

```

1  ///二叉树相关
2
3  ///知道先序、中序，求后序
4  struct TreeNode
5  {
6      struct TreeNode* left;
7      struct TreeNode* right;
8      char elem;
9  };
10 void BinaryTreeFromOrderings(char* inorder, char* preorder, int length)
11 {
12     if(length == 0)
13     {
14         //cout<<"invalid length";
15         return;
16     }
17     TreeNode* node = new TreeNode;//Notice that [new] should be written out.
18     node->elem = *preorder;
19     int rootIndex = 0;
20     for(;rootIndex < length; rootIndex++)
21     {
22         if(inorder[rootIndex] == *preorder)
23             break;
24     }
25     //Left
26     BinaryTreeFromOrderings(inorder, preorder + 1, rootIndex);
27     //Right
28     BinaryTreeFromOrderings(inorder + rootIndex + 1, preorder + rootIndex + 1, length - (rootIndex
+ 1));
29     cout<<node->elem<<endl;
30     return;
31 }
32 int main(int argc, char* argv[])
33 {
34     printf("Hello World!\n");
35     char* pr="GDAFEMHZ";
36     char* in="ADEFGHMZ";
37     BinaryTreeFromOrderings(in, pr, 8);
38     printf("\n");
39     return 0;
40 }
41
42
43 ///知道中序、后序，求先序
44 struct TreeNode
45 {
46     struct TreeNode* left;
47     struct TreeNode* right;
48     char elem;
49 };
50 TreeNode* BinaryTreeFromOrderings(char* inorder, char* aftorder, int length)
51 {
52     if(length == 0)
53     {
54         return NULL;
55     }
56     TreeNode* node = new TreeNode;//Notice that [new] should be written out.
57     node->elem = *(aftorder+length-1);
58     std::cout<<node->elem<<std::endl;
59     int rootIndex = 0;
60     for(;rootIndex < length; rootIndex++)//a variation of the loop
61     {
62         if(inorder[rootIndex] == *(aftorder+length-1))
63             break;
64     }
65     node->left = BinaryTreeFromOrderings(inorder, aftorder , rootIndex);
66     node->right = BinaryTreeFromOrderings(inorder + rootIndex + 1, aftorder + rootIndex , length -
(rootIndex + 1));
67
68     return node;
69 }
70 int main(int argc, char** argv)
71 {
72     char* af="AEFDHZMG";
73     char* in="ADEFGHMZ";
74     BinaryTreeFromOrderings(in, af, 8);
75     printf("\n");

```

```

76     return 0;
77 }
78
79
80 ///层序遍历
81 /*二叉树结构体，并且构建了有参构造函数*/
82 struct BinaryTree
83 {
84     int vec;
85     BinaryTree* left;
86     BinaryTree* right;
87     BinaryTree(int data)
88         :vec(data), left(nullptr), right(nullptr){}
89 };
90 /*队列实现层序遍历*/
91 void printTree(BinaryTree* arr[])
92 {
93     queue<BinaryTree*> rel; //定义一个队列，数据类型是二叉树指针，不要仅是int！！不然无法遍历
94     rel.push(arr[0]);
95     while (!rel.empty())
96     {
97         BinaryTree* front = rel.front();
98         printf("%d\n", front->vec);
99         rel.pop(); //删除最前面的节点
100         if (front->left != nullptr) //判断最前面的左节点是否为空，不是则放入队列
101             rel.push(front->left);
102         if (front->right != nullptr) //判断最前面的右节点是否为空，不是则放入队列
103             rel.push(front->right);
104     }
105 }
106 int main()
107 {
108     /*构建二叉树*/
109     BinaryTree* s_arr[6];
110     s_arr[0] = new BinaryTree(0);
111     s_arr[1] = new BinaryTree(1);
112     s_arr[2] = new BinaryTree(2);
113     s_arr[3] = new BinaryTree(3);
114     s_arr[4] = new BinaryTree(4);
115     s_arr[5] = new BinaryTree(5);
116     s_arr[0]->left = s_arr[1]; // 0
117     s_arr[0]->right = s_arr[2]; // 1 2
118     s_arr[1]->left = s_arr[3]; // 3 5
119     s_arr[3]->left = s_arr[4]; //4
120     s_arr[2]->right = s_arr[5]; //所以层序遍历的结果为：0 1 2 3 5 4
121     /*层次遍历打印所有节点*/
122     printTree(s_arr);
123     /*释放所有空间*/
124     for (int i = 0; i < 6; i++)
125         delete s_arr[i];
126     return 0;
127 }
128
129
130 ///镜面反转
131 void MirrorRecursively(TreeNode* pRoot)
132 {
133     if((pRoot == NULL) || (pRoot->left == NULL && pRoot->right))
134         return;
135
136     TreeNode* pTemp = pRoot->left;
137     pRoot->left = pRoot->right;
138     pRoot->right = pTemp;
139
140     if(pRoot->left)
141         MirrorRecursively(pRoot->left);
142
143     if(pRoot->right)
144         MirrorRecursively(pRoot->right);
145 }
146

