Greedy Algorithms

Name: Ankit Kumar Singh **GaTech ID**: asingh821

1 General Notes

Build solution piece by piece always adding the next piece that gives most obvious and immediate benefit. Sometimes the greedy algorithms are used to find the approximate solutions to the problems as well. Some of the common greedy problems are:

- Minimum spanning tree (because of cut property i.e. pick the lightest edge in the cut)
- Dijkstra's algorithm ()
- Huffman encoding (pick the 2 least frequent ones, combine and repeat)
- Horn's formula
- Fractional Knapsack (pick the one with highest value density from the remaining ones)
- Acitivity selection, Job scheduling problems ()
- Approximate solution to Set cover, TSP.

1.1 Minimum spanning tree

Krushkal - Repeatedly add the next lightest edge that doesn't produce a cycle.
Cut property: if X is set of edges part of an MST T of graph G = (V, E). Any subset of nodes S for which X doesn't cross between S and V - S. Lightest edge across the partition be e. Then X U e is still set of edges part of some other MST.

MST 1 Krushkal
Input: a connected undirected graph G = (V, E) with edge weights w_e
Output: a MST defined by edges X

```
for all u ϵ V do
    makeset(u)
end for
X = {}
sort the edges E by weights
for all edges (u,v) in increasing order of weights
if  find(u) != find(v) then
    add edge (u, v) to X
    union(u, v)
end if
return state
```

We use Disjoint set to find the representative of elements and check the cycle.

Disjoint set 2 Makeset
Input: x

```
pi(x) = x
rank(x) = 0
```

Disjoint set 3 Find

Input: x

```
1: if x != pi(x) then
2:     pi(x) = find(pi(x))
3: end if
```

Disjoint set 4 Union

Input: x, y

```
1: rx = find(x), ry = find(y)
2: if rx == ry then
3:     return
4: end if
5: if rank(rx) ¿ rank(ry) then
6:     pi(rx) = ry
7: else
8:     pi(rx) = ry
9:     if rank(rx) == rank(ry) then rank(ry) = rank(ry) + 1
10:    end if
11: end if
```

Prims algorithm is just like dijkstra's algorithm with only difference is that priority queue ordering. In prims its the value of the node is the weight of the lightest incoming edge from the set S, where as in Dijkstra's it is the length of the entire path from that starting point.

MST 5 Prims

```
1: for all u in G.V: cost(u) = 0, prev(u) = null
2: pick a initial node x, set cost(x) = 0
3: H = makequeue(G.V) with cost values as keys
4: while H is not empty do
5:     v = deletemin(H)
6:     for edge (v,z) in G[v].E do
7:         if w(v, z) ¡ cost(z) then cost(z) = w(v, z), prev(z) = v
8:         end if
9:     end for
10: end while
```

1.2 Huffman's encoding

For the data with high variation in the frequencies of the alphabets involved, most frequent letter can be encoded with lesser length values. Huffman encoding is one such variable length encoding scheme. Encoder metadata needs to be passed in the message for the reciever to be able to decode.

2 Common Problems

Some fo the common Greedy problems from different sources and their variants.

Huffman 6

Input: An array of frequency f[1...n]
Output: Encoding tree with n leaves

```
1: let H be a priority queue of integers ordered by f
2: for i = 1 to n: insert(H, i)
3: for k=n+1:2n-1 do
4:     i = deletemin(H), j = deletemin(H)
5:     create a node numbered k with children i, j
6:     f[k] = f[i] + f[j]
7:     insert(H, k)
8: end for
```

2.1 Dijkstra's algorithm

To find the single source shortest path, we can use dijkstra's algorithm (needs positive weight. atleast no negative weight cycle.) Other algorithms which solves the same problem is Bellman ford (which allows for negative weight cycles).

Single source shortest part 7 Dijkstra

```
1: for all u in G.V: cost(u) = 0, prev(u) = null
2: pick a initial node x, set cost(x) = 0
3: H = makequeue(G.V) with cost values as keys
4: while H is not empty do
5:     v = deletemin(H)
6:     for edge (v,z) in G[v].E do
7:         if w(v, z) ¡ cost(z) then cost(z) = w(v, z), prev(z) = v
8:         end if
9:     end for
10: end while
```

2.2 Activity selection Problem

Problem statement: You are given n activities with their start and finish times. Select the maximum number of activities that can be performed by a single person, assuming that a person can only work on a single activity at a time.
Approach:

- Sort the activities according to their finishing time
- Select the first activity from the sorted array and print it
- Do the following for the remaining activities in the sorted array. If the start time of this activity is greater than or equal to the finish time of the previously selected activity then select this activity and print it.

2.3 Job Sequencing Problem

Problem statement: Given an array of jobs where every job has a deadline and associated profit if the job is finished before the deadline. It is also given that every job takes a single unit of time, so the minimum possible deadline for any job is 1. Maximize the total profit if only one job can be scheduled at

a time.
Approach:

- Sort all jobs in decreasing order of profit.
- Iterate on jobs in decreasing order of profit.
- For each job, do the following :

  Find a time slot i, such that slot is empty and i ¡ deadline and i is greatest. Put the job in this slot and mark this slot filled.

  If no such i exists, then ignore the job.

3 Textbook solutions