SCC0602 - Algoritmos e Estruturas de Dados I

Shortest Path

Professor: André C. P. L. F. de Carvalho, ICMC-USP
PAE: Rafael Martins D'Addio
Monitor: Joao Pedro Rodrigues Mattos

Today

- Single-source shortest paths in weighted graphs
  - Shortest-Path Problems
  - Properties of Shortest Paths
  - Relaxation
  - Dijkstra Algorithm
  - Bellman-Ford Algorithm
  - Shortest-Paths in DAGs

Introduction

- Suppose you are driving your car from the University to your favourite pub
  - There are different ways to get to the pub
    - Each goes through different streets and avenues
  - You forgot to fill the petrol tank and your fuel display is broken
  - Which way do you choose?

Shortest Path

- Possible paths:
  - Directed graph (Digraph) $G = (V,E)$ with weight function W: $E \rightarrow R$ (assign real values to edges)
- Weight of path $p = v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k$:

$$w(p) = \sum_{i=1}^{k-1} w(v_i, v_{i+1})$$

- Shortest path: path of the minimum total weight
- Several applications:
  - Static/dynamic network routing
  - Robot motion planning
  - Map/route generation in traffic

Shortest-Path Problems

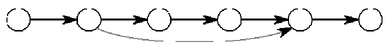
- Unweighted shortest-paths: BFS
- Weighted shortest-paths:
  - **Single-source (single-destination)**
    - Find a shortest path from a given source (vertex s) to each of the vertices
  - **Single-pair**
    - Given two vertices, find a shortest path between them
      - Solution to single-source problem solves this problem efficiently
  - **All-pairs**
    - Find shortest-paths for every pair of vertices
      - Dynamic programming algorithm

Shortest-Path Problems

- The ***single-source shortest-paths problem***:
  - Given a graph G = (V,E), find a shortest path from a given ***source*** vertex s to V to each vertex v $\in$ V

Optimal Substructure

- Theorem: subpaths of shortest paths are shortest paths
- Proof ("cut and paste")
  - If some subpath is not the shortest path
    - Substitute current subpath by the shorter subpath and create a shorter total path



André de Carvalho - ICMC/USP 7

Negative Weights and Cycles?

- Negative edges are OK
  - As long as there are no *negative weight cycles*
    - Otherwise paths with arbitrary small-value "lengths" would be possible
- Shortest-paths can have no cycles
  - Otherwise they could be improved by removing cycles
  - Any shortest-path in graph *G* with n vertices can have no more than *n* – 1 edges

André de Carvalho - ICMC/USP 8

Relaxation

- Used by the shortest-path algorithms
- For each vertex *v* maintain an attribute *d[v]*
  - Shortest-path estimate
  - Upper bound on the weight of a shortest path from source vertex *s* to vertex *v*

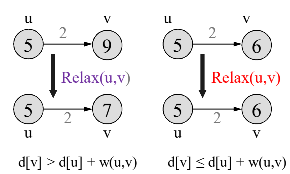
```
INITIALIZE-Single-Source(G, s)
for each v ∈ V[G] do
    d[v] ← ∞;
    π[v] ← NIL
d[s] ← 0
```

```
RELAX (u,v,w)
if (d[v] > d[u]+w(u,v)
then d[v] ← d[u]+w(u,v)
        π[v] ← u
```

André de Carvalho - ICMC/USP 9

Relaxation

- Control *d[v]* for each vertex *v* in the graph
  - Estimate of the shortest path from *s*, initialized to ∞ at the start
- Relaxing an edge (*u,v*):
  - Testing whether it is possible to improve the shortest path to *v* found so far by going through *u*



André de Carvalho - ICMC/USP 10

Dijkstra Algorithm

- For single-source shortest-path problems
- Works only when all graph edge weights are nonnegative
- Greedy algorithm, like the Prim algorithm for MST
- If all weights = 1, like breadth-first search
  - Therefore, BFS can be used

André de Carvalho - ICMC/USP 11

Dijkstra Algorithm

- Use *Q*, a priority queue (PQ) ADT keyed by *d[v]*
  - PQ is re-organized whenever some **d** decreases
    - BFS use a FIFO queue
- Basic idea
  - Maintain a set *S* of solved vertices
  - At each step select "closest" vertex *u*, add it to *S*, and relax all edges from *u*
  - Invariant: Q = V - S