```

```

1  ///搜索相关
2
3  /*
4  迷宫的最短路径:
5      输入迷宫的长宽, s为起点、G为终点, #是墙。
6  */
7
8  #include <bits/stdc++.h>
9  using namespace std;
10
11  const int INF=100000000;
12  const int MAX=1000;
13
14  typedef pair<int, int> P;
15  int N, M;
16  int sx, sy;
17  int gx, gy;
18  char maze[MAX][MAX];
19  //设置数组, 表示到起点的最近距离
20  int d[MAX][MAX];
21  int dx[]={1,0,-1,0};
22  int dy[]={0,1,0,-1};
23
24  int bfs()
25  {
26      queue<P> que;
27      for(int i=0; i<N; i++)
28          for(int j=0; j<M; j++)
29              d[i][j]=INF;
30      //先把起点放进去, 距离设置为零
31      que.push(P(sx, sy));
32      d[sx][sy]=0;
33
34      //直到队列没有元素, 就无路可走了
35      while(que.size())
36      {
37          P p = que.front();
38          que.pop();
39          if(p.first==gx&&p.second==gy)
40              break;
41
42          for(int i=0; i<4; i++)
43          {
44              int nx=p.first+dx[i];
45              int ny=p.second+dy[i];
46              if(nx>=0&&nx<N&&ny>=0&&ny<M&&maze[nx][ny]!='#'&&d[nx][ny]==INF)
47              {
48                  que.push(P(nx,ny));
49                  d[nx][ny] = d[p.first][p.second]+1;
50              }
51          }
52      }
53      return d[gx][gy];
54  }
55
56  int main()
57  {
58      scanf("%d%d", &N, &M);
59      getchar();
60      for(int i=0; i<N; i++)
61      {
62          for(int j=0; j<M; j++)
63              scanf("%c", &maze[i][j]);
64          getchar();
65      }
66      for(int i=0; i<N; i++)
67          for(int j=0; j<M; j++)
68          {
69              if(maze[i][j]=='S')
70              {
71                  sx=i;
72                  sy=j;
73              }
74              if(maze[i][j]=='G')
75              {
76                  gx=i;
77                  gy=j;

```

```

78         }
79     }
80
81     int asw = bfs();
82     printf("%d\n", asw);
83
84
85     return 0;
86 }
87
88
89 /*
90 部分和问题:
91 从整数a1, a2...中判断是否可以从选出若干数, 使得他们的和恰好为k。
92 */
93
94 #include <bits/stdc++.h>
95 using namespace std;
96
97
98 int a[1000];
99 int n, k;
100
101 bool dfs(int i, int sum)
102 {
103     //已经遍历到末尾
104     if(i == n)
105         return sum==k;
106     //不加入a[i]
107     if(dfs(i+1, sum))
108         return true; //同对同错
109     //加入a[i]
110     if(dfs(i+1, sum+a[i]))
111         return true; //同对同错
112     //不管怎样都不行
113     return false;
114 }
115
116
117 int main()
118 {
119     scanf("%d", &n);
120     for(int i=0; i<n; i++)
121         scanf("%d", &a[i]);
122     scanf("%d", &k);
123
124     if(dfs(0, 0))
125         printf("YES\n");
126     else
127         printf("NO\n");
128
129     return 0;
130 }
131

