|  |  |
| --- | --- |
| **SORTING**  int\* **insertInOrderIter**(int b,int \*a, int n){  if (n==0) return a;  int i;  for(i=n-1;a[i]>b && i>=0;i--){  a[i+1]=a[i];  a[i]=b;  }  return a;  }  void **insertInOrder**(int b,int \*a, int n){  int i;  if (n==0) return a;  if (a[n-1]>b){  a[n]=a[n-1];  a[n-1]=b;  insertInOrder(a[n-1],a,n-1);  }  }  int\* **insertSort**(int \*a,int n){  //printf("Hello %d\n",n);  if (n==0) return a;  if (n>0) {  a=insertSort(a,n-1);  //printf("Hello %d",n);  insertInOrder(a[n-1],a,n-1); // can also use insertInOrderIter here  return a;  }  }  void **mergeNotInPlace**(int \*a,int low,int high,int \*c,int low2, int high2,int\* b,int low3, int high3){  if (low > high){  int i;  for (i=low2;i<=high2;i++){  b[low3] = c[i];  low3++;  }  return;  }  else if (low2 > high2){  int i;  for (i=low;i<=high;i++){  b[low3] = a[i];  low3++;  }  return;  }  else if (a[low]<=c[low2]){  b[low3]=a[low];  mergeNotInPlace(a,low+1,high,c,low2,high2,b,low3+1,high3) ;}  else {  b[low3]=c[low2];  mergeNotInPlace(a,low,high,c,low2+1,high2,b,low3+1,high3) ;}  }  void **mergeSort**(int \*a,int low,int high){  if (high-low < 1) return;  int b[high-low+1];  int mid = (high+low)/2;  mergeSort(a,low,mid);  mergeSort(a,mid+1,high);  mergeNotInPlace(a,low,mid,a,mid+1,high,b,0,high-low);  int i;  for (i=0;i<=high-low;i++){  a[i+low]=b[i];  }  }  void **mergeInPlaceIter**(int \*a,int low,int high,int low2, int high2){  int temp = 0,i;  while(low<=high && low2<=high2){  if (a[low]<=a[low2]) low++;  else {  temp = a[low2];  for(i=low2;i>low;i--){  a[i] = a[i-1];  }  a[i]=temp;  low++; high++; low2++;  }  }  return;  }  void **mergeInPlace**(int \*a,int low,int high,int low2, int high2){  int temp = 0,i;  if (low<=high && low2<=high2){  if (a[low]<=a[low2]) {  mergeInPlace(a,low+1,high,low2,high2);  }  else {  temp = a[low2];  for(i=low2;i>low;i--){  a[i] = a[i-1];  }  a[i]=temp;  mergeInPlace(a,low+1,high+1,low2+1,high2);  }  }  return;  }  void **mergeSort2**(int \*a,int low,int high){  if (high-low < 1) return;  //int b[high-low+1];  int mid = (high+low)/2;  mergeSort(a,low,mid);  mergeSort(a,mid+1,high);  mergeInPlace(a,low,mid,mid+1,high); // can also use mergeInPlaceIter here  return;  }  void swap(int \*a,int b, int c){  int temp=a[b];  a[b]=a[c];  a[c]=temp;  }  int pivot(int \*a,int low, int high){ //random  return rand()%(high+1-low) + low;  }  int pivot2(int \*a,int low, int high){ //median of three  int mid =(high+low)/2;  if (a[high]<a[low]) swap(a,low,high);  if (a[mid]<a[low]) swap(a,low,mid);  if (a[high]<a[mid]) swap(a,high,mid);  return mid;  }  int pivot3(int \*a,int low, int high){ //random  return high;  }  int **partition**(int \*a,int low, int high,int piv){  swap(a,low,piv);  int lt =low+1 ; int rt = high ; int pv =a[low];  while(lt<=rt){  for(;lt<=high && a[lt]<=pv;lt++);  for(;a[rt]>pv;rt--);  if(lt < rt) {  swap(a,lt,rt);  lt++; rt--;  }  }  int pPos;  if (lt == rt ) pPos = lt;  else pPos = lt-1;  swap(a,low,pPos);  return pPos;  }  void **quickSort**(int \*a,int low, int high){  if (low<high){  int piv = pivot2(a,low,high);  printf("Pivot %d %d %d\n",piv,low,high);  int i;  int part = partition(a,low,high,piv);  for(i=0;i<6;i++) printf("%d",a[i]);  