LEETCODE

Contents

[LEETCODE 1](#_Toc388598641)

[1. Two Sum 23](#_Toc388598642)

[题目 23](#_Toc388598643)

[解法 24](#_Toc388598644)

[分析 24](#_Toc388598645)

[程序 24](#_Toc388598646)

[2. Median of Two Sorted Arrays 25](#_Toc388598647)

[题目 25](#_Toc388598648)

[解法1 25](#_Toc388598649)

[分析 25](#_Toc388598650)

[程序 25](#_Toc388598651)

[解法2 27](#_Toc388598652)

[分析 27](#_Toc388598653)

[3. Longest Substring Without Repeating Characters 27](#_Toc388598654)

[题目 27](#_Toc388598655)

[解法 27](#_Toc388598656)

[分析 27](#_Toc388598657)

[程序 27](#_Toc388598658)

[4. Add Two Numbers 28](#_Toc388598659)

[题目 28](#_Toc388598660)

[解法 28](#_Toc388598661)

[分析 28](#_Toc388598662)

[程序 29](#_Toc388598663)

[5. Longest Palindromic Substring 30](#_Toc388598664)

[题目 30](#_Toc388598665)

[解法 30](#_Toc388598666)

[分析 30](#_Toc388598667)

[程序 31](#_Toc388598668)

[6. Reverse Integer 32](#_Toc388598669)

[题目 32](#_Toc388598670)

[解法 32](#_Toc388598671)

[分析 32](#_Toc388598672)

[程序 32](#_Toc388598673)

[7. String to Integer (atoi) 33](#_Toc388598674)

[题目 33](#_Toc388598675)

[解法 33](#_Toc388598676)

[分析 33](#_Toc388598677)

[程序 33](#_Toc388598678)

[8. Palindrome Number 35](#_Toc388598679)

[题目 35](#_Toc388598680)

[解法 35](#_Toc388598681)

[分析 35](#_Toc388598682)

[程序 35](#_Toc388598683)

[9. Regular Expression Matching 36](#_Toc388598684)

[题目 36](#_Toc388598685)

[解法 36](#_Toc388598686)

[分析 36](#_Toc388598687)

[程序 36](#_Toc388598688)

[10. Container With Most Water 39](#_Toc388598689)

[题目 39](#_Toc388598690)

[解法 39](#_Toc388598691)

[分析 39](#_Toc388598692)

[程序 40](#_Toc388598693)

[11. Integer to Roman 40](#_Toc388598694)

[题目 40](#_Toc388598695)

[解法 41](#_Toc388598696)

[分析 41](#_Toc388598697)

[程序 41](#_Toc388598698)

[12. Roman to Integer 42](#_Toc388598699)

[题目 42](#_Toc388598700)

[解法 42](#_Toc388598701)

[分析 42](#_Toc388598702)

[程序 43](#_Toc388598703)

[13. Longest Common Prefix 43](#_Toc388598704)

[题目 43](#_Toc388598705)

[解法 43](#_Toc388598706)

[分析 43](#_Toc388598707)

[程序 43](#_Toc388598708)

[14. 3Sum Closest 44](#_Toc388598709)

[题目 44](#_Toc388598710)

[解法 44](#_Toc388598711)

[分析 44](#_Toc388598712)

[程序 45](#_Toc388598713)

[15. 4Sum 46](#_Toc388598714)

[题目 46](#_Toc388598715)

[解法 46](#_Toc388598716)

[分析 46](#_Toc388598717)

[程序 46](#_Toc388598718)

[16. Letter Combinations of a Phone Number 48](#_Toc388598719)

[题目 48](#_Toc388598720)

[解法 48](#_Toc388598721)

[分析 48](#_Toc388598722)

[程序 48](#_Toc388598723)

[17. Remove Nth Node From End of List 49](#_Toc388598724)

[题目 49](#_Toc388598725)

[解法 50](#_Toc388598726)

[分析 50](#_Toc388598727)

[程序 50](#_Toc388598728)

[18. Valid Parentheses 51](#_Toc388598729)

[题目 51](#_Toc388598730)

[解法 51](#_Toc388598731)

[分析 51](#_Toc388598732)

[程序 52](#_Toc388598733)

[19. Generate Parentheses 53](#_Toc388598734)

[题目 53](#_Toc388598735)

[解法 53](#_Toc388598736)

[分析 53](#_Toc388598737)

[程序 53](#_Toc388598738)

[20. Merge k Sorted Lists 54](#_Toc388598739)

[题目 54](#_Toc388598740)

[解法 54](#_Toc388598741)

[分析 54](#_Toc388598742)

[程序 54](#_Toc388598743)

[21. Swap Nodes in Pairs 57](#_Toc388598744)

[题目 57](#_Toc388598745)

[解法 57](#_Toc388598746)

[分析 57](#_Toc388598747)

[程序 58](#_Toc388598748)

[22. Reverse Nodes in k-Group 59](#_Toc388598749)

[题目 59](#_Toc388598750)

[解法 60](#_Toc388598751)

[分析 60](#_Toc388598752)

[程序 60](#_Toc388598753)

[23. Remove Duplicates from Sorted Array 62](#_Toc388598754)

[题目 62](#_Toc388598755)

[解法 62](#_Toc388598756)

[分析 62](#_Toc388598757)

[程序 62](#_Toc388598758)

[24. Remove Element 63](#_Toc388598759)

[题目 63](#_Toc388598760)

[解法 63](#_Toc388598761)

[分析 63](#_Toc388598762)

[程序 63](#_Toc388598763)

[25. Implement strStr() 64](#_Toc388598764)

[题目 64](#_Toc388598765)

[解法 64](#_Toc388598766)

[分析 64](#_Toc388598767)

[程序 64](#_Toc388598768)

[26. Divide Two Integers 66](#_Toc388598769)

[题目 66](#_Toc388598770)

[解法 66](#_Toc388598771)

[分析 66](#_Toc388598772)

[程序 66](#_Toc388598773)

[27. Substring with Concatenation of All Words 68](#_Toc388598774)

[题目 68](#_Toc388598775)

[解法 68](#_Toc388598776)

[分析 68](#_Toc388598777)

[程序 68](#_Toc388598778)

[28. Next Permutation 70](#_Toc388598779)

[题目 70](#_Toc388598780)

[解法 70](#_Toc388598781)

[分析 70](#_Toc388598782)

[程序 70](#_Toc388598783)

[29. Longest Valid Parentheses 71](#_Toc388598784)

[题目 71](#_Toc388598785)

[解法 71](#_Toc388598786)

[分析 71](#_Toc388598787)

[程序 71](#_Toc388598788)

[30. Search in Rotated Sorted Array 72](#_Toc388598789)

[题目 72](#_Toc388598790)

[解法 72](#_Toc388598791)

[分析 72](#_Toc388598792)

[程序 73](#_Toc388598793)

[31. Search for a Range 74](#_Toc388598794)

[题目 74](#_Toc388598795)

[解法 74](#_Toc388598796)

[分析 74](#_Toc388598797)

[程序 74](#_Toc388598798)

[32. Search Insert Position 76](#_Toc388598799)

[题目 76](#_Toc388598800)

[解法 76](#_Toc388598801)

[分析 76](#_Toc388598802)

[程序 76](#_Toc388598803)

[33. Valid Sudoku 77](#_Toc388598804)

[题目 77](#_Toc388598805)

[解法 77](#_Toc388598806)

[分析 77](#_Toc388598807)

[程序 78](#_Toc388598808)

[34. Sudoku Solver 79](#_Toc388598809)

[题目 79](#_Toc388598810)

[解法 80](#_Toc388598811)

[分析 80](#_Toc388598812)

[程序 80](#_Toc388598813)

[35. Count and Say 83](#_Toc388598814)

[题目 83](#_Toc388598815)

[解法 83](#_Toc388598816)

[分析 83](#_Toc388598817)

[程序 83](#_Toc388598818)

[36. Combination Sum 84](#_Toc388598819)

[题目 84](#_Toc388598820)

[解法 85](#_Toc388598821)

[分析 85](#_Toc388598822)

[程序 85](#_Toc388598823)

[37. Combination Sum II 86](#_Toc388598824)

[题目 86](#_Toc388598825)

[解法 86](#_Toc388598826)

[分析 86](#_Toc388598827)

[程序 86](#_Toc388598828)

[38. First Missing Positive 88](#_Toc388598829)

[题目 88](#_Toc388598830)

[解法 88](#_Toc388598831)

[分析 88](#_Toc388598832)

[程序 88](#_Toc388598833)

[39. Trapping Rain Water 89](#_Toc388598834)

[题目 89](#_Toc388598835)

[解法 90](#_Toc388598836)

[分析 90](#_Toc388598837)

[程序 90](#_Toc388598838)

[40. Multiply Strings 90](#_Toc388598839)

[题目 90](#_Toc388598840)

[解法 91](#_Toc388598841)

[分析 91](#_Toc388598842)

[程序 91](#_Toc388598843)

[41. Jump Game II 92](#_Toc388598844)

[题目 92](#_Toc388598845)

[解法 92](#_Toc388598846)

[分析 92](#_Toc388598847)

[程序 92](#_Toc388598848)

[42. Permutations 93](#_Toc388598849)

[题目 93](#_Toc388598850)

[解法1 93](#_Toc388598851)

[分析 93](#_Toc388598852)

[程序 93](#_Toc388598853)

[解法2 94](#_Toc388598854)

[分析 94](#_Toc388598855)

[程序 94](#_Toc388598856)

[43. Permutations II 95](#_Toc388598857)

[题目 95](#_Toc388598858)

[解法 95](#_Toc388598859)

[分析 95](#_Toc388598860)

[程序 95](#_Toc388598861)

[44. Rotate Image 96](#_Toc388598862)

[题目 96](#_Toc388598863)

[解法 96](#_Toc388598864)

[分析 96](#_Toc388598865)

[程序 96](#_Toc388598866)

[45. Anagrams 96](#_Toc388598867)

[题目 96](#_Toc388598868)

[解法 97](#_Toc388598869)

[分析 97](#_Toc388598870)

[程序 97](#_Toc388598871)

[46. Pow(x, n) 97](#_Toc388598872)

[题目 97](#_Toc388598873)

[解法 97](#_Toc388598874)

[分析 97](#_Toc388598875)

[程序 98](#_Toc388598876)

[47. N-Queens 98](#_Toc388598877)

[题目 98](#_Toc388598878)

[解法 99](#_Toc388598879)

[分析 99](#_Toc388598880)

[程序 99](#_Toc388598881)

[48. N-Queens II 100](#_Toc388598882)

[题目 100](#_Toc388598883)

[解法 100](#_Toc388598884)

[分析 100](#_Toc388598885)

[程序 100](#_Toc388598886)

[49. Maximum Subarray 101](#_Toc388598887)

[题目 101](#_Toc388598888)

[解法 101](#_Toc388598889)

[分析 101](#_Toc388598890)

[程序 102](#_Toc388598891)

[50. Spiral Matrix 102](#_Toc388598892)

[题目 102](#_Toc388598893)

[解法 102](#_Toc388598894)

[分析 102](#_Toc388598895)

[程序 103](#_Toc388598896)

[51. Jump Game 104](#_Toc388598897)

[题目 104](#_Toc388598898)

[解法 104](#_Toc388598899)

[分析 104](#_Toc388598900)

[程序 105](#_Toc388598901)

[52. Merge Intervals 105](#_Toc388598902)

[题目 105](#_Toc388598903)

[解法 105](#_Toc388598904)

[分析 105](#_Toc388598905)

[程序 106](#_Toc388598906)

[53. Insert Interval 106](#_Toc388598907)

[题目 106](#_Toc388598908)

[解法 107](#_Toc388598909)

[分析 107](#_Toc388598910)

[程序 107](#_Toc388598911)

[54. Length of Last Word 109](#_Toc388598912)

[题目 109](#_Toc388598913)

[解法 109](#_Toc388598914)

[分析 109](#_Toc388598915)

[程序 109](#_Toc388598916)

[55. Spiral Matrix II 110](#_Toc388598917)

[题目 110](#_Toc388598918)

[解法 110](#_Toc388598919)

[分析 110](#_Toc388598920)

[程序 110](#_Toc388598921)

[56. Permutation Sequence 112](#_Toc388598922)

[题目 112](#_Toc388598923)

[解法 112](#_Toc388598924)

[分析 112](#_Toc388598925)

[程序 112](#_Toc388598926)

[57. Rotate List 113](#_Toc388598927)

[题目 113](#_Toc388598928)

[解法 114](#_Toc388598929)

[分析 114](#_Toc388598930)

[程序 114](#_Toc388598931)

[58. Unique Paths 115](#_Toc388598932)

[题目 115](#_Toc388598933)

[解法 115](#_Toc388598934)

[分析 115](#_Toc388598935)

[程序 115](#_Toc388598936)

[59. Unique Paths II 116](#_Toc388598937)

[题目 116](#_Toc388598938)

[解法 116](#_Toc388598939)

[分析 116](#_Toc388598940)

[程序 116](#_Toc388598941)

[60. Minimum Path Sum 117](#_Toc388598942)

[题目 117](#_Toc388598943)

[解法 117](#_Toc388598944)

[分析 117](#_Toc388598945)

[程序 117](#_Toc388598946)

[61. Merge Two Sorted Lists 118](#_Toc388598947)

[题目 118](#_Toc388598948)

[解法 118](#_Toc388598949)

[分析 118](#_Toc388598950)

[程序 118](#_Toc388598951)

[62. Add Binary 119](#_Toc388598952)

[题目 119](#_Toc388598953)

[解法 120](#_Toc388598954)

[分析 120](#_Toc388598955)

[程序 120](#_Toc388598956)

[63. Plus One 121](#_Toc388598957)

[题目 121](#_Toc388598958)

[解法 121](#_Toc388598959)

[分析 121](#_Toc388598960)

[程序 121](#_Toc388598961)

[64. Text Justification 122](#_Toc388598962)

[题目 122](#_Toc388598963)

[解法 122](#_Toc388598964)

[分析 122](#_Toc388598965)

[程序 122](#_Toc388598966)

[65. Sqrt(x) 124](#_Toc388598967)

[题目 124](#_Toc388598968)

[解法 125](#_Toc388598969)

[分析 125](#_Toc388598970)

[程序 125](#_Toc388598971)

[66. Climbing Stairs 126](#_Toc388598972)

[题目 126](#_Toc388598973)

[解法 126](#_Toc388598974)

[分析 126](#_Toc388598975)

[程序 127](#_Toc388598976)

[67. Simplify Path 127](#_Toc388598977)

[题目 127](#_Toc388598978)

[解法 127](#_Toc388598979)

[分析 127](#_Toc388598980)

[程序 127](#_Toc388598981)

[68. Edit Distance 129](#_Toc388598982)

[题目 129](#_Toc388598983)

[解法 129](#_Toc388598984)

[分析 129](#_Toc388598985)

[程序 129](#_Toc388598986)

[69. Set Matrix Zeroes 130](#_Toc388598987)

[题目 130](#_Toc388598988)

[解法 130](#_Toc388598989)

[分析 130](#_Toc388598990)

[程序 130](#_Toc388598991)

[70. Search a 2D Matrix 131](#_Toc388598992)

[题目 131](#_Toc388598993)

[解法 132](#_Toc388598994)

[分析 132](#_Toc388598995)

[程序 132](#_Toc388598996)

[71. Sort Colors 133](#_Toc388598997)

[题目 133](#_Toc388598998)

[解法 134](#_Toc388598999)

[分析 134](#_Toc388599000)

[程序 134](#_Toc388599001)

[72. Minimum Window Substring 135](#_Toc388599002)

[题目 135](#_Toc388599003)

[解法 135](#_Toc388599004)

[分析 135](#_Toc388599005)

[程序 135](#_Toc388599006)

[73. Combinations 139](#_Toc388599007)

[题目 139](#_Toc388599008)

[解法 139](#_Toc388599009)

[分析 139](#_Toc388599010)

[程序 139](#_Toc388599011)

[74. Subsets 140](#_Toc388599012)

[题目 140](#_Toc388599013)

[解法 141](#_Toc388599014)

[分析 141](#_Toc388599015)

[程序 141](#_Toc388599016)

[75. Word Search 142](#_Toc388599017)

[题目 142](#_Toc388599018)

[解法 142](#_Toc388599019)

[分析 142](#_Toc388599020)

[程序 142](#_Toc388599021)

[76. Remove Duplicates from Sorted Array II 145](#_Toc388599022)

[题目 145](#_Toc388599023)

[解法 145](#_Toc388599024)

[分析 145](#_Toc388599025)

[程序 145](#_Toc388599026)

[77. Search in Rotated Sorted Array II 146](#_Toc388599027)

[题目 146](#_Toc388599028)

[解法 146](#_Toc388599029)

[分析 146](#_Toc388599030)

[程序 146](#_Toc388599031)

[78. Remove Duplicates from Sorted List 149](#_Toc388599032)

[题目 149](#_Toc388599033)

[解法 149](#_Toc388599034)

[分析 149](#_Toc388599035)

[程序 149](#_Toc388599036)

[79. Remove Duplicates from Sorted List II 150](#_Toc388599037)

[题目 150](#_Toc388599038)

[解法 150](#_Toc388599039)

[分析 150](#_Toc388599040)

[程序 150](#_Toc388599041)

[80. Largest Rectangle in Histogram 152](#_Toc388599042)

[题目 152](#_Toc388599043)

[解法 153](#_Toc388599044)

[分析 153](#_Toc388599045)

[程序 153](#_Toc388599046)

[81. Maximal Rectangle 154](#_Toc388599047)

[题目 154](#_Toc388599048)

[解法 155](#_Toc388599049)

[分析 155](#_Toc388599050)

[程序 155](#_Toc388599051)

[82. Partition List 157](#_Toc388599052)

[题目 157](#_Toc388599053)

[解法 157](#_Toc388599054)

[分析 157](#_Toc388599055)

[程序 157](#_Toc388599056)

[83. Scramble String 159](#_Toc388599057)

[题目 159](#_Toc388599058)

[解法 160](#_Toc388599059)

[分析 160](#_Toc388599060)

[程序 160](#_Toc388599061)

[84. Merge Sorted Array 161](#_Toc388599062)

[题目 161](#_Toc388599063)

[解法 162](#_Toc388599064)

[分析 162](#_Toc388599065)

[程序 162](#_Toc388599066)

[85. Gray Code 163](#_Toc388599067)

[题目 163](#_Toc388599068)

[解法 163](#_Toc388599069)

[分析 163](#_Toc388599070)

[程序 164](#_Toc388599071)

[86. Decode Ways 164](#_Toc388599072)

[题目 164](#_Toc388599073)

[解法 165](#_Toc388599074)

[分析 165](#_Toc388599075)

[程序 165](#_Toc388599076)

[87. Subsets II 166](#_Toc388599077)

[题目 166](#_Toc388599078)

[解法 167](#_Toc388599079)

[分析 167](#_Toc388599080)

[程序 167](#_Toc388599081)

[88. Reverse Linked List II 168](#_Toc388599082)

[题目 168](#_Toc388599083)

[解法 168](#_Toc388599084)

[分析 168](#_Toc388599085)

[程序 168](#_Toc388599086)

[89. Restore IP Addresses 171](#_Toc388599087)

[题目 171](#_Toc388599088)

[解法 171](#_Toc388599089)

[分析 171](#_Toc388599090)

[程序 171](#_Toc388599091)

[90. Binary Tree Inorder Traversal 172](#_Toc388599092)

[题目 172](#_Toc388599093)

[解法 173](#_Toc388599094)

[分析 173](#_Toc388599095)

[程序 173](#_Toc388599096)

[91. Unique Binary Search Trees 174](#_Toc388599097)

[题目 174](#_Toc388599098)

[解法 174](#_Toc388599099)

[分析 174](#_Toc388599100)

[程序 175](#_Toc388599101)

[92. Unique Binary Search Trees II 175](#_Toc388599102)

[题目 175](#_Toc388599103)

[解法 176](#_Toc388599104)

[分析 176](#_Toc388599105)

[程序 176](#_Toc388599106)

[93. Interleaving String 178](#_Toc388599107)

[题目 178](#_Toc388599108)

[解法 178](#_Toc388599109)

[分析 178](#_Toc388599110)

[程序 178](#_Toc388599111)

[94. Validate Binary Search Tree 179](#_Toc388599112)

[题目 179](#_Toc388599113)

[解法 180](#_Toc388599114)

[分析 180](#_Toc388599115)

[程序 180](#_Toc388599116)

[95. Recover Binary Search Tree 181](#_Toc388599117)

[题目 181](#_Toc388599118)

[解法 181](#_Toc388599119)

[分析 181](#_Toc388599120)

[程序 181](#_Toc388599121)

[96. Same Tree 182](#_Toc388599122)

[题目 182](#_Toc388599123)

[解法 183](#_Toc388599124)

[分析 183](#_Toc388599125)

[程序 183](#_Toc388599126)

[97. Symmetric Tree 183](#_Toc388599127)

[题目 183](#_Toc388599128)

[解法 184](#_Toc388599129)

[分析 184](#_Toc388599130)

[程序 184](#_Toc388599131)

[98. Binary Tree Level Order Traversal 185](#_Toc388599132)

[题目 185](#_Toc388599133)

[解法 185](#_Toc388599134)

[分析 185](#_Toc388599135)

[程序 185](#_Toc388599136)

[99. Binary Tree Zigzag Level Order Traversal 187](#_Toc388599137)

[题目 187](#_Toc388599138)

[解法 188](#_Toc388599139)

[分析 188](#_Toc388599140)

[程序 188](#_Toc388599141)

[100. Maximum Depth of Binary Tree 189](#_Toc388599142)

[题目 189](#_Toc388599143)

[解法 189](#_Toc388599144)

[分析 189](#_Toc388599145)

[程序 190](#_Toc388599146)

[101. Construct Binary Tree from Preorder and Inorder Traversal 190](#_Toc388599147)

[题目 190](#_Toc388599148)

[解法 191](#_Toc388599149)

[分析 191](#_Toc388599150)

[程序 191](#_Toc388599151)

[102. Construct Binary Tree from Inorder and Postorder Traversal 192](#_Toc388599152)

[题目 192](#_Toc388599153)

[解法 192](#_Toc388599154)

[分析 192](#_Toc388599155)

[程序 193](#_Toc388599156)

[103. Binary Tree Level Order Traversal II 194](#_Toc388599157)

[题目 194](#_Toc388599158)

[解法 194](#_Toc388599159)

[分析 194](#_Toc388599160)

[程序 195](#_Toc388599161)

[104. Convert Sorted Array to Binary Search Tree 196](#_Toc388599162)

[题目 196](#_Toc388599163)

[解法 196](#_Toc388599164)

[分析 196](#_Toc388599165)

[程序 196](#_Toc388599166)

[105. Convert Sorted List to Binary Search Tree 197](#_Toc388599167)

[题目 197](#_Toc388599168)

[解法 197](#_Toc388599169)

[分析 197](#_Toc388599170)

[程序 197](#_Toc388599171)

[106. Balanced Binary Tree 199](#_Toc388599172)

[题目 199](#_Toc388599173)

[解法 199](#_Toc388599174)

[分析 199](#_Toc388599175)

[程序 199](#_Toc388599176)

[107. Minimum Depth of Binary Tree 201](#_Toc388599177)

[题目 201](#_Toc388599178)

[解法 201](#_Toc388599179)

[分析 201](#_Toc388599180)

[程序 201](#_Toc388599181)

[108. Path Sum 202](#_Toc388599182)

[题目 202](#_Toc388599183)

[解法 202](#_Toc388599184)

[分析 202](#_Toc388599185)

[程序 202](#_Toc388599186)

[109. Path Sum II 203](#_Toc388599187)

[题目 203](#_Toc388599188)

[解法 203](#_Toc388599189)

[分析 203](#_Toc388599190)

[程序 204](#_Toc388599191)

[110. Flatten Binary Tree to Linked List 205](#_Toc388599192)

[题目 205](#_Toc388599193)

[解法 206](#_Toc388599194)

[分析 206](#_Toc388599195)

[程序 206](#_Toc388599196)

[111. Distinct Subsequences 207](#_Toc388599197)

[题目 207](#_Toc388599198)

[解法 207](#_Toc388599199)

[分析 207](#_Toc388599200)

[程序 207](#_Toc388599201)

[112. Populating Next Right Pointers in Each Node 209](#_Toc388599202)

[题目 209](#_Toc388599203)

[解法 210](#_Toc388599204)

[分析 210](#_Toc388599205)

[程序 210](#_Toc388599206)

[113. Populating Next Right Pointers in Each Node II 211](#_Toc388599207)

[题目 211](#_Toc388599208)

[解法 212](#_Toc388599209)

[分析 212](#_Toc388599210)

[程序 212](#_Toc388599211)

[114. Pascal's Triangle 213](#_Toc388599212)

[题目 213](#_Toc388599213)

[解法 214](#_Toc388599214)

[分析 214](#_Toc388599215)

[程序 214](#_Toc388599216)

[115. Pascal's Triangle II 215](#_Toc388599217)

[题目 215](#_Toc388599218)

[解法 215](#_Toc388599219)

[分析 215](#_Toc388599220)

[程序 215](#_Toc388599221)

[116. Triangle 216](#_Toc388599222)

[题目 216](#_Toc388599223)

[解法 216](#_Toc388599224)

[分析 216](#_Toc388599225)

[程序 217](#_Toc388599226)

[117. Best Time to Buy and Sell Stock 218](#_Toc388599227)

[题目 218](#_Toc388599228)

[解法 218](#_Toc388599229)

[分析 218](#_Toc388599230)

[程序 218](#_Toc388599231)

[118. Best Time to Buy and Sell Stock II 219](#_Toc388599232)

[题目 219](#_Toc388599233)

[解法 219](#_Toc388599234)

[分析 219](#_Toc388599235)

[程序 219](#_Toc388599236)

[119. Best Time to Buy and Sell Stock III 220](#_Toc388599237)

[题目 220](#_Toc388599238)

[解法 220](#_Toc388599239)

[分析 220](#_Toc388599240)

[程序 220](#_Toc388599241)

[120. Binary Tree Maximum Path Sum 222](#_Toc388599242)

[题目 222](#_Toc388599243)

[解法 222](#_Toc388599244)

[分析 222](#_Toc388599245)

[程序 222](#_Toc388599246)

[121. Valid Palindrome 225](#_Toc388599247)

[题目 225](#_Toc388599248)

[解法 225](#_Toc388599249)

[分析 225](#_Toc388599250)

[程序 225](#_Toc388599251)

[122. Word Ladder 226](#_Toc388599252)

[题目 226](#_Toc388599253)

[解法 227](#_Toc388599254)

[分析 227](#_Toc388599255)

[程序 227](#_Toc388599256)

[123. Word Ladder II 228](#_Toc388599257)

[题目 228](#_Toc388599258)

[解法 229](#_Toc388599259)

[分析 229](#_Toc388599260)

[程序 229](#_Toc388599261)

[124. Longest Consecutive Sequence 231](#_Toc388599262)

[题目 231](#_Toc388599263)

[解法 231](#_Toc388599264)

[分析 231](#_Toc388599265)

[程序 232](#_Toc388599266)

[125. Sum Root to Leaf Numbers 235](#_Toc388599267)

[题目 235](#_Toc388599268)

[解法 235](#_Toc388599269)

[分析 235](#_Toc388599270)

[程序 235](#_Toc388599271)

[126. Surrounded Regions 237](#_Toc388599272)

[题目 237](#_Toc388599273)

[解法 237](#_Toc388599274)

[分析 237](#_Toc388599275)

[程序 237](#_Toc388599276)

[127. Palindrome Partitioning 240](#_Toc388599277)

[题目 240](#_Toc388599278)

[解法 241](#_Toc388599279)

[分析 241](#_Toc388599280)

[程序 241](#_Toc388599281)

[128. Clone Graph 242](#_Toc388599282)

[题目 242](#_Toc388599283)

[解法 243](#_Toc388599284)

[分析 243](#_Toc388599285)

[程序 243](#_Toc388599286)

[129. Gas Station 244](#_Toc388599287)

[题目 244](#_Toc388599288)

[解法 245](#_Toc388599289)

[分析 245](#_Toc388599290)

[程序 245](#_Toc388599291)

[130. Candy 246](#_Toc388599292)

[题目 246](#_Toc388599293)

[解法 246](#_Toc388599294)

[分析 246](#_Toc388599295)

[程序 247](#_Toc388599296)

[131. Single Number 249](#_Toc388599297)

[题目 249](#_Toc388599298)

[解法 249](#_Toc388599299)

[分析 249](#_Toc388599300)

[程序 249](#_Toc388599301)

[132. Single Number II 250](#_Toc388599302)

[题目 250](#_Toc388599303)

[解法 250](#_Toc388599304)

[分析 250](#_Toc388599305)

[程序 250](#_Toc388599306)

[133. Copy List with Random Pointer 251](#_Toc388599307)

[题目 251](#_Toc388599308)

[解法 251](#_Toc388599309)

[分析 251](#_Toc388599310)

[程序 251](#_Toc388599311)

[134. Word Break 253](#_Toc388599312)

[题目 253](#_Toc388599313)

[解法 253](#_Toc388599314)

[分析 253](#_Toc388599315)

[程序 253](#_Toc388599316)

[135. Word Break II 254](#_Toc388599317)

[题目 254](#_Toc388599318)

[解法 254](#_Toc388599319)

[分析 254](#_Toc388599320)

[程序 254](#_Toc388599321)

[136. Linked List Cycle 257](#_Toc388599322)

[题目 257](#_Toc388599323)

[解法 258](#_Toc388599324)

[分析 258](#_Toc388599325)

[程序 258](#_Toc388599326)

[137. Linked List Cycle II 259](#_Toc388599327)

[题目 259](#_Toc388599328)

[解法 259](#_Toc388599329)

[分析 259](#_Toc388599330)

[程序 259](#_Toc388599331)

[138. Reorder List 260](#_Toc388599332)

[题目 260](#_Toc388599333)

[解法 260](#_Toc388599334)

[分析 260](#_Toc388599335)

[程序 260](#_Toc388599336)

[139. Binary Tree Preorder Traversal 262](#_Toc388599337)

[题目 262](#_Toc388599338)

[解法 263](#_Toc388599339)

[分析 263](#_Toc388599340)

[程序 263](#_Toc388599341)

[140. Binary Tree Postorder Traversal 265](#_Toc388599342)

[题目 265](#_Toc388599343)

[解法 265](#_Toc388599344)

[分析 265](#_Toc388599345)

[程序 265](#_Toc388599346)

[141. LRU Cache 267](#_Toc388599347)

[题目 267](#_Toc388599348)

[解法 268](#_Toc388599349)

[分析 268](#_Toc388599350)

[程序 268](#_Toc388599351)

[142. Insertion Sort List 273](#_Toc388599352)

[题目 273](#_Toc388599353)

[解法 273](#_Toc388599354)

[分析 273](#_Toc388599355)

[程序 273](#_Toc388599356)

[143. Sort List 275](#_Toc388599357)

[题目 275](#_Toc388599358)

[解法 275](#_Toc388599359)

[分析 275](#_Toc388599360)

[程序 275](#_Toc388599361)

[144. Evaluate Reverse Polish Notation 278](#_Toc388599362)

[题目 278](#_Toc388599363)

[解法 278](#_Toc388599364)

[分析 278](#_Toc388599365)

[程序 278](#_Toc388599366)

[附录 未完成题目 280](#_Toc388599367)

[145. ZigZag Conversion 280](#_Toc388599368)

[题目 280](#_Toc388599369)

[解法 280](#_Toc388599370)

[分析 280](#_Toc388599371)

[程序 280](#_Toc388599372)

[146. 3Sum 281](#_Toc388599373)

[题目 281](#_Toc388599374)

[解法 281](#_Toc388599375)

[分析 281](#_Toc388599376)

[程序 281](#_Toc388599377)

[147. Wildcard Matching 281](#_Toc388599378)

[题目 281](#_Toc388599379)

[解法 282](#_Toc388599380)

[分析 282](#_Toc388599381)

[程序 282](#_Toc388599382)

[148. Valid Number 282](#_Toc388599383)

[题目 282](#_Toc388599384)

[解法 282](#_Toc388599385)

[分析 282](#_Toc388599386)

[程序 282](#_Toc388599387)

[149. Palindrome Partitioning II 282](#_Toc388599388)

[题目 282](#_Toc388599389)

[解法 283](#_Toc388599390)

[分析 283](#_Toc388599391)

[程序 283](#_Toc388599392)

[150. Max Points on a Line 283](#_Toc388599393)

[题目 283](#_Toc388599394)

[解法 283](#_Toc388599395)

[分析 283](#_Toc388599396)

[程序 283](#_Toc388599397)

[151. Reverse Words in a String 283](#_Toc388599398)

[题目 283](#_Toc388599399)

[解法 284](#_Toc388599400)

[分析 284](#_Toc388599401)

[程序 284](#_Toc388599402)

# Two Sum

## 题目

Given an array of integers, find two numbers such that they add up to a specific target number.

