

Algoritmi i strukture podataka

```
1  1)Stog realiziran statičkim poljem
2
3
4  /*Stog realiziran statičkim poljem*/
5  // #define MAXSTOG 5, npr.
6  //LIFO
7  //puni rastući
8  //vrh pokazuje na zadnju puni element
9
10
11
12  #define MAXSTOG 5
13
14  typedef struct{
15      int vrh;
16      int polje[MAXSTOG];
17  }Stog;
18
19  void init_stog(Stog *stog){
20      stog->vrh=-1;
21  }
22
23  int dodaj_na_stog(Stog *stog, int element){
24      if(stog->vrh>=MAXSTOG-1){ //ako je stog pun
25          return 0;
26      }
27      else{
28          stog->vrh++;
29          stog->polje[stog->vrh]=element;
30          return 1;
31      }
32  }
33
34  int skini_sa_stoga(Stog *stog, int *element){
35      if(stog->vrh<0){ //ako je sto prazan
36          return 0;
37      }
38      else{
39          *element=stog->polje[stog->vrh];
40          stog->vrh--;
41          return 1;
42      }
43  }
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
```

```

66 2)Stog realiziran linearnom listom
67
68
69 /*Stog realiziran linearnom listom*/
70
71 struct at{
72     int element;
73     struct at* sljed;
74 };
75 typedef struct at atom;
76
77 typedef struct{
78     atom *vrh;
79 }Stog;
80
81 void init_stog(Stog *stog){
82     stog->vrh=NULL;
83 }
84
85
86 int dodaj_na_stog(Stog *stog, int element){
87     atom *novi;
88     if((novi=(atom*)malloc(sizeof(atom)))!=NULL){
89         novi->element=element;
90         novi->sljed=stog->vrh;
91         stog->vrh=novi;
92         return 1;
93     }
94     else{
95         return 0;
96     }
97 }
98
99 int skini_sa_stoga(Stog *stog, int *element){
100     atom *p;
101     if(stog->vrh!=NULL){ //ako je stog neprazan
102         *element=stog->vrh->element;
103         p=stog->vrh->sljed;
104         free((void*)stog->vrh);
105         stog->vrh=p;
106         return 1;
107     }
108     else{
109         return 0;
110     }
111 }
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130

```

```

131 3)Linearna jednostruko povezana lista
132
133 /*Linearna jednostruko povezana lista*/
134
135 struct at{
136     int element;
137     struct at* sljed;
138 };
139 typedef struct at atom;
140
141 // Dodavanje u listu
142 // sortiranu po rastucoj vrijednosti elementa
143 // vraca 1 ako uspije, inace 0
144 int dodaj_u_listu(atom **glavap,int element){
145     atom* novi,*p;
146     if((novi=(atom*)malloc(sizeof(atom)))==NULL){
147         return 0;
148     }
149     novi->element=element;
150     if(*glavap==NULL || (*glavap)->element>=element){
151         // Dodavanje na pocetak liste
152         novi->sljed=*glavap;
153         *glavap=novi;
154     }
155     else{
156         // Dodavanje iza postojeceg elementa kad:
157         // a) postojeći atom nema sljedećeg
158         // b) element u sljedećem cvoru je veći ili jednak novome
159         for(p=*glavap;p->sljed && (p->sljed)->element<element;p=p->sljed);
160         novi->sljed=p->sljed;
161         p->sljed=novi;
162     }
163     return 1;
164 }
165
166 int brisi_iz_liste(atom **glavap,int element){
167     atom* p;
168     for(;*glavap && (*glavap)->element!=element;glavap=&((*glavap)->sljed));
169     if(*glavap!=NULL){
170         // *glavap pokazuje na element koji se treba obrisati
171         // (*glavap)->sljed == adesa na sljed. elem, spremljenau atmu koji se brise
172         p=*glavap;
173         *glavap=(*glavap)->sljed;
174         free((void*)p);
175         return 1;
176     }
177     else{
178         return 0;
179     }
180 }
181
182 atom *trazi_elem(atom *glava,int element){
183     atom *p;
184     for(p=glava;p!=NULL && p->element!=element;p=p->sljed);
185     return p;
186 }
187
188 void ispisi_listu(atom *glava){
189     atom *p;
190     for(p=glava;p!=NULL;p=p->sljed){
191         printf("%d ",p->element);
192     }
193 }
194

