Delivery Problem

Brief description:

The delivery company manages one delivery man who has to send several packages that are about to approach the deadline (or have already surpassed the deadline) to multiple places from the headquarter and return back to the headquarter within the shortest path.

Assumptions:

1. **#packages_to_deliver** and #cities require user input. As the complexity of problem is O (n^2 2^n), I carefully limit **0 < #cities ≤ 30** so that the calculate time will not be long. There exists 100000 **#packages** in the headquarters and I limit #**packages_to_deliver** ≤ 10000. In milestone 1, I did not specify the #cities. Since it is still an NP problem, it cannot run in polynomial time, I restrict #city in order not to take lots of time for computing.
2. It only manages one delivery man who has to send #**packages_to_deliver** at a time. Delivery man has to send all of **#packages_to_deliver.**
3. The packages are sent from the same place (headquarter of delivery company).
4. Delivery man does not have to send packages to the place located in the headquarter (#distance is 0) but he/she has to return back to the headquarter after he finishes sending all parcels.
5. The information of existing package (deadline, id) is a 2D **vector<pair<time_t, int>>**.
6. The destination of existing package (id, destination) is Hashmap_{dst}. Id is the key and destination is the value.
7. The distance between each place (city) is known so as to draw the distance matrix

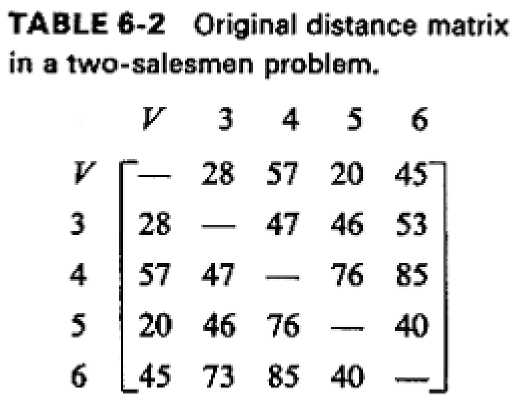


TABLE 6-2 Original distance matrix in a two-salesmen problem.

	V	3	4	5	6
V	—	28	57	20	45
3	28	—	47	46	53
4	57	47	—	76	85
5	20	46	76	—	40
6	45	73	85	40	—

Figure SEQ Figure * ARABIC 1[1]: Distance Matrix

8. It is possible that some parcels have identical destination. The delivery man is able to send all the parcels with the same destination (with the same #city_id)

Functionalities:

1. The delivery company has to figure out #num of packages (id) which are closet to the deadline (or have already surpassed the deadline) from **vector<pair<time_t, int>>**. If the value of deadline is small compared to others, it indicates that the package is approaching the deadline or has already exceeded the deadline. Thus, **Max_Heap** (B. 4.) can be used to generate the results since heap is fast in finding the smallest elements.
2. As hashing data structure have only O (1) time on average in searching the element, using **hashmap** (B.7.) data structure is to find out destination of each packages (#package_id) from Hashmap_{dst}.
3. The delivery man needs to send multiple packages through multiple destinations with the shortest path. Suppose the delivery man started at City A and now he/she is in City j after visiting a few cities. But it's a partial tour. We definitely need to know j, since this will decide which cities are best positioned to visit next. We also need to know all of the cities that we have visited so far, so we don't repeat any of them. Hence, this is a sub-problem. We then can apply this problem to **dynamic programming** (B.8.).
4. The I/O part:
   The input is #**city_num** and #**packages_to_deliver.**
   The output generates the list of shortest paths, i.e. [1, 3, 2, 1]

I confirm that 1,2 functionalities are using max_heap and hashmap data structure separately.

Specify 1,2 functionalities:

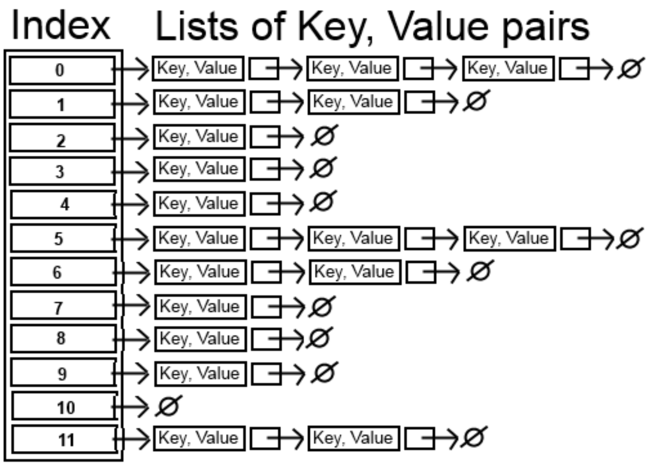
1. **Assumptions: #packages_to_deliver <= 10000, packages = 100000**
   MaxHeap is highly effective on giving the maximum value. After building the heap (O(n)), the maximum value of the max_heap is its root. Therefore, the complexity of finding the maximum value (O (1)) is very considerable. Other data structure like AVL Tree and Red Black Tree will not obtain the maximum value in a constant amount of time.
   In the first functionality, I use max_Heap structure to implement the functionality. The strategy is to build a heap with the first K element (K = **#packages_to_deliver**). Then, I compare the rest of elements in the **vector <pair< time_t, int> >** with the root of max_heap. As the root of max_heap is the maximum of K elements, if one element **E** in the **vector<pair<time_t, int>>** is smaller than the root of max_heap, it indicates that element in max_heap is still not the smallest in the whole vector. It requires to **swap** the root of max_heap with **E** and **max_heapify** the function.
   In the max_Heap method, it includes **max_heapify** and **build_Heap** function**. The complexity** is

O (K + (n-K) log(K)). As I assume that #packages_to_deliver is relatively small so that the method is efficiency.

2. **Assumptions: key and value of Hashmap$_{dst}$ is not null.**

   In the second functionality, I will use hash function to implement the functionality. In this functionality, I need to figure out the destination of **package_ids** immediately through **package_to_deliver.** The strategy of hash function is acted like a dictionary in Python. In a perfect hashing function, it ideally assigns each key to a unique bucket. In this way, the computational complexity of hashing is under a constant amount of time (O (1)). In the normal situation, several keys contain in the same bucket. The computational complexity is still in a constant amount of time O (1+α) (α stands for load factor).

   What is more, Hashmap takes constant amount of time of accessing and it does not have limitation in size. These factors lead it to outperform linked list and array.



   As there exists at least one key and value, utilizing Hashmap structure is a feasible method.

Programming materials:

C++ computer language

Available Platform:

Windows (Visual Studio)

Reference:

1. web.mit.edu. (n.d.). *Section 6-4-11*. [online] Available at: http://web.mit.edu/urban_or_book/www///book/chapter6/6.4.11.html [Accessed 7 Oct. 2020].

2. www.quora.com. (n.d.). *What is HashMap, and why do we use it in programming? - Quora*. [online] Available at: https://www.quora.com/What-is-HashMap-and-why-do-we-use-it-in-programming [Accessed 7 Oct. 2020].