CS 4102 Written HW2 - Greedy

Eric Xie

September 9, 2016

1)
Input: $P = \{p_1, p_2, ..., p_n\}$
Output: $G(P) = \{g_1, g_2, ..., g_i\}$ such that g_k is a unit length interval

```
double temp=0;
for(int i=0;i<P.length;i++){
        if(i==0 || P[i]>temp){
                G.append([P[i],P[i]+1])
                temp=P[i]+1
        }
}
return G
```

2)
Proof:
Let $G(P) = \{g_1, g_2, ..., g_i\}$ be the set of intervals selected by greedy algorithm
Let $O(P) = \{o_1, o_2, ..., o_i\}$ be the set of intervals selected by optimal algorithm
By the greedy algorithm, $[p_1, p_1 + 1] \in G(P)$, which contains p_1
F.S.C, assume $[x, x + 1] \in O(P)$ contains p_1 such that $x \neq p_1$
Since p_1 is the leftmost point in P, if $[x, x + 1]$ is replaced with $[p_1, p_1 + 1]$, the solution is still optimal, which is a contradiction.
The remaining points in P have optimal substructure, since there is no interval in $[p_1, p_1 + 1]$ that contains points in $(p_1 + 1, \infty)$, which means $\{x \in P | x > p_1 + 1\}$ can be solved with the same greedy algorithm.
QED G(P) is optimal

3)
Input: The number of people who need to cross the bridge, n
The speeds of each person $S = \{s_1, s_2, ..., s_n\}$ such that s_n is the time it

takes person n to cross the bridge, sorted in ascending order (Floryan said we could do this)
Output: M, the list of moves required to move all people across in minimal time

```
while(!S.empty()){
        if(S.size==1)
                cross(s1)
        if(S.size==2)
                cross(s1,s2)
        if(S.size==3)
                cross(s1,s2)
                back(s1)
                cross(s1,s3)
        if(S.size>3){
                if(2*s2+sn+s1<=sn+s(n-1)+2*s1)
                        cross(s1,s2)
                        back(s2)
                        cross(sn,sn-1)
                        back(s1)
                else
                        cross(s1,sn)
                        back(s1)
                        cross(s1,sn-1)
                        back(s1)
        }
}
```

Runtime=O(n)

4)
Let O(S) be the set of moves that transfers all people across the bridge in minimal time.
If the moves from O[x,O.length-1] such that 0¡x, is suboptimal, those moves can be substituted with the optimal moves to make O(S) optimal.
Therefore, this problem has optimal substructure
At some point, s_n and s_{n-1} must cross the bridge, which optimally takes 4 trips.
When s_n crosses the bridge, he must cross with either s_{n-1} or not s_{n-1}.
Case 1: s_n crosses with s_{n-1}
To minimize the time of the return trip by some s_i, the optimal choice is

$s_i = s_1$
Thus, s_1 must have crossed the bridge earlier with some s_j, who must bring back the flashlight.
To minimize the time of the return trip by s_j, the optimal choice is $s_j = s_2$
The total cost for these 4 trips is $2s_2 + s_n + s_1$

Case 2: s_n crosses with someone other than s_{n-1}, which we call s_i
To minimize the return trip made by s_i, the optimal choice is $s_i = s_1$
Since s_{n-1} must still cross the bridge, he must cross on the next trip, with some s_j
In order to minimize the return trip by s_j, we pick the fastest person available, which is s_1 again.
The total cost for these 4 trips is $s_n + s_{n-1} + 2s_1$

Thus, in order to minimize the time, the optimal case must be picked based on the minimum between $2s_2 + s_n + s_1$ and $s_n + s_{n-1} + 2s_1$

5)
Input: n, the number of items to be divided
$P_1 = \{I_1, I_2, ...I_n\}$, spouse 1's ranking from highest to lowest value
$P_2 = \{I_1, I_2, ...I_n\}$, spouse 2's ranking from highest to lowest value
Output: True iff it is possible to split the items into two mutually exclusive subsets such that each spouse thinks they are receiving more than half of the total value.
False otherwise

```
for(int i=0;i<p1.length;i++){
        if(i==0 || i==p1.length-1) If it's the first or last item
                spouse1.append(p1[i]) give to spouse1
        else if(p2.indexOf(p1[i]<p1[i+1]) if the first of the next 2
                                         items is worth more to spouse2
                spouse1.append(p1[i+1] give the latter to spouse1
                i++;    increment i for a total of i+=2
        else if(p2.indexOf(p1[i]>p2.indexOf[p1[i+1]) if the second of
                           the next 2 items is worth more to spouse2
                spouse1.append(p1[i]) give the former to spouse1
                i++;    increment i for a total of i+=2
}
int x=0
```

```
for(int i=0;i<p2.length;i++){
        if(spouse1.contains(p2[i]) If spouse1 has the current item
                x—
        else                If spouse1 doesn't have the current item
                x++
        if(x<0)
                return false
}
return true
```

Runtime=O(nlogn)

6)
Inputs: n, the number of servers
$C = \{c_1, c_2, ..., c_n\}$, the costs of storing the file onto server i.
Output: $L \subseteq [1, n]$, the indices of the servers to write to
Brute Force Method:
1. For each possible subset, let servers with the file be represented by 1 and servers without the file be 0. Thus the subset can be represented as a binary number.
2. Set optimal subset index so far to 0, and the optimal cost so far to the cost of subset 0.
3. For each binary number up to n^2, calculate the cost of that subset using the cost equation.
4. If the cost of the current subset is lower than the optimal cost so far, set the optimal cost to the current cost and set the optimal subset to the current index.
5. Return the optimal subset at the end of the loop
Runtime = $O(2^n)$

Better method:

```
int jumpCost=0;
for(int i=n;i>0;i--){
        if(i==n)
                writeTo(Server i)
        else
                jumpCost+=uj (as provided in problem)
        if(jumpCost>c[i])
                writeTo(Server i)
                jumpCost=0
```

}

Runtime=$O(n)$

7)
Base case: For a graph of only 1 vertex, there are no edges for Prim's to add, therefore it is minimal cost
Inductive Hypothesis: Assume Prim's algorithm is optimal for k vertices and j edges
Inductive Step: Then Prim's algorithm is optimal for k+1 vertices
F.S.C, assume that the kth vertex is known, and the optimal solution choose edge f while Prim's chooses edge e
Therefore, cost(f)¡cost(e)
However, by definition Prim's will choose e such that cost(e) is minimal, which is a contradiction
QED Prim's algorithm is optimal for k+1 if prim's is optimal for k