printf("Part %d\n",part);  quickSort(a,low,part-1);  quickSort(a,part+1,high);  }    }  int **partition3way**(int \*a,int low, int high,int piv,int\* eq1,int\* eq2){  swap(a,high,piv);  int lt =low ; int rt = high-1 ; int pv =a[high]; //int mid =(high+low)/2;  printf("Pivot %d\n",pv);  while(lt<rt){  if(a[lt]<pv) {  printf("wapping %d %d\n",lt,low);  swap(a,lt,low);  low++;  lt++;  }  else if(a[lt]>pv) {  printf("swappingRT %d %d\n",lt,rt);  swap(a,lt,rt);  rt--;  }  else if(a[lt]==pv) lt++;  }    printf("swappingLASt %d %d\n",high,rt);  swap(a,high,rt);  printf("eq1 %d eq2 %d\n",low,rt);  \*eq1 = low;  \*eq2 = rt;  int i;  for(i=0;i<10;i++) printf("%d",a[i]);  return rt;  }  void **quickSort3way**(int \*a,int low, int high,int eq1,int eq2){ //eq1 & eq2 are index of equal elements  if (low<high){  int piv = pivot3(a,low,high);  printf("Pivot %d %d %d %d %d %d\n",a[piv],piv,low,high,eq1,eq2);  int i;  int part = partition3way(a,low,high,piv,&eq1,&eq2);    printf("Part %d\n",part);  quickSort3way(a,low,(eq1)-1,eq1,eq2);  quickSort3way(a,(eq2)+1,high,eq1,eq2);  }  }  int **quickSelect**(int \*a,int low, int high,int key){  if (key>high) return -1;  if (low<=high){  int piv = pivot2(a,low,high);  int i;  int part = partition(a,low,high,piv);  if (part==key) return a[part];  else if (part>key){  quickSelect(a,low,part-1,key);  }  else {  quickSelect(a,part+1,high,key);  }  }  }  int main(int argc, char \*argv[]) {  insertSort(a,5);  mergeSort(b,0,4);  mergeSort2(c,0,5);  quickSort(d,0,5);  quickSort3way(f,0,10,eq1,eq2);  }  **BUCKETSORT**  #include <stdio.h>  #include <stdlib.h>  #include <limits.h>  #include <math.h>  #include <stdbool.h>  /\* run this program using the console pauser or add your own getch, system("pause") or input loop \*/  struct bucket{  int key;  struct bucket\* next;  };  struct bucket\*\* insert(struct bucket\*\* temp,int b,int low){  if (temp[b-low]->key==INT\_MIN) {  temp[b-low]->key=b;  printf("%d kiaif\n",b);  return temp;  }  else {  struct bucket\* cur =temp[b-low];  while(cur->next!=NULL) cur= cur->next;  struct bucket\* temp2=(struct bucket\*)malloc(sizeof(struct bucket));  temp2->key=b;  temp2->next=NULL;  cur->next=temp2;  printf("%d kia",b);  }  return temp;  }  int\* **bucketSort**(int a[],int size,int low,int lar){ //double pointer without duplicates linked list) yet to improve  int range =lar-low+1;  int i=0;  struct bucket\*\* temp = (struct bucket\*\* )malloc(range\*sizeof(struct bucket\*));    for(i;i<range;i++){  temp[i]=(struct bucket\*)malloc(sizeof(struct bucket));  temp[i]->key=INT\_MIN;  temp[i]->next=NULL;  }  for(i=0;i<size;i++){  temp=insert(temp,a[i],low);  }    int k=0;  for(i=0;i<range;i++){  if (temp[i]->key!=INT\_MIN){  struct bucket\* cur=temp[i];  while(cur!=NULL){  a[k]=cur->key;  k++;  printf("%d ehllo\n",cur->key);  cur=cur->next;  }  }  }  return a;  }  void **bucketsort2**(int a[],int size, int low, int high){ //array without duplicates (linked list) yet to improve  struct bucket b[high-low+1];  int i=0;  for(i=0;i<(high-low+1);i++){  b[i].key=INT\_MIN;  }  for(i=0;i<size;i++){  if(b[a[i]-low].key==INT\_MIN)  b[a[i]-low].key=a[i];  else {  b[a[i]-low].next = (struct bucket\*)malloc(sizeof(struct bucket));  b[a[i]-low].next = &b[a[i]-low];  b[a[i]-low].key = a[i];  }  }  int j=0;  printf("eher");  for(i=0;i<(high-low+1);i++){  if(b[i].key!