```

```

1  ///数据结构相关
2
3  ///并查集
4  int par[MAX_N]; //父亲
5  int ran[MAX_N]; //树的高度
6  //初始化n个元素
7  void init(int n)
8  {
9      for(int i=0; i<n; i++)
10     {
11         par[i]=i;
12         ran[i]=0;
13     }
14 }
15 //查询树的根
16 int find(int x)
17 {
18     if(par[x]==x)
19         return x;
20     else
21         return par[x]=find(par[x]);
22 }
23 //合并x和y所属的集合
24 void unite(int x, int y)
25 {
26     x=find(x);
27     y=find(y);
28     if(x==y)
29         return ;
30     if(ran[x]<ran[y])
31         par[x]=y;
32     else
33     {
34         par[y]=x;
35         if(ran[x]==ran[y])
36             ran[x]++;
37     }
38 }
39 //判断x和y是否属于同一个集合
40 bool same(int x, int y)
41 {
42     return find(x)==find(y);
43 }
44
45
46 ///优先队列
47 priority_queue<int, vector<int>, greater<int> > que;
48 //pair也可以这么玩
49
50
51 ///二分
52 //注意返回的是指针
53 upper_bound(a, a+n, k) - lower_bound(a, a+n, k) //a中等于k的个数
54 //假定一个解判断是否可行
55 bool C()
56 {
57 }
58 }
59 void solve()
60 {
61     int lb=0, ub=INF;
62
63     for(int i=0; i<100; i++)
64         while(ub-lb>1)
65         {
66             int mid=(lb+ub)/2;
67             if(C(mid))
68                 lb=mid;
69             else
70                 ub=mid;
71         }
72     printf("%d\n", lb);
73 }
74 }
75
76
77 ///迭代器

```

```

78  Template::iterator ite;
79  //注意set和map进行迭代都是按运算符<进行排序过的
80
81
82  ///set(multiset)
83  s.begin()          // 返回指向第一个元素的迭代器
84  s.clear()          // 清除所有元素
85  s.count()          // 返回某个值元素的个数
86  s.empty()          // 如果集合为空, 返回true(真)
87  s.end()            // 返回指向最后一个元素之后的迭代器, 不是最后一个元素
88  s.erase()          // 删除集合中的元素
89  s.find()           // 返回一个指向被查找到元素的迭代器
90  s.insert()         // 在集合中插入元素
91  s.lower_bound()    // 返回指向大于(或等于)某值的第一个元素的迭代器
92  s.size()           // 集合中元素的数目
93  s.upper_bound()    // 返回大于某个值元素的迭代器
94
95
96  ///map(multimap)
97  /* 向map中插入元素 */
98  m[key] = value; // [key]操作是map很有特色的操作,如果在map中存在键值为key的元素对,
                  // 则返回该元素对的值域部分,否则将会创建一个键值为key的元素对,值域为默认值。所以可以用该操作向map中插入
                  // 元素对或修改已经存在的元素对的值域部分。
99  m.insert(make_pair(key, value)); //
                  // 也可以直接调用insert方法插入元素对,insert操作会返回一个pair,当map中没有与key相匹配的键值时,其first是
                  // 指向插入元素对的迭代器,其second为true;若map中已经存在与key相等的键值时,其first是指向该元素对的迭代器
                  // ,second为false。
100
101  /* 查找元素 */
102  int i = m[key]; //
                  // 要注意的是,当与该键值相匹配的元素对不存在时,会创建键值为key(当另一个元素是整形时, m[key]=0)的元素
                  // 对。
103  map<string, int>::iterator it = m.find(key); //
                  // 如果map中存在与key相匹配的键值时,find操作将返回指向该元素对的迭代器,否则,返回的迭代器等于map的end()(
                  // 参见vector中提到的begin()和end()操作)。
104
105  /* 删除元素 */
106  m.erase(key); // 删除与指定key键值相匹配的元素对,并返回被删除的元素的个数。
107  m.erase(it); // 删除由迭代器it所指定的元素对,并返回指向下一个元素对的迭代器。
108
109  /* 其他操作 */
110  m.size(); // 返回元素个数
111  m.empty(); // 判断是否为空
112  m.clear(); // 清空所有元素
113
114  ///vector
115  s[i] // 直接以下标方式访问容器中的元素
116  s.front() // 返回首元素
117  s.back() // 返回尾元素
118  s.push_back(x) // 向表尾插入元素x
119  s.size() // 返回表长
120  s.empty() // 表为空时, 返回真, 否则返回假
121  s.pop_back() // 删除表尾元素
122  s.begin() // 返回指向首元素的随机存取迭代器
123  s.end() // 返回指向尾元素的下一个位置的随机存取迭代器
124  reverse(s.begin(), s.end()); // 反转
125  s.insert(it, val) // 向迭代器it指向的元素前插入新元素val
126  s.insert(it, n, val) // 向迭代器it指向的元素前插入n个新元素val
127  s.insert(it, first, last) //
128  // 将由迭代器first和last所指定的序列[first, last)插入到迭代器it指向的元素前面
129  s.erase(it) // 删除由迭代器it所指向的元素
130  s.erase(first, last) // 删除由迭代器first和last所指定的序列[first, last)
131  s.clear() // 删除容器中的所有元素
132  s.swap(v) // 将s与另一个vector对象进行交换
133

