Algoritmica LAB

Algoritmi made by
Calogero Turco && Zicco

//1.1 Funzione Swap di Interi

swap(&x,&y);//chiamata

```
void swap(int *p,int *q){//swap
int tmp=*p;
*p=*q;
*q=tmp;
}
```

Swap3(&x,&y,&z);

```
void swap3(int *p,int *q,int *z){
int tmp;
tmp=*p;
*p=*z;
*z=*q;
*q=tmp;
}
```

//1.2 inverti array

```
void invert (int a[],int dim){
int tmp;
int i;
for(i=0;i<((dim/2)+1);i++){
    tmp=a[i];
    a[i]=a[dim-i];
    a[dim-i]=tmp;
   }
}
```

//1.3 Max tra due interi

```
int max(int a,int b){
if (a>b)
return a;
if(a<b)
return b;
return a;
}
```

//1.4 Min tra due interi

```
int min(int a,int b){
 if (a<b)
return a;
 if(a>b)
return b;
return a; }
```

//1.5confronto lessicografico

```
int main (){
    char s1[DIM];
    char s2[DIM];
     scanf("%s",s1);
     scanf("%s",s2);
    Int i=mystrcmp(s1,s2);
    switch (i){
    case(1):
            printf("+%d",i);
            break;
    case(-1):
            printf("%d",i);
            break;
    case(0):
           printf("%d",i);
           break;
    }printf("\n");
    return 0;
}
```

int mystrcmp(char *string1,char *string2){

```
int i;
while(string1[i]!='\0' && string2[i]!='\0'){
      if(string1[i]==string2[i])
       i++;
      else if(string1[i]>string2[i])
       return 1;
       else return -1;
}
       if(string1[i]=='\0' && string2[i]=='\0')
       return 0;
       if (string1[i]!='\0')  return 1;
       return -1;
}
```

//1.4intersezione due array ordinati

int intersz(int a[],int b[],int d1,int d2){

```
 int k,i,j;
 k=i=j=0;
 while(i<dim1&&i<dim2){
```

```
    if(a[i]==b[j]){
      k++;
      i++;
      j++;
     }
    else if(a[i]<b[j]){
       i++;
     }
     else
       j++;
     }
     return k;
}
```

```
c=malloc(d3*sizeof(int));
while(k<d3){
   if(i<d1 && j<d2){
        if(a[i]==b[j]){
         c[k]=a[i];
         k++;
         j++;
         i++;
         }
         else if(a[i]<b[j]){
          c[k++]=a[i++];
         }
      else c[k++]=b[j++];
   }
   else if(i<dim1)   c[k++]=a[i++];
   else c[k++]=b[j++];
  }
return c;}
```

Algoritmi Teoria

//2.1 INSERTION SORT SU STRINGHE

```
void insort(char **a,int dim){
    int i=0;
    int j=0;
    char *key;
for(i=1;i<dim;i++){
    key=a[i];
    j=i-1;
while(j>=0 && strcmp(a[j],key)>0){
    a[j+1]=a[j];
    j--;
   }
  a[j+1]=key;
  }
 }
```

//2.2 Ricerca Binaria su Stringhe

```
int ricerca(char **a,int ini,int dim,char *x){
      if(ini>dim)
         return -1;
       else{
       int cx=(ini+dim)/2;
       int y=strcmp(a[cx],x);
       if(y==0)
         return cx;
        else if(y>0)
            return ricerca(a,ini,cx-1,x);
               else return ricerca(a,cx+1,dim,x);
    }
 }
```

//1.5 Unione di array ordinati

```
int* unione(int a[],int b[],int d1,int d2,int d3){
int i,k,j;
i=k=j=0;
int *c;
```

