

1. Intro. We output an Eulerian trail of the (undirected) graph named on the command line. (Each edge is considered to be two directed arcs; thus it is traversed in both directions.)

If the graph isn't connected, we consider only the vertices that are reachable from the first one, *g-vertices*.

```
#include <stdio.h>
#include <stdlib.h>
#include "gb_graph.h"
#include "gb_save.h"
⟨Subroutine 3⟩
main(int argc, char *argv[])
{
    register int k;
    Graph *g;
    Vertex *u, *v;
    Arc *a;
    ⟨Input the graph 2⟩;
    ⟨Traverse depth first 4⟩;
    ⟨Output the trail 5⟩;
    printf("\n");
}
```

2. ⟨Input the graph 2⟩ \equiv

```
if (argc  $\neq$  2  $\vee$   $\neg$ ( $g = \text{restore\_graph}(argv[1])$ )) {
    fprintf(stderr, "Usage: %s foo.gb\n", argv[0]);
    exit(-1);
}
fprintf(stderr, "OK, I've input '%s'.\n", argv[1]);
gb_new_edge(g-vertices, g-vertices + g-n, 0); /* dummy edge */
```

This code is used in section 1.

3. Subroutine *dfs*(*u*, *v*) sets *v-parent* = *u* and *v-nav* to the vertex that follows *u* in *v*'s adjacency list. It also explores all vertices reachable from *v* that haven't already been seen.

```
#define parent v.V
#define nav w.A
⟨Subroutine 3⟩  $\equiv$ 
void dfs(register Vertex *u, register Vertex *v)
{
    register Vertex *w;
    register Arc *a;
    v-parent = u;
    for (a = v-arcs; a; a = a-next) {
        w = a-tip;
        if (w  $\equiv$  u) v-nav = a-next;
        else if (w-parent  $\equiv$   $\Lambda$ ) dfs(v, w);
    }
}
```

This code is used in section 1.

4. ⟨Traverse depth first 4⟩ \equiv

```
dfs(g-vertices + g-n, g-vertices);
```

This code is used in section 1.

5. Now the Eulerian traversal is beautifully simple.

⟨ Output the trail 5 ⟩ \equiv

```

for ( $v = g\text{-vertices}; v \neq g\text{-vertices} + g\text{-}n; \text{ } \rangle \{$ 
```

```

     $\text{printf}(\text{"\s"}, v\text{-name});$ 
     $a = v\text{-nav};$ 
    if ( $\neg a$ )  $a = v\text{-arcs};$ 
     $v\text{-nav} = a\text{-next};$ 
     $v = a\text{-tip};$ 

```

This code is used in section 1.

6. Index.*a*: [1](#), [3](#).**Arc**: [1](#), [3](#).*arcs*: [3](#), [5](#).*argc*: [1](#), [2](#).*argv*: [1](#), [2](#).*dfs*: [3](#), [4](#).*exit*: [2](#).*fprintf*: [2](#).*g*: [1](#).*gb_new_edge*: [2](#).**Graph**: [1](#).*k*: [1](#).*main*: [1](#).*name*: [5](#).*nav*: [3](#), [5](#).*next*: [3](#), [5](#).*parent*: [3](#).*printf*: [1](#), [5](#).*restore_graph*: [2](#).*stderr*: [2](#).*tip*: [3](#), [5](#).*u*: [1](#), [3](#).*v*: [1](#), [3](#).**Vertex**: [1](#), [3](#).*vertices*: [1](#), [2](#), [4](#), [5](#).*w*: [3](#).

- ⟨Input the graph 2⟩ Used in section 1.
- ⟨Output the trail 5⟩ Used in section 1.
- ⟨Subroutine 3⟩ Used in section 1.
- ⟨Traverse depth first 4⟩ Used in section 1.

EULER-TRAIL

	Section	Page
Intro	1	1
Index	6	3