```

```

1  ///数学相关
2
3
4  ///辗转相除法
5  int gcd(int a, int b)
6  {
7      if(b==0)
8          return a;
9      return gcd(b, a%b);
10 }
11
12
13 ///杨辉三角
14 long long int MOD = 1e9+7;
15 long long int c[2010][2010];
16
17
18     for(int i = 0; i < 2005; i++)
19     {
20         c[i][0] = 1;
21         c[i][i] = 1;
22     }
23     for(int i = 1; i < 2005; i++)
24         for(int j = 1; j < i; j++)
25             c[i][j] = (c[i-1][j-1] + c[i-1][j]) % MOD;
26
27
28 ///快速幂运算
29 typedef long long ll;
30 ll mod_pow(ll x, ll n, ll mod)
31 {
32     ll res=1;
33     while(n>0)
34     {
35         if(n&1)
36             res=res*x%mod; //如果二进制最低位为1, 则乘上x^(2^i)
37         x=x*x%mod; //将x平方
38         n>>=1;
39     }
40     return res;
41 }
42
43
44 ///埃氏筛法(Sieve of Eratosthenes)
45 int prime[MAX_N]; //第i个素数, 此为存放素数的数组
46 bool is_prime[MAX_N]; //is_prime[]为true时为素数
47
48 //返回n以内素数的个数
49 int sieveErato(int n)
50 {
51     int p=0;
52     for(int i=0; i<=n; i++)
53         is_prime[i]=true;
54     is_prime[0]=false;
55     is_prime[1]=false;
56     for(int i=2; i<=n; i++)
57     {
58         if(is_prime[i])
59         {
60             prime[p++]=i;
61             for(int j=2*i; j<=n; j+=i)
62                 is_prime[j]=false;
63         }
64     }
65     return p;
66 }
67
68
69 ///素性测试:
70 bool is_prime(int n)
71 {
72     for(int i=2; i*i<=n; i++)
73     {
74         if(n%i==0)
75             return false;
76     }
77     return n!=1; //1是例外

```

```

78     }
79
80
81     ///约数枚举：将约数存入一个vector中
82     vector<int> divisor(int n)
83     {
84         vector<int> res;
85         for(int i=1; i*i<=n; i++)
86         {
87             if(n%i==0)
88             {
89                 res.push_back(i);
90                 if(i!=n/i)
91                     res.push_back(n/i);
92             }
93         }
94         return res;
95     }
96
97     ///整数分解：将分解的数存入map的下标中，map值为因子的个数
98     map<int, int> prime_factor(int n)
99     {
100         map<int, int> res;
101         for(int i=2; i*i<=n; i++)
102         {
103             while(n%i==0)
104             {
105                 ++res[i];
106                 n/=i;
107             }
108         }
109         if(n!=1)
110             res[n]=1;
111         return res;
112     }
113
114     /*
115     多重组合数问题：
116     有n种物品，第i种物品有ai个。不同种类的物品可以互相区分但相同种类的不行，
117     从其中取出m个，有多少种取法？求方法数模M的值。
118     */
119
120     #include <cstdio>
121     #include <algorithm>
122     using namespace std;
123
124     int n, m;
125     int a[MAX_N];
126
127     int dp[MAX_N][MAX_M];
128
129     void solve()
130     {
131         //一个都不取得总是只有一种
132         for(int i=0; i<=n; i++)
133             dp[i][0]=1;
134         for(int i=0; i<n; i++)
135             for(int j=1; j<=m; j++)
136                 if(j-1-a[i+1]>=0)
137                     //在有取余的情况下，要避免减法运算的结果出现负数
138                     dp[i+1][j]=(dp[i+1][j-1]+dp[i][j]-dp[i][j-1-a[i+1]]+M)%M;
139                 else
140                     dp[i+1][j]=(dp[i+1][j-1]+dp[i][j])%M;
141     }
142
143
144     /*
145     划分数问题：
146     有n个无区别的物品，将它们划分成不超过m组，求划分方法数模M的余数。
147     1<=m<=n<=1000, 2<=M<=10000
148     (被称为n的m划分)
149     */
150     #include <cstdio>
151     #include <algorithm>
152     using namespace std;
153     const int MAX_M=1007, MAX_N=1007;
154     const int M=1e9+7;

