Algoritmi i strukture podataka

1)Stog realiziran statičkim poljem

```
/*Stog realiziran statičkim poljem*/
// #define MAXSTOG 5, npr.
//LIFO
//puni rastući
//vrh pokazuje na zadnju puni element



#define MAXSTOG 5

typedef struct{
	int vrh;
	int polje[MAXSTOG];
}Stog;

void init_stog(Stog *stog){
	stog->vrh=-1;
}

int dodaj_na_stog(Stog *stog, int element){
	if(stog->vrh>=MAXSTOG-1){ //ako je stog pun
		return 0;
	}
	else{
		stog->vrh++;
		stog->polje[stog->vrh]=element;
		return 1;
	}
}

int skini_sa_stoga(Stog *stog, int *element){
	if(stog->vrh<0){  //ako je sto prazan
		return 0;
	}
	else{
		*element=stog->polje[stog->vrh];
		stog->vrh--;
		return 1;
	}
}
```

2)Stog realiziran linearnom listom

```
/*Stog realiziran linearnom listom*/

struct at{
	int element;
	struct at* sljed;
};
typedef struct at atom;

typedef struct{
	atom *vrh;
}Stog;

void init_stog(Stog *stog){
	stog->vrh=NULL;
}


int dodaj_na_stog(Stog *stog, int element){
	atom *novi;
	if((novi=(atom*)malloc(sizeof(atom)))!=NULL){
		novi->element=element;
		novi->sljed=stog->vrh;
		stog->vrh=novi;
		return 1;
	}
	else{
		return 0;
	}
}

int skini_sa_stoga(Stog *stog, int *element){
	atom *p;
	if(stog->vrh!=NULL){  //ako je stog neprazan
		*element=stog->vrh->element;
		p=stog->vrh->sljed;
		free((void*)stog->vrh);
		stog->vrh=p;
		return 1;
	}
	else{
		return 0;
	}
}
```

3)Linearna jednostruko povezana lista

```
/*Linearna jodnostruko povezana lista*/

struct at{
	int element;
	struct at* sljed;
};
typedef struct at atom;

// Dodavanje u listu
// sortiranu po rastucoj vrijednosti elementa
// vraca 1 ako uspije, inace 0
int dodaj_u_listu(atom **glavap,int element){
	atom* novi,*p;
	if((novi=(atom*)malloc(sizeof(atom)))==NULL){
		return 0;
	}
	novi->element=element;
	if(*glavap==NULL || (*glavap)->element>=element){
		// Dodavanje na pocetak liste
		novi->sljed=*glavap;
		*glavap=novi;
	}
	else{
		// Dodavanje iza postojeceg elementa kad:
		// a) postojeći atom nema sljedećeg
		// b) element u sljedećem cvoru je veći ili jednak novome
		for(p=*glavap;p->sljed && (p->sljed)->element<element;p=p->sljed);
		novi->sljed=p->sljed;
		p->sljed=novi;
	}
	return 1;
}

int brisi_iz_liste(atom **glavap,int element){
	atom* p;
	for(;*glavap && (*glavap)->element!=element;glavap=&((*glavap)->sljed));
	if(*glavap!=NULL){
		//*glavap pokazuje na element koji se treba obrisati
		//(*glavap)->sljed == adesa na sljed. elem, spremljenau atmu koji se brise
		p=*glavap;
		*glavap=(*glavap)->sljed;
		free((void*)p);
		return 1;
	}
	else{
		return 0;
	}
}

atom *trazi_elem(atom *glava,int element){
	atom *p;
	for(p=glava;p!=NULL && p->element!=element;p=p->sljed);
	return p;
}

void ispisi_listu(atom *glava){
	atom *p;
	for(p=glava;p!=NULL;p=p->sljed){
		printf("%d ",p->element);
	}
}
```

4)Linearna dvostruko povezana lista

```
/*Linearna dvostruko povezana lista*/
//lista predstavlja red
//skidanje elementa se vrši na glavi
//dodavanje elementa se vrši na repu
//moguće brisanje elementa s zadanim ključem

struct at{
	int element;
	struct at *sljed;
	struct at *preth;
};
typedef struct at atom;
//atom *glava,*rep;

//dodavanje se vrsi na repu
int dodaj_u_listu(atom **glavap,atom **repp,int element){
	atom *novi;
	if((novi=(atom*)malloc(sizeof(atom)))!=NULL){
		novi->element=element;
		novi->sljed=NULL;
		novi->preth=NULL;
		if(*glavap==NULL){    //ako je lista prazna
			*glavap=novi;
			*repp=novi;
		}
		else{  //lista nije prazna
			novi->preth=*repp;
			*repp=novi;
		}
		return 1;
	}
	else{
		return 0;
	}
}

//skidanje s glave reda
int skini_iz_reda(atom **glavap,atom **repp,int *element){
	atom *stari;
	//ako lista nije prazna
	if(*glavap){
		*element=(*glavap)->element;
		if(*glavap==*repp){   //lista sadrži samo jedan element
			stari=*glavap;
			*glavap=NULL;
			*repp=NULL;
		}
		else{  //lista sadži više elemenata
			(*glavap)->sljed->preth=NULL;
			stari=*glavap;
			*glavap=(*glavap)->sljed;
		}
		free((void*)stari);
		return 1;
	}
	else{
		return 0;
	}
}
```

```
//brisanje elementa
int brisi_iz_reda(atom **glavap,atom **repp,int element){
	atom *p;
	if(*glavap){//ako red nije prazan
		for(p=*glavap;p && p->element!=element;p=p->sljed);
		if(p){		//ako element postoji
			if(p==*glavap){		//ako je elem na prvom mjestu
				*glavap=p->sljed;
				if(p->sljed){ //ako je prvi ali ne i jedini
					p->sljed->preth=NULL;
				}
				else{  //ako je jedini
					*glavap=NULL;
					*repp=NULL;
				}
			}
			else if(p==*repp){    //ako je zadnji, ali ne i jedini
				(*repp)->preth->sljed=NULL;
				*repp=(*repp)->preth;
			}
			else{  //nije ni zadnji ni prvi
				p->preth->sljed=p->sljed;
				p->sljed->preth=p->preth;
			}
			free((void*)p);
			return 1;
		}
	}
	return 0;
}


void ispis_reda(atom* glava){
	atom *p;
	for(p=glava;p;p=p->sljed){
		printf("%d ",p->element);
	}
}
```