André de Carvalho - ICMC/USP 12

Dijkstra Algorithm

```
DIJKSTRA (G, w, s)
INITIALIZE-SINGLE-SOURCE(V, s) // d and π values
S ← ∅ // Set S is used to explain the algorithm
Q ← V[G]  // Q is a priority queue ADT
while Q ≠ ∅ do
    u ← EXTRACT-MIN(Q)  // First iteration, u ← s
    S ← S ∪ {u}
    for each vertex v ∈ Adj[u] do  // Relaxing edges
        RELAX(u, v, w)
```

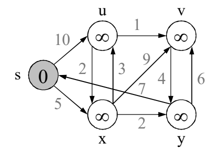
```
INITIALIZE-Single-Source(G, s)
for each v ∈ V[G] do
    d[v] ← ∞;
    π[v] ← NIL
d[s] ← 0
```

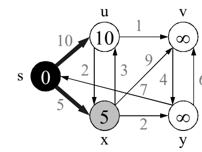
Running time:
$O(VlgV + ElgV) = O(ElgV)$

Example

```
DIJKSTRA (G, w, s)
INITIALIZE-SINGLE-SOURCE(V, s)
S ← ∅
  Q ← V[G]
while Q ≠ ∅ do
    u ← EXTRACT-MIN(Q)
    S ← S ∪ {u}
    for each vertex v ∈ Adj[u] do
        RELAX(u, v, w)
```

```
RELAX (u,v,w)
if (d[v] > d[u]+w(u,v)
then d[v] ← d[u]+w(u,v)
     π[v] ← u
```

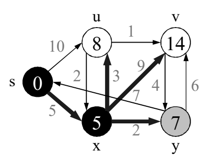


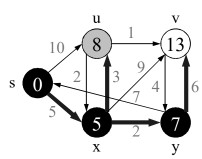


Example

```
DIJKSTRA (G, w, s)
INITIALIZE-SINGLE-SOURCE(V, s)
S ← ∅
  Q ← V[G]
while Q ≠ ∅ do
    u ← EXTRACT-MIN(Q)
    S ← S ∪ {u}
    for each vertex v ∈ Adj[u] do
        RELAX(u, v, w)
```

```
RELAX (u,v,w)
if (d[v] > d[u]+w(u,v)
then d[v] ← d[u]+w(u,v)
     π[v] ← u
```

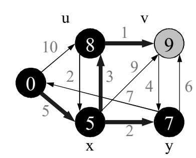


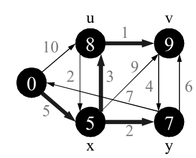


Example

```
DIJKSTRA (G, w, s)
INITIALIZE-SINGLE-SOURCE(V, s)
S ← ∅
  Q ← V[G]
while Q ≠ ∅ do
    u ← EXTRACT-MIN(Q)
    S ← S ∪ {u}
    for each vertex v ∈ Adj[u] do
        RELAX(u, v, w)
```

```
RELAX (u,v,w)
if (d[v] > d[u]+w(u,v)
then d[v] ← d[u]+w(u,v)
     π[v] ← u
```





Dijkstra Running Time

- Extract-Min executed $|V|$ time
- Decrease-Key executed $|E|$ time
- Time = $|V|\ T_{\text{Extract-Min}} + |E|\ T_{\text{Decrease-Key}}$
- T depends on different Q implementations

Q	T(Extract-Min)	T(Decrease-Key)	Total
array	$O(V)$	$O(1)$	$O(V^2)$
binary heap	$O(\lg V)$	$O(\lg V)$	$O(E \lg V)$
Fibonacci heap	$O(\lg V)$	$O(1)$ (amort.)	$O(V \lg V + E)$

About Dijkstra

- Taught introductory computer courses without using computers
  - Pencil and paper programming
- Would not read his e-mail
- His staff had to print out his e-mails and put them in his mailbox

Bellman-Ford Algorithm

- Dijkstra algorithm does not work when there are negative edges:
  - Cannot be greedy any more, since the lengths of paths will only increase in the future
- Bellman-Ford algorithm
  - Deal with negative edges in single-source shortest-path problems
  - Detects negative cycles (returns *false*) or returns the shortest path-tree

