注：其中本人第十题未通过测试，本次写的代码是比赛结束后经过修改并测试的代码

1. LCS
2. #include<stdio.h>
3. #include<string.h>
4. **char** a[1001],b[1001];   //创两个char数组变量用来接受输入
5. **int** dp[1001][1001],len1,len2;   //使用dp二位数组保存回溯输出结果
6. **void** lcs(**int** i,**int** j){
7. **for**(i=1; i<=len1; i++){
8. **for**(j=1; j<=len2; j++){           //两个for循环
9. **if**(a[i-1]==b[j-1])              //判断两相同下标元素的相等关系
10. dp[i][j]=dp[i-1][j-1]+1;    //dp[i][j]直接为左上角数字加一
11. **else** **if**(dp[i-1][j]>dp[i][j-1])      //否则的话找左或上的最大值
12. dp[i][j]=dp[i-1][j];
13. **else**
14. dp[i][j]=dp[i][j-1];
15. }
16. }
17. printf("%d\n",dp[len1][len2]);
18. }
19. **int** main(){
20. **int** test;
21. **int** x;
22. scanf("%d\n",&x);
23. test=x;
24. **while**(1)
25. {
26. **if**(test)
27. {
28. test--;
29. //  printf("%d",test);
30. scanf(" %s",a);
31. scanf(" %s",b);
32. memset(dp,0,**sizeof**(dp));   //使用memset初始化数组
33. len1=strlen(a);    //两个数组长度
34. len2=strlen(b);
35. lcs(len1,len2);
36. }
37. **else**
38. **return** 0;
39. }
40. **return** 0;
41. }

2. Knapsack1.0

1. #include <iostream>
2. #include<algorithm>
3. #include<iomanip>
4. **using** **namespace** std;
5. **struct** work //定义work结构体，用于保存各自之后的平均量
6. {
7. **double** weight,value,avg;
8. };
9. **bool** cmp(work a ,work b)//cmp数组。用来比较平均量
10. {
11. **return** a.avg<b.avg;
12. }
13. **int**  main()
14. {
15. **int** n,i,k,x;
16. work \*wk1;
17. **double** w;
18. **while**(cin>>n>>w)   //利用while，首先输入n和w
19. {
20. **int** weightp=n;
21. **if**(n==0 && w==0) **break**;   //两个都为0直接跳出循环
22. wk1=**new** work[n];  //定义新结构体
23. i=k=0;
24. **while**(n--)
25. {
26. cin>>wk1[i].value;//输入各自的价值
27. i++;
28. }
29. **while**(weightp--)  //类似上个while循环，用来输入各自重量
30. {
31. cin>>wk1[k].weight;
32. wk1[k].avg=wk1[k].value/wk1[k].weight;   //计算avg，这是1.0和2.0的区别（可分解）
33. k++;
34. }
35. i--;
36. sort(wk1,wk1+i+1,cmp);    //sort进行排序
37. **int** j;
38. **double** sum=0;
39. **for**(j=i;j>=0;j--)
40. {
41. **if**(w>=wk1[j].weight)  //判断w和物品的重量关系
42. {
43. sum+=wk1[j].value;//装上，并减去相应重量
44. w-=wk1[j].weight;
45. //  cout<<sum<<' ';
46. }
47. **else**
48. {
49. sum+=wk1[j].avg\*w;  //否则sum加上avg乘背包承重
50. //  cout<<sum<<' ';
51. **break**;
52. }
53. }
54. cout<<fixed<<setprecision(3)<<sum<<endl;  //按题意输出
55. }
56. **return** 0;
57. }

