CS180 final

Junhong Wang

TOTAL POINTS

54 / 100

QUESTION 1

Problem 1 20 pts

1.1 1.a 10 / 10

- **- 5 pts** Incorrect statement of augmenting path
- **- 5 pts** Inorrect statement of cut and its capacity
- ✓ **- 0 pts** **Correct**
- **- 2 pts** Incomplete proof of statement of augmenting path
- **- 2 pts** Incomplete proof of statement of cut and its capacity
- **- 1 pts** minor error in proof of augmenting path
- **- 1 pts** minor error in proof of cut and its capacity

1.2 1.b 10 / 10

- **- 4.5 pts** Incorrect residual graph of first step
- **- 4.5 pts** Incorrect second step
- **- 1 pts** Incorrect max flow
- ✓ **- 0 pts** **Correct**
- **- 5 pts** Partial incorrectness
- **- 3 pts** missing capacity

QUESTION 2

2 Problem 2 0 / 15

- **+ 3 pts** Proper data structure
- **+ 2 pts** Proper Initialization
- **+ 7 pts** complete/correct core routine (maybe minor issues)
- **+ 5 pts** Partially complete/correct core routine
- **+ 3 pts** Partially complete/correct core routine
- **+ 3 pts** Time complexity
- ✓ **+ 0 pts** **Incorrect approach**
- **+ 0 pts** no answer

QUESTION 3

3 Problem 3 10 / 20

a.

- ✓ **+ 7 pts** **Correct algorithm**
- ✓ **+ 2 pts** **Correct algorithm proof**
- ✓ **+ 1 pts** **Correct time complexity analysis**
- **+ 4 pts** Partial solution
- **+ 0 pts** Wrong solution

b.

- **+ 10 pts** Full proof
- **+ 5 pts** Partially correct proof
- **+ 3 pts** Proof correct for sorting lower bound, but no correct link to decision trees
- ✓ **+ 0 pts** **Wrong proof**

QUESTION 4

4 Problem 4 11 / 15

- **- 0 pts** Correct
- **- 6 pts** Missing/wrong algorithm
- **- 4 pts** Major errors in algorithm
- **- 2 pts** Minor errors/unclear statements in algorithm description
- **- 1 pts** Minor error/unclear statement in algorithm description
- **- 6 pts** Missing/wrong proof of correctness for algorithm
- ✓ **- 4 pts** **Faulty reasoning/unclear statements in proof of correctness**
- **- 3 pts** Non-rigorous reasoning in proof of correctness
- **- 2 pts** Incorrect/unclear statements in proof of correctness
- **- 1 pts** Unclear statement in proof of correctness
- **- 3 pts** Missing/Incorrect time complexity analysis
- **- 1 pts** Minor problem in time complexity analysis
- **- 2 pts** Missing/Incorrect explanation for time

complexity.

QUESTION 5

5 Problem 5 5 / 15

- **0 pts** Correct

✓ **- 15 pts** **completely incorrect / see specific comments**

+ 5 Point adjustment

5 points for the attempt

QUESTION 6

6 Problem 6 8 / 15

✓ **+ 2 pts** **has a base case**

+ 8 pts The recursion is correct

✓ **+ 5 pts** **the recursion has some minor error**

+ 3 pts the recursion has a major error

+ 2 pts the subproblems are evaluated correctly

+ 3 pts b. has correct time complexity analysis

✓ **+ 1 pts** **b. has time complexity analysis but the algorithm is incorrect / time complexity analysis is incorrect**

+ 0 pts incorrect solution (not using dynamic programming)/ no solution

Name(last, first): Wang Junhong

5049411113

ID (rightmost 4 digits): 1113

U C L A Computer Science Department

CS 180 Algorithms & Complexity

Final Exam **Total Time: 3 hours** **December 10, ~~2018~~ 2019**

*** Write all algorithms in bullet form (as done in the past) ***

You need to prove EVERY answer that you provide.

There are a total of 8 pages including this page.

Name(last, first): Wang Junhong

1. (20 points: each part has 10 points)

a. Consider a S-T network N. **Prove** that if **f** is a maxflow in the network N then there is a cut C with its capacity equal to **f**.

