Greedy Algorithms

1. Shortest Paths from a Vertex
   - problem statement
   - Dijkstra's algorithm
   - code for Dijkstra's algorithm
   - cost analysis
2. Minimum Spanning Tree
   - problem statement
   - Prim's algorithm
   - a multiple key map representation for a tree
   - code for Prim's algorithm

MCS 360 Lecture 40
Introduction to Data Structures
Jan Verschelde, 1 December 2017

Greedy Algorithms

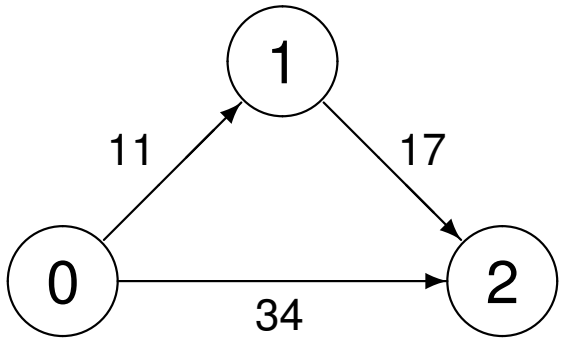
1. Shortest Paths from a Vertex
   - problem statement
   - Dijkstra's algorithm
   - code for Dijkstra's algorithm
   - cost analysis
2. Minimum Spanning Tree
   - problem statement
   - Prim's algorithm
   - a multiple key map representation for a tree
   - code for Prim's algorithm

shortest paths from a vertex

Given is a directed weighted graph, $G = (V, E)$;
and a start vertex $s \in V$.

For every vertex $v \in V$, compute the shortest path from s to v.

Consider: $s = 0$,

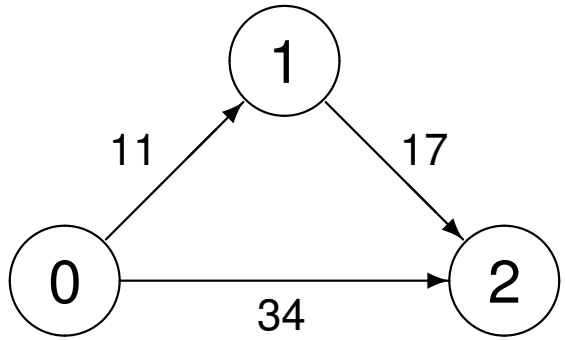


Observe: the direct path from 0 to 2 is 34,
whereas $0 \to 1 \to 2$ costs $11 + 17 = 28$.

The direct path is not always the shortest.

improve an initial solution

Consider again: $s = 0$,



$\mathrm{d}[v]$ is the distance from s to v,
$\mathrm{p}[v]$ is the predecessor in the path to v.
We initialize with the weight of (s, v):

v	$\mathrm{d}[v]$	$\mathrm{p}[v]$
1	11	0
2	34	0

$\longrightarrow$

v	$\mathrm{d}[v]$	$\mathrm{p}[v]$
1	11	0
2	28	1

$\mathrm{d}[1] + w(1,2) = 11 + 17 = 28 < 34 = \mathrm{d}(2)$, so $\mathrm{d}[2] = 28$, $\mathrm{p}[2] = 1$.
The path $0 \to 1 \to 2$ is encoded by $\mathrm{p}[2] = 1, \mathrm{p}[1] = 0$.

Greedy Algorithms

1. Shortest Paths from a Vertex
   - problem statement
   - Dijkstra's algorithm
   - code for Dijkstra's algorithm
   - cost analysis
2. Minimum Spanning Tree
   - problem statement
   - Prim's algorithm
   - a multiple key map representation for a tree
   - code for Prim's algorithm

Dijkstra's algorithm, initialization

Input: $G = (V, E)$, a weighted directed graph;
$s \in V$, a start vertex.
Output: $\mathrm{d}[v]$ shortest distance from s to v;
$\mathrm{p}[v]$ predecessor to v in path from s.

done = $\{s\}$; // *processed vertices*
todo = $\{0, 1, \ldots, n-1\} \setminus \{s\}$;

for all $v \in$ todo do // *initialize with weights of edges from s*
    $\mathrm{p}[v] = s$;
    if $(s, v) \in E$ then
        $\mathrm{d}[v] = w(s, v)$; // *weight of edge* (s, v)
    else
        $\mathrm{d}[v] = \infty$.