3. Knapsack2.0

1. #include <stdio.h>
2. #define MAX\_N 1005
3. #define MAX\_W 10005
4. #define max(a,b) a>b?a:b //宏定义max定义
5. **int** dp[MAX\_N][MAX\_W];   //dp二维数组，用来保存运算结果
6. **int** N, W, v[MAX\_N], w[MAX\_N];
8. **int** main() {
9. scanf("%d %d", &N, &W); //输入个数n和总重量w
10. **for** (**int** i = 0; i < N; i++) {
11. scanf("%d", &v[i]); //输入value
12. }
13. **for** (**int** i = 0; i < N; i++) {
14. scanf("%d", &w[i]); //输入weight
15. }
16. **for** (**int** j = 0; j <= W; j++) {
17. dp[0][j] = 0;
18. **for** (**int** i = 1; i <= N; i++) {       //核心代码，背包问题一行一行构建dp数组
19. **if** (w[i - 1] > j) {  //如果该重量大于j
20. dp[i][j] = dp[i - 1][j];    //则不装
21. } **else** {
22. dp[i][j] = max(dp[i - 1][j], v[i - 1] + dp[i - 1][j - w[i - 1]]);   //否则就比较装下和不装影响的价值差别，并取最大
23. }
24. }
25. }
26. printf("%d\n", dp[N][W]);
27. **return** 0;
28. }

4. MAZE

1. #include <iostream>
2. #include <queue>
3. **using** **namespace** std;
4. **const** **int** INF=10000000;
5. **typedef** pair<**int**,**int**> P;  //第一次知道的用法，用来把两个类型的物品装到一个中
6. **const** **int** MAX\_N=1000;
7. **char** maze[MAX\_N][MAX\_N];
8. **int** N,M;
9. **int** sx,sy;//起点坐标
10. **int** gx,gy;//终点坐标
11. **int** d[MAX\_N][MAX\_N];//最短距离
12. **int** dx[4]={1,0,-1,0};
13. **int** dy[4]={0,1,0,-1};
15. **int** bfs()   //使用广度优先搜索进行查询
16. {
17. queue<P> que;
18. **for**(**int** i=0;i<N;i++)
19. **for**(**int** j=0;j<M;j++)
20. {
21. d[i][j]=INF;
22. **if**(maze[i][j]=='S') sx=i,sy=j;  //如果是S，就记录下起点
23. **if**(maze[i][j]=='E') gx=i,gy=j;  //同理，记录下终点
24. }
25. que.push(P(sx,sy));
26. d[sx][sy]=0;
28. **while**(que.size())
29. {
30. P p=que.front();
31. que.pop();
32. **if**(p.first==gx&&p.second==gy) **break**;
33. **for**(**int** i=0;i<4;i++) //使用方向
34. {
35. **int** nx=p.first+dx[i],ny=p.second+dy[i];
36. **if**(0<nx&&nx<N&&0<=ny&&ny<M&&maze[nx][ny]!='1'&&d[nx][ny]==INF)//d[x][y]是否等于INF判断改点是否已经被搜索过
37. {//(nx, ny)没有出界，不是障碍且没被访问过
38. que.push(P(nx,ny)); //入栈
39. d[nx][ny]=d[p.first][p.second]+1;
40. }
41. }
42. }
43. **return** d[gx][gy];   //返回
44. }
46. **void** solve()
47. {
48. **int** res=bfs();
49. cout<<res<<endl;
50. }
52. **int** main(**int** argc, **char** **const** \*argv[])
53. {
54. cin>>N>>M;      //输入行数和列数
55. **for**(**int** i=0;i<N;i++)
56. **for**(**int** j=0;j<M;j++)
57. {
58. cin>>maze[i][j];  //输入迷宫构成
59. }
60. solve();    //进行解决
61. **return** 0;
62. }

5. Shortest Distance

1. #include<iostream>
2. **using** **namespace** std;
3. **int** main()  //使用Bellman-Ford算法
4. {
5. **int** dis[105],i,k,n,m,u[105],v[105],w[105],flag=0;   //因为题目要求n，w上限都是100，所以没有定太大
6. **int** inf=9999999;    //近似的无限大
7. cin>>n>>m;      //输入点数和边
8. **for**(i=1;i<=m;i++)
9. cin>>u[i]>>v[i]>>w[i];    //按要求输入
10. **for**(i=1;i<=n;i++)
11. dis[i]=inf;
12. dis[1]=0;
13. **for**(k=1;k<=n-1;k++)  //外循环循环了n-1次（n为顶点个数）
14. **for**(i=1;i<=m;i++)    //内循环循环了m次（m为边的个数），即枚举每一条边
15. **if**(dis[v[i]] > dis[u[i]+w[i]])
16. dis[v[i]] = dis[u[i]+w[i]];
17. //其中外循环因为在一个含有n个顶点的图中，任意两点之间的最短路径最多包含n-1条边
18. {
19. **for**(i=2;i<=n;i++)
20. **if**(dis[i]!=9999999)//根据dis[i]来判断是否联通
21. {
23. **if**(i>2)
24. cout<<" ";
25. cout<<dis[i];}    //联通，直接输出最短路径
26. **else**
27. {
29. **if**(i>2)
30. cout<<" ";
31. cout<<-1;}    //否则输出-1
32. }
33. printf("\n");
34. **return** 0;
35. }

