CS180 HW5

Xilai Zhang

TOTAL POINTS

100 / 100

QUESTION 1

1 Problem 1 25 / 25

✓ - 0 pts Correct

- 7 pts dp algorithm not correct
- 5 pts time complexity not correct; no run time analysis
- 20 pts wrong answer but showed efforts
- 25 pts wrong answer, no answer
- 10 pts No detailed algorithm, no dp equations
- 5 pts Does not show algorithm
- 20 pts Does not answer the question
- 25 pts No answer found

QUESTION 2

Problem 2 25 pts

2.1 2.a 15 / 15

✓ - 0 pts Correct

- 5 pts did not update at each position
- 10 pts wrong answer but showed efforts
- 15 pts no answer

2.2 2.b 10 / 10

✓ - 0 pts Correct

- 7 pts wrong answer but showed efforts
- 10 pts no answer

QUESTION 3

3 Problem 3 25 / 25

✓ - 0 pts Correct

- 10 pts Need more explanation
- 5 pts Fail to make MST or visited status accessible among all processors
- 5 pts Fail to show time complexity analysis
- 25 pts No answer found

QUESTION 4

4 Problem 4 25 / 25

✓ - 0 pts Correct

1.

Initialize an empty 2D array A that stores the cube of extra white space.

Initialize a 2D array B of length n that stores the minimum sum up to l_n . the first element of B is 0, all others are infinity.

Initialize an empty array C of length n to store where the line feed occur.

For 0<i<=n:

    For i<=j<=n:

        Initialize temp variable x to be M − j + i − $\sum_{k=i}^{j} l_k$.

        If x<0:

            A[i][j]= infinity

        Else:

            A[i][j]=x^3

For 0<i<=n:

    For 0<=j<i:

        If B[i]> B[j]+A[j+1][i]:

            B[i]= B[j] + A[j+1][i]

            C[i]=j

Return C.

Correctness:

The correctness of this algorithm relies on the recursion that, the minimum sum up to word with index i is the minimum of, all sums of ((minimum of sum up to j where 0<=j<i) and (sum in a line from j+1 to i))

Which is expressed as B[i]= min(B[j] + A[j+1][i]) above. After that we can employ dynamic programming to generate the result.

Complexity:

The two loops will both cost $O(N^2)$ complexity, so in total it is still $O(N^2)$ complexity.

1 Problem 1 **25 / 25**

✓ **- 0 pts Correct**

- 7 pts dp algorithm not correct

- 5 pts time complexity not correct; no run time analysis

- 20 pts wrong answer but showed efforts

- 25 pts wrong answer, no answer

- 10 pts No detailed algorithm, no dp equations

2.

(a) Initialize an array B of length n. initialize all elements to be 1.
Initialize max to be 0.
For 0<i<=n:
    For 0<=j<i:
        If (A[j] < A[i]) && (B[j]+1 > B[i]):
            B[i]= B[j]+1
For 0<i<length of B:
    If B[i]>max:
        Max=B[i]
Return max

Correctness:
We divide the problem by finding the longest ascending sequence that ends at each index i. We iterate through the index sequentially. For each index i, the longest ascending sequence that ends at i will either be itself, because A[i] is larger than all of its processors, or be the longest ascending sequence that ends at j (B[j]) and append A[i] where j<i and A[j]<A[i]. In the later case, we find the maximum B[j] and store B[j]+1 in B[i]. Finally, we iterate through B to find the maximum, which is the longest ascending sequence that ends at a specific index.

Complexity:
The first loop cost $O(N^2)$ and the second loop cost O(N). So in total the complexity is $O(N^2)$.

(b)
Initialize an array B of length n, the first element is A[1] and all others empty.
Initialize index j to be 1 to track the number of elements added to the array.

For 0<i<=n:
    If A[i]> B[j-1]:
        J=j+1
        B[j-1]= A[i]
    If A[i]< B[0]:
        B[0]=A[i]
    Else:
        Find the largest index k in B such that B[k]<=A[i]
        B[k+1]=A[i]
Return j

Correctness:
Array B is an array of end elements of many ascending subsequences that has the possibility of being longest ascending subsequence. If A[i] is bigger than the largest end elements so far, we append A[i] to the longest list. If A[i] is smaller than the

2.1 2.a **15 / 15**

✓ **- 0 pts** **Correct**

- 5 pts did not update at each position

- 10 pts wrong answer but showed efforts

- 15 pts no answer

2.

(a) Initialize an array B of length n. initialize all elements to be 1.
Initialize max to be 0.
For 0<i<=n:
    For 0<=j<i:
        If (A[j] < A[i]) && (B[j]+1 > B[i]):
            B[i]= B[j]+1
For 0<i<length of B:
    If B[i]>max:
        Max=B[i]
Return max

Correctness:
We divide the problem by finding the longest ascending sequence that ends at each index i. We iterate through the index sequentially. For each index i, the longest ascending sequence that ends at i will either be itself, because A[i] is larger than all of its processors, or be the longest ascending sequence that ends at j (B[j]) and append A[i] where j<i and A[j]<A[i]. In the later case, we find the maximum B[j] and store B[j]+1 in B[i]. Finally, we iterate through B to find the maximum, which is the longest ascending sequence that ends at a specific index.