//2.3 MergeSort

```
int main () {
  int a[DIM];
  for(int i=0;i<DIM;i++)   scanf("%d",&a[i]);
  mergesort(a,0,DIM-1);
```

```
  printf("\n");
  for(int i=0;i<DIM;i++)  printf("%d\n",a[i]);
  return 0;
}
void mergesort (int a[],int p,int dim){
    if(p<dim){
         int m = (p+dim)/2;
         mergesort(a,p,m);
         mergesort(a,m+1,dim);
         merge(a,p,m,dim);
          }
     }
void merge(int a[],int p,int q,int r){
   int dim1=q-p+1;
   int dim2=r-q;
   int *sx; int *dx;
   sx=(int*)malloc(dim1*sizeof(int));
   dx=(int*)malloc(dim2*sizeof(int));
   int i,j,k;
   i=j=k=0;
   for(i=0;i<dim1;i++){
         sx[i]=a[p+i];
   }
   for(j=0;j<dim2;j++){
         dx[j]=a[q+j+1];
   }
   i=j=0;
   for(k=p;k<=r;k++){
        if(j>=dim2){
          a[k]=sx[i];
          i++;
         }
         else if(i>=dim1){
            a[k]=dx[j];
             j++;
          }
         else if(sx[i]<=dx[j]){
            a[k]=sx[i];
             i++;
          }
else {
           a[k]=dx[j];
           j++;
          }
     }
}
```

//2.4 QuickSort interi

```
void quicksort(int a[],int q,int r){
  if(q<r){
     int p=partizione(a,q,r);
     quicksort(a,q,p-1);
     quicksort(a,p+1,q);
    }
}

int partizione(int a[],int q,int r){
int pivot=r-1;
int j=0;
int i=q-1;
int temp=0;
for(j=q;j<r-1;j++){
5 | Pag.if (a[j]<=a[pivot]){
temp=a[++i];
a[i]=a[j];
a[j]=temp;
}
else j++;
}
temp=a[i++];
a[i]=a[j+1];
a[j+1]=temp;
return i;
}
```

//3.1 QSort su stringhe

```
int compare(const void*a,const void*b) {
char **x = (char**)a;
char **y=(char**)b;
return strcmp(*y,*x);
}
int main (){
int n;
char **s;
int i;
scanf("%d",&n);
s=(char**)malloc(n*sizeof(char*));
scanf("%d",&n);
for(i=0;i<n;i++){
s[i]=(char*)malloc(DIM*sizeof(char));
scanf("%s",s[i]);
```

```
}
qsort(s,n,sizeof(char*),compare);
for(i=0;i<n;i++)
printf("%s\n",s[i]);
return 0;
}
```

//3.2 QSort su StringheStruct

```
typedef struct stlengh1{
char *a;
int len;
}stlen;
int compare(const void*a,const void*b) {
stlen *x = (stlen*)a;
stlen *y=(stlen*)b;
if((x->len)==(y->len))  return strcmp(x->a,y->a);
if((x->len)>(y->len))  return 1;
else return -1;
}
int main (){
stlen *arr;
int n,i;
scanf("%d",&n);
arr=(stlen*)malloc(n*sizeof(stlen));
for(i=0;i<n;i++){
    arr[i].a=(char*)malloc(DIM*sizeof(char));
    scanf("%s",arr[i].a);
    arr[i].len=strlen(arr[i].a);
}
     qsort(arr,n,sizeof(stlen),compare);
     for(i=0;i<n;i++) printf("%s\n",arr[i].a);
     return 0;
}
```

//SP.1Three Way Qsort

```
#include <stdlib.h>
#include <stdio.h>
void swap(int *p,int *q){
int tmp=*p;
*p=*q;
*q=tmp;
}
void distribuzione(int a[], int sx, int px, int dx,int
*left,int *right)
{
int i,j,k;
swap(&a[px],&a[sx]);
k=sx;
j=dx;
for(i=sx;i<=j;i++){
if(a[i]<a[k]){
swap(&a[i],&a[k]);
k++;
}
else if (a[i]>a[k]){
swap(&a[i],&a[j]);
j--;
i--;
}
}
*left=k;
*right=j;
}
void quicksort( int a[], int sx, int dx) {
int pivot;
int tiny,big;
if( sx < dx ) {
pivot = (rand() % (dx-sx+1) )+sx;
distribuzione(a, sx, pivot, dx,&tiny,&big);
quicksort(a, sx, tiny-1);
quicksort(a, big+1, dx);
}}quicksort(A, 0, n-1);
```

//SP.2COUNT SORT

```
int* countsort(int a[],int n,int k);
int main(){
   int n,i;
   int *a;
   int *b;
   int k=0;
   scanf("%d",&n);
   printf("insert the interval from 0 to ...");
   scanf("%d",&k);
   a=(int*)malloc(n*sizeof(int));
   for(i=0;i<n;i++) scanf("%d",&a[i]);
   b=countsort(a,n,k);
   printf("\n");
   for(i=0;i<n;i++) printf("%d\n",b[i]);
   return 0;
}
```

```
int* countsort(int a[],int n,int k){
  int *c;
  int i;
  int *b;
  b=malloc(n*sizeof(int));
  c=(int*)malloc((k+1)*sizeof(int));
  for(i=0;i<k;i++)  c[i]=0;
  for(i=0;i<n;i++)  c[a[i]]++;
  for(i=1;i<=k;i++)  c[i]+=c[i-1];
  for(i=(n-1);i>=0;i--){
      b[c[a[i]]-1]=a[i];
      c[a[i]]--;
  }
  return b;
}
```

