Huff man Coding

Huff man coding is a lossless data compression algorithm.
The most frequency character gets the smallest code and the least frequent character gets the largest code.

⊛ একটার code অন্যটার prefix code হবে না।

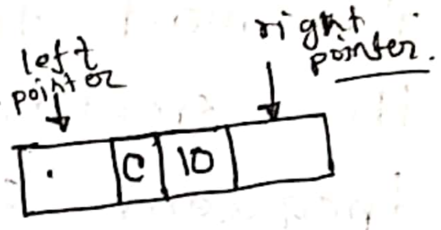
Huffman ~~tree~~ coding এর দুইটা part থাকে।

1. Build a Huffman Tree from ~~ip~~ input character.
2. Traverse the Huffman Tree and assign code to character.

code of huffman

```
#include <stdio.h>
# include <stdlib.h>
#define    Max_Tree_hight  100 ;

struct  MinHeapNode{
        char  data;
        int  freq;
        struct  MinHeapNode *left, *right;
};
```



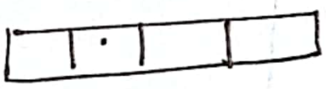
// অনেক গুলা MinHeap Node নিয়ে একটা
MinHeap বানাবো তাই যে Array তে
MinHeap এর Node থাকবে সেই Array
অবশ্যই কোনো value কে না বরং পয়েন্টার
রাখতে পারে এমন হওয়া উচিত তাই
এই অংশে Array এর সামনে দুইটা **
দিতে হবে। //

```
struct  MinHeap {
        int  size ;  // হিপের সাইজ
        int  capacity ;
        struct  MinHeapNode **array ;
};
```

// এই অংশে বা function এ আমরা শুধুমাত্র char এবং frequency নিবে এবং নতুন memory Block ক্রিয়েট করে শুধু Assign করবো। আর কোনো কাজ নাই, হুবহু BST এর New Node ক্রিয়েটের মতোন। //

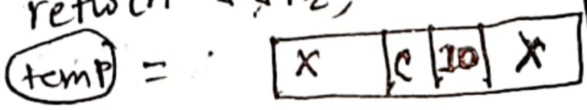
```
struct MinHeapNode* newNode(char data,
                            int freq)
{
    struct MinHeapNode *temp =
      (struct MinHeapNode*) malloc(sizeof(
                              struct MinHeapNode));

    // memory block তৈরি শেষ!!
```



```
    temp->left = temp->right = Null;
    temp->data = data;
    temp->freq = freq;
    return temp;
```

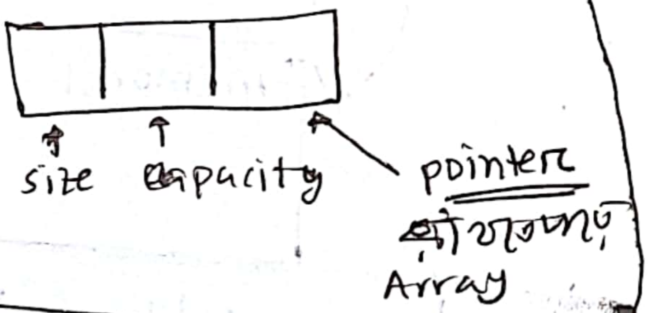
Basically এইটা return করছি,



given capacity অনুযায়ী MinHeap তৈরি করতে হবে

```
struct MinHeap* createMinHeap(int capacity)
{
        struct MinHeap* minHeap
        = (struct MinHeap*)malloc(sizeof(struct
                                        MinHeap));
```

minHeap নামে Memory block বানানো হয়েছে



```
    minHeap -> size = 0;
    minHeap -> capacity = capacity;
    minHeap -> array = (struct MinHeapNode**)
              malloc(Minheap -> capacity * sizeof
                        (struct MinHeapNode*));
    return minHeap;
}
```

যেকোনো দুইটা কে swap করার জন্য ফাংশন বানাই আমরা এখানে ~~pointer~~ Array of pointer এর দুইটার মধ্যে swap করবো তাই * ** sign দিয়ে Address কে swap করছি.

```
void swapMinHeapNode( struct MinHeapNode **a,
                      struct MinHeapNode **b)
{
    struct MinHeapNode *t = *a;
                        *a = *b;
                        *b = t;
}
```

এই অংশে standard minHeapify function যেভাবে লিখতে হয় সেভাবেই লেখা হচ্ছে বাট নরমাল array দিয়ে আমরা যেভাবে Heapyfy করছে সেটা থেকে একটু আলাদাভাবে structure এর field গুলো নিয়ে কাজ করতে হবে

```
void   minHeapify(struct MinHeap* minHeap,
                                          int idx)
{    int  smallest = idx;
     int  left = 2*idx+1;
     int  right = 2*idx+2;

  if ((left < minHeap→size) &&
     (minHeap→array[left])→ freq <
         minHeap→array[smallest]→ freq)
     smallest = left;

  if ( right < minHeap→size  &&
       minHeap→ array[right] → freq <
          minHeap → array[smallest → freq)

    smallest = right;

  if (smallest != idx)   // যদি smallest
                         // হয় root নয়
  { swapMinHeapNode( &minHeap→array[smallest],
                          & minHeap→ array[idx]);

    minHeapify(minHeap, smallest)
  }
}
```

লিখছি ফাংশন, যে Heapsize 1 হইলো কি না তথা আমার আর দুইটা করে minimum value extract করা বন্ধ করা উচিত কি না তা জানার জন্য !

```
int isSizeOne(struct MinHeap* minHeap)
  { return (minHeap->size == 1);

  }
```

