**310．**最长子串

**题目描述**

给定一个长度为n的数组a，数组a的子串[i,j]表示a[i],a[i+1],a[i+2]……a[j]。求最大长度的子串，该子串必须满足：m1<=max(a[i],a[i+1]……a[j])-min(a[i],a[i+1]……a[j])<=m2。如果最大长度的子串有多个，请找出子串和最大的那个。对于子串[i,j]，子串和指的是Sum=a[i]+a[i+1]+a[i+2]+……a[j]，子串长度指的是length=j-i+1。如果没有子串满足，请输出“0 0”。

**解答要求**时间限制：2000ms, 内存限制：64MB

**输入**

每个测试用例占两行，第一行为三个整数n,m1,m2。第二行为n个整数，从左到右依次为a[1],a[2]……a[n]。  
1<=n<=100000,0<=m1,m2,ai<=1000000。

**输出**

每个用例占一行，包含两个整数a,b。a表示你找到的子串的长度，b表示该子串的和。

**样例**

输入样例 1 复制

3 0 0

1 1 1

输出样例 1

3 3

**提示**

解题报告：  
假设，我们现在有一堆数，这一堆数的最大值减去最小值为ans。那么，我们如果再向这堆数中添加一个数，那么ans要么增大，要么不变。同样的，如果我们从这堆数中删掉一个数，ans要么变小，要么不变。  
我们使用“双指针”技巧来完成这道题。我们定义两个指针l和r，对于每个l我们要求出最大的r使得区间[l,r]是满足要求的。由上面的结论，如果l1<l2，那么r1<=r2.  
算法具体实现时，我们只要让l从1循环到n，r不断的向右跑以满足要求就可以了。求一堆数的最小最大值，并可以进行插入删除操作。这一部分用任意一种二叉搜索树就可以完成，比如红黑树。

<http://oj.rnd.huawei.com/discuss/problems/discussions/966fdcc8-5890-4eb7-97ae-6220151c5765?navName=%E5%8F%8C%E6%8C%87%E9%92%88>

解题报告：  
假设，我们现在有一堆数，这一堆数的最大值减去最小值为ans。那么，**我们如果再向这堆数中添加一个数，那么ans要么增大，要么不变。同样的，如果我们从这堆数中删掉一个数，ans要么变小，要么不变。**  
我们使用“双指针”技巧来完成这道题。我们定义两个指针l和r，对于每个l我们要求出最大的r使得区间[l,r]是满足要求的。由上面的结论，如果l1<l2，那么r1<=r2.  
算法具体实现时，我们只要让l从1循环到n，r不断的向右跑以满足要求就可以了。求一堆数的最小最大值，并可以进行插入删除操作。这一部分用任意一种二叉搜索树就可以完成，比如红黑树。

1. #include<set>
2. #define MAX(a, b) (a > b ? a : b)
3. #define MIN(a, b) (a < b ? a : b)
4. long n, m1, m2, num[100005], sum[100005];
5. long a, b;
6. multiset<long, less<long>> subarr;
7. int main(){
8. cin >> n >> m1 >> m2;
9. for(int i = 1; i <= n; i++) {
10. cin >> num[i];
11. sum[i] = num[i] + sum[i - 1];
12. }
13. subarr.insert(num[1]);
14. int l = 1, r = 1;
15. while (l <= n && r <= n && l <= r) {
16. long ans = \*(--subarr.end()) - \*subarr.begin();
17. if (m1 <= ans && ans <= m2) {
18. if (r - l + 1 > a) {
19. a = r - l + 1;
20. b = sum[r] - sum[l - 1];
21. } else if (r - l + 1 == a) {
22. b = MAX(b, sum[r] - sum[l - 1]);
23. }
24. subarr.insert(num[++r]);
25. } else if (ans > m2) {
26. subarr.erase(subarr.find(num[l++]));
27. } else if (ans < m1) {
28. subarr.insert(num[++r]);
29. }
30. }
31. cout << a << ' ' << b << endl;
32. return 0;
33. }

**219.**运树：

**题目描述**

在一条直线上有n处地方，从左到右依次编号为1, 2, …, n，每处地方都有一定量的树木，每处地方的单位量树木运送单位距离都需要一定的费用，例如在1处有数量为a的树木，单位量的树木运送单位距离的费用是b，那么把这么多树木一共运送c的距离所需要的费用为a\*b\*c。现在需要把这n处地方的树木送往加工中心处理，为了使得路径单一，所有地方的树木只能向右边（编号较大的地方的方向）运送，已知第n处的地方已经存在了一个加工中心。为了减少树木运送的费用，在这条路径上最多可以添加k个加工中心，使得总运费最少。问最小的运费是多少？