=INT\_MIN){  struct bucket\* cur = &b[i];  while (cur!=NULL){  a[j++]=cur->key;  cur= cur->next;  }  }  }  }  void **bucketsort3**(int a[],int size, int low, int high){ //array with duplicates, not stable , not linked list  int b[high+1];  int i=0;  for(i=0;i<(high+1);i++){  b[i]=INT\_MIN;  }  for(i=0;i<size;i++){  ++b[a[i]];  }  int j=0,k;  for(i=0,j=0;j<(high+1);j++){  for(k=b[j];k!=0 && k!=INT\_MIN;--k){  a[i++]=j;  }  }  }  void printarr(int \*a,int size){  int i=0;  //printf("ehre");  for (i;i<size;i++){  //printf("ehre");  printf("%d--%d\t",i,\*(a+i));  }  return;  }  //COunting SOrt  void **countingsort**(int \*a,int size,int high){  int c[high],i,j;  for(i=0;i<high;i++){  c[i]=0;  }  for(j=0;j<size;j++){  c[a[j]]++;  }  //printarr(c,high);  for(i=1;i<high;i++){  c[i]=c[i]+c[i-1]; // no. of elements <=i  }  //printarr(c,high);  //printf("here");  int b[size];  for(j=0;j<size;j++){  b[c[a[j]]-1]=a[j];  c[a[j]]--;  }  printarr(b,7);  }  //Radix Sort  struct radixnode;  typedef struct radixnode\* NODE;  struct radixnode{  int key;  NODE next;  };  NODE\* empty(NODE \* bucket){  int i=0;    for (i=0;i<10;i++){  bucket[i]->key=INT\_MIN;  bucket[i]->next=NULL;  }  return bucket;  }  NODE\* **insertradix**(NODE\* b, int mod,int key){  if (b[mod]->key==INT\_MIN) b[mod]->key=key;  else {  NODE cur =b[mod];  while (cur->next!=NULL) cur=cur->next;  NODE temp = (NODE)malloc(sizeof(struct radixnode));  cur->next=temp;  printf("\nelse %d---%d\n",mod,key);  cur=cur->next;  cur->key=key;  cur->next=NULL;  }  return b;  }  int\* radix(int \*a,int size, int n){  NODE\* bucket = (NODE\*)malloc(10\*sizeof(NODE));  int i;  for (i=0;i<10;i++){  bucket[i]=(NODE)malloc(sizeof(struct radixnode));  }  int b,mod;  bucket = empty(bucket);  printf("going into insert\n");  for(i=0;i<size;i++){  b=a[i]/pow(10,n);  mod = b%10;  printf("inserting %d---%d\n",mod,a[i]);  bucket=insertradix(bucket,mod,a[i]);  }  int k=0;  for(i=0;i<10;i++){  NODE cur=bucket[i];  while(cur!=NULL && cur->key!=INT\_MIN){  a[k++]=cur->key;  cur=cur->next;  }  }  printarr(a,size);  return a;  }  void **radixsort**(int \*a,int size,int len){  int i=0;  for(i;i<len;i++){  printf("going for %d\n",i);  a=radix(a,size,i);  }  }  //Bucket find add delete  typedef struct radixnode bucketing;  typedef NODE bucketNODE;  bucketNODE\* insertbucketing(bucketNODE\* b, int mod,int key){  if (b[mod]->key==INT\_MIN) b[mod]->key=key;  else {  bucketNODE cur =b[mod];  while (cur->next!=NULL) cur=cur->next;  bucketNODE temp = (bucketNODE)malloc(sizeof(bucketing));  cur->next=temp;  printf("\nelse %d---%d\n",mod,key);  cur=cur->next;  cur->key=key;  cur->next=NULL;  }  return b;  }  bucketNODE\* create(bucketNODE\* bn, int a[],int size){  int i,mod;  for(i=0;i<size;i++){  mod = a[i]%10;  printf("inserting %d---%d\n",mod,a[i]);  bn=insertbucketing(bn,mod,a[i]);  }  return bn;  }  int find(bucketNODE\* bn,int key){  int mod = key%10;  bucketNODE cur = bn[mod];  while(cur!=NULL && cur->key!=INT\_MIN){  if(cur->key==key) return 1;  cur=cur->next;  }  return 0;  }  bool member(bucketNODE\* bn,int key){  int mod = key%10;  bucketNODE cur = bn[mod];  while(cur!