```



```

155
156 int n, m;
157 int dp[MAX_M][MAX_N];
158
159 void solve()
160 {
161     dp[0][0]=1;
162     for(int i=1; i<=m; i++)
163         for(int j=0; j<=n; j++)
164             {
165                 if(j-i>=0)
166                     dp[i][j]=(dp[i-1][j]+dp[i][j-1])%M;
167                 else
168                     dp[i][j]=dp[i-1][j];
169             }
170 }
171
172 int main()
173 {
174     while(scanf("%d%d", &n, &m)!=EOF)
175     {
176         solve();
177         printf("%d %d\n", dp[m][n], dp[m-1][n]);
178     }
179
180     return 0;
181 }
182
183
184 /*
185 加练题: Joseph_POJ 1012
186 经典的约瑟夫问题, 找出递推式:
187 dp[i]表示为第i轮杀掉谁,
188 dp[i]=(dp[i-1]+m-1)%(n-i+1).
189 */
190 #include <cstdio>
191 #include <cstring>
192 #include <iostream>
193 #include <algorithm>
194 using namespace std;
195
196 int J[14];
197 int dp[29];
198
199 int main()
200 {
201     for(int k=1; k<14; k++)
202     {
203         int total=2*k;
204         int m=1;
205         for(int i=1; i<=k; i++)
206             {
207                 dp[i]=(dp[i-1]+m-1)%(total-i+1);
208                 if(dp[i]<=k-1)
209                     {
210                         m++;
211                         i=0;
212                     }
213             }
214         J[k]=m;
215     }
216
217     int k;
218     while(scanf("%d", &k)!=EOF)
219     {
220         if(k==0)
221             break;
222
223         printf("%d\n", J[k]);
224     }
225
226     return 0;
227 }
228

```

```

1  ///图论相关
2
3
4  /*
5  单源最短路问题(Bellman-Ford Algorithm):
6  单源最短路是固定一个起点, 求它到其他所有点之间的距离的最短路。(和两点间最短路问题复杂度一样)
7  常用于求解有负边的情况。
8  */
9  struct edge{
10     int from, to, cost;
11 }es[MAX_E];
12
13 int d[MAX_V]; //最短距离
14 int V, E;
15
16 void shortest_path(int s)
17 {
18     for(int i=0; i<V; i++)
19         d[i]=INF;
20     d[s]=0;
21     while(true) //如果不存在负圈, 复杂度是 $O(|V| \times |E|)$ 
22     {
23         bool update=false;
24         for(int i=0; i<E; i++)
25         {
26             edge e=es[i];
27             if(d[e.from]!=INF && d[e.to]>d[e.from]+e.cost)
28             {
29                 d[e.to]=d[e.from]+e.cost;
30                 update=true;
31             }
32         }
33         if(!update)
34             break;
35     }
36 }
37
38 /*
39 单源最短路问题(Dijkstra Algorithm):
40 由于不存在负边, 找到d[i]最小的顶点就是最短距离已经确定的顶点。
41 */
42 struct edge {
43     int to, cost;
44 };
45
46 typedef pair<int, int> P; //利用first表示最短距离, second表示顶点的编号
47
48 int V;
49 vector<edge> G[MAX_V]; //下标充当from
50 int d[MAX_V];
51
52 void dijkstra(int s)
53 {
54     //按照first从小到大顺序排列, 先取出最短距离和对应的顶点
55     priority_queue<P, vector<P>, greater<P>> > que;
56     fill(d, d+V, INF);
57     d[s]=0;
58     que.push(P(0,s));
59
60     while(que.size()) //复杂度为 $O(|E| \times \log |V|)$ 
61     {
62         P p = que.top();
63         que.pop();
64         int v=p.second;
65         if(d[v]<p.first)
66             continue;
67         for(int i=0; i<G[v].size(); i++)
68         {
69             edge e=G[v][i];
70             if(d[e.to]>d[v]+e.cost)
71             {
72                 d[e.to]=d[v]+e.cost;
73                 que.push(P(d[e.to],e.to));
74             }
75         }
76     }
77 }