Consider these three statements:

(1) f is a max flow in N

(2) The residual network N_f with respect to flow f doesn't have augmenting path

(3) There exists a cut C with its capacity equal to f

Suppose (1) holds

we want to show (3) holds

we will prove this by showing (1) $\Rightarrow$ (2) $\Rightarrow$ (3)

(1) $\Rightarrow$ (2).

Suppose f is a max flow in N

Suppose on the contrary, the residual network N_f has an augmenting path.

Then we can add at least 1 flow to that path and obtain a flow of at least $f+1$.

This contradicts the assumption that f is a max flow in N □

(2) $\Rightarrow$ (3)

Suppose the residual network N_f has no augmenting path

Remove saturated edges from the network N

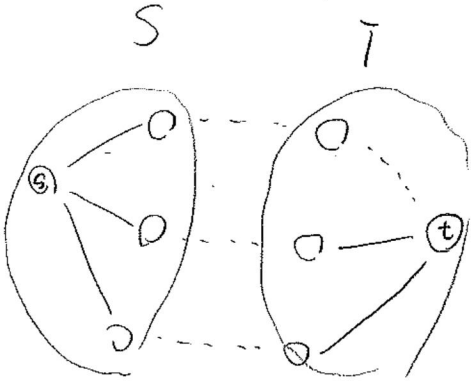
Let S be a set of vertices reachable from the source node s. Since the residual network N_f has no augmenting path, the sink node $t \notin S$.

Let T be the rest of vertices (i.e. vertices not in S)

Clearly the sink node $t \in T$.

So, the flow f is going out of S, and coming into T.

Since all the edges connecting from a node in S to a node in T were saturated, these edges form a cut C with capacity equal to f.

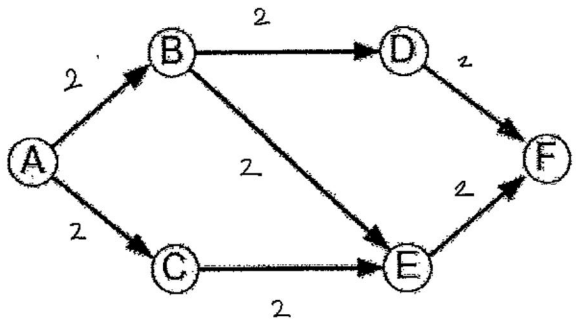


Name(last, first): Wang Junhong

b. Consider the following network with source **A** and sink **F**. If the Ford-Fulkerson max flow algorithm initially finds the path **A,B,E,F** in the network below and sends **2 units** of flow on it, **show the residual network** (also known as the augmented network) and all subsequent steps of Ford-Fulkerson algorithm on this network (all capacities are equal to **2**).

step 1 :

(start)

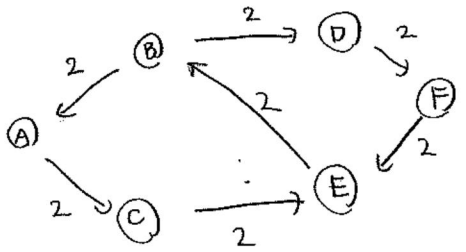


Finds the path

A, B, E, F

step 2 :

current flow = 2

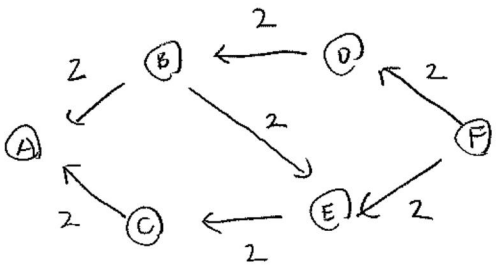


Finds the path

A, C, E, B, D, F

step 3 :

current flow = 4.



