Red (Queue)

Red je posebna vrsta liste: od svih operacija dozvoljeno je samo ubacivanje elementa na jedan kraj liste (začelje, kraj reda), te gledanje sadržaja elementa i uklanjanje elementa s drugog kraja liste (početak reda, čelo). Red zovemo i FIFO – first in first out. čelo $\rightarrow a_0, a_1, ..., a_n \leftarrow$ začelje

a.t.p. Queue

`elementtype`	. . .	bilo koji tip
`Queue`	. . .	konačan niz podataka tipa `elementtype`
`QuMakeNull(&Q)`	. . .	pretvara `Q` u prazan red
`QuEmpty(Q)`	. . .	vraća `1` ako je red `Q` prazan, `0` ako nije
`QuEnqueue(x,&Q)`	. . .	ubacuje element `x` na začelje reda `Q`
`QuDequeue(&Q)`	. . .	izbacuje element s čela reda `Q`; nije definirano ako je red prazan
`QuFront(Q)`	. . .	vraća vrijednost elementa na čelu reda `Q`; nije definirano ako je red prazan

Primjer. `Q:` $\rightarrow$ `Enqueue(2,&Q)` $\rightarrow$ `Q: 2` $\rightarrow$ `Enqueue(5,&Q)` $\rightarrow$ `Q: 2 5` $\rightarrow$ `Enqueue(7,&Q)` $\rightarrow$ `Q: 2 5 7` $\rightarrow$ `Dequeue(&Q)` $\rightarrow$ `Q: 5 7`

Primjena. U takozvanim Breadth-First-Search algoritmima (BFS). Kod tih se algoritama zadatak rješava razlaganjem na manje podzadatke, oni na još manje podzadatke i tako dalje... Formira se red podzadataka koje još treba riješiti. Program skida podzadatak s čela i rješava ga; ako se pritom jave novi podzadaci, njih stavlja na začelje reda. Problem je riješen kad se red isprazni. Algoritam:

```
Queue Q;
QuMakeNull(&Q);
QuEnqueue(Z, &Q);
while (!QuEmpty(Q)) {
    elementtype P = QuFront(Q); QuDequeue(&Q);
    for each subproblem PP of P
        QuEnqueue(PP, &Q);
}
```

Zadatak 9.

Dan je orijentirani graf čiji čvorovi su numerirani s $0, 1, 2, ..., n-1$. Naprimjer:

Napišite funkciju koja ispisuje skup dostupnih vrhova S_i za dani čvor i, tj. sve čvorove u koje se može doći iz čvora i nekim putom u G.

Pretpostavite da je graf zadan poljem G redova koji predstavljaju popise susjeda pojedinih čvorova. Naprimjer, za graf na slici je: `G[0]` $=$ `(1,2)`, `G[1]` $=$ `(5)`, `G[2]` $=$ `(1,3)`, itd.

Rješenje. Ti čvorovi su: i, svi susjedi od i, svi susjedi od susjeda od i, i tako dalje... tj. vrijedi:

$$\text{skup dostupnih čvorova od } i \;=\; \{\, i \,\} \;\cup\; \text{unija skupova dostupnih od svih susjeda od } i.$$

Algoritam:

```
stavimo cvor i u red Q;
sve dok se red Q ne isprazni {
    uzmemo cvor s pocetka reda Q i ispisemo ga (ako vec nismo);
    u red Q stavimo sve susjede od tog cvora (koji nisu vec bili u redu);
}
```

Stanja reda za gornji primjer: `Q: 0` $\overset{0}{\rightarrow}$ `Q: 1 2` $\overset{1}{\rightarrow}$ `Q: 2 5` $\overset{2}{\rightarrow}$ `Q: 5 3` $\overset{5}{\rightarrow}$ `Q: 3` $\overset{3}{\rightarrow}$ `Q:`

```
#define N 1000

void accessible(Queue G[], int pocetni) {
    int bio[N], i;
    Queue Q;
    for (i=0; i<N; i++)
        bio[i]=0;
    QuMakeNull(&Q);
    QuEnqueue(pocetni, &Q); bio[pocetni]=1;
    while (!QuEmpty(Q)) {
        int vrh = QuFirst(Q);
        printf("%d", vrh);
        while (!QuEmpty(G[vrh])) {
            int susjed = QuFirst(G[vrh]);
            QuDequeue(&G[vrh]);
            if (!bio[susjed]) {
                QuEnqueue(susjed, &Q);
                bio[susjed]=1;
            }
       }
       QuDequeue(&Q);
    }
}
```

Napomena. Može li se ovaj zadatak riješiti pomoću stoga? Kojim se redom u tom slučaju obilaze vrhovi?

Zadatak za DZ

Napišite funkciju `int distance(Queue G[], int start, int cilj)` koja vraća udaljenost između vrhova `start` i `cilj`. Uputa: modificirajte malo rješenje prethodnog zadatka. Ovo se zove BFS.

Na predavanju smo promatrali implementaciju reda pomoću cirkularnog polja:

Zadatak za DZ

Razradite implementaciju reda pomoću cirkularnog polja, kursora na čelo i začelje te ćelije koja označava je li red prazan.

Napomena. Ovo je nepotrebna komplikacija koja uštedi svega `1 sizeof(elementtype)`. Implementacije samo s poljem (bez dinamičkog alociranja) su općenito loše zbog mogućnosti prepunjenja. Bolje su implementacije s dinamičkom realokacijom.

Zadatak 10.

Napišite funkciju `void QSort(Queue* Q)` koja prima red Q čiji su elementi cijeli brojevi i sortira elemente tog reda silazno. Smijete upotrijebiti samo još jedan pomoćni stog; ne smijete upotrijebiti druge a.t.p.-ove niti polja. Rješenje treba biti neovisno o implementaciji a.t.p. Stack i Queue. (Zadaci za vježbu za prvi kolokvij, Zadatak 11.)