The function twoSum should return indices of the two numbers such that they add up to the target, where index1 must be less than index2. Please note that your returned answers (both index1 and index2) are not zero-based.

You may assume that each input would have exactly one solution.

Input: numbers={2, 7, 11, 15}, target=9

Output: index1=1, index2=2

## 解法

### 分析

针对所有元素，建立hash表，对每个元素，可以算出得到结果需要的“补数”。利用hash表，查找这个补数在原集合中是否存在。

需要注意的是，补数可能和元素相同，这时候不能把元素数据存在当成结果。需要判断是否有两个这样的值在集合中存在。

### 程序

class Solution {

public:

    vector<int> twoSum(vector<int> &numbers, int target) {

        vector<int> ret;

        if(numbers.size()<2)return ret;

        unordered\_multimap<int, int> map;

        for(size\_t i=0;i<numbers.size();i++){

            map.insert(pair<int,int>(numbers[i],i));

        }

        for(int i=0;i<(int)numbers.size();i++){

            int j = target - numbers[i];

            if(j!=numbers[i]){

                auto it=map.find(j);

                if(it!=map.end()){

                    int k=it->second;

                    if(i<k){

                        ret.push\_back(i+1);

                        ret.push\_back(k+1);

                        return ret;

                    }

                    else{

                        ret.push\_back(k+1);

                        ret.push\_back(i+1);

                        return ret;

                    }

                }

            }

            else

            {

                vector<int> tmp;

                auto its = map.equal\_range(j);

                for (auto it = its.first; it != its.second; ++it) {

                    tmp.push\_back(it->second);

                }

                if(tmp.size()<2)continue;

                if(tmp[0]<tmp[1]){

                    ret.push\_back(tmp[0]+1);

                    ret.push\_back(tmp[1]+1);

                    return ret;

                }

                else{

                    ret.push\_back(tmp[1]+1);

                    ret.push\_back(tmp[0]+1);

                }

            }

        }

        return ret;

    }

};

# Median of Two Sorted Arrays

## 题目

There are two sorted arrays A and B of size m and n respectively. Find the median of the two sorted arrays. The overall run time complexity should be O(log (m+n)).

## 解法1

### 分析

采用笨办法，逐个搜索，其实也只需O(n)的时间。

### 程序

class Solution {

public:

int findK(int A[], int m, int B[], int n, int k){

int a,b,i;

a=0;

b=0;

i=0;

for(i=0;i<k;i++){

if(a>=m)

return B[k-m];

if(b>=n)

return A[k-n];

if(A[a]<B[b])

a++;

else

b++;

}

if(a>=m)

return B[k-m];

if(b>=n)

return A[k-n];

if(A[a]<B[b])

return A[a];

//else

return B[b];

}

double findMedianSortedArrays(int A[], int m, int B[], int n) {

double ret;

int k = m+n;

if(k&0x1){

ret = findK(A,m,B,n,k/2);

}

else{

double a,b;

a = findK(A,m,B,n,k/2-1);

b = findK(A,m,B,n,k/2);

ret = (a+b)/2;

}

return ret;

}

};

## 解法2

### 分析

待续

# Longest Substring Without Repeating Characters

## 题目

Given a string, find the length of the longest substring without repeating characters. For example, the longest substring without repeating letters for "abcabcbb" is "abc", which the length is 3. For "bbbbb" the longest substring is "b", with the length of 1.

## 解法

### 分析

 只有小写字母，可以建立一个26个字母的数组，每个字母记录它最近出现的位置。

…hijabcdefga…

一个字母和它自身上一个字母出现的位置之间，这个字母不会出现重复。但是其他字母有可能重复。处理这种情况，可以在其他字母出现重复时，把所有字母的最近位置更新一遍。如果有字母的最近位置早于当前发现的重复字母，也需要把它们的最近位置改为当前字母的前一次出现。

例如，上面的字符串，a字母第二次出现时，bcdefg的最近出现可以不用变化，但是hij的最近出现需要更新为第一个a的位置。

扫描一遍整个字符串，每个字符都更新一下26个（最多）字母的最近位置表。每个字母都和上一次最近出现比较，即可得出当前最长无重复字母的串长度。记录其中最长的即可。

这个算法的复杂度是O(n)的。

### 程序

class Solution {

public:

int lengthOfLongestSubstring(string s) {

int maxLen = 0;

int recentPostion[26];

for(char c='a';c<='z';c++){

recentPostion[c-'a']=-1;

}

for(int i=0;i<(int)s.length();i++){

if(recentPostion[s.at(i)-'a']==-1){

maxLen=i+1;

recentPostion[s.at(i)-'a']=i;

continue;

}

//!= -1

if((i-recentPostion[s.at(i)-'a'])>maxLen){

maxLen = i-recentPostion[s.at(i)-'a'];

}

for(char c='a';c<='z';c++){

if(recentPostion[c-'a']<recentPostion[s.at(i)-'a']){

recentPostion[c-'a']=recentPostion[s.at(i)-'a'];

}

}

recentPostion[s.at(i)-'a'] = i;

}

return maxLen;

}

};

# Add Two Numbers

## 题目

You are given two linked lists representing two non-negative numbers. The digits are stored in reverse order and each of their nodes contain a single digit. Add the two numbers and return it as a linked list.

Input: (2 -> 4 -> 3) + (5 -> 6 -> 4)

Output: 7 -> 0 -> 8

## 解法

### 分析

 题目很简单，不解释了。

### 程序

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

class Solution {

public:

ListNode \*addTwoNumbers(ListNode \*l1, ListNode \*l2) {

ListNode\* head=NULL;

ListNode\* tail=NULL;

int carrySet = 0;

int val;

while((l1!=NULL)||(l2!=NULL)){

ListNode\* tmp = new ListNode(0);

if(head==NULL) {

head=tmp;

tail=tmp;

}

else{

tail->next = tmp;

tail = tmp;

}

if(l1==NULL)

{

val = carrySet + l2->val;

carrySet = val / 10;

val = val - carrySet\*10;

tmp->val = val;

l2 = l2->next;

}

else if(l2==NULL){

val = carrySet + l1->val;

carrySet = val / 10;

val = val - carrySet\*10;

tmp->val = val;

l1 = l1->next;

}

else{

val = carrySet + l1->val + l2->val;

carrySet = val / 10;

val = val - carrySet\*10;

tmp->val = val;

l1 = l1->next;

l2 = l2->next;

}

}

if(carrySet!=0){

ListNode\* tmp = new ListNode(carrySet);

{

tail->next = tmp;

tail = tmp;

}

}

return head;

}

};

# Longest Palindromic Substring

## 题目

Given a string S, find the longest palindromic substring in S. You may assume that the maximum length of S is 1000, and there exists one unique longest palindromic substring.

## 解法

### 分析

 寻找字符串中最长回文串。这个问题最常见的解法是后缀树，加树中任意两个节点的最远公共祖先查询（LCA），按照算法书的分析，可以达到O(n)。但是这个解法编程过于复杂，很难实现。

查询网络获得如下算法 (Manacher’s algorithm)，也可以达到O(n)的复杂度，而且编程简单。很巧妙。

下面解释拷贝网络。





### 程序

class Solution {

public:

void PreString(string &s, string &t){

int i;

t.at(0)='^';

for(i=1;i<=(int)s.length();i++){

t.at(i\*2-1)='#';

t.at(i\*2)=s.at(i-1);

}

t.at(i\*2-1)='#';

t.at(i\*2)='$';

}

string longestPalindrome(string s) {

int length = 3+s.length()\*2;

int p[length];

string t(length,' ');

PreString(s,t);

int c,r;

int max\_pos=0;

p[0] = 0;

c=r=0;

for(int i = 1; i<length-1; i++){

int i\_mirror = (c<<1) - i;

p[i]=(r>i)?min(r-i,p[i\_mirror]):0;

while(t[i+1+p[i]]==t[i-1-p[i]])

p[i]++;

if((i+p[i])>r){

c=i;

r=c+p[i];

}

if(p[i]>p[max\_pos]){

max\_pos = i;

}

}

int b;

b=max\_pos-1-p[max\_pos];

b/=2;

return s.substr(b,p[max\_pos]);

}

};

# ZigZag Conversion

## 题目

. The string "PAYPALISHIRING" is written in a zigzag pattern on a given number of rows like this: (you may want to display this pattern in a fixed font for better legibility)

P A H N

A P L S I I G

Y I R

And then read line by line: "PAHNAPLSIIGYIR"

Write the code that will take a string and make this conversion given a number of rows:

string convert(string text, int nRows);

convert("PAYPALISHIRING", 3) should return "PAHNAPLSIIGYIR"..

## 解法

### 分析

基本上是一道不太复杂的数学题

### 程序

.JAVA

**public** **class** Solution {

**public** String convert(String s, **int** nRows) {

**if**(nRows==1)

**return** s;

StringBuffer buf = **new** StringBuffer();

**int** n = 2\*nRows - 2;

**for**(**int** i=0;i<nRows;i++){

**int** j=i;

**int** k=n-j;

**while**(**true**){

**boolean** q1,q2;

q1=q2=**false**;

**if**(j<s.length()){

buf.append(s.charAt(j));

j+=n;

}

**else**

q1=**true**;

**if**(k!=n&&i!=k&&k<s.length()){

buf.append(s.charAt(k));

k+=n;

}

**else**

q2=**true**;

**if**(q1&&q2)

**break**;

}

}

**return** buf.toString();

}

}

# Reverse Integer

## 题目

Reverse digits of an integer.

Example1: x = 123, return 321

Example2: x = -123, return -321

## 解法

### 分析

 简单算法也可以pass。复杂一点的，可以增加内存开销要求，不能使用额外内存，原地反转。这里没有深入考虑。

### 程序

class Solution {

public:

int reverse(int x) {

int ret=0;

int sign;

if(x==0)return 0;

if(x<0){

x = -1\*x;

sign = -1;

}

else{

sign = 1;

}

char str[128];

int i;

for(i=0;(x!=0)&&(i<128);i++){

str[i] = x % 10;

x = x/10;

}

for(int j=0;j<i;j++){

ret = ret\*10+str[j];

}

return ret\*sign;

}

};

# String to Integer (atoi)

## 题目

Implement atoi to convert a string to an integer.

Hint: Carefully consider all possible input cases. If you want a challenge, please do not see below and ask yourself what are the possible input cases.

Notes: It is intended for this problem to be specified vaguely (ie, no given input specs). You are responsible to gather all the input requirements up front.

## 解法

### 分析

 算法上没有难度。只是要考虑各种输入可能。

### 程序

class Solution {

public:

int atoi(const char \*str) {

int i=0;

int signed\_flag=0;

for(;\*str!='\0';str++){

if(\*str==' ')

{

if(signed\_flag!=0)

return i;

else

continue;

}

if(\*str=='+'){

if(signed\_flag!=0)

return 0;

signed\_flag = 1;

continue;

}

if(\*str=='-'){

if(signed\_flag!=0)

return 0;

signed\_flag = -1;

continue;

}

int val=\*str - '0';

if(val<=9&&val>=0){

if(signed\_flag==0)

signed\_flag = 1;

if(signed\_flag==1){

int j=i\*10+val;

if(i>0&&j/i<10)

return 2147483647;

else

i=j;

}

else if(signed\_flag==-1){

int j=i\*10-val;

if(i<0&&j/i<10)

return -2147483648;

else

i=j;

}

}

else{

return i;

}

}

return i;

}

};

# Palindrome Number

## 题目

Determine whether an integer is a palindrome. Do this without extra space.

## 解法

### 分析

 唯一难度是不准使用额外空间，不过用一点也没有关系，cases无法发现。

### 程序

class Solution {

public:

bool isPalindrome(int x) {

char content[13];

snprintf(content, 12, "%d", x);

int r = strlen(content)-1;

int l = 0;

while(l<r){

if(content[l]!=content[r])

return false;

l++;r--;

}

return true;

}

};

# Regular Expression Matching

## 题目

Implement regular expression matching with support for '.' and '\*'.

'.' Matches any single character.

'\*' Matches zero or more of the preceding element.

The matching should cover the entire input string (not partial).

The function prototype should be:

bool isMatch(const char \*s, const char \*p)

Some examples:

isMatch("aa","a") → false

isMatch("aa","aa") → true

isMatch("aaa","aa") → false

isMatch("aa", "a\*") → true

isMatch("aa", ".\*") → true

isMatch("ab", ".\*") → true

isMatch("aab", "c\*a\*b") → true

## 解法

### 分析

 需要注意的一点是，\*匹配的是前一个字符重复任意次（包括0次），也就是正则表达式通常用法。而不是在DOS路径命令中的用法。

引申一点，”.\*”匹配”.”重复任意次。因此，”.\*”不仅仅能够匹配”aaaa”, ”bbbbb”, “”, 也能够匹配”abcde”。

这道题可以用递归方法做，但是重复计算量很大，其实，它是典型的动态规划应用。

定义d[i][j]为s[0..i-1], p[0..j]是否匹配。原串需要为空串定义值，模式串则不需要。（为了应对d与a\*b\*c\*d的判断）。

最后的结果是d[s.length()][p.length()-1].

### 程序

class Solution {

public:

struct pattern\_t{

char ch;

int star; //0, without \*; 1 with \*

pattern\_t(char in\_ch, int in\_star=0):

ch(in\_ch),star(in\_star){

}

};

bool chMatch(const char s, const char p){

if((s==p)||(p=='.'))

return true;

else

return false;

}

bool isMatch(const char \*s, const char \*p) {

if(\*s=='\0'&&\*p=='\0')

return true;

if(\*s!='\0'&&\*p=='\0')

return false;

string source(s);

vector<pattern\_t> pattern;

pattern.clear();

char \*l\_p = (char\*)p;

while(\*l\_p!='\0'){

if(\*(l\_p+1)=='\*'){

pattern.push\_back(pattern\_t(\*l\_p,1));

l\_p += 2;

}

else{

pattern.push\_back(pattern\_t(\*l\_p));

l\_p += 1;

}

}

int width = source.length();

int heigh = pattern.size();

bool result[width+1][heigh+1];

if(pattern[0].star==1)

result[0][0]=true;

else

result[0][0]=false;

for(int j=1;j<heigh;j++){

if(result[0][j-1]&&pattern[j].star==1)

result[0][j]=true;

else

result[0][j]=false;

}

if(chMatch(source[0],pattern[0].ch))

{

result[1][0]=true;

}

else

{

result[1][0]=false;

}

for(int i=2; i<(width+1); i++ )

{

if(pattern[0].star==0)

{

result[i][0]=false;

}

else {//\*

if(result[i-1][0]&&chMatch(source[i-1],pattern[0].ch))

result[i][0]=true;

else

result[i][0]=false;

}

}

for(int j=1;j<heigh;j++){

for(int i=1;i<width+1;i++){

//source[i-1]

//pattern[j]

if(pattern[j].star==0)

{

if(result[i-1][j-1]&&chMatch(source[i-1],pattern[j].ch))

result[i][j]=true;

else

result[i][j]=false;

}

else

{

if(result[i][j-1])

result[i][j]=true;

else if(result[i-1][j-1]&&chMatch(source[i-1],pattern[j].ch))

result[i][j]=true;

else if(result[i-1][j]&&chMatch(source[i-1],pattern[j].ch))

result[i][j]=true;

else

result[i][j]=false;

}

}

}

return result[width][heigh-1];

}

};

# Container With Most Water

## 题目

Given n non-negative integers a1, a2, ..., an, where each represents a point at coordinate (i, ai). n vertical lines are drawn such that the two endpoints of line i is at (i, ai) and (i, 0). Find two lines, which together with x-axis forms a container, such that the container contains the most water.

## 解法

### 分析

 题目的解法很简单，但是需要一点脑筋来证明这个方法是对的。

用两个指针，从数组两头开始往中间推，每次移动其中一个一个步长。移动的，是两个中高度较低的一个。

这样复杂度只有O(n)，但它为什么是对的呢？

这道题，最简单的方法，是把N^2/2种组合全部检查一遍。但是，实际上不需要逐个检查，有很多是可以跳过的。下面的分析介绍了为什么这个方法中跳过的那些选项不可能是答案。

分析如下

Here is a simple proof for the solution. Use v[low, high] indicates the volume of container with low and high. suppose height[low] < height[high], then we move low to low+1, that means we ingored v[low, high-1],v[low, high-2],etc, if this is safe, then the algorithm is right, and it's obvious that v[low, high-1],high[low, high-2]...... can't be larger than v[low, high] since its width can't be larger than high-low, and its height is limited by height[low].

因此，最大的一种组合，必定出现在我们检查过的集合中。

### 程序

class Solution {

public:

int max(int a, int b){

if(a>b)return a;

return b;

}

int min(int a, int b){

if(a>b)return b;

return a;

}

int maxArea(vector<int> &height) {

int left = 0;

int right = height.size()-1;

int maxSize=0;

while(left<right){

maxSize =

max(maxSize,min(height[right],height[left])\*(right-left));

if(height[right]>height[left]){

left++;

}

else

{

right--;

}

}

return maxSize;

}

};

# Integer to Roman

## 题目

Given an integer, convert it to a roman numeral.

Input is guaranteed to be within the range from 1 to 3999.

## 解法

### 分析

 这是一道编程题，没有算法难点。

### 程序

const static char GeWei[4][10][5] =

{

{

"",

"M",

"MM",

"MMM"

},

{

"",

"C",

"CC",

"CCC",

"CD",

"D",

"DC",

"DCC",

"DCCC",

"CM"

},

{

"",

"X",

"XX",

"XXX",

"XL",

"L",

"LX",

"LXX",

"LXXX",

"XC"

},

{

"",

"I",

"II",

"III",

"IV",

"V",

"VI",

"VII",

"VIII",

"IX"

}

};

class Solution {

public:

string intToRoman(int num) {

string ret;

char str[5];

snprintf(str,5,"%04d",num);

for(int i=0;i<4;i++){

ret += GeWei[i][str[i]-'0'];

}

return ret;

}

};

# Roman to Integer

## 题目

Given a roman numeral, convert it to an integer.

Input is guaranteed to be within the range from 1 to 3999.

## 解法

### 分析

 编程题。下面的程序是网上看到的，挺巧妙的。

### 程序

class Solution {

public:

int romanToInt(string s) {

int ret=0;

for(int i=0;i<(int)s.size();i++){

if (s[i]=='I') ret += 1;

else if(s[i]=='V') ret += 5;

else if(s[i]=='X') ret += 10;

else if(s[i]=='L') ret += 50;

else if(s[i]=='C') ret += 100;

else if(s[i]=='D') ret += 500;

else if(s[i]=='M') ret += 1000;

}

if(s.find("IV")!=s.npos)ret -= 2;

if(s.find("IX")!=s.npos)ret -= 2;

if(s.find("XL")!=s.npos)ret -= 20;

if(s.find("XC")!=s.npos)ret -= 20;

if(s.find("CD")!=s.npos)ret -= 200;

if(s.find("CM")!=s.npos)ret -= 200;

return ret;

}

};

# Longest Common Prefix

## 题目

Write a function to find the longest common prefix string amongst an array of strings.

## 解法

### 分析

 很简单，不解释。不知道是否有什么神奇方法。

### 程序

class Solution {

public:

string longestCommonPrefix(vector<string> &strs) {

string ret;

int num=strs.size();

if(num==0)return ret;

for(int i=0;;i++){

int j;

for(j=0;j<num;j++){

if((int)strs[j].length()<(i+1)||strs[j][i]!=strs[0][i])

break;

}

if(j!=num)

break;

ret += strs[0][i];

}

return ret;

}

};

# 3Sum

## 题目

. Given an array S of n integers, are there elements a, b, c in S such that a + b + c = 0? Find all unique triplets in the array which gives the sum of zero.

Note:

Elements in a triplet (a,b,c) must be in non-descending order. (ie, a ≤ b ≤ c)

The solution set must not contain duplicate triplets.

For example, given array S = {-1 0 1 2 -1 -4},

A solution set is:

(-1, 0, 1)

(-1, -1, 2).

## 解法

### 分析

可以找到O(n^2)的算法。

先把数组排序。从头到尾逐一枚举每个元素。

对于每个枚举的元素a，从剩下未访问过的元素里，寻找和为-a的两个元素。可以从剩下的所有元素两端开始，向中间靠近。首尾两个元素如果和大于-a，尾向前移动。如果小于，首向后移动。

注意防重。方法是和刚刚找到的比较，看是否相同。为什么可行？可以从道理上说明，这里不详细解释。

### 程序

**JAVA**

**public** **class** Solution {

**public** ArrayList<ArrayList<Integer>> threeSum(**int**[] num) {

// sort array

Arrays.*sort*(num);

ArrayList<ArrayList<Integer>> res = **new** ArrayList<ArrayList<Integer>>();

**for** (**int** i = 0; i < num.length - 2; i++) {

**if**(i>0&&num[i]==num[i-1])

**continue**;

**int** left = i+1;

**int** right = num.length-1;

**while**(left<right){

**int** tmp = num[left] + num[right] + num[i];

**if**(tmp==0){

//check if same with last

**int** index = res.size()-1;

**if**(res.size()>0

&&res.get(index).get(0)==num[i]

&&res.get(index).get(1)==num[left]

&&res.get(index).get(2)==num[right]

){

}

**else**{

ArrayList<Integer> each = **new** ArrayList<Integer>();

each.add(num[i]);

each.add(num[left]);

each.add(num[right]);

res.add(each);

}

left++;

right--;

}

**else** **if**(tmp<0)

left ++;

**else**

right --;

}

}

**return** res;

}

}

# 3Sum Closest

## 题目

Given an array S of n integers, find three integers in S such that the sum is closest to a given number, target. Return the sum of the three integers. You may assume that each input would have exactly one solution.

For example, given array S = {-1 2 1 -4}, and target = 1.

The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).

## 解法

### 分析

 如果是一位数组，寻找任意两个元素之和与指定值的差距最小值，可以先将数组排序，然后从两头往中间夹逼。如果和偏大，就将右端向左移。如果和偏小，就将左端向右移。差距最小者，必定在此过程中出现。复杂度为O(N)。

对于这个题目，可以从左端开始逐个枚举数组中一个元素，然后在右半侧剩余数组中，用上述方法寻找指定值差距最小者（target-枚举值）。复杂度为O(N^2)。

### 程序

class Solution {

public:

int mabs(int a, int b, int c, int t){

int r=a+b+c-t;

if(r<0)r=0-r;

return r;

}

int threeSumClosest(vector<int> &num, int target) {

sort(num.begin(),num.end());

int n=num.size();

int minGap=mabs(num[0],num[1],num[n-1],target);

int ret=num[0]+num[1]+num[n-1];

for(int i=0;i<n-2;i++){

int l=i+1;

int r=n-1;

while(l<r){

int tmp = mabs(num[i],num[l],num[r],target);

if(tmp<minGap){

ret=num[i]+num[l]+num[r];

minGap=tmp;

}

if(minGap==0)

return target;

else if((num[i]+num[l]+num[r])>target)

r--;

else //if((num[i]+num[l]+num[r])<target)

l++;

}

}

return ret;

}

};

# 4Sum

## 题目

Given an array S of n integers, are there elements a, b, c, and d in S such that a + b + c + d = target? Find all unique quadruplets in the array which gives the sum of target.

Note:

Elements in a quadruplet (a,b,c,d) must be in non-descending order. (ie, a ≤ b ≤ c ≤ d)

The solution set must not contain duplicate quadruplets.

For example, given array S = {1 0 -1 0 -2 2}, and target = 0.

A solution set is:

(-1, 0, 0, 1)

(-2, -1, 1, 2)

(-2, 0, 0, 2)

## 解法

### 分析

 建立一个map，将数组中任意两个元素的和作为key，两个元素的序号作为内容。每个key可能对应多个元素对，因此这个map的value是一个vector。

枚举map中的每个key值，每个值计算补值（target-枚举key值）并作为另一个key值。这样可以获得两个vector，其中的元素逐个配对（笛卡尔积）。每个都需要检查是否有重复元素。如果没有重复元素，这个配对就是一个答案。

### 程序

class Solution {

public:

vector<vector<int> > fourSum(vector<int> &num, int target) {

vector<vector<int> > ret;

unordered\_map<int,vector<pair<int,int>> > map;

set<int> checked;

set<vector<int> > set1;

int n=num.size();

for(int i=0;i<n-1;i++)

for(int j=i+1;j<n;j++){

int k=num[i]+num[j];

map[k].push\_back(pair<int,int>(i,j));

}

for(auto e:map){

int tmp = target - e.first;

if(checked.count(tmp)!=0)

continue;

if(map.find(tmp)!=map.end()){

int i1=e.second.size();

int j1=map[tmp].size();

for(int i=0;i<i1;i++)

for(int j=0;j<j1;j++)

{

int a1,a2,b1,b2;

a1 = e.second[i].first;

a2 = e.second[i].second;

b1 = map[tmp][j].first;

b2 = map[tmp][j].second;

if(

(a1 == b1)

||(a1 == b2)

||(a2 == b1)

||(a2 == b2)

)

{

continue;

}

vector<int> t;

t.push\_back(num[a1]);

t.push\_back(num[a2]);

t.push\_back(num[b1]);

t.push\_back(num[b2]);

sort(t.begin(),t.end());

set1.insert(t);

}

checked.insert(tmp);

checked.insert(e.first);

}

}

for(auto e:set1){

ret.push\_back(e);

}

return ret;

}

};

# Letter Combinations of a Phone Number

## 题目

Given a digit string, return all possible letter combinations that the number could represent.

A mapping of digit to letters (just like on the telephone buttons) is given below.



Input:Digit string "23"

Output: ["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"].

## 解法

### 分析

 很简单的递归即可实现。

### 程序

static char content[][5]={

"",

"",

"abc",

"def",

"ghi",

"jkl",

"mno",

"pqrs",

"tuv",

"wxyz"

};

class Solution {

public:

void recur(string digits, char\* one, int pos, vector<string>& ret){

int dl=digits.length();

if(dl==0){

ret.push\_back(string(one));

return;

}

int n=digits[0]-'0';

int l= strlen(content[n]);

for(int i=0;i<l;i++){

one[pos] = content[n][i];

recur(digits.substr(1,dl-1),one,pos+1,ret);

}

}

vector<string> letterCombinations(string digits) {

vector<string> ret;

char one[digits.size()+1];

one[digits.size()]='\0';

recur(digits,one,0, ret);

return ret;

}

};

# Remove Nth Node From End of List

## 题目

Given a linked list, remove the nth node from the end of list and return its head.

For example,

Given linked list: 1->2->3->4->5, and n = 2.

After removing the second node from the end, the linked list becomes 1->2->3->5.

Note:

Given n will always be valid.

Try to do this in one pass.

## 解法

### 分析

 移除链表尾部倒数第n个元素。算法很简单，这是一道编程题。

用两个指针，间隔n+1个元素，从头扫描到尾，第二个元素遇到NULL时，第一个元素就指向倒数第n+1个元素。然后就可以移除需要的元素了。

一些特殊场景需要额外注意。

### 程序

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

class Solution {

public:

ListNode \*removeNthFromEnd(ListNode \*head, int n) {

ListNode \*tail, \*body,\*last=NULL;

if(head==NULL)return NULL;

tail=head;

for(int i=0;i<n-1;i++){

if(tail!=NULL)

tail=tail->next;

else

return head;

}

body = head;

while(tail->next!=NULL){

tail=tail->next;

last = body;

body=body->next;

}

if(last==NULL){

int val = head->val;

body=head->next;

if(body!=NULL){

head->val=body->val;

head->next = body->next;

body->val = val;

body->next = NULL;

return head;

}

else{

return NULL;

}

}

else{

last->next=body->next;

body->next = NULL;

return head;

}

}

};

#

# Valid Parentheses

## 题目

Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

The brackets must close in the correct order, "()" and "()[]{}" are all valid but "(]" and "([)]" are not.

## 解法

### 分析

典型的动态规划题，d[i][j]表示从第i个元素开始，长度为j的字符串，是否合法。

j为奇数，必定为false。

余下三种情况

1. 首尾字符匹配，而且去掉首尾之后余下的串合法，则整个串合法；
2. 以某个偶数位置分原串为两部分，且两部分分别合法，则整个串合法
3. 所有其他情况均不合法。

### 程序

static const int SIZE\_C=200;

static bool result[SIZE\_C][SIZE\_C];

class Solution {

public:

bool isMatch(char a, char b){

if((a=='(')&&(b==')'))return true;

if((a=='{')&&(b=='}'))return true;

if((a=='[')&&(b==']'))return true;

return false;

}

bool isValid(string s) {

int n=s.length();

if(n%2!=0)return false;

for(int i=0;i<=n;i++){

result[i][0]=true;

}

for(int j=2;j<=n;j+=2){

for(int i=0;i<=n-j;i+=1){

result[i][j]=false;

if(isMatch(s[i],s[i+j-1])&&result[i+1][j-2])

result[i][j]=true;

else{

for(int k=2;k<=j-2;k+=2){

if(result[i][k]&&result[i+k][j-k])

result[i][j]=true;

}

}

}

}

return result[0][n];

}

};

#

# Generate Parentheses

## 题目

Given n pairs of parentheses, write a function to generate all combinations of well-formed parentheses.

For example, given n = 3, a solution set is:

"((()))", "(()())", "(())()", "()(())", "()()()"

## 解法

### 分析

用递归，每个位置最多两种可能，”(”或者”)”，但右括弧能否使用，受制于已经出现的左、右括弧数目；左括弧能否使用，受制于是否还有空闲。

### 程序

class Solution {

public:

void recurr(char\* word, int curr, int left, int right, int all, vector<string> &ret){

if(curr>=2\*all){

word[curr]='\0';

string tmp(word);

ret.push\_back(tmp);

return;

}

if(left==right){

word[curr]='(';

recurr(word,curr+1,left+1, right,all, ret);

}

else if(left<all){

word[curr]='(';

recurr(word,curr+1,left+1,right,all,ret);

word[curr]=')';

recurr(word,curr+1,left,right+1,all,ret);

}

else{

word[curr]=')';

recurr(word,curr+1,left,right+1,all,ret);

}

}

vector<string> generateParenthesis(int n) {

vector<string> ret;

char word[2\*n+1];

recurr(word,0,0,0,n,ret);

return ret;

}

};

# Merge k Sorted Lists

## 题目

Merge k sorted linked lists and return it as one sorted list. Analyze and describe its complexity.

## 解法

### 分析

 编程题，关键是k个值如何快速定位最小值。最容易想到的就是heap结构。

如果k是个恒定值，这个优化效果不大，因为最后复杂度仍旧是总量O(N)（k可以被视为常量c）。如果k是个变量而且可能较大，则复杂度由O(kN)降为O(Nlgk)。

### 程序

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

class Solution {

public:

void adjustHeap(vector<ListNode \*> &lists, int pos){

int l,r,n;

n=lists.size();

l=2\*pos+1;

r=2\*pos+2;

if(l>=n)

return;

if(r>=n){

if(lists[pos]->val>lists[l]->val){

ListNode \*tmp;

tmp=lists[pos];

lists[pos]=lists[l];

lists[l]=tmp;

}

return;

}

if(

(lists[pos]->val>=lists[l]->val)&&

(lists[l]->val>=lists[r]->val)

)

{

ListNode \*tmp;

tmp=lists[pos];

lists[pos]=lists[r];

lists[r]=tmp;

adjustHeap(lists,r);

}

else if(

(lists[pos]->val>=lists[r]->val)&&

(lists[r]->val>=lists[l]->val)

)

{

ListNode \*tmp;

tmp=lists[pos];

lists[pos]=lists[l];

lists[l]=tmp;

adjustHeap(lists,l);

}

else if(

(lists[pos]->val>=lists[l]->val)

)

{

ListNode \*tmp;

tmp=lists[pos];

lists[pos]=lists[l];

lists[l]=tmp;

adjustHeap(lists,l);

}

else if(

(lists[pos]->val>=lists[r]->val)

)

{

ListNode \*tmp;

tmp=lists[pos];

lists[pos]=lists[r];

lists[r]=tmp;

adjustHeap(lists,r);

}

}

ListNode \*mergeKLists(vector<ListNode \*> &lists) {

ListNode\* head,\*tail;

head=tail=NULL;

int n=lists.size();

if(n==0)return NULL;

for(int i=n-1;i>=0;i--){

if(lists[i]==NULL){

if(i!=n-1){

lists[i]=lists[n-1];

}

lists.erase(lists.begin()+n-1);

n--;

}

}

if(n==0)return NULL;

for(int i=n-1;i>=0;i--){

adjustHeap(lists,i);

}

while(1){

if(head==NULL)

{

head=tail=lists[0];

}

else{

tail->next=lists[0];

tail=tail->next;

}

if(lists.size()==1)

break;

if(lists[0]->next!=NULL){

lists[0]=lists[0]->next;

adjustHeap(lists,0);

}

else{

n = lists.size();

lists[0] = lists[n-1];

lists.erase(lists.begin()+n-1);

adjustHeap(lists,0);

}

}

return head;

}

};

# Swap Nodes in Pairs

## 题目

Given a linked list, swap every two adjacent nodes and return its head.

For example,

Given 1->2->3->4, you should return the list as 2->1->4->3.