So max flow = 4

① → ② → ③ ④ → ⑤ ⑥ → ⑦ → ⑧ → ⑨

O(V+E)

Name(last, first): Wang Junhong

2. (15 points) a. Given a n x n matrix where all numbers are distinct, design an **efficient algorithm** that finds the maximum length path (starting from any cell) such that all cells along the path are in increasing order with a difference of 1.

b. Analyze the time complexity of your algorithm

We can move in 4 directions from a given cell (i, j), i.e., we can move to (i+1, j) or (i, j+1) or (i-1, j) or (i, j-1) with the condition that the adjacent cells have a difference of 1.

```
Input: mat[][] = {{1, 2, 9}
                  {5, 3, 8}
                  {4, 6, 7}}
Output: 4
The longest path is 6-7-8-9.
```

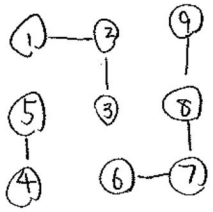
(a)

[Algorithm]

1. Construct a graph G as the follows:

For each entry (i,j) of the matrix, create a vertex Vij

Create an edge between Vij and Vke if mat[i][j] and mat[k][l] is adjacent to each other and different by 1.



2. Topological sort the graph G

① → ② → ③ ④ → ⑤ ⑥ → ⑦ → ⑧ → ⑨

3. Find the longest path as the follows:

Mark the distance to every node as $-\infty$

Let l := 0

For every vertex V_i visited in topological order:

If distance to V_i is $-\infty$:

Mark distance to V_i as 0

For every edge $e_{ij} = (V_i, V_j)$ of V_i:

If (distance to V_j) < (distance to V_i + 1):

Mark distance to V_j as $V_i + 1$

Update l to be the max distance assigned to a vertex

Return l+1

①	②	③	④	⑤	⑥	⑦	⑧	⑨
dist 0	1	2	0	1	0	1	2	3

check the back of this page 4

[proof of correctness]

We want to show the answer we get from solving the maximum length of path is equivalent to solving the original problem.

Since we construct the graph G such that there's an edge between V_{ij} and V_{kl} iff mat[i][j] and mat[k][l] are adjacent cells and different by 1, the path from V_{ij} to V_{kl} in G implies there is a path from mat[i][j] to mat[k][l] with increasing order of 1.

(b) Step 1 of the algorithm takes $O(n^2)$ because it needs to check every entry of the matrix

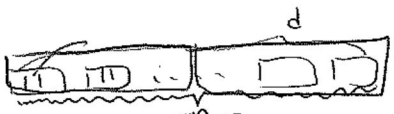
Step 2 of the algorithm takes $O(n^2 \log n^2)$ because it uses a heap to keep track of minimum indegrees.

Step 3 of the algorithm takes $O(n^4)$ because it traverse every vertex ($O(n^2)$) and for each vertex it checks its edges, which takes $O(n^4)$ throughout the algorithm.

(There are n^2 nodes, so, at most there are $\frac{1}{2} n^2(n^2-1)$ edges)

Therefore, time complexity of this algorithm is $O(n^4)$

Name(last, first): Wang Junhang



3. (20 points: Each part has 10 points)

a. Consider **d** sorted array of integers each containing n1 n2 .. nd numbers. The numbers ni's can be very different. The total number of all elements is **n** (sum of all ni's). Design an **O(n log d)** algorithm that merges all arrays into one sorted list. You may wish to use a data structure that we have discussed in class.

b. Prove a lower bound on sorting **n** numbers in the decision tree model (using comparison exchange).

a.

[Algorithm]

Assume arrays are 1-indexed.

Define a recursive function as follows:

```
sort (arrays, d):
    If d=1: Return array[1]
    return merge(sort(arrays[1], ..., arrays[d/2], d/2),
                 sort(arrays[d/2+1], ..., arrays[d]))
```

Note merge(arr1, arr2) is the merge function from merge sort we discussed in class.

[Proof of Correctness]

This algorithm uses divide and conquer to break the problem into smaller subproblems we can prove it by induction:

(Base Case) we just return the only array we have, which is already sorted by assumption.

(Inductive case) We have left subarray and right subarray that are both sorted by assumption. Merge() function will make sure the merged array is sorted □

[Time Complexity]

Let $T(d)$ be time complexity of this algorithm.