//SP.3HEAP

```
int parent(int i){
return (i/2);
}
int left(int i){
return ((i*2)+1);
}
int right (int i){
return ((i*2)+2);
}
int main () {
int n;
int *a;
int i;
scanf("%d",&n);
a=(int*)malloc(n*sizeof(int));
for(i=0;i<n;i++) scanf("%d",&a[i]);
heapsort(a,n);
buildmaxheap(a,0,n-1);
for(i=0;i<n;i++) printf("%d ",a[i]);
printf("\n");
return 0;
}
void maxheapify(int a[],int sx,int hpsize){
int largest;
int l=left(sx);
int r=right(sx);
if(l<=hpsize && a[sx]<a[l])
largest=l;
else largest=sx;
if(r<=hpsize && a[largest]<a[r])
largest=r;
if(largest!=sx){
swap(&a[sx],&a[largest]);
maxheapify(a,largest,hpsize);
}
}
void buildmaxheap(int a[],int sx,int hpsize){
int i;
for(i=hpsize/2;i>=0;i--){
maxheapify(a,i,hpsize);
}
}
void heapsort(int a[],int n){
int i,scambio;
    buildmaxheap(a,dim);
    i=dim;
    do{
      scambio=a[i];
      a[i]=a[0];
      a[0]=scambio;
      dim--;
      maxheapify(a,0,dim);
      i--;
     }while(i>0);
    }
}
```

//Qsort 3.3 su struct punti

```
typedef struct points{
int x;
int y;
}point;
int cmp(const void *a,const void *b){
point *p=(point*)a;
point *q=(point*)b;
if(p->x!=q->x) return (p->x-q->x)
else return ((q->y) - (p->y));
}
int main(){
point *a;
int n;
int i;
scanf("%d",&n);
a=(point*)malloc(n*sizeof(point));
```

```
for(i=0;i<n;i++){
scanf("%d",&(a[i].x));
scanf("%d",&(a[i].y));
}
qsort(a,n,sizeof(point),cmp);
for(i=0;i<n;i++){
printf("%d ",a[i].x);
printf("%d",a[i].y);
printf("\n");
}
return 0;
}
```

//3.4compare su interi per qsort

```
int compare (const void*a,const void*b){
int v = *( (int*)a );
int v2 = *( (int*)b );
int rs = abs(v % 2);
int rs2 = abs(v2 % 2);
if (rs != rs2) return rs - rs2;
else if (rs == 0) return v - v2;
else return v2 - v;
}
```

//PARI CRESCENTI , DISPARI DECRESCENTI
prima pari , poi I dispari , se voglio il contrario ,
inverto (rs!=rs2) con (rs==0);

//4-PRIME K STRINGHE PIU' FREQUENTI

```
typedef struct {
  char parola[100];
  int count;
} elenco;
int ricerca(elenco* v,char* key,int dim){
int i;
for(i=0;i<dim;i++){
   if(strcmp(v[i].parola,key)==0) return i;
 }
 return -1;
}
int compare(const void* a,const void* b){
  elenco v1=*(elenco *)a;
  elenco v2=*(elenco *)b;
  return v2.count-v1.count;
}
```

```
int compare2(const void* a,const void* b){
  elenco v1=*(elenco *)a;
  elenco v2=*(elenco *)b;
  return strcmp(v1.parola,v2.parola);
}

void funct(elenco *a,int *dim,int *k){
  int i,j,res;
  j=0; //POS ULTIMA STRINGA//
  char temp[100];
  scanf("%d",dim);
  scanf("%d",k);
  a=(elenco*) malloc (*dim * sizeof(elenco));
  for(i=0;i<*dim;i++){
       scanf("%s",temp);
       res=ricerca(a,temp,*dim);
       if(res==-1) {
       strcpy(a[j].parola,temp); //non c'è l'aggiungo
        a[j].count=1;
        j++;
        }
       else  a[res].count++; //c'è gia , incremento
      }
  qsort(a,j,sizeof(elenco),compare);
  qsort(a,*k,sizeof(elenco),compare2);
  print_output(a,*k);
}
```

```
int main(){
  int i,k,dim;
  elenco* array;
  funct(array,&dim,&k);
}
```