Your algorithm should use only constant space. You may not modify the values in the list, only nodes itself can be changed.

## 解法

### 分析

 方法应该很多，但是挑选一个合适的方法可以降低编程的难度。这里我选择的是先拆分为奇偶两个链表，再重新组合的方式。复杂度都是O(N)。

### 程序

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

class Solution {

public:

ListNode \*swapPairs(ListNode \*head) {

ListNode\* p1\_head;

ListNode\* p2\_head;

ListNode\* p1\_tail;

ListNode\* p2\_tail;

p1\_head=p1\_tail=head;

if(p1\_head==NULL)return NULL;

p2\_head=p2\_tail=head->next;

if(p2\_head==NULL)return head;

while(1){

p1\_tail->next=p2\_tail->next;

p1\_tail=p1\_tail->next;

if(p1\_tail==NULL){

p2\_tail->next=NULL;

break;

}

p2\_tail->next=p1\_tail->next;

p2\_tail=p2\_tail->next;

if(p2\_tail==NULL){

p1\_tail->next=NULL;

break;

}

}

ListNode\* nHead;

ListNode\* tmp=new ListNode(0);

ListNode\* nTail=tmp;

nHead=p2\_head;

while(1){

ListNode\* t1=p1\_head->next;

ListNode\* t2=p2\_head->next;

nTail->next=p2\_head;

nTail=nTail->next;

nTail->next=p1\_head;

nTail=nTail->next;

p1\_head=t1;

p2\_head=t2;

if(p1\_head==NULL)break;

if(p2\_head==NULL)break;

}

if(p1\_head!=NULL){

nTail->next = p1\_head;

}

delete(tmp);

return nHead;

}

};

# Reverse Nodes in k-Group

## 题目

Given a linked list, reverse the nodes of a linked list k at a time and return its modified list.

If the number of nodes is not a multiple of k then left-out nodes in the end should remain as it is.

You may not alter the values in the nodes, only nodes itself may be changed.

Only constant memory is allowed.

For example,

Given this linked list: 1->2->3->4->5

For k = 2, you should return: 2->1->4->3->5

For k = 3, you should return: 3->2->1->4->5

## 解法

### 分析

 一道编程题。我的程序不简单。不知道是否有更简便的方法。

分配k个指针，每次将这批指针赋值为下一个group的k个节点。如果能够读取k个值，则反转链接这个group。如果已经无法获取k个值（到达链表尾部），则正向链接这些值。

### 程序

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

class Solution {

public:

ListNode \*reverseKGroup(ListNode \*head, int k) {

ListNode\* ret=NULL;

ListNode\* p[k];

ListNode\* p\_head[k];

ListNode\* p\_tail[k];

if(head==NULL)

return NULL;

if(k==0||k==1)

return head;

for(int i=0;i<k;i++){

p\_head[i]=p\_tail[i]=NULL;

}

ListNode\* tail=head;

while(1){

p[0]=tail;

int i;

for(i=1;i<k;i++){

p[i]=p[i-1]->next;

if(p[i]==NULL)

break;

}

if(i==k){

for(i=0;i<k;i++){

if(p\_head[i]==NULL){

p\_head[i]=p[i];

p\_tail[i]=p[i];

}

else{

p\_tail[i]->next = p[i];

p\_tail[i] = p[i];

}

}

tail=p[k-1]->next;

if(tail==NULL)

break;

}

else{

break;

}

}

if(p\_head[0]==NULL)

return head;

for(int i=0;i<k;i++){

p\_tail[i]->next = NULL;

}

ret=p\_head[k-1];

ListNode \*tmp=new ListNode(0);

ListNode \*head1=tmp;

while(1){

for(int j=k;j>0;j--){

head1->next = p\_head[j-1];

head1 = head1->next;

p\_head[j-1]=p\_head[j-1]->next;

}

if(p\_head[0]==NULL)

break;

}

head1->next=tail;

delete(tmp);

return ret;

}

};

# Remove Duplicates from Sorted Array

## 题目

Given a sorted array, remove the duplicates in place such that each element appear only once and return the new length.

Do not allocate extra space for another array, you must do this in place with constant memory.

For example,

Given input array A = [1,1,2],

Your function should return length = 2, and A is now [1,2].

## 解法

### 分析

 题目简单，不解释。

### 程序

class Solution {

public:

int removeDuplicates(int A[], int n) {

int ret;

if(n==0)

return 0;

int read;

int write;

read=write=0;

while(1){

read++;

if(read>=n)

break;

if(A[read]==A[write])

continue;

else{

write++;

if(read!=write)

A[write]=A[read];

}

}

return write+1;

}

};

# Remove Element

## 题目

Given an array and a value, remove all instances of that value in place and return the new length.

The order of elements can be changed. It doesn't matter what you leave beyond the new length.

## 解法

### 分析

 简单不解释。

### 程序

class Solution {

public:

int removeElement(int A[], int n, int elem) {

int r,w;

r=w=0;

while(1){

if(r>=n)

break;

if(A[r]!=elem){

if(r!=w){

A[w]=A[r];

}

w++;

}

r++;

}

return w;

}

};

# Implement strStr()

## 题目

Returns a pointer to the first occurrence of needle in haystack, or null if needle is not part of haystack.

## 解法

### 分析

要通过测试，必须使用KMP算法。关键是next数组的计算方法。从网上搜到的相应解释。



### 程序

class Solution {

public:

void get\_nextval(char\* p, int len, int\* next) {

next[0] = -1;

int j = 0;

int k = -1;

while (j < len - 1) {

if (k == -1 || p[j] == p[k]) {

next[++j] = ++k;

} else {

k = next[k];

}

}

return;

}

void printNext(char\* p, int \*next,int l){

printf("Char List: [");

for(int i=0;i<l;i++){

printf("%2c,",p[i]);

}

printf("]\n");

printf("Next List: [");

for(int i=0;i<l;i++){

printf("%2d,",next[i]);

}

printf("]\n");

}

char \*strStr(char \*haystack, char \*needle) {

char \*s=haystack;

char \*p=needle;

int l\_p = strlen(p);

int l\_s = strlen(s);

int next[l\_p];

get\_nextval(p,l\_p,next);

//printNext(p,next,l\_p);

int i=0;

int j=0;

while(i<l\_s&&j<l\_p){

if(j==-1||s[i]==p[j])

{

i++;

j++;

}

else{

j=next[j];

}

}

if(j>=l\_p){

return s+i-l\_p;

}

return NULL;

}

};

# Divide Two Integers

## 题目

Divide two integers without using multiplication, division and mod operator.

## 解法

### 分析

不准用除法和取模，计算两个整数相除的结果。

思路，用左移和减法。 整数二进制左移就是乘2。除数先左移至恰好小于被除数。计算结果初始化为零。减去这个值后，计算结果加上2的相应次方。

剩余部分将除数右移，继续。

实际上就是把被除数分解为除数的2^n的级数。

### 程序

class Solution {

public:

unsigned int l\_divide(unsigned int dividend, unsigned int divisor) {

unsigned int ret=1;

int o\_divisor=divisor;

if(dividend<divisor)

return 0;

if(dividend==divisor)

return 1;

while(1){

if(divisor<dividend){

divisor <<= 1;

ret<<=1;

}

else{

break;

}

}

ret>>=1;

divisor>>=1;

return ret+divide((dividend-divisor),o\_divisor);

}

int divide(int dividend, int divisor) {

if(divisor==0)

return 0;

if(dividend==0)

return 0;

int ret;

int sign = 1;

if(

(dividend>0&&divisor>0)

||(dividend<0&&divisor<0)

)

{

}

else

{

sign = -1;

}

if(dividend<0)

dividend = 0 - dividend;

if(divisor<0)

divisor = 0 - divisor;

ret = (int)l\_divide(dividend, divisor);

if(sign==-1){

ret = 0 - ret;

}

return ret;

}

};

# Substring with Concatenation of All Words

## 题目

You are given a string, S, and a list of words, L, that are all of the same length. Find all starting indices of substring(s) in S that is a concatenation of each word in L exactly once and without any intervening characters.

For example, given:

S: "barfoothefoobarman"

L: ["foo", "bar"]

You should return the indices: [0,9].

(order does not matter).

## 解法

### 分析

 编程题，算法无难度。

### 程序

struct WordNode{

int id;

int num;

WordNode(int i,int j):id(i),num(j){};

WordNode():id(0),num(0){};

};

class Solution {

public:

vector<int> findSubstring(string S, vector<string> &L) {

vector<int> ret;

if(L.size()==0)

return ret;

int wordNum = L.size();

int wordLen = L[0].length();

if((int)S.length()<wordNum\*wordLen)

return ret;

int wordMap[wordNum];

map<string,WordNode> mMap;

for(int i=0;i<(int)L.size();i++){

if(mMap.count(L[i])==0){

mMap.insert(make\_pair(L[i],WordNode(i,1)));

}

else{

mMap[L[i]].num ++;

}

}

for(int i=0;i<=(S.length()-wordNum\*wordLen);i++){

int j;

for(j=0;j<wordNum;j++){

wordMap[j]=0;

}

for(j=0;j<wordNum;j++){

string oneWord(S.substr(i+wordLen\*j,wordLen));

if(mMap.count(oneWord)!=0){

if(wordMap[mMap[oneWord].id]<mMap[oneWord].num)

wordMap[mMap[oneWord].id]++;

else

break;

}

else{

break;

}

}

if(j>=wordNum)

ret.push\_back(i);

}

return ret;

}

};

# Next Permutation

## 题目

Implement next permutation, which rearranges numbers into the lexicographically next greater permutation of numbers.

If such arrangement is not possible, it must rearrange it as the lowest possible order (ie, sorted in ascending order).

The replacement must be in-place, do not allocate extra memory.

Here are some examples. Inputs are in the left-hand column and its corresponding outputs are in the right-hand column.

1,2,3 → 1,3,2

3,2,1 → 1,2,3

1,1,5 → 1,5,1

## 解法

### 分析

 这道题目的关键是找出方法来计算下一个值。分析排列规律即可。

### 程序

class Solution {

public:

void nextPermutation(vector<int> &num) {

int i;

for(i=num.size()-1;i>0;i--){

if(num[i-1]<num[i]){

break;

}

}

if(i==0){

reverse(num.begin(),num.end());

return;

}

int sign\_pos = i-1;

int exch\_pos;

for(exch\_pos=num.size()-1;exch\_pos>0;exch\_pos--){

if(num[exch\_pos]>num[sign\_pos])

break;

}

int tmp;

tmp = num[exch\_pos];

num[exch\_pos] = num[sign\_pos];

num[sign\_pos] = tmp;

reverse(num.begin()+sign\_pos+1,num.end());

return;

}

};

# Longest Valid Parentheses

## 题目

Given a string containing just the characters '(' and ')', find the length of the longest valid (well-formed) parentheses substring.

For "(()", the longest valid parentheses substring is "()", which has length = 2.

Another example is ")()())", where the longest valid parentheses substring is "()()", which has length = 4.

## 解法

### 分析

 这是一道动态规划题，O(N^3)的d函数比较容易想到。O(N)的d函数有点难度。

假设d[i]表示从第i个字符开始能够组成合法串的最大长度。从后向前推导。

1. 如果当前字符是”)”，d值为0.
2. 否则的话，看跳过d[i+1]个字符后的位置（假设为j），是否是”)”
3. 如果不是，d值也是0
4. 如果是，d值是d[i+1]+2（和后面那个位置的”)”构成一个合法串），再加上d[j]（和前面那个合法串相邻的下一个最长合法串）。

### 程序

static const int MAX\_SIZE3 = 16000;

static int dp[MAX\_SIZE3];

class Solution {

public:

int longestValidParentheses(string s) {

int len = s.length();

if(len<=1)

return 0;

for(int i=0;i<len;i++){

dp[i]=0;

}

dp[len-1]=0;

for(int i=len-2;i>=0;i--){

if(s[i]==')')

dp[i]=0;

else{ // s[i]=='('

int j;

j=dp[i+1]+i+1;

if(j<len&&s[j]==')'){

dp[i]=dp[i+1]+2;

if((j+1)<len)

dp[i]+=dp[j+1];

}

else{

dp[i]=0;

}

}

}

int max\_len = 0;

for(int i=0;i<len;i++){

if(dp[i]>max\_len)

max\_len=dp[i];

}

return max\_len;

}

};

# Search in Rotated Sorted Array

## 题目

Suppose a sorted array is rotated at some pivot unknown to you beforehand.

(i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2).

You are given a target value to search. If found in the array return its index, otherwise return -1.

You may assume no duplicate exists in the array.

## 解法

### 分析

 用二分法，算法上没有大的难度。主要是找到分割点。

### 程序

class Solution {

public:

void findPosition(int A[], int n, int &pos){

int l,r;

l=0;r=n-1;

pos=n-1;

while(l<r){

if(A[r]>A[l]){

pos=r;

return;

}

if((r-l)<10)

{

for(int j=l;j<=(r-1);j++){

if(A[j]>A[j+1]){

pos=j;

break;

}

}

return;

}

int tmp=(l+r)/2;

if(A[tmp]>A[0])

{

l=tmp;

}

else

{

r=tmp;

}

}

}

int searchTarget(int A[],int begin,int end, int target){

while(begin<end){

if(A[begin]==target)

return begin;

else if(A[end-1]==target)

return end-1;

else if((end-begin)<10){

for(int i=begin;i<end;i++){

if(A[i]==target){

return i;

}

}

return -1;

}

int tmp=(end+begin)/2;

if(A[tmp]>=target){

end=tmp+1;

}

else{

begin=tmp;

}

}

return -1;

}

int search(int A[], int n, int target) {

if(n<=0)return -1;

if(n==1){

if(A[0]==target)

return 0;

return -1;

}

int pos;

findPosition(A,n,pos);

if(target>=A[0]){

return searchTarget(A,0,pos+1,target);

}

return searchTarget(A,pos+1,n,target);

}

};

# Search for a Range

## 题目

Given a sorted array of integers, find the starting and ending position of a given target value.

Your algorithm's runtime complexity must be in the order of O(log n).

If the target is not found in the array, return [-1, -1].

For example,

Given [5, 7, 7, 8, 8, 10] and target value 8,

return [3, 4].

## 解法

### 分析

 基本思路为二分法，但是由于元素可以重复，寻找左右边界需要注意。

### 程序

class Solution {

public:

bool getTargetPosition(int A[], int l, int r, int target, int &pos){

if(r<l)

return false;

if((r-l)<10){

for(int i=l;i<=r;i++){

if(A[i]==target){

pos = i;

return true;

}

}

return false;

}

int mid = (l+r)/2;

if(A[mid]==target){

pos = mid;

return true;

}

else if(A[mid]<target){

return getTargetPosition(A,mid+1,r,target, pos);

}

return getTargetPosition(A,l,mid-1,target, pos);

}

void getLEdgePosition(int A[], int l, int r, int dir, int& pos){

int target = A[r];

if(r-l<10){

for(int i=l;i<=r;i++){

if(A[i]==target){

pos = i;

return;

}

}

}

int mid=(r+l)/2;

if(A[mid]==target){

getLEdgePosition(A,l,mid,dir,pos);

}

else{

getLEdgePosition(A,mid,r,dir,pos);

}

}

void getREdgePosition(int A[], int l, int r, int dir, int& pos){

int target = A[l];

if(r-l<10){

for(int i=r;i>=l;i--){

if(A[i]==target){

pos = i;

return;

}

}

}

int mid=(r+l)/2;

if(A[mid]==target){

getREdgePosition(A,mid,r,dir,pos);

}

else{

getREdgePosition(A,l,mid,dir,pos);

}

}

vector<int> searchRange(int A[], int n, int target) {

vector<int> ret;

int pos;

if(!getTargetPosition(A,0,n-1,target,pos)){

ret.push\_back(-1);

ret.push\_back(-1);

return ret;

}

int firstPos = -1;

int lastPos = -1;

getLEdgePosition(A,0,pos,1,firstPos);

getREdgePosition(A,pos,n-1,2,lastPos);

ret.push\_back(firstPos);

ret.push\_back(lastPos);

return ret;

}

};

# Search Insert Position

## 题目

. Given a sorted array and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You may assume no duplicates in the array.

Here are few examples.

[1,3,5,6], 5 → 2

[1,3,5,6], 2 → 1

[1,3,5,6], 7 → 4

[1,3,5,6], 0 → 0.

## 解法

### 分析

 算法无难度。

### 程序

. class Solution {

public:

void getInsertPosition(int A[], int l, int r, int target, int &pos)

{

if(l>r)

return;

if((r-l)<10){

int i;

for(i=l;i<=r;i++){

if(A[i]==target){

pos = i;

return;

}

else if(A[i]>target){

pos = i;

return;

}

}

pos=r+1;

return;

}

int mid=(r+l)/2;

if(target==A[mid]){

pos = mid;

return;

}

else if(target>A[mid]){

getInsertPosition(A,mid,r,target,pos);

}

else{

getInsertPosition(A,l,mid,target,pos);

}

}

int searchInsert(int A[], int n, int target) {

int pos=-1;

getInsertPosition(A,0,n-1,target,pos);

return pos;

}

};.

# Valid Sudoku

## 题目

. Determine if a Sudoku is valid, according to: Sudoku Puzzles - The Rules.

The Sudoku board could be partially filled, where empty cells are filled with the character '.'.



A partially filled sudoku which is valid.

Note:

A valid Sudoku board (partially filled) is not necessarily solvable. Only the filled cells need to be validated..

## 解法

### 分析

 .数独的规定，每行每列每个九宫格，都必须包含1-9九个数字，不能重复不能遗漏.

这个题目的一个注意点是，合法数独不一定有解。只要数独已经填上的数字不违反规定即可。

这个题目很简单。

### 程序

. class Solution {

public:

bool isSetValid(vector<int> &list){

int num[9];

for(int i=0;i<9;i++){

num[i]=0;

}

for(int i=0;i<(int)list.size();i++){

if(num[list[i]]==0){

num[list[i]]=1;

}

else

{

return false;

}

}

return true;

}

bool isRowValid(int row,vector<vector<char> > &board){

vector<int> l;

for(int i=0;i<9;i++){

int k=board[row][i]-'1';

if(k==('.'-'1'))

continue;

if(k>8||k<0)

return false;

l.push\_back(k);

}

return isSetValid(l);

}

bool isColValid(int col,vector<vector<char> > &board){

vector<int> l;

for(int i=0;i<9;i++){

int k=board[i][col]-'1';

if(k==('.'-'1'))

continue;

if(k>8||k<0)

return false;

l.push\_back(k);

}

return isSetValid(l);

}

bool isBlockValid(int i,vector<vector<char> > &board){

int row,col;

row=i/3;

row\*=3;

col=i%3;

col\*=3;

vector<int> list;

for(int l=0;l<3;l++){

for(int m=0;m<3;m++){

int k=board[row+l][col+m]-'1';

if(k==('.'-'1'))

continue;

if(k>8||k<0)

return false;

list.push\_back(k);

}

}

return isSetValid(list);

}

bool isValidSudoku(vector<vector<char> > &board) {

for(int i=0;i<9;i++){

if(!isRowValid(i,board))

return false;

if(!isColValid(i,board))

return false;

if(!isBlockValid(i,board))

return false;

}

return true;

}

};.

# Sudoku Solver

## 题目

. Write a program to solve a Sudoku puzzle by filling the empty cells.

Empty cells are indicated by the character '.'.

You may assume that there will be only one unique solution.



A sudoku puzzle...



...and its solution numbers marked in red..

## 解法

### 分析

 .解一个数独，比上一题增加了一点难度，不过也不是很麻烦。深度优先搜索即可。

先根据数独原始形态，求出每个空白点在这个时候可能的候选值。然后深度优选搜索，逐个点试用每个候选值。.

### 程序

. struct DFSCandidate{

int row;

int col;

vector<int> candidate;

DFSCandidate():row(0),col(0){}

};

struct DFSNode{

int step;

int select;

DFSNode(int i,int j):step(i),select(j){}

};

class Solution {

public:

bool isSetValid(vector<int> &list){

int num[10];

for(int i=0;i<10;i++){

num[i]=0;

}

for(int i=0;i<(int)list.size();i++){

if(num[list[i]]==0){

num[list[i]]=1;

}

else

{

return false;

}

}

return true;

}

void getRowSet(int row,vector<vector<char> > &board, vector<int> &l){

for(int i=0;i<9;i++){

int k=board[row][i]-'0';

if(k==('.'-'0'))

continue;

if(k>9||k<1)

return ;

l.push\_back(k);

}

}

bool isRowValid(int row,vector<vector<char> > &board){

vector<int> l;

getRowSet(row, board, l);

return isSetValid(l);

}

void getColSet(int col,vector<vector<char> > &board, vector<int> &l){

for(int i=0;i<9;i++){

int k=board[i][col]-'0';

if(k==('.'-'0'))

continue;

if(k>9||k<1)

return ;

l.push\_back(k);

}

}

bool isColValid(int col,vector<vector<char> > &board){

vector<int> l;

getColSet(col, board, l);

return isSetValid(l);

}

void getBlockSet(int r, int c,vector<vector<char> > &board, vector<int> &list){

int row, col;

row = ((int)(r/3))\*3;

col = ((int)(c/3))\*3;

for(int l=0;l<3;l++){

for(int m=0;m<3;m++){

int k=board[row+l][col+m]-'0';

if(k==('.'-'0'))

continue;

if(k>9||k<1)

return ;

list.push\_back(k);

}

}

}

bool isBlockValid(int row, int col, vector<vector<char> > &board){

vector<int> list;

getBlockSet(row, col, board, list);

return isSetValid(list);

}

void getCandidateSet(int r, int c, vector<vector<char> > &board, vector<int> &list){

vector<int> l1;

vector<int> l2;

vector<int> l3;

getRowSet(r,board,l1);

getColSet(c,board,l2);

getBlockSet(r,c,board,l3);

bool isExist[10];

for(int i=0;i<10;i++)

isExist[i] = false;

for(int i=0;i<(int)l1.size();i++){

isExist[l1[i]]=true;

}

for(int i=0;i<(int)l2.size();i++){

isExist[l2[i]]=true;

}

for(int i=0;i<(int)l3.size();i++){

isExist[l3[i]]=true;

}

for(int i=1;i<10;i++){

if(!isExist[i])

list.push\_back(i);

}

}

void solveSudoku(vector<vector<char> > &board) {

int i,j;

vector<DFSCandidate> candidateList;

for(i=0;i<9;i++){

for(j=0;j<9;j++){

if(board[i][j]=='.'){

DFSCandidate oneStep;

oneStep.row = i;

oneStep.col = j;

getCandidateSet(i,j,board,oneStep.candidate);

if(oneStep.candidate.size()==0)

return;

candidateList.push\_back(oneStep);

}

}

}

stack<DFSNode> mStack;

mStack.push(DFSNode(0,0));

while(!mStack.empty()){

DFSNode curr = mStack.top();

mStack.pop();

if(curr.step>=(int)candidateList.size()){

//find valid solver

return;

}

int r,c;

r=candidateList[curr.step].row;

c=candidateList[curr.step].col;

while(curr.select < (int)candidateList[curr.step].candidate.size())

{

board[r][c]=candidateList[curr.step].candidate[curr.select]+'0';

if(

isRowValid(r,board)

&&isColValid(c,board)

&&isBlockValid(r,c,board)

)

{

mStack.push(DFSNode(curr.step,curr.select+1));

mStack.push(DFSNode(curr.step+1,0));

break;

}

curr.select ++;

}

if(curr.select >= (int)candidateList[curr.step].candidate.size())

board[r][c]='.';

}

return;

}

};.

# Count and Say

## 题目

. The count-and-say sequence is the sequence of integers beginning as follows:

1, 11, 21, 1211, 111221, ...

1 is read off as "one 1" or 11.

11 is read off as "two 1s" or 21.

21 is read off as "one 2, then one 1" or 1211.

Given an integer n, generate the nth sequence.

Note: The sequence of integers will be represented as a string..

## 解法

### 分析

 .这是一道模拟题，没有发现有什么巧妙解法，迭代计算即可通过测试.

### 程序

. struct MListNode{

int val;

int count;

MListNode(int i,int j):val(i),count(j){}

};

class Solution {

public:

void analyzeCurr(string &s,vector<MListNode> &content){

content.clear();

int i=0;

int val=s[i]-'0';

int curr;

int count=1;

for(;i<s.length()-1;i++){

curr=s[i+1]-'0';

if(curr!=val){

content.push\_back(MListNode(val,count));

count=1;

val = curr;

}

else{

count++;

}

}

content.push\_back(MListNode(val,count));

}

void generateNext(string &s,vector<MListNode> &content){

s.clear();

char buf[128];

for(int i=0;i<(int)content.size();i++){

snprintf(buf,128,"%d%d",content[i].count,content[i].val);

s += buf;

}

}

string countAndSay(int n) {

string currStr;

vector<MListNode> content;

currStr.assign("1");

for(int i=0;i<n-1;i++){

analyzeCurr(currStr,content);

generateNext(currStr,content);

}

return currStr;

}

};.

# Combination Sum

## 题目

. Given a set of candidate numbers (C) and a target number (T), find all unique combinations in C where the candidate numbers sums to T.

The same repeated number may be chosen from C unlimited number of times.

Note:

All numbers (including target) will be positive integers.

Elements in a combination (a1, a2, … , ak) must be in non-descending order. (ie, a1 ≤ a2 ≤ … ≤ ak).

The solution set must not contain duplicate combinations.

For example, given candidate set 2,3,6,7 and target 7,

A solution set is:

[7]

[2, 2, 3].

## 解法

### 分析

 .由于可以重复使用某一个元素，这道题目比较简单。递归即可。最麻烦的其实是如何避免重复解。.

### 程序

. class Solution {

public:

bool isSameVector(vector<int> &a,vector<int> &b){

if(a.size()!=b.size()){

return false;

}

for(int i=0;i<(int)a.size();i++){

if(a[i]!=b[i])

return false;

}

return true;

}

vector<vector<int> > getSum(vector<int> &candidates, int sum){

vector<vector<int> > ret;

if(sum<candidates[0])

return ret;

for(int i=0;i<(int)candidates.size();i++){

int part = sum - candidates[i];

if(part==0){

vector<int> tmp;

tmp.push\_back(candidates[i]);

ret.push\_back(tmp);

}

else{

vector<vector<int> > tmp = getSum(candidates, part);

if(tmp.size()>0){

for(int j=0;j<(int)tmp.size();j++){

tmp[j].push\_back(candidates[i]);

sort(tmp[j].begin(),tmp[j].end());

//need to check whether ret contain same set

int k=0;

for(;k<(int)ret.size();k++){

if(isSameVector(ret[k],tmp[j]))

break;

}

if(k>=(int)ret.size())

ret.push\_back(tmp[j]);

}

}

}

}

return ret;

}

vector<vector<int> > combinationSum(vector<int> &candidates, int target) {

vector<vector<int> > ret;

sort(candidates.begin(),candidates.end());

return getSum(candidates,target);

}

};.

# Combination Sum II

## 题目

. Given a collection of candidate numbers (C) and a target number (T), find all unique combinations in C where the candidate numbers sums to T.

Each number in C may only be used once in the combination.

Note:

All numbers (including target) will be positive integers.

Elements in a combination (a1, a2, … , ak) must be in non-descending order. (ie, a1 ≤ a2 ≤ … ≤ ak).

The solution set must not contain duplicate combinations.

For example, given candidate set 10,1,2,7,6,1,5 and target 8,

A solution set is:

[1, 7]

[1, 2, 5]

[2, 6]

[1, 1, 6].

## 解法

### 分析

 .由于每个元素不能重复，这道题比上一题难度增加很多。动态规划可以解。

假设d[i][j]表示从第i个元素开始到队尾的数组中，能够得到总和为j的组合。那么，d[i][j]相当于两种情况的合并

1. d[i+1][j]，也就是不采用当前元素
2. d[i+1][j-num[i]] + num[i]，（d[i+1][j-num[i]]的每个单元，在加上一个num[i]），也就是采用当前元素.

### 程序

. const int MAX\_SIZE=100;

vector<vector<int> > result[MAX\_SIZE][MAX\_SIZE];

bool visited[MAX\_SIZE][MAX\_SIZE];

class Solution {

public:

vector<vector<int> > dFunction(int i, int target, vector<int> &num){

if(visited[i][target])

return result[i][target];

vector<vector<int> > t1, t2;

if(i>=(int)num.size()){

visited[i][target]=true;

result[i][target] = t1;

return t1;

}

if(target<=0){

visited[i][target]=true;

result[i][target] = t1;

return t1;

}

t1 = dFunction(i+1,target,num);

if(target==num[i]){

vector<int> tmp;

tmp.push\_back(num[i]);

t2.push\_back(tmp);

}

else{

t2 = dFunction(i+1,target-num[i],num);

if(t2.size()!=0){

for(int k=0;k<(int)t2.size();k++){

t2[k].push\_back(num[i]);

}

}

}

if(t1.empty()){

visited[i][target]=true;

result[i][target] = t2;

return t2;

}

if(t2.empty()){

visited[i][target]=true;

result[i][target] = t1;

return t1;

}

for(int k=0;k<(int)t2.size();k++){

t1.push\_back(t2[k]);

}

visited[i][target]=true;

result[i][target] = t1;

return t1;

}

bool isSameVector(vector<int> &a,vector<int> &b){

if(a.size()!=b.size()){

return false;

}

for(int i=0;i<(int)a.size();i++){

if(a[i]!=b[i])

return false;

}

return true;

}

void initResult(){

for(int i=0;i<MAX\_SIZE;i++){

for(int j=0;j<MAX\_SIZE;j++){

visited[i][j]=false;

}

}

}

vector<vector<int> > combinationSum2(vector<int> &num, int target) {

initResult();

vector<vector<int> > t1, t2;

t1 = dFunction(0,target, num);

for(int i=0;i<(int)t1.size();i++){

sort(t1[i].begin(),t1[i].end());

}

for(int i=0;i<(int)t1.size();i++){

int j;

for(j=0;j<(int)t2.size();j++){

if(isSameVector(t1[i],t2[j])){

break;

}

}

if(j>=(int)t2.size())

t2.push\_back(t1[i]);

}

return t2;

}

};.

# First Missing Positive

## 题目

. Given an unsorted integer array, find the first missing positive integer.

For example,

Given [1,2,0] return 3,

and [3,4,-1,1] return 2.

Your algorithm should run in O(n) time and uses constant space..

## 解法

### 分析

 .寻找第一个不存在的正整数。从头到尾梳理每个值，数值i就放在第i-1位置上，小于等于0，和大于等于n的值都不管。最后从位置0开始往上找，第一个数值不是位置加一的地方，就是缺失的正整数。

一些异常需要注意，特别是有重复值，在某些位置，可能无法在当前位置上找到自己寻找的值而陷入死循环。.

### 程序

. class Solution {

public:

int firstMissingPositive(int A[], int n) {

int ret=-1;

for(int i=0;i<n;i++){

while(1){

if(A[i]<=0||A[i]>=n||A[i]==i+1)

break;

//swap A[i] A[A[i]-1]

{

int a = A[i];

A[i] = A[a-1];

A[a-1] = a;

if(A[i]==a)

break;

}

}

}

for(int i=0;i<n;i++){

if(A[i]!=i+1){

ret = i+1;

break;

}

}

if(ret<0)

ret = n+1;

return ret;

}

};.

# Trapping Rain Water

## 题目

. Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it is able to trap after raining.

For example,

Given [0,1,0,2,1,0,1,3,2,1,2,1], return 6.



The above elevation map is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped. Thanks Marcos for contributing this image!.

## 解法

### 分析

 .这题目是三维版本的简化版。

从两边向中间递归。每次取两侧中较矮的一端。以这端为外墙，计算能够储水的最大值。一直到遇到墙高超出外墙的位置为止（这里已经不可能再以起始点为外墙了）。.

### 程序

. class Solution {

public:

int trap(int A[], int n) {

if(n<=2)

return 0;

if(A[0]<=A[n-1])

{

int i=1;

int water=0;

while(A[i]<A[0])

{

water+=A[0]-A[i];

i++;

}

return water + trap(A+i,n-i);

}

else //(A[0]>A[n-1])

{

int i=n-2;

int water=0;

while(A[i]<A[n-1])

{

water+=A[n-1]-A[i];

i--;

}

return water + trap(A,i+1);

}

return 0;

}

};

.

# Multiply Strings

## 题目

. Given two numbers represented as strings, return multiplication of the numbers as a string.

Note: The numbers can be arbitrarily large and are non-negative..

## 解法

### 分析

 .其实就是计算超大整数乘法。采用不同的乘法计算方法，程序编写的难度大不相同。这是我在网上找到的一个方法。很简单。.

