Strutture Dati

Lezione 7
Il problema del labirinto

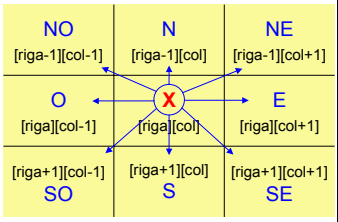
Oggi parleremo di …

- Un'applicazione dello ***stack***
  - il problema del labirinto
  - rappresentazione di un labirinto
  - come muoverci all'interno del labirinto
  - come trovare un percorso dall'entrata all'uscita.

0	1	0	0	1	1	1	1
1	0	1	1	1	0	1	0
1	0	1	1	0	1	1	0
1	0	1	0	1	1	0	1
0	1	0	1	1	0	1	1
0	1	1	1	0	1	0	1
1	0	0	0	1	0	0	1
1	1	1	1	1	1	**0**	1

Rappresentazione di un labirinto

- La scelta più ovvia è un array bidimensionale `lab[riga][col]`, in cui
  - 0 indica un percorso aperto
  - 1 indica una barriera
- Per ogni `lab[riga][col]`, quali sono le possibili mosse?
- Si circonda il labirinto con una barriera di 1
- Un labirinto *m* x *p* richiede un array (*m*+2) x (*p*+2)



Rappresentazione delle mosse

Nome	dir	mossa[dir].vert	mossa[dir].orizz
N	0	-1	0
NE	1	-1	1
E	2	0	1
SE	3	1	1
S	4	1	0
SO	5	1	-1
O	6	0	-1
NO	7	-1	-1

```
typedef struct {
        int vert;
        int orizz;
    } offset;

offset mossa[8]={
   {-1,0}, {-1,1}, {0,1},
   {1,1},  {1,0},  {1,-1},
   {0,-1}, {-1,-1}};
```

riga_succ = riga + mossa[dir].vert;

col_succ = col + mossa[dir].orizz;

Ricerca di un percorso dall'entrata all'uscita

- E' possibile scegliere una qualunque direzione.
- Si salva la posizione corrente e si sceglie, arbitrariamente, una mossa possibile
  - consente di ritornarvi e provare un altro percorso (se si giunge ad una strada senza uscita).
- Si esaminano le mosse partendo da nord spostandosi in senso orario.
- Si registrano le posizioni attraversate in un secondo array `segna`.

Algoritmo di ricerca

```
Inizializzare uno stack con le coordinate di entrata del labirinto
   e direzione nord;

while (lo stack non è vuoto) {
   <riga,col,dir> = cancella dallo stack;
   while (esistono altre mosse dalla posizione corrente) {
         <riga_succ, col_succ> = coordinate della mossa successiva;
         dir = direzione della mossa;
         if ((riga_succ == RIGA_USCITA) && (col_succ == COL_USCITA))
                successo;
         if ((lab[riga_succ][col_succ] == 0) &&
                (segna[riga_succ][col_succ] == 0)) {

                /*posizione ammessa e mai visitata */
                segna[riga_succ][col_succ] = 1;

                /* salva la direzione e la posizione corrente */
                add <riga,col,dir> allo stack;
                riga = riga_succ;
                col = col_succ;
                dir = nord;
                }
         }
}
printf("Non esiste un percorso");
```

Rappresentazione dello stack

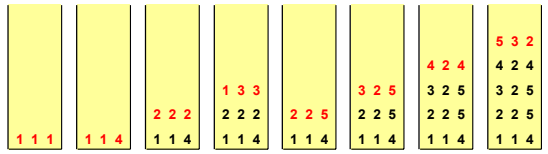
```
#define MAX_STACK_SIZE 300

typedef struct {
        int riga;
        int col;
        int dir;
     } elemento;

elemento stack[MAX_STACK_SIZE];
int top = -1;
```

Lo stack deve avere una capacità di m x p.

Dopo aver attraversato un labirinto, lo stack conterrà tutte le posizioni del percorso trovato (tranne l'uscita).

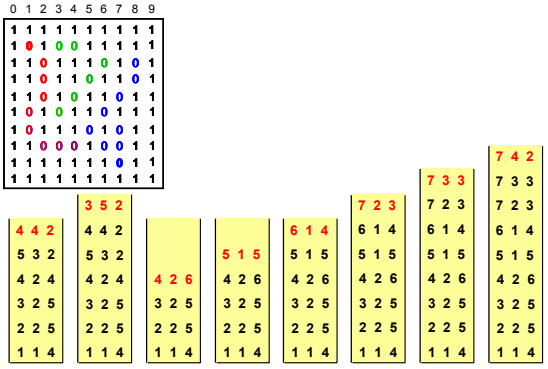
Esempio di labirinto

	0	1	2	3	4	5	6	7	8	9
0	1	1	1	1	1	1	1	1	1	1
1	1	0	1	0	0	1	1	1	1	1
2	1	1	0	1	1	1	0	1	0	1
3	1	1	0	1	1	0	1	1	0	1
4	1	1	0	1	0	1	1	0	1	1
5	1	0	1	0	1	1	0	1	1	1
6	1	0	1	1	1	0	1	0	1	1
7	1	1	0	0	0	1	0	0	1	1
8	1	1	1	1	1	1	1	0	1	1
9	1	1	1	1	1	1	1	1	1	1



