University of Waterloo
CS 341 — Algorithms
Spring 2014
Assignment 3

Written Portion Due: Wed, June 18, 2014, at 3:00pm.
Programming Portion Due: Wed, June 25, 2014, at 3:00pm.

Assignment Guidelines

- Use A3-coversheet.pdf for the first page of your assignment or format the first page of your assignment in *exactly* the same manner.
- Answers the questions in the same order that they are given in the assignment.
- Make it very clear to the marker where the answer to one question ends and the answer to the next one begins.
- You may lose up to 20% on a question because it is difficult to read or difficult to understand.
- You may only use the techniques discussed in class.
- Your whole assignment is considered late if either the programming question or the written portion is submitted late.

Assignment Questions

1. **[8 points]** Consider the following instance of the Stable Marriage Problem (SMP) with 3 men and 3 women, denoted M_1, M_2, M_3 and W_1, W_2, W_3, respectively. The preference lists are as follows (where ">" signifies "prefers over"):

   $M_1 : W_1 > W_2 > W_3$ $\quad$ $W_1 : M_2 > M_1 > M_3$
   $M_2 : W_1 > W_3 > W_2$ $\quad$ $W_2 : M_2 > M_3 > M_1$
   $M_3 : W_3 > W_2 > W_1$ $\quad$ $W_3 : M_1 > M_2 > M_3$

   (a) List all possible matchings (with respect to the SMP). For each matching, find all instabilities (blocking pairs) by listing the 2-tuple of the couple that have caused the instability (i.e. the couple that would prefer to be together over their current arrangements).

   $Matching_1 : (M_1, W_1), (M_2, W_2), (M_3, W_3)$. Instabilities: (M_2, W_1) and (M_2, W_3)
   $Matching_2 : (M_1, W_1), (M_2, W_3), (M_3, W_2)$. Instabilities: (M_2, W_1)
   $Matching_3 : (M_1, W_2), (M_2, W_1), (M_3, W_3)$. Instabilities: None, its stable
   $Matching_4 : (M_1, W_2), (M_2, W_3), (M_3, W_1)$. Instabilities: (M_1, W_1), (M_2, W_1) and

(M_3, W_2)
$Matching_5 : (M_1, W_3), (M_2, W_1), (M_3, W_2)$. Instabilities: None, its stable
$Matching_6 : (M_1, W_3), (M_2, W_2), (M_3, W_1)$. Instabilities: (M_1, W_1) and (M_2, W_1)

(b) Consider the Gale-Shapely algorithm with the alteration that the women propose to the men. Show each step of the execution of this algorithm on the problem instance given. Use statements of the form "x proposes to y, y accepts/rejects" to describe the execution and give the final matching.

W_1 proposes to M_2. M_2 accepts.
W_2 proposes to M_2. M_2 rejects.
W_3 proposes to M_1. M_1 accepts.
W_2 proposes to M_3. M_3 accepts.
The result is $Matching_5$.

2. **[10 points]** Yoshi is standing at position X on the left shore of a creek and wants to cross to the other side, to position Y. By using his "flutter jump", he is able to leap a distance of L feet. Unfortunately, the creek is much wider than L feet. However, fortunately, the creek has a path of n lily pads across it from X to Y where consecutive lily pads are less than L feet apart so Yoshi can cross the pond by jumping on them.

Denote the lily pads by $lp_1, \ldots, lp_n$ and the distance $d[1] = distance(X, lp_1)$, $d[i] = distance(lp_{i-1}, lp_i)$ and $d[n+1] = distance(lp_n, Y)$.

(a) Give an efficient greedy algorithm to determine the selection of lily pads that will minimize the number of jumps Yoshi must take from X to Y.

For convenience, identify X with position 0, Y with position $n+1$ and lily pad i, lp_i, with position i. We then start at position 0 and end at position $n+1$.

The greedy strategy we apply is to jump as far as possible from our current position. If our current position is i, and $i < n+1$, then jump to maximal position j such that the constraint $d[i+1] + d[i+2] + \ldots + d[j] \leq L$ is satisfied.

(b) Prove that your algorithm gives an optimal solution.