Dijkstra's algorithm, process the vertices

while $\texttt{todo} \neq \emptyset$ do

let $u \in \texttt{todo} : \texttt{d}[u] = \min_{x \in \texttt{todo}} \texttt{d}[x]$; // *u is closest vertex*

$\texttt{done} = \texttt{done} \cup \{u\}$;
$\texttt{todo} = \texttt{todo} \setminus \{u\}$;

for all $v \in \texttt{todo}$, $(u, v) \in E$ do

if $\texttt{d}[u] + w(u, v) < \texttt{d}[v]$ then
$\texttt{d}[v] = \texttt{d}[u] + w(u, v)$;
$\texttt{p}[v] = u$.

solving an example

Consider the graph defined by the adjacency matrix:

$$\begin{bmatrix} 0 & 36 & 33 & 63 & 0 \\ 20 & 70 & 74 & 55 & 0 \\ 0 & 0 & 46 & 0 & 0 \\ 0 & 0 & 0 & 0 & 53 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Compute all shortest paths from 0.

v	$\mathrm{d}[v]$	$\mathrm{p}[v]$
1	36	0
2	63	0
3	∞	0
4	∞	0

$u = 1$

v	$\mathrm{d}[v]$	$\mathrm{p}[v]$
1	36	0
2	63	0
3	91	3
4	∞	0

$u = 2$

v	$\mathrm{d}[v]$	$\mathrm{p}[v]$
1	36	0
2	63	0
3	91	3
4	144	4

$u = 3$

Greedy Algorithms

1. Shortest Paths from a Vertex
   - problem statement
   - Dijkstra's algorithm
   - code for Dijkstra's algorithm
   - cost analysis
2. Minimum Spanning Tree
   - problem statement
   - Prim's algorithm
   - a multiple key map representation for a tree
   - code for Prim's algorithm

prototype of the function

```
void shortest_paths
  ( vector< vector<double> >& mat, size_t start,
    vector<double>& dis, vector<size_t>& prv,
    bool verbose );
/*
    Given the adjacency matrix mat for a
    directed weighted graph and a vertex start,
    returns two vectors dis and prv.
    On return, dis[k] the shortest distance of vertex
    start to vertex k, and prv[k] is the previous
    vertex in the shortest path from start to k.
    If verbose, then intermediate values of dis
    and prv are printed. */
```

representing ∞

```
#include <limits>
```

Then we can define ∞ as

```
const double oo = numeric_limits<double>::infinity();
```

The output of

```
cout << oo;
```

is

```
inf
```

code for the initialization

```
void initialize
  ( vector< vector<double> >& mat, size_t start,
    vector<double>& dis, vector<size_t>& prv,
    set<size_t>& done, set<size_t>& todo )
{
    const size_t dim = mat.size();
    const double oo = numeric_limits<double>::infinity();

    for(size_t i=0; i<dim; i++)
    {
       dis[i] = (mat[start][i] == 0.0 ? oo : mat[start][i]);
       prv[i] = start;
    }
    done.insert(start);
    for(size_t i=0; i<dim; i++)
       if(i != start) todo.insert(i);
}
```

structure of the function

```
void shortest_paths
  ( vector< vector<double> >& mat, size_t start,
    vector<double>& dis, vector<size_t>& prv,
    bool verbose )
{
   set<size_t> done; // vertices already processed
   set<size_t> todo; // vertices left to process
   initialize(mat, start, dis, prv, done, todo);

   while(not todo.empty())
   {
      // body of the while loop
   }
}
```

the body of the while loop

```
size_t vtx = idx_min_dis(todo, dis);
if(vtx == dis.size())
{
   if(verbose) cout << "No distance < oo." << endl;
   break;
}
done.insert(vtx); todo.erase(vtx);

for(set<size_t>::const_iterator i=todo.begin();
                                i!=todo.end(); i++)
   if(mat[vtx][*i] != 0.0)
      if(dis[vtx] + mat[vtx][*i] < dis[*i])
      {
         dis[*i] = dis[vtx] + mat[vtx][*i];
         prv[*i] = vtx;
      }
```