### 程序

. class Solution {

public:

string multiply(string num1, string num2) {

string ret;

vector<int> result;

int l=num1.length()+num2.length();

result.reserve(num1.length()+num2.length());

for(int i=0;i<l;i++){

result.push\_back(0);

}

reverse(num1.begin(),num1.end());

reverse(num2.begin(),num2.end());

for(int i=0;i<num1.length();i++){

for(int j=0;j<num2.length();j++){

result[i+j] += (num1[i]-'0')\*(num2[j]-'0');

}

}

int val;

int carry=0;

for(int i=0;i<result.size();i++){

result[i] += carry;

val = result[i]%10;

carry = result[i]/10;

ret += val+'0';

}

for(int i=ret.length();i>0;i--){

if(ret[i-1]=='0'){

ret.erase(ret.end()-1);

}

else

break;

}

if(ret.empty()){

ret += '0';

}

reverse(ret.begin(),ret.end());

return ret;

}

};.

# Wildcard Matching

## 题目

. Implement wildcard pattern matching with support for '?' and '\*'.

'?' Matches any single character.

'\*' Matches any sequence of characters (including the empty sequence).

The matching should cover the entire input string (not partial).

The function prototype should be:

bool isMatch(const char \*s, const char \*p)

Some examples:

isMatch("aa","a") → false

isMatch("aa","aa") → true

isMatch("aaa","aa") → false

isMatch("aa", "\*") → true

isMatch("aa", "a\*") → true

isMatch("ab", "?\*") → true

isMatch("aab", "c\*a\*b") → false.

## 解法

### 分析

动态规划可以解决，需要从空字符串开始比较起，用d[i,j]表示目标串的前i位和模式串的前j位是否能够匹配。有两点提示：

1. 先把模式串简化一下，所有的重复\*都缩减为单个\*，可以简化很多处理，而且结果是一样的。
2. 如果目标串为空，那么，当模式串为\*时是可以匹配的；其他则不可以匹配。.

### 程序

. public class Solution {

public static final int sampleLength=1000;

public static final int patternLength=100;

boolean[][] result= new boolean[sampleLength][patternLength];

private String checkPattern(String p){

StringBuffer t=new StringBuffer();

char lastCh='#';

for(int i=0;i<p.length();i++){

if(p.charAt(i)==lastCh&&lastCh=='\*'){

}

else{

t.append(p.charAt(i));

lastCh = p.charAt(i);

}

}

return t.toString();

}

public boolean isMatch(String s, String p) {

String p1 = checkPattern(p);

int length1 = s.length();

int length2 = p1.length();

{

int i = s.length();

int j = p.length();

if(j==0){

if(i==0)

return true;

else

return false;

}

if(i==0){

if(j==1&&p1.charAt(0)=='\*')

return true;

else

return false;

}

}

if(length1>sampleLength-2)

return false;

if(length2>patternLength-2)

return false;

for(int i=0;i<=length1;i++){

for(int j=0;j<=length2;j++){

if(i==0&&j==0){

result[i][j]=true;

}

else if(i==0){

if(j==1&&p1.charAt(0)=='\*')

result[i][j]=true;

else

result[i][j]=false;

}

else if(j==0){

result[i][j]=false;

}

else{

if(p1.charAt(j-1)=='?'){

if(result[i-1][j-1])

result[i][j]=true;

else

result[i][j]=false;

}

else if(p1.charAt(j-1)=='\*'){

if(result[i-1][j-1]||result[i-1][j]||result[i][j-1])

result[i][j]=true;

else

result[i][j]=false;

}

else{

if(s.charAt(i-1)==p1.charAt(j-1)){

if(result[i-1][j-1])

result[i][j]=true;

else

result[i][j]=false;

}

else{

result[i][j]=false;

}

}

}

}

}

return result[length1][length2];

}

}.

# Jump Game II

## 题目

. Given an array of non-negative integers, you are initially positioned at the first index of the array.

Each element in the array represents your maximum jump length at that position.

Your goal is to reach the last index in the minimum number of jumps.

For example:

Given array A = [2,3,1,1,4]

The minimum number of jumps to reach the last index is 2. (Jump 1 step from index 0 to 1, then 3 steps to the last index.).

## 解法

### 分析

 .建立一个链表，每个元素表示跳到当前这个点需要的最少步数。从头开始向后扫描。每个点计算能够跳到的最远位置，从那个位置向回赋值最少步数，赋值到已经有值的位置为止。（再往前不可能值更小了）。

由于每个位置最多赋值一次，复杂度为O(n)

### 程序

. int minStep[40000];

class Solution {

public:

int jump(int A[], int n) {

if(n==1)

return 0;

int maxDis=0;

for(int i=0;i<n+1;i++){

minStep[i]=-1;

}

maxDis=A[0];

for(int i=0;i<=maxDis;i++){

minStep[i]=1;

}

int curr=0;

while(maxDis<n){

curr++;

if(curr>maxDis)

break;

if(curr>n)

break;

if(curr+A[curr]>maxDis){

maxDis = curr+A[curr];

if(maxDis>n){

maxDis = n-1;

}

int j = maxDis;

while(minStep[j]==-1){

minStep[j]=minStep[curr]+1;

j--;

}

}

}

if(minStep[n-1]==-1)

return 0;

return minStep[n-1];

}

};.

# Permutations

## 题目

. Given a collection of numbers, return all possible permutations.

For example,

[1,2,3] have the following permutations:

[1,2,3], [1,3,2], [2,1,3], [2,3,1], [3,1,2], and [3,2,1]..

## 解法1

### 分析

 .经典递归方法，很简单。

.

### 程序

. class Solution {

public:

void swapVec(vector<int> &v, int a, int b){

if(a==b)return;

int tmp = v[a];

v[a] = v[b];

v[b] = tmp;

}

void mPermuteFunc(vector<int> &num, vector<vector<int> > &ret, int cur){

if(cur==(int)num.size()-1){

ret.push\_back(num);

return;

}

for(int i=cur;i<(int)num.size();i++){

swapVec(num, cur, i);

mPermuteFunc(num,ret,cur+1);

swapVec(num, cur, i);

}

}

vector<vector<int> > permute(vector<int> &num) {

vector<vector<int> > ret;

mPermuteFunc(num,ret,0);

return ret;

}

};.

## 解法2

### 分析

 .类似于产生整数的方法，每次计算值更大的下一个排列.

### 程序

. class Solution {

public:

void swapVec(vector<int> &v, int a, int b){

if(a==b)return;

int tmp = v[a];

v[a] = v[b];

v[b] = tmp;

}

bool getNextPermute(vector<int> &num){

int i;

for(i=num.size()-2;i>=0;i--){

if(num[i]<num[i+1])

break;

}

if(i<0)

return false;

int j;

for(j=num.size()-1;j>i;j--){

if(num[j]>num[i])

break;

}

swapVec(num,i,j);

sort(num.begin()+i+1,num.end());

return true;

}

vector<vector<int> > permute(vector<int> &num) {

vector<vector<int> > ret;

sort(num.begin(),num.end());

do{

ret.push\_back(num);

}

while(getNextPermute(num));

return ret;

}

};.

# Permutations II

## 题目

. Given a collection of numbers that might contain duplicates, return all possible unique permutations.

For example,

[1,1,2] have the following unique permutations:

[1,1,2], [1,2,1], and [2,1,1]..

## 解法

### 分析

 .如果上一题采用方法2，这一题很简单。（实际上上一题的方法2，我就是从这一题反推回去的。）

每次产生下一个排列，确保不会产生重复排列。.

### 程序

. class Solution {

public:

void swapVec(vector<int> &v, int a, int b){

if(a==b)return;

int tmp = v[a];

v[a] = v[b];

v[b] = tmp;

}

bool getNextPermute(vector<int> &num){

int i;

for(i=num.size()-2;i>=0;i--){

if(num[i]<num[i+1])

break;

}

if(i<0)

return false;

int j;

for(j=num.size()-1;j>i;j--){

if(num[j]>num[i])

break;

}

swapVec(num,i,j);

sort(num.begin()+i+1,num.end());

return true;

}

vector<vector<int> > permuteUnique(vector<int> &num) {

vector<vector<int> > ret;

sort(num.begin(),num.end());

do{

ret.push\_back(num);

}

while(getNextPermute(num));

return ret;

}

};.

# Rotate Image

## 题目

. You are given an n x n 2D matrix representing an image.

Rotate the image by 90 degrees (clockwise).

Follow up:

Could you do this in-place?.

## 解法

### 分析

 .其实是一道数学题，难度不大，主要是分析如何计算旋转后点的位置。下面是推荐的一种思路

1. 每四个点是一组循环。
2. 坐标轴变换思想，使得四个点的坐标值呈现对称关系，然后再反变化得到原始坐标轴，周长为奇数和为偶数需要分别对待
3. 周长为奇数时需要注意，选择处理的四分之一区域时，不要重复处理某些点（有一条坐标轴上的点需要排除）.

### 程序

. class Solution {

public:

void rotate(vector<vector<int> > &matrix) {

int n=matrix.size();

int l1=matrix.size()/2;

int l2=l1;

if(matrix.size()%2)

l1++;

for(int i=0;i<l1;i++){

for(int j=0;j<l2;j++){

int tmp=matrix[i][j];

matrix[i][j] = matrix[n-j-1][i];

matrix[n-j-1][i] = matrix[n-i-1][n-j-1];

matrix[n-i-1][n-j-1] = matrix[j][n-i-1];

matrix[j][n-i-1] = tmp;

}

}

}

};.

# Anagrams

## 题目

. Given an array of strings, return all groups of strings that are anagrams.

Note: All inputs will be in lower-case..

## 解法

### 分析

 .Anagrams的解释不是很确定。这里理解为，字母出现的种类和次数都一致的字符串，比如，都是由2个a，3个b构成的字符串。

把字符串按照字母序排序，则Anagrams字符串排序后的结果应该是一样的。利用这个寻找所有存在Anagrams关系的字符串。程序很简单。

这道题的返回值是一维数组，似乎有些奇怪。从结果来看，题目暗含的一个约束是，每一个输入其实都只有一个合法的group（成员数量超过一个）。

### 程序

. class Solution {

public:

vector<string> anagrams(vector<string> &strs) {

vector<string> ret;

if(strs.size()<=1)

return ret;

map<string, int> anag;

for(int i=0;i<(int)strs.size();i++){

string s(strs[i]);

sort(s.begin(),s.end());

if(anag.find(s)==anag.end()){

anag.insert(make\_pair(s,i));

}

else{

if(anag[s]>=0){

ret.push\_back(strs[anag[s]]);

anag[s] = -1;

}

ret.push\_back(strs[i]);

}

}

return ret;

}

};.

# Pow(x, n)

## 题目

. Implement pow(x, n)..

## 解法

### 分析

 .计算n次方。题目很简单，最笨的方法也能达到O(n)复杂度。稍微优化一下，可以达到O(lgn)复杂度。.

### 程序

. class Solution {

public:

double pow(double x, int n) {

if(n==0)

return 1.0;

int flag=1;

if(n<0)

{

n \*= -1;

flag = -1;

}

double ret = 1.0;

double exp\_x = x;

while(n>0){

if(n&1){

ret \*= exp\_x;

}

exp\_x \*= exp\_x;

n>>=1;

}

if(flag<0)

ret = 1/ret;

return ret;

}

};.

# N-Queens

## 题目

. The n-queens puzzle is the problem of placing n queens on an n×n chessboard such that no two queens attack each other.



Given an integer n, return all distinct solutions to the n-queens puzzle.

Each solution contains a distinct board configuration of the n-queens' placement, where 'Q' and '.' both indicate a queen and an empty space respectively.

For example,

There exist two distinct solutions to the 4-queens puzzle:

[

[".Q..", // Solution 1

"...Q",

"Q...",

"..Q."],

["..Q.", // Solution 2

"Q...",

"...Q",

".Q.."]

].

## 解法

### 分析

 .解八皇后问题，深度优先搜索。.

### 程序

. class Solution {

public:

bool checkValid(vector<string> &one, int size, int cur, int pos){

for(int i=0;i<cur;i++){

if(one[i][pos]=='Q')

return false;

int p1 = pos + cur - i;

int p2 = pos - cur + i;

if(p1>=0&&p1<size&&one[i][p1]=='Q')

return false;

if(p2>=0&&p2<size&&one[i][p2]=='Q')

return false;

}

return true;

}

void searchRes(

vector<vector<string> > &res,

vector<string> &one,

int size,

int cur

)

{

if(cur>=size)

{

res.push\_back(one);

return;

}

for(int i=0;i<size;i++){

one[cur][i]='Q';

if(checkValid(one,size, cur,i))

searchRes(res,one,size,cur+1);

one[cur][i]='.';

}

}

vector<vector<string> > solveNQueens(int n) {

vector<vector<string> > res;

vector<string> one;

string s;

for(int i=0;i<n;i++){

s += '.';

}

for(int i=0;i<n;i++){

one.push\_back(s);

}

searchRes(res, one, n, 0);

return res;

}

};.

# N-Queens II

## 题目

. Follow up for N-Queens problem.

Now, instead outputting board configurations, return the total number of distinct solutions..

## 解法一

### 分析

 .继续N皇后问题。这次只要计算个数，不需要求出解。

但是我没有找到简便方法，仍旧是求出所有解后，得到个数。不知道是否有更好办法。.

### 程序

. class Solution {

public:

bool checkValid(vector<string> &one, int size, int cur, int pos){

for(int i=0;i<cur;i++){

if(one[i][pos]=='Q')

return false;

int p1 = pos + cur - i;

int p2 = pos - cur + i;

if(p1>=0&&p1<size&&one[i][p1]=='Q')

return false;

if(p2>=0&&p2<size&&one[i][p2]=='Q')

return false;

}

return true;

}

void searchRes(

vector<vector<string> > &res,

vector<string> &one,

int size,

int cur

)

{

if(cur>=size)

{

res.push\_back(one);

return;

}

for(int i=0;i<size;i++){

one[cur][i]='Q';

if(checkValid(one,size, cur,i))

searchRes(res,one,size,cur+1);

one[cur][i]='.';

}

}

int totalNQueens(int n) {

vector<vector<string> > res;

vector<string> one;

string s;

for(int i=0;i<n;i++){

s += '.';

}

for(int i=0;i<n;i++){

one.push\_back(s);

}

searchRes(res, one, n, 0);

return res.size();

}

};.

## 解法二

### 分析

 .在网上看到一个聪明的办法，只是时间节省的不是很多，不过代码更简洁，而且理论上复杂度更低，可能在较大的输入量下可以体现优势。上一题可以采用同意的方法简化，我在leetcode上实验通过。这里不再重复贴解释了。

从网上拷贝来的解释：

使用位运算来求解N皇后的高效算法

核心代码如下：

[cpp] [view plaincopy](http://blog.csdn.net/hackbuteer1/article/details/6657109)

void test(int row, int ld, int rd)

{

    int pos, p;

    if ( row != upperlim )

    {

        pos = upperlim & (~(row | ld | rd ));

        while ( pos )

        {

            p = pos & (~pos + 1);

            pos = pos - p;

            test(row | p, (ld | p) << 1, (rd | p) >> 1);

        }

    }

    else

        ++Ans;

}

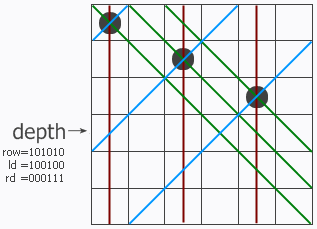
        初始化：　upperlim =  (1 << n)-1; Ans = 0;

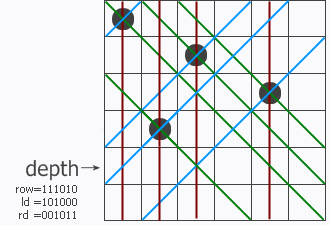
        调用参数：test(0, 0, 0);

         和普通算法一样，这是一个递归函数，程序一行一行地寻找可以放皇后的地方。函数带三个参数row、ld和rd，分别表示在纵列和两个对角线方向的限制条件下这一行的哪些地方不能放。位于该行上的冲突位置就用row、ld和rd中的1来表示。把它们三个并起来，得到该行所有的禁位，取反后就得到所有可以放的位置（用pos来表示）。

        p = pos & (~pos + 1)其结果是取出最右边的那个1。这样，p就表示该行的某个可以放子的位置，把它从pos中移除并递归调用test过程。

        注意递归调用时三个参数的变化，每个参数都加上了一个禁位，但两个对角线方向的禁位对下一行的影响需要平移一位。最后，如果递归到某个时候发现row=upperlim了，说明n个皇后全放进去了，找到的解的个数加一。





注：  
        upperlime：=（1 << n）-1 就生成了n个1组成的二进制数。  
        这个程序是从上向下搜索的。  
        pos & -pos 的意思就是取最右边的 1 再组成二进制数，相当于 pos &（~pos +1），因为取反以后刚好所有数都是相反的（怎么听着像废话），再加 1 ，就是改变最低位，如果低位的几个数都是1，加的这个 1 就会进上去，一直进到 0 ，在做与运算就和原数对应的 1 重合了。举例可以说明：

        原数 0 0 0 0 1 0 0 0    原数 0 1 0 1 0 0 1 1

        取反 1 1 1 1 0 1 1 1    取反 1 0 1 0 1 1 0 0  
        加1    1 1 1 1 1 0 0 0    加1  1 0 1 0 1 1 0 1

  与运算    0 0 0 0 1 0 0 0    and  0 0 0 0 0 0 0 1  
      其中呢，这个取反再加 1 就是补码，and 运算 与负数，就是按位和补码与运算。  
       (ld | p)<< 1 是因为由ld造成的占位在下一行要右移一下；  
       (rd | p)>> 1 是因为由rd造成的占位在下一行要左移一下。  
        ld rd row 还要和upperlime 与运算 一下，这样做的结果就是从最低位数起取n个数为有效位置，原因是在上一次的运算中ld发生了右移，如果不and的话，就会误把n以外的位置当做有效位。.

### 程序

. public class Solution {

private void collect(int[] res, int row, int rd, int ld, int curr, int n){

if(curr>=n){

res[0]++;

return;

}

int mask = ~(~0<<n);

int avail = (~(row|rd|ld))&mask;

int pos;

while(avail>0){

pos = avail & (-1\*avail);

avail -= pos;

collect(res, row|pos, (rd|pos)<<1, (ld|pos)>>1, curr+1, n);

}

}

public int totalNQueens(int n) {

int[] res = new int[1];

res[0] = 0;

collect(res, 0, 0, 0, 0, n);

return res[0];

}

}.

# Maximum Subarray

## 题目

. Find the contiguous subarray within an array (containing at least one number) which has the largest sum.

For example, given the array [?2,1,?3,4,?1,2,1,?5,4],

the contiguous subarray [4,?1,2,1] has the largest sum = 6..

## 解法

### 分析

 .经典问题，求和最大的连续子数组。动态规划可解。不详细解释了。.

### 程序

. class Solution {

public:

int maxSubArray(int A[], int n) {

if(n<=0)

return 0;

int maxTotal = A[0];

int maxEnd = A[0];

for(int i=1;i<n;i++){

if(maxEnd<0){

maxEnd=A[i];

}

else{

maxEnd+=A[i];

}

if(maxTotal<A[i])

maxTotal = A[i];

if(maxTotal<maxEnd)

maxTotal=maxEnd;

}

return maxTotal;

}

};.

# Spiral Matrix

## 题目

.

Given a matrix of m x n elements (m rows, n columns), return all elements of the matrix in spiral order.

For example,

Given the following matrix:

[

[ 1, 2, 3 ],

[ 4, 5, 6 ],

[ 7, 8, 9 ]

]

You should return [1,2,3,6,9,8,7,4,5]..

## 解法

### 分析

 .不知道该归为什么类型问题，深度优先？算法很简单，模拟人解决的思路即可。不知道是否有更巧妙解法。.

### 程序

. class Solution {

public:

void nextStep(int &i, int &j, int direct){

// Directory:

// 0: right

// 1: down

// 2: left

// 3: up

switch(direct){

case 0:

j ++;

break;

case 1:

i ++;

break;

case 2:

j --;

break;

case 3:

i --;

break;

default:

break;

}

}

bool getNextPos(vector<vector<bool> > &visited,int &i, int &j,int &direct){

int row,col;

row = visited.size();

col = visited[0].size();

int origin\_dir = direct;

while(1){

int next\_i,next\_j;

next\_i = i;

next\_j = j;

nextStep(next\_i,next\_j,direct);

if(

next\_i>=0

&&next\_i<row

&&next\_j>=0

&&next\_j<col

&&!visited[next\_i][next\_j]

)

{

i = next\_i;

j = next\_j;

return true;

}

direct = (direct+1)%4;

if(direct == origin\_dir)

return false;

}

return false;

}

vector<int> spiralOrder(vector<vector<int> > &matrix) {

vector<int> ret;

int row,col;

row = matrix.size();

if(row==0)

return ret;

col = matrix[0].size();

if(col==0)

return ret;

vector<vector<bool> > visited;

vector<bool> oneLine;

for(int i=0;i<col;i++){

oneLine.push\_back(false);

}

for(int i=0;i<row;i++){

visited.push\_back(oneLine);

}

oneLine.clear();

int direct = 0;

int i,j;

i=j=0;

while(1){

ret.push\_back(matrix[i][j]);

visited[i][j]=true;

if(!getNextPos(visited,i,j,direct))

break;

}

return ret;

}

};.

# Jump Game

## 题目

. Given an array of non-negative integers, you are initially positioned at the first index of the array.

Each element in the array represents your maximum jump length at that position.

Determine if you are able to reach the last index.

For example:

A = [2,3,1,1,4], return true.

A = [3,2,1,0,4], return false..

## 解法

### 分析

 .这个题目初看很简单。但是测试的cases要求很严，而且某些程序写法会导致大量的重复步骤。下面的程序是网上看到的，程序简单，执行效率很高。

.

### 程序

. class Solution {

public:

bool canJump(int A[], int n) {

int canReach=0;

if(n==0)

return false;

if(n==1)

return true;

for(int i=0;i<n;i++){

if(canReach<i)break;

if(i+A[i]>canReach)

canReach=i+A[i];

if(canReach>=n-1)

return true;

}

return canReach>=n-1;

}

};.

# Merge Intervals

## 题目

. Given a collection of intervals, merge all overlapping intervals.

For example,

Given [1,3],[2,6],[8,10],[15,18],

return [1,6],[8,10],[15,18].

/\*\*

\* Definition for an interval.

\* struct Interval {

\* int start;

\* int end;

\* Interval() : start(0), end(0) {}

\* Interval(int s, int e) : start(s), end(e) {}

\* };

\*/

.

## 解法

### 分析

 .所有区间按照起点位置排序。然后每个区间逐个向后检查，是否重叠。重叠则合并。

.

### 程序

. static bool comp(const struct Interval &a,const struct Interval &b){

if(a.start==b.start){

return a.end<b.end;

}

else

return a.start < b.start;

}

class Solution {

public:

bool isJoint(const struct Interval &a,const struct Interval &b){

if(b.start>a.end)

return false;

return true;

}

void mergeInt(struct Interval &a,const struct Interval &b){

if(a.end<b.end){

a.end = b.end;

}

}

vector<Interval> merge(vector<Interval> &intervals) {

vector<Interval> res;

if(intervals.size()<=0)

return res;

sort(intervals.begin(),intervals.end(), comp);

res.push\_back(intervals[0]);

for(int i=1;i<(int)intervals.size();i++){

int j=res.size();

if(isJoint(res[j-1],intervals[i])){

mergeInt(res[j-1],intervals[i]);

}

else{

res.push\_back(intervals[i]);

}

}

return res;

}

};.

# Insert Interval

## 题目

. Given a set of non-overlapping intervals, insert a new interval into the intervals (merge if necessary).

You may assume that the intervals were initially sorted according to their start times.

Example 1:

Given intervals [1,3],[6,9], insert and merge [2,5] in as [1,5],[6,9].

Example 2:

Given [1,2],[3,5],[6,7],[8,10],[12,16], insert and merge [4,9] in as [1,2],[3,10],[12,16].

This is because the new interval [4,9] overlaps with [3,5],[6,7],[8,10]..

## 解法

### 分析

 . 这个问题如果选择方法不当，编程会非常复杂，而且执行复杂度也很容易超标。

我的方法是，区间本身是不会重叠的，把整个空间按照被这些区间分割的情况，逐个编号。找到新区间落在哪个编号的区间中（可能是空闲区间，也可能是被占据区间）。然后判断处置方法。.

### 程序

. /\*\*

\* Definition for an interval.

\* struct Interval {

\* int start;

\* int end;

\* Interval() : start(0), end(0) {}

\* Interval(int s, int e) : start(s), end(e) {}

\* };

\*/

class Solution {

public:

int getSegment(vector<Interval> &iv, int val){

int l = 0;

int r = iv.size();

int pos = -1;

while(l<r){

if(r-l<10){

for(int i=l;i<r;i++){

if(iv[i].start>val){

pos = i;

break;

}

}

if(pos<0){

pos=r;

}

break;

}

int mid = (r+l)/2;

if(iv[mid].start==val){

pos = mid+1;

break;

}

else if(iv[mid].start>val){

r = mid;

}

else{

l = mid;

}

}

if(pos==0)

return 0;

pos --;

if(iv[pos].end>=val){

return pos\*2+1;

}

return pos\*2+2;

}

vector<Interval> insert(vector<Interval> &intervals, Interval newInterval) {

if(intervals.empty()){

intervals.push\_back(newInterval);

return intervals;

}

int first, second;

first = getSegment(intervals,newInterval.start);

second = getSegment(intervals,newInterval.end);

if(first==second){

if(first%2){

return intervals;

}

else{

intervals.insert(intervals.begin()+first/2, newInterval);

return intervals;

}

}

else{

int nStart,nEnd;

if(first%2){

nStart=intervals[first/2].start;

}

else{

nStart = newInterval.start;

}

if(second%2){

nEnd = intervals[second/2].end;

}

else{

nEnd = newInterval.end;

}

intervals.erase(intervals.begin()+first/2, intervals.begin()+(second+1)/2);

intervals.insert(intervals.begin()+first/2, Interval(nStart,nEnd));

return intervals;

}

return intervals;

}

};.

# Length of Last Word

## 题目

. Given a string s consists of upper/lower-case alphabets and empty space characters ' ', return the length of last word in the string.

If the last word does not exist, return 0.

Note: A word is defined as a character sequence consists of non-space characters only.

For example,

Given s = "Hello World",

return 5..

## 解法

### 分析

 .算法无难度，编程题。.

### 程序

. class Solution {

public:

bool isLetter(char ch){

if(ch>='a'&&ch<='z')

return true;

if(ch>='A'&&ch<='Z')

return true;

return false;

}

int lengthOfLastWord(const char \*s) {

int p1,p2;

bool inWord=false;

p1 = p2 = -1;

int curr=0;

while(1){

if(s[curr]=='\0'){

if(inWord){

return curr - p1;

}

else if(p1!=-1&&p2!=-1){

return p2-p1;

}

return 0;

}

if(isLetter(s[curr])&&!inWord){

inWord=true;

p1 = curr;

}

else if(!isLetter(s[curr])&&inWord){

inWord=false;

p2 = curr;

}

curr++;

}

return p2-p1;

}

};.

# Spiral Matrix II

## 题目

. Given an integer n, generate a square matrix filled with elements from 1 to n2 in spiral order.

For example,

Given n = 3,

You should return the following matrix:

[

[ 1, 2, 3 ],

[ 8, 9, 4 ],

[ 7, 6, 5 ]

].

## 解法

### 分析

 .和I简直一样，模拟人解决方法即可。.

### 程序

. class Solution {

public:

void nextStep(int &i, int &j, int direct){

// Directory:

// 0: right

// 1: down

// 2: left

// 3: up

switch(direct){

case 0:

j ++;

break;

case 1:

i ++;

break;

case 2:

j --;

break;

case 3:

i --;

break;

default:

break;

}

}

bool getNextPos(vector<vector<bool> > &visited,int &i, int &j,int &direct){

int row,col;

row = visited.size();

col = visited[0].size();

int origin\_dir = direct;

while(1){

int next\_i,next\_j;

next\_i = i;

next\_j = j;

nextStep(next\_i,next\_j,direct);

if(

next\_i>=0

&&next\_i<row

&&next\_j>=0

&&next\_j<col

&&!visited[next\_i][next\_j]

)

{

i = next\_i;

j = next\_j;

return true;

}

direct = (direct+1)%4;

if(direct == origin\_dir)

return false;

}

return false;

}

vector<vector<int> > generateMatrix(int n) {

vector<vector<int> > matrix;

if(0==n)

return matrix;

int row,col;

row = n;

col = n;

vector<vector<bool> > visited;

vector<int> oneLineI;

vector<bool> oneLineB;

for(int i=0;i<col;i++){

oneLineB.push\_back(false);

oneLineI.push\_back(0);

}

for(int i=0;i<row;i++){

visited.push\_back(oneLineB);

matrix.push\_back(oneLineI);

}

oneLineB.clear();

oneLineI.clear();

int direct = 0;

int i,j;

i=j=0;

int curr=1;

while(1){

matrix[i][j]=curr;

visited[i][j]=true;

if(!getNextPos(visited,i,j,direct))

break;

curr++;

}

return matrix;

}

};.

# Permutation Sequence

## 题目

. The set [1,2,3,…,n] contains a total of n! unique permutations.

By listing and labeling all of the permutations in order,

We get the following sequence (ie, for n = 3):

"123"

"132"

"213"

"231"

"312"

"321"

Given n and k, return the kth permutation sequence.

Note: Given n will be between 1 and 9 inclusive..

## 解法

### 分析

 .这是一道数学题。N个数的所有排列方式将是N!，N较大的时候穷举的方式是非常慢的。

假设要解决的是1..9的排列，可以观察的是，以1开始的(n-1)!种排列，将占据最前面的(n-1)!个位置。如果要求的是第k个排列，k与(n-1)!相除的结果为i(0<=i<=9)，则第一个位置需要放置的数字将是1..9中的第i+1个值。后面的位置以此类推。

### 程序

. class Solution {

public:

int factorial(int n){

int ret = 1;

for(int i=2;i<=n;i++){

ret \*= i;

}

return ret;

}

string getPermutation(int n, int k) {

string s;

int total = factorial(n);

if(k>total)

return s;

vector<bool> usage;

for(int i=0;i<n;i++){

usage.push\_back(false);

}

for(int i=0;i<n;i++){

if(k==0){

for(int j=n-1;j>=0;j--){

if(!usage[j]){

s += j+1+'0';

usage[j]=true;

}

}

return s;

}

int vol = factorial(n-i-1);

int pos;

if(k%vol)

pos = 1+k/vol;

else

pos = k/vol;

k = k % vol;

{

for(int j=0;j<n;j++){

if(!usage[j]){

pos --;

}

if(pos==0){

s += j+1+'0';

usage[j]=true;

break;

}

}

}

}

return s;

}

};.

# Rotate List

## 题目

. Given a list, rotate the list to the right by k places, where k is non-negative.

For example:

Given 1->2->3->4->5->NULL and k = 2,

return 4->5->1->2->3->NULL..

## 解法

### 分析

 ..算法无难度。需要注意，k有可能大于链表长度，这时候也需要能正确处理。

### 程序

. /\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

class Solution {

public:

ListNode \*rotateRight(ListNode \*head, int i\_k) {

if(head==NULL)

return head;

ListNode\* p1,\*p2;

int length;

int k=i\_k;

retry:

if(k==0)

return head;

p1=head;

while(1){

p1=p1->next;

k--;

if(p1==NULL){

length=i\_k-k;

k = i\_k % length;

goto retry;

}

if(k<=0)

break;

}

p2=head;

while(p1->next!=NULL){

p1=p1->next;

p2=p2->next;

}

p1->next = head;

head = p2->next;

p2->next = NULL;

return head;

}

};.

# Unique Paths

## 题目

. A robot is located at the top-left corner of a m x n grid (marked 'Start' in the diagram below).

The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid (marked 'Finish' in the diagram below).

How many possible unique paths are there?

Above is a 3 x 7 grid. How many possible unique paths are there?

Note: m and n will be at most 100..

## 解法

### 分析

 ..这是一道排列组合数学题。

### 程序

. class Solution {

public:

int uniquePaths(int m, int n) {

long long ret=1;

int l,r;

if(m>n){

r=m-1;

l=n-1;

}

else{

r=n-1;

l=m-1;

}

for(int i=r+1;i<=(l+r);i++)

{

ret \*= (long long)i;

}

for(int i=2;i<=l;i++){

ret /= (long long)i;

}

return (int)ret;

}

};.

# Unique Paths II

## 题目

. Follow up for "Unique Paths":

Now consider if some obstacles are added to the grids. How many unique paths would there be?

An obstacle and empty space is marked as 1 and 0 respectively in the grid.

For example,

There is one obstacle in the middle of a 3x3 grid as illustrated below.

[

[0,0,0],

[0,1,0],

[0,0,0]

]

The total number of unique paths is 2.

Note: m and n will be at most 100..

## 解法

### 分析

 .动态规划，到达每个block的走法，取决于到达它上方和左方的block的走法。.

