Data Structure and Algorithm

Homework #4

Due: 2:20pm, Thursday, May 16, 2013

TA email: dsa1@csie.ntu.edu.tw

=== Homework submission instructions ===

- For Problem 1, submit your source codes, a Makefile to compile the source, and a brief documentation to the SVN server (katrina.csie.ntu.edu.tw). You should create a new folder "`hw4`" and put these three files in it.
- The filenames of the Makefile, and the documentation file should be "`Makefile`" and "`report.txt`", respectively; you can use any filenames for the source codes, but the filename of the executable file which is generated after executing "`make`" command should be "`main.exe`". You will get some penalties in your grade if your submission does not follow the naming rule. The documentation file should be in plain text format (`.txt` file). In the documentation file you should explain how your code works, **the reference**, and anything you would like to convey to the TAs.
- For Problem 2 through 5, submit the answers via the SVN server (electronic copy) or to the TA at the beginning of class on the due date (hard copy). Please hand in your homework with A4 paper.
- Except the programming assignment, each student may only choose to submit the homework in only one way; either all in hard copies or all via SVN. If you submit your homework partially in one way and partially in the other way, you might only get the score of the part submitted as hard copies or the part submitted via SVN (the part that the grading TA chooses).
- If you choose to submit the answers of the writing problems through SVN, please combine the answers of all writing problems into only ONE file in the pdf format, with the file name in the format of "`hw4_[student ID].pdf`" (e.g. "`hw4_b01902888.pdf`"); otherwise, you might only get the score of one of the files (the one that the grading TA chooses).
- Discussions with others are encouraged. However, you should write down your solutions by your own words. In addition, for each problem you have to specify the references (the Internet URL you consulted with or the people you discussed with) on the first page of your solution to that problem, but on `report.txt` for the programming assignment.
- **NO LATE SUBMISSION IS ALLOWED** for the homework submission in hard copies - no score will be given for the part that is submitted after the deadline. For submissions via SVN (including the programming assignment and electronic copies of the writing problems), up to one day of delay is allowed; however, the score of the part that is submitted after the deadline will get some penalties according to the following rule (the time will be in seconds):

$$\text{LATE SCORE} = \text{ORIGINAL SCORE} \times (1 - DelayTime/86400)$$

Problem 1. The Topmost Hotel (30%)

You are a software engineer in *Super Algorithm Design Co., Ltd.* As the most promising engineer in the company, your boss assigned an important work to you.

This is the case from a top 10 hotel.

To be an excellent hotel, they provided lots of useful services. After analyzing their statistical data, they found that many customers travel from city a, stay in the hotel for a few days, and then go to city b. As a result, they built a new tool so that the customers can plan their travel in a much easier way. The tool collects information about all possible routes, using trains, buses, taxis, flights, etc., and gives the customers an optimal itinerary that has the minimum price.

At first, the web developer in the hotel used DFS to calculate the price for all possible solutions to get the optical itinerary in the algorithm. However, with the growth of the hotel, the data became much larger and the number of queries grew up to a few millions per minute. Of course, an algorithm like DFS does not work anymore due to its high time complexity, so the web developer came to your company for help.

Your program will be given m routes between n cities and q queries.

The cities will be indexed from 1 to n, and the hotel will always be located in city 1.

Then, there are m routes. The i-th route costs people p_i dollar(s) to take them from s_i to t_i, but **NOT** from t_i to s_i.

For each query, your program should find the optimal itinerary that travels from a to the hotel in city 1 and then to city b with the minimum price.

Input Format

The test data will be given in the following format.

The 1st line contains 2 integers n, m, separated by a single space character.

In the next m lines, the i-th line contains 3 integers s_i, t_i, p_i ($1 \le s_i, t_i \le n$) separated by single space characters.

The next line contains an integer q.

In the next q lines, the j-th line contains 2 integers a_j, b_j ($1 \le a_j, b_j \le n$) separated by a single space character.

Output Format

For each query a_i, b_i, your program should output an integer indicating the minimum cost of the trip $a_i \to 1 \to b_i$. This means that the customer begins the trip from city a_i, stops over in the hotel (city 1), then goes on to city b_i.