=NULL && cur->key!=INT\_MIN){  if(cur->key==key) return true;  cur=cur->next;  }  return false;  }  bucketNODE\* delete(bucketNODE\* bn,int key){  int mod = key%10;  bucketNODE cur = bn[mod];  bucketNODE par = bn[mod];  if (bn[mod]->key==key) {  bn[mod]=bn[mod]->next;  }  while(cur!=NULL && cur->key!=INT\_MIN){  if(cur->key==key) {  par->next=NULL;  free(cur);  return bn;  }  par=cur;  cur=cur->next;  }  return bn;  **READDATA INT**  **int\* readData(int N){**  **int\* arr = (int \*)malloc(sizeof(int));**  **int i=0;**  **for(i=0;i<N;i++){**  **//printf("here\n");**  **scanf("%d",&arr[i]);**  **//printf("here2\n");**  **}**  **return arr;**  **}**  **RANDOMCONSTRUCT**  **randomconstruct(BinaryTree b,int\* arr,int N){**  **int i=0;**  **for (i = 0; i < N; i++) { // shuffle array**  **int temp = arr[i];**  **int randomIndex = rand() % N;**  **arr[i] = arr[randomIndex];**  **arr[randomIndex] = temp;**  **}**  **for(i=0;i<N;i++){**  **b=insert(b,arr[i]);**  **//printf("inserting\n");**  **}**  **}**  **INPUT**  **scanf("%d",&d);**  **while(1){**  **if(d==0){ }**  **else if(d==-1){**  **break;**  **}**  **scanf("%d",&d);**  **}**  **MEMORY**  **int curheapsize;**  **int maxheapsize;**  **void\* mymalloc(unsigned int size){**  **curheapsize+=size;**  **if (curheapsize>maxheapsize){**  **maxheapsize=curheapsize;**  **}**  **return malloc(size);**  **}**  **void memProf(){**  **printf("%d\t%d\n",curheapsize,maxheapsize);**  **void myfree(void \*ptr){**  **curheapsize-=sizeof(ptr);**  **free(ptr);** | **HASHING**  struct bucket;  typedef struct bucket\* NODE;  typedef NODE\* hashtable;  struct bucket{  int key;  NODE next;  };  /// String  struct bucket2;  typedef struct bucket2\* NODE2;  typedef NODE2\* hashtable2;  struct bucket2{  char\* key;  NODE2 next;  };  int hashfunc(int key){ ///k mod m  return key%10;  }  int hashfunc2(int key){ //2^p  return key%32;  }  int hashfunc3(int key){ //k mod prime no  return key%41;  }  int hashfunc4(int key){ //MAD  return 5\*key + hashfunc2(5);  }  int hashfunc5(int key){ //MAD  return floor(5\*(key\*(sqrt(5)-1)/2));  }  int hashfunc8(int key){ // Uniform hashing (((a\*k+b)mod p)mod m)  return ((5\*key+6)%17)%10 ; //a=5,b=6,p=17,m=10  }  int hashfunc9(int key){ ///k mod m  return (key+1)%10;  }  hashtable create(int numBins){  hashtable h = (hashtable)malloc(numBins\*sizeof(NODE));  int i=0;  //printf("%d",numBins);  for(i=0;i<numBins;i++){  h[i]= (NODE)malloc(sizeof(struct bucket));  h[i]->key=INT\_MIN;  h[i]->next=NULL;  }  return h;  }  int find(hashtable h,int key){  int mod = hashfunc(key);  NODE cur = h[mod];  while(cur!=NULL && cur->key!=INT\_MIN){ // search in linked list  if(cur->key==key) return key;  cur=cur->next;  }  return 0;  }  void delete(hashtable bn,int key){  int mod = hashfunc(key);  NODE cur = bn[mod];  NODE par = bn[mod];  if (bn[mod]->key==key) {  bn[mod]=bn[mod]->next;  }  while(cur!=NULL && cur->key!=INT\_MIN){ // delete in linked list  if(cur->key==key) {  par->next=NULL;  free(cur);  }  par=cur;  cur=cur->next;  }  //return bn;  }  void insert(hashtable b, int key){  int mod = hashfunc8(key); //change it here  printf(" at %d\n",mod);  if (b[mod]->key==INT\_MIN) b[mod]->key=key;  else { //insert in linked list  NODE cur =b[mod];  while (cur->next!