### 程序

. const int MAX\_SIZE=800;

int result[MAX\_SIZE][MAX\_SIZE];

class Solution {

public:

void initRes(int row, int col){

for(int i=0;i<row;i++){

for(int j=0;j<col;j++){

result[i][j]=-1;

}

}

}

int calResult(int i,int j,vector<vector<int> > &g){

if(result[i][j]!=-1)

return result[i][j];

if(g[i][j]==1)

{

result[i][j]=0;

return 0;

}

if(i==0&&j==0){

result[i][j]=1;

return result[i][j];

}

if(i==0){

result[i][j]=calResult(i,j-1,g);

return result[i][j];

}

if(j==0){

result[i][j]=calResult(i-1,j,g);

return result[i][j];

}

int r1,r2;

r1=calResult(i-1,j,g);

r2=calResult(i,j-1,g);

result[i][j]=r1+r2;

return result[i][j];

}

int uniquePathsWithObstacles(vector<vector<int> > &obstacleGrid) {

int row,col;

row=obstacleGrid.size();

if(row==0)

return 0;

col=obstacleGrid[0].size();

if(col==0)

return 0;

initRes(row, col);

return calResult(row-1,col-1,obstacleGrid);

}

};.

# Minimum Path Sum

## 题目

. Given a m x n grid filled with non-negative numbers, find a path from top left to bottom right which minimizes the sum of all numbers along its path.

Note: You can only move either down or right at any point in time..

## 解法

### 分析

 .与上一题几乎一样.

### 程序

. const int MAX\_SIZE=800;

int result[MAX\_SIZE][MAX\_SIZE];

class Solution {

public:

int minPathSum(vector<vector<int> > &grid) {

int row = grid.size()-1;

int col = grid[0].size()-1;

for(int i=0;i<(int)grid.size();i++){

for(int j=0;j<(int)grid[0].size();j++){

if(i==0&&j==0)

{

result[i][j]=grid[i][j];

continue;

}

else if(i==0){

result[i][j]=result[i][j-1]+grid[i][j];

continue;

}

else if(j==0){

result[i][j]=result[i-1][j]+grid[i][j];

continue;

}

int r1,r2;

r1=result[i][j-1]+grid[i][j];

r2=result[i-1][j]+grid[i][j];

if(r1<r2){

result[i][j]=r1;

}

else{

result[i][j]=r2;

}

}

}

return result[row][col];

}

};.

# Merge Two Sorted Lists

## 题目

. Merge two sorted linked lists and return it as a new list. The new list should be made by splicing together the nodes of the first two lists..

## 解法

### 分析

 ..简单编程，无难度。复杂度O(m+n)，似乎不太可能有更好办法。

### 程序

. class Solution {

public:

ListNode \*mergeTwoLists(ListNode \*l1, ListNode \*l2) {

ListNode \* head=NULL;

ListNode \* tail=NULL;

if(l1==NULL&&l2==NULL){

return NULL;

}

else if(l1==NULL){

return l2;

}

else if(l2==NULL){

return l1;

}

if(l1->val<l2->val){

head = tail = l1;

l1=l1->next;

tail->next = NULL;

}

else{

head = tail = l2;

l2=l2->next;

tail->next = NULL;

}

while(1){

if(l1==NULL&&l2==NULL)

break;

else if(l1==NULL){

tail->next = l2;

break;

}

else if(l2==NULL){

tail->next = l1;

break;

}

if(l1->val<l2->val){

tail->next = l1;

l1 = l1->next;

}

else{

tail->next = l2;

l2 = l2->next;

}

tail = tail->next;

tail->next = NULL;

}

return head;

}

};.

# Add Binary

## 题目

. Given two binary strings, return their sum (also a binary string).

For example,

a = "11"

b = "1"

Return "100"..

## 解法

### 分析

 .不分析.

### 程序

. class Solution {

public:

string addBinary(string a, string b) {

string s;

reverse(a.begin(),a.end());

reverse(b.begin(),b.end());

int carrybit=0;

int i=0;

while(1){

int val=0;

val += carrybit;

if(i<(int)a.length()){

val += a[i]-'0';

}

if(i<(int)b.length()){

val += b[i]-'0';

}

if(val%2){

s += '1';

}

else{

s += '0';

}

carrybit = val/2;

i++;

if(i>=(int)a.length() && i>=(int)b.length() && carrybit==0)

break;

}

reverse(s.begin(),s.end());

return s;

}

};.

# Plus One

## 题目

. Given a non-negative number represented as an array of digits, plus one to the number.

The digits are stored such that the most significant digit is at the head of the list..

## 解法

### 分析

 ..不解释

### 程序

. class Solution {

public:

vector<int> plusOne(vector<int> &digits) {

vector<int> ret;

int num = digits.size();

int i=0;

int carrybit=1;

do{

int val = digits[num-i-1]+carrybit;

carrybit=val/10;

val %= 10;

ret.push\_back(val);

i++;

}while(i<num);

if(carrybit>0){

ret.push\_back(carrybit);

}

reverse(ret.begin(),ret.end());

return ret;

}

};.

# Text Justification

## 题目

. Given an array of words and a length L, format the text such that each line has exactly L characters and is fully (left and right) justified.

You should pack your words in a greedy approach; that is, pack as many words as you can in each line. Pad extra spaces ' ' when necessary so that each line has exactly L characters.

Extra spaces between words should be distributed as evenly as possible. If the number of spaces on a line do not divide evenly between words, the empty slots on the left will be assigned more spaces than the slots on the right.

For the last line of text, it should be left justified and no extra space is inserted between words.

For example,

words: ["This", "is", "an", "example", "of", "text", "justification."]

L: 16.

Return the formatted lines as:

[

"This is an",

"example of text",

"justification. "

]

Note: Each word is guaranteed not to exceed L in length..

## 解法

### 分析

 ..编程题，细节很多。算法无难度。

### 程序

. class Solution {

public:

vector<string> fullJustify(vector<string> &words, int L) {

vector<string> ret;

int b;

int e;

bool meetEnd=false;

if(words.size()==0){

string oneLineStr;

for(int i=0;i<L;i++)

oneLineStr += ' ';

ret.push\_back(oneLineStr);

return ret;

}

e=0;

while(e<(int)words.size()){

int oneline=0;

int wordsWidth=0;

b=e;

while((oneline+(int)words[e].length())<= L){

oneline += words[e].length();

wordsWidth += words[e].length();

e++;

if(oneline!=L)

oneline ++; // for ' '

if(e>=(int)words.size()){

meetEnd = true;

break;

}

}

string oneLineStr;

oneLineStr.clear();

if(meetEnd){

int allSpaceNum = (L - wordsWidth - e+b+1);

for(int i=0;i<(e-b-1);i++){

oneLineStr += words[i+b];

oneLineStr += ' ';

}

oneLineStr += words[e-1];

for(int i=0;i<allSpaceNum;i++){

oneLineStr += ' ';

}

}

else{

if((e-b-1)==0){

oneLineStr += words[b];

for(int i=0;i<(L - wordsWidth);i++){

oneLineStr += ' ';

}

}

else{

int allSpaceNum = (L - wordsWidth)/(e-b-1);

int extSpace = (L - wordsWidth)%(e-b-1);

for(int i=0;i<(e-b-1);i++){

oneLineStr += words[i+b];

for(int j=0;j<allSpaceNum;j++){

oneLineStr += ' ';

}

if(extSpace>0){

oneLineStr += ' ';

}

extSpace --;

}

oneLineStr += words[e-1];

}

}

ret.push\_back(oneLineStr);

}

return ret;

}

};.

# Sqrt(x)

## 题目

.. Implement int sqrt(int x).

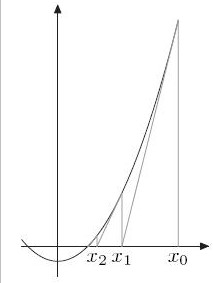
Compute and return the square root of x.

## 解法

### 分析

 ..最简单的方法是二分法。

更好的方法是牛顿迭代法。



为了方便理解，就先以本题为例：

计算x2 = n的解，令f(x)=x2-n，相当于求解f(x)=0的解，如左图所示。

首先取x0，如果x0不是解，做一个经过(x0,f(x0))这个点的切线，与x轴的交点为x1。

同样的道理，如果x1不是解，做一个经过(x1,f(x1))这个点的切线，与x轴的交点为x2。

以此类推。

以这样的方式得到的xi会无限趋近于f(x)=0的解。

判断xi是否是f(x)=0的解有两种方法：

一是直接计算f(xi)的值判断是否为0，二是判断前后两个解xi和xi-1是否无限接近。

经过(xi, f(xi))这个点的切线方程为f(x) = f(xi) + f’(xi)(x - xi)，其中f'(x)为f(x)的导数，本题中为2x。令切线方程等于0，即可求出xi+1=xi - f(xi) / f'(xi)。

继续化简，xi+1=xi - (xi2 - n) / (2xi) = xi - xi / 2 + n / (2xi) = xi / 2 + n / 2xi = (xi + n/xi) / 2。

有了迭代公式xi+1= (xi + n/xi) / 2，程序就好写了。关于牛顿迭代法，可以参考wikipedia以及百度百科。

### 程序

. class Solution {

public:

int sqrt(int x) {

int ret=0;

if(x<0)return -1;

if(x==0)return x;

if(x<4)return 1;

long long l=1;

long long r=x/2;

while(l<r){

if((r-l)<10){

for(long long i=l;i<=r;i++){

long long a1=i\*i;

long long a2=(i+1)\*(i+1);

if(x>=a1&&x<a2)

return i;

}

}

long long mid=(l+r)/2;

long long t=mid\*mid;

if(t==x)

return mid;

if(t<x){

l=mid;

}

else{

r=mid;

}

}

return ret;

}

};.

# Climbing Stairs

## 题目

. You are climbing a stair case. It takes n steps to reach to the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?.

## 解法

### 分析

 .可以用动态规划优化。对于每一个台阶i来说，到达这个台阶有多少种走法，取决于i-1, i-2. F(i)=f(i-1)+f(i-2)

F(0)=1

F(1)=2

这其实就是斐波纳切数列。

### 程序

. const int MAX\_SIZE=800;

int result[MAX\_SIZE];

class Solution {

public:

int climbStairs(int n) {

int ret=0;

for(int i=0;i<=n;i++){

if(i<=1)

result[i]=1;

else

result[i]=result[i-1]+result[i-2];

}

return result[n];

}

};.

# Simplify Path

## 题目

.. Given an absolute path for a file (Unix-style), simplify it.

For example,

path = "/home/", => "/home"

path = "/a/./b/../../c/", => "/c"

## 解法

### 分析

 ..建立一个堆栈，每个被”/”包围的entity作为一个节点。

遇到..，出栈，遇到.，不变，遇到其他串，进栈。

### 程序

. class Solution {

public:

void dividePath(string &path,vector<string> &list){

int i=0;

string one;

while(i<(int)path.length()){

if(path[i]=='/'){

if(one.length()>0){

list.push\_back(one);

one.clear();

}

}

else{

one += path[i];

}

i++;

}

if(one.length()>0){

list.push\_back(one);

}

}

string simplifyPath(string path) {

string ret;

vector<string> list;

vector<string> stack;

dividePath(path, list);

for(int i=0;i<(int)list.size();i++){

if(list[i].compare("..")==0){

if(!stack.empty())

stack.pop\_back();

}

else if(list[i].compare(".")==0){

}

else{

stack.push\_back(list[i]);

}

}

if(path[0]=='/')

ret += '/';

for(int i=0;i<(int)stack.size();i++){

ret += stack[i];

if(i!=(stack.size()-1))

ret += '/';

}

return ret;

}

};.

# Edit Distance

## 题目

.. Given two words word1 and word2, find the minimum number of steps required to convert word1 to word2. (each operation is counted as 1 step.)

You have the following 3 operations permitted on a word:

a) Insert a character

b) Delete a character

c) Replace a character

## 解法

### 分析

 ..标准的动态规划，

### 程序

. const int MAX\_SIZE=800;

int result[MAX\_SIZE][MAX\_SIZE];

class Solution {

public:

int min(int a,int b,int c){

int i=a;

if(i>b)

i=b;

if(i>c)

i=c;

return i;

}

int minDistance(string word1, string word2) {

int l1=word1.length();

int l2=word2.length();

for(int a1=0;a1<=l1;a1++){

for(int a2=0;a2<=l2;a2++){

if(a1==0)

result[a1][a2]=a2;

else if(a2==0)

result[a1][a2]=a1;

else{

if(word1[a1-1]==word2[a2-1])

result[a1][a2]=result[a1-1][a2-1];

else{

result[a1][a2]=min(

result[a1-1][a2]+1,

result[a1][a2-1]+1,

result[a1-1][a2-1]+1

);

}

}

}

}

return result[l1][l2];

}

};.

# Set Matrix Zeroes

## 题目

. Given a m x n matrix, if an element is 0, set its entire row and column to 0. Do it in place..

## 解法

### 分析

 ..不分析。感觉很简单。

### 程序

. class Solution {

public:

void getRowAndCol(vector<vector<int> > &m,set<int> &rows,set<int> &cols){

for(int i=0;i<(int)m.size();i++){

for(int j=0;j<(int)m[0].size();j++){

if(m[i][j]==0){

rows.insert(i);

cols.insert(j);

}

}

}

}

void setRowAndCol(vector<vector<int> > &m,set<int> &rows,set<int> &cols){

for(int i=0;i<(int)m.size();i++){

for(int j=0;j<(int)m[0].size();j++){

if(rows.count(i)>0||cols.count(j)>0)

m[i][j]=0;

}

}

}

void setZeroes(vector<vector<int> > &matrix) {

set<int> rows;

set<int> cols;

getRowAndCol(matrix,rows,cols);

setRowAndCol(matrix,rows,cols);

return;

}

};.

# Search a 2D Matrix

## 题目

.. Write an efficient algorithm that searches for a value in an m x n matrix. This matrix has the following properties:

Integers in each row are sorted from left to right.

The first integer of each row is greater than the last integer of the previous row.

For example,

Consider the following matrix:

[

[1, 3, 5, 7],

[10, 11, 16, 20],

[23, 30, 34, 50]

]

Given target = 3, return true.

## 解法

### 分析

 ..编程题，算法无难点。

### 程序

. class Solution {

public:

bool getLines(vector<vector<int> > &m, int target, int &line){

if(target < m[0][0])return false;

int up=0;

int bottom=m.size()-1;

while(1){

if((bottom-up)<10){

for(int i=up;i<bottom;i++){

if(target>=m[i][0]&&target<m[i+1][0]){

line = i;

return true;

}

}

line = bottom;

return true;

}

int mid = (bottom + up)/2;

if(target>=m[mid][0]){

up = mid;

}

else{

bottom = mid - 1;

}

}

return true;

}

bool getResult(vector<int> m, int target){

bool ret=false;

int l=0;

int r=m.size()-1;

while(1){

if((r-l)<10){

for(int i=l;i<=r;i++){

if(m[i]==target)

return true;

}

return false;

}

int mid=(r+l)/2;

if((m[r]==target)||(m[l]==target)||(m[mid]==target))

return true;

if(m[r]<target)

return false;

if(m[mid]<target)

l=mid;

else

r=mid;

}

return ret;

}

bool searchMatrix(vector<vector<int> > &matrix, int target) {

int line;

if(getLines(matrix, target, line)==false)

return false;

return getResult(matrix[line],target);

}

};.

# Sort Colors

## 题目

.. Given an array with n objects colored red, white or blue, sort them so that objects of the same color are adjacent, with the colors in the order red, white and blue.

Here, we will use the integers 0, 1, and 2 to represent the color red, white, and blue respectively.

Note:

You are not suppose to use the library's sort function for this problem.

## 解法

### 分析

 ..计数排序

### 程序

. class Solution {

public:

void sortColors(int A[], int n) {

int c[3];

c[0]=c[1]=c[2]=0;

for(int i=0;i<n;i++)

{

if(A[i]<0||A[i]>2)

continue;

c[A[i]]++;

}

for(int i=1;i<3;i++){

c[i]+=c[i-1];

}

for(int i=0;i<n;i++){

if(i<c[0]){

A[i]=0;

}

else if(i>=c[0]&&i<c[1]){

A[i]=1;

}

else if(i>=c[1]&&i<c[2]){

A[i]=2;

}

}

return;

}

};.

# Minimum Window Substring

## 题目

.. Given a string S and a string T, find the minimum window in S which will contain all the characters in T in complexity O(n).

For example,

S = "ADOBECODEBANC"

T = "ABC"

Minimum window is "BANC".

Note:

If there is no such window in S that covers all characters in T, return the emtpy string "".

If there are multiple such windows, you are guaranteed that there will always be only one unique minimum window in S.

## 解法

### 分析

 .用两个游标扫描整个字符串，当游标之间的区域不包含所需要的所有字符时，右边的游标向右滑动。当游标之间的区域已经包括所有需要的字符时，左边游标向右滑动。找到两个游标之间的最小距离。

实际上，右边游标的滑动必须以每个字符为单位；而左边游标的滑动，其实只需要在目标字符之间跳动就可以了。

判断是否包含所有需要的字符，是一个难点。这里我采用了一个空缺字符集合，一个hash表（表中每个元素都是一个链表），加一个链表的方式。链表是游标之间所有目标元素按照顺序的一个排列。Hash表是每个目标字符的排列。大链表用于判断左边游标的下一个跳跃位置；hash表用于维护空缺字符集合；空缺字符集合用于判断区域是否包含所有需要的字符。

### 程序

. struct MListNode {

int pos;

MListNode \*next;

MListNode(int x) : pos(x), next(NULL) {}

};

struct HashNode{

MListNode\* link;

int count;

int target;

};

class Solution {

public:

HashNode hash\_head[256];

MListNode\* hash\_tail[256];

queue<int> q;

set<char> all\_ch;

set<char> all\_tmp;

void insert\_hash(char ch,int pos){

int i=ch;

if(hash\_tail[i]==NULL){

hash\_tail[i] = new MListNode(pos);

hash\_head[i].link = hash\_tail[i];

hash\_head[i].count ++;

}

else{

hash\_tail[i]->next = new MListNode(pos);

hash\_tail[i] = hash\_tail[i]->next;

hash\_head[i].count ++;

}

if(hash\_head[i].count>=hash\_head[i].target){

set<char>::iterator it;

it = all\_tmp.find(ch);

if(it!=all\_tmp.end()){

all\_tmp.erase(it);

}

}

}

void remove\_hash(char ch,int pos){

int i=ch;

if(hash\_head[i].link==NULL){

return;

}

else{

MListNode\* tmp=hash\_head[i].link;

if(tmp->pos!=pos)

return;

hash\_head[i].link = hash\_head[i].link->next;

delete(tmp);

if(hash\_head[i].link==NULL){

hash\_tail[i]=NULL;

}

hash\_head[i].count --;

if(hash\_head[i].count<hash\_head[i].target)

all\_tmp.insert(ch);

}

return;

}

void init(string T){

all\_ch.clear();

all\_tmp.clear();

for(int i=0;i<256;i++){

hash\_head[i].link=NULL;

hash\_head[i].count=0;

hash\_head[i].target=0;

hash\_tail[i]=NULL;

}

for(int i=0;i<(int)T.length();i++){

all\_ch.insert(T[i]);

all\_tmp.insert(T[i]);

int j=T[i];

hash\_head[j].target ++;

}

}

bool isAllExisted(){

return all\_tmp.empty();

}

void insertChar(char ch, int pos){

if(all\_ch.count(ch)!=0){

q.push(pos);

insert\_hash(ch,pos);

}

}

void deleteChar(char ch, int pos){

if(all\_ch.count(ch)!=0){

int i = q.front();

if(i!=pos)

return;

q.pop();

remove\_hash(ch,pos);

}

}

string minWindow(string S, string T) {

string ret;

bool found =false;

if(T.length()==0)

return "";

if(S.length()==0)

return "";

init(T);

//find all char

int begin=0;

int end=0;

int min\_len = S.length()+1;

int min\_b=0;

char last\_ch='#';

int last\_pos;

while(1){

while(!isAllExisted()&&end<(int)S.length()){

end++;

insertChar(S[end-1],end-1);

}

while(isAllExisted()){

begin = q.front();

last\_pos= begin;

last\_ch = S[begin];

if(end-begin<min\_len){

found = true;

min\_len = end -begin;

min\_b = begin;

}

deleteChar(last\_ch,last\_pos);

}

if(end>=(int)S.length()){

break;

}

}

if(found)

return S.substr(min\_b,min\_len);

return "";

}

};.

# Combinations

## 题目

. Given two integers n and k, return all possible combinations of k numbers out of 1 ... n.

For example,

If n = 4 and k = 2, a solution is:

[

[2,4],

[3,4],

[2,3],

[1,2],

[1,3],

[1,4],

].

## 解法

### 分析

 .简单递归即可完成.

### 程序

. class Solution {

public:

void recurr(

vector<int> &one,

vector<vector<int> > &ret,

int n,

int curr,

int seat)

{

if(seat>(n+1-curr))

return;

if(seat==0){

ret.push\_back(one);

return;

}

one.push\_back(curr);

recurr(one,ret,n,curr+1,seat-1);

one.pop\_back();

recurr(one,ret,n,curr+1,seat);

}

vector<vector<int> > combine(int n, int k) {

vector<vector<int> > ret;

if(k>n)

k=n;

vector<int> one;

recurr(one,ret,n,1,k);

return ret;

}

};.

# Subsets

## 题目

. Given a set of distinct integers, S, return all possible subsets.

Note:

Elements in a subset must be in non-descending order.

The solution set must not contain duplicate subsets.

For example,

If S = [1,2,3], a solution is:

[

[3],

[1],

[2],

[1,2,3],

[1,3],

[2,3],

[1,2],

[]

].

## 解法

### 分析

 .简单递归.

### 程序

. class Solution {

public:

void recurr(

vector<bool> &map,

vector<int> &S,

vector<vector<int> > &ret,

int curr

)

{

if(curr>=(int)S.size()){

vector<int> one;

for(int i=0;i<(int)S.size();i++){

if(map[i]){

one.push\_back(S[i]);

}

}

ret.push\_back(one);

return;

}

for(int i=0;i<2;i++){

map[curr]=i;

recurr(map,S,ret,curr+1);

}

}

vector<vector<int> > subsets(vector<int> &S) {

vector<vector<int> > ret;

vector<bool> map(false,S.size());

sort(S.begin(),S.end());

recurr(map,S,ret,0);

return ret;

}

};.

# Word Search

## 题目

. Given a 2D board and a word, find if the word exists in the grid.

The word can be constructed from letters of sequentially adjacent cell, where "adjacent" cells are those horizontally or vertically neighboring. The same letter cell may not be used more than once.

For example,

Given board =

[

["ABCE"],

["SFCS"],

["ADEE"]

]

word = "ABCCED", -> returns true,

word = "SEE", -> returns true,

word = "ABCB", -> returns false..

## 解法

### 分析

 .深度优先搜索，算法无难度.

### 程序

. struct DFSNode {

short dir;

short x;

short y;

DFSNode( short in\_dir, short in\_x, short in\_y)

: dir(in\_dir),x(in\_x),y(in\_y){}

};

static vector<DFSNode> path;

class Solution {

public:

bool searchPath(vector<DFSNode> &path,int x, int y){

for(int tmp=0;tmp<(int)path.size();tmp++){

if(path[tmp].x==x&&path[tmp].y==y)

{

return true;

}

}

return false;

}

bool DFS(vector<vector<char> > &board,vector<DFSNode> &path,string &word){

while(!path.empty()){

int top=path.size();

if(top>=(int)word.length()){

return true;

}

int dir= path[top-1].dir + 1;

if(dir>3){

path.pop\_back();

continue;

}

int x=path[top-1].x;

int y=path[top-1].y;

switch(dir){

case 0:

x-=1;

break;

case 1:

y-=1;

break;

case 2:

x+=1;

break;

case 3:

y+=1;

break;

default:

break;

}

if(

(x>=0)&&(x<(short)board.size())

&&(y>=0)&&(y<(short)board[0].size())

&&(board[x][y]==word[top])

&&(!searchPath(path,x,y))

)

{

path[top-1].dir = dir;

path.push\_back(DFSNode(-1,x,y));

}

else

{

path[top-1].dir = dir;

}

}

return false;

}

bool exist(vector<vector<char> > &board, string word) {

path.clear();

if(word.length()==0)

return true;

if(board.size()==0)

return false;

if(board[0].size()==0)

return false;

int h=board.size();

int w=board[0].size();

if(word.length()>(h\*w))

return false;

for(int i=0;i<h;i++){

for(int j=0;j<w;j++){

if(board[i][j]==word[0]){

path.push\_back(DFSNode(-1,i,j));

if(DFS(board,path,word))

return true;

}

}

}

return false;

}

};.

# Remove Duplicates from Sorted Array II

## 题目

. Follow up for "Remove Duplicates":

What if duplicates are allowed at most twice?

For example,

Given sorted array A = [1,1,1,2,2,3],

Your function should return length = 5, and A is now [1,1,2,2,3]..

## 解法

### 分析

 .无难度.

### 程序

. class Solution {

public:

int removeDuplicates(int A[], int n) {

int ret;

int read;

int write;

int twice=0;

if(n==0)

return 0;

read=write=0;

while(1){

read++;

if(read>=n)

break;

if(A[read]==A[write]){

if(twice==0){

write++;

twice=1;

if(read!=write)

A[write]=A[read];

}

else{

continue;

}

}

else{

write++;

twice=0;

if(read!=write)

A[write]=A[read];

}

}

return write+1;

}

};.

# Search in Rotated Sorted Array II

## 题目

. Follow up for "Search in Rotated Sorted Array":

What if duplicates are allowed?

Would this affect the run-time complexity? How and why?

Write a function to determine if a given target is in the array..

## 解法

### 分析

 .二分加遍历.

### 程序

. class Solution {

public:

void findPosition(int A[], int n, int &pos){

int l,r;

l=0;r=n-1;

while(l<r){

if(A[r]>A[l]){

pos=r;

return;

}

if((r-l)<10)

{

pos=r;

for(int j=l;j<r;j++){

if(A[j]>A[j+1]){

pos=j;

break;

}

}

return;

}

int tmp=(l+r)/2;

if(A[tmp]>A[tmp+1]){

pos=tmp;

return;

}

if(A[tmp-1]>A[tmp]){

pos=tmp-1;

return;

}

if(A[tmp]>A[0])

{

l=tmp;

}

else if(A[tmp]<A[0])

{

r=tmp;

}

else//A[tmp]==A[0]

{

int i=tmp;

for(i=tmp;i>l;i--){

if(A[i-1]>A[i])

{

pos = i-1;

return;

}

}

l=tmp;

//search all left half

//1. a small part of left half is target

//2. all left half has same value

}

}

}

int searchTarget(int A[],int begin,int end, int target){

while(begin<end){

if(A[begin]==target)

return begin;

else if(A[end-1]==target)

return end-1;

else if((end-begin)<10){

for(int i=begin;i<end;i++){

if(A[i]==target){

return i;

}

}

return -1;

}

int tmp=(end+begin)/2;

if(A[tmp]==target){

return tmp;

}

else if(A[tmp]>target){

end=tmp;

}

else{

begin=tmp+1;

}

}

return -1;

}

bool search(int A[], int n, int target) {

if(n<=0)return -1;

if(n==1){

if(A[0]==target)

return true;

return false;

}

int pos;

findPosition(A,n,pos);

if(target==A[0]){

return true;

}

else if(target>A[0]){

return (-1!=searchTarget(A,0,pos+1,target));

}

return (-1!=searchTarget(A,pos+1,n,target));

}

};.

# Remove Duplicates from Sorted List

## 题目

.. Given a sorted linked list, delete all duplicates such that each element appear only once.

For example,

Given 1->1->2, return 1->2.

Given 1->1->2->3->3, return 1->2->3.

## 解法

### 分析

 .很简单.

### 程序

.. /\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

class Solution {

public:

ListNode \*deleteDuplicates(ListNode \*head) {

ListNode\* step=head;

if(step==NULL)

return head;

while(step!=NULL&&step->next!=NULL){

if(step->next->val==step->val){

ListNode\* tmp=step->next;

step->next = step->next->next;

delete(tmp);

}

else{

step = step->next;

}

}

return head;

}

};

# Remove Duplicates from Sorted List II

## 题目

. Given a sorted linked list, delete all nodes that have duplicate numbers, leaving only distinct numbers from the original list.

For example,

Given 1->2->3->3->4->4->5, return 1->2->5.

Given 1->1->1->2->3, return 2->3.

## 解法

### 分析

 .分解为两个链表，单个元素链表和重复元素链表

### 程序

. /\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

class Solution {

public:

ListNode \*deleteDuplicates(ListNode \*head) {

ListNode \*p1\_head,\*p1\_tail;//single

ListNode \*p2\_head,\*p2\_tail;//duplicate

p1\_head = p1\_tail = NULL;

p2\_head = p2\_tail = NULL;

ListNode\* tmp=head;

while(tmp!=NULL){

if(tmp->next!=NULL&&tmp->val==tmp->next->val){

if(p2\_head==NULL){

p2\_head=p2\_tail=tmp;

tmp=tmp->next;

p2\_tail->next = NULL;

}

else{

p2\_tail->next=tmp;

p2\_tail=tmp;

tmp=tmp->next;

p2\_tail->next=NULL;

}

}

else if(p2\_tail!=NULL&&tmp->val==p2\_tail->val){

p2\_tail->next=tmp;

p2\_tail=tmp;

tmp=tmp->next;

p2\_tail->next=NULL;

}

else{

if(p1\_head==NULL){

p1\_head=p1\_tail=tmp;

tmp=tmp->next;

p1\_tail->next = NULL;

}

else{

p1\_tail->next=tmp;

p1\_tail=tmp;

tmp=tmp->next;

p1\_tail->next=NULL;

}

}

}

while(p2\_head!=NULL){

ListNode\* tmp=p2\_head;

p2\_head=p2\_head->next;

delete(tmp);

}

return p1\_head;

}

};

# Largest Rectangle in Histogram

## 题目

. Given n non-negative integers representing the histogram's bar height where the width of each bar is 1, find the area of largest rectangle in the histogram.



Above is a histogram where width of each bar is 1, given height = [2,1,5,6,2,3].



The largest rectangle is shown in the shaded area, which has area = 10 unit.

For example,

Given height = [2,1,5,6,2,3],

return 10.

## 解法

### 分析

 .这一题很有趣，网上搜到的一个程序，方法被人大量转载。但是我看懂了的解释很少。后来是看了一个老外的解释，总是搞明白了。现在找不到老外的原文了。

实际想法是，针对每个桩子，如果能以它的高度为最低高度，向左向右尽可能延伸，计算出一个最大面积，则最后的结果，必定在这n个面积中。

在上面的图中，

第一个桩子无法延伸。（左边没有桩子了，右边的比它矮）

第二个桩子可以延伸到头。（两侧所有桩子都比它高，以本桩为最低高度，完全可行）

第三个桩子只能向右延伸一个。（左侧比它矮，右侧只有一个比它高）

……

以此类推。

如果这些计算可以在O(n)复杂度内完成，结果就可以在O(n)时间内出来了。

方法是，用一个堆栈，记录前面比当前桩子矮的桩子的位置。如果当前桩子比栈顶的要高，则栈顶的桩子可以出栈了（它能涵盖的面积可以在这个时候通过栈内前一个元素的位置，以及当前桩子的位置直接计算出来）。

这个方法非常巧妙。

### 程序

. class Solution {

public:

int largestRectangleArea(vector<int> &height) {

stack<int> s;

if(height.empty())

return 0;

int maxResult = height[0];

int i=0;

int n=height.size();

int tp,tmp;

while(i<n){

if(s.empty()||height[s.top()]<=height[i]){

s.push(i);

i++;

}

else{

tp = s.top();

s.pop();

if(s.empty())

tmp = i\*height[tp];

else

tmp = (i - s.top() - 1)\*height[tp];

if(tmp > maxResult){

maxResult = tmp;

}

}

}

while(!s.empty()){

tp = s.top();

s.pop();

if(s.empty()){

tmp = i\*height[tp];

}

else{

tmp = (i-s.top()-1)\*height[tp];

}

if(tmp > maxResult){

maxResult = tmp;

}

}

return maxResult;

}

};

# Maximal Rectangle

## 题目

. Given a 2D binary matrix filled with 0's and 1's, find the largest rectangle containing all ones and return its area.

## 解法

### 分析

 .不详细记载思路，记一点提示，这一题可以通过转化，变形为上一题。复杂度为O(n^2)

### 程序

. class Solution {

public:

int largestRectangleArea(vector<int> &height) {

stack<int> s;

if(height.empty())

return 0;

int maxResult = height[0];

int i=0;

int n=height.size();

int tp,tmp;

while(i<n){

if(s.empty()||height[s.top()]<=height[i]){

s.push(i);

i++;

}

else{

tp = s.top();

s.pop();

if(s.empty())

tmp = i\*height[tp];

else

tmp = (i - s.top() - 1)\*height[tp];

if(tmp > maxResult){

maxResult = tmp;

}

}

}

while(!s.empty()){

tp = s.top();

s.pop();

if(s.empty()){

tmp = i\*height[tp];

}

else{

tmp = (i-s.top()-1)\*height[tp];

}

if(tmp > maxResult){

maxResult = tmp;

}

}

return maxResult;

}

int maximalRectangle(vector<vector<char> > &matrix) {

int x,y;

y=matrix.size();

if(y==0)

return 0;

x=matrix[0].size();

if(x==0)

return 0;

vector<vector<int> > l\_matrix;

vector<int> one;

for(int i=0;i<x;i++){

one.push\_back(0);

}

for(int j=0;j<y;j++){

for(int i=0;i<x;i++){

if(matrix[j][i]=='1')

one[i]=1;

else

one[i]=0;

}

l\_matrix.push\_back(one);

}

for(int j=1;j<y;j++){

for(int i=0;i<x;i++){

if(l\_matrix[j][i]==1){

l\_matrix[j][i]+=l\_matrix[j-1][i];

}

}

}

int max = 0;

for(int j=0;j<y;j++){

int tmp = largestRectangleArea(l\_matrix[j]);

if(tmp>max)

max = tmp;

}

return max;

}

};

# Partition List

## 题目

. Given a linked list and a value x, partition it such that all nodes less than x come before nodes greater than or equal to x.