Complexity:
The first loop cost $O(N^2)$ and the second loop cost O(N). So in total the complexity is $O(N^2)$.

(b)
Initialize an array B of length n, the first element is A[1] and all others empty.
Initialize index j to be 1 to track the number of elements added to the array.

For 0<i<=n:
    If A[i]> B[j-1]:
        J=j+1
        B[j-1]= A[i]
    If A[i]< B[0]:
        B[0]=A[i]
    Else:
        Find the largest index k in B such that B[k]<=A[i]
        B[k+1]=A[i]
Return j

Correctness:
Array B is an array of end elements of many ascending subsequences that has the possibility of being longest ascending subsequence. If A[i] is bigger than the largest end elements so far, we append A[i] to the longest list. If A[i] is smaller than the

smallest end element, we initialize a new list of length 1 which has end element A[i]. If A[i] is in between the largest and smallest end elements of all subsequences, we find the largest end element that is smaller than A[i], append A[i] to that subsequence, and discard all other subsequences of the same length. The last element of B is the end element of the longest ascending subsequence. Thus we can find the length of longest ascending subsequence by finding the length of B.

Complexity:
The spec says insert and find operations are all O(log N) complexity. Thus by iterating through the n elements, we will have in total O(NlogN) complexity.

2.2 2.b 10 / 10

✓ **- 0 pts** **Correct**

- 7 pts wrong answer but showed efforts

- 10 pts no answer

3. we can use Bruvka's algorithm and star contraction to solve this problem.

Initialize an array A of n cells corresponding to n vertices. For each processor Pi, store its input in the two cells that corresponds to its endpoints. Thus we have an adjacency list.
Initialize an array C of n cells that stores the mapping of each vertex, initially each vertex maps to itself.
Initialize an array D of e cells that stores whether each edge has been included in MST so far. Initially all edges are not included.
Initialize an array E of n cells that stores the new categories of clusters.
Initialize an array F of n cells that track the current clusters. F maps a cluster name to head or tail. Initially there are n clusters corresponding to n vertices.

While number of clusters in F is not 1:
    Clear all clusters in E.
    Generate head or tail for each cluster in F.
    Initialize an array B to record the minimum outgoing edge for each cluster in F.
    If a cluster gets assigned a head, remove the cluster from F and write the cluster to E.
    For each vertex z, if it is not the first iteration, find the cluster s that z belongs to in array C, otherwise iterate through F:
        If cluster s get assigned a tail:
            For each vertex x that z connects to in A:
                If vertex x is in cluster t which get assigned a head and s is not t:
                    If weight xy is smaller than current weight in B.
                        write information of edge xy in B.
    For each edge in B, include the edge in D. map the endpoint of B that belongs to tail cluster U to the other endpoint which belongs to head cluster V. Update all mappings in C. remove U from F.
    Map remaining clusters in F to itself. write these clusters to E.
    Replace clusters in F with clusters in E.

Correctness:
This algorithm is based on the correctness of Bruvka's algorithm and star contraction. Between two clusters which one gets assigned head and the other gets assigned tail, we connect the minimum edge between them and merge the two clusters. Head and tail are assigned randomly for each cluster. We will keep merging clusters until there is only one cluster left. This satisfies the definition of MST that it has all minimum edges between cuts.

Complexity:
Star contraction will contract the graph into a single cluster in O(logN) round. In each round, we go through all the edges and include minimum weight edge between clusters, which is O(N) complexity. This in total is O(NlogN) complexity.

3 Problem 3 25 / 25

✓ **- 0 pts** **Correct**

- 10 pts Need more explanation

- 5 pts Fail to make MST or visited status accessible among all processors

- 5 pts Fail to show time complexity analysis

- 25 pts No answer found

4.

Sort the knapsacks based on their value per capacity. Let the volumes of knapsacks be V1,V2,…Vn, and let their values be A1,A2,.. An, we can sort the sequence of Ak/ Vk for each k to generate sequence Q such that $\frac{A_1}{V_1} > \frac{A_2}{V_2} > \frac{A_3}{V_3} > \cdots > \frac{A_n}{V_n}$.

Pick knapsack according to Q until the total volume exceeds W, let the last volume that if added would exceed W be V_m . Then, we either pick the sum of values A1,A2,A3…Am-1 or just Am.

Now we prove that this greedy algorithm is at least half of the optimal. We know that if we let partial of Vm to fit such that the sums of volumes add up to exactly W, we will get the sum of values at least better than the optimal. That is, if V1+V2+V3+…+ p* Vm= W, then A1+A2+A3+…+p*Am >= optimal sum of value. Here p is just a portion factor. From this we get A1+A2+A3+…+Am > optimal sum of value. Thus if we pick either A1+A2+…+Am-1 or Am, at least one of them will be at least half of the optimal value.

Finally we show a counter example to illustrate that the solution produced by greedy algorithm is not always the optimal solution. Let A1=2,A2=2,A3=3,V1=1,V2=1,V3=2 and W=3. Then the greedy algorithm will produce the sum of value to be either 4 or 3, which is smaller than the optimal sum of value 5.

4 Problem 4 25 / 25

✓ **- 0 pts** **Correct**

- 5 pts Does not show algorithm

- 20 pts Does not answer the question

- 25 pts No answer found