6. MST

1. #include <cstdio>
2. #include <algorithm>
3. #include <cstring>
4. #define maxn 10005  //题意是10000，所以定义10005
5. **using** **namespace** std;
6. **int** pre[maxn];  //前驱数组
7. **long** **long** sum ;
8. **int** \_cout;
9. **struct**  edge    //定义的边，包括起点终点权值
10. {
11. **int** u;
12. **int** v;
13. **int** w;
14. };
15. edge e[maxn];   //初始化边
16. **int** Find(**int** x)
17. {    **int** r =x;
18. **while**(pre[r] != r) //若两者不等
19. r = pre[r];  //则令pre数组和r
20. **int** i=x,j;
21. **while**(i != r)  //利用中间值改变pre[i]数组值为r
22. {
23. j =pre[i];
24. pre[i] = r;
25. i = j;
26. }
27. **return** r;
28. }
29. **int** join(**int** x,**int** y)   //判断是否当成子树进行连接
30. {      **int** fx = Find(x);
31. **int** fy = Find(y);
32. **if**(fx != fy)
33. {
34. pre[fx] = fy;   //若fx前驱为fy，直接返回1
35. **return** 1;
36. }
37. **return** 0;
38. }
39. **bool** cmp(edge a, edge b)    //比较权重的函数
40. {
41. **return** a.w<b.w;
42. }
43. **int** main()
44. {
45. **long** **long** temp[10];
46. **int** n,m,i,tmp;
47. scanf("%d",&i); //输入计算次数
48. tmp = i;
49. **while**(i--)
50. {scanf("%d %d",&n,&m); //输入点数n和边数m
51. sum = 0;
52. \_cout = 0;
53. memset(e,0,**sizeof**(e));
54. memset(pre,0,**sizeof**(pre));
55. **for**(**int** i=1;i<=m;i++)
56. scanf("%d %d %d",&e[i].u,&e[i].v,&e[i].w);  //输入边的属性
57. sort(e+1,e+m+1,cmp);    //进行排序
58. **for**(**int** i=1;i<=n;i++)    //利用循环控制pre数组的值
59. pre[i] = i;
60. **for**(**int** i=1;i<=m;i++)
61. {
62. **if**(join(e[i].u , e[i].v))    //如果是否入最小生成树，即加入集合
63. {
64. \_cout ++ ;     //记录边数
65. sum += e[i].w;     //加上此权值
66. }
67. **if**(\_cout == n-1) //最后到了n-1的时候，直接退出
68. **break**;
69. }
70. temp[i] = sum;
71. //printf("%d\n",sum);
72. }
73. **for**(**int** k=tmp-1;k>-1;k--)
74. printf("%lld\n",temp[k]);
75. **return** 0;
76. }

7. The Huffman tree

1. #include <stdio.h>
2. #include <string.h>
3. #include <algorithm>
4. #include<iostream>
5. #include<stack>
6. **using** **namespace** std;
7. **int** result[5005];
8. **int** main(){
9. **int** n,i,num,k,temp;
10. **long** **long** sum,tmp[10];  //最后的值可能是long long类型，因为这个增值幅度很大
11. scanf("%d",&k); //输入的次数
12. temp = k;   //为之后输出做铺垫
13. **while**(k--){
14. scanf("%d",&n); //n个点
15. memset(result,0,n);
16. **for**(i=0;i<n;i++)
17. scanf("%d",&result[i]); //输入点的权值
18. sort(result,result+n);  //直接排序
19. i=1;
20. sum=0;
21. **while**(i<n){
22. sort(result+i-1,result+n);   //首先挨个排序
23. num = result[i-1]+result[i]; //找到两个最小的
24. sum+=num;        //然后加到sum中
25. result[i]=num;
26. i++;
27. }
28. tmp[k]=sum;
29. }
30. **for**(**int** x =temp-1;x>-1;x-- ) //在这里强行改变了多组输入一组输出
31. printf("%lld\n",tmp[x]);
32. **return** 0;
33. }

