TotalSeconds];

/////////////////////////////////////////////////////////////////////////

if (taken[i] != true && folder1.remaincap >= current.Value.Duration.TotalSeconds &&

!folder1.files.Contains(current.Value))

folder1.addFile(current.Value);

else

folder1 = new Folder(maxcap);

////////////////////////////////////////////////////////////////////////

if (folder1.remaincap <= folder2.remaincap)

{

Timeline[i, w] = folder1;

}

else

Timeline[i, w] = folder2;

}

else

Timeline[i, w] = Timeline[i - 1, w];

}

//

if(i!=0)

current=current.Next;

//

}

myFolders.AddLast(Timeline[N, maxcap]);

LinkedListNode<AudioFile> current2 = Timeline[N, maxcap].files.First;

for (int y = 0; y < Timeline[N, maxcap].files.Count; y++)

{

int index = input.Select((item, inx) => new { item, inx }).First(x => x.item == current2.Value).inx;

taken[index] = true;

input.Remove(current2.Value);

current2 = current2.Next;

}

}

return myFolders;

}

public static LinkedList<Folder> folderFilling2(List<AudioFile> input, int maxcap)

{

AudioFile[] inputArray = input.ToArray();

MinHeap.HeapSort(inputArray); //O(Nlog(N))

LinkedList<AudioFile> inputLL = new LinkedList<AudioFile>(inputArray);

return folderFilling(inputLL, maxcap);

}

Overall complexity of the function is **by O(N2×D), N is number of audio files, D is the desired duration per folder.**