You should preserve the original relative order of the nodes in each of the two partitions.

For example,

Given 1->4->3->2->5->2 and x = 3,

return 1->2->2->4->3->5.

## 解法

### 分析

 .遍历一边，分拆为两个链表，再合并。

### 程序

. /\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

class Solution {

public:

ListNode \*partition(ListNode \*head, int x) {

bool foundFirst = false;

bool foundSec = false;

ListNode\* cursor = head;

ListNode\* first = NULL;

ListNode\* second = NULL;

ListNode\* newHead = NULL;

ListNode\* newHead2 = NULL;

while(cursor!=NULL){

if(cursor->val<x){

if(newHead==NULL)

newHead = cursor;

foundFirst = true;

first = cursor;

}

else{

if(newHead2==NULL)

newHead2 = cursor;

foundSec = true;

second = cursor;

}

cursor = cursor->next;

if(foundFirst&&foundSec){

break;

}

else{

}

}

if(cursor==NULL&&(!foundFirst||!foundSec))

return head;

while(cursor!=NULL){

if(cursor->val<x){

first->next = cursor;

first = cursor;

}

else{

second->next = cursor;

second = cursor;

}

cursor = cursor->next;

}

first->next = newHead2;

second->next = NULL;

return newHead;

}

};

# Scramble String

## 题目

. Given a string s1, we may represent it as a binary tree by partitioning it to two non-empty substrings recursively.

Below is one possible representation of s1 = "great":

great

/ \

gr eat

/ \ / \

g r e at

/ \

a t

To scramble the string, we may choose any non-leaf node and swap its two children.

For example, if we choose the node "gr" and swap its two children, it produces a scrambled string "rgeat".

rgeat

/ \

rg eat

/ \ / \

r g e at

/ \

a t

We say that "rgeat" is a scrambled string of "great".

Similarly, if we continue to swap the children of nodes "eat" and "at", it produces a scrambled string "rgtae".

rgtae

/ \

rg tae

/ \ / \

r g ta e

/ \

t a

We say that "rgtae" is a scrambled string of "great".

Given two strings s1 and s2 of the same length, determine if s2 is a scrambled string of s1.

## 解法

### 分析

 .利用递归，如果是Scramble，必定在某一点分割S1后，前后两个部分，分别是Scramble。

### 程序

. class Solution {

public:

bool isScramble(string s1, string s2) {

int l1 = s1.length();

int l2 = s2.length();

if(l1!=l2)

return false;

if(l1==1){

if(s1[0]==s2[0])

return true;

else

return false;

}

if(l1==2){

if(s1==s2)

return true;

else if(s1[0]==s2[1]&&s1[1]==s2[0])

return true;

else

return false;

}

if(s1==s2)

return true;

string t1(s1);

string t2(s2);

sort(t1.begin(),t1.end());

sort(t2.begin(),t2.end());

if(t1!=t2)

return false;

for(int i=1;i<l1;i++){

string s11,s12,s21,s22;

s11.assign(s1.substr(0,i));

s12.assign(s1.substr(i,l1-i));

s21.assign(s2.substr(0,l1-i));

s22.assign(s2.substr(l1-i,i));

if(isScramble(s11,s22)&&isScramble(s12,s21))

return true;

s11.assign(s1.substr(0,i));

s12.assign(s1.substr(i,l1-i));

s21.assign(s2.substr(0,i));

s22.assign(s2.substr(i,l1-i));

if(isScramble(s11,s21)&&isScramble(s12,s22))

return true;

}

return false;

}

};

# Merge Sorted Array

## 题目

. Given two sorted integer arrays A and B, merge B into A as one sorted array.

Note:

You may assume that A has enough space (size that is greater or equal to m + n) to hold additional elements from B. The number of elements initialized in A and B are m and n respectively.

## 解法

### 分析

 .从尾部向前填充A数组。

### 程序

. class Solution {

public:

void merge(int A[], int m, int B[], int n) {

if(n==0)

return;

if(m==0){

for(int i=0;i<n;i++){

A[i]=B[i];

}

return;

}

int a\_end = m-1;

int b\_end = n-1;

int t\_end = m+n-1;

while(b\_end>=0){

if(a\_end<0){

A[t\_end]=B[b\_end];

b\_end --;

}

else if(A[a\_end]>B[b\_end]){

A[t\_end]=A[a\_end];

a\_end --;

}

else{

A[t\_end]=B[b\_end];

b\_end --;

}

t\_end --;

}

}

};

# Gray Code

## 题目

.

The gray code is a binary numeral system where two successive values differ in only one bit.

Given a non-negative integer n representing the total number of bits in the code, print the sequence of gray code. A gray code sequence must begin with 0.

For example, given n = 2, return [0,1,3,2]. Its gray code sequence is:

00 - 0

01 - 1

11 - 3

10 - 2

Note:

For a given n, a gray code sequence is not uniquely defined.

For example, [0,2,3,1] is also a valid gray code sequence according to the above definition.

For now, the judge is able to judge based on one instance of gray code sequence. Sorry about that.

## 解法

### 分析

 .本题的关键其实是找到一个合适的派生规律。找到了，编程很简单。

0

1

11

10

110

111

101

100

1100

1101

1111

1110

1010

1011

1001

1000

如果还是看不出来，把红色的1全部去掉，再观察上面16个二进制数的排列特点。

### 程序

. class Solution {

public:

vector<int> grayCode(int n) {

vector<int> ret;

if(n<0)

return ret;

ret.push\_back(0);

if(n==0)

return ret;

ret.push\_back(1);

if(n==1)

return ret;

for(int i=1;i<n;i++){

int bit = 1<<i;

int j = ret.size();

for(;j>0;j--){

int one = ret[j-1]+bit;

ret.push\_back(one);

}

}

return ret;

}

};

# Decode Ways

## 题目

. A message containing letters from A-Z is being encoded to numbers using the following mapping:

'A' -> 1

'B' -> 2

...

'Z' -> 26

Given an encoded message containing digits, determine the total number of ways to decode it.

For example,

Given encoded message "12", it could be decoded as "AB" (1 2) or "L" (12).

The number of ways decoding "12" is 2.

## 解法

### 分析

 .典型的动态规划

### 程序

. static int d[2200];

class Solution {

public:

bool oneCharValid(char ch){

if(ch>'0'&&ch<='9')

return true;

return false;

}

bool twoCharValid(char ch1,char ch2){

if(ch1>'9'||ch1<='0')

return false;

if(ch2>'9'||ch2<'0')

return false;

int i;

i=(ch1-'0')\*10+ch2-'0';

if(i<=26&&i>0)

return true;

return false;

}

int numDecodings(string s) {

d[0]=1;

if(oneCharValid(s[0])){

d[1]=1;

}

else{

return 0;

}

for(int i=2;i<=(int)s.length();i++){

int tmp=0;

if(oneCharValid(s[i-1])){

tmp += d[i-1];

}

if(twoCharValid(s[i-2],s[i-1])){

tmp += d[i-2];

}

d[i]=tmp;

}

return d[s.length()];

}

};

# Subsets II

## 题目

. Given a collection of integers that might contain duplicates, S, return all possible subsets.

Note:

Elements in a subset must be in non-descending order.

The solution set must not contain duplicate subsets.

For example,

If S = [1,2,2], a solution is:

[

[2],

[1],

[1,2,2],

[2,2],

[1,2],

[]

]

## 解法

### 分析

 .递归，思路与Subsets相同，唯一的问题是防重。我想到一个有趣的方法，也很有效。

先把S排序，递归的时候，如果遇到重复元素，一旦前面有item未选，后面的都不能选。也就是说，如果遇到8,8,8，则选择的时候，只有100, 110, 111，不会出现001, 010, 101这样的选法。

### 程序

. class Solution {

public:

void recurr(vector<int> &S, bool\* select, int curr, vector<vector<int> > &ret){

if(curr>=(int)S.size()){

vector<int> one;

for(int i=0;i<(int)S.size();i++){

if(select[i])

one.push\_back(S[i]);

}

ret.push\_back(one);

return;

}

select[curr] = false;

recurr(S,select,curr+1,ret);

if(curr>0&&S[curr]==S[curr-1]&&select[curr-1]==false){

}

else{

select[curr] = true;

recurr(S,select,curr+1,ret);

}

}

vector<vector<int> > subsetsWithDup(vector<int> &S) {

vector<vector<int> > ret;

sort(S.begin(),S.end());

bool select[S.size()];

recurr(S, select, 0, ret);

return ret;

}

};

# Reverse Linked List II

## 题目

. Reverse a linked list from position m to n. Do it in-place and in one-pass.

For example:

Given 1->2->3->4->5->NULL, m = 2 and n = 4,

return 1->4->3->2->5->NULL.

Note:

Given m, n satisfy the following condition:

1 ≤ m ≤ n ≤ length of list.

## 解法

### 分析

 .编程题，算法无难点。

### 程序

. /\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

class Solution {

public:

ListNode \*reverseBetween(ListNode \*head, int m, int n) {

ListNode \*p1,\*p2,\*first,\*last;

p1=p2=first=last=NULL;

if(m>=n)

return head;

if(head==NULL)

return head;

if(m==1){

first = head;

}

else{

p1 = head;

int count = m-2;

while(count>0){

if(p1!=NULL){

p1 = p1->next;

}

else

{

return head;

}

count --;

}

if(p1==NULL){

return head;

}

else{

first = p1->next;

}

}

ListNode \*prev,\*curr,\*next;

prev = curr = next = NULL;

prev=first;

if(prev!=NULL){

curr = prev->next;

}

if(curr!=NULL){

next = curr->next;

}

if(prev==NULL||curr==NULL||next==NULL){

if(curr!=NULL){

if(p1!=NULL){

p1->next = curr;

curr->next = prev;

prev->next = NULL;

}

else{

head = curr;

curr->next = prev;

prev->next = NULL;

}

}

return head;

}

for(int i=m+1;i<=n;i++){

curr->next = prev;

if(i<n){

if(next==NULL)

break;

prev = curr;

curr = next;

next = curr->next;

}

}

last = curr;

p2 = next;

if(p1!=NULL){

p1->next = last;

first->next = p2;

}

else{

head = last;

first->next = p2;

}

return head;

}

};

# Restore IP Addresses

## 题目

. Given a string containing only digits, restore it by returning all possible valid IP address combinations.

For example:

Given "25525511135",

return ["255.255.11.135", "255.255.111.35"]. (Order does not matter)

## 解法

### 分析

 .简单递归

### 程序

. class Solution {

public:

bool isValidStr(string s){

if(s.length()>3)

return false;

if(s[0]=='0'){

if(s.length()==1)

return true;

else

return false;

}

int ip = atoi(s.c\_str());

if(ip>255||ip==0)

return false;

return true;

}

void recurr(string s, int pos, int field, string IP[], vector<string> &ret){

int l = s.length();

if(pos>=l)

return;

if(field==3){

string tmp = s.substr(pos,l-pos);

if(isValidStr(tmp)){

IP[3] = tmp;

ret.push\_back(

IP[0]+"."+IP[1]+"."+IP[2]+"."+IP[3]

);

}

return;

}

for(int i=1;i<=3;i++){

string tmp = s.substr(pos,i);

if(isValidStr(tmp)){

IP[field] = tmp;

recurr(s, pos+i, field+1, IP, ret);

}

}

return;

}

vector<string> restoreIpAddresses(string s) {

vector<string> ret;

string IPStr[4];

recurr(s,0,0,IPStr,ret);

return ret;

}

};

# Binary Tree Inorder Traversal

## 题目

. Given a binary tree, return the inorder traversal of its nodes' values.

For example:

Given binary tree {1,#,2,3},

1

\

2

/

3

return [1,3,2].

Note: Recursive solution is trivial, could you do it iteratively?

## 解法

### 分析

 .算法不难。

### 程序

. /\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

struct StackNode{

TreeNode\* tree;

int visited;

StackNode():tree(NULL),visited(0){};

StackNode(TreeNode\* t):tree(t),visited(0){};

};

class Solution {

public:

vector<int> inorderTraversal(TreeNode \*root) {

stack<StackNode> s;

vector<int> res;

if(root==NULL)

return res;

s.push(StackNode(root));

while(!s.empty()){

StackNode tp = s.top();

s.pop();

TreeNode \*tree = tp.tree;

if(tp.visited==0){

tp.visited=1;

s.push(tp);

if(tree->left!=NULL){

s.push(StackNode(tree->left));

}

continue;

}

res.push\_back(tree->val);

if(tree->right!=NULL){

s.push(StackNode(tree->right));

}

}

return res;

}

};

# Unique Binary Search Trees

## 题目

. Given n, how many structurally unique BST's (binary search trees) that store values 1...n?

For example,

Given n = 3, there are a total of 5 unique BST's.

1 3 3 2 1

\ / / / \ \

3 2 1 1 3 2

/ / \ \

2 1 2 3

## 解法

### 分析

 .应该是卡塔兰数，动态规划可以求得。

### 程序

. static const int MAX\_SIZE=800;

static int gResult[MAX\_SIZE][MAX\_SIZE];

class Solution {

public:

int numTrees(int n) {

for(int i=0;i<=n;i++){

for(int j=0;j<=n-i;j++){

//gResult[j][i]

if(i==0||i==1)

gResult[j][i]=1;

else{

gResult[j][i]=0;

for(int k=0;k<i;k++){

gResult[j][i]+=gResult[j][k]\*gResult[j+k+1][i-k-1];

}

}

}

}

return gResult[0][n];

}

};

# Unique Binary Search Trees II

## 题目

. Given n, generate all structurally unique BST's (binary search trees) that store values 1...n.

For example,

Given n = 3, your program should return all 5 unique BST's shown below.

1 3 3 2 1

\ / / / \ \

3 2 1 1 3 2

/ / \ \

2 1 2 3

## 解法

### 分析

 .生成所有的树。仍旧采用动态规划法。为了简便，最后产生的树实际上是相互纠结的，而不是相互独立的。

### 程序

. static const int MAX\_SIZE=100;

static vector<TreeNode \*> gResult[MAX\_SIZE][MAX\_SIZE];

class Solution {

public:

void clearResult(){

for(int i=0;i<MAX\_SIZE;i++){

for(int j=0;j<MAX\_SIZE;j++){

while(!gResult[i][j].empty()){

TreeNode \* tmp = gResult[i][j].back();

gResult[i][j].pop\_back();

delete(tmp);

}

}

}

}

vector<TreeNode \*> generateTrees(int n) {

if(n==0){

vector<TreeNode \*> ret;

ret.push\_back(NULL);

return ret;

}

clearResult();

for(int i=0;i<=n;i++){

for(int j=0;j<=n-i;j++){

//gResult[j][i]

if(i==0){

}

else if(i==1){

TreeNode \* tmp= new TreeNode(j+1);

gResult[j][i].push\_back(tmp);

}

else{

for(int k=0;k<i;k++){

int left = gResult[j][k].size();

int right = gResult[j+k+1][i-k-1].size();

bool l\_null=false;

bool r\_null=false;

if(left==0){

l\_null = true;

left = 1;

}

if(right==0){

r\_null = true;

right = 1;

}

for(int m=0;m<left;m++){

for(int m2=0;m2<right;m2++){

TreeNode \* tmp= new TreeNode(j+k+1);

if(!l\_null)

tmp->left = gResult[j][k][m];

if(!r\_null)

tmp->right = gResult[j+k+1][i-k-1][m2];

gResult[j][i].push\_back(tmp);

}

}

}

}

}

}

return gResult[0][n];

}

};

# Interleaving String

## 题目

. Given s1, s2, s3, find whether s3 is formed by the interleaving of s1 and s2.

For example,

Given:

s1 = "aabcc",

s2 = "dbbca",

When s3 = "aadbbcbcac", return true.

When s3 = "aadbbbaccc", return false.

## 解法

### 分析

 .动态规划，d[i][j]定义为，s1[0..i-1]和s2[0..j-1]能否合并出s3[0..i+j-1]

### 程序

. static const int MAX\_SIZE=800;

static bool gResult[MAX\_SIZE][MAX\_SIZE];

class Solution {

public:

bool isInterleave(string s1, string s2, string s3) {

int l1,l2,l3;

l1=s1.length();

l2=s2.length();

l3=s3.length();

if(l3!=(l1+l2))

return false;

if(l3==0)

return true;

for(int i=0;i<=l1;i++){

for(int j=0;j<=l2;j++){

if(i==0&&j==0){

gResult[i][j]=true;

}

else if(i==0){

if(s2[j-1]==s3[j-1]&&gResult[i][j-1])

gResult[i][j]=true;

else

gResult[i][j]=false;

}

else if(j==0){

if(s1[i-1]==s3[i-1]&&gResult[i-1][j])

gResult[i][j]=true;

else

gResult[i][j]=false;

}

else{

if(

(s1[i-1]==s3[i+j-1]&&gResult[i-1][j])

||(s2[j-1]==s3[i+j-1]&&gResult[i][j-1])

)

{

gResult[i][j]=true;

}

else

{

gResult[i][j]=false;

}

}

}

}

return gResult[l1][l2];

}

};

# Validate Binary Search Tree

## 题目

. Given a binary tree, determine if it is a valid binary search tree (BST).

Assume a BST is defined as follows:

The left subtree of a node contains only nodes with keys less than the node's key.

The right subtree of a node contains only nodes with keys greater than the node's key.

Both the left and right subtrees must also be binary search trees.

## 解法

### 分析

 .转成中序遍历列表，看是否递增。

### 程序

. /\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

class Solution {

public:

void BSTToList(TreeNode \*t, vector<TreeNode\*> &l){

if(t->left!=NULL)

BSTToList(t->left,l);

l.push\_back(t);

if(t->right!=NULL)

BSTToList(t->right,l);

}

bool isValidBST(TreeNode \*root) {

if(root==NULL)

return true;

vector<TreeNode\*> array;

BSTToList(root, array);

for(int i=0;i<(int)array.size()-1;i++){

if(array[i]->val>=array[i+1]->val){

array.clear();

return false;

}

}

array.clear();

return true;

}

};

# Recover Binary Search Tree

## 题目

. Two elements of a binary search tree (BST) are swapped by mistake.

Recover the tree without changing its structure.

Note:

A solution using O(n) space is pretty straight forward. Could you devise a constant space solution?

## 解法

### 分析

 .方法同上，转成列表，找顺序不正确的位置。

### 程序

. /\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

class Solution {

public:

void BSTToList(TreeNode \*t, vector<TreeNode\*> &l){

if(t->left!=NULL)

BSTToList(t->left,l);

l.push\_back(t);

if(t->right!=NULL)

BSTToList(t->right,l);

}

void recoverTree(TreeNode \*root) {

if(root==NULL)

return;

vector<TreeNode\*> array;

BSTToList(root, array);

TreeNode\* first=NULL;

TreeNode\* second=NULL;

for(int i=0;i<(int)array.size()-1;i++){

if(array[i]->val>array[i+1]->val){

if(first==NULL){

first = array[i];

second = array[i+1];

}

else{

second = array[i+1];

}

}

}

if(first==NULL)

return;

int tmp = first->val;

first->val = second->val;

second->val = tmp;

return;

}

};

# Same Tree

## 题目

. Given two binary trees, write a function to check if they are equal or not.

Two binary trees are considered equal if they are structurally identical and the nodes have the same value.

## 解法

### 分析

 .递归很简单。不准用递归的话，有点晕。

### 程序

. /\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

class Solution {

public:

bool isSameTree(TreeNode \*p, TreeNode \*q) {

if(p==NULL&&q==NULL)

return true;

else if(p==NULL||q==NULL)

return false;

if(p->val!=q->val)

return false;

if(isSameTree(p->left,q->left)&&isSameTree(p->right,q->right))

return true;

return false;

}

};

# Symmetric Tree

## 题目

. Given a binary tree, check whether it is a mirror of itself (ie, symmetric around its center).

For example, this binary tree is symmetric:

1

/ \

2 2

/ \ / \

3 4 4 3

But the following is not:

1

/ \

2 2

\ \

3 3

Note:

Bonus points if you could solve it both recursively and iteratively.

## 解法

### 分析

 .同样，递归很简单。题目说不用递归可以加分，没想出来好办法。

### 程序

. /\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

class Solution {

public:

bool isTwoTreeSymmetric(TreeNode\* t1, TreeNode\* t2){

if(t1==NULL&&t2==NULL)

return true;

else if(t1==NULL||t2==NULL)

return false;

if(t1->val!=t2->val)

return false;

if(isTwoTreeSymmetric(t1->left, t2->right)

&&isTwoTreeSymmetric(t1->right,t2->left))

return true;

return false;

}

bool isSymmetric(TreeNode \*root) {

if(root==NULL)

return true;

return isTwoTreeSymmetric(root->left, root->right);

}

};

# Binary Tree Level Order Traversal

## 题目

. Given a binary tree, return the level order traversal of its nodes' values. (ie, from left to right, level by level).

For example:

Given binary tree {3,9,20,#,#,15,7},

3

/ \

9 20

/ \

15 7

return its level order traversal as:

[

[3],

[9,20],

[15,7]

]

confused

## 解法

### 分析

 .广度优先。

### 程序

. /\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

struct BFSNode{

TreeNode\* ptr;

int level;

BFSNode(TreeNode\* p, int l): ptr(p), level(l) {}

};

class Solution {

public:

vector<vector<int> > levelOrder(TreeNode \*root) {

vector<vector<int> > ret;

if(root==NULL)

return ret;

vector<int> oneList;

queue<BFSNode> q;

q.push(BFSNode(root, 0));

int lastLevel = 0;

while(!q.empty()){

BFSNode tmp = q.front();

q.pop();

if(tmp.level!=lastLevel){

ret.push\_back(oneList);

oneList.clear();

lastLevel = tmp.level;

}

oneList.push\_back(tmp.ptr->val);

if(tmp.ptr->left!=NULL)

q.push(BFSNode(tmp.ptr->left,tmp.level+1));

if(tmp.ptr->right!=NULL)

q.push(BFSNode(tmp.ptr->right,tmp.level+1));

}

if(oneList.size()!=0){

ret.push\_back(oneList);

oneList.clear();

}

return ret;

}

};

# Binary Tree Zigzag Level Order Traversal

## 题目

. Given a binary tree, return the zigzag level order traversal of its nodes' values. (ie, from left to right, then right to left for the next level and alternate between).

For example:

Given binary tree {3,9,20,#,#,15,7},

3

/ \

9 20

/ \

15 7

return its zigzag level order traversal as:

[

[3],

[20,9],

[15,7]

]

confused what "{1,#,2,3}" means? > read more on how binary tree is serialized on OJ.

OJ's Binary Tree Serialization:

The serialization of a binary tree follows a level order traversal, where '#' signifies a path terminator where no node exists below.

Here's an example:

1

/ \

2 3

/

4

\

5

The above binary tree is serialized as "{1,2,3,#,#,4,#,#,5}".

## 解法

### 分析

 .广度优先加一点额外处理。

### 程序

. /\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

struct BFSNode{

TreeNode\* ptr;

int level;

BFSNode(TreeNode\* p, int l): ptr(p), level(l) {}

};

class Solution {

public:

vector<vector<int> > zigzagLevelOrder(TreeNode \*root) {

vector<vector<int> > ret;

if(root==NULL)

return ret;

vector<int> oneList;

queue<BFSNode> q;

q.push(BFSNode(root, 0));

int lastLevel = 0;

while(!q.empty()){

BFSNode tmp = q.front();

q.pop();

if(tmp.level!=lastLevel){

if(lastLevel%2){

reverse(oneList.begin(),oneList.end());

}

ret.push\_back(oneList);

oneList.clear();

lastLevel = tmp.level;

}

oneList.push\_back(tmp.ptr->val);

if(tmp.ptr->left!=NULL)

q.push(BFSNode(tmp.ptr->left,tmp.level+1));

if(tmp.ptr->right!=NULL)

q.push(BFSNode(tmp.ptr->right,tmp.level+1));

}

if(oneList.size()!=0){

if(lastLevel%2){

reverse(oneList.begin(),oneList.end());

}

ret.push\_back(oneList);

oneList.clear();

}

return ret;

}

};

# Maximum Depth of Binary Tree

## 题目

. Given a binary tree, find its maximum depth.

The maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

## 解法

### 分析

 .递归很简单。

### 程序

. /\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

class Solution {

public:

int maxDepth(TreeNode \*root) {

if(root==NULL)

return 0;

if(root->left==NULL&&root->right==NULL)

return 1;

int d1=maxDepth(root->left);

int d2=maxDepth(root->right);

if(d1>d2)

return d1+1;

return d2+1;

}

};

# Construct Binary Tree from Preorder and Inorder Traversal

## 题目

. Given preorder and inorder traversal of a tree, construct the binary tree.

Note:

You may assume that duplicates do not exist in the tree.

## 解法

### 分析

 .递归，难度不大

### 程序

. /\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

class Solution {

public:

TreeNode \*lBuildTree(

vector<int> &preorder,

int p\_b,

int p\_e,

vector<int> &inorder,

int i\_b,

int i\_e)

{

TreeNode\* ret=NULL;

int l1=p\_e-p\_b;

int l2=i\_e-i\_b;

if(l1!=l2||l1==0)

return NULL;

int mid = preorder[p\_b];

int pos=i\_b;

for(pos=i\_b;pos<i\_e;pos++){

if(inorder[pos]==mid)

break;

}

if(pos>=i\_e){

return NULL;

}

int i\_left = pos-i\_b;

ret = new TreeNode(mid);

ret->left = lBuildTree(preorder, p\_b+1, p\_b+i\_left+1, inorder, i\_b, pos);

ret->right = lBuildTree(preorder, p\_b+i\_left+1, p\_e, inorder, pos+1,i\_e);

return ret;

}

TreeNode \*buildTree(vector<int> &preorder, vector<int> &inorder) {

int l1=preorder.size();

int l2=inorder.size();

if(l1!=l2||l1==0)

return NULL;

return lBuildTree(preorder,0,l1,inorder,0,l2);

}

};

# Construct Binary Tree from Inorder and Postorder Traversal

## 题目

. Given inorder and postorder traversal of a tree, construct the binary tree.

Note:

You may assume that duplicates do not exist in the tree.

## 解法

### 分析

 .和前一题类似。

### 程序

. /\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

class Solution {

public:

TreeNode \*lBuildTree(

vector<int> &inorder, int i\_b, int i\_e,

vector<int> &postorder, int p\_b, int p\_e)

{

TreeNode \*ret=NULL;

if(inorder.size()==0)

return ret;

int l1=i\_e-i\_b;

int l2=p\_e-p\_b;

if((l1!=l2)||(l1==0))

return ret;

int mid=postorder[p\_e-1];

int pos;

for(pos=i\_b;pos<i\_e;pos++){

if(inorder[pos]==mid)

break;

}

if(pos>=i\_e)

return NULL;

int l\_left;

l\_left=pos-i\_b;

//l\_right=i\_e-pos-1;

ret = new TreeNode(mid);

ret->left = lBuildTree(inorder, i\_b, pos, postorder, p\_b, p\_b+l\_left);

ret->right = lBuildTree(inorder, pos+1, i\_e, postorder, p\_b+l\_left, p\_e-1);

return ret;

}

TreeNode \*buildTree(vector<int> &inorder, vector<int> &postorder) {

TreeNode \*ret=NULL;

int l\_i = inorder.size();

int l\_p = postorder.size();

ret = lBuildTree(inorder,0,l\_i,postorder,0,l\_p);

return ret;

}

};

# Binary Tree Level Order Traversal II

## 题目

. Given a binary tree, return the bottom-up level order traversal of its nodes' values. (ie, from left to right, level by level from leaf to root).

For example:

Given binary tree {3,9,20,#,#,15,7},

3

/ \

9 20

/ \

15 7

return its bottom-up level order traversal as:

[

[15,7]

[9,20],

[3],

]

## 解法

### 分析

 .利用vector reverse，很简单。

### 程序

. /\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

struct BFSNode{

TreeNode\* ptr;

int level;

BFSNode(TreeNode\* p, int l):ptr(p), level(l){}

};

class Solution {

public:

vector<vector<int> > levelOrderBottom(TreeNode \*root) {

vector<vector<int> > ret;

if(root==NULL)

return ret;

vector<int> oneList;

queue<BFSNode> bfsList;

int LastLevel=1;

bfsList.push(BFSNode(root,1));

while(!bfsList.empty()){

BFSNode tmp=bfsList.front();

bfsList.pop();

if(tmp.level!=LastLevel){

ret.push\_back(oneList);

oneList.clear();

}

oneList.push\_back(tmp.ptr->val);

if(tmp.ptr->left!=NULL){

bfsList.push(BFSNode(tmp.ptr->left, tmp.level+1));

}

if(tmp.ptr->right!=NULL){

bfsList.push(BFSNode(tmp.ptr->right, tmp.level+1));

}

LastLevel = tmp.level;

}

if(oneList.size()!=0){

ret.push\_back(oneList);

oneList.clear();

}

reverse(ret.begin(),ret.end());

return ret;

}

};

# Convert Sorted Array to Binary Search Tree

## 题目

. Given an array where elements are sorted in ascending order, convert it to a height balanced BST.

## 解法

### 分析

 .每次选取数组中点，先插入树。然后左右递归。

### 程序

. /\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

class Solution {

public:

TreeNode \*lArrayToBST(vector<int> &num, int b, int e){

TreeNode\* ret=NULL;

if(b==e){

return NULL;

}

int mid=(b+e)/2;

ret = new TreeNode(num[mid]);

ret->left = lArrayToBST(num,b,mid);

ret->right = lArrayToBST(num,mid+1,e);

return ret;

}

TreeNode \*sortedArrayToBST(vector<int> &num) {

TreeNode\* ret=NULL;

ret=lArrayToBST(num, 0, num.size());

return ret;

}

};

# Convert Sorted List to Binary Search Tree

## 题目

. Given a singly linked list where elements are sorted in ascending order, convert it to a height balanced BST.

## 解法

### 分析

 .与上一题类似，主要难点是如何快速找出中点位置。数组可以直接计算。链表可以通过两个指针，一个一次走两步，一个一次走一步的方式，较快找到。

### 程序

. /\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

/\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

class Solution {

public:

TreeNode\* listToBST(ListNode\* h, ListNode\* t){

TreeNode\* ret=NULL;

if(h==NULL||h==t)

return ret;

ListNode \*slowPtr, \*fastPtr;

slowPtr=h;

fastPtr=h->next;

while(fastPtr!=NULL&&fastPtr!=t){

slowPtr=slowPtr->next;

fastPtr=fastPtr->next;

if(fastPtr==NULL||fastPtr==t){

break;

}

else{

fastPtr=fastPtr->next;

}

}

ret = new TreeNode(slowPtr->val);

ret->left = listToBST(h,slowPtr);

ret->right = listToBST(slowPtr->next,t);

return ret;

}

TreeNode \*sortedListToBST(ListNode \*head) {

TreeNode\* ret;

ret = listToBST(head, NULL);

return ret;

}

};

# Balanced Binary Tree

## 题目

. Given a binary tree, determine if it is height-balanced.

For this problem, a height-balanced binary tree is defined as a binary tree in which the depth of the two subtrees of every node never differ by more than 1.

## 解法

### 分析

 .

### 程序

. /\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

class Solution {

public:

bool checkBalanced(TreeNode \*root, int &deep){

if(root->left==NULL&&root->right==NULL){

deep=1;

return true;

}

if(root->left==NULL){

int r\_deep;

bool b = checkBalanced(root->right,r\_deep);

deep=r\_deep+1;

if(b&&r\_deep<=1)

return true;

else

return false;

}

if(root->right==NULL){

int l\_deep;

bool b = checkBalanced(root->left,l\_deep);

deep=l\_deep+1;

if(b&&l\_deep<=1)

return true;

else

return false;

}

int l\_deep,r\_deep;

bool b1 = checkBalanced(root->left,l\_deep);

bool b2 = checkBalanced(root->right,r\_deep);

int diff = l\_deep-r\_deep;

if(l\_deep<r\_deep)

deep=r\_deep+1;

else

deep=l\_deep+1;

if(b1&&b2&&diff<=1&&diff>=-1)

return true;

else

return false;

}

bool isBalanced(TreeNode \*root) {

int deep;

if(root==NULL)

return true;

return checkBalanced(root,deep);

}

};

# Minimum Depth of Binary Tree

## 题目

. Given a binary tree, find its minimum depth.