Let $[p_1, p_2, \ldots, p_k]$ be the positions chosen by the greedy strategy, $p_1 = 0$ and $p_k = n+1$. We will prove by induction on i that partial solution $[p_1, p_2, \ldots, p_i]$ can be extended to an optimal solution for $1 \leq i \leq k$. The result then follows when $i = k$.

Base case, $i = 1$, we start here so it must be in any optimal solution.
Induction hypothesis: Assume partial solution $[p_1, p_2, \ldots, p_{i-1}]$ can be extended to an optimal solution, for some $2 \leq i \leq k$. We need to prove it holds for i.
The induction hypothesis gives the existence of an optimal solution of the form $[p_1, p_2, \ldots, p_{i-1}, o_i, o_{i+1}, \ldots, o_m]$.
Because we choose p_i to be the farthest away that Yoshi can jump, we know that the distance between position p_i and o_{i+1} is not more than the distance between o_i and o_{i+1}. Thus, we can replace choice o_i with p_i and the new solution remains

optimal and has 1 more index that matches the greedy solution. $m = k$ since there are no more lily pads after the end position.

3. [**10 points**] Suppose we need to deliver n distinct items to our home from the SuperCheapStore (SCS) where each item in the store is priced at \$1. Unfortunately,

   - we can fit only one item in our truck
   - it takes one day to drive home and back.

   The more unfortunate news is that SCS charges us for the storage of undelivered items and the charge for the storage of item i grows exponentially as the original price times a constant factor $c_i > 1$ each day: if item i is picked up d days from now, the charge will be $1 \cdot (c_i)^d$. You may assume that $c_i \neq c_j$ for $i \neq j$.

   (a) Give an efficient greedy algorithm that determines the order we should pick up the items from SCS and minimize the charges.

   Greedy solution: sort items in decreasing order of c_i. Pick them up in this order. The total price will be minimized.

   (b) Give and briefly justify the runtime of your algorithm.

   Dominated by sort which is $\Theta(nlogn)$.

   (c) Prove that your algorithm gives an optimal solution.

   Let $i_1, i_2, \ldots, i_n$ represent an optimal order of picking up items. Let $v_1, v_2, \ldots, v_n$ be the sequence of price grow factors c_i corresponding to the optimal order. Suppose it is not a descending sequence (as chosen by the greedy method). We can build up another solution with smaller total price. It is built as follows: there should be at least one index t for which $v_t < v_{t+1}$. Swap them, keeping the rest of the sequence the same. The old order had price $v_t^t + v_{t+1}^{t+1}$ corresponding to those elements; new order has price $v_{t+1}^t + v_t^{t+1}$. Observe that since $v_{t+1} > v_t$:
   $v_t^t + v_{t+1}^{t+1} - v_t^{t+1} - v_{t+1}^t$
   $= v_{t+1}^t(v_{t+1} - 1) - v_t^t(v_t - 1)$
   $> v_{t+1}^t(v_t - 1) - v_t^t(v_t - 1)$
   $= (v_{t+1}^t - v_t^t)(v_t - 1)$
   > 0
   This means swapping the order of the two items improves the optimal solution which contradicts its optimality.

4. [**12 points**] An orphanage has recently received a donation of a pair of shoes for each of the n children at the orphange. However, the shoes are of various sizes and so are the children's feet - some children will find a pair that fits well, others will find the shoes to be too big and others will find them to be too small. More formally, a child i has foot size f_i and pair of shoes j has shoe size s_j. The goal is to distribute the shoes to minimize the average difference of each child's foot size with the shoe size they are given.

(a) Give an efficient greedy algorithm that assigns shoes to children and minimizes the absolute difference between a child's foot size and the size of the shoes they are given.

The greedy alorithm is to assign the largest pair of shoes to the child with the largest feet and the second largest pair of shoes to the child with the second largest feet and so on. Sort the children by feet size and the shoes by shoe size then match them up. This gives a minimum average.

(b) Give an exchange proof to show that your algorithm produces an optimal solution.

Assume the children are sorted by feet size and label them in this order from 1 to n. Let g_i be the pair of shoes assigned to child i by the greedy algorithm and consider an alternative optimal assignment that assigns pair of shoes s_i to child i, for $1 \leq i \leq n$.