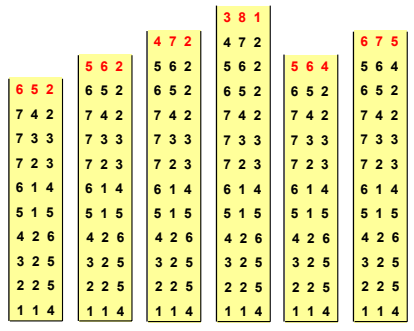
Esempio di labirinto

	0	1	2	3	4	5	6	7	8	9
0	1	1	1	1	1	1	1	1	1	1
1	1	0	1	0	0	1	1	1	1	1
2	1	1	0	1	1	1	0	1	0	1
3	1	1	0	1	1	0	1	1	0	1
4	1	1	0	1	0	1	1	0	1	1
5	1	0	1	0	1	1	0	1	1	1
6	1	0	1	1	1	0	1	0	1	1
7	1	1	0	0	0	1	0	0	1	1
8	1	1	1	1	1	1	1	0	1	1
9	1	1	1	1	1	1	1	1	1	1



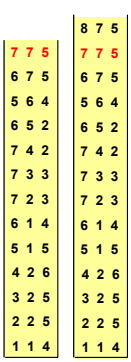
Esempio di labirinto

	0	1	2	3	4	5	6	7	8	9
0	1	1	1	1	1	1	1	1	1	1
1	1	0	1	0	0	1	1	1	1	1
2	1	1	0	1	1	1	0	1	0	1
3	1	1	0	1	1	0	1	1	0	1
4	1	1	0	1	0	1	1	0	1	1
5	1	0	1	0	1	1	0	1	1	1
6	1	0	1	1	1	0	1	0	1	1
7	1	1	0	0	0	1	0	0	1	1
8	1	1	1	1	1	1	1	0	1	1
9	1	1	1	1	1	1	1	1	1	1



Esempio di labirinto

	0	1	2	3	4	5	6	7	8	9
0	1	1	1	1	1	1	1	1	1	1
1	1	0	1	0	0	1	1	1	1	1
2	1	1	0	1	1	1	0	1	0	1
3	1	1	0	1	1	0	1	1	0	1
4	1	1	0	1	0	1	1	0	1	1
5	1	0	1	0	1	1	0	1	1	1
6	1	0	1	1	1	0	1	0	1	1
7	1	1	0	0	0	1	0	0	1	1
8	1	1	1	1	1	1	1	0	1	1
9	1	1	1	1	1	1	1	1	1	1



Il percorso nella forma (riga, col, direzione) è

1 1 3
2 2 4
3 2 4
4 2 5
5 1 4
6 1 3
7 2 2
7 3 2
7 4 1
6 5 1
5 6 3
6 7 4
7 7 4
8 7

Funzione di ricerca

$O(m \times p)$

```
void path(void)
{
 int i, riga, col, riga_succ, col_succ, dir, trovato = FALSE;
 elemento posizione;
 int segna[MAX_RIGA][MAX_COL];

 for (riga = 0; riga < MAX_RIGA; riga++)
     for (col = 0; col < MAX_COL; col++)  segna[riga][col] = 0;
 segna[RIGA_ENTRATA][COL_ENTRATA] = 1;  top = 0;
 stack[0].riga = RIGA_ENTRATA;          stack[0].col = COL_ENTRATA;    stack[0].dir = 0;
 while (top>-1 && !trovato) {
   posizione = del(&top);
   riga = posizione.riga;     col = posizione.col;     dir = posizione.dir;
   while (dir<8 && !trovato) {
          /*mossa nella direzione dir */
          riga_succ = riga + mossa[dir].vert;
          col_succ  = col + mossa[dir].orizz;
          if(riga_succ==RIGA_USCITA && col_succ==COL_USCITA)      trovato = TRUE;
          else if(!lab[riga_succ][col_succ] && !segna[riga_succ][col_succ]) {
                    segna[riga_succ][col_succ] = 1;
                    posizione.riga = riga;   posizione.col = col;   posizione.dir = ++dir;
                    add(&top, posizione);
                    riga = riga_succ;     col = col_succ;     dir = 0;
                  }
          else ++dir;
    }
   }
 if(trovato) {
         printf("Il percorso e\':\n");    printf("riga  col\n");
          for (i=0; i <= top; i++)      printf("%2d%6d\n", stack[i].riga, stack[i].col);
          printf("%2d%6d\n", riga, col);    printf("%2d%6d\n", RIGA_USCITA, COL_USCITA);}
 else  printf("Il labirinto non ha un percorso di uscita\n");
}
```