Then $T(d) = \frac{1}{2}T(\frac{d}{2}) + \frac{1}{2}T(\frac{d}{2}) + O(n)$ → at most n elements to merge;

$= T(\frac{d}{2}) + O(n)$

$= T(\frac{d}{4}) + O(n) + O(n)$

$\vdots$

$= \underbrace{O(n) + O(n) + \cdots + O(n)}_{\text{about } \log d \text{ of them}}$

$= O(n \log d)$

3 4 4

Name(last, first): Wang Junhang

4. (15 points)

Consider an array a1,...,an of n integers, that is hidden from us. We have access to this array through an procedure knapsack(.,.). For a set $S \subseteq \{1,...,n\}$ and an integer k, knapsack(S,k) will output "yes" if there is a subset $T \subseteq S$ such that the numbers indexed in T add up to k, and it will output "no" otherwise.

Design an algorithm that calls knapsack only O(n) times and outputs a set $S \subseteq \{1,...,n\}$ such that the numbers indexed in S add up to k, if such a set exists. You can use ONLY the knapsack function (e.g., you cannot sort the numbers or do any other operations on them).

For example, suppose a1 = 2, a2 = 4, a3 = 3, a4 = 1, and k = 7. Then, knapsack({1,2,3,4},7) returns "yes" and knapsack({1,3,4},7) returns "no". In this case your algorithm can output either of the sets {1,2,4} or {2,3}. Note that for example {1,2,4} are indices of the numbers, that is, a1, a2, and a4.

[Algorithm]

Let $S = \{1, 2, \dots, n\}$

For each element i in S:

Call knapsack($S \setminus \{i\}$, k)

If "Yes":

Delete i from S

Return S

[Proof of correctness]

Suppose $S' = \{i_1, i_2, \dots i_l\}$ and S' adds up to k and l is minimum.

If any element $i_m \in S'$ is not included for the call of knapsack, then it will return "No" and we know i_m is in the solution set.

If it returns "Yes", then it means the element not included is not in the solution set, and thus it's safe to delete that element.

[Time Complexity Analysis]

This algorithm takes O(n) because the loop takes O(n) and during each iteration, we are only calling knapsack once.

Name(last, first): Wang Junhao

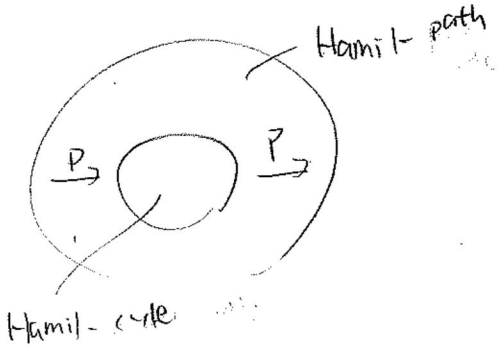
5. (15points)

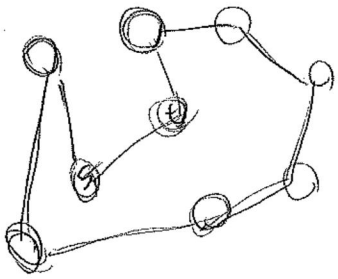
A **Hamiltonian cycle** in a graph with **n** vertices is a cycle of length n, i.e., it is a cycle that visits all vertices exactly once and returns back to the starting point. A **Hamiltonian path** in a graph with n vertices is a path of length n−1, i.e., it is a path that visits all vertices of the graph exactly once.

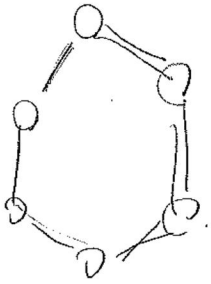
Hamil-cycle problem is defined as follows: Given a graph G = (V,E), does it have a Hamiltonian cycle? Hamil-path problem is defined as follows: Given a graph G = (V,E), does it have a Hamiltonian path?

Prove that Hamil-path is polynomial-time transformable to Hamil-cyle.

That is Hamil-path ≤P Hamil-cycle.