Bellman-Ford Algorithm

- Returns a boolean value
  - FALSE if graph contains a negative-weight cycle that is reachable from the source
    - Indicates that there is no solution
  - TRUE otherwise
    - Algorithm returns the shortest paths and their weights
- Relaxation progressively reduces estimate of d[v] until the shortest path-weight is obtained
  - Estimates the weight of a shortest path from the source s to each vertex v ∈ V

Bellman-Ford Algorithm

```
BELLMAN-FORD (G, w, s)
INITIALIZE-SINGLE-SOURCE(V, s)        ← Θ(V)
for i ← 1 to |V[G]| - 1 do            ← O(V) } O(VE)
    for each edge (u, v) ∈ E do        ← O(E) }
        RELAX (u, v, w)
for each edge (u, v) ∈ E do           ← O(E)
    if d[v] > d[u] + w(u, v)
    then return FALSE
return TRUE
```

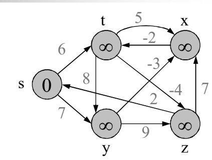
```
INITIALIZE-SINGLE-SOURCE(G, s)
for each v ∈ V[G] do
    d[v] ← ∞;
    π[v] ← NIL
d[s] ← 0
```

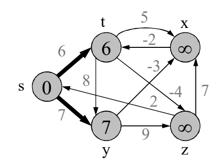
Running time:
O(V+VE+E)=O(VE)

Example

```
BELLMAN-FORD (G, w, s)
INITIALIZE-SINGLE-SOURCE(V, s)
for i ← 1 to |V[G]| - 1 do
    for each edge (u, v) ∈ E do
        RELAX (u, v, w)
for each edge (u, v) ∈ E do
    if d[v] > d[u] + w(u, v)
    then return FALSE
return TRUE
```

```
RELAX (u,v,w)
if (d[v] > d[u]+w(u,v)
then d[v] ← d[u]+w(u,v)
     π[v] ← u
```

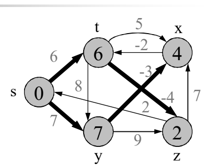


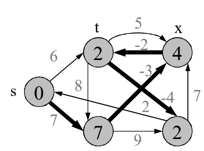


Example

```
BELLMAN-FORD (G, w, s)
INITIALIZE-SINGLE-SOURCE(V, s)
for i ← 1 to |V[G]| - 1 do
    for each edge (u, v) ∈ E do
        RELAX (u, v, w)
for each edge (u, v) ∈ E do
    if d[v] > d[u] + w(u, v)
    then return FALSE
return TRUE
```

```
RELAX (u,v,w)
if (d[v] > d[u]+w(u,v)
then d[v] ← d[u]+w(u,v)
     π[v] ← u
```

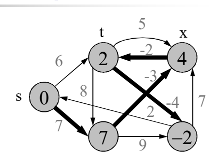




Bellman-Ford Example

```
BELLMAN-FORD (G, w, s)
INITIALIZE-SINGLE-SOURCE(V, s)
for i ← 1 to |V[G]| - 1 do
    for each edge (u, v) ∈ E do
        RELAX (u, v, w)
for each edge (u, v) ∈ E do
    if d[v] > d[u] + w(u, v)
    then return FALSE
return TRUE
```

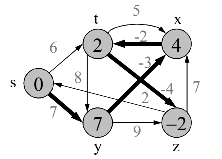
```
RELAX (u,v,w)
if (d[v] > d[u]+w(u,v)
then d[v] ← d[u]+w(u,v)
     π[v] ← u
```



Bellman-Ford Example

```
BELLMAN-FORD (G, w, s)
INITIALIZE-SINGLE-SOURCE(V, s)
for i ← 1 to |V[G]| - 1 do
    for each edge (u, v) ∈ E do
        RELAX (u, v, w)
for each edge (u, v) ∈ E do
    if d[v] > d[u] + w(u, v)
    then return FALSE
return TRUE
```

```
RELAX (u,v,w)
if (d[v] > d[u]+w(u,v)
then d[v] ← d[u]+w(u,v)
     π[v] ← u
```



(t,x), (t,y), (t,z), (x,t), (y,x) (y,z), (z,x), (z,s), (s,t), (s,y)
Returns TRUE

- Bellman-Ford running time:
  - (|V|-1)|E| + |E| = Θ(VE)