The minimum depth is the number of nodes along the shortest path from the root node down to the nearest leaf node.

## 解法

### 分析

 .递归，很简单。

### 程序

. /\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

class Solution {

public:

int minDepth(TreeNode \*root) {

if(root==NULL)

return 0;

if(root->left==NULL&&root->right==NULL)

return 1;

if(root->left==NULL)

return minDepth(root->right)+1;

if(root->right==NULL)

return minDepth(root->left)+1;

int r1=minDepth(root->left);

int r2=minDepth(root->right);

if(r1>r2)

return r2+1;

return r1+1;

}

};

# Path Sum

## 题目

. Given a binary tree and a sum, determine if the tree has a root-to-leaf path such that adding up all the values along the path equals the given sum.

For example:

Given the below binary tree and sum = 22,

5

/ \

4 8

/ / \

11 13 4

/ \ \

7 2 1

return true, as there exist a root-to-leaf path 5->4->11->2 which sum is 22.

## 解法

### 分析

 .很简单

### 程序

. /\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

class Solution {

public:

bool hasPathSum(TreeNode \*root, int sum) {

if(root==NULL){

return false;

}

if(root->left==NULL&&root->right==NULL&&sum==root->val)

return true;

return hasPathSum(root->left,sum-root->val)||hasPathSum(root->right,sum-root->val);

}

};

# Path Sum II

## 题目

. Given a binary tree and a sum, find all root-to-leaf paths where each path's sum equals the given sum.

For example:

Given the below binary tree and sum = 22,

5

/ \

4 8

/ / \

11 13 4

/ \ / \

7 2 5 1

return

[

[5,4,11,2],

[5,8,4,5]

]

## 解法

### 分析

 .需要探索顶点到每个叶子的路径，除非半路上已经发现超过sum了。广度优先搜索。

### 程序

. /\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

struct BFSNode {

int sum;

TreeNode \*self;

BFSNode():sum(0),self(NULL){};

BFSNode(int x,TreeNode \*s) : sum(x), self(s){}

};

class Solution {

public:

vector<vector<int> > pathSum(TreeNode \*root, int sum) {

vector<vector<int> > res;

if(root==NULL)

return res;

vector<BFSNode> bfs;

map<TreeNode \*, TreeNode \* > parent;

bfs.push\_back(BFSNode(root->val,root));

parent[root] = NULL;

int read = 0;

while(read<(int)bfs.size()){

if(bfs[read].self->left==NULL

&&bfs[read].self->right==NULL

&&bfs[read].sum==sum)

{

//find one

vector<int> one;

TreeNode\* cur = bfs[read].self;

while(cur!=NULL){

one.push\_back(cur->val);

cur=parent[cur];

}

reverse(one.begin(),one.end());

res.push\_back(one);

}

else {

if(bfs[read].self->left!=NULL){

int val = bfs[read].sum+bfs[read].self->left->val;

bfs.push\_back(BFSNode(val,bfs[read].self->left));

parent.insert(make\_pair(bfs[read].self->left,bfs[read].self));

}

if(bfs[read].self->right!=NULL){

int val = bfs[read].sum+bfs[read].self->right->val;

bfs.push\_back(BFSNode(val,bfs[read].self->right));

parent.insert(make\_pair(bfs[read].self->right,bfs[read].self));

}

}

read++;

}

return res;

}

};

# Flatten Binary Tree to Linked List

## 题目

. Given a binary tree, flatten it to a linked list in-place.

For example,

Given

1

/ \

2 5

/ \ \

3 4 6

The flattened tree should look like:

1

\

2

\

3

\

4

\

5

\

6

click to show hints.

Hints:

If you notice carefully in the flattened tree, each node's right child points to the next node of a pre-order traversal.

## 解法

### 分析

 .前序遍历，转为链表

### 程序

. /\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

static vector<TreeNode\*> m\_list;

class Solution {

public:

void DFS(TreeNode \*root, vector<TreeNode\*> &list){

list.push\_back(root);

if(root->left!=NULL)

DFS(root->left,list);

if(root->right!=NULL)

DFS(root->right,list);

}

void flatten(TreeNode \*root) {

if(root==NULL)

return;

m\_list.clear();

DFS(root,m\_list);

int i;

for(i=0;i<(int)m\_list.size()-1;i++){

m\_list[i]->left = NULL;

m\_list[i]->right = m\_list[i+1];

}

m\_list[i]->left = NULL;

m\_list[i]->right = NULL;

}

};

# Distinct Subsequences

## 题目

. Given a string S and a string T, count the number of distinct subsequences of T in S.

A subsequence of a string is a new string which is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (ie, "ACE" is a subsequence of "ABCDE" while "AEC" is not).

Here is an example:

S = "rabbbit", T = "rabbit"

Return 3.

## 解法

### 分析

 .利用动态规划，d[i][j]表示S[0..i], T[0..j]有多少种匹配方式。

### 程序

. static const int MAX\_S = 11000;

static const int MAX\_T = 60;

static int gResult[MAX\_S][MAX\_T];

class Solution {

public:

void initResult(int a,int b){

for(int i=0;i<a;i++){

for(int j=0;j<b;j++){

gResult[i][j]=-1;

}

}

}

int numDistinct(string S, string T) {

int s\_len = S.length();

int t\_len = T.length();

if(s\_len==0||t\_len==0)

return 0;

initResult(s\_len,t\_len);

for(int j=0;j<t\_len;j++){

for(int i=0;i<s\_len;i++){

if(j==0){

if(i==0){

if(S[0]==T[0])

gResult[0][0] = 1;

else

gResult[0][0] = 0;

}

else{

if(S[i]==T[j]){

gResult[i][j] = gResult[i-1][j] +1;

}

else{

gResult[i][j] = gResult[i-1][j];

}

}

}

else{

if(i<j){

gResult[i][j] = 0;

}

else{

if(S[i]==T[j]){

gResult[i][j] = gResult[i-1][j] + gResult[i-1][j-1];

}

else{

gResult[i][j] = gResult[i-1][j];

}

}

}

}

}

return gResult[s\_len-1][t\_len-1];

}

};

# Populating Next Right Pointers in Each Node

## 题目

. Given a binary tree

struct TreeLinkNode {

TreeLinkNode \*left;

TreeLinkNode \*right;

TreeLinkNode \*next;

}

Populate each next pointer to point to its next right node. If there is no next right node, the next pointer should be set to NULL.

Initially, all next pointers are set to NULL.

Note:

You may only use constant extra space.

You may assume that it is a perfect binary tree (ie, all leaves are at the same level, and every parent has two children).

For example,

Given the following perfect binary tree,

1

/ \

2 3

/ \ / \

4 5 6 7

After calling your function, the tree should look like:

1 -> NULL

/ \

2 -> 3 -> NULL

/ \ / \

4->5->6->7 -> NULL

## 解法

### 分析

 .利用广度优先可解决。但是不知道是否满足只使用常量级别内存空间，优化后的广度优先，也需要使用树宽度的两倍空间。

### 程序

. /\*\*

\* Definition for binary tree with next pointer.

\* struct TreeLinkNode {

\* int val;

\* TreeLinkNode \*left, \*right, \*next;

\* TreeLinkNode(int x) : val(x), left(NULL), right(NULL), next(NULL) {}

\* };

\*/

struct BFSNode{

int deep;

TreeLinkNode\* tree;

BFSNode():deep(0),tree(NULL){};

BFSNode(int i,TreeLinkNode\* t):deep(i),tree(t){};

};

class Solution {

public:

void connect(TreeLinkNode \*root) {

if(root==NULL)

return;

vector<BFSNode> bfs;

bfs.push\_back(BFSNode(0,root));

int r;

r=0;

while(r<(int)bfs.size()){

TreeLinkNode\* t;

t=bfs[r].tree;

if(t->left!=NULL){

bfs.push\_back(BFSNode(bfs[r].deep+1,t->left));

}

if(t->right!=NULL){

bfs.push\_back(BFSNode(bfs[r].deep+1,t->right));

}

r++;

}

int i;

for(i=0;i<(int)bfs.size()-1;i++){

TreeLinkNode \*t1,\*t2;

t1=bfs[i].tree;

t2=bfs[i+1].tree;

if(bfs[i].deep==bfs[i+1].deep){

t1->next = t2;

}

else

{

t1->next = NULL;

}

}

bfs[i].tree->next = NULL;

return;

}

};

# Populating Next Right Pointers in Each Node II

## 题目

. Follow up for problem "Populating Next Right Pointers in Each Node".

What if the given tree could be any binary tree? Would your previous solution still work?

Note:

You may only use constant extra space.

For example,

Given the following binary tree,

1

/ \

2 3

/ \ \

4 5 7

After calling your function, the tree should look like:

1 -> NULL

/ \

2 -> 3 -> NULL

/ \ \

4-> 5 -> 7 -> NULL

## 解法

### 分析

 .方法同上一题

### 程序

. /\*\*

\* Definition for binary tree with next pointer.

\* struct TreeLinkNode {

\* int val;

\* TreeLinkNode \*left, \*right, \*next;

\* TreeLinkNode(int x) : val(x), left(NULL), right(NULL), next(NULL) {}

\* };

\*/

struct BFSNode{

int deep;

TreeLinkNode\* tree;

BFSNode():deep(0),tree(NULL){};

BFSNode(int i,TreeLinkNode\* t):deep(i),tree(t){};

};

class Solution {

public:

void connect(TreeLinkNode \*root) {

if(root==NULL)

return;

vector<BFSNode> bfs;

bfs.push\_back(BFSNode(0,root));

int r;

r=0;

while(r<(int)bfs.size()){

TreeLinkNode\* t;

t=bfs[r].tree;

if(t->left!=NULL){

bfs.push\_back(BFSNode(bfs[r].deep+1,t->left));

}

if(t->right!=NULL){

bfs.push\_back(BFSNode(bfs[r].deep+1,t->right));

}

r++;

}

int i;

for(i=0;i<(int)bfs.size()-1;i++){

TreeLinkNode \*t1,\*t2;

t1=bfs[i].tree;

t2=bfs[i+1].tree;

if(bfs[i].deep==bfs[i+1].deep){

t1->next = t2;

}

else

{

t1->next = NULL;

}

}

bfs[i].tree->next = NULL;

return;

}

};

# Pascal's Triangle

## 题目

. Given numRows, generate the first numRows of Pascal's triangle.

For example, given numRows = 5,

Return

[

[1],

[1,1],

[1,2,1],

[1,3,3,1],

[1,4,6,4,1]

]

## 解法

### 分析

 .直接迭代

### 程序

. class Solution {

public:

vector<vector<int> > generate(int numRows) {

vector<vector<int> > ret;

vector<int> one;

if(numRows<=0)

return ret;

one.push\_back(1);

ret.push\_back(one);

if(numRows==1)

return ret;

for(int i=2;i<=numRows;i++){

one.push\_back(1);

for(int j=i-2;j>0;j--){

one[j]=ret[i-2][j-1]+ret[i-2][j];

}

ret.push\_back(one);

}

return ret;

}

};

# Pascal's Triangle II

## 题目

. Given an index k, return the kth row of the Pascal's triangle.

For example, given k = 3,

Return [1,3,3,1].

Note:

Could you optimize your algorithm to use only O(k) extra space?

## 解法

### 分析

 .开始用数学方法计算组合数，发现较大的值无法计算（会超出long long的限制）。后改用滚动数组，pass

### 程序

. class Solution {

public:

vector<int> getRow(int rowIndex) {

vector<int> last;

vector<int> curr;

curr.push\_back(1);

if(rowIndex==0)

return curr;

int i=0;

while(1){

i++;

if(i%2){

last.assign(curr.begin(),curr.end());

last.push\_back(1);

for(int i=last.size()-2;i>0;i--){

last[i]=curr[i-1]+curr[i];

}

}

else{

curr.assign(last.begin(),last.end());

curr.push\_back(1);

for(int i=curr.size()-2;i>0;i--){

curr[i]=last[i-1]+last[i];

}

}

if(i==rowIndex)

break;

}

if(i%2)

return last;

else

return curr;

}

};

# Triangle

## 题目

. Given a triangle, find the minimum path sum from top to bottom. Each step you may move to adjacent numbers on the row below.

For example, given the following triangle

[

[2],

[3,4],

[6,5,7],

[4,1,8,3]

]

The minimum path sum from top to bottom is 11 (i.e., 2 + 3 + 5 + 1 = 11).

Note:

Bonus point if you are able to do this using only O(n) extra space, where n is the total number of rows in the triangle.

## 解法

### 分析

 .最明显的动态规划

### 程序

. class Solution {

public:

int minimumTotal(vector<vector<int> > &triangle) {

if(triangle.empty())

return 0;

vector<int> one;

for(int i=0;i<(int)triangle.size();i++){

one.push\_back(0);

}

one[0]=triangle[0][0];

vector<int> last;

for(int i=1;i<(int)triangle.size();i++){

last.assign(one.begin(),one.end());

for(int j=0;j<=i;j++){

if(j==0){

one[j]=last[j]+triangle[i][j];

}

else if(j==i){

one[j]=last[j-1]+triangle[i][j];

}

else{

int a1,a2;

a1 = last[j-1]+triangle[i][j];

a2 = last[j]+triangle[i][j];

if(a1<a2)

one[j]=a1;

else

one[j]=a2;

}

}

last.clear();

}

int min = one[0];

for(int i=1;i<(int)one.size();i++){

if(one[i]<min)

min = one[i];

}

return min;

}

};

# Best Time to Buy and Sell Stock

## 题目

. Say you have an array for which the ith element is the price of a given stock on day i.

If you were only permitted to complete at most one transaction (ie, buy one and sell one share of the stock), design an algorithm to find the maximum profit.

## 解法

### 分析

 .简单不分析

### 程序

. class Solution {

public:

int maxProfit(vector<int> &prices) {

if(prices.empty())

return 0;

int min = prices[0];

int profit = 0;

for(int i=0;i<(int)prices.size();i++){

if(prices[i]<min)

min=prices[i];

if(prices[i]-min>profit)

profit = prices[i]-min;

}

return profit;

}

};

# Best Time to Buy and Sell Stock II

## 题目

. Say you have an array for which the ith element is the price of a given stock on day i.

Design an algorithm to find the maximum profit. You may complete as many transactions as you like (ie, buy one and sell one share of the stock multiple times). However, you may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

## 解法

### 分析

 .很简单

### 程序

. class Solution {

public:

int maxProfit(vector<int> &prices) {

int buy;

int sell;

int profit = 0;

buy=sell=-1;

for(int i=0;i<(int)prices.size();i++){

if(buy==-1){

buy = i;

}

else if(buy!=-1&&sell==-1){

if(prices[i]>prices[buy])

sell = i;

else

buy = i;

}

else{

if(prices[i]>prices[sell])

sell = i;

else{

profit += prices[sell] - prices[buy];

buy = i;

sell = -1;

}

}

}

if(buy!=-1&&sell!=-1)

profit += prices[sell] - prices[buy];

return profit;

}

};

# Best Time to Buy and Sell Stock III

## 题目

. Say you have an array for which the ith element is the price of a given stock on day i.

Design an algorithm to find the maximum profit. You may complete at most two transactions.

Note:

You may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

## 解法

### 分析

 .三遍遍历，第一次，从头到尾，计算前n天只交易一次，能够获取的最大利润（和第一题相同），需要O(n)

第二遍，从尾到头，计算后n天之交易一次，能够获取的最大利润，和第一题类似。需要O(n)

第三遍，计算前n天和后l-n天之和（最多各交易一次）。

总的时间复杂度为O(n)。

### 程序

. class Solution {

public:

int maxProfit(vector<int> &prices) {

int l = prices.size();

if(l<=1)

return 0;

int maxPrevN[l];

int maxSuccN[l];

maxPrevN[0]=0;

maxPrevN[1]=0;

int min,max;

min=max=prices[0];

for(int i=2;i<=l;i++){

if(prices[i-1]-min>maxPrevN[i-1]){

maxPrevN[i]=prices[i-1]-min;

}

else{

maxPrevN[i] = maxPrevN[i-1];

}

if(prices[i-1]<min){

min = prices[i-1];

}

if(prices[i-1]>max){

max = prices[i-1];

}

}

maxSuccN[0]=0;

maxSuccN[1]=0;

min=max=prices[l-1];

for(int i=2;i<=l;i++){

if(max - prices[l-i]>maxSuccN[i-1]){

maxSuccN[i] = max - prices[l-i];

}

else{

maxSuccN[i] = maxSuccN[i-1];

}

if(prices[l-i]>max){

max = prices[l-i];

}

if(prices[l-i]<min){

min = prices[l-i];

}

}

max=0;

for(int i=0;i<=l;i++){

int tmp = maxPrevN[i]+maxSuccN[l-i];

if(tmp>max)

max = tmp;

}

return max;

}

};

# Binary Tree Maximum Path Sum

## 题目

. Given a binary tree, find the maximum path sum.

The path may start and end at any node in the tree.

For example:

Given the below binary tree,

1

/ \

2 3

Return 6.

## 解法

### 分析

 .利用递归，看下面的程序理解思路不难。

### 程序

. /\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

class Solution {

public:

int mMaxPathSum;

void maxPath(TreeNode \*root, int &mMaxPath){

if(root==NULL)

return;

if(root->left==NULL&&root->right==NULL){

mMaxPath = root->val;

if(mMaxPath>mMaxPathSum){

mMaxPathSum = mMaxPath;

}

return;

}

if(root->left==NULL){

int pr;

maxPath(root->right,pr);

if(pr>0)

mMaxPath = pr+root->val;

else

mMaxPath = root->val;

if(mMaxPath>mMaxPathSum){

mMaxPathSum = mMaxPath;

}

return;

}

if(root->right==NULL){

int pl;

maxPath(root->left,pl);

if(pl>0)

mMaxPath = pl+root->val;

else

mMaxPath = root->val;

if(mMaxPath>mMaxPathSum){

mMaxPathSum = mMaxPath;

}

return;

}

int pr,pl;

maxPath(root->left,pl);

maxPath(root->right,pr);

if(pl<0&&pr<0)

mMaxPath = root->val;

else if(pl<pr)

mMaxPath = root->val+pr;

else

mMaxPath = root->val+pl;

int tmp=root->val;

if(pl>0)

tmp+=pl;

if(pr>0)

tmp+=pr;

if(tmp>mMaxPathSum){

mMaxPathSum = tmp;

}

return;

}

int maxPathSum(TreeNode \*root) {

mMaxPathSum = -0x1FFFFFFF;

int i;

maxPath(root,i);

return mMaxPathSum;

}

};

# Valid Palindrome

## 题目

. Given a string, determine if it is a palindrome, considering only alphanumeric characters and ignoring cases.

For example,

"A man, a plan, a canal: Panama" is a palindrome.

"race a car" is not a palindrome.

Note:

Have you consider that the string might be empty? This is a good question to ask during an interview.

For the purpose of this problem, we define empty string as valid palindrome.

## 解法

### 分析

 .简单不分析

### 程序

. class Solution {

public:

bool isLetter(char c){

if(c>='0'&&c<='9')

return true;

if(c>='a'&&c<='z')

return true;

if(c>='A'&&c<='Z')

return true;

return false;

}

char getLetter(char c){

if(c>='A'&&c<='Z')

return c+'a'-'A';

return c;

}

bool isPalindrome(string s) {

if(s.empty())

return true;

int l=0;

int r=s.length()-1;

while(l<r){

if(!isLetter(s[l]))

l++;

else if(!isLetter(s[r]))

r--;

else{

char lCh;

char rCh;

lCh = getLetter(s[l]);

rCh = getLetter(s[r]);

if(lCh!=rCh)

return false;

l++;

r--;

}

}

return true;

}

};

# Word Ladder

## 题目

. Given two words (start and end), and a dictionary, find the length of shortest transformation sequence from start to end, such that:

Only one letter can be changed at a time

Each intermediate word must exist in the dictionary

For example,

Given:

start = "hit"

end = "cog"

dict = ["hot","dot","dog","lot","log"]

As one shortest transformation is "hit" -> "hot" -> "dot" -> "dog" -> "cog",

return its length 5.

## 解法

### 分析

 .广度优先

### 程序

. class Solution {

public:

struct QueueNode{

string s;

int distance;

QueueNode(string s1,int n):s(s1),distance(n){}

};

int ladderLength(string start, string end, unordered\_set<string> &dict) {

int result=0;

unordered\_set<string> checked;

queue<QueueNode> q;

q.push(QueueNode(start,1));

unordered\_set<string>::iterator iter1;

unordered\_set<string>::iterator iter2;

while(!q.empty()){

QueueNode node = q.front();

q.pop();

//create all neighbor of node.s

string s;

s.assign(node.s);

for(int i=0;i<(int)s.size();i++){

if(result!=0)

break;

for(int j=0;j<26;j++){

if(node.s.at(i)==(j+'a'))

continue;

s.at(i)=j+'a';

if(s==end){

result=node.distance+1;

break;

}

iter1 = dict.find(s);

iter2 = checked.find(s);

if(

(iter1!=dict.end())

&&(iter2==checked.end())

)

{

q.push(QueueNode(s,node.distance+1));

checked.insert(s);

}

}

s.at(i)=node.s.at(i);

}

}

return result;

}

};

# Word Ladder II

## 题目

. Given two words (start and end), and a dictionary, find all shortest transformation sequence(s) from start to end, such that:

Only one letter can be changed at a time

Each intermediate word must exist in the dictionary

For example,

Given:

start = "hit"

end = "cog"

dict = ["hot","dot","dog","lot","log"]

Return

[

["hit","hot","dot","dog","cog"],

["hit","hot","lot","log","cog"]

]

Note:

All words have the same length.

All words contain only lowercase alphabetic characters.

## 解法

### 分析

 .主要思想仍旧是广度优先。但是实现有很多细节需要注意。否则很容易导致重复运算。

1. 到达某些节点，可能有多条同样长度的路径

### 程序

. class Solution {

public:

void getPath(

string &end, string &start,

unordered\_map<string, unordered\_set<string>>&father,

vector<string> &path,

vector<vector<string> > &ret)

{

path.push\_back(end);

if(end==start)

ret.push\_back(vector<string>(path.rbegin(),path.rend()));

else{

for(auto e:father[end]){

getPath(e,start,father,path,ret);

}

}

path.pop\_back();

}

void BFS(string &start, string &end, unordered\_set<string> &dict, vector<vector<string> > &ret){

if(start.size()!=end.size()||start.size()==0||start==end) return;

queue<string> q;

unordered\_map<string, unordered\_set<string>> father;

unordered\_set<string> used;

unordered\_set<string> levelUsed;

q.push(start);

int levelCounter=1;

bool found=false;

while(!q.empty()){

string s=q.front();

string curr(s);

q.pop();

levelCounter--;

for(size\_t i=0;i<s.size();i++){

for(char a='a';a<='z';a++){

if(a==curr[i])continue;

s[i]=a;

if(s==end){

found = true;

father[s].insert(curr);

break;

}

if(used.count(s)==0&&dict.count(s)!=0){

levelUsed.insert(s);

father[s].insert(curr);

}

}

//if(found)break;

s[i]=curr[i];

}

if(levelCounter==0){

if(found)break;

levelCounter = levelUsed.size();

for(auto e : levelUsed){

used.insert(e);

q.push(e);

}

levelUsed.clear();

}

}

if(found){

vector<string> path;

getPath(end, start, father, path, ret);

}

}

vector<vector<string> > findLadders(string start, string end, unordered\_set<string> &dict) {

vector<vector<string> > ret;

BFS(start, end, dict, ret);

return ret;

}

};

# Longest Consecutive Sequence

## 题目

. Given an unsorted array of integers, find the length of the longest consecutive elements sequence.

For example,

Given [100, 4, 200, 1, 3, 2],

The longest consecutive elements sequence is [1, 2, 3, 4]. Return its length: 4.

Your algorithm should run in O(n) complexity.

## 解法

### 分析

 . 这题有点意思,我从网上搜到下面这个方法.

我们在判断某个数的连续序列时，会分别往减小和增大的方向找下一个连续数在不在数组中。然后把两个方向的长度加起来即为包含该数的一个连续序列。需要注意的是，当前数的长度计数只需要出现在一个方向的查找中计算即可，否则就重复了。要找一个数是不是在数组中，不可能用遍历的方法实现，这样时间复杂度就超过O(n)了。而要降低时间复杂度，一个经典的方案就是空间换时间。用增加空间复杂度的方法来换取时间复杂度的降低。所以我们可以先对数组进行一次预处理，生成一份包含数组元素的哈希表。这样在求解某个数字在不在数组时就可以得到O(1)的时间复杂度。

### 程序

. class Solution {

public:

struct HashNode{

int val;

HashNode\* next;

HashNode(int x) : val(x), next(NULL) {}

};

const static int HASHTABLESIZE = 30000;

HashNode\* hashTable[HASHTABLESIZE];

int getPosition(int val){

int low,high;

low = val & 0xFFFF;

high = val >> 16;

return (low+high)%HASHTABLESIZE;

}

void insertHash(int val){

int pos = getPosition(val);

HashNode\* head = hashTable[pos];

HashNode\* node = new HashNode(val);

node->next = head;

hashTable[pos] = node;

}

void initHashTable(){

for(int i=0;i<HASHTABLESIZE;i++){

hashTable[i]=NULL;

}

}

void clearHashTable(){

HashNode\* node;

for(int i=0;i<HASHTABLESIZE;i++){

if(hashTable[i]!=NULL){

node = hashTable[i];

while(node!=NULL){

HashNode\* tmp = node->next;

delete(node);

node = tmp;

}

hashTable[i]=NULL;

}

}

}

bool CheckHashAndDelete(int val){

int pos = getPosition(val);

HashNode\* node = hashTable[pos];

if(node==NULL){

return false;

}

while(node->next!=NULL){

if(node->next->val==val){

HashNode\* tmp = node->next;

node->next = tmp->next;

free(tmp);

return true;

}

}

if(node->val==val){

hashTable[pos]=node->next;

free(node);

return true;

}

return false;

}

int longestConsecutive(vector<int> &num) {

int result=0;

initHashTable();

vector<int>::iterator iter;

int min,max;

min=max=num[0];

for(iter=num.begin();iter!=num.end();iter++){

insertHash(\*iter);

if(\*iter>max)max=\*iter;

if(\*iter<min)min=\*iter;

}

for(iter=num.begin();iter!=num.end();iter++){

int base = \*iter;

if(!CheckHashAndDelete(base)){

continue;

}

int current = 1;

for(int i=base+1;i<=max;i++){

if(CheckHashAndDelete(i)){

current++;

}

else{

break;

}

}

for(int i=base-1;i>=min;i--){

if(CheckHashAndDelete(i)){

current++;

}

else{

break;

}

}

if(current>result)result=current;

}

clearHashTable();

return result;

}

};

# Sum Root to Leaf Numbers

## 题目

. Given a binary tree containing digits from 0-9 only, each root-to-leaf path could represent a number.

An example is the root-to-leaf path 1->2->3 which represents the number 123.

Find the total sum of all root-to-leaf numbers.

For example,

1

/ \

2 3

The root-to-leaf path 1->2 represents the number 12.

The root-to-leaf path 1->3 represents the number 13.

Return the sum = 12 + 13 = 25.

## 解法

### 分析

 .深度优先搜索,算法无难度

### 程序

. /\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

class Solution {

public:

int createNumber(vector<int> &list){

int i=0;

vector<int>::iterator iter;

for(iter=list.begin();iter!=list.end();iter++){

i = i\*10+(\*iter);

}

return i;

}

void DFS(TreeNode \*root, vector<int> &stack, vector<int> &set){

if(root==NULL)return;

if((root->left==NULL)&&(root->right==NULL)){

stack.push\_back(root->val);

set.push\_back(createNumber(stack));

stack.pop\_back();

return;

}

stack.push\_back(root->val);

if(root->left!=NULL){

DFS(root->left, stack, set);

}

if(root->right!=NULL){

DFS(root->right, stack, set);

}

stack.pop\_back();

return;

}

int calSum(vector<int> &list){

int i=0;

vector<int>::iterator iter;

for(iter=list.begin();iter!=list.end();iter++){

i += (\*iter);

}

return i;

}

int sumNumbers(TreeNode \*root) {

if(root==NULL)return 0;

vector<int> mStack;

vector<int> numSet;

DFS(root,mStack,numSet);

int result = calSum(numSet);

numSet.clear();

return result;

}

};

# Surrounded Regions

## 题目

. Given a 2D board containing 'X' and 'O', capture all regions surrounded by 'X'.

A region is captured by flipping all 'O's into 'X's in that surrounded region.

For example,

X X X X

X O O X

X X O X

X O X X

After running your function, the board should be:

X X X X

X X X X

X X X X

X O X X

## 解法

### 分析

 .广度优先搜索,把在边界上有缺口的空白区域用特殊字符填满。剩余空白是被包围的部分，用X填充。特殊字符换回空白。

### 程序

. class Solution {

public:

struct point{

int x;

int y;

point(int in\_x, int in\_y):x(in\_x),y(in\_y){}

};

void printBoard(vector<vector<char> > &board){

return;

for(int i=0;i<(int)board.size();i++){

for(int j=0;j<(int)board[0].size();j++){

printf("%c",board[i][j]);

}

printf("\n");

}

printf("\nFinished\n");

}

void initSeed(queue<struct point> &q, vector<vector<char> > &board){

int x,y;

y=board.size();

if(y==0) return;

x=board[0].size();

if(x==0) return;

for(int i=0;i<y;i++){

if(board[i][0]=='O'){

board[i][0]='S';

q.push(point(0,i));

}

if(board[i][x-1]=='O'){

board[i][x-1]='S';

q.push(point(x-1,i));

}

}

for(int i=0;i<x;i++){

if(board[0][i]=='O'){

board[0][i]='S';

q.push( point(i,0));

}

if(board[y-1][i]=='O'){

board[y-1][i]='S';

q.push( point(i,y-1));

}

}

printBoard(board);

}

void BFD(queue<struct point> &q, vector<vector<char> > &board){

int x,y;

y=board.size();

if(y==0) return;

x=board[0].size();

if(x==0) return;

while(!q.empty()){

struct point \*tmp=&(q.front());

//x-1,y

if((tmp->x>=1)&&(board[tmp->y][tmp->x-1]=='O')){

board[tmp->y][tmp->x-1]='S';

q.push( point(tmp->x-1, tmp->y ));

}

//x+1,y

if((tmp->x<(x-1))&&(board[tmp->y][tmp->x+1]=='O')){

board[tmp->y][tmp->x+1]='S';

q.push( point(tmp->x+1, tmp->y ));

}

//x,y-1

if((tmp->y>=1)&&(board[tmp->y-1][tmp->x]=='O')){

board[tmp->y-1][tmp->x]='S';

q.push( point(tmp->x, tmp->y-1));

}

//x,y+1

if((tmp->y<(y-1))&&(board[tmp->y+1][tmp->x]=='O')){

board[tmp->y+1][tmp->x]='S';

q.push( point(tmp->x, tmp->y+1));

}

board[tmp->y][tmp->x]='E';

q.pop();

printBoard(board);

}

}

void updateValue(vector<vector<char> > &board){

int x,y;

y=board.size();

if(y==0) return;

x=board[0].size();

if(x==0) return;

for(int i=0;i<y;i++){

for(int j=0;j<x;j++){

if(board[i][j]=='O')

board[i][j]='X';

else if(board[i][j]!='X')

board[i][j]='O';

}

}

}

void solve(vector<vector<char> > &board) {

int x,y;

y=board.size();

if(y==0) return;

x=board[0].size();

if(x==0) return;

queue<struct point> q;

initSeed(q, board);

BFD(q,board);

updateValue(board);

}

};

# Palindrome Partitioning

## 题目

. Given a string s, partition s such that every substring of the partition is a palindrome.

Return all possible palindrome partitioning of s.

For example, given s = "aab",

Return

[

["aa","b"],

["a","a","b"]

]

## 解法

### 分析

 .编程题，算法无难度。递归产生所有分割方式。

### 程序

. class Solution {

public:

bool isPalindrome(string s){

int i = s.length();

if(i==0)return false;

int l;

l = i;

i /= 2;

for(;i>0;i--){

if(s.at(i-1)!=s.at(l-i))

return false;

}

return true;

}

inline void printDebug(string s1, string s2){

return;

printf(":%s:,:%s:\n",s1.c\_str(),s2.c\_str());

}

void mPartition(

string s,

vector<vector<string> > &result,

vector<string> &oneResult)

{

int l = s.length();

for(int length=1;length<=l;length++){

printDebug(s.substr(0,length),s.substr(length,l));

if(isPalindrome(s.substr(0,length))){

if(length==(l)){

oneResult.push\_back(s.substr(0,length));

result.push\_back(oneResult);

oneResult.pop\_back();

}

else

{

oneResult.push\_back(s.substr(0,length));

mPartition(s.substr(length,l), result, oneResult);

oneResult.pop\_back();

}

}

}

return;

}

vector<vector<string> > partition(string s) {

vector<vector<string> > result;

vector<string> oneResult;

if(s.length()==0) return result;

mPartition(s, result, oneResult);

return result;

}

};

# Clone Graph

## 题目

. Clone an undirected graph. Each node in the graph contains a label and a list of its neighbors.

