NANYANG TECHNOLOGICAL UNIVERSITY

CE2001/ CZ2001: Algorithms

Graphs

Ke Yiping, Kelly

NANYANG TECHNOLOGICAL UNIVERSITY

Greedy Algorithms

NANYANG TECHNOLOGICAL UNIVERSITY

Learning Objectives

At the end of this lecture, students should be able to:

- Explain the strategy of Greedy algorithms
- Solve single-source shortest paths problem using Dijkstra's algorithm
- Prove the correctness of Dijkstra's algorithm
- Describe Prim's algorithm for finding minimum spanning trees (MSTs)
- Prove the correctness of Prim's algorithm

NANYANG TECHNOLOGICAL UNIVERSITY

Greedy Algorithms

- In optimisation problems, the algorithm needs to make a series of choices whose overall effect is to minimise the total cost, or maximise the total benefit, of some system.
- There is a class of algorithms, called the **greedy algorithms**, in which we can find a solution by using only knowledge available at the time when the next choice (or guess) must be made.
- Each individual choice is the best within the knowledge available at the time.

Greedy Algorithms

- Each individual choice is not very expensive to compute.
- A choice cannot be undone, even if it is found to be a bad choice later.
- Greedy algorithms cannot guarantee to produce the optimal solution for a problem.

Dijkstra's Algorithm

Dijkstra's Algorithm

Shortest Path Problem:

The problem of finding the shortest path from one vertex in a graph to another vertex. "Shortest" may be the least number of edges, or the least total weight, etc.

Dijkstra's Algorithm:

This is an algorithm to find the shortest paths from a single source vertex to all other vertices in a **weighted**, **directed** graph. All weights must be **nonnegative**.

Dijkstra's Algorithm

Dijkstra's algorithm keeps two sets of vertices:

- **S** ---- the set of vertices whose shortest paths from the source node have already been determined [they form the tree]
- **V – S** ---- the remaining vertices

The other data structures needed are:

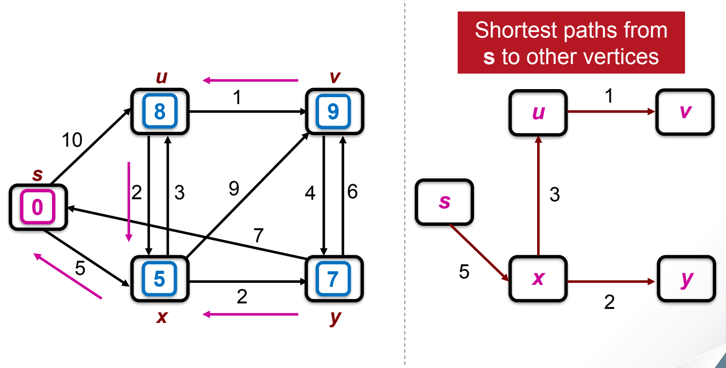
- **d** ---- array of estimates for the lengths of shortest paths from source node to all vertices
- **pi** ---- an array of predecessors for each vertex

Basic Steps

The basic steps are:

1. Initialise **d** and **pi**
2. Set **S** to empty
3. While there are still vertices in **V** – **S**
   i. Move ***u***, the vertex in **V** – **S** that has the shortest path estimate from source, to **S**
   ii. For all the vertices in **V** – **S** that are connected to ***u***, update their estimates of shortest distances to the source

A Toy Example



Pseudocode of Dijkstra's Algorithm

```
Dijkstra_ShortestPath ( Graph G, Node source ) {
  for each vertex v {
       d[v] = infinity;
       pi[v] = null pointer;
       S[v] = 0;
  }
  d[source] = 0;
  put all vertices in priority queue, Q, in d[v]'s increasing order;
  while not Empty(Q) {
    u = ExtractCheapest(Q);
    S[u] = 1; /* Add u to S */
```


Pseudocode of Dijkstra's Algorithm

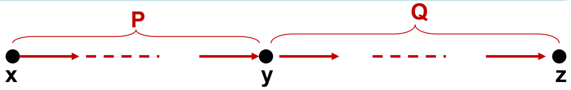
```
    for each vertex v adjacent to u
      if (S[v] ≠ 1 and d[v] > d[u] + w[u, v]) {
        remove v from Q;
        d[v] = d[u] + w[u, v];
        pi[v] = u;
        insert v into Q according to its d[v];
      }
  } // end of while loop
}
```

Worst case time complexity of Dijkstra's algorithm is $O(|V|^2)$ (analysis not required).