If there is no way to achieve the goal, output "-1".

Grading

There are some changes in the submission instructions in the first page compared to previous homework, so please read them carefully.

- 3 points for the report, including the reference.
  Remember to write the reference of *Problem 1* in `report.txt`.
- 2 points for Makefile, including whether the compilation of your source codes is successful.
- 1 point for each set of test data; 25 points in total.
  The execution time limit for each set of test data is 3 seconds.

The following are some conditions you can assume for the test data.

- In 2 sets of the test data, $n \leq 8$, $m = \frac{n(n-1)}{2}$, $q \leq 64$.
- In 4 sets of the test data, $n \leq 200$, $m = \frac{n(n-1)}{2}$, $q \leq 100$.
- In 6 sets of the test data, $n \leq 2000$, $q \leq 100$.
- In 10 sets of the test data, $n \leq 2000$.
- In all test data, $n \leq 400\ 000$, $m \leq 600\ 000$, $q \leq 400\ 000$.
- In 4 sets of the test data, $p_i = 1$.
- In all test data, $0 < p_i \leq 1\ 000\ 000\ 000$ (10^9).
- In 8 sets of the test data, the solutions of all queries always exsit.
- In 20 sets of the test data, for any two cities a, b, there is at most 1 route from a to b, and no route can bring people from a city back to itself.

The input format of the test data is always correct and as specified in this problem description.

Sample Input

```
5 7
2 1 3
2 3 1
3 1 10
1 4 1
4 5 1
1 5 3
3 4 1
4
2 5
3 5
5 2
1 1
```

Sample Output

```
5
12
-1
0
```

More Samples

We provide 3 sets of test data, available on the course website. The files with both the UNIX newline format and DOS newline format are provided. You can download them and test your program locally.

- `sample-small`: The distribution of the cities is the same as the sample input but queries of all pairs are given.
- `sample-medium`: A random generated test data with 2000 cities. Most conditions are satisfied.
- `sample-large`: Generated with the same generator as the last 4 sets of the real test data (to be used for grading). Be careful for all possible cases.

Hints and Technical reminders

- Read and finish *Problem 2* before you begin to solve this problem. There are some ideas of the algorithm design. They will make this problem much easier.
- Use adjacency lists instead of adjacency matrices, otherwise the time complexity and the space complexity of your program will be $O(n^2)$.

  It will become the bottleneck of your program, even if your algorithm can do better (in time and in space).
- Some answers of query will be less than $2n \max(q_i)$; this can NOT be stored in a 32-bit integer. Overflow may occur with some test data.

  Therefore, use `long long int` when calculating the cost. Use `scanf("%lld")` to read and `printf("%lld")` to write.

  For WinXP user, use `"%I64d"` when testing your program locally. **But, remember to change it back to `"%lld"` before committing.**
- Do NOT use the Bellman-Ford algorithm, even if it is much easier to implement, due to its worse time complexity of $O(nm)$.

Suggestions and Reference

- Split your program into functions and complete the test of a data structure or an algorithm before using it in the program.

  For example, after implementing a $O(n^3)$ algorithm, confirm that it is correct. After that, continue to improve it.

  Using this trick would make the debug process easier.
- If you could not solve this problem completely, DO NOT GIVE UP. Solve even just some special cases and you will earn partial points. (Conditions are given in the "Grading" section.)
- A 2.0 GHz CPU can more or less execute 2×10^9 arithmetic operations in 1 second, though some operations, such as multiplication and division, could take longer than others. To be on the safe side, we can assume that it takes 10^{-7} seconds for per "unit" of input size in the algorithm. You can use this to roughly estimate the running time of your program.

  For example, an $O(nm)$ algorithm spends about $nm/10^7$ second (s).
- The length of readable official solution is about 190 lines.
- Each test data can be solved in 1 second on the CSIE 217 work stations by the official solution without any compiler optimization.

***Problem* 2.** Programming Guidelines for "The Topmost Hotel" (20%)

In the following questions, you can assume that the edges in the graph all have positive weights and the graph is directed.