যদি 1 হয় তবে 1 রিটার্ন করবে মানে সত্য হলে 1 আর মিথ্যা হলে 0 রিটার্ন করবে । ইচ্ছে হলে if, else ব্যবহার করেও করা যাবে

এই ~~ট~~ ফাংশন দিয়ে ~~দুইটা~~ minHeap node কে বের করতে পারবো বা ডিলিট করে দিবো easy way তে ~~অ~~ অর্থাৎ size কমায়ে

```
struct MinHeapNode* extractMin(struct MinHeap* minHeap)
{
   struct MinHeapNode* temp = minHeap->array[0];
  minHeap->array[0] = minHeap->array[minHeap->size-1];
       --minHeap->size;
  minHeapify(minHeap, 0);
  return temp;
}
```

bexitrol® F

Insert সংক্রান্ত, আলোচনা নম্বর (২) Heap এর ইনডেক্সিং এর উপর তার child (left আর right) এর ফর্মুলা depend করে। আর penent এর formula ও change হবে। এখানে 0 based indexing তাই (i-1)/2 নেওয়া হয়েছে i/2 এর পরিবর্তে। সাবধান!

```
void insertMinHeap(struct MinHeap * minHeap,
                         struct MinHeapNode * minHeapNode)

{   ++ minHeap → size;
    int  i = minHeap → size -1;
 while ( i && MinHeapNode → freq <
            minHeap → array [(i-1)/2] → freq)
   {  minHeap → array[i] = minHeap → array[(i-1)/2
         i = (i-1)/2 ;
   }
 minHeap → array[i] = MinHeapNode ;
}
```

minHeap নিয়মের জন্য প্রথমে child হতে বাদ দিয়ে শুরু করব এমন একটি node থেকে যার leaf আছে। এখানে MinHeap create করার ফাংশন

```
void buildMinHeap (struct MinHeap* minHeap)
{
        int n = minHeap -> size - 1;
        int i;
        for (i = (n-1) /2 ; i>=0 ; --i)
            minHeapify (minHeap, i);
}
```

Array print এর ফাংশন কিছু নাই বিশেষত্ব দরকার

```
void  printArray ( int arr[], int n)
{
    ~~voi~~ int i;
      for (i=0; i<n; ++i)
         print ("%d", arr[i]);
      printf ("\n");
}
```

কোনো node টি এর leaf কি না তা চেক করবো। কিভাবে? যদি memory block এর left pointer এবং right pointer null থাকে তবে leaf নাহলে leaf না; ☺

```
int isLeaf(struct MinHeapNode *root)
{
        return !(root → leaf) && !(root →
                                        right);
}
```

minHeap creat ~~করার~~ জন্য Buid করবো আমাদের আগে creatMinHeap করাই আছে তাই লাগবেনা!

```
struct MinHeap* createAndBuildMinHeap
            ( char data[], int freq[], int size)
{   struct MinHeap * minHeap= createMinHeap
                                        (size);
  for(int i=0; i<size; ++i)
        minHeap → array[i] = newNode (data[i],
                                        freq);
```

```
    minHeap->size = size;
    buildMinHeap (minHeap);
    return   minHeap;
}
```

এতক্ষণে, অরিন্দম! ~~[illegible]~~ ~~bu~~ huffman ট্রি Build করছি অবশেষে! কিছু না শুধু ছোট দুইটা extract করা আর Add করে push করা!

```
struct MinHeapNode * buildHuffmanTree(
                         char data[], int freq[], int size)
{   struct MinHeapNode *left, *right, *top;
    struct MinHeap* minHeap
          = createAndBuildMinheap (data, freq, size)
    whild ( ! isSizeone( minHeap))
     {    left = extractMin(minHeap);
          right = extractMin( minHeap);
```

```
        top = newNode('$', left->freq +
                                 right->freq);

        top -> left = left;
        top -> right = right;
      insertMinHeap (minHeap, top);

  }   }
 return   extractMin (minHeap);

}
```

leaf node
কারণে মধ্যবর্তী
সবগুলোতেই ডলার
সাইন char হিসেবে

```
void  printCodes ( struct MinHeapNode *root,
                            int arr[], int top)

 {   if (root -> left)
       { arr[top] = 0;
         printCodes (root -> left, arr, top+1);
     if (root -> right)
       { array [top] = 1;
         printCodes (root -> righ, arr, top+1);
       }
if ( isleaf (root))
  { printf ("%c : " ; root -> data) ;
    printArr (arr, top) ;
  }
}
```

উপরের print code function টি [illegible] ম্যাম দিয়েছেন এখানে, কিভাবে recursion কল হচ্ছে আবার stack (arr[]) তে 0 এবং 1 জমা হচ্ছে তার সিমুলেশন। একটু নিচে সিমুলেট করলেই বুঝা যাবে ব্যাপারটা

~~এই [illegible] হতে~~

এই ফাংশন হচ্ছে কোডের মূল অংশ যে মেইন ফাংশনটাকে কল করাতে হবে।

```
void HuffmanCodes (char ~~data~~ data[],
            int freq[], int size)
{   struct MinHeapNode *root =
            buildHuffmanTree(data, freq,
                                  size);
  int arr[Max_Tree_HT], top = 0;
  print code ( root, arr, top);

}
```

মেইন ফাংশন

```
int main()
{
   char[] = { 'a', 'b', 'c', 'd', 'e', 'f'};
   int freq[] = { 5, 9, 12, 13, 16, 45};
   int size = sizeof(arr) / sizeof(arr[0]);
   HuffmanCodes (array, freq, size);
return 0;
}
```