5)Red realiziran statičkim poljem

```
/*Red realiziran statičkim poljem*/
// #define MAXRED 10,npr.
//red je pun ako je ((ulaz+1)%MAXRED==izlaz)
//red je prazan ako je ulaz==izlaz
/*algoritam za dodavanje i skidanje elemenata:
	1)povećaj
	2)skaliraj
	3)obavi operaciju
*/

#define MAXRED 10
#define N 10

typedef struct{
	int polje[MAXRED];
	int ulaz,izlaz;
}Red;

int dodaj_u_red(Red *red,int element){
	if((red->ulaz+1)%N == red->izlaz){ //ako je red pun
		return 0;
	}
	else{
		red->ulaz++;  //povecaj
		red->ulaz%=N;  //skaliraj
		red->polje[red->ulaz]=element;  //obavi operaciju
		return 1;
	}
}

int skini_iz_reda(Red *red,int *element){
	if(red->ulaz==red->izlaz){ //ako je red prazan
		return 0;
	}
	else{
		red->izlaz++;  //povecaj
		red->izlaz%=N;  //skaliraj
		*element=red->polje[red->izlaz];  //obavi operaciju
	}
}
```

6)Red realiziran linearnom listom

```
/*red realiziran listom*/
//dodajemo na ulaz
//skidamo na izlazu

struct at{
	int element;
	struct at *sljed;
};
typedef struct at atom;

typedef struct{
	atom *ulaz;
	atom *izlaz;
}Red;

//inicijalizacija reda
void init_red(Red *red){
	red->ulaz=NULL;
	red->izlaz=NULL;
}

int dodaj_u_red(Red *red, int element){
	atom *novi;
	if((novi=(atom*)malloc(sizeof(atom)))!=NULL){
		novi->element=element;
		novi->sljed=NULL;
		if(red->izlaz==NULL){ //ako je red prazan
			red->izlaz=novi;
		}
		else{ //nije prazan, dodaj na kraj
			(red->izlaz)->sljed=novi;
		}
		red->ulaz=novi;
		return 1;
	}
	else{
		return 0;
	}
}

//skini element s izlaza i obrisi atom
int skini_iz_reda(Red *red,int *element){
	atom *p;
	if(red->izlaz!=NULL){  //ako red nije prazan
		*element=red->izlaz->element;
		p=red->izlaz;
		red->izlaz=red->izlaz->sljed;
		free((void*)p);
		if(red->izlaz==NULL){  //ako je red sada ostao prazan
			red->ulaz=NULL;
		}
		return 1;
	}
	else{
		return 0;
	}
}
```

7)Binarno stablo

```
/*Binarno stablo*/
//cvor *korijen=NULL;
//lijevo idu manji elementi, desno veci

struct cv{
	int element;
	struct cv *lijevo;
	struct cv *desno;
};
typedef struct cv cvor;

cvor *noviElement(int element){
	cvor *novi;
	if((novi=(cvor*)malloc(sizeof(cvor)))!=NULL){
		novi->element=element;
		novi->desno=NULL;
		novi->lijevo=NULL;
		return novi;
	}
	else{
		return NULL;
	}
}
//korijen=dodaj(...)
cvor  *dodaj_u_stablo(cvor *korijen,int element){
	if(korijen==NULL){
		return noviElement(element);
	}
	else{
		if(element<=korijen->element){
			korijen->lijevo=dodaj_u_stablo(korijen->lijevo,element);
		}
		else{
			korijen->desno=dodaj_u_stablo(korijen->desno,element);
		}
		return korijen;
	}
}

int trazi(cvor *korijen,int element){
	if(korijen==NULL){
		return 0;
	}
	else if(element==korijen->element){
		return 1;
	}
	else{
		if(element<korijen->element){
			return trazi(korijen->lijevo,element);
		}
		else{
			return trazi(korijen->desno,element);
		}
	}
}
```

```
519  /*INORDER ispis stabla*/
520   /* Ispisuje se od najmanjeg do najvećeg elementa u stablu */
521   void ispisiInorder(cvor *korijen){
522          if(korijen!=NULL){
523                 ispisiInorder(korijen->lijevo);
524                 printf("\n%d",korijen->element);
525                 ispisiInorder(korijen->desno);
526          }
527   }
528
529   /*PREORDER ispis stabla */
530    void ispisiPreorder(cvor*korijen){
531          if(korijen!=NULL){
532                 printf("\n%d",korijen->element);
533                 ispisiPreorder(korijen->lijevo);
534                 ispisiPreorder(korijen->desno);
535          }
536
537   }
538
539    /*POSTORDER ispis stabla */
540   void ispisiPostorder(cvor *korijen){
541          if(korijen!=NULL){
542                 ispisiPostorder(korijen->lijevo);
543                 ispisiPostorder(korijen->desno);
544                 printf("\n%d",korijen->element);
545          }
546   }
547
```