2.1 *The shortest way to the hotel (4%)*

Consider the path P_{ACB} from city A through city C to city B. We can divide P_{ACB} into P_{AC} and P_{CB}. Please prove that P_{AC} and P_{CB} are respectively the shortest paths for $A \to C$ and $C \to B$ *if and only if* P_{ACB} is the shortest path for $A \to C \to B$. As a result, we can divide Problem 1 into two parts: finding P_{A1} and P_{1B}.

2.2 *One for all and all for one (2%)*

In Problem 1, we have to calculate the cost of the travel. Assume that the running time of the Dijkstra algorithm on the given graph is T. Therefore, if we build the table that we used in the Dijkstra algorithm for every query, the running time for q queries would be $q \times T$. This is actually not necessary, since the source vertex is the same for all queries. Please describe an algorithm to pre-calculate a static table (just ONE table) and each query would only take $O(1)$ time.

2.3 *More heap, more help (2%)*

To calculate this static table in the Dijkstra algorithm, we may need a good data structure to bring down the time complexity. The ideal one may be a heap, from which we can determine the vertex with the smallest distance[] value (see the lecture slides) in $O(\log N)$ instead of $O(N)$, when using only an ordinary array. However, since the cost (the priority value used in the heap) of the item needs to be updated in every iteration of the Dijkstra algorithm, it is necessary to create a heap with complex structure designs that can perform quick edits to specific nodes in $O(\log N)$.

One easier solution is that, when distance[] is updated, instead of updating the priority of the corresponding item in the heap, we simply insert a new item with the new priority value into the heap. In this case, there will be multiple items in the heap representing the same vertex. Interestingly, even so, the size of the heap will still be $O(E)$ in the worst case. As a result, the time complexities of both the insert operation and the delete operation are $O(\log E)$. Describe how to deal with the duplicate items that represent the same vertex (but have different priority values) in the heap when running the Dijkstra algorithm? **HINT: Take extra care when deleting the minimum priority item from the heap.**

2.4 *"Artskjid" (4%)*

The Dijkstra algorithm introduced in the class can determine the shortest paths from a specific source vertex to all other vertices in the graph. However, what if we would like to determine shortest paths *to a specific destination vertex* from all other vertices in the graph? Please briefly describe how to modify the original Dijkstra algorithm to construct the shortest path table. **HINT: Try to reverse the direction of the edges in the graph.**

2.5 *Am I tricked (8%)*

Please specify whether the following statements are **true** or **false**. If it is true, explain why it is true. Otherwise, please give counter-examples.

2.5.a It is NOT possible that there exists a cycle in the shortest path P_{AB} from city A to city B. *(i.e., there is a city that is visited twice)*

2.5.b It is NOT possible that there exists an edge that is travelled through twice in the shortest path P_{AB} from city A to city B.

2.5.c It is NOT possible that there exists a cycle in the shortest path P_{ACB} from city A through city C to city B. *(i.e., there is a city that is visited twice)*

2.5.d It is NOT possible that there exists an edge that is travelled through twice in the shortest path P_{ACB} from city A through city C to city B.

Problem 3. Minimum Cost Spanning Tree (15%)

