Uvod

Osnovni pojmovi

Dijagram koji prikazuje razvoj programa metodom postepenog profinjavanja:

		matematički model		apstraktni tipovi podataka		strukture podataka		
problem	→		→		→		→	rješenje problema
		neformalno opisan algoritam		algoritam zapisan u pseudo-jeziku (koriste se operacije iz a.t.p.)		algoritam zapisan u programskom jeziku (operacije iz a.t.p. su potprogrami)		

Osnovni pojmovi: ćelija, polje, zapis, pointer, kursor.

Promatrat ćemo sada jedan konkretan problem i razvit ćemo program koji ga rješava. Identificirat ćemo sve faze s dijagrama.

Problem

Raskršće izgleda kao na slici. Putovi C i E su jednosmjerni, ostali su dvosmjerni. Postoji 13 načina prolazaka kroz raskršće. Neki parovi prolazaka (npr. AB i EC) mogu se obavljati istovremeno, a neki (npr. AD i EB) ne mogu jer im se putanje sijeku. Na raskršće će se postaviti semafori. Potrebno je smisliti režim rada semafora. Režim rada sastoji se od nekoliko faza. Zahtijeva se:

1. Za vrijeme jedne faze nekim prolascima je slobodno, a nekima zabranjeno. Pritom prolasci kojima se putanje sijeku ne smiju imati oba slobodno.
2. Za svaki od 13 prolazaka je u barem jednoj fazi slobodno.
3. Broj faza je što manji.

Matematički model

Problem modeliramo neorijentiranim grafom. Vrhovi predstavljaju prolaske. Bridovi povezuju prolaske koji se ne mogu odvijati istodobno.

Polazni problem je ekvivalentan s problemom bojanja grafa: pobojati vrhove grafa sa što manje boja tako da susjedni vrhovi budu različitih boja. Svaka boja je jedna faza rada semafora. (Postoji mala razlika: u ovako formuliranom problemu svaki prolazak može biti u najviše jednoj fazi rada semafora, čega u početnom problemu nema. Ipak, kažemo da su problemi ekvivalentni jer ako imamo optimalno rješenje jednog imamo i drugog i obratno.)

Neformalno opisan algoritam

Problem bojanja grafa poznat je u teoriji algoritama kao tvrdokoran (NP težak). Mi ćemo zbog toga koristiti jedan heuristički (približni) algoritam koji daje prilično dobro rješenje. Rješenje nije nužno optimalno. Algoritam spada među takozvane pohlepne (greedy) algoritme.

```
void Greedy(ColoredGraph G) {
    colortype newclr;
    do {
        newclr = bilo koja neupotrebljena boja;
        for (svaki neobojeni vrh v grafa G)
            if (v nije susjedan niti jednom vrhu obojenom s newclr)
                oboji vrh v s newclr;
    } while (jos nisu svi vrhovi od G obojeni);
}
```

Uzmemo jednu neupotrebljenu boju i redom bojimo sve vrhove koje njome smijemo obojati. Zatim uzmemo sljedeću boju i tako dalje dok ne obojimo sve vrhove.

Apstraktni tip podataka

Uvodimo a.t.p. `ColoredGraph` koji odgovara matematičkom pojmu obojenog (neusmjerenog) grafa, s operacijama koje se pojavljuju u prethodno opisanom algoritmu. A.t.p. je zadan sljedećim popisom tipova i operacija:

`vertex`	. . .	podaci ovog tipa identificiraju vrhove
`colortype`	. . .	podaci ovog tipa označavaju boje; uključena je i posebna boja 'blank' koja označava odsustvo boje
`ColoredGraph`	. . .	podatak ovog tipa je (neorijentirani) graf sastavljen od različitih vrhova tipa `vertex`; svaki vrh je obojen bojom tipa `colortype`
`Color(v,G)`	. . .	funkcija koja vraća boju vrha `v` u grafu `G`
`Adjacent(v1,v2,G)`	. . .	logička funkcija, vraća `1` ako su vrhovi `v1` i `v2` susjedni, a `0` inače
`SetColor(c,v,&G)`	. . .	procedura pridružuje boju `c` vrhu `v` grafa `G`