8. KMP

1. #include <stdio.h>
2. #include <string.h>
3. #define MAXL 100001 //没有规定定义多大，保险起见定义见到的数组
4. **char** S[MAXL];
5. **char** T[MAXL];
6. **int** failure[MAXL], mL, pL;  //用来记录匹配失败的数组
7. **int** matchPositions[MAXL];   //记录匹配点的数组
9. **int** stringMatch();
10. **void** makeFailure(**int**);
12. **int** main () {
13. **int** k, matchNum;
14. scanf("%s %s", S, T);   //输入两个数组
15. mL = strlen(S); //两个数组
16. pL = strlen(T);
17. matchNum = stringMatch();   //stringMatch函数来判断是否成功
18. **if**(matchNum)
19. printf("YES\n");    //根据stringMatch返回值来判断是否正确
20. **else**
21. printf("NO\n");
22. **return** 0;
23. }
24. **int** stringMatch() { //关键函数
25. **int** mk = 0, pk = 0, foundCnt = 0;
27. makeFailure(pL);
29. **while**(mk < mL) { //进行匹配，其中次数是第一个数组的长度
31. **if**(T[pk] == S[mk]) {    //如果匹配
32. **if**(pk == pL - 1) {  //匹配进行到最后
33. matchPositions[foundCnt] = mk - pk; //匹配位置数组进行记录
34. foundCnt++;
35. pk = failure[pk];   //pk进行记录失败匹配数组
36. **if**(pk == -1)
37. pk++, mk++;
38. }
39. **else**
40. pk++, mk++;
41. }
42. **else** {
43. pk = failure[pk];
45. **if**(pk == -1)
46. pk++, mk++;
47. }
48. }
49. **return** foundCnt;    //返回是否找到
50. }
52. **void** makeFailure(**int** pL) {
53. **int** curLen = 0;
54. **int** k = 1;
56. failure[0] = -1;
57. failure[1] = 0;
59. **while**(k < pL) {  //循环后一个数组的个数
61. **if**(T[k] == T[curLen]) { //判断当前长度字符和k字符是否相等
62. curLen++;
63. failure[k + 1] = curLen;
64. k++;//同时向下移动
65. }
67. **else** {
68. **if**(curLen == 0) {   //判断当前长度大小
69. failure[k + 1] = 0;//令测试失败数组k+1置为0
70. k++;
71. }
73. **else**
74. curLen = failure[curLen];   //否则记录当前长度
75. }
76. }
78. **return**;
79. }

9. 5G is coming!!!

1. #include <cstdio>
2. #include <iostream>
3. #include <algorithm>
4. **using** **namespace** std;
5. **int** n, c;
6. **int** pos[100005];
7. **bool** judge(**int** k)    //判断函数，其中参数是判断的长度
8. {
9. **int** cnt = 1;
10. **int** st = pos[0];
11. **for**(**int** i = 1; i < n; ++i)   //便利n-1次
12. {
13. **if**(pos[i] - st >= k) //倘若所选点-st比参数大
14. {
15. ++cnt;
16. **if**(cnt >= c)
17. **return** **true**;    //次数大于所输个数，直接返回
18. st = pos[i];        //否则重置st的值
19. }
20. }
21. **return** **false**;
22. }
24. **int** binary\_search() // 二分枚举满足条件的最大距离
25. {
26. **int** left = 0;   //设置左右范围
27. **int** right = pos[n-1] - pos[0];
28. **int** mid = (left + right) / 2;
29. **while**(left <= right)
30. {
31. **if**(judge(mid)) // 若所求距离 >= mid，可以继续增大试探
32. left = mid+1;
33. **else**           // 否则所求距离 < mid,必须减小来试探
34. right = mid-1;
35. mid = (left + right) >> 1;
36. }
37. **return** left-1;
38. }
40. **int** main()
41. {
42. **int** i,tmp,temp[5];
43. scanf("%d",&i);
44. tmp = i;    //输入计算次数
45. **while**(i--)
46. {scanf("%d%d", &n, &c);//输入基站位置数和个数
47. **for**(**int** i = 0; i < n; ++i)
48. scanf("%d", &pos[i]);
49. sort(pos, pos+n);   //首先进行排序
50. temp[i] = binary\_search();//通过二分枚举的贪心算法实现
51. }
52. **for**(**int** k=tmp-1;k>-1;k--)
53. printf("%d\n",temp[k]);
54. **return** 0;
55. }