Proof of Correctness

Property of Shortest Path

Lemma 1: In a weighted graph G, suppose that a shortest path from x to z consists of a path P from x to y followed by a path Q from y to z. Then P is a shortest path from x to y and Q is a shortest path from y to z.



Proof (By Contradiction):
Assume that P is not the shortest path from x to y. Then there will be another path from x to y, P' which is shorter than P. As a result P' followed by Q will be a path **shorter** than P followed by Q. But it was known that P followed by Q is the **shortest** path. Contradiction. Same can be said about Q.

Greedy choice is optimal

Theorem D1: Let G = (V, E, W) be a weighted graph with nonnegative weights. Let S be a subset of V and let s be a member of S. Assume that d[y] is the shortest distance in G from s to y, for each y in S. Let z be the next vertex chosen to go into S. If edge (y, z) is chosen to minimise d[y] + W(y, z) over all edges with one vertex in S and one vertex in V – S, then the path consisting of a shortest path from s to y followed by the edge (y, z) is the shortest path from s to z.

Proof:
We will show that there is no other path from s to z that is shorter.

Greedy choice is optimal

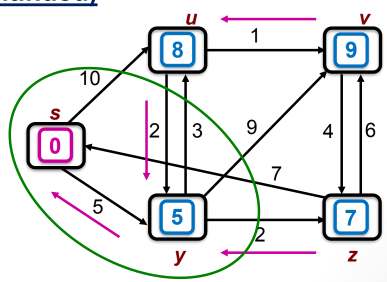
Proof of Theorem D1 (continued)

P: s →y → z (shortest path for z)

W(P) = d[y] + W(y, z)

P': s → y → u → … → z
(an alternative shortest path)

W(P') = d[y] + W(y, u)
+ distance from u to z



Because d[y] + W(y, u) ≥ d[y] + W(y, z), and distance from u to z is nonnegative, therefore W(P) ≤ W(P').

Edge (y, z) is chosen to minimise d[y] + W(y, z) over all edges with one vertex in S and one vertex in V – S

Greedy choice is optimal

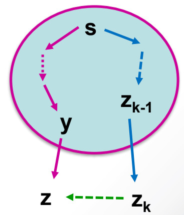
Proof of Theorem D1 (continued)

Let P be a shortest path from **s** to **y** followed by edge (y, z)
Let W(P) = the distance travelled along P
Let P' = any shortest path different from P, i.e., P' = s, z_1, …, z_k, …, z
Assume that z_k is the first vertex in P' not in set S.

W(P) = d[y] + W(y, z)

W(P') = $d[z_{k-1}] + W(z_{k-1}, z_k)$ + distance from z_k to z

Note that: $d[z_{k-1}] + W(z_{k-1}, z_k) \geq d[y] + W(y, z)$



Since distance from z_k to z is non-negative, therefore, **W(P) <= W(P').**

Theorem D2 and Proof

Theorem D2: Given a directed weighted graph G with nonnegative weights and a source vertex s, Dijkstra's algorithm computes the shortest distance from s to each vertex of G that is reachable from s.

Proof (By induction):
We will show by induction that as each vertex v is added into set S, d[v] is the shortest distance from s to v.

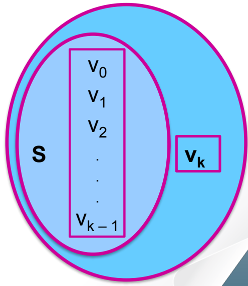
Basis:
The algorithm assigns d[s] to zero when the source vertex s is added to S. So d[s] is the shortest distance from s to s when S has the first vertex in it.

Theorem D2 and Proof (Continued)

Inductive Hypothesis:
Assume the theorem is true when S has **k** vertices. That is, assume $v_0, v_1, v_2, \ldots, v_{k-1}$ are added where $d[v_1], d[v_2]\ldots$ are the shortest distances.

When v_k is chosen by Dijkstra's algorithm, it means an edge (v_i, v_k), where $i \in \{0, 1, 2, \ldots, k-1\}$, is chosen to minimise $d[v_i] + W(v_i, v_k)$ among all edges with one vertex in S and one vertex not in S.