Shortest-Path in DAGs

- Find shortest paths in DAGs is much easier
  - Shortest paths are always well defined in a DAG
  - Even if there are negative-weight edges, there are no negative-weight cycles
  - Because it is easy to find an order in which to do relaxations: Topological sorting
    - According to the topological sorting of its vertices, shortest path of a single source can be calculated in Θ(V+E)

Topological sorting

- Produces a sequence of vertices
  - If there is a path from vertex *u* to vertex *v*, *u* precedes *v* in the topological sort
- Algorithm makes just one pass over the vertices in the topologically sorted order
  - As each vertex is processed, each edge that leaves the vertex is relaxed
  - The predecessor of a vertex v is always processed before v is processed

Shortest-Path in DAGs

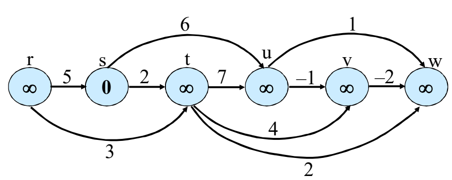
```
DAG-Shortest-Paths(G,w,s)
Topologically sort vertices in G                          O(V + E)
INITIALIZA-SINGLE-SOURCE (G, s) // d and π values
for each vertex u taken in topologically sorted order do  O(V + E)
    for each vertex v in Adj[u] do
        Relax(u, v, w)
```

```
INITIALIZE-SINGLE-SOURCE(G, s)
for each v ∈ V[G] do
    d[v] ← ∞;
    π[v] ← NIL
d[s] ← 0
```

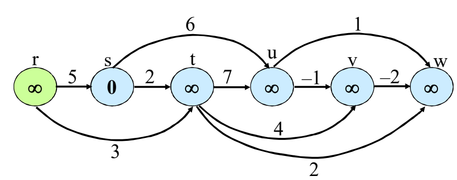
```
Topological-Sorting (L)
G ← ∅
while (L is not empty)
  find a vertex u without incoming edges
  delete u from L
  add u to G
return the linked list of vertices
```

- Running time:
  - Θ(V+E): only one relaxation for each edge

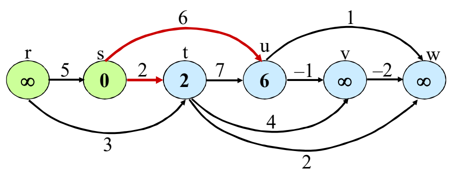
Example



Example

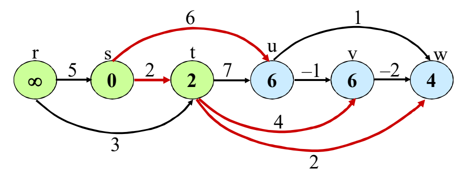


Example



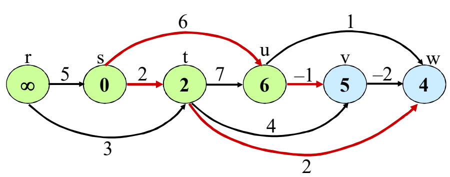
André de Carvalho - ICMC/USP 31

Example



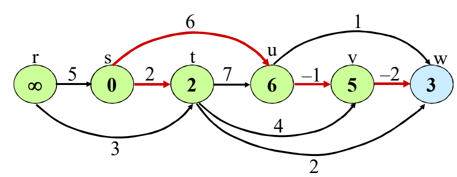
André de Carvalho - ICMC/USP 32

Example



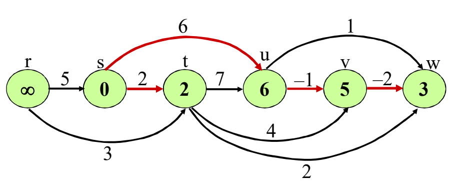
André de Carvalho - ICMC/USP 33

Example



André de Carvalho - ICMC/USP 34

Example



André de Carvalho - ICMC/USP 35

Acknowledgement

- A large part of this material were adapted from
  - Simonas Šaltenis, Algorithms and Data Structures, Aalborg University, Denmark
  - Mary Wootters, Design and Analysis of Algorithms, Stanford University, USA
  - George Bebis, Analysis of Algorithms CS 477/677, University of Nevada, Reno
  - David A. Plaisted, Information Comp 550-001, University of North Carolina at Chapel Hill

André de Carvalho - ICMC/USP 36

Questions

André de Carvalho - ICMC/USP 37