Lab 09 - Chapter 9

I. El-Shaarawy & M. Touny

December 16, 2023

Contents

Exercises

9.1.1

Hints.

- Use quotient and mod operations.
- Observe why the quotient yields the maximum possible count of some coin.

Solution.

```
# input: non negative amount n, and a decreasing array of coins D
# output: array C where C[i] is number of coins of ith denomination D[i]
def greedyCoins(integer n, D[1..m])
  # for each coin
  for i in 1..m

      # take max possible number of it
      C[i] = floor( n/D[i] )

      # remaining amount for next iteration
      n = n \mod D[i]

  # if there is still a remaining amount
  if n != 0 return "no solution"

  # otherwise given n is partitioned by coins
  return C
```

9.1.15

Homework.

9.2.3

Hints.

- Observe `Kruskal` works with global edges, unlike `Prim` which searches within local neighbour edges.
- What is error you think we will encounter upon running `Kruskal` on a a tree with more than one component?
- Why does looping on $|V| - 1$ works in `Kruskal`?
- Modify the `while` condition to accommodate any forest.

Solution.

Modify the `while` condition in `Kruskal` to be `ecounter < |E|`, So it terminates if there are no more edges.

Bonus. Modify `Prim` then use it as a subroutine to solve the general forest case.

9.2.5

Homework.

9.3.1

Hints.

- (c) Use *Transform-and-conquer* strategy.
- (c) Fixing vertices, What kind of modification is required on edges?
- (d) Use *Transform-and-conquer* strategy.
- (d) We will use *Dijkstra* as a subroutine, So the graph will be transformed to the usual form given in the book.

Solution.

(a)

A data structure which considers directed edges.

(b)

Same algorithm. You may terminate once you find the destination.

(d)

Each vertex v_i is mapped to v_i^{st} and v_i^{en}, with directed edge (v_i^{st}, v_i^{en}) whose weight is the number labeled on v_i. Any vertex in G neighbour to v_i, can travel to v_i^{st} but not v_i^{en} in G'. Only vertices v_i^{en} but not v_i^{st} can travel to other vertices. Those edges in G' are assigned zero weights.

```
# input: graph G with weighted vertices
# output: graph G with weighted edges and no weighted vertices
def vertexWeightToEdgeWeight(G)

    construct empty graph G'

    for each vertex v in G(V)
        add vertex v_st to G'
        add vertex v_en to G'
        set (v_st, v_en).weight to v.weight
        add edge (v_st, v_en) to G'

    for each edge e = {a,b} in G(E)
        set (a_en, b_st).weight = 0
        add edge (a_en, b_st) to G'
        set (b_en, a_st).weight = 0
```

```
        add edge (b_en, a_st) to G'

    return G'
```

(c)

Set the destination as source then reverse paths. If graph is directed reverse paths before running the algorithm also.

```
# input: graph G
# output: same graph but whose edges are reversed
def reverseEdges(G)
    construct empty graph G'
    clone vertices G'(V) = G(V)

    for every vertex v in G(V)
        for every edge e = (v,t) in G(E)
            add edge (t,v) to G'

    return G'

# input: undirected graph G, destination d
# output: shortest-paths of given d
def undirectedGraphSingleDistination(G, d)
    compute Dijkstra(G, d) in graph G
    return reverseEdges(G)

# input: directed graph G, destination d
# output: shortest-paths of given d
def directedGraphSingleDestination(G, d)
    G = reverseEdges(G)
    compute Dijkstra(G, d) in graph G
    return reverseEdges(G)
```

Homework.

A data-structure based implementation is left to students. In fact this is an excellent illustration of abstraction in algorithm design.

9.3.7

Homework.

9.4.5

Homework.

9.4.7

Hints.

- A basic recursive algorithm traversal works.

Solution.

```
def allHuffmanCodes(root)
    if root is NULL
        return [ ]

    # if root is a leaf
    if root.rightChild is NULL and root.leftChild is NULL
        return [ root.character ]

    # if exactly one child is NULL, Concatenating an empty list does no harm
    childCodes = allHuffmanCodes(root.leftChild) + allHuffmanCodes(root.rightChild)

    # prefix each code in child with root's character
    return [ root.character + code for code in childCodes ]
```

We leave it to students to modify the algorithm so that it generates a 2d-array of symbols-codes as a **homework.**