CS180 Midterm

Thomas Joseph Garcia

TOTAL POINTS

61 / 100

QUESTION 1

1 BFS Shortest Path (5 / 25)

- **- 0** Correct
- **- 5** (a) right about the idea of splitting, but not correct for the rest
- **- 20** (a) Blank
- **- 15 (a) Showed minor steps, but not a complete and correct solution**
- **- 10** (a) Showed something, but not a correct algorithm
- **- 5** (b) blank
- **- 5** (b) Didn't prove anything
- **- 3** (b) Showed something, but not correct or complete
- **- 5** (b) Not to the point
- **- 5 (b) Not a correct proof**

QUESTION 2

2 Max and Min (25 / 25)

- **- 0 Correct**
- **- 15** Showed something, but not correct and complete
- **- 13** The algorithm works, but doesn't comply with the time complexity requirement of this problem
- **- 25** No answer
- **- 13** Not a deterministic solution
- **- 10** Mostly right, but missing some steps

QUESTION 3

3 String Subsequence (18 / 25)

- **- 7** relevant for part a
- **- 7 what if the case both A&B are equal to C**
- **- 5** no answer for part(c)
- **- 4** not correct for part(c)
- **- 0 correct for part(a)**
- **- 10** no answer for part(b)
- **- 0** correct for part(b)
- **- 0 correct for part(c)**
- **- 9** irrelevant for a
- **- 7** the complexity of part(a) is not correct
- **- 9** irrelevant for b
- **- 7** the complexity of part(b) is not correct
- **- 10** no answer for part a
- **- 4** almost correct for a
- **- 1** almost correct for b
- **- 5** correct for some part of b

QUESTION 4

4 Approximate TSP (13 / 25)

- **- 0** Correct
- **- 5** (a): Dijstra algorithm does not output a tour
- **- 3** (a): not a greedy algorithm for TSTP
- **- 1** (a): the algorithm did not explain how to get a tour back to the source
- **- 5** (c).i: no answer
- **- 5** (c).i: TSP is larger than MST
- **- 5** (c).i: incorrect answer and proof
- **- 2 (c).i: Not a correct proof for MST<TSP**
- **- 5 (c).ii: No answer**
- **- 3** (c).ii: Didn't show how to use MST to get a TSTP tour.
- **- 1** (c).ii: didn't show why the it is no more than double of optimal length
- **- 5 (c).iii: no answer**
- **- 5** (c).iii: use previous algorithm and remove the reused edges by go by straight lines
- **- 1** (c).iii: didn't show why the it is no more than double of optimal length
- **- 15** (c): no answer
- **- 20** (a)(c): no answer
- **- 0** Click here to replace this description.
- **- 2** (c).I: What's the answer, Yes or No?

- **5** (a): No answer
- **0** Click here to replace this description.
- **3** (a): The algorithm does not return a tour

CS 180: Introduction to Algorithms and Complexity
Midterm Exam (May 9, 2016)

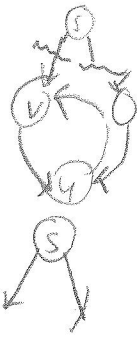
Name	Thomas Garcia
UID	904606029
Section	1D

1	2	3	4	Total

★ **Please write your answers ONLY on the front of the exam pages. Anything on the back of page will NOT be graded.**

★ **Print your name, UID and section number in the boxes above, and print your name at the top of every page.**

- The exam is closed book. You can bring one sheet letter size cheat sheet.
- There are 4 problems. Each problem is worth 25 points.
- Do not write code using C or some programming language. Use English or clear and simple pseudo-code. Explain the idea of your algorithm and show why it works.
- Your answer are supposed to be in a simple and understandable manner. Sloppy answers are expected to receiver fewer points.
- Brute force (exponential time) algorithm does not get any point.
- Don't spend too much time on any single problem. If you get stuck, move on to something else and come back later.



1. Given an directed weighted graph $G = \{V, E\}$ and a source vertex s. The weights of edges are restricted to be either 1 or 2.

   (a) Describe an $O(|E|)$ algorithm to compute the shortest path from the source vertex s to all other vertices. (Hint: Dijkstra's algorithm is not $O(|E|)$ in general.) **[20 pts]**

   (b) Prove the correctness of your algorithm. **[5 pts]**

a) Perform a BFS and number as you visit them. Set initial distance to ∞, except set length(s) ← 0.

```
for i in range 1 to n:
    for incoming edge (u, v_i, w):
        if length(u) + w < length(v_i):
            length(v_i) ← length(u) + w
            parent(v_i) ← u
```

b) Take an arbitrary edge (u, v, w). If w = 1, then this is the shortest path between u & v, because no edges are less than 1. If w = 2, it is still the shortest path, because any other path has at least two edges, & therefore a minimum total weight of 2.

a) Do Dijkstra's except use an ordered linked list instead of queue. Because lengths are 1 or 2, new elements are 1 or 2 elements below the top, & therefore insertion is constant, & so is removing the top. Therefore, the algorithm become O(|V|+|E|) |V| ≤ |E| (assuming the graph is connected) so it is O(|E|).

b) We know Dijkstra's works if the queue is sorted. This replaces the queue with a new data structure, which is sorted as described in part a).

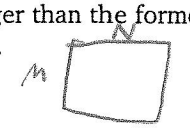
2. Given n keys, we can calculate the maximum or the minimum by comparing the keys. A simple minded algorithm to calculate both the max and the min will take "about" $2n$ comparisons. The "about" means it is $2n$ add or minus some constants, i.e., you can calculate them using $2n-2$ comparisons. Give an algorithm to calculate both the max and min using about $(3/2)n$ comparisons. **[25 pts]**

```
max ← -∞
min ← ∞
for i in range 0 to (n/2 - 1):
    if k_2i < k_2i+1:
        min_i ← k_2i
        max_i ← k_2i+1
    else:
        min_i ← k_2i+1
        max_i ← k_2i
    if max_i > max:
        max ← max_i
    if min_i < min:
        min ← min_i

return (max, min)
```

If there is an odd number of keys, just check the last key against max & min by itself.