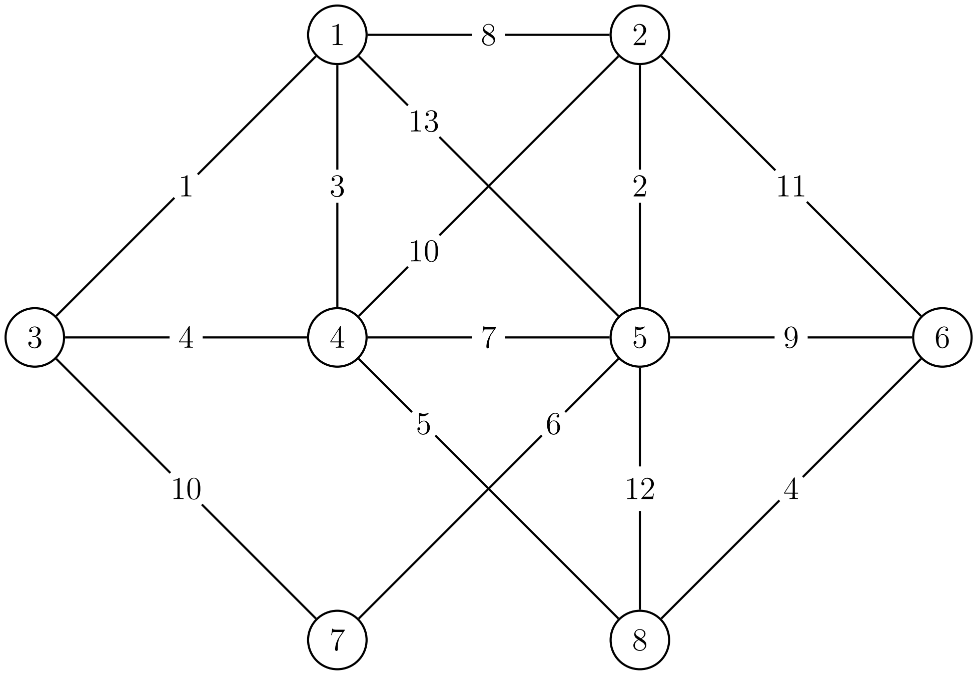

Figure 1: An Undirected Graph

3.1. (5%) Use Kruskal's algorithm to find the MST of the graph in Figure 1. Please draw your results and fill the table below. Each row represents an iteration during the execution of the algorithm. The first column should show the selected edge in each iteration, and the second column should show whether the selected edge is part of the MST. The third column should show the edges that have been selected to be part of the MST in that iteration. The fourth column should show the sum of the weights of the edges in the third column.

Edge	T/F	The set of selected edges in the MST	Weight Sum
$(1,3)$	T	$\emptyset$	0
		$(1,3)$	1
end			

3.2. (5%) Use Prim's algorithm to find the MST of Figure 1 from vertex 1. Please draw your result and fill the table below.

Edge	The set of selected edges in the MST	Weight Sum
$(1,3)$	$\emptyset$	0
	$(1,3)$	1
end		

3.3. (5%) Kruskal's algorithm could build different MSTs if the graph contains edges with the same weight. Is that the case for the graph in Figure 1? If so, draw two different MSTs of the graph in Figure 1. If not, explain the reason briefly.

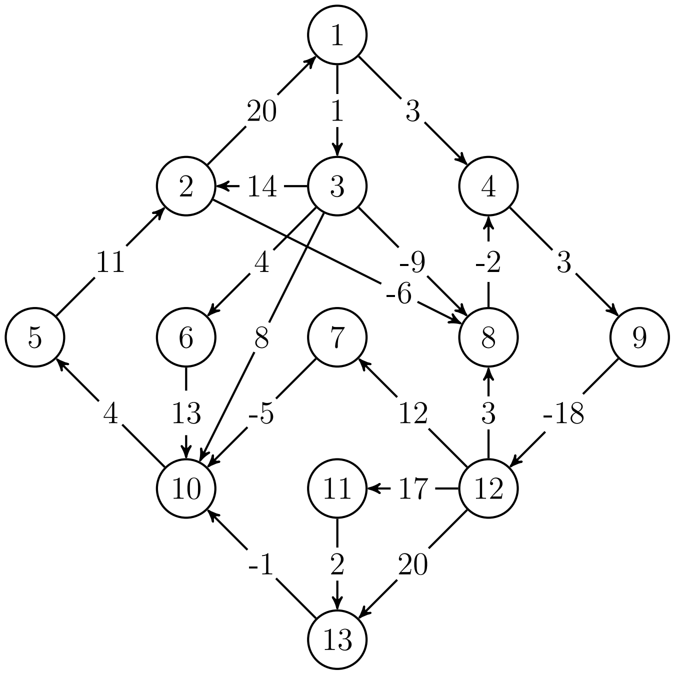

Figure 2: An Directed Graph

Problem 4. Bellman-Ford Algorithm (15%)

In the class, we learned that the Bellman-Ford algorithm can find the shortest path in a graph with negative-weight edges. Figure 2 is a directed graph with negative-weight edges. Please answer the following questions about the Bellman-Ford algorithm.

4.1. (9%) Use the Bellman-Ford algorithm to find the shortest paths from vertex 1 to all other vertices in the graph in Figure 2. Please calculate the results of each iteration and fill the table below.