**解答要求**时间限制：2000ms, 内存限制：64MB

**输入**

输入数据一共包括4行，第1行输入n和k(2<=n<=10000, 1<=k<=min(200, n-1))，表示n处地方，最多添加k个加工中心。第2行输入n个正整数，表示每个地方的树木总量，第3行输入n个正整数，表示每个地方单位数量单位距离运送的费用。第4行输入n-1个正整数，表示从左到右相邻两个地方的距离。除n以外，所有的数字都<=300。

**输出**

输出一行一个整数，表示最小的运费。

**样例**

输入样例 1 复制

3 1

1 2 3

3 2 1

2 2

输出样例 1

6

**提示**

Carry\_tree简单题解

不难证明，加工厂只能建在节点上  
比较容易想到状态方程：dp[k][n] = min{0<=i<n}(dp[k - 1][i] + delta(i + 1, n))  
其中delta(i + 1, n)表示为从i + 1到n的所有树木运送到第n号节点的耗费总和  
复杂度O(k\*n\*n)，会超时，需要优化。  
采用斜率优化，我们令设几个变量。  
sum[n]表示从1~n的所有的树木搬到n号节点所需要的代价总和  
a[n]表示从1~n的所有的树木所搬运单位路程所需的代价  
d[n]表示从1~n的距离  
那么delta(i + 1, n) = sum[n] - sum[i] - a[i] \* (d[n] - d[i])  
所以原方程式化为dp[k][n] - sum[n] = min{0<=i<n}(dp[k - 1][i] - sum[i] + a[i] \* d[i] - a[i] \* d[n])  
因为a[i]单调递增，因而采用斜率优化。

<http://oj.rnd.huawei.com/discuss/problems/discussions/3b406a32-bb38-4673-b5fe-671bfe9f3227?navName=%E6%96%9C%E7%8E%87%E4%BC%98%E5%8C%96DP>

斜率优化动态规划，这题有难度，如果不理解可以先忽略，不是精选题目

优化的推导过程参考：<https://www.jianshu.com/p/f9fb15246e14>

1. #include<vector>
2. #define MIN(a, b) (a < b ? a : b)
3. long k, n, value[10005], dist[10005], f[10005][205];
4. long DeltaY(int p, int s, int o) {
5. return (f[p][o] - f[p][0] + value[p] \* dist[p]) - (f[s][o] - f[s][0] + value[s] \* dist[s]);
6. }
7. long DeltaX(int p, int s) {
8. return value[p] - value[s];
9. }
10. int main(){
11. cin >> n >> k;
12. for (int i = 1; i <= n; i++) {
13. cin >> value[i];
14. }
15. for (int i = 1; i <= n; i++) {
16. cin >> value[10004];
17. value[i] = value[10004] \* value[i] + value[i - 1];
18. }
19. for (int i = 2; i <= n; i++) {
20. cin >> dist[i];
21. dist[i] += dist[i - 1];
22. for (int j = 1; j <= i; j++) {
23. f[i][0] += (value[j] - value[j - 1]) \* (dist[i] - dist[j]);
24. }
25. }
26. for (int s = 1; s <= k + 1; s++) {
27. int head = 0, tail = 0;
28. vector<int> q(n + 1);
29. for (int i = s; i <= n; i++) {
30. int o = s - 1;
31. while(head < tail && DeltaY(q[head + 1], q[head], o) <= DeltaX(q[head + 1], q[head]) \* dist[i])
32. head++;
33. f[i][s] = f[q[head]][o] + f[i][0] - f[q[head]][0] - value[q[head]] \* (dist[i] - dist[q[head]]);
34. while(head < tail && DeltaY(i, q[tail], o) \* DeltaX(q[tail], q[tail - 1]) <= DeltaY(q[tail], q[tail - 1], o) \* DeltaX(i, q[tail]))
35. tail--;
36. q[++tail] = i;
37. }
38. }
39. cout << f[n][k] << endl;
40. return 0;
41. }