//4.2 ORDINAMENTO PER ANAGRAMMA PRINCIPALE E CRESCENZA

```
typedef struct stringanagramma{
    unsigned char* string;
    int *caratteri;
    int len;
}stgram;
int main(){
stgram *b;
int N,i;
scanf("%d",&N);
```

```
b=(stgram*)malloc(N*sizeof(stgram));
for(i=0;i<N;i++){
b[i].string=(unsigned char*)malloc(MAXLEN*
sizeof(unsigned char));
scanf("%s",b[i].string);
b[i].len=strlen((char*)b[i].string);
b[i].caratteri=(int*)malloc((b[i].len)*sizeof(int));
stringchar(b[i].string,b[i].caratteri,b[i].len);
qsort(b[i].caratteri,b[i].len,sizeof(int),cmp1);
}
qsort(b,N,sizeof(stgram),cmp2);
print(b,N);
return 0;
}
void stringchar(unsigned char *s,int a[],int dim){
int i;
for(i=0;i<dim;i++) a[i]=(int)s[i];
}
int cmp1(const void *x,const void *y){
int a=*(int*)x;
int b=*(int*)y;
return a-b;
}
int cmp2(const void *x,const void *y){
stgram *a=(stgram*)x;
stgram *b=(stgram*)y;
int i;
if(a->len==b->len){

for(i=0;i<a->len;i++){
if(a->caratteri[i]<b->caratteri[i])
return -1;
else if(a->caratteri[i]>b->caratteri[i])
return 1;
}
return strcmp((char*)a->string,(char*)b->string);
}
else if (a->len<b->len){
for(i=0;i<a->len;i++){
if(a->caratteri[i]<b->caratteri[i])
return -1;
else if(a->caratteri[i]>b->caratteri[i])
return 1;
}
return -1;
}
else if (a->len>b->len){
for(i=0;i<a->len;i++){
if(a->caratteri[i]<b->caratteri[i])
return -1;
else if(a->caratteri[i]>b->caratteri[i])
return 1;
}
return 1;
}
return 0;
}
void print(stgram *a,int N){
int i=0;
printf("%s ",a[i].string);
for(i=1;i<N;i++){
if(a[i].len==a[i-1].len &&
arraycompare(a[i].caratteri,a[i-
1].caratteri,a[i].len))
printf("%s ",a[i].string);
else printf("\n%s ",a[i].string);
}
printf("\n");
}
int arraycompare(int *x,int *y,int dim){
for(int i=0;i<dim;i++){
if(x[i]!=y[i])
return 0;}return 1;}
```

//4.3 COLORI DISTINTI

```
typedef struct colors{
      int x;
      int y;
      int c;
} colori;
typedef struct interrogazione{
     int x1;
     int y1;
     int x2;
     int y2;
     int ris;
}query;
typedef struct interisultato{
     int occ;
     int colore;
```

```
}qures;
int cmp(const void *x,const void *y){
colori *a=(colori*)x;
colori *b=(colori*)y;
return (a->c)-(b-> c);
}
int coloricalc(colori *a,int dim);
void trovacolori(colori *a,int n,query *qu,int m);
int main (){
int N;
int M;
colori *a;
query *qu;
int i;
scanf("%d",&N);
scanf("%d",&M);
a=malloc(N*sizeof(colori));
for(i=0;i<N;i++){
scanf("%d",&a[i].x);
scanf("%d",&a[i].y);
scanf("%d",&a[i].c);
}
qu=(query*)malloc(M*sizeof(query));
for(i=0;i<M;i++){
scanf("%d",&qu[i].x1);
scanf("%d",&qu[i].y1);
scanf("%d",&qu[i].x2);
scanf("%d",&qu[i].y2);
qsort(a,N,sizeof(colori),cmp);
trovacolori(a,N,qu,M);
for(i=0;i<M;i++){
printf("%d\n",qu[i].ris);
}
return 0;
}
void trovacolori(colori *a,int n,query *qu,int m){
int i;
int j;
int lastcolor;
for(i=0;i<m;i++){
lastcolor=-1;
qu[i].ris=0;
for(j=0;j<n;j++){
if(qu[i].x1<=a[j].x && a[j].x<=qu[i].x2){
if(qu[i].y1<=a[j].y && a[j].y<=qu[i].y2){
if(lastcolor!=a[j].c){
qu[i].ris++; // nel primo ciclo parte da 1
perché è sempre positivo(quindi sempre diverso
da -1)
lastcolor=a[j].c;//se incontro un colore
diverso dall'ultimo aggiorno(gli uguali saranno
sempre vicini)
}}}}}}
```