By Theorem D1, $d[v_k]$ is the shortest distance from source to v_k. So the theorem is true when S has **k + 1** vertices.



NANYANG TECHNOLOGICAL UNIVERSITY

Minimum Spanning Tree

Minimum Spanning Tree

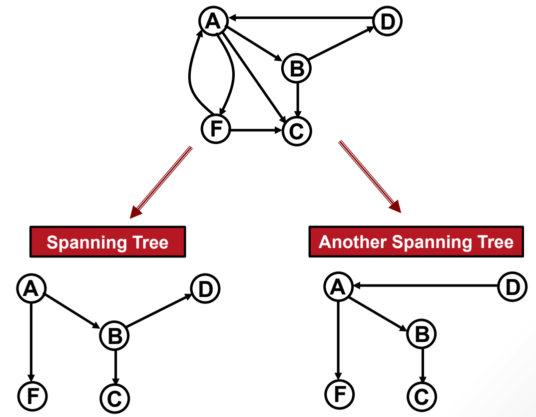
Definition of Spanning Tree

A connected, acyclic subgraph containing all the vertices of a graph.

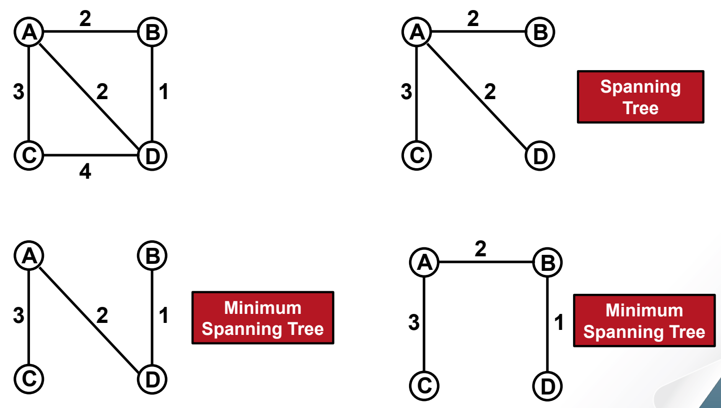
Definition of Minimum Spanning Tree

A minimum-weight spanning tree in a weighted graph.

Spanning Tree



Minimum Spanning Tree



Main Idea of Prim's Algorithm

Prim's Algorithm

- It works on undirected graph.
- It builds upon a single partial minimum spanning tree, at each step adding an edge connecting the vertex nearest to but not already in the current partial minimum spanning tree.
- At first a vertex is chosen, this vertex will be the first node in *T*.
- Set ***P*** is initialised: ***P*** = set of vertices not in tree *T* but are adjacent to some vertices in *T*.

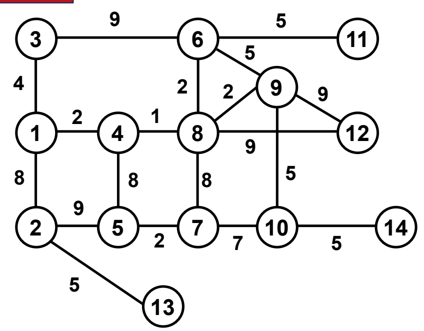
Main Idea of Prim's Algorithm

Prim's Algorithm (Cont.)

- In every iteration in the Prim's Algorithm, a new vertex ***u*** from set ***P*** will be connected to the tree *T*. The vertex ***u*** will be deleted from the set ***P***. The vertices adjacent to ***u*** and not already in ***P*** will be added to ***P***.
- When all vertices are connected into *T*, ***P*** will be empty. This means the end of the algorithm.
- The new vertex in every iteration will be chosen by using **greedy** method, i.e. among all vertices in ***P*** which are connected to some vertices already inserted in the tree *T* but themselves are not in *T*, we choose one with the minimum cost.

