1) 20 pts

Mark the following statements as **TRUE** or **FALSE**. No need to provide any justification.

[TRUE/FALSE] False

To prove that a problem X is NP-complete, it is sufficient to prove that 3SAT is polynomial time reducible to X.

[TRUE/FALSE] False

Finding the minimum element in a binary max heap of n elements takes O(log n) time

[TRUE/FALSE] False

We are told that in the worst case, algorithm A runs in O(n log n) and algorithm B runs in $O(n^2)$. Based on these facts, there must be some N that when n>N, algorithm A runs faster than algorithm B. $3n\log n$ n^2

[TRUE/FALSE] True

The following recurrence equation $T(n)=3T(n/3) + 0.1\ n$ has the solution: $T(n)= \Theta(n \log(n^2))$. $n^{\log_3 3}$

[TRUE/FALSE] True

Every problem in NP can be solved in exponential time by a deterministic Turing machine

[TRUE/FALSE] True

In Kruskal's MST algorithm, if we choose edges in decreasing (instead of increasing) order of cost, we will end up with a spanning tree of maximum total cost

[TRUE/FALSE] False

If all edges in a graph have capacity 1, then Ford-Fulkerson runs in linear time.

[TRUE/FALSE] False

If problem X can be solved using dynamic programming, then X belongs to P.

[TRUE/FALSE] True

If Vertex-Cover $\in$ P then SAT $\in$ P.

[TRUE/FALSE] False

Assuming P!=NP, and X is a problem belonging to class NP. There is no polynomial time algorithm for X.

Q2 Score: **15.0**

2) 15 pts

We have a directed graph G=(V, E) where each edge (u,v) has a length l(u,v) that is a positive integer. Let n denote the number of vertices in G. Suppose we have previously computed a n x n matrix d[·,·], where for each pair of vertices u,v ∈ V, d[u,v] stores the length of the shortest path from u to v in G. The d[·,·] matrix is provided for you.

Now we add a single edge (a,b) to get the graph G' = (V,E'), where E' = E ∪{(a,b)}. Let l(a,b) denote the length of the new edge. Your job is to compute a new distance matrix d'[·,·], which should be filled in so that d'[u,v] holds the length of the shortest path from u to v in G', for each u,v ∈ V.

(a) Write a concise and efficient algorithm to fill in the d'[·,·] matrix. You should not need more than about 3 lines of pseudocode. (12 pts)

As we just need to consider the effect of newly added directed edge (a, b), we can do the following:

```
for each node u in V:
    for each node v in V:
        d'[u, v] = min( d[u,v], d[u,a] + l(a,b) + d[b, v] )
```

(b) What is the asymptotic worst-case running time of your algorithm? Express your answer in O(·) notation, as a function of n and |E|. (3 pts)

The worst-case running time is $O(n^2)$, as we need 2 nested loops to calculate the new shortest distance between all pairs of nodes.

Q3 Score: **10.0**

3) 10 pts.

Recall the 0/1 knapsack problem:
Input: n items, where item i has profit p_i and weight w_i, and knapsack size is W.
Output: A subset of the items that maximizes profit such that the total weight of the set $\leq$ W.
You may also assume that all $p_i \geq 0$, and $0 < w_i \leq W$.

We have created the following greedy approximation algorithm for 0/1 knapsack:
Sort items according to decreasing p_i/w_i (breaking ties arbitrarily).
While we can add more items to the knapsack, pick items in this order.

Show that this approximation algorithm has no nonzero constant approximation ratio. In other words, if the value of the optimal solution is P*, prove that there is no constant ρ $(1 > \rho > 0)$, where we can guarantee our greedy algorithm to achieve an approximate solution with total profit ρ P*.

Proof: Suppose there exists a constant ρ that achieve an approximate solution with total profit $\rho \cdot P^*$

We construct such a counter example: there are 2 items A and B, the ~~value~~ profit of A is ρ and the profit of B is 2. The weight of A is 1 and the weight of B is $\frac{3}{\rho} > 1$ (as $0 < \rho < 1$). Let knapsack size $W = \frac{3}{\rho}$.

According to this approximation algorithm, we will choose item A first because $\frac{P_A}{W_A} = \rho$. $\frac{P_B}{W_B} = \frac{2}{3}\rho$ so $\frac{P_A}{W_A} > \frac{P_B}{W_B}$.

Then we cannot choose item B because the size is $\frac{3}{\rho}$. Then we get total profit of ρ. $(0 < \rho < 1)$

However, the optimal solution is to choose item B to get profit of 2. And the approximation ratio is $\frac{\rho}{2} < \rho$. $\Rightarrow$ contradiction!

So there is no such constant ρ.

Q4 Score: **15.0**

4) 15 pts.
The graph five-coloring problem is stated as follows: Determine if the vertices of G can be colored using 5 colors such that no two adjacent vertices share the same color.

