Parma University, Math. Department

Algorithms and Data Structures
Single source shortest path

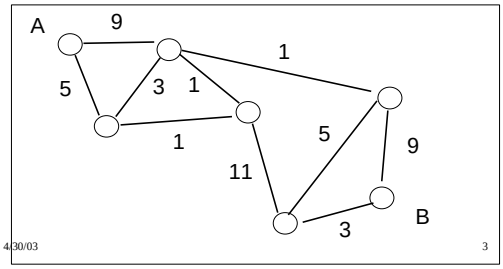
Marco Pellegrini

Summary

- This lecture: shortest path in weighted directed graphs
- Material in CLR 25.1-2.

Weighted graph

- If you want to go from A to B in a graph where the edges are roads and the nodes are crossroads, then the shortest path is given by summing the length of the arcs, not by the number of arcs.



Weighted graphs (2)

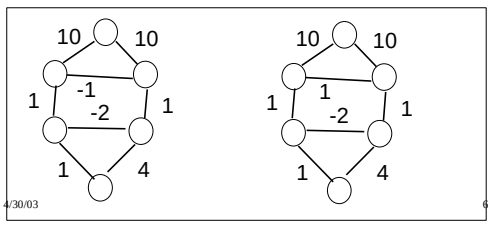
- To model this situation we introduce weighted graphs, consisting in a directed graph G=(V,E) and in a weight function w: E→R.
- For a path p = $(v_1, v_2,..., v_k)$ the weight of the path is the sum of the weight of the edges of the path. $W(p) = \Sigma_{i=1..k-1} w(v_i,v_{i+1})$
- The shortest path between a vertex v and a vertex u in G is the path with smallest weight, or infinity if no path exists between v and u.
- We call D(v,u) the weight of the shortest path between v and u.

Shortest path problems

- (A) Single-source shortest path problem: Given a vertex s (source) find the shortest path weight from s to any node in V.
- (B) Single-pair shortest path problem: Given vertices s and t, find the shortest path length between s and t.
- (C) All-pair shortest path problem: Find the shortest distance between any pair of vertices in V.
- No algorithm to solve (B) is known that is asymptotically faster than the best algorithm to solve (A).
- We shall see more of (C) later in the course.

Negative weights.

- If G has a cycle of negative weight, the shortest path problems are not well defined.
- If G has negative weights, but no cycle of negative weight, the problem is well defined (although more difficult)

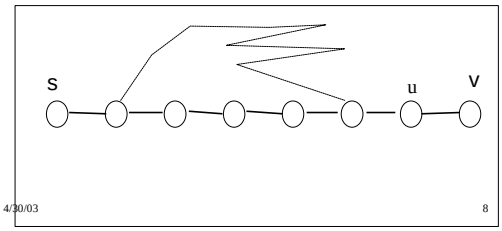


Dijkstra algorithm for SSSP problem

- We assume al weights to be positive or zero.
- We keep a function (that is an array) d[v] that for each v in V contains an upper estimate of the shortest distance D(s,v).
- We see the an operation called *Relaxation(u,v)* on two adjacent nodes u and v.
- Intuitively Relaxation test wether at a certain point it is conceivable that the shortest path from s to v passes through u.
- We shall see useful properties of Relaxation and of shortest path which we shall use to prove to correctness of Dijkstra's algorithm.

Shortest path properties:

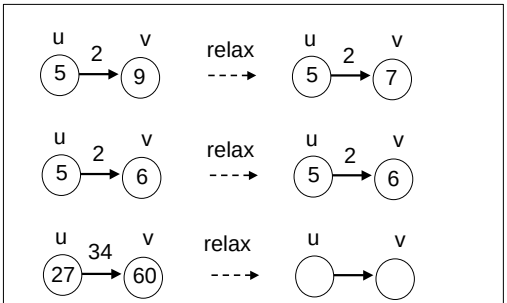
- Lemma 25.1: Every sub-path of a shortest path is a shortest-path.
- Lemma 25.2: If (s,...,u,v) is a shortest path between s and v, then D(s,v) = D(s,u) + w(u,v).