ALGORITMI LISTE MONODIREZIONA

5-1.Genera Lista Ordinata

```
void add_ordered (NodeList *head,int v)
{
NodeList new_val=malloc(sizeof(Node));
NodeList prev=NULL;
NodeList cur=*head;
new_val->value=v;
while (cur != NULL && cur->value < v){
prev=cur;
cur=cur->next;}
if (prev != NULL){
prev->next=new_val;
new_val->next=cur;}
else{
add_h(head,v);
}
```

Add Head

```
void add_h (NodeList *head,int v){
NodeList new_val=malloc(sizeof(Node));
new_val->next=*head;
new_val->value=v;
*head=new_val;
}
```

5.2 Add Tail

```
void add_tail (NodeList *head,int v){
```

```
NodeList cur=*head;
NodeList new_val=malloc(sizeof(Node));
new_val->value=v;
new_val->next=NULL;
if (*head==NULL)
*head=new_val;
else{
while (cur->next != NULL){
if (cur->next != NULL){
cur=cur->next;}}
cur->next=new_val;
}
}
```

5.3 Rimuovi Duplicati

```
void delDupl(node *head){
node *x=head;
node *prev;
node *cur;
while (x!=NULL){
cur=x->next;
prev=x;
while (cur!=NULL){
if (cur->key==x->key){
node *tmp=cur;
prev->next=tmp->next;
cur=cur->next;
free(tmp);}
else{
prev=cur;
cur=cur->next;}}
x=x->next;}
}
```

5.4 Inverti Lista

```
void inverti_lista(Listadielementi* list)
{
Listadielementi aux=*list;
Listadielementi aux2;
*list=NULL;
while (aux!=NULL){
aux2=aux;
aux=aux->next;
aux2->next=*list;
*list=aux2;}
}
```

5.5 Remove N Nodes:

```
void drop_n (ListEl *head,int n){
ListEl cur=*head;
ListEl tmp;
int i=0;
if (*head!=NULL){
while (i<n && (*head)!=NULL){
tmp=*head;
free(tmp);
*head=(*head)->next;
i++;}}
}
```

5-6 Append

```
void add_tail (NodeList *head,int v){
NodeList cur=*head;
NodeList new_val=malloc(sizeof(Node));
new_val->value=v;
new_val->next=NULL;
if (*head==NULL)
*head=new_val;
else{
while (cur->next != NULL){
if (cur->next != NULL){
cur=cur->next;}}
cur->next=new_val;
}
}
```

5-7 Remove multiples

```
void del_multiple (ListEl *head,int v){
ListEl tmp;
if (*head!=NULL){
if (multiple((*head)->value,v)){
tmp=*head;
*head=tmp->next;
free(tmp);
del_multiple(&(*head),v);}
else
del_multiple(&(*head)->next,v);}
}
```

5-8 Member

```
int member (NodeList head,int el)
{
NodeList cur=head;
int ok=0;
while (cur != NULL && ok==0){
if (cur->value==el){
ok=1;}
else
cur=cur->next;}
return ok;
```

5-9 PrintList

```
void print_list (NodeList head){
NodeList cur=head;
while (cur != NULL){
printf("%d\n",cur->value);
cur=cur->next;}
}
```

5-10 ADD-after 4th

```
void add_4 (ListEl head,int el){
int i;
if (lenght_REC(head) <= 4)
add_tail(&head,el);
else{
ListEl new=malloc(sizeof(Node));
ListEl cur=head;
new->value=el;
ListEl tmp;
for (i=0;i<3;i++){
cur=cur->next;}
tmp=cur->next;
cur->next=new;
new->next=tmp;}
}
```

5-11 Remove Head

```
void del_head (ListEl *head){
ListEl tmp;
if (*head!=NULL){
if ((*head)->next==NULL){
tmp=*head;
*head=NULL;
free(tmp);}
else{
tmp=*head;
*head=(*head)->next;
free(tmp);}}
}
```