```

4)Linearna dvostruko povezana lista

```
195 /*Linearna dvostruko povezana lista*/
196 //lista predstavlja red
197 //skidanje elementa se vrši na glavi
198 //dodavanje elementa se vrši na repu
199 //moguće brisanje elementa s zadanim ključem
200
201 struct at{
202     int element;
203     struct at *sljed;
204     struct at *preth;
205 };
206 typedef struct at atom;
207 //atom *glava,*rep;
208
209 //dodavanje se vrši na repu
210 int dodaj_u_listu(atom **glavap,atom **repp,int element){
211     atom *novi;
212     if((novi=(atom*)malloc(sizeof(atom)))!=NULL){
213         novi->element=element;
214         novi->sljed=NULL;
215         novi->preth=NULL;
216         if(*glavap==NULL){ //ako je lista prazna
217             *glavap=novi;
218             *repp=novi;
219         }
220         else{ //lista nije prazna
221             novi->preth=*repp;
222             *repp=novi;
223         }
224         return 1;
225     }
226     else{
227         return 0;
228     }
229 }
230
231 //skidanje s glave reda
232 int skini_iz_reda(atom **glavap,atom **repp,int *element){
233     atom *stari;
234     //ako lista nije prazna
235     if(*glavap){
236         *element=(*glavap)->element;
237         if(*glavap==*repp){ //lista sadrži samo jedan element
238             stari=*glavap;
239             *glavap=NULL;
240             *repp=NULL;
241         }
242         else{ //lista sadrži više elemenata
243             (*glavap)->sljed->preth=NULL;
244             stari=*glavap;
245             *glavap=(*glavap)->sljed;
246         }
247         free((void*)stari);
248         return 1;
249     }
250     else{
251         return 0;
252     }
253 }
254
255 }
```

```

261 //brisanje elementa
262 int brisi_iz_reda(atom **glavap,atom **repp,int element){
263     atom *p;
264     if(*glavap){//ako red nije prazan
265         for(p=*glavap;p && p->element!=element;p=p->sljed);
266         if(p){ //ako element postoji
267             if(p==*glavap){ //ako je elem na prvom mjestu
268                 *glavap=p->sljed;
269                 if(p->sljed){ //ako je prvi ali ne i jedini
270                     p->sljed->preth=NULL;
271                 }
272                 else{ //ako je jedini
273                     *glavap=NULL;
274                     *repp=NULL;
275                 }
276             }
277             else if(p==*repp){ //ako je zadnji, ali ne i jedini
278                 (*repp)->preth->sljed=NULL;
279                 *repp=(*repp)->preth;
280             }
281             else{ //nije ni zadnji ni prvi
282                 p->preth->sljed=p->sljed;
283                 p->sljed->preth=p->preth;
284             }
285             free((void*)p);
286             return 1;
287         }
288     }
289     return 0;
290 }
291
292
293 void ispis_reda(atom* glava){
294     atom *p;
295     for(p=glava;p;p=p->sljed){
296         printf("%d ",p->element);
297     }
298 }
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326