10. School Information Management System

1. #include<stdio.h>
2. #include<stdlib.h>    //使用了malloc等函数
3. #include<string.h>    //字符串比较等函数
4. **typedef** **struct** student  //定义结构体student
5. {
6. **char** name[20];
7. **int** sex;
8. **int** id;
9. **int** class1;
10. **struct** student\* next;   //一个链表指针
11. }student;
12. student\* head = NULL;   //首先指空
13. **int** length = 0; //首先链表长度置零
14. **int** state[20];  //状态，用来当作删除黑名单
15. **int** cont = -1;  //记录命令的次数，在主函数中使用
16. **void** create()   //一开始的create()函数，用来创建链表
17. {
18. student\* p1, \* p2;
19. p1 = (student\*)malloc(**sizeof**(student)); //指向创建位置的指针
20. p1->id = -1;
21. **if** (head == NULL)
22. head = p1;
23. p2 = (student\*)malloc(**sizeof**(student));
24. scanf("%s %d %d %d", p2->name, &p2->sex, &p2->id, &p2->class1); //输入
25. **for** (**int** i = 0; i < cont; i++)
26. **if** (state[i] == p2->id)  //判断是否被删除
27. **return**;
28. length++;   //增加长度
29. p1->next = p2;
30. p2->next = NULL;
31. p1 = p1->next;
32. **return**;
33. }
34. **void** insert()   //插入函数，在链表存在的情况下使用
35. {
36. **if** (length == 0)    //长度为0直接跳到create()函数，并返回
37. {
38. create();
39. **return**;
40. }
41. **int** num, i;
42. student\* p, \* q;
43. p = head;
44. num = length;
45. {
46. q = (student\*)malloc(**sizeof**(student));
47. scanf("%s %d %d %d", q->name, &q->sex, &q->id, &q->class1);
48. **for** (**int** i = 0; i < cont; i++)
49. **if** (state[i] == q->id)   //仍然是黑名单的判断
50. **return**;
51. **while** (p != NULL)
52. {
53. **if** (p->id == q->id)   //如果有重复id，会不向下处理链表
54. {
55. **return**;
56. }
57. p = p->next;
58. }
59. p = head;
60. **for** (i = 0; i < num; i++)
61. p = p->next;
62. q->next = p->next;
63. p->next = q;
64. length++;
65. **return**;
66. }
67. }
68. **void** Delete()   //删除函数
69. {
70. **int** num;
71. student\* p, \* q;
72. q = head, p = head->next;
73. scanf("%d", &num);
74. state[cont] = num;  //设置黑名单
75. **while** (p != NULL)
76. {
77. **if** (p->id == num)    //判断黑名单是否有此id
78. {
79. q->next = p->next;
80. free(p);
81. length--;
82. **return**;
83. }
84. p = p->next;
85. q = q->next;
86. }
87. **if** (p == NULL)
88. {
89. **return**;
90. }
91. }
92. **void** search()   //查询id函数
93. {
94. **int** num;
95. student\* p = head->next;
96. scanf("%d", &num);
97. **while** (p != NULL)
98. {
100. **if** (p->id == num)    //while循环找到id
101. {
102. printf("%s %d %02d %03d\n", p->name, p->sex, p->id, p->class1);
103. **return**;
104. }
106. p = p->next;
107. }
108. **if** (p == NULL)  //没有找到的话直接输出-1
109. printf("-1\n");
110. **return**;
111. }
112. **void** search1()  //查询class函数
113. {
114. **int** num, count;
115. student\* p = head->next;
116. **char**  temp[50];
117. **char** str[20][50];
118. scanf("%d", &num);
119. count = 0;
120. **while** (p != NULL)
121. {
123. **if** (p->class1 == num)
124. {
125. strcpy(str[count], p->name);
126. count++;    //之后的函数要使用的变量，即计数变量
127. //              printf("%s\n",p->name);
128. }
129. p = p->next;
130. }
132. printf("%d\n", count);
133. **if** (count == 0)
134. **return**;
135. //  print(str);
136. **for** (**int** i = 0; i < count; i++) {
137. **for** (**int** j = 0; j < count - i; j++) {
138. **if** (strcmp(str[j], str[j + 1]) > 0) {    //按字母表顺序输出
139. strcpy(temp, str[j]);
140. strcpy(str[j], str[j + 1]);
141. strcpy(str[j + 1], temp);
142. }
143. }
144. }

