MA/CSSE 473 Day 36

Coin Changing

Kruskal's Algorithm

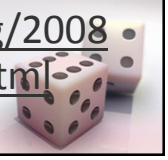
Prim's Algorithm

Coin Changing
Kruskal
Prim

MORE GREEDY ALGORITHMS

Coin-changing problem

- We have a collection of coins of various denominations (e.g., 1, 5, 10, 25)
- We want to be able to make change, a total of A cents, using the smallest number of coins.
- Assume that we have as many coins of each denomination as we need.
- http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Greedy/greedyIntro.htm
- http://www.oreillynet.com/onlamp/blog/2008/04/python_greedy_coin_changer_alg.html



Greedy coin-changing

S = ∅
Repeat until A is 0:
    Choose (add to S) the largest-denomination coin that is less than A.
    Reduce A by the denomination of that coin.
Return S

- Note that for some denomination sets, this algorithm always produces an optimal solution:
  - e.g., { 25, 10, 5, 1 }
- For others it does not:
  - e.g., { 10, 6, 1} (try A=12)
- There is a non-greedy approach that always works.
  - See exercise 8.4.9

Q1

Spanning Trees for a Graph

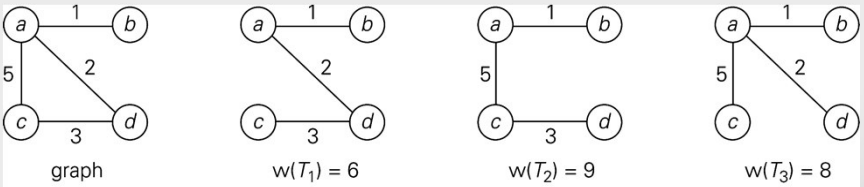
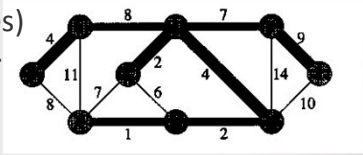


FIGURE 9.1 Graph and its spanning trees; T_1 is the minimum spanning tree

Q2

Minimal Spanning Tree (MST)

- Suppose that we have a connected network G (a graph whose edges are labeled by numbers, which we call **weights**)
- We want to find a tree T that
  - spans the graph (i.e. contains all nodes of G).
  - minimizes (among all spanning trees) the sum of the weights of its edges.
- Is this MST unique?



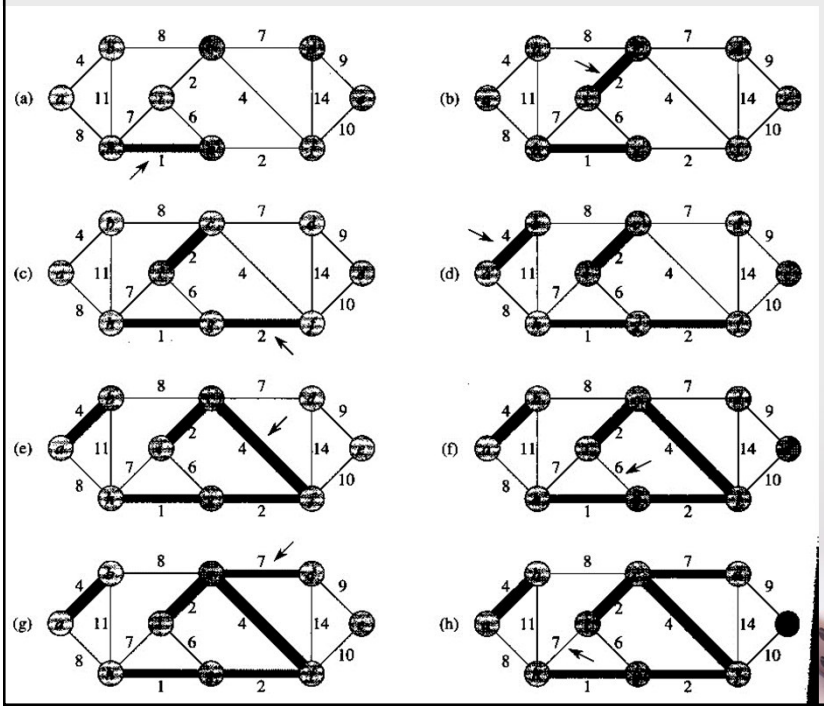
- One approach: Generate all spanning trees and determine which is minimum
- Problems:
  - The numberof trees grows exponentially with N
  - Not easy to generate
  - Finding a MST directly is simpler and faster