4.2. (6%) In the class, we learned that the Bellman-Ford algorithm can find the shortest paths in $|V| - 1$ iterations. In this problem, you need to do three more iterations continuing from the previous problem and fill the table below.

Please describe the phenomenon occurs in these three iterations and find out the cause of this phenomenon. By this phenomenon and its cause, can you give an application of the Bellman-Ford algorithm?

Table 1: For problem 4.1

Iter.	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	0												
2	0												
3	0												
4	0												
5	0												
6	0												
7	0												
8	0												
9	0												
10	0												
11	0												
12	0												

Table 2: For problem 4.2

Iter.	1	2	3	4	5	6	7	8	9	10	11	12	13
13	0												
14	0												
15	0												

Problem 5. Second-best Minimum Cost Spanning Tree (20%)

Given a weighted, undirected, and connected graph $G = (V, E)$, in which all edge weights are distinct. The cost of a tree T is the sum of the weights of the edges in T. You may assume that graphs or trees in this problem are given by adjacency lists.

5.1. (5%) Show that the minimum cost spanning tree T^* of G is unique, due to the uniqueness of edge weights in G. (Hint: if there exist two different minimum cost spanning tree T^* and T^{**}, we can find a contradiction that there exists a spanning tree whose cost is less than T^*)

5.2. (5%) A **second-best minimum cost spanning tree** of G is a spanning tree T' of G that has a cost no greater than any other spanning tree of G, except T^*. The minimum cost spanning tree T^* is unique, but the second-best minimum cost spanning tree may not be. Please give an example of a graph G' to show that the second-best minimum cost spanning tree need not be unique. Remember that all edge weights of G' should be distinct. Draw two different second-best minimum cost spanning tree clearly.

5.3. (5%) The property of a spanning tree is that, for a graph G and its minimum cost spanning tree T^*, there exist two edges $e \in T^*$ and $e' \in E - T^*$ such that $T^* + e' - e$ is a second-best minimum cost spanning tree. More clearly, after adding an edge e' to T^*, another edge $e \in T^*$ should be removed from the cycle that is just formed. (See Figure 3 for more details).

Briefly describe (or write the pseudo code of) an algorithm to find the edge e that minimize the cost of $T^* + e' - e$ in $O(|V|)$ time when given G, T^*, and e'. With this subroutine, we can derive a $O(|E||V|)$ algorithm for finding the second-best minimum cost spanning tree by testing the case of each edge in $E - T^*$ being added.

5.4. (5%) Modify your algorithm in problem 5.3 (or design a new one) to calculate the matrix A$[u][v]$ for each $u, v \in V$ in totally $O(|V|^2)$ time. A$[u][v]$ means if the added edge is $e' = (u, v)$, the corresponding removed edge is $e =$ A$[u][v]$. With matrix A$[u][v]$, testing of adding each edge can be done in $O(1)$ and then we get a $O(|V|^2 + |E|)$ algorithm for finding the second-best minimum cost spanning tree.

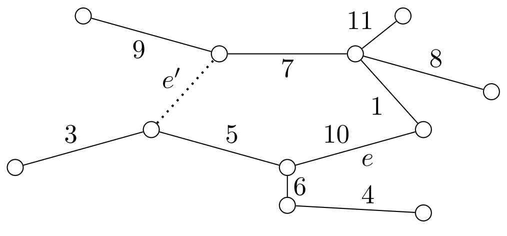

Figure 3: The minimum spanning tree T^*, the added edge e' and the edge e that should be removed. Note that removing any of edges 5,10,1,7 can result in new trees, but only when removing edge 10 the cost of the new tree is minimized.

Problem 6. Strongly Connected Components (0%)

In the class, we learned that strongly connected components are the maximal sub-graphs which are strongly connected. But we did not learn how to find strongly connected components in a graph. Please read section 22.5 (pp.615-620) in the textbook *"Introduction to Algorithms"*. You will learn a beautiful algorithm and its proof. There are also some interesting findings about strongly connected components in the textbook. **Incentive: the related problems have a HIGH probability to appear in the final exam.**