running the code

```
The adjacency matrix :
 0.0 0.1 7.5 0.5 9.8 0.0 7.6 1.4 0.0 3.1
 3.1 0.0 8.0 0.0 0.2 0.3 0.0 0.0 6.7 2.0
 0.0 0.0 0.0 0.0 0.0 0.0 0.0 8.9 0.0 7.1
 8.5 0.0 9.6 2.1 3.1 9.4 0.0 0.0 3.7 0.0
 3.2 0.0 7.8 1.2 8.9 0.0 0.0 0.0 5.6 0.0
 4.4 9.3 0.0 3.1 5.5 5.2 0.0 2.0 3.6 0.0
 0.0 0.0 2.2 0.0 0.0 0.0 0.1 7.3 2.8 5.0
 1.1 0.0 4.3 2.2 4.5 0.0 2.8 0.0 3.0 0.1
 5.9 0.7 0.0 9.4 0.0 0.7 1.5 0.0 2.7 6.5
 4.4 3.3 7.6 0.0 7.8 6.4 4.1 0.0 9.5 0.0

The distances :
 inf 0.1 5.7 0.5 0.3 0.4 4.2 1.4 4.0 1.5
The predecessors :
 0 0 7 0 1 1 7 0 5 7
```

```
The shortest paths from 0 :
0 <- 0 : 0.0 : inf
1 <- 0 : 0.1 : 0.1
2 <- 7 <- 0 : 5.7 : 5.7
3 <- 0 : 0.5 : 0.5
4 <- 1 <- 0 : 0.3 : 0.3
5 <- 1 <- 0 : 0.4 : 0.4
6 <- 7 <- 0 : 4.2 : 4.2
7 <- 0 : 1.4 : 1.4
8 <- 5 <- 1 <- 0 : 4.0 : 4.0
9 <- 7 <- 0 : 1.5 : 1.5
```

Greedy Algorithms

1. Shortest Paths from a Vertex
   - problem statement
   - Dijkstra's algorithm
   - code for Dijkstra's algorithm
   - cost analysis
2. Minimum Spanning Tree
   - problem statement
   - Prim's algorithm
   - a multiple key map representation for a tree
   - code for Prim's algorithm

cost analysis

The set `todo` is initialized with $n-1$ elements, $n = |V|$.
The body of the `while` loop contains two more loops:

```
while(not todo.empty())
   size_t vtx = idx_min_dis(todo, dis);
   for(set<size_t>::const_iterator i=todo.begin();
      i!=todo.end(); i++)
```

In each stage of the `while`, one element is removed from `todo`.
The number of steps is therefore

$$n-1+n-2+\cdots+2+1 = \frac{n(n-1)}{2} = O(n^2).$$

Improvement: use priority queues (heaps) for the vector `dis`.

Greedy Algorithms

1. Shortest Paths from a Vertex
   - problem statement
   - Dijkstra's algorithm
   - code for Dijkstra's algorithm
   - cost analysis

2. Minimum Spanning Tree
   - problem statement
   - Prim's algorithm
   - a multiple key map representation for a tree
   - code for Prim's algorithm

the minimum spanning tree

Given a graph $G = (V, E)$,
a *spanning tree for G* is a subset of E such that

1. between any two vertices there is only one edge; and
2. all the vertices are connected.

The *cost* of a spanning tree is the sum of all the weights of the edges.

The *minimum spanning tree* is the spanning tree with the smallest cost.

Prim's algorithm solves this problem.
Prim's algorithm is very similar to Dijkstra's algorithm.

Both algorithms are examples of a *greedy algorithm*,
at each stage it makes the locally optimal choice.

Greedy Algorithms

1. Shortest Paths from a Vertex
   - problem statement
   - Dijkstra's algorithm
   - code for Dijkstra's algorithm
   - cost analysis

2. Minimum Spanning Tree
   - problem statement
   - Prim's algorithm
   - a multiple key map representation for a tree
   - code for Prim's algorithm

Prim's algorithm, initialization

Input: $G = (V, E)$, a weighted undirected graph;
$s \in V$, a start vertex.
Output: $\mathrm{d}[v]$ shortest edge from a vertex to v;
$\mathrm{p}[v]$ source vertex for the shortest edge to v;
Set T of edges of the minimum spanning tree.

done = $\{s\}$; // *processed vertices*
todo = $\{0, 1, \ldots, n-1\} \setminus \{s\}$;

for all $v \in$ todo do // *initialize with weights of edges from s*
    $\mathrm{p}[v] = s$;
    if $(s, v) \in E$ then
        $\mathrm{d}[v] = w(s, v)$; // *weight of edge* (s, v)
    else
        $\mathrm{d}[v] = \infty$.

Prim's algorithm, process the vertices

$T = \emptyset$;
while $\texttt{todo} \neq \emptyset$ do

let $u \in \texttt{todo} : \texttt{d}[u] = \min_{x \in \texttt{todo}} \texttt{d}[x]$; // *u vertex on shortest edge*

$\texttt{done} = \texttt{done} \cup \{u\}$;
$\texttt{todo} = \texttt{todo} \setminus \{u\}$;
$T = T \cup (u, \texttt{p}[u])$;

for all $v \in \texttt{todo}$, $(u, v) \in E$ do

if $w(u, v) < \texttt{d}[v]$ then
$\texttt{d}[v] = w(u, v)$;
$\texttt{p}[v] = u$.