Find the first place, i, where the optimal solution disagrees with the greedy solution. The greedy solution assigns shoes g_i to child i while the optimal solution assigns shoes s_i. Since both the greedy solution and the optimal solution agree on assignments 1 through $i-1$, then pair of shoes g_i is assigned to some child later in the optimal solution, say child j so $s_j = g_i$. Also, s_i is assigned later in the greedy solution so when ordering the shoes by shoe size for the greedy solution, s_j came before s_i (or they could have the same size).

Let $\delta_f = |f_i - f_j|$, the absolute difference in feet size between children i and j and $\delta_s = |s_j - s_i|$, the absolute difference in shoe size between s_i and s_j.

Now consider exchanging s_j and s_i in the optimal solution. Since both foot size and shoe sizes use the same scale, child i is ordered before child j and s_j is ordered before s_i so we consider the following orderings.

Case 1: s_j, s_i, f_i, f_j
The original assignment assigned s_i to f_i and s_j to f_j. Consider the distance between pairings segmented at each point. The distance between s_i and f_i is the length of segment $s_i f_i$ and the distance between s_j and f_j is the sum of segments δ_s, $s_i f_i$, and δ_f. Exchanging the assignments, the distance between s_j and f_i is the sum of segments δ_s and $s_i f_i$ and the distance between s_i and f_j is the sum of segments $s_i f_i$ and δ_f. Thus, in this case the two assignments give the same combined sum of distances and with the exchange is one step closer to the greedy solution.

Case 2: s_j, f_i, s_i, f_j
A similar comparison between the given optimal solution and the exchanged optimal solution shows the exchanged optimal solution saves a distance of twice the length of segment $f_i s_j$ so the solution one step closer to the greedy solution is no worse than the original optimal solution.

Case 3: s_j, f_i, f_j, s_j
The exchanged optimal solution saves a distance of twice δ_f so the solution one step closer to the greedy solution is no worse than the original optimal solution.

Case 4: f_i, s_j, s_i, f_j
The exchanged optimal solution saves a distance of twice δ_s so the solution one step closer to the greedy solution is no worse than the orignal optimal solution.

Case 5: f_i, s_j, f_j, s_i
The exchanged optimal solution saves a distance of twice the length of segment $s_j f_i$ so the solution one step closer to the greedy solution is no worse than the original optimal solution.

Case 6: f_i, f_j, s_j, s_i
Similar to case 1 where there is no change in the overall sum of combined distances.

Thus, in all cases, moving one step closer to the greedy solution from the optimal solution makes each step no worse than the optimal solution. Hence the greedy solution is also an optimal solution.

5. [**15 points**] Programming Question

Suppose we are given a set M of n musicians, one of which is Elvis, denoted m_e. A number of the musicians are related in that they have been on stage together at various performances. In this problem, we want to calculate the *Elvis number* of each musician. The *Elvis number* of a musician m in M is defined to be the smallest d such that there exists a sequence of musicians $m_e, m_1, m_2, \ldots m_d$ where $m_d = m$ and each $m_i \in M$ and each pair of consecutive musicians has been on stage together; i.e. Elvis has *Elvis number* 0, someone who has performed on stage with Elvis has *Elvis number* 1, etc.

Your program will input the size of the set of musicians, n (containing musicians 1, 2, 3, 4, 5), followed by lines containing the relationships of each musician. Line i lists the musicians that musician i has been on stage with. For example:

```
5
2 3
1 5 4
1
5 2
2 4
```

Musician 1 has been on stage with musicians 2 and 3, musician 2 has been on stage with musicians 1, 5 and 4.

You may assume that Elvis is musician 1. The output to your program should be the *Elvis number* of each musician, in order, one per line. The output for the above input is:

```
0
1
```

```
1
2
2
```

(a) Write an efficient program in C++ that solves this program by searching a graph of performance relationships (two musicians that have been on stage together are related). Store the graph using an adjacency list representation.

(b) In comments at the beginning of your code, compare the complexity of your program that uses the adjacency list representation with the alternative, an adjacency matrix implemenatation.