Properties of shortest-path (cntd)

- Lemma 25.3: If u, and v are any two adjacent nodes then: D(s,v) ≤ D(s,u) + w(u,v).
- Initialization. To make the algorithm work d[v] must always be larger or equal to D(s,v) so initially
-
```
Initialize(G,s)
  FOR each v IN V DO
     d[v] := infinity
  END FOR
  d[s] := 0
```
-
```
Relax(u,v)
  IF d[v] > d[u] + w(u,v)
    THEN d[v] := d[u] + w(u,v)
  END IF
```


Examples

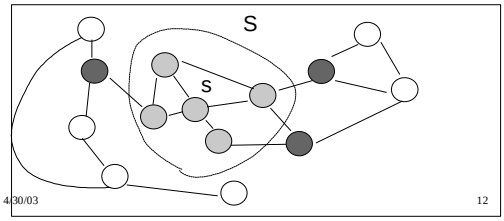


Properties of relaxation:

- Lemma 25.4 After the call Relax(u,v) it is true that d[v] ≤ d[u] + w(u,v).
- Lemma 25.5 After the initialization, if we update d[] only by relaxation operations then
  (1) always d[u] ≥D(s,u) for any u at any time of the algorithm
  (2) if at a certain time d[u]=D(s,u) then d[u] does not change from that moment.
- Lemma 25.7 Suppose that (s,...,u,v) is a shortest path from s to u and Relax(u,v) is executed when d[u] = D(s,u), then after the execution d[v] = D(s,v).

Dijkstra's algorithm, main idea

- The main idea is to have a set S of nodes for which we are sure that the shortest path has been computed and we try to make it larger.
- To make S larger we choose the v in V/S with the smallest value of d[v].

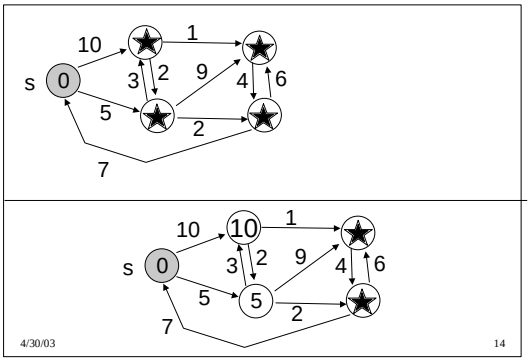


Dijkstra's algorithm

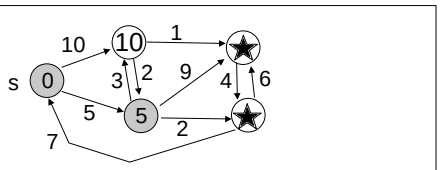
We use an auxiliary priority queue Q, which is based on the d[v] value and allows operations EXTRACT-MIN and DECREASE-KEY(Q, vertex, new-key)

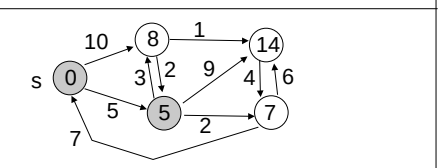
```
Dijkstra(G,s)
    Initialize(G)
    S := empty-set
    Q := make-priority-queue(V)
    WHILE not empty(Q) DO
      u := extract-min(Q)
      S := S + {u}
      FOR each v adjacent to u DO
        Relax(u,v)
        decrease-key(Q, v, d[v])
      END FOR
    END WHILE
```


Example 1.

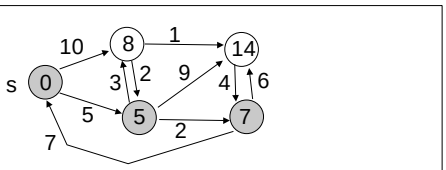


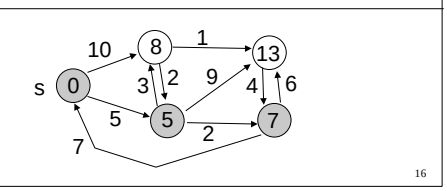
Example (2)



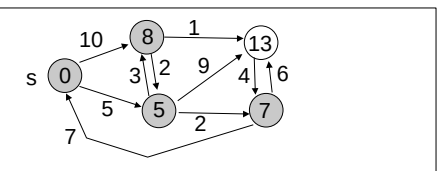


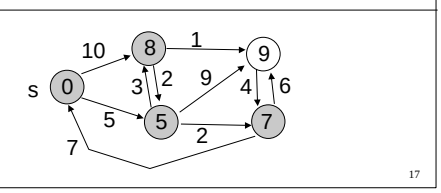
Example (3)



