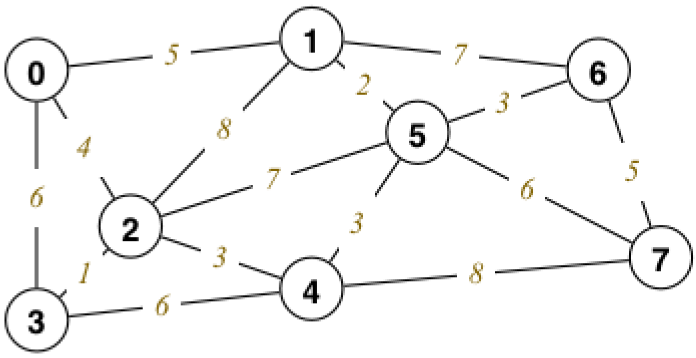
目录

Week 9 Exercises

Dijkstra.txt

Dijkstra's algorithm computes the minimum path costs from the source node 0 to all the other vertices, resulting in the Shortest Path Tree (SPT). For the following graph:



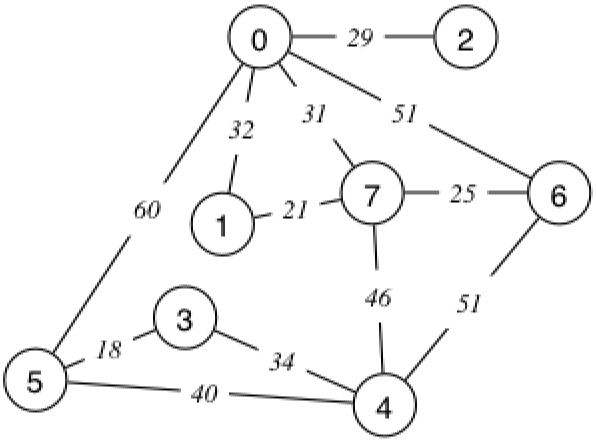
a. show the order that vertices are visited, and the path cost *pacost[]* of each of the vertices
   - (when faced with a choice, select the edge to the lowest vertex)

b. there are 4 vertices that undergo *non-trivial edge relaxation*. What are they, and what is the reduction in cost for each vertex?

c. draw the SPT

Prim.txt

Prim's Algorithm generates a Minimum Spanning Tree (MST). For the following graph:



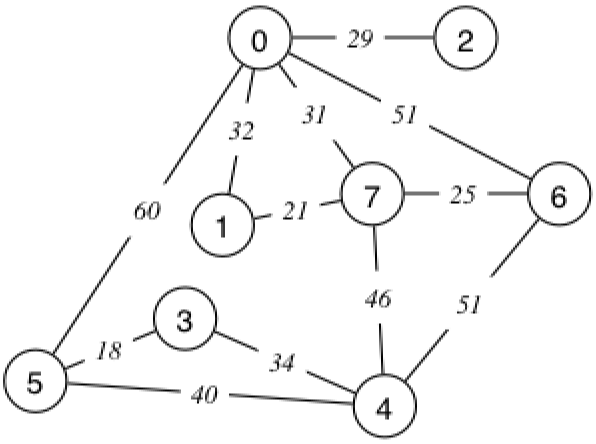
a. by hand, use the sets *mst* and *rest* to build an MST one vertex at a time Do this in the form of a table, where the first 2 lines have been completed for you.

mst	rest vertex	cost
{0}	2	29
{0,2}	7	31
...	...	...

b. draw the MST
c. how many edges in the MST?
d. what is the cost of the MST?