3. A common subsequence of A and B is a subsequence of both A and B. For example, the string AB, is a common subsequence of ACBCACBC, and BCACBCAC. So is CCCC. But the latter is longer than the former. A LCS (Longest Common Subsequence) is a common subsequence which is the longest.



   (a) Describe an LCS algorithm. Your algorithm should run in $O(mn)$. **[10 pts]**

   (b) Given three strings A, B and C, of lengths m, n, and $m+n$, respectively. Describe an $O(mn)$ time algorithm to determine whether C is formed by the interleaving of A and B. For example, you are given $A =$ AABCC, $B =$ DBBCA, you should return "TRUE" when $C =$ AADBBCBCAC and return "FALSE" when $C =$ AADBBBACCC. **[10 pts]**

   (c) In the homework, we have seen the problem of Longest Palindrome Subsequence (LPS): given a string $A[1..n]$ calculate the *length* of the longest subsequence that is also a palindrome. Design an algorithm for LPS by reducing it to the LCS problem. That is, assume you have an LCS "black box", and you can invoke it *constant* number of times by giving it two input strings of your choice. Do not give the "direct" solution that we have done in the homework! **[5 pts]**

a)

$$opt[i,j] = \max \begin{cases} opt[i-1,j] \\ opt[i,j-1] \\ 1 + opt[i-1,j-1] \text{ if } A_i = B_i \end{cases}$$

$$Opt[0,*] = opt[*,0] = 0$$

This recurrence shows us how to find the length of the longest sequence. There are mn locations in the opt array, & filling each takes constant time, so it is $O(mn)$. The result is in $Opt[m,n]$. Finding the actual string can be done by starting at $Opt[m,n]$ & backtracking through how the decisions were made, which is $O(m+n)$.

b)

```
is_subsequence(sub, S):
    i ← 1
    for j in range 1 to len(S):
        if S_j = sub_i & S_j is unmarked:
            i += 1
            mark S_j
    if i > len(sub):
        return true
    return false.
```

On the left is an algorithm for checking whether a string is a subsequence of another. It is $O(M)$, where M is the length of S. Run it once on A & C, then once on B & C, without clearing the markings between. If both calls returned true, and all

(extra space at next page)

(extra space for problem 3)

letters are marked, then C is an interleaving of A & C. This algorithm is in total $O(m+n)$, & $O(m+n) \leq O(mn)$, so it is also $O(mn)$.

c) Let B be the reverse of A. Run LCS on A & B, and find the length of this LCS. This is the length of the longest palindrome, because a palindrome is a subsequence of both a string and its reverse.

4. Given a weighted undirected complete graph $G = \{V, E\}$, the Traveling Salesman Tour Problem (TSTP) asks for the shortest tour that starts at node s and visit all other nodes ending in s. A tour is just a sequence of consecutive edges (path) that allows repetition of nodes and edges. The Traveling Salesman Problem (TSP) restricts the tour to be a simple cycle (i.e., a cycle visits every vertex exactly once).

   (a) Propose a greedy algorithm for TSTP (Unfortunately, your algorithm might *not* output the optimal tour). **[5 pts]**

   (b) Show by an example that your greedy algorithm returns a tour that is more than *double* the length of the optimal tour. (Hint: consider all the nodes are located on a line. Place nodes as to force the Greedy TSTP to go through s between two consecutive new nodes visits. Then do not calculate. Look globally, as supposedly your learn in this class, and make a simple argument why the tour length is worse than a factor of 2 from optimal.) **[5 pts]**

   (c) Consider the graph is a *geometric* graph, where the vertices are in the plane, edges are straight line segments between any two vertices, and lengths are defined by the Euclidean distance. Hence, edge lengths satisfies the *triangle inequality:* $d(u,w) \leq d(u,v) + d(v,w)$ for any vertices u,v and w.

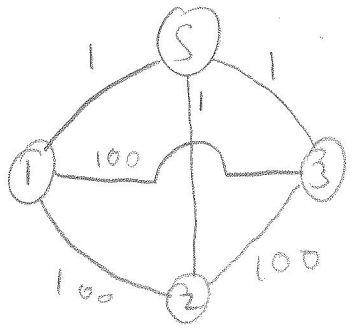
      i. Is the length of the TSP tour larger than the length of Minimum Spanning Tree (MST)? Why? **[5 pts]**

      ii. Design a polynomial time algorithm for TSTP problem to get a *tour* whose length is guaranteed to be at most *double* the length of the optimal TSTP tour. (Hint: use the MST) **[5 pts]**

      iii. Design a polynomial time algorithm for TSP to get a *tour* but the tour should be a simple cycle rather than a general tour. Your algorithm should have the same property (i.e., the tour is at most *double* the length of the optimal TSP tour). **[5 pts]**

a) Mark the source node as visited. Find the shortest edge going from the source node to an unvisited node. Add that edge, and go to the node at the other end. Repeat this process for the new node (mark it, find shortest edge to unvisited edge, etc.), & keep repeating until all nodes are visited. Then add the edge connecting the current node to the source.

b)



The TSTP is 6, as you just return to s after visiting each node. My algorithm gives a path of 202, because it never goes back to a visited node.

(extra space at next page)

(extra space for problem 4)

c) i) Yes, because the TSP tour is a cycle.