```

```

78
79
80 /*
81 任意两点间最短路问题(Floyd-Warshall Algorithm):
82 用DP求解, 递推式为:
83  $d[0][i][j]=cost[i][j]$ 
84  $d[k][i][j]=\min(d[k-1][i][j], d[k-1][i][k]+d[k-1][k][j])$ 
85 依旧可以改进为 $d[i][j]$ 的形式。
86 复杂度:  $O(|V|^3)$ 。
87 可以处理边是负数的问题; 而判断是否有负圈, 只需要判断是否存在 $d[i][i]<0$ 。
88 */
89 #include <bits/stdc++.h>
90 using namespace std;
91
92 const int MAX_V = 107; //弗法能不能用都看的是 顶点数大不大
93 const int INF = 100007;
94
95 int d[MAX_V][MAX_V]; //表示边的权值 (不存在的时候设为INF, 无负边的时候设 $d[i][i]=0$ )
96 int V, E;
97
98 void warshall_floyd()
99 {
100     for(int k=0; k<V; k++)
101         for(int i=0; i<V; i++)
102             for(int j=0; j<V; j++)
103                  $d[i][j]=\min(d[i][j], d[i][k]+d[k][j]);$ 
104 }
105
106 int main()
107 {
108     scanf("%d%d", &V, &E);
109
110     for(int i=0; i<V; i++)
111     {
112         for(int j=0; j<V; j++)
113              $d[i][j]=INF;$ 
114          $d[i][i]=0;$ 
115     }
116
117     for(int i=0; i<E; i++)
118     {
119         int buf1, buf2, buf3;
120         scanf("%d%d%d", &buf1, &buf2, &buf3);
121          $d[buf1-1][buf2-1]=buf3;$ 
122          $d[buf2-1][buf1-1]=buf3;$ 
123     }
124
125     warshall_floyd();
126     printf("%d\n",  $d[0][V-1]$ );
127
128     return 0;
129 }
130
131
132
133
134 /*
135 最小生成树(Minimum Spanning Tree)问题(Prim Algorithm):
136 Prim和Dijkstra很类似, 都是从顶点出发不断加边的过程(直到顶点集合的数量 $X==V$ );
137 查找最小权值的边mincost[]同Dijkstra一样用优先队列维护。
138 */
139 #include <bits/stdc++.h>
140 using namespace std;
141
142 int cost[MAX_V][MAX_V];
143 int mincost[MAX_V];
144 bool used[MAX_V];
145 int V;
146
147 int prim()
148 {
149     for(int i=0; i<V; i++)
150     {
151         mincost[i]=INF;
152         used[i]=false;
153     }
154     mincost[0]=0;

```

```

155     int res=0;
156
157     while(true)
158     {
159         int v=-1;
160         //从不属于x的顶点中选取从x到其权值最小的顶点
161         for(int u=0; u<V; u++)
162         {
163             if(!used[u] && (v==-1||mincost[u]<mincost[v]))
164                 v=u;
165         }
166
167         if(v==-1)
168             break;
169         used[v]=true; //把顶点v加入x
170         res+=mincost[v]; //把边的长度加入到结果里
171
172         for(int u=0; u<V; u++)
173         {
174             mincost[u]=min(mincost[u], cost[v][u]);
175         }
176     }
177
178     return res;
179 }
180
181
182 //树的直径
183 bfs(1);
184 int Md=-1, Mn;
185 for(int i=0; i<V; i++)
186 {
187     if(Md<dis[i])
188     {
189         Md=dis[i];
190         Mn=i;
191     }
192 }
193 bfs(Mn);
194 int L=-1;
195 for(int i=0; i<V; i++)
196 {
197     if(L<dis[i])
198         L=dis[i];
199 }
200

```

```

1  ///博弈论相关
2
3  /**
4  非常重要的一种模型：
5      Alice先手，Bob后手；
6      N堆石子Xi，
7      K种取法Ai，
8      利用Grundy数转化为Nim问题，异或为零必败。
9  */
10 #include <cstdio>
11 #include <set>
12 #include <algorithm>
13 using namespace std;
14
15 const int MAX_N=1000007, MAX_K=107, MAX_X=10007;
16
17 int N, K, X[MAX_N], A[MAX_K];
18
19 //利用动态规划求grundy值的数组
20 int grundy[MAX_X];
21
22 int main()
23 {
24     //输入
25     scanf("%d%d", &N, &K);
26     for(int i=0; i<K; i++)
27         scanf("%d", &A[i]);
28     for(int i=0; i<N; i++)
29         scanf("%d", &X[i]);
30
31     //轮到自己时剩余0必败
32     grundy[0]=0;
33
34     //计算grundy值
35     int max_x=*max_element(X, X+N);
36     for(int j=1; j<=max_x; j++)
37     {
38         set<int> s;
39         for(int i=0; i<K; i++)
40             if(A[i]<=j)
41                 s.insert(grundy[j-A[i]]);
42         int g=0;
43         while(s.count(g)!=0)
44             g++;
45         grundy[j]=g;
46     }
47
48     //Nim判断胜负
49     int x=0;
50     for(int i=0; i<N; i++)
51         x^=grundy[X[i]];
52     if(x!=0)
53         puts("Alice");
54     else
55         puts("Bob");
56
57     return 0;
58 }
59