Kruskal.txt

Kruskal's Algorithm also generates an MST. For the following graph:



a. show which edges are considered at each step and ...
    - whether the edge gets accepted, or rejected (because it causes a cycle)
b. draw the MST.
    - is the MST the same of different to the MST generated by Prim's algorithm?

eulerianCycle.c

a. What determines whether a graph is Eulerian or not?
b. Write a C program that reads a graph, prints the graph, and determines whether an input graph is Eulerian or not.
    - if the graph is Eulerian, the program prints an Eulerian path

    - you should start with vertex 0
    - note that you may use the function *findEulerianCycle()* from the lecture on Graph Search Applications
  - if it is not Eulerian, the program prints the message *Not Eulerian*

For example,

- The graph:

  ```
  #4
  0 1 0 2 0 3 1 2 2 3
  ```

  is not Eulerian (*can you see why?*). Using this as input, your program should output:

  ```
  V=4, E=5
  <0 1> <0 2> <0 3>
  <1 0> <1 2>
  <2 0> <2 1> <2 3>
  <3 0> <3 2>
  Not Eulerian
  ```

- In the above-named lecture I showed a 'concentric squares' graph (called *concsquares*):

  ```
  #8
  0 7 7 5 5 1 1 0
  6 0 6 7
  2 5 2 7
  4 1 4 5
  3 0 3 1
  ```

  which is Eulerian, although I've labelled the vertices differently here. For this input your program should produce the output:

  ```
  V=8, E=12
  <0 1> <0 3> <0 6> <0 7>
  <1 0> <1 3> <1 4> <1 5>
  <2 5> <2 7>
  <3 0> <3 1>
  <4 1> <4 5>
  <5 1> <5 2> <5 4> <5 7>
  <6 0> <6 7>
  <7 0> <7 2> <7 5> <7 6>
  Eulerian cycle: 0 1 4 5 2 7 5 1 3 0 6 7 0
  ```

  Draw *concsquares*, label it as given in the input file above, and check the cycle is indeed Eulerian.
- The function *findEulerCycle()* in the lecture notes does not handle disconnected graphs. In a disconnected Eulerian graph, each subgraph has an Eulerian cycle.
  - Modify this function to handle disconnected graphs.
  - With this change, your program should now work for the graph consisting of 2 disconnected triangles:

    ```
    #6
    0 1 0 2 1 2 3 4 3 5 4 5
    ```

    It should now find 2 Eulerian paths:

    ```
    V=6, E=6
    <0 1> <0 2>
    <1 0> <1 2>
    ```

```
<2 0> <2 1>
<3 4> <3 5>
<4 3> <4 5>
<5 3> <5 4>
Eulerian cycle: 0 1 2 0
Eulerian cycle: 3 4 5 3
```

unreachable.c

Write a program that uses a *fixed-point computation* to find all the vertices in a graph that are *unreachable* from the start vertex (assume it to be 0). Note the following:

- the fixed-point computation should be iterative
- **you should not use recursion, or stacks or queues**

If a graph is disconnected:

- then those vertices not reachable (say vertices 8 and 9) should be output as follows:

```
Unreachable vertices = 8 9
```

If a graph is connected then all vertices are reachable and the output is :

```
Unreachable vertices = none
```

For example:

- Here is a graph that consists of 2 disconnected triangles:

  ```
  #6
  0 1 0 2 1 2 3 4 3 5 4 5
  ```

  If the start vertex is 0, then the output should be:

  ```
  V=6, E=6
  <0 1> <0 2>
  <1 0> <1 2>
  <2 0> <2 1>
  <3 4> <3 5>
  <4 3> <4 5>
  <5 3> <5 4>
  Unreachable vertices = 3 4 5
  ```

  because obviously the vertices in the second triangle are not reachable from the first.
- here is a connected graph:

  ```
  #5
  0 1 1 2 2 3 3 4 4 0
  1 3 1 4
  2 4
  ```

  Starting at any vertex, the result should be:

  ```
  V=5, E=8
  <0 1> <0 4>
  <1 0> <1 2> <1 3> <1 4>
  <2 1> <2 3> <2 4>
  <3 1> <3 2> <3 4>
  ```

```
<4 0> <4 1> <4 2> <4 3>
Unreachable vertices = none
```

Week9Exercises (2019-08-22 14:04:43由AlbertNymeyer编辑)