OJ's undirected graph serialization:

Nodes are labeled uniquely.

We use # as a separator for each node, and , as a separator for node label and each neighbor of the node.

As an example, consider the serialized graph {0,1,2#1,2#2,2}.

The graph has a total of three nodes, and therefore contains three parts as separated by #.

First node is labeled as 0. Connect node 0 to both nodes 1 and 2.

Second node is labeled as 1. Connect node 1 to node 2.

Third node is labeled as 2. Connect node 2 to node 2 (itself), thus forming a self-cycle.

Visually, the graph looks like the following:

1

/ \

/ \

0 --- 2

/ \

\\_/

## 解法

### 分析

 .广度优先，处理每个未处理的结点。

### 程序

. /\*\*

\* Definition for undirected graph.

\* struct UndirectedGraphNode {

\* int label;

\* vector<UndirectedGraphNode \*> neighbors;

\* UndirectedGraphNode(int x) : label(x) {};

\* };

\*/

class Solution {

public:

UndirectedGraphNode \*cloneGraph(UndirectedGraphNode \*node) {

if (node == NULL)

return NULL;

map<int, UndirectedGraphNode\*> mMap;

queue<UndirectedGraphNode\*> q;

UndirectedGraphNode \*head = new UndirectedGraphNode(node->label);

mMap.insert(pair<int, UndirectedGraphNode \*>(head->label, head));

q.push(node);

while (!q.empty()) {

UndirectedGraphNode \*origin\_from;

UndirectedGraphNode \*own\_from;

int label\_from = q.front()->label;

origin\_from = q.front();

q.pop();

map<int, UndirectedGraphNode \*>::iterator iter;

iter = mMap.find(label\_from);

if (iter == mMap.end()) {

return NULL;

}

own\_from = iter->second;

vector<UndirectedGraphNode \*>::iterator child;

for (child = origin\_from->neighbors.begin();

child != origin\_from->neighbors.end(); child++) {

UndirectedGraphNode \*to;

iter = mMap.find((\*child)->label);

if (iter == mMap.end()) {

to = new UndirectedGraphNode((\*child)->label);

mMap.insert(

pair<int, UndirectedGraphNode \*>(to->label, to));

q.push((\*child));

} else {

to = iter->second;

}

own\_from->neighbors.push\_back(to);

}

}

mMap.clear();

return head;

}

};

# Gas Station

## 题目

. There are N gas stations along a circular route, where the amount of gas at station i is gas[i].

You have a car with an unlimited gas tank and it costs cost[i] of gas to travel from station i to its next station (i+1). You begin the journey with an empty tank at one of the gas stations.

Return the starting gas station's index if you can travel around the circuit once, otherwise return -1.

Note:

The solution is guaranteed to be unique.

## 解法

### 分析

 .程序无难度

### 程序

. class Solution {

public:

void initAcc(

vector<int> &gas,

vector<int> &cost,

vector<int> &acc\_gas,

vector<int> &acc\_cost)

{

acc\_gas.push\_back(0);

acc\_cost.push\_back(0);

for(int i=0; i < (int)gas.size(); i++){

acc\_gas.push\_back(acc\_gas[i]+gas[i]);

acc\_cost.push\_back(acc\_cost[i]+cost[i]);

}

}

bool checkStartPoint(int start,vector<int> &acc\_gas, vector<int> &acc\_cost){

int size = acc\_gas.size()-1;

for(int i=0;i<size;i++){

if(i<start){

if(

(acc\_gas[size]-acc\_gas[start]+acc\_gas[i+1]-acc\_gas[0])

<

(acc\_cost[size]-acc\_cost[start]+acc\_cost[i+1]-acc\_cost[0])

)

{

return false;

}

}

else{

if((acc\_gas[i+1]-acc\_gas[start])<(acc\_cost[i+1]-acc\_cost[start]))

return false;

}

}

return true;

}

int canCompleteCircuit(vector<int> &gas, vector<int> &cost) {

vector<int> acc\_gas;

vector<int> acc\_cost;

initAcc(gas, cost, acc\_gas, acc\_cost);

for(int i=0;i<(int)gas.size();i++){

if(checkStartPoint(i,acc\_gas, acc\_cost))

return i;

}

return -1;

}

};

# Candy

## 题目

. There are N children standing in a line. Each child is assigned a rating value.

You are giving candies to these children subjected to the following requirements:

Each child must have at least one candy.

Children with a higher rating get more candies than their neighbors.

What is the minimum candies you must give?

## 解法

### 分析

 .比两侧的孩子rate都低的（谷底的），分配一根

从谷底向两侧扩张

### 程序

. class Solution {

public:

int candy(vector<int> &ratings) {

vector<int> result;

for(int i=0;i<(int)ratings.size();i++){

result.push\_back(-1);

}

queue<int> q;

//search for seeds

for(int i=0;i<(int)ratings.size();i++){

if(i==0){

if(ratings[i]<=ratings[i+1]){

q.push(i);

result[i]=1;

}

}

else if(i==(ratings.size()-1)){

if(ratings[i]<=ratings[i-1]){

q.push(i);

result[i]=1;

}

}

else{

if(

(ratings[i]<=ratings[i-1])

&&(ratings[i]<=ratings[i+1])

){

q.push(i);

result[i]=1;

}

}

}

//BFD

while(!q.empty()){

int i=q.front();

q.pop();

int j=i-1;

if(j>=0){

if(

(result[j]==-1)

&&(ratings[j]>=ratings[i])

)

{

if(ratings[j]>ratings[i])

result[j]=result[i]+1;

else

result[j]=1;

q.push(j);

}

else if(

(result[j]!=-1)

&&(ratings[j]>=ratings[i])

)

{

if(ratings[j]>ratings[i])

result[j]=result[j]>(result[i]+1)?result[j]:(result[i]+1);

else{

}

}

}

j=i+1;

if(j<=(ratings.size()-1)){

if(

(result[j]==-1)

&&(ratings[j]>=ratings[i])

)

{

if(ratings[j]>ratings[i])

result[j]=result[i]+1;

else

result[j]=1;

q.push(j);

}

else if(

(result[j]!=-1)

&&(ratings[j]>=ratings[i])

)

{

if(ratings[j]>ratings[i])

result[j]=result[j]>(result[i]+1)?result[j]:(result[i]+1);

else{

}

}

}

}

int res=0;

for(int i=0;i<(int)result.size();i++){

res += result[i];

}

return res;

}

};

# Single Number

## 题目

. Given an array of integers, every element appears twice except for one. Find that single one.

Note:

Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory?

## 解法

### 分析

 .编程之美原题，利用异或运算。

### 程序

. class Solution {

public:

int singleNumber(int A[], int n) {

int i=0;

for(int j=0;j<n;j++){

i ^= A[j];

}

return i;

}

};

# Single Number II

## 题目

. Given an array of integers, every element appears three times except for one. Find that single one.

Note:

Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory?

## 解法

### 分析

 .思路很巧。

针对每个bit考虑。由于其他元素都是三个三个出现。如果全集针对一个bit求和，用3去除的余数，就是目标数字的值。

### 程序

. class Solution {

public:

int singleNumber(int A[],int l){

int n=sizeof(int)\*8;

int bit[n];

for(int i=0;i<n;i++){

bit[i]=0;

}

for(int j=0;j<l;j++){

for(int i=0;i<n;i++){

bit[i]+=(A[j]>>i)&1;

}

}

int result=0;

for(int i=0;i<n;i++){

result += (bit[i]%3)<<i;

}

return result;

}

};

# Copy List with Random Pointer

## 题目

. A linked list is given such that each node contains an additional random pointer which could point to any node in the list or null.

Return a deep copy of the list.

## 解法

### 分析

 .原来的链表中，每个元素后面插入一个新元素，这个新元素是新链表的节点。则任意指针的赋值就很简单了。

### 程序

. /\*\*

\* Definition for singly-linked list with a random pointer.

\* struct RandomListNode {

\* int label;

\* RandomListNode \*next, \*random;

\* RandomListNode(int x) : label(x), next(NULL), random(NULL) {}

\* };

\*/

class Solution {

public:

RandomListNode \*copyRandomList(RandomListNode \*head) {

if(head==NULL)return NULL;

RandomListNode\* a;

RandomListNode\* b;

RandomListNode\* result;

a=head;

while(a!=NULL)

{

RandomListNode\* tmp = (RandomListNode\*)malloc(sizeof(RandomListNode));

tmp->label = a->label;

tmp->next = a->next;

a->next = tmp;

a = tmp->next;

}

a=head;

while(a!=NULL)

{

b = a->next;

if(a->random!=NULL)

b->random = a->random->next;

else

b->random=NULL;

a=b->next;

}

//split two list

a=head;

b=a->next;

result = b;

while(1)

{

if(b->next!=NULL)

{

a->next = b->next;

a = b->next;

b->next = a->next;

b = a->next;

}

else

{

a->next = NULL;

break;

}

}

return result;

}

};

# Word Break

## 题目

. Given a string s and a dictionary of words dict, determine if s can be segmented into a space-separated sequence of one or more dictionary words.

For example, given

s = "leetcode",

dict = ["leet", "code"].

Return true because "leetcode" can be segmented as "leet code".

## 解法

### 分析

 .递归法程序很简单，但是时间复杂度很高。

利用动态规划可解。

d[i][j]表示从i到j-1的串可用字典分割

### 程序

. class Solution {

public:

bool result[1000][1000];

bool wordBreak(string s, unordered\_set<string> &dict) {

int len = s.length();

for(int i=0;i<(len+1);i++){

for(int j=0;j<(len+1);j++){

result[i][j]=false;

}

}

for(int leng=1;leng<len+1;leng++){

for(int start=0;start<(len-leng+1);start++)

{

unordered\_set<string>::iterator iter;

iter = dict.find(s.substr(start,leng));

// printf("%s\n",s.substr(start,leng).c\_str());

if(iter!=dict.end())

{

result[start][leng]=true;

continue;

}

else{

for(int j=1;j<leng;j++){

// printf("%s\n",s.substr(start,j).c\_str());

// printf("%s\n",s.substr(start+j,leng-j).c\_str());

if(result[start][j]){

if(result[start+j][leng-j]){

result[start][leng]=true;

continue;

}

}

}

}

}

}

return result[0][len];

}

};

# Word Break II

## 题目

. Given a string s and a dictionary of words dict, add spaces in s to construct a sentence where each word is a valid dictionary word.

Return all such possible sentences.

For example, given

s = "catsanddog",

dict = ["cat", "cats", "and", "sand", "dog"].

A solution is ["cats and dog", "cat sand dog"].

## 解法

### 分析

 .

### 程序

. class Solution {

public:

static const int matrix\_size=200;

bool result\_exist[matrix\_size][matrix\_size];

set<string> result\_set[matrix\_size][matrix\_size];

bool result[matrix\_size][matrix\_size];

bool wordBreakEx(string s, unordered\_set<string> &dict) {

int len = s.length();

for(int i=0;i<(len+1);i++){

for(int j=0;j<(len+1);j++){

result[i][j]=false;

}

}

for(int leng=1;leng<len+1;leng++){

for(int start=0;start<(len-leng+1);start++)

{

unordered\_set<string>::iterator iter;

iter = dict.find(s.substr(start,leng));

// printf("%s\n",s.substr(start,leng).c\_str());

if(iter!=dict.end())

{

result[start][leng]=true;

continue;

}

else{

for(int j=1;j<leng;j++){

// printf("%s\n",s.substr(start,j).c\_str());

// printf("%s\n",s.substr(start+j,leng-j).c\_str());

if(result[start][j]){

if(result[start+j][leng-j]){

result[start][leng]=true;

continue;

}

}

}

}

}

}

return result[0][len];

}

vector<string> wordBreak(string s, unordered\_set<string> &dict) {

vector<string> result;

if(!wordBreakEx(s,dict))return result;

int len = s.length();

for(int i=0;i<(len+1);i++){

for(int j=0;j<(len+1);j++){

result\_exist[i][j]=false;

result\_set[i][j].clear();

}

}

for(int leng=1;leng<len+1;leng++){

for(int start=0;start<(len-leng+1);start++)

{

unordered\_set<string>::iterator iter;

iter = dict.find(s.substr(start,leng));

//printf("%s\n",s.substr(start,leng).c\_str());

if(iter!=dict.end())

{

result\_exist[start][leng]=true;

result\_set[start][leng].insert(s.substr(start,leng));

//continue;

}

{

for(int j=1;j<leng;j++){

//printf("%s\n",s.substr(start,j).c\_str());

//printf("%s\n",s.substr(start+j,leng-j).c\_str());

if(result\_exist[start][j]){

if(result\_exist[start+j][leng-j]){

result\_exist[start][leng]=true;

set<string>::iterator iter\_a;

set<string>::iterator iter\_b;

for(iter\_a = result\_set[start][j].begin();

iter\_a != result\_set[start][j].end();iter\_a++){

for(iter\_b = result\_set[start+j][leng-j].begin();

iter\_b != result\_set[start+j][leng-j].end();iter\_b++){

string c1;

c1 = (\*iter\_a) + " " + (\*iter\_b);

//printf("%s\n",c1.c\_str());

set<string>::iterator iter\_c;

iter\_c = result\_set[start][leng].find(c1);

if(iter\_c!=result\_set[start][leng].end()){

}

else{

result\_set[start][leng].insert(c1);

}

}

}

//continue;

}

}

}

}

}

}

//construct final result

set<string>::iterator it;

//vector<string> result;

for(it = result\_set[0][len].begin();it!=result\_set[0][len].end();it++){

string str(\*it);

result.push\_back(str);

}

return result;

}

};

# Linked List Cycle

## 题目

. Given a linked list, determine if it has a cycle in it.

Follow up:

Can you solve it without using extra space?

## 解法

### 分析

 .很老的面试题

### 程序

. /\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

class Solution {

public:

bool hasCycle(ListNode \*head) {

bool result=false;

ListNode \*slow,\*fast;

slow=fast=head;

if(slow==NULL)return false;

for(;;){

fast = fast->next; if(fast==NULL)return false;

fast = fast->next; if(fast==NULL)return false;

slow = slow->next;

if(slow==fast) return true;

}

return result;

}

};

# Linked List Cycle II

## 题目

. Given a linked list, return the node where the cycle begins. If there is no cycle, return null.

Follow up:

Can you solve it without using extra space?

## 解法

### 分析

 .用hash表查看指针是否出现过。但是需要额外空间。

### 程序

. /\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

class Solution {

public:

ListNode \*detectCycle(ListNode \*head) {

if(head==NULL)return NULL;

ListNode \*curr;

map<ListNode\*,int> mMap;

curr=head;

while(curr!=NULL){

mMap.insert(pair<ListNode\*,int>(curr,0));

curr=curr->next;

if(curr==NULL){

mMap.clear();

return NULL;

}

map<ListNode\*,int>::iterator iter;

iter = mMap.find(curr);

if(iter!=mMap.end())

{

mMap.clear();

return curr;

}

}

return NULL;

}

};

# Reorder List

## 题目

. Given a singly linked list L: L0→L1→…→Ln-1→Ln,

reorder it to: L0→Ln→L1→Ln-1→L2→Ln-2→…

You must do this in-place without altering the nodes' values.

For example,

Given {1,2,3,4}, reorder it to {1,4,2,3}.

## 解法

### 分析

 .使用DeQ这个问题好像很简单.

### 程序

. /\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

class Solution {

public:

struct DeQueueNode{

ListNode\* val;

DeQueueNode\* prev;

DeQueueNode\* next;

};

DeQueueNode\* mHead;

DeQueueNode\* mTail;

Solution():mHead(NULL), mTail(NULL){

}

~Solution(){

ClearInstance();

}

void ClearInstance(){

DeQueueNode\* tmp=mHead;

while(tmp!=NULL){

mHead=mHead->next;

free(tmp);

tmp=mHead;

}

}

void ConstructDeQueue(ListNode \*head){

if(head==NULL){

mHead=mTail=NULL;

return;

}

mHead = (DeQueueNode\*)malloc(sizeof(DeQueueNode));

mHead->val=head;

mHead->prev=NULL;

mHead->next=NULL;

mTail=mHead;

head=head->next;

DeQueueNode\* tmp;

while(head!=NULL){

tmp = (DeQueueNode\*)malloc(sizeof(DeQueueNode));

tmp->val = head;

tmp->next = NULL;

tmp->prev = mTail;

mTail->next = tmp;

mTail = tmp;

head=head->next;

}

return;

}

void reorderList(ListNode \*head) {

if(head==NULL)return;

ConstructDeQueue(head);

DeQueueNode\* lHead=mHead;

DeQueueNode\* lTail=mTail;

while(lHead!=lTail){

lHead->val->next=lTail->val;

lHead = lHead->next;

if(lHead==lTail){

lTail->val->next=NULL;

break;

}

lTail->val->next = lHead->val;

lTail = lTail->prev;

}

lTail->val->next=NULL;

ClearInstance();

}

};

# Binary Tree Preorder Traversal

## 题目

. Given a binary tree, return the preorder traversal of its nodes' values.

For example:

Given binary tree {1,#,2,3},

1

\

2

/

3

return [1,2,3].

Note: Recursive solution is trivial, could you do it iteratively?

## 解法

### 分析

 .不准用递归。不过，用堆栈模拟即可。

### 程序

. /\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

class Solution {

public:

struct StackNode{

TreeNode \* treeNode;

int state;//0, none; 1, left accessed; 2 both accessed;

};

vector<int> preorderTraversal(TreeNode \*root) {

vector<int> nodeList;

stack<StackNode\*> mStack;

if(root==NULL)return nodeList;

StackNode \*stackNode=(StackNode\*)malloc(sizeof(StackNode));

StackNode \*tmp;

stackNode->treeNode = root;

stackNode->state = 0;

mStack.push(stackNode);

while(!mStack.empty()){

tmp = mStack.top();

if(tmp->state==0)

{

nodeList.push\_back(tmp->treeNode->val);

if(tmp->treeNode->left!=NULL){

tmp->state=1;

stackNode = (StackNode\*)malloc(sizeof(StackNode));

stackNode->treeNode = tmp->treeNode->left;

stackNode->state = 0;

mStack.push(stackNode);

continue;

}

else if(tmp->treeNode->right!=NULL){

tmp->state=2;

stackNode = (StackNode\*)malloc(sizeof(StackNode));

stackNode->treeNode = tmp->treeNode->right;

stackNode->state = 0;

mStack.push(stackNode);

continue;

}

else{

mStack.pop();

free(tmp);

}

continue;

}

else if(tmp->state==1)

{

if(tmp->treeNode->right!=NULL){

tmp->state=2;

stackNode = (StackNode\*)malloc(sizeof(StackNode));

stackNode->treeNode = tmp->treeNode->right;

stackNode->state = 0;

mStack.push(stackNode);

continue;

}

else

{

mStack.pop();

free(tmp);

continue;

}

}

else if(tmp->state==2)

{

mStack.pop();

free(tmp);

}

}

return nodeList;

}

/\*

vector<int> preorderTraversal(TreeNode \*root) {

}\*/

};

# Binary Tree Postorder Traversal

## 题目

. Given a binary tree, return the postorder traversal of its nodes' values.

For example:

Given binary tree {1,#,2,3},

1

\

2

/

3

return [3,2,1].

Note: Recursive solution is trivial, could you do it iteratively?

## 解法

### 分析

 .同上一题

### 程序

. /\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

class Solution {

public:

struct StackNode{

TreeNode \* treeNode;

int state;//0, none; 1, left accessed; 2 both accessed;

};

vector<int> postorderTraversal(TreeNode \*root) {

vector<int> nodeList;

stack<StackNode\*> mStack;

if(root==NULL)return nodeList;

StackNode \*stackNode=(StackNode\*)malloc(sizeof(StackNode));

StackNode \*tmp;

stackNode->treeNode = root;

stackNode->state = 0;

mStack.push(stackNode);

while(!mStack.empty()){

tmp = mStack.top();

if(tmp->state==2)

{

nodeList.push\_back(tmp->treeNode->val);

mStack.pop();

free(tmp);

continue;

}

else if(tmp->state==1)

{

if(tmp->treeNode->right!=NULL){

tmp->state=2;

stackNode = (StackNode\*)malloc(sizeof(StackNode));

stackNode->treeNode = tmp->treeNode->right;

stackNode->state = 0;

mStack.push(stackNode);

continue;

}

else

{

nodeList.push\_back(tmp->treeNode->val);

mStack.pop();

free(tmp);

continue;

}

}

else if(tmp->state==0)

{

if(tmp->treeNode->left!=NULL){

tmp->state=1;

stackNode = (StackNode\*)malloc(sizeof(StackNode));

stackNode->treeNode = tmp->treeNode->left;

stackNode->state = 0;

mStack.push(stackNode);

continue;

}

else if(tmp->treeNode->right!=NULL){

tmp->state=2;

stackNode = (StackNode\*)malloc(sizeof(StackNode));

stackNode->treeNode = tmp->treeNode->right;

stackNode->state = 0;

mStack.push(stackNode);

continue;

}

else

{

nodeList.push\_back(tmp->treeNode->val);

mStack.pop();

free(tmp);

continue;

}

}

}

return nodeList;

}

};

# LRU Cache

## 题目

. Design and implement a data structure for Least Recently Used (LRU) cache. It should support the following operations: get and set.

get(key) - Get the value (will always be positive) of the key if the key exists in the cache, otherwise return -1.

set(key, value) - Set or insert the value if the key is not already present. When the cache reached its capacity, it should invalidate the least recently used item before inserting a new item.

## 解法

### 分析

 .算法上没有大的问题，主要就是结合堆和hash两种数据结构（这里的程序我用的是map，其实是红黑树，如果用hash应该更好）

### 程序

. class LRUCache{

public:

struct HeapNode {

int key;

int lastAccess;

};

struct MapBody {

int value;

int HeapIndex;

MapBody(int v, int i):value(v),HeapIndex(i){}

};

HeapNode\* mHeap;

int mHeapSize;

int mHeapCapacity;

map<int,struct MapBody> mMap;

int mCurrTime;

int exchangeHeapNode(int parent, int n\_id){

int tmp;

tmp = mHeap[parent].lastAccess;

mHeap[parent].lastAccess = mHeap[n\_id].lastAccess;

mHeap[n\_id].lastAccess = tmp;

tmp = mHeap[parent].key;

mHeap[parent].key = mHeap[n\_id].key;

mHeap[n\_id].key = tmp;

map<int ,struct MapBody >::iterator l\_it;

l\_it=mMap.find(mHeap[parent].key);

if(l\_it!=mMap.end())

{

l\_it->second.HeapIndex = parent;

}

l\_it=mMap.find(mHeap[n\_id].key);

if(l\_it!=mMap.end())

{

l\_it->second.HeapIndex = n\_id;

}

return 0;

}

int adjustHeapNodeUp(int n\_id)

{

if(n\_id==0)return 0;

int parent =(int)((n\_id-1)/2);

int child = n\_id;

if(mHeap[n\_id].lastAccess < mHeap[parent].lastAccess)

{

exchangeHeapNode(parent, child);

adjustHeapNodeUp(parent);

return 0;

}

return 0;

}

int adjustHeapNodeDown(int n\_id)

{

int l\_child, r\_child;

l\_child = n\_id\*2 + 1;

r\_child = n\_id\*2 + 2;

if(l\_child >= mHeapSize) return 0;

int parent, child;

parent = child = -1;

if(r\_child >= mHeapSize)

{

if(mHeap[n\_id].lastAccess>mHeap[l\_child].lastAccess)

{

parent = n\_id;

child = l\_child;

}

}

else

{

if(

(mHeap[n\_id].lastAccess<mHeap[l\_child].lastAccess)

&&

(mHeap[n\_id].lastAccess<mHeap[r\_child].lastAccess)

)

{

}

else if(

(mHeap[n\_id].lastAccess<mHeap[l\_child].lastAccess)

&&

(mHeap[n\_id].lastAccess>mHeap[r\_child].lastAccess)

)

{

parent = n\_id;

child = r\_child;

}

else if(

(mHeap[n\_id].lastAccess>mHeap[l\_child].lastAccess)

&&

(mHeap[n\_id].lastAccess<mHeap[r\_child].lastAccess)

)

{

parent = n\_id;

child = l\_child;

}

else

{

if(mHeap[l\_child].lastAccess<mHeap[r\_child].lastAccess)

{

parent = n\_id;

child = l\_child;

}

else

{

parent = n\_id;

child = r\_child;

}

}

}

if(child>=0){

exchangeHeapNode(parent, child);

adjustHeapNodeDown(child);

}

return 0;

}

int insertHeapNode(int key, int lastAccess, int value) //

{

if( mHeapSize < mHeapCapacity ){

mHeap[mHeapSize].key = key;

mHeap[mHeapSize].lastAccess = lastAccess;

struct MapBody mapBody(value,mHeapSize);

mapBody.HeapIndex = mHeapSize;

mapBody.value = value;

mMap.insert(pair<int,struct MapBody>(key,mapBody));

mHeapSize++;

adjustHeapNodeUp(mHeapSize-1);

}

else

{

//Heap is FULL, the latest key must be cleared

// map<int ,struct MapBody >::iterator l\_it;

// l\_it=mMap.find(mHeap[0].key);

// if(l\_it!=mMap.end())

// {

// mMap.erase(l\_it);

// }

mMap.erase(mHeap[0].key);

mHeap[0].key = key;

mHeap[0].lastAccess = lastAccess;

struct MapBody mapBody(value,0);

mMap.insert(pair<int,struct MapBody>(key,mapBody));

adjustHeapNodeDown(0);

}

return 0;

}

int updateHeapNode(int n\_id, int key, int lastAccess) //

{

if(mHeap[n\_id].key != key) return 0;

mHeap[n\_id].lastAccess = lastAccess;

adjustHeapNodeDown(n\_id);

return 0;

}

LRUCache(int capacity):

mHeapSize(0),

mHeapCapacity(capacity),

mCurrTime(1)

{

mHeap = (HeapNode\*)malloc(sizeof(HeapNode)\*capacity);

mMap.clear();

}

~LRUCache(){

mMap.clear();

free(mHeap);

}

int get(int key) {

mCurrTime++;

map<int ,struct MapBody >::iterator l\_it;

l\_it=mMap.find(key);

if(l\_it!=mMap.end())

{

updateHeapNode(l\_it->second.HeapIndex, key, mCurrTime);

return l\_it->second.value;

}

return -1;

}

void set(int key, int value) {

mCurrTime++;

map<int ,struct MapBody >::iterator l\_it;

l\_it=mMap.find(key);

if(l\_it!=mMap.end())

{

l\_it->second.value = value;

updateHeapNode(l\_it->second.HeapIndex, key, mCurrTime);

return;

}

insertHeapNode(key, mCurrTime, value);

return;

}

};

# Insertion Sort List

## 题目

. Sort a linked list using insertion sort.

## 解法

### 分析

 .插入排序排序一个链表。算法无难度

### 程序

. /\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

class Solution {

public:

struct ListNodeEx {

ListNode\* node;

ListNodeEx \*prev;

ListNodeEx \*next;

};

ListNode \*insertionSortList(ListNode \*head) {

if(head==NULL) return head;

ListNode \*origin = head;

ListNodeEx \*last, \*curr;

last=(ListNodeEx \*)malloc(sizeof(ListNodeEx \*));

last->prev = NULL;

last->next = NULL;

last->node = head;

head = head->next;

while(head!=NULL){

curr=(ListNodeEx \*)malloc(sizeof(ListNodeEx \*));

curr->prev = last;

curr->next = NULL;

curr->node = head;

ListNodeEx \*search=curr;

while(search->prev!=NULL&&(search->prev->node->val>search->node->val)){

int tmp = search->node->val;

search->node->val = search->prev->node->val;

search->prev->node->val = tmp;

search = search->prev;

}

last = curr;

head = head->next;

}

curr = last;

while(curr!=NULL){

last = curr->prev;

free((void\*)curr);

curr=last;

}

return origin;

}

};

# Sort List

## 题目

. Sort a linked list in O(n log n) time using constant space complexity.

## 解法

### 分析

 .快排序

### 程序

. /\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

class Solution {

public:

ListNode \*partition(ListNode \*head,ListNode \*tail, int mid){

if(head==NULL) return NULL;

if(head->next==NULL) return head;

ListNode \*begin,\*end,\*previous;

int tmp;

begin=end=head;

previous=NULL;

while(end!=tail){

if(end->val<mid){

if(begin!=end){

tmp=end->val;

end->val=begin->val;

begin->val=tmp;

}

previous=begin;

begin=begin->next;

}

end=end->next;

}

return previous;

}

void part\_sort(ListNode \*head,ListNode \*tail){

if(head==NULL) return;

if(head==tail) return;

if(head->next==tail) return;

int mid = head->val;

ListNode\* mid\_node;

mid\_node = partition(head->next, tail, mid);

if(mid\_node==NULL)

{

part\_sort(head->next, tail);

}

else

{

head->val = mid\_node->val;

mid\_node->val = mid;

part\_sort(head, mid\_node);

part\_sort(mid\_node->next,tail);

}

}

ListNode\* countSort(ListNode\* head)

{

int min,max,gap;

min=max=head->val;

ListNode\* tmp=head;

ListNode\* one\_min;

while(tmp!=NULL)

{

if(tmp->val>max)

max=tmp->val;

else if(tmp->val<min)

{

one\_min = tmp;

min=tmp->val;

}

tmp=tmp->next;

}

if((max-min)>1000)return NULL;

one\_min->val=head->val;

head->val=min;

gap=max-min+1;

ListNode\* counter[gap];

ListNode\* tail[gap];

for(int i=0;i<gap;i++)

counter[i]=NULL;

ListNode\* iter=head->next;

int index;

while(iter!=NULL)

{

tmp=iter->next;

index = iter->val-min;

if(counter[index]==NULL)tail[index]=iter;

iter->next=counter[index];

counter[index] = iter;

iter=tmp;

}

ListNode \*last,\*first;

last=first=NULL;

int i;

for(i=0;i<gap;i++)

{

if(counter[i]!=NULL)

{

first = counter[i];

last = tail[i];

i++;

break;

}

}

for(;i<gap;i++)

{

if(counter[i]!=NULL)

{

last->next = counter[i];

last = tail[i];

}

}

head->next = first;

return head;

}

ListNode \*sortList(ListNode \*head) {

if(head==NULL)return NULL;

ListNode\* result=countSort(head);

if(result!=NULL){

head = result;

return head;

}

part\_sort(head, NULL);

return head;

}

};

# Evaluate Reverse Polish Notation

## 题目

. Evaluate the value of an arithmetic expression in Reverse Polish Notation.

Valid operators are +, -, \*, /. Each operand may be an integer or another expression.

Some examples:

["2", "1", "+", "3", "\*"] -> ((2 + 1) \* 3) -> 9

["4", "13", "5", "/", "+"] -> (4 + (13 / 5)) -> 6

## 解法

### 分析

 .很简单的堆栈应用

### 程序

. class Solution {

public:

int evalRPN(vector<string> &tokens) {

stack<int> mstack;

vector<string>::iterator iter;

try

{

for(iter=tokens.begin();iter!=tokens.end();iter++)

{

if(\*iter=="+"){

int a,b;

a = mstack.top();

mstack.pop();

b = mstack.top();

mstack.pop();

mstack.push(b+a);

}

else if(\*iter=="-")

{

int a,b;

a=mstack.top();

mstack.pop();

b=mstack.top();

mstack.pop();

mstack.push(b-a);

}

else if(\*iter=="\*")

{

int a,b;

a=mstack.top();

mstack.pop();

b=mstack.top();

mstack.pop();

mstack.push(b\*a);

} //

else if(\*iter=="/")

{

int a,b;

a=mstack.top();

mstack.pop();

b=mstack.top();

mstack.pop();

mstack.push(b/a);

}

else

{

mstack.push(atoi(iter->c\_str()));

}

}

}

catch(int i)

{

}

return mstack.top();

}

};

附录 未完成题目

# Valid Number

## 题目

.. Validate if a given string is numeric.

Some examples:

"0" => true

" 0.1 " => true

"abc" => false

"1 a" => false

"2e10" => true

Note: It is intended for the problem statement to be ambiguous. You should gather all requirements up front before implementing one.

## 解法

### 分析

 ..

### 程序

..

# Palindrome Partitioning II

## 题目

. Given a string s, partition s such that every substring of the partition is a palindrome.

Return the minimum cuts needed for a palindrome partitioning of s.

For example, given s = "aab",

Return 1 since the palindrome partitioning ["aa","b"] could be produced using 1 cut.

## 解法

### 分析

 .

### 程序

.

# Max Points on a Line

## 题目

. Given n points on a 2D plane, find the maximum number of points that lie on the same straight line.

## 解法

### 分析

 .

### 程序

.

# Reverse Words in a String

## 题目

. Given an input string, reverse the string word by word.

For example,

Given s = "the sky is blue",

return "blue is sky the".

## 解法

### 分析

 .

### 程序

.