=NULL) cur=cur->next;  NODE temp = (NODE)malloc(sizeof(struct bucket));  cur->next=temp;  printf("\nelse %d---%d\n",mod,key);  cur=cur->next;  cur->key=key;  cur->next=NULL;  }  //return b;  }  hashtable insertlist(hashtable bn, int a[],int size){  int i;  for(i=0;i<size;i++){  printf("inserting %d",a[i]);  insert(bn,a[i]);  }  return bn;  }  void printhash(hashtable h,int size){  int i=0;  for(i;i<size;i++){  NODE cur = h[i];  printf("%d--",i);  while(cur!=NULL && cur->key!=INT\_MIN && cur->key!=0){ //0 to handle deletion  printf("%d\t",cur->key);  cur=cur->next;  }  printf("\n");  }  }  //For strings  int hashfunc6(char\* key){ // sum of ascii  int len = strlen(key);  int i,sum=0;  for(i=0;i<len;i++){  sum+=((int)key[i])%10;  }  return sum;  }  int hashfunc7(char\* key){ // sum of ascii mutiply by power  int len = strlen(key);  int i,sum=0;  for(i=0;i<len;i++){    sum+=((int)key[i])\*pow(17,len-i-1);  printf("ascii valeu %d\n--sum %d",(int)key[i],sum);  }  return sum%100;  }  hashtable2 create2(int numBins){  hashtable2 h = (hashtable2)malloc(numBins\*sizeof(NODE));  int i=0;  //printf("%d",numBins);  for(i=0;i<numBins;i++){  h[i]= (NODE2)malloc(sizeof(struct bucket2));  h[i]->key="";  h[i]->next=NULL;  }  return h;  }  void insert2(hashtable2 b, char\* key){ //string  int mod = hashfunc7(key); //change it here  printf(" at %d\n",mod);  if (b[mod]->key=="") b[mod]->key=key;  else { //insert in linked list  NODE2 cur =b[mod];  while (cur->next!=NULL) cur=cur->next;  NODE2 temp = (NODE2)malloc(sizeof(struct bucket2));  cur->next=temp;  printf("\nelse %d---%s\n",mod,key);  cur=cur->next;  cur->key=key;  cur->next=NULL;  }  //return b;  }  hashtable2 insertlist2(hashtable2 bn, char a[5][10],int size){  int i;  for(i=0;i<size;i++){  printf("inserting %s",a[i]);  insert2(bn,a[i]);  }  return bn;  }  void printhash2(hashtable2 h,int size){  int i=0;  char c[]="";  for(i;i<size;i++){  NODE2 cur = h[i];  printf("%d--",i);  while(cur!=NULL && cur->key!="" && cur->key!='\0' && strcmp(cur->key,"")!=0){  //printf("ehre");  printf("%s\t",cur->key);  cur=cur->next;  }  printf("\n");  }  }  // For open Addressing and Rehashing  int m =10;  int linearprob(int mod,int key,int j){  return (mod+j)%m ;  }  int quadraticprob(int mod,int key,int j){  return (mod+(int)pow(j,2))%m ;  }  int expoprob(int mod,int key,int j){  return (mod+(int)pow(2,j))%m ;  }  int doubleprob(int mod,int key,int j){  return (mod+j\*hashfunc2(key))%m;  }  **//rehash**  hashtable rehash(hashtable h2,int numBins){  printf("\nRehashing\n");  hashtable h = (hashtable)malloc(numBins\*sizeof(NODE));  int i=0;  //printf("%d",numBins);  for(i=0;i<numBins;i++){  h[i]= (NODE)malloc(sizeof(struct bucket));  h[i]->key=INT\_MIN;  h[i]->next=NULL;  }  for(i=0;i<(numBins/2);i++){  h[i]= h2[i];  }  return h;  }  hashtable add3(hashtable h, int key){  int mod = hashfunc(key);  if (h[mod]->key==INT\_MIN || h[mod]->key==0) {  printf(" at %d\n",mod);  h[mod]->key=key;  return h;  }  int j=0;  int mod2=mod;  while(h[mod]->key!=INT\_MIN && h[mod]->key!