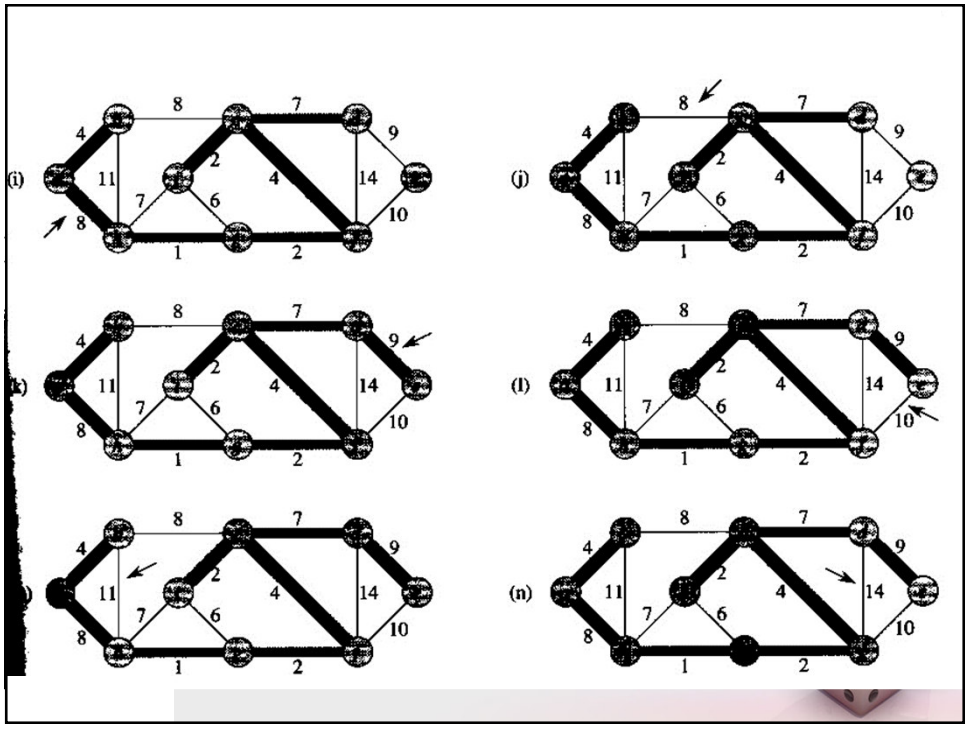
Q3-4

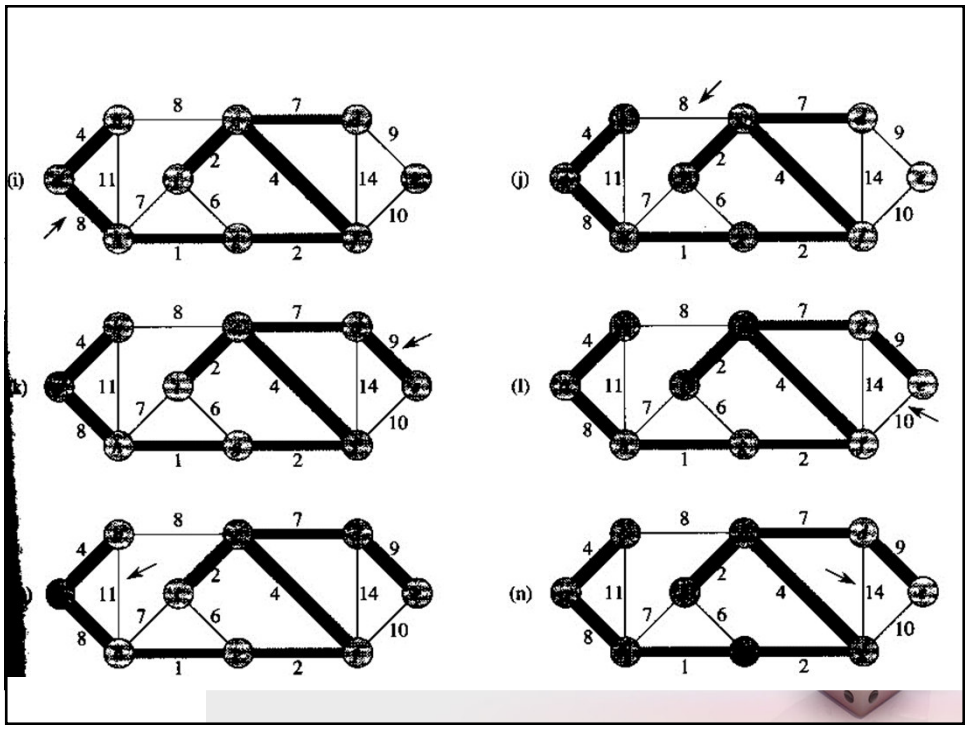
Kruskal's algorithm

- To find a MST:
- Start with a graph T containing all of G's n vertices and none of its edges.
- for i = 1 to n – 1:
  - Among all of G's edges that can be added without creating a cycle, add to T an edge that has minimal weight.