```

```

1  ///计算几何相关
2
3
4  /** 多边形的面积:
5   * 要求按照逆时针方向输入多边形顶点
6   * 可以是凸多边形或凹多边形
7   */
8  double area_of_polygon(int vcount, Lpoint plg[])
9  {
10     int i;
11     double s;
12     if (vcount < 3)
13     {
14         return 0;
15     }
16     s = plg[0].y * (plg[vcount - 1].x - plg[1].x);
17     for (i = 1; i < vcount; i++)
18     {
19         s += plg[i].y * (plg[(i - 1)].x - plg[(i + 1) % vcount].x);
20     }
21     return s / 2;
22 }
23
24
25 ///求圆相交的面积
26 double Area_of_overlap(Point c1, double r1, Point c2, double r2)
27 {
28     double d = dist(c1, c2);
29     if (r1 + r2 < d + eps)
30     {
31         return 0;
32     }
33     if (d < fabs(r1 - r2) + eps)
34     {
35         double r = min(r1, r2);
36         return PI * r * r;
37     }
38     double x = (d * d + r1 * r1 - r2 * r2) / (2 * d);
39     double t1 = acos(x / r1);
40     double t2 = acos((d - x) / r2);
41     return r1 * r1 * t1 + r2 * r2 * t2 - d * r1 * sin(t1);
42 }
43
44
45
46 /** 求矩形相交的面积
47  * x[]、y[]存储矩阵对角线顶点（只需要任意一条）
48  */
49 double Area_of_overlap_rec(double x[], double y[])
50 {
51     // 将两个矩形全部统一为主对角线
52     sort(x, x + 2);
53     sort(x + 2, x + 4);
54     sort(y, y + 2);
55     sort(y + 2, y + 4);
56
57     if (x[1] <= x[2] || x[0] >= x[3] || y[0] >= y[3] || y[1] <= y[2]) // 相离
58     {
59         return 0.0;
60     }
61     else
62     {
63         sort(x, x + 4);
64         sort(y, y + 4);
65         return (x[2] - x[1]) * (y[2] - y[1]);
66     }
67 }
68

```

```

1  ///字符串相关
2
3  ///匹配
4  char str[]="1234xyz";
5  char *str1=strstr(str,"34");
6  cout << str1 << endl;
7
8
9  /*
10  最长公共子序列问题(LCS):
11      注: 求的是子数列, 不是连续子数列; 而且只要求了长度。
12      复杂度: O(s.size() X t.size())
13  */
14  #include <bits/stdc++.h>
15  using namespace std;
16
17  string s, t;
18
19  int dp[1007][1007];
20
21  void solve()
22  {
23      for(int i=0; i<s.size(); i++)
24          for(int j=0; j<t.size(); j++)
25              if(s[i]==t[j])
26                  dp[i+1][j+1]=dp[i][j]+1;
27              else
28                  dp[i+1][j+1]=max(dp[i+1][j], dp[i][j+1]);
29
30      printf("%d\n", dp[s.size()][t.size()]);
31  }
32
33
34  int main()
35  {
36      getline(cin, s);
37      getline(cin, t);
38
39      solve();
40
41      return 0;
42  }
43
44
45  /*
46  最长上升子序列(LIS, Longest Increasing Subsequence)问题:
47      solve1(): 定义dp[i]为以ai为末尾的最长上升子数列的长度, O(n^2);
48      solve2(): 定义dp[i]为长度为i+1的上升子数列中末尾元素的最小值;
49      由于知道这种dp[]除了INF之外是单调递增的, 可以利用二分搜索得到更快的O(nlogn)。
50  */
51
52  #include <bits/stdc++.h>
53  using namespace std;
54
55  const int MAX_N=1007, INF=0x3f3f3f3f;
56
57  int n;
58  int a[MAX_N];
59
60  int dp[MAX_N];
61
62  void solve1()
63  {
64      int res=0;
65      for(int i=0; i<n; i++)
66      {
67          dp[i]=1;
68          for(int j=0; j<i; j++)
69          {
70              if(a[j]<a[i])
71                  dp[i]=max(dp[i], dp[j]+1);
72          }
73          res=max(res, dp[i]);
74      }
75      printf("%d\n", res);
76  }
77