An Example of Prim's Algorithm

Prim's MST



Black vertices: unseen vertices | Pink vertices: tree vertices | Blue vertices: fringe vertices

3 subsets of vertices

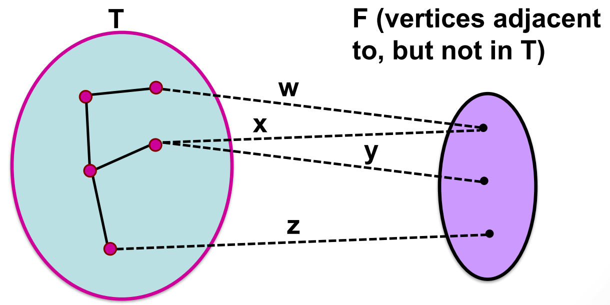
Prim's Algorithm classifies vertices into three disjoint categories:

- Tree vertices - in the tree being constructed so far
- Fringe vertices - not in the tree but adjacent to some vertices in the tree
- Unseen vertices - all others

Greedy choice of Prim's Algo

- Key step in the algorithm is the selection of a vertex from the fringe (which, of course, depends on the weights on incident edges).
- Prim's Algorithm always chooses a minimum weight edge from tree vertex to fringe vertex.

Main Idea of Prim's Algorithm



Choose min(w, x, y, z)

Pseudocode of Prim's Algo

```
primMST(G, s, n)   // outline of Prim's algorithm
{
    Initialise all vertices as unseen.
    Reclassify s as tree vertex.
    Reclassify all vertices adjacent to s as fringe.
    While (there are fringe vertices)
    {
        Select an edge of minimum weight between a tree
            vertex t and a fringe vertex v;
        Reclassify v as tree; add edge tv to the tree;
        Reclassify all unseen vertices adjacent to v as fringe.
    }
}
```


Implementing Prim's Algo

Data Structures Used:

- Array **d**: distance of a fringe vertex from the tree
- Array **pi**: vertex connecting a fringe vertex to a tree vertex
- Array **S**: whether a vertex is in the minimum spanning tree being built
- Priority queue pq: queue of fringe vertices in the order of the distances from the tree

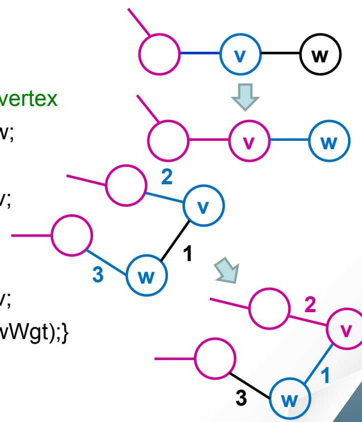
At the end of the algorithm, array pi has the minimum spanning tree.

Implementing Prim's Algo

```
primMST(G, s, n)  {
    initialise priority queue pq as empty;
    for each vertex v {
        d[v] = infinity; S[v] = 0;
        pi[v] = null pointer; }
    d[s] = 0; S[s] = 1;
    insert(pq, s, 0);
    while (pq is not empty) {
        u = getMin(pq);  deleteMin(pq);
        S[u] = 1;
        updateFringe(pq, G, u);  }
}
```


Update Fringe Set of Vertices

```
updateFringe(pq, G, v) {
  for all vertices w adjacent to v {
    if (S[w] != 1) { //if w is not a tree vertex
        newWgt = weight of edge vw;
        if (d[w] == infinity) {
            d[w] = newWgt; pi[w] = v;
            insert(pq, w, newWgt);
        } else if (newWgt < d[w]) {
            d[w] = newWgt; pi[w] = v;
            decreaseKey(pq, w, newWgt);}
    } // if w is not a tree vertex
  } // for all vertices
}
```



MST Property

Minimum Spanning Tree Property

Let T be a spanning tree of G, where $G = (V, E, W)$ is a connected, weighted graph. Suppose that for every edge (u, v) of G that is not in T, if (u, v) is added to T it creates a cycle such that (u, v) is a maximum-weight edge on that cycle. Then T has the **Minimum Spanning Tree Property** (or **MST Property**, in short).

Lemma 1 and Proof

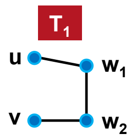
Lemma 1: In a connected weighted graph $G = (V, E, W)$, if T_1 and T_2 are two spanning trees that have the MST property, then they have the same total weight.

Proof by induction on $\boldsymbol{k}$, the **number of edges in $\boldsymbol{T_1}$ but not $\boldsymbol{T_2}$** (there are also $\boldsymbol{k}$ edges in $\boldsymbol{T_2}$ but not in $\boldsymbol{T_1}$).

Basis:
$\boldsymbol{k} = 0$; i.e. $\boldsymbol{T_1} = \boldsymbol{T_2}$. Therefore, they have the same weight.