5-12 Lenght Rec

```
int lenght (ListEl head){
int len=0;
if (head!=NULL)
len=1+lenght(head->next);
return len;
}
```

5-13 Move to Front Mono

```
typedef struct link{

int info;

   struct link* next;
   struct link* prec;
```

```
   int freq;
}element;
int main(){
int i,n,k;
int occ=0;
element *head;
head=NULL;
scanf("%d",&n);
for(i=0;i<n;i++){
scanf("%d",&k);
insert(&head,k);
}
tunefre(&head); //SETTA A 0 LE OCC
while(occ!=-1){
scanf("%d",&k);
occ=movetofront(&head,k);
printf("%d\n",occ);
}
return 0;
}
int movetofront(element **head,int k){
int distance=0;
element *corr;
corr=*head;
while(corr!=NULL && corr->info!=k ){
corr=corr->next;
distance++;
}
if(corr==NULL)
return -1;
corr->freq++;
shift(head,&corr);
return distance;
}

void shift(element **head,element **el){
element *node;
node=*el;
if(node->prec==NULL){
*head=node;
return;
}
if((node->prec->freq)<(node->freq)){
element *precedent=node->prec;
precedent->next=node->next;
if(node->next!=NULL){
node->next->prec=precedent;
}
node->next=precedent;
if(precedent->prec==NULL){
node->prec=NULL;
}
else if(node->prec->prec!=NULL){
node->prec->prec->next=node;
node->prec=node->prec->prec;
}
precedent->prec=node;
shift(head,&node);
}}
```

//6.1 Ricerca ABR

```
typedef struct _node{
int key;
struct nodes* left;
struct nodes* right;
}node;
int main(){
node *T;
T=NULL;
int n,i,k;
int result;
scanf("%d",&n);
for(i=0;i<n;i++){
     scanf("%d",&k);
     insert(&T,k);
}
scanf("%d",&i);
while(i>=0){
     result=search(T,i,0);
     if(result>=0) printf("%d\n",result);
     else printf("NO\n");
     scanf("%d",&i);
}
libera(T);
return 0;
}
```

```
void insert(node **t,int k){
if(*t==NULL){
  node* new=malloc(sizeof(node));
  new->key=k;
  new->right=new->left=NULL;
  *t=new;
  return;
}
if((*t)→key<k) insert(&(*t)->right,k);
else insert(&(*t)->left,k);
}

int search(node *t,int i,int prof){
if(t==NULL){
return -1;
}
if(t->key==i){
return prof;
}
if(t->key>i)
return search(t->left,i,prof+1);
else return search(t->right,i,prof+1);
}
void libera(node *t){
if(t!=NULL){
     libera(t->left);
     libera(t->right);
     free(t);
   }
}
```

//6.2 Altezza Albero

```
int altezza(node*t){
if(t==NULL){
return 0;
}
else {
       int altsx=altezza(t->left);
       int altdx=altezza(t->right);
       int h=max(altsx,altdx);
       return 1+h;
    }}
```

//6.4 Ricerca elemento k

```
int ricerca(node *t,node *t2,int k){
if(t==NULL && t2==NULL) return 1;
if(t==NULL && t2!=NULL) return 0;
if(t!=NULL && t2==NULL) return 0;
else{
    if(t->key!=t2->key)
return 0;
else{
      if(t->key>k)
          return ricerca(t→left,t2→left,k);
      else if(t->key<k)
           return ricerca(t->right,t2->right,k);
      else if(t->key==k)
           return 1;
       }
   }
return 0;
}
```