```

```

78 void solve2()
79 {
80     fill(dp, dp+n, INF); //INF 改为 -1 变为LDS
81     for(int i=0; i<n; i++)
82     {
83         *lower_bound(dp, dp+n, a[i]) = a[i]; //, greater<int>() 变为LDS; 改为upper变成最长不下降。
84     }
85     printf("%d\n", lower_bound(dp, dp+n, INF)-dp); //, greater<int>(); INF 改为 -1 变为LDS
86 }
87
88 int main()
89 {
90     scanf("%d", &n);
91     for(int i=0; i<n; i++)
92         scanf("%d", &a[i]);
93
94     //solve1();
95     solve2();
96
97     return 0;
98 }
99
100
101 /*
102 最长公共子序列问题(LCS):
103 输出LCS
104 */
105 #include <cstdio>
106 #include <cstring>
107 #include <iostream>
108 using namespace std;
109
110 #define MAX 1001
111 #define max(x,y) ((x)>(y)?(x):(y))
112
113 char s1[MAX], s2[MAX];
114 int maxLen[MAX][MAX];
115 char ans[MAX];
116
117 int main()
118 {
119     int k = 0;
120     scanf("%s%s", s1,s2);
121     int len1 = strlen(s1);
122     int len2 = strlen(s2);
123
124     for (int i = 1; i <= len1; i++)
125     {
126         for (int j = 1; j <= len2; j++)
127         {
128             if (s1[i-1] == s2[j-1])
129                 maxLen[i][j] = maxLen[i - 1][j - 1] + 1;
130             else maxLen[i][j] = max(maxLen[i][j - 1], maxLen[i-1][j]);
131         }
132     }
133
134     int i = len1;
135     int j = len2;
136     while(i)
137     {
138         if (maxLen[i][j] > maxLen[i - 1][j])
139         {
140             if (maxLen[i][j] > maxLen[i][j - 1])
141                 ans[maxLen[i][j] - 1] = s1[i - 1];
142             else i++; //左平移
143             j--; //减小一个规模
144         }
145         i--; //上平移
146     }
147
148     printf("%s\n", ans);
149     return 0;
150 }
151
152
153 /*
154 最长回文子串:

```



```

155     Manacher算法
156 */
157 #include<iostream>
158 #include<string.h>
159 #include<algorithm>
160 using namespace std;
161
162 char s[1000];
163 char s_new[2000];
164 int p[2000];
165
166 int Init()
167 {
168     int len = strlen(s);
169     s_new[0] = '$';
170     s_new[1] = '#';
171     int j = 2;
172
173     for (int i = 0; i < len; i++)
174     {
175         s_new[j++] = s[i];
176         s_new[j++] = '#';
177     }
178
179     s_new[j] = '\0'; //别忘了哦
180
181     return j; //返回s_new的长度
182 }
183
184 int Manacher()
185 {
186     int len = Init(); //取得新字符串长度并完成向s_new的转换
187     int maxLen = -1; //最长回文长度
188
189     int id;
190     int mx = 0;
191
192     for (int i = 1; i < len; i++)
193     {
194         if (i < mx)
195             p[i] = min(p[2 * id - i], mx - i);
196         else
197             p[i] = 1;
198
199         while (s_new[i - p[i]] == s_new[i + p[i]]) //不需边界判断, 因为左有 '$', 右有 '\0'
200             p[i]++;
201
202         if (mx < i + p[i])
203             //我们每走一步i, 都要和mx比较, 我们希望mx尽可能的远, 这样才能更有机会执行if (i <
204             //mx)这句代码, 从而提高效率
205             {
206                 id = i;
207                 mx = i + p[i];
208             }
209
210         maxLen = max(maxLen, p[i] - 1);
211     }
212
213     return maxLen;
214 }
215
216 int main()
217 {
218     while (printf("请输入字符串: \n"))
219     {
220         scanf("%s", s);
221         printf("最长回文长度为 %d\n\n", Manacher());
222     }
223
224     return 0;
225 }

```