Laboratory work # 6

Student: *Zhanghao Chen*Student ID: 21321205  
Timus Name: *BennyChan*

Mail: [1824362950@qq.com](mailto:1824362950@qq.com)

Problem # 1628 *White Streaks*

Screenshot from Timus:



Explanation of algorithm:

1. Create two vectors of size *m*+1 and *n*+1, and then adds *n*+1 and *m*+1 to each element of *p*[*i*] and *q*[*i*] by a traversal loop to ensure that any element in the vector has a comparison object.

2. Add the input values *x* and *y* to the *p* and *q* vectors through another loop. Then I use two nested loops to iterate over each element in the *p* and *q* vector, using the "temp" variable to keep track of previous elements. If the difference between the current element and the "temp" variable is greater than 2, the "sum" variable is incremented by 1. If the difference is greater than 1, I use another loop to traverse the elements of the *p*[*t\_q*] vector, where *t\_q* is a specific value, to find the element corresponding to the current element. If the condition is met, the "sum" variable is incremented by 1.

3. Finally, I output the value of "sum".

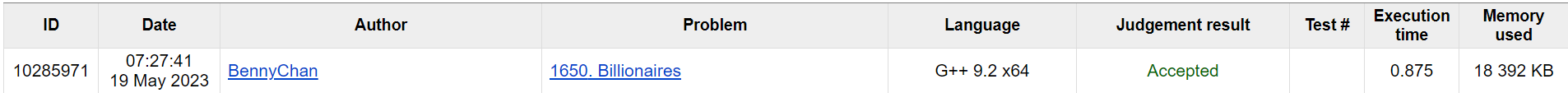
Computational complexity of algorithm:

Source code:

1. #include <iostream>
2. #include <vector>
3. #include <algorithm>
5. **using** **namespace** std;
7. **int** main() {
8. **int** m, n, k;
9. cin >> m >> n >> k;
10. **int** sum = 0;
11. vector<vector<**int**>> p(m+1);
12. vector<vector<**int**>> q(n+1);
14. **for** (**int** i = 1; i <= m; i++) {
15. p[i].push\_back(n+1);
16. }
17. **for** (**int** i = 1; i <= n; i++) {
18. q[i].push\_back(m+1);
19. }
20. **for** (**int** i = 0; i < k; i++) {
21. **int** x, y;
22. cin >> x >> y;
23. p[x].push\_back(y);
24. q[y].push\_back(x);
25. }
27. **for** (**int** i = 1; i <= m; i++) {
28. **int** temp = 0;
29. sort(p[i].begin(), p[i].end());
30. **for** (**int** j = 0; j < p[i].size(); j++) {
31. **if** (p[i][j]-temp > 2) sum++;
32. temp = p[i][j];
33. }
34. }
35. **for** (**int** i = 1; i <= n; i++) {
36. **int** temp = 0;
37. sort(q[i].begin(), q[i].end());
38. **for** (**int** j = 0; j < q[i].size(); j++) {
39. **if** (q[i][j]-temp > 2) sum++;
40. **else** **if** (q[i][j]-temp > 1) {
41. **int** t\_q = q[i][j]-1;
42. temp = 0;
43. **for** (**int** l = 0; l < p[t\_q].size(); l++) {
44. **if** (p[t\_q][l] > i) {
45. **if** (p[t\_q][l]-temp <= 2) sum++;
46. **break**;
47. }
48. temp = p[t\_q][l];
49. }
50. }
51. temp = q[i][j];
52. }
53. }
54. cout << sum << endl;
55. **return** 0;
56. }

Problem # 1650 *Billionaires*

Screenshot from Timus:



Explanation of algorithm:

1. Initialize some data structures, including pairs and maps.

2. Create a vector of pairs, load in the values of *m* and *q*, and enter another loop that keeps reading in day, name, and city, adding them to the vector. Then, we sort the vector and then enter a loop where for each *m*, we check the elements of the vector whose day is equal to the current *m* and update the map and set based on name and city.

3. Check the size of the set, and if the size is greater than 1, we remove the largest element and compare it with the next largest element. If the largest element has more money than the second largest element, the counter for that city is incremented by 1.

4. Reinsert the largest element into set and output the names of all cities whose counter is not 0.

Computational complexity of algorithm:

* *q* is the number of queries and *m* is the number of days.

Source code:

1. #include <iostream>
2. #include <map>
3. #include <set>
4. #include <utility>
5. #include <vector>
7. **using** ll = **long** **long**;
9. **int** main() {
10. **int** num;
11. std::cin >> num;
13. std::set<std::pair<ll, std::string>> cities;
14. std::map<std::string, ll> moneys;
15. std::map<std::string, ll> bankirs;
16. std::map<std::string, std::string> bankirsCity;
18. **for** (**int** i = 0; i < num; i++) {
19. std::string name, city;
20. ll money;
21. std::cin >> name >> city >> money;
22. bankirs[name] = money;
23. bankirsCity[name] = city;
24. moneys[city] += money;
25. }
27. **for** (**const** auto&[city, money] : moneys) {
28. cities.insert(std::make\_pair(money, city));
29. }
31. std::vector<std::pair<**int**, std::pair<std::string, std::string>>> v;
32. **int** m, q;
33. std::cin >> m >> q;
35. **for** (**int** i = 0; i < q; i++) {
36. **int** day;
37. std::string name, city;
38. std::cin >> day >> name >> city;
39. v.push\_back(std::make\_pair(day, std::make\_pair(name, city)));
40. }
42. std::sort(v.begin(), v.end());
44. std::map<std::string, **int**> ans;
46. **int** curr = 0;
47. **for** (**int** i = 0; i <= m - 1; i++) {
48. **while** (curr < v.size() && v[curr].first == i) {
49. std::string bankir, oldCity, newCity;
50. ll sum;
51. bankir = v[curr].second.first;
52. oldCity = bankirsCity[bankir];
53. newCity = v[curr].second.second;
54. sum = bankirs[bankir];
56. **if** (cities.count(std::make\_pair(moneys[oldCity], oldCity))) {
57. cities.erase(std::make\_pair(moneys[oldCity], oldCity));
58. }
59. **if** (cities.count(std::make\_pair(moneys[newCity], newCity))) {
60. cities.erase(std::make\_pair(moneys[newCity], newCity));
61. }
62. bankirsCity[bankir] = newCity;
63. moneys[oldCity] -= sum;
64. moneys[newCity] += sum;
65. cities.insert(std::make\_pair(moneys[oldCity], oldCity));
66. cities.insert(std::make\_pair(moneys[newCity], newCity));
68. curr++;
69. }
71. **if** (cities.size() > 1) {
72. auto a = \*cities.rbegin();
73. cities.erase(a);
74. auto b = \*cities.rbegin();
75. **if** (a.first > b.first) {
76. ans[a.second]++;
77. }
78. cities.insert(a);
79. } **else** {
80. ans[(\*cities.rbegin()).second]++;
81. }
82. }
84. **for** (**const** auto&[city, count] : ans) {
85. **if** (count == 0) {
86. **continue**;
87. }
88. std::cout << city << " " << count << std::endl;
89. }
90. }