```

```

327 5)Red realiziran statičkim poljem
328
329 /*Red realiziran statičkim poljem*/
330 // #define MAXRED 10,npr.
331 //red je pun ako je ((ulaz+1)%MAXRED==izlaz)
332 //red je prazan ako je ulaz==izlaz
333 /*algoritam za dodavanje i skidanje elemenata:
334     1)povećaj
335     2)skaliraj
336     3)obavi operaciju
337 */
338
339 #define MAXRED 10
340 #define N 10
341
342 typedef struct{
343     int polje[MAXRED];
344     int ulaz,izlaz;
345 }Red;
346
347 int dodaj_u_red(Red *red,int element){
348     if((red->ulaz+1)%N == red->izlaz){ //ako je red pun
349         return 0;
350     }
351     else{
352         red->ulaz++; //povecaj
353         red->ulaz%=N; //skaliraj
354         red->polje[red->ulaz]=element; //obavi operaciju
355         return 1;
356     }
357 }
358
359 int skini_iz_reda(Red *red,int *element){
360     if(red->ulaz==red->izlaz){ //ako je red prazan
361         return 0;
362     }
363     else{
364         red->izlaz++; //povecaj
365         red->izlaz%=N; //skaliraj
366         *element=red->polje[red->izlaz]; //obavi operaciju
367     }
368 }
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386

```

```

387 6)Red realiziran linearnom listom
388
389 /*red realiziran listom*/
390 //dodajemo na ulaz
391 //skidamo na izlazu
392
393 struct at{
394     int element;
395     struct at *sljed;
396 };
397 typedef struct at atom;
398
399 typedef struct{
400     atom *ulaz;
401     atom *izlaz;
402 }Red;
403
404 //inicijalizacija reda
405 void init_red(Red *red){
406     red->ulaz=NULL;
407     red->izlaz=NULL;
408 }
409
410 int dodaj_u_red(Red *red, int element){
411     atom *novi;
412     if((novi=(atom*)malloc(sizeof(atom)))!=NULL){
413         novi->element=element;
414         novi->sljed=NULL;
415         if(red->izlaz==NULL){ //ako je red prazan
416             red->izlaz=novi;
417         }
418         else{ //nije prazan, dodaj na kraj
419             (red->izlaz)->sljed=novi;
420         }
421         red->ulaz=novi;
422         return 1;
423     }
424     else{
425         return 0;
426     }
427 }
428
429 //skini element s izlaza i obrisi atom
430 int skini_iz_reda(Red *red,int *element){
431     atom *p;
432     if(red->izlaz!=NULL){ //ako red nije prazan
433         *element=red->izlaz->element;
434         p=red->izlaz;
435         red->izlaz=red->izlaz->sljed;
436         free((void*)p);
437         if(red->izlaz==NULL){ //ako je red sada ostao prazan
438             red->ulaz=NULL;
439         }
440         return 1;
441     }
442     else{
443         return 0;
444     }
445 }
446
447
448
449
450
451
452

```

```

453 7)Binarno stablo
454
455 /*Binarno stablo*/
456 //cvor *korijen=NULL;
457 //lijevo idu manji elementi, desno veci
458
459 struct cv{
460     int element;
461     struct cv *lijevo;
462     struct cv *desno;
463 };
464 typedef struct cv cvor;
465
466 cvor *noviElement(int element){
467     cvor *novi;
468     if((novi=(cvor*)malloc(sizeof(cvor)))!=NULL){
469         novi->element=element;
470         novi->desno=NULL;
471         novi->lijevo=NULL;
472         return novi;
473     }
474     else{
475         return NULL;
476     }
477 }
478 //korijen=dodaj(...)
479 cvor *dodaj_u_stablo(cvor *korijen,int element){
480     if(korijen==NULL){
481         return noviElement(element);
482     }
483     else{
484         if(element<=korijen->element){
485             korijen->lijevo=dodaj_u_stablo(korijen->lijevo,element);
486         }
487         else{
488             korijen->desno=dodaj_u_stablo(korijen->desno,element);
489         }
490         return korijen;
491     }
492 }
493
494 int trazi(cvor *korijen,int element){
495     if(korijen==NULL){
496         return 0;
497     }
498     else if(element==korijen->element){
499         return 1;
500     }
501     else{
502         if(element<korijen->element){
503             return trazi(korijen->lijevo,element);
504         }
505         else{
506             return trazi(korijen->desno,element);
507         }
508     }
509 }
510
511
512
513
514
515
516
517
518

```



```

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

```