SHORTEST PATH DIJKSTRA ALGORITHM

Ndindi Kilonzo Rachael

Database and Algorithms
MAT0040
Universita Degli Studi di Torino
MSc. Stochastics and Data Science
08/06/2021

Road Network Data

- Road Network Data Source is DIMACS (Center for Discrete Mathematics & Theoretical Computer Science)
- http://www.diag.uniroma1.it//challenge9/data/rome/rome99.gr
- Directional road of the City of Rome as we've been provided with source and destination vertices.
- The structure is : Source, Destinations and Weights with 3353 vertices and 8870 edges.
- Literature: Guidline Micah Schute on Cantor Paradise.

Dijkstra Algorithm Implementation

- Road network graph Implemented through adjacency matrix:
    - Intuitive and the image is easier to visually conjure up.
- Vertex object used and their indices show the row / column in the adjacency matrix.
- Helper method to allows the use of either the index of a vertex or the object itself
- The Graph implementation: links (from, to, direction)

  Add links

  weights

Djikstra Algorithm

- Set provisional distance of all vertices from the source to infinity.
- Define an empty set of visited vertices.
- Set provisional distance of the source to equal 0. There's an array representing the hops taken to just include the source itself.
- While we have not visited all vertices:
    - Set current vertex to the one with the smallest provisional distance in the entire graph-
    - Add current to the seen visited set
    - Update the provisional distance of each of current vertices neighbors to be the distance from current vertex to source * the edge length from current to that neighbor
    - End While

Dijkstra Algorithm

- The source vertex was arbitrarily selected as the 1st row 1st column entry. All other vertices are destinations and we are looking for the shortest .
- Implement the Dijkstra
- Issue = Used a queue, O(n) operation n times. We have $O(n^2)$

=Adjacency matrix used, we looked through an entire row of size n

to find links. Another O(n) time.

Possible solutions =Use Fibonacci heap O(1)

=Use adjacency list instead

Fibonnaci Heap

- Heap ordered trees (parent larger than children)
- **<u>Pointer</u>** = To minimum vertex
- Set of marked vertices
- root list
- degree

Fibonacci Heaps

PSEUDOCODE

1) Make-Fibonacci-Heap()

```
n[H] = 0
return H
```

O(1) amortized cost

Fibonacci Heap

2) **<u>Insert</u>**

Create a new singleton tree.

Add to root list; update min pointer (if necessary).

<u>Pseudocode</u>

Fibonacci-Heap-Insert(H,x)

```
degree[x] = 0
p[x] = NIL
child[x] = NIL
left[x] = x
right[x] = x
mark[x] = FALSE
concatenate the root list containing x
with root list H
if min[H] = NIL or key[x]<key[min[H]]
        then min[H] := x
n[H]:= n[H]+1
```

Amortized : O(1)

Fibonacci Heap

3) **Fibonacci-Heap-Minimum(H)**

return min[H]

O(1) actual cost

4)**Delete min.**

Delete min; meld its children into root list; update min.

Consolidate trees so that no two roots have same rank.

O(log n) actual cost

PSEUDOCODE

```
z:= min[H]
if x <> NIL
        then for each child x of z
            do add x to the root list of H
                p[x]:= NIL
         remove z from the root list of H
         if z = right[z]
            then min[H]:=NIL
            else min[H]:=right[z]
                 CONSOLIDATE(H)
         n[H] := n[H]-1
return z
```

Fibonacci Heap

5) **Decrease Key**

Intuition for deceasing the key of node x.

1. If heap-order is not violated, just decrease the key of x.
2. Otherwise, cut tree rooted at x and meld into root list.
3. To keep trees flat: as soon as a vertex has its second child cut, cut it off and meld into root list (and unmark it).

O(1) cost

- Case 1. [heap order not violated]

1. Decrease key of x.
2. Change heap min pointer (if necessary).

Case 2a. [heap order violated]

1. Decrease key of x.
2. Cut tree rooted at x, meld into root list, and unmark.

FIBONACCI HEAP

3. If parent p of x is unmarked (hasn't yet lost a child), mark it; Otherwise, cut p, meld into root list, and unmark (and do so recursively for all ancestors that lose a second child).

3. If parent p of x is unmarked (hasn't yet lost a child), mark it;Otherwise, cut p, meld into root list, and unmark (and do so recursively for all ancestors that lose a second child).

- Case 2b. [heap order violated]

1. Decrease key of x.
2. Cut tree rooted at x, meld into root list, and unmark.

Fibonacci Heap

Fibonacci-Heap-Link(H,y,x)

remove y from the root list of H
make y a child of x
degree[x] := degree[x] + 1
mark[y] := FALSE

CONSOLIDATE(H)

```
For i:=0 to D(n[H])
Do A[i] := NIL
For each node w in the root list of H
        do x:= w
        d:= degree[x]
        while A[d] <> NIL
```

FIBONACCI HEAP

```
do y:=A[d]
     if key[x]>key[y]
     then exchange x<->y
      Fibonacci-Heap-Link(H, y, x)
       A[d]:=NIL
        d:=d+1
       A[d]:=x
min[H]:=NIL
for i:=0 to D(n[H])
do if A[i]<> NIL
          then add A[i] to the root list of H
               if min[H] = NIL or key[A[i]]<key[min[H]]
                  then min[H]:= A[i]
```

FIBONACCI HEAP

- **Fibonacci-Heap-Union(H1,H2)**

```
H := Make-Fibonacci-Heap()
min[H] := min[H1]
Concatenate the root list of H2 with the root list of H
if (min[H1] = NIL) or (min[H2] <> NIL and min[H2] < min[H1])
   then min[H] := min[H2]
n[H] := n[H1] + n[H2]
free the objects H1 and H2
return H
```

FIBONACCI HEAP

- **Fibonacci-Heap-Union(H1,H2)**

```
H := Make-Fibonacci-Heap()
min[H] := min[H1]
Concatenate the root list of H2 with the root list of H
if (min[H1] = NIL) or (min[H2] <> NIL and min[H2] < min[H1])
   then min[H] := min[H2]
n[H] := n[H1] + n[H2]
free the objects H1 and H2
return H
```

FIBONACCI HEAP

- **Fibonacci-Heap-Decrease-Key(H,x,k)**

```
        if k > key[x]
 then error "new key is greater than current key"
        key[x] := k
        y := p[x]
        if y <> NIL and key[x]<key[y]
          then CUT(H, x, y)
             CASCADING-CUT(H,y)
        if key[x]<key[min[H]]
          then min[H] := x
```

- CUT(H,x,y)

```
        Remove x from the child list of y,
decrementing degree[y]
        Add x to the root list of H
        p[x]:= NIL
        mark[x]:= FALSE
```

FIBONACCI HEAP

- **CASCADING-CUT(H,y)**

```
z:= p[y]
if z <> NIL
  then if mark[y] = FALSE
         then mark[y]:= TRUE
         else CUT(H, y, z)
              CASCADING-CUT(H, z)
```