VE 281 Exam

Wang Yichao, ID: 517370910011

- **Exercise 1.**

  1. In linear partition problem, we need to divide a chain into several groups and try to minimize the maximum of the sum of each group.

  2. No. For example if we divide according to average size for $\{1, 2, 3, 6\}$, we get 9. However, there do exist one division $\{1, 2, 3\}$ and $\{6\}$ that leads to 6.

  3. Assume that the chain S has n elements. Then $M(n, k) = \min\left(\max_{i=1}^{n}(M(n-i, k-1), sum\ of\ the\ last\ i\ elements)\right)$.

  4. The complexity would be factorial because we need to call M for n times, which leads to $n!$.

  5. The M values that is prior to M(n, k) can be stored to save time and space.

  6.

Algorithm 1 dynamic programming

Input: n elements from s_1 to s_n and k is the number of ranges

Output: M(n, k), which is the minimum of the cost of the largest sum of elements in all the division groups.

```
1:  for $int\ i = 1; i < k + 1; i++$ do
2:      M(1, k) = $s_1$
3:  end for
4:  M(0, 1) = 0
5:  for $int\ i = 1; i < n + 1; i++$ do
6:      M(i, 1) = M(i-1, 1) + $s_i$
7:  end for
8:  for $int\ i = 2; i < n + 1; i++$ do
9:      for $int\ j = 2; i < k + 1; j++$ do M(i, j) = $+\infty$
10:         for $int\ k = 1; k < i; k++$ do
11:             $M(i, j) = \min(M(i, j), \max(M(k, j-1), s[i] - s[k]))$
12:         end for
13:     end for
14: end for
```

  7. The idea is similar to recursion, we only use the table to store M(i,j), so it should still be correct. In a maths course, we can use induction to prove it. Basically we only need to focus on the min and max in the loop. We should be safe.

  8. The complexity depends on the loops, so the time complexity should be $\mathcal{O}(k) + \mathcal{O}(n) + \mathcal{O}(n \times k \times n) = \mathcal{O}(kn^2)$.

  9. We create a partition table just like M in dynamic programming, and update the partition when we change the value of corresponding M.

- **Exercise 2.**

  To fulfill the range 0-7, we should run the box B twice. First is used to decide $n \equiv ?$ mod 4(if the result of B is 4, just run again until result is not 4), the second should decide whether it is larger than 3 or not. For example, we can regard 0 and 1 as larger, while 2 and 3 as not larger. Then we can generate random number in 0-7.

  There is no restriction on n. The essence of this this question is the binary representation. But we can go with 5 base as well. It is obvious that every natural number can be represented using base 5. After that we use the 0-4 generator to generate each digit of this number in base 5.(m row for a m-digit n). Something may go wrong with the MSB. For example, if $n = 10$, or 20 in base 5. But we row the 0-4 dice and get something like 21 in base 5(which means out of range or larger than n). In this case, we should row all the dices again. Notice that we are expected to row for 5 times at most before getting it in range. However, the worst case is not guaranteed(but we can live with it). The uniform randomness is obvious.

  Here is the algorithm.

Algorithm 2 random generator

```
Input: the number n which indicates the range $0 \to n$
Output: A random number generated between the range $0 \to n$
1: $digits = \lceil (\log_5 n) + 1 \rceil$
2: $result = +\infty$
3: while $result > n$ do
4:     Run the $0 \to 4$ generator for digits times
5:     Assume the numbers in last step are $d_1$ to $d_{digits}$
6:     $result = \sum_{n=1}^{digits} (d_n \times 5^{n-1})$
7: end while
8: return result
```

- **Exercise 3.** A naive approach. Notice that the length of negative circle will lead to smaller and smaller distance in Bellman-Ford algorithm. Another point is that the length of a cycle is less than $|V|$(assume $|V|$ points in graph).

  Warning: The cost can be very big. If there is a negative cycle that is unconnected to the starting point we choose in Bellman-Ford, my naive algorithm will return false though there do exist a negative cycle. So we should do something learned in VE444 textbook and divide the given graph into several directed connected components, which is beyond the scope.

  Another approach to cope with the warn is simple run my naive algorithm for $|V|$ times, each time we choose a different vertex so that we will not miss any negative cycle unconnected. There is a negative cycle if and only if all $|V|$ times we return False in naive.

Algorithm 3 naive

```
Input: A graph G(V, E)
Output: A bool value that decide if there is a negative cycle.
1: Run the Bellman-Ford once($|V|$ columns)
2: We can get an $|V| \times |V|$ table.
3: Run the Bellman-Ford once again(another $|V| - 1$ rounds)
4: We can get another $|V| - 1 \times |V|$ table.
5: Then we compare the $|V|$th column and the $(2|V| - 1)$th column
6: if the value decreases then
7:     return True
8: end if
9: return False
```

- **Exercise 4.**

  ignored this time

- **Exercise 5.** I will suppose that the Euclidean metric is used to calculate the distance. Please check the pseudo code below.

Algorithm 4 naive

```
Input: k Internet hostspots(with position $p_1, ..., p_k$), a range parameter r, a load L that represent the max number
    of users can be connected to a single hostspot, n clients(with position $q_1, ...q_n$).
Output: Whether all clients can be satisfied(arranged to a hostspot).
 1: for $int\ i = 1; i < n + 1; i + +$ do
 2:     for $int\ j = 1; j < k + 1; j + +$ do
 3:         if $(metric(q_i, p_j) \leq r$ then
 4:             connect $q_i$ and $p_j$
 5:         end if
 6:     end for
 7: end for
 8: So we now have a graph G with k hostspots and n clients with edge between indicating if they can be connected.
 9: Now we add a source and destination to transfer the problem into a network flow problem.
10: for $int\ i = 1; i < n + 1; i + +$ do
11:     connect source to $q_i$, set the capacity to be 1
12: end for
13: for $int\ i = 1; i < k + 1; i + +$ do
14:     connect $p_i$ to destination, set the capacity to be L
15: end for
16: for each connected edge between client and hostspot do
17:     set the capacity to be 1
18: end for
19: $max\_flow = EdmondsKarp(G)$
20: return $max\_flow == n$
```

correctness:

Since the flow out of source is at most n(just add n edges with capacity 1 up). So either the max_flow is n or smaller than n. We will discuss these two cases.

If $max\_flow == n$, we have a flow that satisfy the load constrain L and there is a flow out of each edge from source, which means that every client is connected.

If $max\_flow < n$, then we have at least an edge out of source that has no flow. However, for our graph,

complexity: In my first step, loop in loop to connect the clients with hostspots. So $\mathcal{O}(n \times k)$. The two loops that deal with source and destination node does not matter at all. In the end the Edmonds-Karp algorithm is the most expensive, according to the slides, the time complexity is $\mathcal{O}(|V||E|^2)$, in our case, this is $\mathcal{O}((n + k)n^2k^2)$. So in total the time complexity would be $\mathcal{O}((n + k)n^2k^2)$.

Note: (Integer linear programming can not be used)