6.5 Convert Tree to Array

```
int main(){
node *T;
T=NULL;
int n,i,k;
scanf("%d",&n);
for(i=0;i<n;i++){
   scanf("%d",&k);
   insert(&T,k);
     }
   arraytree(T);
}

int dim(node*t){
if(t!=NULL){
return 1+dim(t->left)+dim(t->right);
}
return 0;  // CALCOLA DIM ALBERO
}

int* arraytree(node*t){
int n=dim(t);
int i;
int *a=malloc(n*sizeof(int));
arraytree_aux(t,a,-1);
for(i=0;i<n;i++){
printf("%d\n",a[i]);
```

```
}
return a;
}

int arraytree_aux(node *t,int *a,int i){
if(t==NULL){
return i;
}
int index;
index=arraytree_aux(t->left,a,i);
++index;
a[index]=t->key;
index=arraytree_aux(t->right,a,index);
return index;
}

//6.5 albero ternario
typedef struct nodes {
   int key;
   struct nodes* left;
   struct nodes* center;
   struct nodes* right;
}node;

void insert(node **t,int k){
if(*t==NULL){
node* new=malloc(sizeof(node));
new->key=k;
new->right=new->left=NULL;
*t=new;
return;
}
if((*t)->key>k)
insert(&(*t)->left,k);
else if((k%((*t)->key))!=0)
insert(&(*t)->right,k);
else insert(&(*t)->center,k);
}

int conta(node *t){ //conta le foglie
if(t==NULL)
return 0;
if(t->right!=NULL && t->left!=NULL && t->center!=NULL){
return 1+conta(t->right)+conta(t->left)+conta(t->center);
}
else return conta(t->right)+conta(t->left)+conta(t->center);
}
```

//7.1 PRDYN Taglio corda

```
void tagliocorda(int *p,int *r,int *s,int n);
void printbest(int *s,int n);
int main(){
int n,i;
int *p,*s,*r;
scanf("%d",&n);
p=malloc((n+1)*sizeof(int));
s=malloc((n+1)*sizeof(int));
r=malloc((n+1)*sizeof(int));
p[0]=0;
r[0]=0;
for(i=1;i<=n;i++){
      scanf("%d",&p[i]);
      r[i]=p[i];
      s[i]=i;
}
tagliocorda(p,r,s,n);
printf("%d\n",r[n]);
printbest(s,n);
return 0;
}
void tagliocorda(int *p,int *r,int *s,int n){
int i,j,q
for(i=1;i<=n;i++){
      q=p[i];
      for(j=2;j<i;j++){
            if(q<p[j]+r[i-j]){
                  q=p[j]+r[i-j];
                  s[i]=j;
            }
       }
        r[i]=q;
   }
}
void printbest(int *s,int n){
if(n==0)
printf("0\n");
else{
int t=n;
while(t>0){
       printf("%d ",s[t]);
       t-=s[t];
     }
printf("\n");
}
```

//8.1 HASHTABLE

```
typedef struct s{
int key;
struct s* next;
}node;

void insert(int x,node **T){
node *new=(node*)malloc(sizeof(node));
new->key=x;
new->next=*T;
*T=new;
}
void print_list(node *T ){
if(T==NULL){
printf("NULL\n");
return;
}
else{
printf("%d,",T->key);
print_list(T->next);
}
}
int lenght(node* T){
if(T==NULL)
return 0;
return 1+lenght(T->next);
}
int hash(int x,int a,int b,int n){
int p=999149;
return ((a*x+b)%p)%(n*2);
}

int main(){
int n,a,b,i,x;
node **T;
int indice;
int maxlen=0;
int currlen=0;
int len=0;
scanf("%d",&n);
scanf("%d",&a);
scanf("%d",&b);
T=(node**)malloc(2*n*sizeof(node*));
for(i=0;i<2*n;i++){
      T[i]=NULL;
}
```

```
for(i=0;i<n;i++){
        scanf("%d",&x);
        indice=hash(x,a,b,n);
        insert(x,&T[indice]);
}
for(i=0;i<2*n;i++){
        currlen=lenght(T[i]);
        if(currlen>0)    len+=(currlen-1);
        if(currlen>maxlen)    maxlen=currlen;
        }
printf("%d\n",maxlen);
printf("%d\n",len);
return 0;
}
```