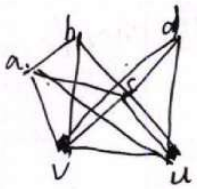
Prove that the five-coloring problem is NP-complete.

Hint: You can assume that graph 3-coloring is NP-complete

Proof: ① The five-coloring problem is in NP. Because the certificate is given any coloring solution, we can check in $O(|V|+|E|)$ time (using BFS) whether there are 2 adjacent nodes that share the same color. So it can be verified in polynomial time. (the verification is: and there are at most 5 colors.)

②

We can reduce 3-coloring problem to 5-coloring problem. Given a graph $G=(V,E)$ and we want to check whether there is a 3-coloring solution. We construct a graph G' by adding two nodes to original graph G and connect v with all the other nodes, connect u with all the other nodes.



Then graph G has a 3-coloring if and only if graph $G'=(V',E')$ has a 5-coloring solution.

Proof: "⇒" If the original graph G has a 3-coloring, then by coloring node v and u with two different colors (which are different from the 3 colors of nodes in G), we get a 5-coloring solution for graph G'.

"⇐" If the graph G' has a 5-color solution, then v and u must be colored with 2 different colors (because v and u are connected) and these 2 colors must be different from the colors of any node belonging to G (because v and u are connected to all the nodes). So all the nodes in G are colored with 3 different colors at most. which forms a 3-coloring solution for G.

As 3-coloring is NP-Complete and 5-coloring is in NP. So 5-coloring is NP complete.

Q5 Score: **15.0**

5) 15 pts.

The price of the recently introduced cryptocurrency, Crypcoin, has been changing rapidly over the last two months. Given the hourly price chart of this currency for the last two months, you are tasked to find out the maximum profit one could have made by buying 100 Crypcoins in one transaction and subsequently selling 100 Crypcoins in one transaction within this period.

Specifically, given the price chart of Crypcoin over this period P[i] for i = 1 to n, we're looking for the maximum value of (P[k]−P[j])*100 for some indices j, k, where $1 \le j < k \le n$.

Examples: If the array is [2, 3, 10, 6, 4, 8, 1] then the returned value should be 8 *100 (Diff between 2 and 10, times 100 Cyrpcoins). If the array is [7, 9, 5, 6, 3, 2] then the returned value should be 2 * 100 (Diff between 7 and 9, times 100 Cyrpcoins)

Describe a divide and conquer algorithm to solve the problem in O(n log n) time. You do not need to prove that your algorithm is correct.

For the input array of size n, we can divide it into 2 parts, each of which has size $\lceil \frac{n}{2} \rceil$ (or $\lceil \frac{n}{2} \rceil + 1$). We recursively get the maximum profit from the left part (P_1) and the right part (P_2), and then we find the maximum profit P_3 of buying 100 Crypcoins at some time in the left part (when the price is lowest) and selling it at some time in the right part (when the price is highest). Finally we return the maximum value of P_1, P_2, P_3.

```
function maxprofit ( Array P[1...n] )
{ if size of P is smaller than 2: return 0.
  Let mid = n/2
  Split the array P into 2 parts:
  Array PLeft = P[1 ... mid]
  Array PRight = P[mid+1 ... n]
  Let P1 = maxprofit (PLeft)
      P2 = maxprofit (PRight)
  Let b = 1 (denote the buy time)
      S = mid+1 (denote the sell time)
  for i = 2 to mid:
      if P[i] < P[b]: let b = i
  for i = mid+2 to n:
      if P[i] > P[s]: let s = i
  Let P3 = (P[s] - P[b]) x100
  return max(P1, P2, P3)
}
```

Let The total complexity be $T(n)$

Then $T(n) = 2T(\frac{n}{2}) + \Theta(n)$

According to Master Theorem, Case 2.

$T(n) = \Theta(n \log n) = O(n \log n)$.

Q6 Score: **10.0**

6) 10 pts.

Recall the Circulation with Lower Bounds problem:

- Each directed edge e has a lower bound l_e and an upper bound c_e on flow
- Node v has demand value d_v, where d_v could be positive, negative, or zero
- Objective is to find a feasible circulation that meets capacity constraints over each edge and satisfies demand conditions for all nodes

We now add the following constraints and objective to this problem:

- For each node v with demand value $d_v = 0$, the flow value through that node should be greater than l_v and less than c_v
- Since we suspect there may be some issues at a specific node x in the network, we want to minimize the flow going through this node as much as possible.

Give a linear programming formulation for this problem.

Let $x_{u,v}$ denote the flow value of edge (u,v)

The objection: Minimize $\max\left(\sum_{(v,x)\in E} x_{(v,x)}, \sum_{(x,u)} x_{(x,u)}\right)$

(We assume the flow ~~through~~ through node x is defined by the maximum of the flow into it and the flow out of it).