Greedy Algorithms

1. Shortest Paths from a Vertex
   - problem statement
   - Dijkstra's algorithm
   - code for Dijkstra's algorithm
   - cost analysis

2. Minimum Spanning Tree
   - problem statement
   - Prim's algorithm
   - a multiple key map representation for a tree
   - code for Prim's algorithm

storing a tree in a multiple key map

Consider the edges in a tree:

`(0,2)(0,9)(0,4)(3,5)(4,3)(5,7)(5,8)(6,1)(9,6)`

The edges are represented by pairs: source and destination.
With `multimap<size_t, size_t> tree`, a query `map[source]` for the `destination` starting at `source` is efficient.
Such queries are needed to print the tree:

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

printing the tree

```
void write_tree
  ( multimap<size_t, size_t>& tree,
    size_t root, size_t level )
{
   for(size_t k=0; k<level; k++) cout << "   ";
   cout << root << endl;

   for(multimap<size_t, size_t>::const_iterator
        it = tree.begin();
        it != tree.end(); it++)
      if(it->first == root)
         write_tree(tree, it->second, level+1);
}
```

Greedy Algorithms

1. Shortest Paths from a Vertex
   - problem statement
   - Dijkstra's algorithm
   - code for Dijkstra's algorithm
   - cost analysis

2. Minimum Spanning Tree
   - problem statement
   - Prim's algorithm
   - a multiple key map representation for a tree
   - code for Prim's algorithm

prototype of the function

```
void minimum_spanning_tree
  ( vector< vector<double> >& mat, size_t start,
    vector<double>& dis, vector<size_t>& prv,
    multimap<size_t, size_t>& mintree,  bool verbose );
/*
    Given the adjacency matrix mat for an
    undirected weighted graph and a vertex start,
    returns two vectors dis, prv, and the set
    of edges in the minimum spanning tree.
    On return, dis[k] the distance of the shortest
    edge to vertex k, prv[k] is the source of the
    shortest edge to vertex k, and mintree contains
    the edges in the minimum spanning tree,
    the index in the multiple key map is the source
    vertex of each edge.  If verbose, intermediate
    values of dis and prv are printed. */
```

new code in the definition of the loop

The initialization is the same as in Dijkstra's algorithm.

The new code concerns the multiple key map:

```
pair<size_t, size_t> edge;
edge.first = prv[vtx];
edge.second = vtx;
mintree.insert(edge);
```

The rest of the loop is similar to Dijkstra's algorithm.

running the code

```
The adjacency matrix :                          Minimum spanning tree :
 0.0 0.0 1.9 0.0 8.2 0.0 0.0 9.9 0.0 2.5   0
 0.0 8.7 0.0 0.0 0.0 0.0 3.1 0.0 0.0 0.0     2
 1.9 0.0 4.7 0.0 0.0 0.0 0.0 0.0 0.0 0.0     9
 0.0 0.0 0.0 0.0 4.7 4.2 0.0 0.0 0.0 0.0       6
 8.2 0.0 0.0 4.7 0.0 0.0 1.2 7.4 0.0 0.0         1
 0.0 0.0 0.0 4.2 0.0 0.0 0.0 3.0 9.5 0.0     4
 0.0 3.1 0.0 0.0 1.2 0.0 8.9 0.0 0.0 9.2       3
 9.9 0.0 0.0 0.0 7.4 3.0 0.0 0.0 6.9 0.0         5
 0.0 0.0 0.0 0.0 0.0 9.5 0.0 6.9 1.1 0.0           7
 2.5 0.0 0.0 0.0 0.0 0.0 9.2 0.0 0.0 0.0           8
The edges in the minimum spanning tree :
(0,2)(0,9)(0,4)(3,5)(4,3)(5,7)(5,8)(6,1)(9,6)
The cost of the spanning tree : 46.3
Sum of distances of shortest edges : 46.3
```

Summary + Exercises

The shortest paths and minimum spanning trees are problems which are solved well by greedy algorithms.

Exercises:

1. Run Dijkstra's algorithm for large dimensions, e.g.: $n = 1024, 2048, 4096, \ldots$. Record timings of running the code. Do you notice the $O(n^2)$ cost?
2. Improve the implementation of Dijkstra's algorithm and store `dis` with a heap. Verify that your code is correct by computing the shortest paths twice, once with and once without a heap.
3. Run your improved version of Dijkstra's algorithm for large dimensions, e.g.: $n = 1024, 2048, 4096, \ldots$. Compare the running times with the times of exercise 1.
4. What is the cost of the improved version of Dijkstra's algorithm, when `dis` is stored with a heap?