//10.1 DFS COMPLETA CON TIME E LETTURA GRAFO

```
typedef struct _edges{
int grado;
int *adiacenti;
}edges;
typedef struct _node{
int info;
struct _node* next;
}node;

void readgraph(edges **G,int n){
int i,g,j;
*G=(edges*)malloc(n*sizeof(edges));
for(i=0;i<n;i++){
     scanf("%d",&g);
     (*G)[i].grado=g;
     (*G)[i].adiacenti=(int*)malloc(g*sizeof(int));
     for(j=0;j<g;j++){
        scanf("%d",(*G)[i].adiacenti+j);
     }
   }
}
void printgraph(edges *G,int n){
int i,j;
for(i=0;i<n;i++){
printf("vertice : %d adj-> ",i);
for(j=0;j<G[i].grado;j++){
printf("%d -",*(G[i].adiacenti+j));
}
printf("\n"); }}

void DFS(edges *G,int n){
int *colors;
int *timestart;
int *timeend;
int i;
int time;
colors=malloc(n*sizeof(int));
timestart=malloc(n*sizeof(int));
timeend=malloc(n*sizeof(int));
for(i=0;i<n;i++){
colors[i]=0;
timestart[i]=-1;
timeend[i]=-1;
}
time=0;
for(i=0;i<n;i++){
if(colors[i]==0)
DFSVISIT(G,i,colors,timestart,timeend,&time);
}
for(i=0;i<n;i++){
printf("start %d,finish %d,color %d
/n",timestart[i],timeend[i],colors[i]);
}
}
void DFSVISIT(edges *G,int src,int *colors,int
*timestart,int *timeend,int *time){
int i,v;
timestart[src]= (++*time);
colors[src]=1;
for(i=0;i<G[src].grado;i++){
v=G[src].adiacenti[i];
if(colors[v]==0)
DFSVISIT(G,v,colors,timestart,timeend,time);
}
colors[src]=2;
timeend[src]=(++*time);
}
int main(){
edges *G;
int n;
scanf("%d",&n); //numero di vertici
readgraph(&G,n);
printgraph(G,n);
DFS(G,n);
return 0;}
```

// 10.2 GRAFO BIPARTITO

```
int bipartito(edges *E,int n){
int *colore;
int i;
 colore=(int*)malloc(n*sizeof(int));
 colore[0]=1;
for(i=1;i<n;i++){
        colore[i]=0;
}
for(i=0;i<n;i++){
     if(!DFS(E,i,colore))
        return 0;
    }
    return 1;
}
int DFS(edges *E,int src,int *colore){
int i;
int adj;
for(i=0;i<E[src].grado;i++){
    adj=E[src].adiacenti[i];
    if(colore[adj]==0){
        colore[adj]=-colore[src];
         DFS(E,adj,colore);
        }
        else if (colore[adj]==colore[src]){
                return 0;
     }
  }
return 1;
}
```

//10.3 Grafo Connesso

```
typedef struct graph{
     int grado;
     int* adiacenti;
     int colore;
}edges;
 int connesso(edges *E,int n){
  int src=0;
  int i;
  E[src].colore=1;
  DFS(E,src);
  for(i=0;i<n;i++){
  if(!E[i].colore)
  return 0;
}
return 1;
}
void DFS(edges *E,int src){
int i;
int el;
for(i=0;i<E[src].grado;i++){
      el=E[src].adiacenti[i];
      if(E[el].colore==0){
             E[el].colore=1;
              DFS(E,el);
        }
     }
}
```

//10.4 DISTANZA MINORE

```
int BFS(edges *E,int n,int src,int goal){
int *colors;
int *distance;
int i,u,v;
colors=malloc(n*sizeof(int));
distance=malloc(n*sizeof(int));
for(i=0;i<n;i++){
       colors[i]=0;
       distance[i]=-1;
}
colors[src]=1;
distance[src]=0;
coda q;
nit(&q,n);
accoda(&q,src);
while(!codavuota(&q)){
    u=decoda(&q);
    for(i=0;i<E[u].grado;i++){
        v=E[u].adiacenti[i];
        if(colors[v]==0){
                distance[v]=distance[u]+1;
                 if(v==goal){
                     return distance[v];
                   }
            colors[v]=1;
            accoda(&q,v);
         }
      }
}
deinit(&q);
return distance[goal];}
```

//10.5 SIMULAZIONE ESAME 2019-GRAFO SENZA CICLI

```
int verifica(edges *E,int n){
 int sum=0;
 int i;
 for(i=0;i<n;i++){
         sum+=E[i].grado;
  }
  if((sum/2)!=(n-1)){
    return 0;
  }
  int src=0;
  int *colori=malloc(n*sizeof(int));
  for(i=0;i<n;i++)  colori[i]=0;
  colori[src]=1;
  if(!DFS(E,src,src,colori)){
  return 0;
  }
return 1;
}
int DFS(edges *E,int src,int father,int *colori){
int i;
int el;
for(i=0;i<E[src].grado;i++){
     el=E[src].adiacenti[i];
     if(colori[el]==0){
        colori[el]=1;
        DFS(E,el,src,colori);
    }
    else if(el!=father){
          return 0;
     }
   }
return 1;
}
```