Laboratory work # 5

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

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

Problem # 1521 *War Games 2*

Screenshot from Timus:



Explanation of algorithm:

The solveArrayCopy () function takes two inputs: *cnt* and *k*. It creates a vector '*arr*' of size *cnt* and fills it with values ranging from 1 to *cnt*. The algorithm then proceeds to iterate *cnt* times. In each iteration, it selects the *k*th element from the start index and prints it. It then removes this element from the vector by shifting the remaining elements left or right and adjusting the indices accordingly. The algorithm continues this process until there is only one element left in the vector.

Computational complexity of algorithm:

Source code:

1. #include <bits/stdc++.h>
2. **using** **namespace** std;
4. **void** solveArrayCopy() {
5. **int** cnt, k;
6. cin >> cnt >> k;
7. vector<**int**> arr(cnt);
8. **for** (**int** i = 1; i <= cnt; i++) arr[i - 1] = i;
9. **int** left = 0, right = cnt - 1, start = 0, tmp1 = 0;
10. **for** (**int** leaveCnt = 0; leaveCnt < cnt; leaveCnt++) {
11. **if** (right == left) {
12. cout << arr[left] << " ";
13. **break**;
14. }
15. tmp1 = start + (k - 1);
16. **if** (tmp1 > right) {
17. tmp1 = (tmp1 - right) % (right - left + 1) + left - 1;
18. **if** (tmp1 < left) tmp1 = right;
19. }
20. cout << arr[tmp1] << " ";
21. **if** (tmp1 - left > right - tmp1) {
22. **for** (**int** i = tmp1 + 1; i <= right; i++) arr[i - 1] = arr[i];
23. right--;
24. start = tmp1;
25. } **else** {
26. **for** (**int** i = tmp1 - 1; i >= left; i--) arr[i + 1] = arr[i];
27. left++;
28. start = tmp1 + 1;
29. }
30. }
31. cout << endl;
32. }
34. **int** main() {
35. solveArrayCopy();
36. **return** 0;
37. }

Problem # 1494 *Monobilliards*

Screenshot from Timus:



Explanation of algorithm:

To address the problem at hand, we can implement a solution that involves creating two arrays. Array *A* will be used to store the input value (i.e., the number of the ball taken out), while array *B* will store the ball into the bag in order. Next, we will compare the values in arrays *A* and *B*. If a value in array *A* is equal to that in array *B*, we will update the corresponding value in array *B* and retrieve the next value in array *A*. This process will continue until all values in array *A* have been processed. By following this approach, we can ensure that the balls are bagged in the required order, and all the numbers in the final array *B* will be retrieved as expected.

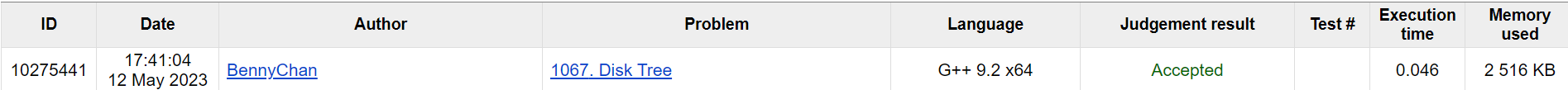
Computational complexity of algorithm:

Source code:

1. #include <cstdio>
2. **using** **namespace** std;
4. **int** a[100002], stack[100002];
6. **int** main()
7. {
8. **int** n;
9. scanf("%d", &n);
10. **for** (**int** i = 1; i <= n; ++i)
11. {
12. scanf("%d", &a[i]);
13. }
14. **int** top = 0, index = 1;
15. **for** (**int** i = 1; i <= n; ++i)
16. {
17. stack[++top] = i;
18. **while** (top && stack[top] == a[index])
19. {
20. top--;
21. index++;
22. }
23. }
24. **if** (top)
25. {
26. printf("Cheater");
27. }
28. **else**
29. {
30. printf("Not a proof");
31. }
32. **return** 0;
33. }

Problem # 1067 *Disk Tree*

Screenshot from Timus:



Explanation of algorithm:

We use a **tree structure** using a linked list and a node class to represent the directory structure, where each node represents a directory or file. Each node object is stored in a list which acts as a container for the entire directory tree. The program builds a tree structure by adding nodes step by step and uses a recursive method to print out all nodes and their children. The put () function adds nodes level by level by splitting the path into different levels of directories. The putSorted () function is responsible for inserting new nodes into the node list at each level in alphabetical order. The print Directory () function traverses the tree in a depth-first manner and prints out the names of each directory, with each level indented by a certain number of spaces. The main () function initializes an empty directory tree, reads in a number '*N*' of directory paths to add, and calls the put function for each path. It then calls printDirectory () function to output the entire directory tree.

Computational complexity of algorithm:

Source code:

1. #include <iostream>
2. #include <list>
3. #include <string>
4. **using** **namespace** std;
6. **class** Node {
7. **public**:
8. string date;
9. list<Node\*> next;
10. Node(string date) {
11. **this**->date = date;
12. }
13. string toString() {
14. **return** date;
15. }
16. };
18. Node\* putSorted(list<Node\*>& tree, string name) {
19. list<Node\*>::iterator it = tree.begin();
20. Node\* newDir = **new** Node(name);
21. **bool** inserted = **false**;
22. **while** (it != tree.end()) {
23. Node\* dir = \*it;
24. **int** compare = dir->date.compare(name);
25. **if** (compare == 0) {
26. inserted = **true**;
27. newDir = dir;
28. **break**;
29. }
30. **else** **if** (compare > 0) {
31. inserted = **true**;
32. tree.insert(it, newDir);
33. **break**;
34. }
35. it++;
36. }
37. **if** (!inserted) {
38. tree.push\_back(newDir);
39. }
40. **return** newDir;
41. }
43. **void** put(list<Node\*>& tree, string directory) {
44. **int** slashId = directory.find('\\\\');
45. **if** (slashId == -1) {
46. putSorted(tree, directory);
47. }
48. **else** {
49. string currentDirectory = directory.substr(0, slashId);
50. string nextDirectory = directory.substr(slashId + 1);
51. Node\* dir = putSorted(tree, currentDirectory);
52. **if** (dir->next.empty())
53. dir->next = list<Node\*>();
54. put(dir->next, nextDirectory);
55. }
56. }
58. **void** printDirectory(list<Node\*>& tree, **int** level) {
59. **if** (!tree.empty()) {
60. list<Node\*>::iterator it = tree.begin();
61. **while** (it != tree.end()) {
62. Node\* dir = \*it;
63. **for** (**int** j = 0; j < level; j++) {
64. cout << ' ';
65. }
66. cout << dir->date << endl;
67. printDirectory(dir->next, level + 1);
68. it++;
69. }
70. }
71. }
73. **int** main() {
74. list<Node\*> tree = list<Node\*>();
75. **int** N;
76. cin >> N;
77. cin.ignore();
78. **for** (**int** i = 0; i < N; i++) {
79. string directory;
80. getline(cin, directory);
81. put(tree, directory);
82. }
83. printDirectory(tree, 0);
84. **return** 0;
85. }