Algoritam zapisan u pseudo-jeziku

```
void Greedy(ColoredGraph* G) {
    int finished, found;
    colortype newclr;
    do {
        finished = 1;
        newclr = bilo koja jos neiskoristena boja;
        for(svaki vrh i od *G)
            if (Color(i, *G) == blank) {
                finished = 0;
                found = 0;
```

```
                for(svaki vrh j od *G)
                    if (Adjacent(i, j, *G) && Color(j, *G) == newclr)
                        found = 1;
                if (!found)
                    SetColor(newclr, i, G);
            }
    } while (!finished);
}
```

Struktura podataka

Ograničavamo se na grafove s fiksnim unaprijed zadanim brojem vrhova `N`. Tipove `colortype` i `vertex` kodiramo na sljedeći način:

```
#define N ...
typedef int vertex;
typedef int colortype;
```

Doduše, vrhovi u našem grafu bili su imenovani AB, AC, AD, ..., no možemo ih lako preimenovati u `0`, `1`, ..., `N-1`. Boja 'blank' označena je s `0`. Sam obojani graf (dakle podatak tipa `ColoredGraph`) prikazujemo jednom matricom i jednim poljem:

```
typedef struct {
    int adj[N][N];
    colortype clr[N];
} ColoredGraph;
```

(i, j)-ti element matrice `adj` označava da li je i-ti vrh susjed j-tog vrha. Budući da je graf neusmjeren, matrica je nužno simetrična. i-ti element polja `clr` sadrži boju i-tog vrha. Za naš konkretni graf vrijednosti su sljedeće:

		AB	AC	AD	BA	BC	BD	DA	DB	DC	EA	EB	EC	ED		
	adj	0	1	2	3	4	5	6	7	8	9	10	11	12		clr
AB	0					1	1	1			1				0	0
AC	1						1	1	1		1	1			1	0
AD	2										1	1	1		2	0
BA	3														3	0
BC	4	1							1			1			4	0
BD	5	1	1					1				1	1		5	0
DA	6	1	1				1					1	1		6	0
DB	7		1			1							1		7	0
DC	8														8	0
EA	9	1	1	1											9	0
EB	10		1	1		1	1	1							10	0
EC	11			1			1	1	1						11	0
ED	12														12	0

Da bismo dobili potpunu implementaciju a.t.p.-a, trebamo operacije iz a.t.p. napisati kao potprograme. No, za naše operacije to je trivijalno: svaka se svodi na jedan logički uvjet ili jedno pridruživanje.

```
colortype Color(vertex v, ColoredGraph G) {
    return G.clr[v];
}
```

```
int Adjacent(vertex v1, vertex v2, ColoredGraph G) {
    return G.adj[v1][v2];
}

void SetColor(colortype c, vertex v, ColoredGraph* G) {
    (*G).clr[v] = c;
}
```

Algoritam zapisan u programskom jeziku (potprogram)

```
void Greedy(ColoredGraph* G) {
    int finished, found;
    colortype newclr;
    vertex i, j;
    newclr = 0;
    do {
        finished = 1;
        newclr++;
        for(i = 0; i < N; i++)
            if (Color(i, *G) == 0) {
                finished = 0;
                found = 0;
                for(j = 0; j < N; j++)
                    if (Adjacent(i, j, *G) && Color(j, *G) == newclr)
                        found = 1;
                if (!found)
                    SetColor(newclr, i, G);
            }
    } while (!finished);
}
```

Ovaj potprogram bi još trebalo uklopiti u glavni program. Na početku glavnog programa definirala bi se konstanta `N` jednaka `13` te svi potrebni tipovi. U tijelu glavnog programa inicijalizirali bi se podaci koji opisuju graf, zatim bi se pozvala procedura `Greedy`, a na kraju bi se interpretirali rezultati.

Rješenje koje program daje za naš konkretan problem raskršća glasi:

FAZA	PROLASCI
1	AB, AC, AD, BA, DC, ED
2	BC, BD, EA
3	DA, DB
4	EB, EC

Lako je pokazati da je ovo optimalno rješenje. Naime, u grafu postoji tzv. "klika" reda 4: skup od četiri vrha koji su svi međusobno povezani (AC, DA, BD, EB). Očito nam trebaju barem četiri boje da obojimo ovu kliku. Postoje i druga optimalna rješenja, dakle i drugi načini da se zadani graf oboji s četiri boje.