The constraints:

$$x_{u,v} \geq l_{(u,v)} \quad \forall (u,v)\in E$$

$$x_{u,v} \leq C_{(u,v)} \quad \forall (u,v)\in E$$

$$\sum_{(u,v)\in E} x_{u,v} - \sum_{(v,p)\in E} x_{v,p} = d_v \quad \forall v \in V$$

$$\sum_{(u,v)\in E} x_{u,v} \geq l_v \quad \forall v\in V \text{ and } d_v = 0.$$

$$\sum_{(u,v)\in E} x_{u,v} \leq C_v \quad \forall v\in V \text{ and } d_v = 0.$$

7) 15 pts

Woody will cut a given log of wood, at any place you choose, for a price equal to the length of the given log. Suppose you have a log of length L, marked to be cut in n different locations labeled $1, 2, \ldots, n$. For simplicity, let indices 0 and n + 1 denote the left and right endpoints of the original log of length L. Let di denote the distance of mark i from the left end of the log, and assume that $0 = d_0 < d_1 < d_2 < \ldots < d_n < d_{n+1} = L$. The wood-cutting problem is the problem of determining the sequence of cuts to the log that will cut the log at all the marked places and minimize Woody's total payment. Remember that for a single cut on a given log, Woody gets paid an amount equal to the length of that log. Give a dynamic programming algorithm to solve this problem.

a) Define (in plain English) subproblems to be solved. (4 pts)

Let $F[i,j]$ denote the ~~maximal~~ minimal total payment we can get from the ~~log~~ mark d_i to d_j of the original log. We need to compute $F[0, n+1]$ ~~$i+1<j$~~ $(i \le j)$

b) Write the recurrence relation for subproblems. (7 pts)

(k is where to cut first)

$F[i,j] = \min(F[i,k] + F[k,j] + d_j - d_i)$. $i < k < j$.

For base cases. $F[i,i] = 0$. for $i = 0, 1, 2 \ldots n+1$.

$F[i, i+1] = 0$.

c) Compute the runtime of the algorithm in terms of n. (4 pts)

```
For i=0 to n+1:
    F[i, i]= 0 , F[i,i+1]=0
For j=1 to n+1:
  For i= j-1 downto 0:
      ~~For k=i to j:~~
      F[i,j] = +∞
      For k= i+1 to j-1:
  F[i,j]= min( F[i,j], F[i,k]+F[k,j]+dj-di )
      Endfor
  Endfor
Return F[0, n+1]
```

There're 3 nested loops in total. so the runtime is $O(n^3)$

Additional Space

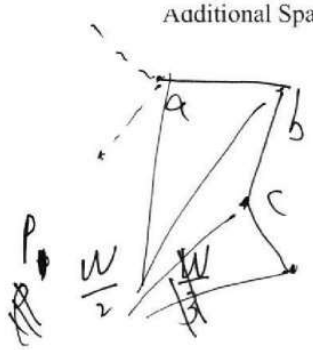
Q9 Not graded

B11C17EB-F16B-4C8D-904F-903989101572

csci570-fa17-midterm3

#242 10 of 10

Additional Space

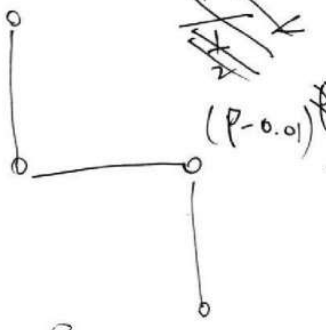


$y > x(p-0.01)$

P_2 ~~$\frac{2}{3}W$~~

$\frac{1}{2}$ $\frac{1}{2}$

$y > \frac{p-0.01}{2}$



$(p-0.01)$ ~~$P_2 > P_2$~~

~~$x \cdot y$~~

$\frac{1}{10}p$

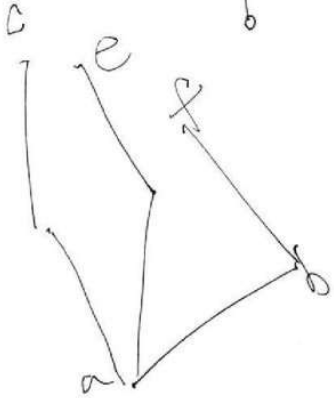
$\frac{y(p-0.01)}{x}$

$\frac{9p}{5} y > 1$

$x + y > 1.$

$x \geq \frac{1}{2}$

$y > \frac{5}{9p}$ $y > \frac{2}{p}$



~~$\frac{pP_2}{x}$~~ ~~$\frac{2P_2}{y}$~~

$\frac{p}{1} > \frac{2}{y}$

$\frac{y}{x} > \frac{2}{p}$

$y > \frac{2}{e}x$

$2 \times \frac{5}{9} \times p \cdot \frac{9}{10}$

$(1 + \frac{2}{e})x > 1.$

$\frac{p+2}{p} x > 1$ $x > \frac{p+2}{p}$