Q5



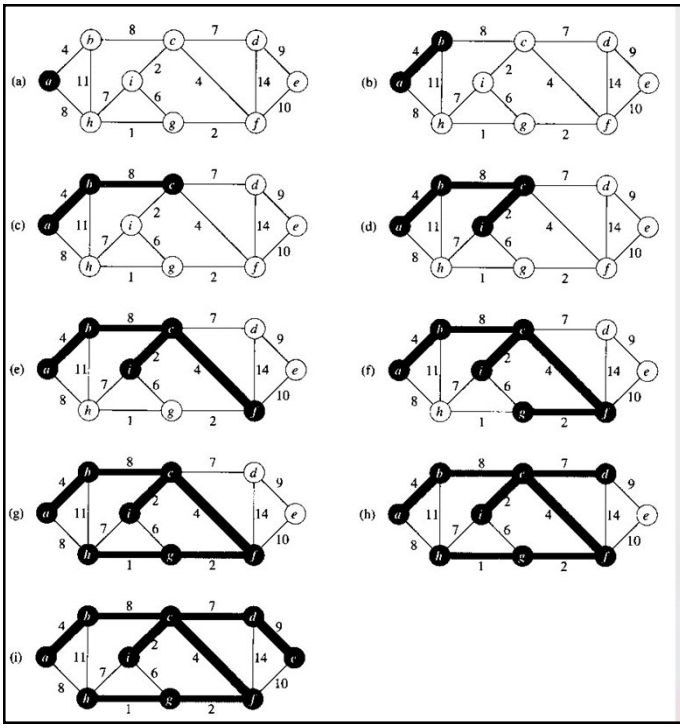

(i)
(j)
(k)
(l)
(n)


(i)
(j)
(k)
(l)
(n)

Prim's algorithm

- Start with T as a single vertex of G (which *is* a MST for a single-node graph).
- for i = 1 to n – 1:
  - Among all edges of G that connect a vertex in T to a vertex that is not yet in T, add a minimum-weight edge (and the vertex at the other end of T).

Q6



Example of Prim's algorithm

Correct?

- These algorithms seem simple enough, but do they really produce a MST?
- We begin with a lemma.

MST lemma

Let G be a weighted connected graph with a MST T; let G' be any subgraph of T, and let C be any connected component of G'.
If we add to C an edge *e=(v,w)* that has minimum-weight among all edges that have one vertex in C and the other vertex not in C, then G has an MST that contains the union of G' and *e*.
[WLOG v is the vertex of e that is in C, and w is not in C]

Q7-9

MST lemma

Let G be a weighted connected graph with a MST T; let G' be any subgraph of T, and let C be any connected component of G'.
If we add to C an edge *e=(v,w)* that has minimum-weight among all edges that have one vertex in C and the other vertex not in C, then G has an MST that contains the union of G' and *e*.
[WLOG v is the vertex of e that is in C, and w is not in C]

Proof:

✓If e is in T, we are done, so we assume that e is not in T.

✓If T does not contain edge e, adding e to T creates a cycle.

✓Removing any edge of that cycle from T∪{e} gives us another spanning tree.

✓In order to have that tree be an MST for G that contains G' and e, we must choose the "removable" edge carefully.

✓Details on next page…

Choosing the edge to remove

- ✓ Along the unique simple path in T from v to w, let w' be the first vertex that is not in C, and let v' be the vertex immediately before it.
- ✓ Then e' = (v', w') is also an edge from C to G-C.
- ✓ Note that by the minimal-weight choice of e, weight(e') ≥ weight(e) .
- ✓ Let T' be the (spanning) tree obtained from T by removing e' and adding e.
- ✓ Note that the removed edge is **not** in G',
- ✓ Because e and e' are the only edges that are different, weight(T) ≥ weight(T').
- ✓ Because T is a MST, weight(T) ≤ weight(T').
- ✓ Thus the weights are equal, and T' is an MST containing G' and e, which is what we wanted.

Recap: MST lemma

Let G be a weighted connected graph with a MST T;
let G′ be any subgraph of T, and let C be any connected component of G′.
If we add to C an edge *e=(v,w)* that has minimum-weight among all edges that have one vertex in C and the other vertex not in C,
then G has an MST that contains the union of G′ and *e*.
[WLOG v is the vertex of e that is in C, and w is not in C]

Recall Kruskal's algorithm

- To find a MST:
- Start with a connected weighted graph containing all of G's n vertices and none of its edges.
- for i = 1 to n – 1:
  - Among all of G's edges that can be added without creating a cycle, add one that has minimal weight.

Does this algorithm actually produce an MST for G?

Does Kruskal produce a MST?

- Claim: After every step of Kruskal's algorithm, we have a set of edges that is part of an MST
- Base case …
- Induction step:
  - Induction Assumption: before adding an edge we have a subgraph of an MST
  - We must show that after adding the next edge we have a subgraph of an MST
  - Suppose that the most recently added edge is e = (v, w).
  - Let C be the component (of the "before adding e" MST subgraph) that contains v
    - Note that there must be such a component and that it is unique.
  - Are all of the conditions of MST lemma met?
  - Thus the new graph is a subgraph of some MST of G

Does Prim produce an MST?

- Proof similar to Kruskal (but slightly simpler)
- It's done in the textbook