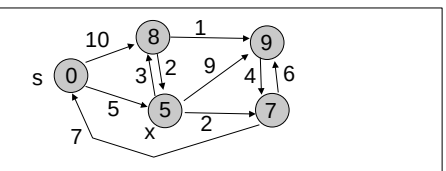


Example (4)





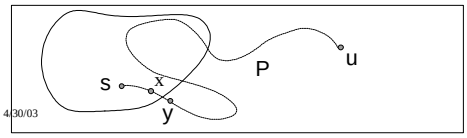
Example (5)



Exercise: simulate Dijkstra's algorithm on the same graph, starting from node x

Proof of correctness

- It suffices to show that when u is inserted in S, d[u] = D(s,u).
- Suppose by contradiction that u is the first node inserted in S for which d[u] ≠D(s,u).
- u cannot be s because d[s]=0=D(s,s). There must be a path from s to u, otherwise d[u]=infinity=D(s,u).
- So there are paths from s to u. Let us call P a shortest such path.



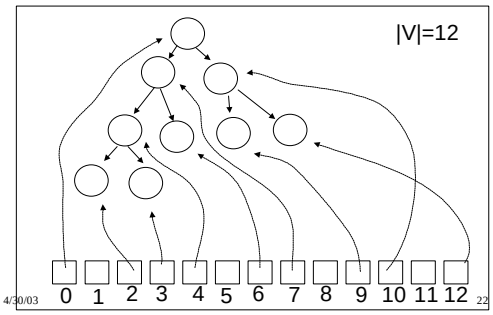
- Call y the first node found on path P outside the set S and x the predecessor of y. So x is in S.
- Since x was in S before we placed u in S we have d[x] = D(s,x). But, being P the shortest path from s to u, also it contains a path (s,..,x,y). So by Lemma 25.7 d[y] = D(s,y) after the call of Relax(x,y).
- But y is before u in the shortest path P from s to u. Therefore D(s,y) ≤ D(s,u), because all weights are positive and add up.
- We have inequalities: d[y] = D(s,y) ≤ D(s,u) ≤ d[u].
- If d[y] < d[u] the algorithms gets y from Q, not u. Contraddiction.
- Or d[y]=d[u] but then also d[u]=D[s,u] which contradicts our hypothesis.

Time analysis

- Since a node is put in the queue Q once and then only deleted the WHILE Loop is executed O(|V|) times.
- In the loop for each vertex we scan its adjacency list so we visit each edge only once, that is the inner FOR loop is executed O(|E|) times.
- Relax is a O(1) time operation.
- So the time bound is O(|V|+|E|) times the cost of one Extract-min and decrease-key operation.
- On a heap implementation extract-min takes time O(log n).
- Decrease-key? The problem here is that even if we have the name of a node we do not know where it is in the heap. Sequential search on the array takes time O(|V|).

A smart heap.

- Combine a heap with a direct-address table.



Modification to the Heap procedures

- We have to Modify Heapify, and extract-min to update also the direct address table.
- We can use the direct address table to access in time O(1) an element in the heap by its name. Then we can decrease its key value and then move it up to satisfy the heap property.
- Homework 4.9 Write pseudocode for operations: extract-min, decrease-key, heapify which run in time O(log n) on a heap implemented as an array + auxiliary direct-address table.
- If you do the HW 4.9 then you can solve SSSP in time O((|E| + |V|) log n)).

Conclusion

We have seen Directed Weighted Graphs as a natural tool to model certain optimization problems.

We have seen useful properties of shortest paths and of the Relax procedure which is used to update an upper bound estimate on shortest path lengths from a source to a node.

We have seen Dijkstra's algorithm for the SSSP problem which uses relaxation and an auxiliary (smart) priority queue.

We have proved the correctness of the Dijkstra's algorithm and analyzed the time bound.

Finally, by simple modification of the code (similar to the trick used in the BFS code) we can record a tree representing all shortest paths form s to any node in G.