=0){  j++;  printf("Key: %d not at %d \t \n",key,mod);  mod = linearprob(mod2,key,j);  printf("Key: looking at %d \t",mod);  if (mod==mod2) { //for rehashing  h=rehash(h,m+10);  m=m+10;  printf("sssssssssssssssssssss %d dddddddddddddddddd",j);    }  }  printf(" 2at %d\n",mod);  h[mod]->key=key;  return h;  }  int find3(hashtable h,int key){  int mod = hashfunc(key);  int j=0;  int mod2=mod;  int first = h[mod]->key;  while(h[mod]->key!=INT\_MIN){  if(h[mod]->key==key) return mod;  j++;  //printf("not at %d \t",mod);  mod = linearprob(mod2,key,j);  if(mod==mod2) break;  }  return -1;  }  hashtable delete3(hashtable h, int key){  int mod = find3(h,key);  if (mod ==-1) return h;  printf("%d here",mod);  h[mod]->key=0;  return h;  }  hashtable insertlist3(hashtable bn, int a[],int size){  int i;  for(i=0;i<size;i++){  printf("inserting %d",a[i]);  bn=add3(bn,a[i]);  }  return bn;  }  **//Cuckoo hahsing**  void swap(int\* a, int\* b){  int temp = \*a;  \*a = \*b;  \*b =temp;  }  void add4 (hashtable\* h,hashtable\* h2,int key){  int mod = hashfunc(key);  if ((\*h)[mod]->key==INT\_MIN || (\*h)[mod]->key==0) {  (\*h)[mod]->key=key;  return;  }  else {  swap(&key,&(\*h)[mod]->key);  int mod2 = hashfunc8(key); // use hashfunc8 for no collision // use hashfunc9 for collison and rehashisng  printf("\nin table2 %d of key %d\n",mod2,key); //for rehashing copy the code from add3 suitably.  if ((\*h2)[mod2]->key==INT\_MIN || (\*h2)[mod2]->key==0 ){  (\*h2)[mod2]->key=key;  return;  }  swap(&key,&(\*h2)[mod2]->key);  add4(h,h2,key);  }    }  void insertlist4(hashtable\* h,hashtable\* h2, int a[],int size){  int i;  for(i=0;i<size;i++){  printf("inserting %d\n",a[i]);  add4(h,h2,a[i]);  printhash(\*h,10);  printhash(\*h2,10);  }  return ;  }  int find4(hashtable\* h,hashtable\* h2,int key){  int mod = hashfunc(key);  int mod2 = hashfunc8(key);  if ((\*h)[mod]->key==key || (\*h2)[mod2]->key==key) return 1;  return -1;  }  **//Bloom Filters**  int bloomhashfunc(int key){  return (5\*key)%47;  }  int bloomhashfunc2(int key){  return (key+hashfunc(key))%47;  }  int bloomhashfunc3(int key){  return (key+hashfunc2(key))%47;  }  int bloomhashfunc4(int key){  return (key+hashfunc3(key))%47;  }  hashtable insertbloom(hashtable h, int a[],int size){  int i;  for(i=0;i<size;i++){  printf("inserting %d\t",a[i]);  int mod1 = bloomhashfunc(a[i]);  int mod2 = bloomhashfunc2(a[i]);  int mod3 = bloomhashfunc3(a[i]);  int mod4 = bloomhashfunc4(a[i]);  h[mod1]->key=1;  h[mod2]->key=1;  h[mod3]->key=1;  h[mod4]->key=1;  printf("MOD: %d %d %d %d\n",mod1,mod2,mod3,mod4);  }  return h;  }  int findbloom(hashtable h, int key){  int i;  printf("fidning %d\t",key);  int mod1 = bloomhashfunc(key);  int mod2 = bloomhashfunc2(key);  int mod3 = bloomhashfunc3(key);  int mod4 = bloomhashfunc4(key);  printf("MOD: %d %d %d %d\n",mod1,mod2,mod3,mod4);  if (h[mod1]->key==1 && h[mod2]->key==1 && h[mod3]->key==1 && h[mod4]->key==1 ) return 1;  return 0;  }  int main(int argc, char \*argv[]) {  char b[5][10]={"roht","hat","rat","ooty","thor"};  insertlist2(h2,b,5);  printf("mia %s\n",h2[0]->key);  printhash2(h2,100);  printf("\n Open addressing and Rehahsinhg\n");  hashtable h3 = create(10);  int c[]={45,23,24,57,90,33,88,23,7,11,20,32,43,69};  h3=insertlist3(h3,c,12);  printhash(h3,m);  printf("FInd : %d",find3(h3,33));  h3 = delete3(h3,33);  printhash(h3,m);  printf("FInd : %d",find3(h3,33));  add3(h3,43);  printhash(h3,m);  printf("\n Cuckoo HAshing\n");  hashtable h4 = create(10);  hashtable h5 = create(10);  int d[]={45,23,24,57,90,33,88,7,11,20,32,43,69};  insertlist4(&h4,&h5,d,12);  printhash(h4,10);  printhash(h5,10);  printf("\n FInd: %d",find4(&h4,&h5,26));  printf("\n Bloom Filters\n");  hashtable h6 = create(47);  int e[]={45,23,24,57,90,33};  h6=insertbloom(h6,e,6);  printhash(h6,47);  printf("%d",findbloom(h6,34));  return 0;  }  **LINUX**  struct treenode;  typedef struct treenode \*tree;  struct treenode{  char\* dir;  tree\* child;  int nc;  int filled;  };  tree createtree(int nc,char root[20]){  tree t = (tree)malloc(sizeof(struct treenode));  t->child = (tree \*)malloc(nc\*sizeof(tree));  t->nc=nc;  t->dir=(char\*)malloc(sizeof(char));  t->dir=root;  t->filled=0;  //printf("%d %d\n",root,t->dir);  //printf("%c %c\n",root[0],\*((t->dir)+1));  //printf("%d %d %d %d \n",sizeof(t->dir),sizeof(t->child),sizeof(t),sizeof(root));  int i=0;  for(i;i<nc;i++){  t->child[i]=NULL;  }  return t;  }  int haschild(tree par,tree child){  int i=0;  printf("Comparing %s in %s %d\n",child->dir,par->dir,par->filled);  for(i;i<par->filled;i++){  if(strcmp(par->child[i]->dir,child->dir)==0){  printf("already there %s\n",par->child[i]->dir);  return 1;  }  }  return 0;  }  tree getchild(tree par,tree child){  int i=0;  //printf("Comparing %s in %s %d\n",child->dir,par->dir,par->filled);  for(i;i<par->filled;i++){  if(strcmp(par->child[i]->dir,child->dir)==0){  //printf("already there %s\n",par->child[i]->dir);  return par->child[i];  }  }  return NULL;    }  tree insert(tree par,tree child){  if (haschild(par,child)){  return getchild(par,child);  }  par->child[par->filled]=child;  par->filled++;  printf("Returning %s with filled %d\n",par->dir,par->filled);  return child;  }  **void readdata(int N,tree t){**  **int i=0;**  **for(i;i<N;i++){**  **char\* a=(char\*)malloc(sizeof(char));**  **scanf("%s",a);**  **printf("A:::%d %s\n",a,a);**  **int len =strlen(a);**  **int j=0,k=0;**  **tree root=t;**  **for(j;j<len;j++){**  **char\* temp=(char\*)malloc(sizeof(char));**  **k=0;**  **while(a[j]!='/'&&a[j]!='\0'){**  **temp[k++]=a[j];**  **j++;**  **}**  **temp[k]='\0';**  **if(strcmp(temp,"")!=0){**  **//printf("%d",strcmp(temp,""));**  **printf("Temp:::%d %s\n",temp,temp);**  **tree t1=createtree(100,temp);**  **root=insert(root,t1);**  **}**  **}**    **}**  **return;**  **}**  void search(tree t,char\* a){  int len =strlen(a);  int j=0,k=0;  tree root=t;  for(j;j<len;j++){  char\* temp=(char\*)malloc(sizeof(char));  k=0;  while(a[j]!='/'&&a[j]!='\0'){  temp[k++]=a[j];  j++;  }  temp[k]='\0';  if(strcmp(temp,"")!=0){  printf("Temp:::%d %s\n",temp,temp);  tree t1=createtree(100,temp);  if (haschild(root,t1)){  root=getchild(root,t1);  }  else{  printf("\nNot found\n");  return;  }  }  }  printf("FOund");  }  int main(){  char\* a=(char\*)malloc(sizeof(char));  int c;  scanf("%d",&c);  scanf("%s",a);  **//getchar();**  printf("%s",a);  tree t=createtree(100,a);  printf("reading data\n");  readdata(4,t);  printf("searching data\n");  char \*sea="dev/bin";  search(t,sea); |