147. **for** (**int** i = 0; i < count; i++) {    //根据第n个输出判断输出状态
148. **if** (i == count - 1)
149. printf("%s\n", str[i]);
150. **else**
151. printf("%s ", str[i]);
152. }

155. **return**;
156. }
157. **void** search2()  //根据性别搜索
158. {
159. **int**  num;
160. student\* p = head->next;
161. //  printf("sex");
162. **char**  temp[50];
163. **char** str[20][50];
164. scanf("%d", &num);
165. **int** count = 0;
166. **while** (p != NULL)
167. {
169. **if** (p->sex == num)   //和class同理
170. {
171. strcpy(str[count], p->name);
172. count++;
173. //          printf("%s\n",p->name);
174. }
175. p = p->next;
176. }
178. printf("%d\n", count);
179. **if** (count == 0)
180. **return**;
181. **for** (**int** i = 0; i < count; i++) {
182. **for** (**int** j = 0; j < count - i; j++) {
183. **if** (strcmp(str[j], str[j + 1]) > 0) {
184. strcpy(temp, str[j]);
185. strcpy(str[j], str[j + 1]);
186. strcpy(str[j + 1], temp);
187. }
188. }
189. }

192. **for** (**int** i = 0; i < count; i++) {
193. **if** (i == count - 1)
194. printf("%s\n", str[i]);
195. **else**
196. printf("%s ", str[i]);
197. }

200. **return**;
201. }
203. **void** init() //初始化函数，这个函数只是做一些初始化操作，在不同环境下运行会出现很奇怪的bug
204. {
205. **int**  num;
206. student\* p = head->next;
207. //  printf("sex");
208. **char**  temp[50];
209. **char** str[20][50];
210. num = 0;
211. **int** count = 0;
212. **while** (p != NULL)
213. {
215. **if** (p->sex == num)
216. {
217. strcpy(str[count], p->name);
218. count++;
219. //          printf("%s\n",p->name);
220. }
221. p = p->next;
222. }
223. **if** (count == 0)
224. **return**;
225. //  printf("%d\n",count);
227. **for** (**int** i = 0; i < count; i++) {
228. **for** (**int** j = 0; j < count - i; j++) {
229. **if** (strcmp(str[j], str[j + 1]) > 0) {
230. strcpy(temp, str[j]);
231. strcpy(str[j], str[j + 1]);
232. strcpy(str[j + 1], temp);
233. }
234. }
235. }


239. **return**;
240. }
242. **int** main()  //主函数
243. {
244. **int** a;
245. **int** j;
246. scanf("%d", &j);
247. **while** (j--)
248. {//printf("%d\n",cont);
249. cont++;
250. scanf("%d", &a);
251. **switch** (a)
252. {
253. **case** 0:
254. **return** 0;
255. **case** 1:
256. insert();
257. **break**;
258. **case** 2:
259. **if** (head)
260. {
261. Delete();
262. }
263. **else**
264. {
265. }
266. **break**;
267. **case** 3:
268. **if** (head)
269. {
270. search();
271. }
272. **else**
273. {
274. }
275. **break**;
276. **case** 4:
277. **if** (head)
278. {
279. init();
280. search1();
281. }
282. **else**
283. {
284. }
285. **break**;
286. **case** 5:
287. **if** (head)
288. {
289. init();
290. search2();
291. }
292. **else**
293. {
294. }
295. **break**;

298. }

301. }
302. }