We want to show Hamil-path ≤p Hamil-cycle

Let G be an arbitrary instance of Hamil-path

Suppose G has n vertices.

We can construct n(n-1) instances of G'.

by adding an edge between two vertices in G

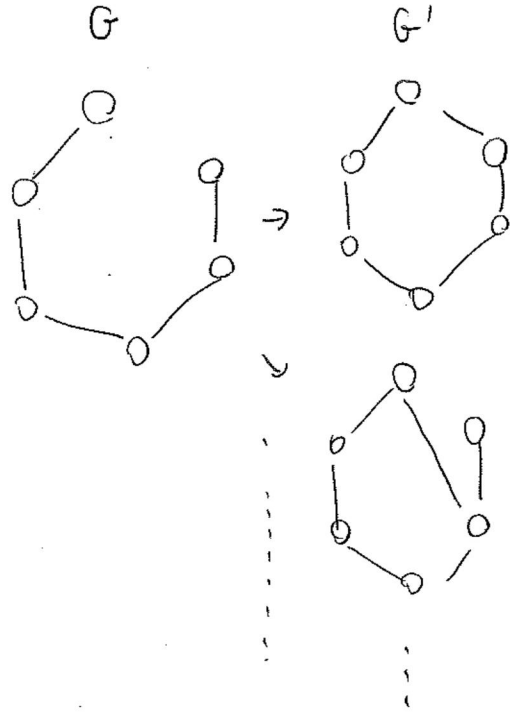
This takes polynomial time because there

are at most n(n-1) instances of G'.

If any one of G' has Hamil-cycle,

then its original graph G must at least

have a Hamil-path



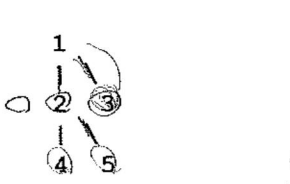
Name(last, first): Wang Junhong

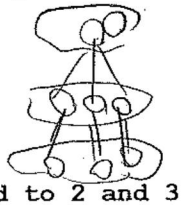
6. (15 points)

You are given a tree T where every node **i** has weight **wi** ≥ 0.

a. Design a polynomial time algorithm to find the weight of the largest weight independent set in T: among all independent sets one with maximum sum of the weights (an **independent set** is a subset of vertices where there are no edges between any of them).

For example, suppose in the following picture w1 = 3, w2 = 1, w3 = 4, w4 = 3, w5 = 6. The maximum independent set has nodes 3,4,5 with weight 4 + 3 + 6 = 13.

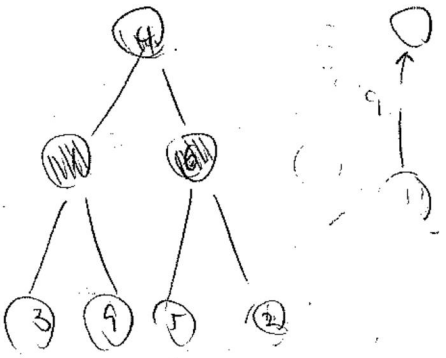




O(V+E)

(1 is connected to 2 and 3. And 2 is connected to 4 and 5)

b. Analyze the time complexity of your algorithm.



a. [Algorithm]

We can solve the problem with recursive approach

We will traverse the tree from the root to the leaves.

At each step, we either include the node to the solution set

or we don't. If we do include it, then we can't include

its children to the solution set

The algorithm is as the follows:

```
maxSum ( node, isNodeAvailable)
    If node is null : Return 0
    Let maxweight := 0
    For every child node Ci of node :
        If isNodeAvailable :
            maxweight = max ( maxweight,
                              node.weight + maxSum( Ci, false)
                              maxSum (Ci, true))

        Else :
            maxweight = max ( maxweight,
                              maxSum ( Ci, true))
```

[proof of correctness]

This algorithm performs exhaustive search, and

thus it is correct.

b.

For every node we either

choose it or we don't,

The recursion goes like

BFS where every node is

visited once and every edge is

visited thrice.

this algorithm takes O(V+E)

where V is # of nodes and

E is # of edges