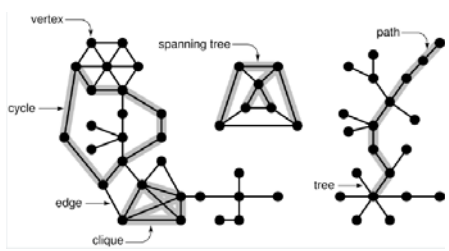
Undirected graphs

anhtt-fit@mail.hut.edu.vn

Undirected graphs

- A graph G=(V, E) where V is a set of vertices connected pairwise by edges E.
- Why study graph algorithms?
  - Interesting and broadly useful abstraction.
  - Challenging branch of computer science and discrete math.
  - Hundreds of graph algorithms known.
  - Thousands of practical applications.
    - Communication, circuits, transportation, scheduling, software systems, internet, games, social network, neural networks, …

Graph terminology

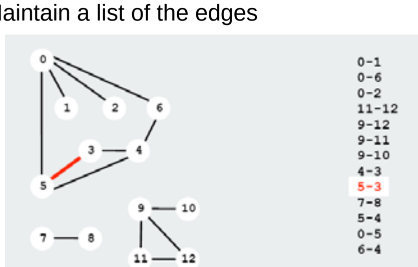


Some graph-processing problems

- Path: Is there a path between s to t?
- Shortest path: What is the shortest path between s and t?
- Cycle: Is there a cycle in the graph?
- Euler tour: Is there a cycle that uses each edge exactly once?
- Hamilton tour: Is there a cycle that uses each vertex exactly once?
- Connectivity: Is there a way to connect all of the vertices?
- MST: What is the best way to connect all of the vertices?
- Biconnectivity: Is there a vertex whose removal disconnects the graph?

Graph representation (1)

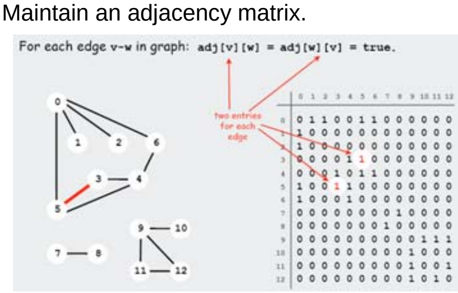
- Maintain a list of the edges



- Not suitable for searching

Graph representation (2)

- Maintain an adjacency matrix.



- Suitable for random accesses to the edges

A graph data structure

- Use a dynamic array to represent a graph as the following

```
typedef struct {
   int * matrix;
   int sizemax;
} Graph;
```

- Define the following API

```
Graph createGraph(int sizemax);
void addEdge(Graph graph, int v1, int v2);
int adjacent(Graph graph, int v1, int v2);
int getAdjacentVertices(Graph graph, int vertex, int* output); //
   return the number of adjacent vertices.
void dropGraph(Graph graph);
```

How to use the API?

```
int i, n, output[100];
Graph g = createGraph(100);
addEdge(g, 0, 1);
addEdge(g, 0, 2);
addEdge(g, 1, 2);
addEdge(g, 1, 3);
n = getAdjacentVertices (g, 1, output);
if (n==0) printf("No adjacent vertices of node 1\n");
else {
  printf("Adjacent vertices of node 1:");
  for (i=0; i<n; i++) printf("%5d", output[i]);
}
```

Quiz 1

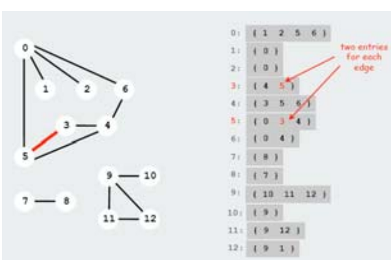
- Write the implementation for the API defined in the previous slide
- Use the example to test your API

Solution

- graph_matrix.c

Graph representation (3)

- Maintain an adjacency list.



Comparison

- Adjacency List is usually preferred, because it provides a compact way to represent sparse graphs – those for which $|E|$ is much less than $|V|^2$
- Adjacency Matrix may be preferred when the graph is dense, or when we need to be able to tell quickly if their is an edge connecting two given vertices

Implementation

- The red black tree can be used to store such a graph where each node in the tree is a vertex and its value is a list of adjacent vertices.
- Such a list of adjacent vertices can be stored in a red black tree as well.

Quiz 2

- Rewrite the API defined for graphs using the libfdr library as the following

```
#include "jrb.h"
typedef JRB Graph;

Graph createGraph();
void addEdge(Graph graph, int v1, int v2);
int adjacent(Graph graph, int v1, int v2);
int getAdjacentVertices (Graph graph, int v, int*
   output);
void dropGraph(Graph graph);
```

Instructions (1)

- To create a graph

  Simply call make_jrb()
- To add a new edge (v1, v2) to graph g

```
tree = make_jrb();
jrb_insert_int(g, v1, new_jval_v(tree));
jrb_insert_int(tree, v2, new_jval_i(1));
```
- If the node v1 is already allocated in the graph

```
node = jrb_find_int(g, v1);
tree = (JRB) jval_v(node->val);
jrb_insert_int(tree, v2, new_jval_i(1));
```

Instructions (2)

- To get adjacent vertices of v in graph g

```
node = jrb_find_int(g, v);
tree = (JRB) jval_v(node->val);
total = 0;
jrb_traverse(node, tree)
   output[total++] = jval_i(node->key);
```
- To delete/free a graph

```
jrb_traverse(node, graph)
   jrb_free_tree( jval_v(node->val) );
```

Solution

- graph_jrb.c

Homework

- In order to describe the metro lines of a city, we can store the data in a file as the following.

```
[STATIONS]
S1=Name of station 1
S2=Name of station 2
…
[LINES]
M1=S1 S2 S4 S3 S7
M2=S3 S5 S6 S8 S9
…
```

- Make a program to read such a file and establish the network of metro stations in the memory using the defined API.
- Write a function to find all the stations adjacent to a station given by its name.