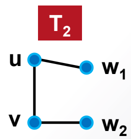
Proof of Lemma 1 (continued)

Inductive hypothesis: For $k > 0$, assume the lemma holds when there are j differing edges where $0 \le j < k$.



Let *uv* be the minimum weight edge among the differing edges (assume *uv* is in T_2 but not T_1).

Look at unique path in T_1 from u to v.



Suppose it is made up of $w_0, w_1, \ldots, w_p$ where $w_0 = u, \ldots, w_p = v$.
This path must contain some edge different from T_2's.

Let $w_i w_{i+1}$ be this differing edge.

By MST property of T_1, $w_i w_{i+1}$ cannot be > *uv*'s weight.

Proof of Lemma 1 (continued)

But since *uv* was chosen to be the minimum weight among differing edges, $w_i w_{i+1}$ cannot have weight less than *uv*.

Therefore, $W(w_i w_{i+1}) = W(uv)$.

Add *uv* to T_1 (creating a cycle). Remove $w_i w_{i+1}$ leaving tree T'_1 (which has the same weight as T_1).

But T'_1 and T_2 differ only on $k - 1$ edges.

So by inductive hypothesis, T'_1 and T_2 have the same total weight.
Therefore, T_1 and T_2 have same weight.

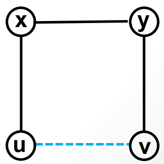
Page 392 Baase & Van Gelder

Theorem 1 and Proof

Theorem 1: In a connected weighted graph G = (V, E, W), a tree *T* is a minimum spanning tree if and only if *T* has the MST property.

Proof (**Only if**): Assume *T* is an MST for graph G.

Suppose *T* does not satisfy the MST property, i.e. there is some edge *uv* that is not in *T* such that adding *uv* creates a cycle, in which some other edge *xy* has weight W(*xy*) > W(*uv*).



Then, by removing *xy* and adding *uv* ,we create a new spanning tree whose total weight is < W(*T*); This contradicts the assumption that *T* is an MST.

Proof of Theorem 1 (continued)

Theorem 1: In a connected weighted graph G = (V, E, W), a tree *T* is a minimum spanning tree if and only if *T* has the MST property.

Proof (**Only if**): Assume *T* is an MST for graph G.

(Cont.)

(**If**) Assume *T* has MST property.

If T_{min} is an MST, then T_{min} has MST property by the first half of the proof.

By Lemma 1, W(*T*) = W(T_{min}), so *T* is also an MST.

Prim's Algorithm is Optimal

Lemma 2: Let G = (V, E, W) be a connected weighted graph; Let T_k be the tree with k vertices constructed by Prim's Algorithm, for k = 1, 2, …, n; and let G_k be the subgraph induced by the vertices of T_k. Then T_k has the MST property in G_k. (**Proof is not required**)

Theorem 2: Prim's Algorithm outputs a minimum spanning tree.

Proof:

- From Lemma 2, T_n has the MST property.
- By Theorem 1, T_n is a minimum spanning tree.

Priority Queue for MST (Optional)

- Inserted by order of priority (not chronological, as in 'normal' queues – FIFO)
- Elements to be inserted have a 'key' - contains the priority; element with highest priority will be selected first. [priority can be largest value (e.g. if we're computing max profit) or smallest value (e.g. if we're interested in min cost)]
- Think of pq as a sequence of pairs: (id_1,w_1), (id_2,w_2),…, (id_k,w_k). The order is in increasing w_i and id is a unique identifier for an element

Methods of Priority Queue (Optional)

The Priority Queue consists of:

Create: Constructor to set up PQ
isEmpty; **getMin**; **getPriority:** Access functions
insert; **deleteMin**; **decreaseKey:** Manipulation procedures

Insert(pq, id, w): Inserts (id, w) into an existing pq - position depends on w

decreaseKey(pq, id, neww): Rearranges pq based on new wt of element id

getMin(pq): Returns id_1;
getPriorty(pq): Returns weight of min element

Summary

- Greedy algorithm is a general strategy to solve optimization problems
- Dijkstra's algorithm finds single-source shortest paths in a weighted graph of nonnegative edge weights
- Prim's algorithm finds the minimum spanning trees in weighted graphs
- Both are greedy algorithms, and use priority queue