**249. Nanjing\_Travel:**

**题目描述**

华华假期的时候独自去南京玩，出发前他想制定一个旅行计划。假设南京有N个景点，华华给每个景点定义了一个开心值Si，也就是说当他游玩这个景点后他的总开心值会加Si，同时，游玩第i个景点会花费Ci的时间。由于没有妹子陪，所以他想在限定的时间T内，从起始点S，有选择的游览一些景点，最后到达终点E。当然他想让这次旅行所得的开心值最大。  
注意，华华可以为了走近路而只是路过一些景点，不去游玩（包括S和E）。而且他有一个怪癖就是要去玩的下一个景点的开心值一定要大于之前玩的景点（例如他游玩景点i获得的开心值为10，那么他游玩的下一个景点的开心值必须大于10）。此外，景点间的路是双向的，路上也要耗费时间，而且各个点之间可能不止一条路，华华当然会走最短了啦。

**解答要求**时间限制：1000ms, 内存限制：64MB

**输入**

每组测试数据格式如下：  
第一行包括5个整数：N M T S E。N代表景点的数量1<N<100，M代表道路的数量0<M<1000，T代表时间限制0<T<=300，S代表起点，E代表终点，0<=S,E<N。  
第二行包括N个整数Ci（0<=Ci<=T）,表示游玩景点i所要花费的时间。  
第三行包括N个整数Si（0<=Si<100）,表示游玩景点i可以得到的开心值。  
接下来M行，每行包括3个整数u,v,w,表示在u和v间有一条双向路，在路上要耗费w的时间(0<=u,v<N,0<=w<=T)。  
[/Input]

**输出**

输出一个整数，表示这次旅行可以获得的最大开心值。当然，如果不能再T时间内到达E，只需要输出“0”（没有引号）。

**样例**

输入样例 1 复制

4 4 22 0 3

1 1 1 1

5 7 9 12

0 1 10

1 3 10

0 2 10

2 3 10

输出样例 1

21

**提示**

先用Floyd算法求得任意两点间最短路,存到mp[i][j]里。用dp[i][j]表示在第i个景点游玩，时间为j的时候可以获得的最大开心值。则dp[i][k]=max(dp[i][k],dp[j][k-p[i].c-mp[i][j]]+p[i].s);其中p[i].c为游览的花费，p[i].s为游览可以获得的最大开心值。

<http://oj.rnd.huawei.com/discuss/problems/discussions/8d4d8e36-13fe-471b-9b15-5233d2a09e6a?navName=Floyd%20%2B%20%E9%98%9F%E5%88%97%EF%BC%8C0ms>

状态转移方程f[i][j]表示当在第i号景点happy并且还剩j个时间的最大幸福度。  
因为可以从任何**可达**的景点走过来，所以在处理状态的时候非常容易brainfuck。  
本题的实现思想就是状态转移，但是形式上采用队列。每当访问到景点e的时候就更新结果，可以不在终点happy。

1. #include <queue>
2. #define MIN(a,b) (a < b ? a : b)
3. #define MAX(a,b) (a > b ? a : b)
4. int map[105][105], cost[105], value[105], ans;
5. struct Node {
6. int pos, time, lastValue, value;
7. };
8. int main(){
9. int n, m, t, s, e;
10. cin >> n >> m >> t >> s >> e;
11. for (int i = 0; i < n; i++) cin >> cost[i];
12. for (int i = 0; i < n; i++) cin >> value[i];
13. for (int i = 0; i < n; i++)
14. for (int j = 0; j < n; j++)
15. if (i != j) map[i][j] = t + 1;
16. for (int i = 0; i < m; i++) {
17. int a, b, c;
18. cin >> a >> b >> c;
19. map[a][b] = MIN(map[a][b], c);
20. map[b][a] = map[a][b];
21. }
22. // floyd
23. for (int k = 0; k < n; k++) {
24. for (int i = 0; i < n; i++) {
25. for (int j = 0; j < n; j++) {
26. if (map[i][j] > map[i][k] + map[k][j])
27. map[i][j] = map[i][k] + map[k][j];
28. }
29. }
30. }
31. if (map[s][e] > t) {
32. cout << 0 << endl;
33. } else {
34. queue<Node> q;
35. Node temp;
36. temp.pos = s, temp.time = t, temp.lastValue = -1, temp.value = 0;
37. q.push(temp);
38. while(q.size()) {
39. Node head = q.front();
40. q.pop();
41. int pos = head.pos;
42. for(int i = 0; i < n; i++) {
43. if(i == e && (head.time - map[pos][i] >= 0)) ans = MAX(ans, head.value);
44. if (value[i] > head.lastValue) {
45. Node temp;
46. int time = head.time - map[pos][i] - cost[i];
47. if (time >= 0) {
48. temp.pos = i;
49. temp.lastValue = value[i];
50. temp.value = head.value + value[i];
51. temp.time = time;
52. q.push(temp);
53. }
54. }
55. }
56. }
57. cout << ans << endl;
58. }
59. return 0;
60. }

**214.黑白配：**

<http://oj.rnd.huawei.com/discuss/problems/discussions/6a68b43a-c923-4063-9677-4e73c1cf6313?navName=%E7%A6%BB%E6%95%A3%E5%8C%96%20%2B%20Multiset>

**题目描述**

设B = {b1, b2, …, bn} 和 W = {w1, w2, …, wn}分别为平面上黑点和白点的两个集合。一黑点bi = (xi, yi) 与一白点wj = (xj, yj) 匹配当且仅当 xi <= xj 和 yi <= yj 同时成立，每个点最多只能用于一次匹配，请找出黑白点之间的最大匹配数目。  
黑点和白点各自的数量均不超过100000；  
平面为(0, 0)到(10000, 10000)的矩形中的整数点  
黑点白点坐标可能相同，B集合、W集合中也可能包含相同的元素。

**解答要求**时间限制：2000ms, 内存限制：64MB

**输入**

输入的第一行包括一个整数T (1 <= T <= 10)，表示有T组测试数据.  
对于每组测试数据：  
第一行两个整数，n(黑点个数) m(白点个数)，0 < n, m <= 100000  
接下来n行每行有两个整数并用空格隔开：  
黑点的横坐标x 黑点的纵坐标y  
再接下来m行每行有两个整数同样用空格隔开：  
白点的横坐标x 白点的纵坐标y

**输出**

对于每组测试数据，输出一行一个整数，表示最大匹配数。

**样例**

输入样例 1 复制

2

2 2

1 0

0 1

1 1

0 0

1 1

1 1

0 0

输出样例 1

1

0

**提示**

将所有黑白色棋子按照x轴从小到大排序，这样我们减少考虑一维。每次遇到一个黑点，将其y坐标插入平衡树中，每次遇到一个白点，寻找平衡树中小于等于该白点y坐标的黑点去掉，答案加一。因为这样去对后面是最优的。如果你去小了，可能后面比较小的白点就没有办法匹配了。

**思路上可能有点贪心的思想，**假设问题是一维的，对所有黑点白点做离散化，然后从小到大扫描。因为白点总要大于等于黑点才可以匹配，所以每当我们扫描到一个白点时，将离他最近的黑点匹配到一起。

二维的情况就需要维护一个查找，这个查找每次能将Y轴最近的黑点与白点匹配。

离散化的时候注意x轴坐标相等就排序Y轴，x与y均相等，要把白点坐标排在后面。

1. #include<set>
2. #include<vector>
3. #include<algorithm>
4. struct Node {
5. int x, y;
6. bool black;
7. inline bool operator () (const Node &a, const Node &b) {
8. bool f = a.x < b.x;
9. if (a.x == b.x) {
10. f = a.y < b.y;
11. if (a.y == b.y) {
12. f = a.black;
13. }
14. }
15. return f;
16. }
17. };
18. int t;
19. int main(){
20. cin >> t;
21. while(t--) {
22. int n, m, ans = 0;
23. multiset<int, greater<int>> tree;
24. cin >> n >> m;
25. vector<Node> bw(n + m);
26. for(int i = 0; i < n; i++) {
27. Node d;
28. cin >> d.x >> d.y;
29. d.black = true;
30. bw[i] = d;
31. }
32. for(int i = 0; i < m; i++) {
33. Node d;
34. cin >> d.x >> d.y;
35. d.black = false;
36. bw[n + i] = d;
37. }
38. sort(bw.begin(), bw.end(), Node());
39. for (int i = 0; i < n + m;i++) {
40. Node &d = bw[i];
41. if (d.black) {
42. tree.insert(d.y);
43. } else {
44. multiset<int>::iterator lower = tree.lower\_bound(d.y);
45. if(lower != tree.end() && \*lower <= d.y) {
46. ans++;
47. tree.erase(lower);
48. }
49. }
50. }
51. cout << ans << endl;
52. }
53. return 0;
54. }