

```

1  /** GRAPH THEORY **/
2  /** GT.01 - Dijkstra Template **/
3  struct node{
4      int u,w;
5      node(int a,int b){
6          u=a;    w=b;
7      }
8  };
9  bool operator < (node A,node B){
10     return A.w>B.w;
11 }
12 void dijkstra(int s){
13     for(int i=0;i<sz;i++){
14         d[i]=intlimit;    par[i]=-1;
15     }
16     priority_queue<node>pq;
17     pq.push(node(s,0));    d[s]=0;
18     while(!pq.empty()){
19         node top=pq.top();    pq.pop();
20         int u=top.u;
21         if(top.w > d[u])    continue;
22         for(int i=0;i<graph[u].size();i++){
23             int v=graph[u][i];
24             if(top.w+cost[u][i] < d[v]){
25                 d[v]=top.w+cost[u][i];
26                 pq.push(node(v,d[v]));
27                 par[v]=u;
28             }
29         }
30     }
31 }
32 /** GT.02 - 0/1 BFS **/
33 void BFS(int start)
34 {
35     deque<int>q;
36     q.push_back(start);    dis[start] = 0;
37     while(!q.empty()){
38         int u = q.front();    q.pop_front();
39         for(int i=0 ;
40             i<(int)graph[u].size() ; i++){
41             int v = graph[u][i];
42             if(dis[v] > dis[u]+cost[u][i]){
43                 dis[v] = dis[u]+cost[u][i];
44                 (cost[u][i] == 0)?
45                     q.push_front(v) : q.push_back(v);

```

```

81 #define sz 10005
82 struct edge{
83     int u,v,cost;
84     bool operator < (const edge& p)const
85         return cost < p.cost;
86 };
87 vector<edge>e;    int par[sz];
88 void makeset(int n){
89     for(int i=1;i<=n;i++)
90         par[i]=i;
91 }
92 int Find(int x){
93     if(par[x] == x)
94         return x;
95     return par[x]=Find(par[x]);
96 }
97 int MST(int n){
98     sort(e.begin(),e.end());
99     makeset(n);
100     int cnt=0,s=0;
101     for(int i=0 ; i<e.size() ; i++){
102         int u=Find(e[i].u);
103         int v=Find(e[i].v);
104         if(u != v){
105             par[u]=v;    cnt++;
106             s += e[i].cost;
107             if(cnt == n-1)    break;
108         }
109     }
110     if(cnt == n-1)    return s;    /** All
111 the nodes are connected **/
112     else    return -1;    /** There are
113 twice or more component **/
114 }
115 /** GT.06 - SCC Template **/
116 vi graph[1005],rgraph[1005],comp[1005];
117 bool vis[1005];    stack<int>stk;
118 void DFS1(int s){
119     vis[s]=1;
120     for(int i=0;i<graph[s].size();i++){
121         int u=graph[s][i];
122         if(!vis[u])    DFS1(u);

```

```

41     }
42 }
43 }
44 }
45 /** GT.03 - TopSort **/
46 struct node{
47     int x;
48     node(int a){    x=a;    }
49 };
50 bool operator < (node A,node B){
51     return A.x>B.x;
52 }
53 vector<int>res,graph[10005];
54 int in[10005];
55 void topBFS(int n){
56     priority_queue<node>q;
57     for(int i=1;i<=n;i++){
58         if(in[i] == 0)    q.push(node(i));
59     }
60     while(!q.empty()){
61         int u=q.top().x;    res.pb(u);
62         q.pop();
63         for(int i=0;i<graph[u].size();i++){
64             in[graph[u][i]]--;
65             if(in[graph[u][i]] == 0)
66                 q.push(node(graph[u][i]));
67         }
68     }
69 }
70 /** GT.04 - TopSort **/
71 vector<int>graph[1005];
72 bool vis[1005];    stack<int>ans;
73 void topDFS(int start){
74     vis[start]=1;
75     for(int i=0 ; i<graph[start].size() ;
76         i++){
77         int u=graph[start][i];
78         if(!vis[u])    topDFS(u);
79     }
80     ans.push(start);
81 }
82 /** GT.05 - Minimum Spanning Tree **/

```

```

120     }
121     stk.push(s);    return;
122 }
123 void DFS2(int s,int x){
124     vis[s]=x;
125     comp[x].pb(s);
126     for(int i=0;i<rgraph[s].size();i++){
127         int u=rgraph[s][i];
128         if(!vis[u])    DFS2(u,x);
129     }
130     return;
131 }
132 int main()
133 {
134     int n,e,i,j,k,u,v,x;
135     while(scin2(n,e) != EOF){
136         for(i=0;i<1005;i++){
137             graph[i].clear();
138             rgraph[i].clear();
139             comp[i].clear();    vis[i]=0;
140         }
141         for(i=1;i<=e;i++){
142             scin2(u,v);
143             graph[u].pb(v);
144             rgraph[v].pb(u);
145         }
146         for(i=1;i<=n;i++){
147             if(!vis[i])    DFS1(i);
148         }
149         x=0;    ms(vis,0);
150         while(!stk.empty()){
151             int top=stk.top();    stk.pop();
152             if(!vis[top])    DFS2(top,++x);
153         }
154         for(i=1;i<=x;i++){
155             pf("%d->",i);
156             for(j=0;j<comp[i].size();j++){
157                 pf(" %d",comp[i][j]);
158             }
159             pf("\n");
160         }
161     }
162     return 0;
163 }

```

```

161  /** GT.07 - Articulation Point **/
162  #define node 10005
163  vector<int> graph[node];
164  int tim=0, par[node], disc[node], low[node];
165  bool vis[node], AP[node];
166  void DFSAP(int u) {
167      vis[u]=true;
168      low[u]=disc[u]=++tim;
169      int child=0;
170      for(int i=0; i<graph[u].size(); i++){
171          int v=graph[u][i];
172          if(vis[v] == false){
173              child++; par[v]=u;
174              DFSAP(v);
175              low[u]=min(low[u], low[v]);
176              if(par[u]!=-1 && low[v]>=disc[u]) AP[u]=1;
177              if(par[u]==-1 && child>1) AP[u]=1;
178              else if(v != par[u]) low[u]=min(low[u], disc[v]);
179          }
180      }
181  }
182  int main()
183  {
184      int n,e,u,v,i,j,cnt=0;
185      scin2(n,e);
186      for(i=1; i<=e; i++){
187          scin2(u,v); graph[u].pb(v);
188          graph[v].pb(u);
189          ms(par,-1);
190          DFSAP(1);
191          for(i=1; i<=n; i++){
192              if(AP[i]) cnt+=1;
193          }
194          cout<<cnt<<endl;
195          return 0;
196      }
197  /** GT.08 - Articulation Bridge **/
198  vl graph[100005];

```

```

226      sc("%lld %lld", &u, &m);
227      for(ll j = 0; j < m; j++){
228          scln(v); make_graph(u,v);
229      }
230  }
231  for(ll i = 0; i < n; i++)
232  if(visited[i]==0) DFS(i);
233  sort(all(rasta));
234  CASEL(t);
235  pf("%lld critical links\n", (ll)rasta.size());
236  for(ll i = 0; i < rasta.size(); i++) pf("%lld - %lld\n", rasta[i].ff, rasta[i].ss);
237  rasta.clear();
238  for(ll i = 0; i < n; i++) graph[i].clear();
239  return 0;
240  }
241  /** GT.09 - MaxFlow Template (Ford Fulkerson's Algorithm)
242  FAQ:
243  ১) একাধিক সোর্স বা সিংক থাকলে কি করতে হবে?
244  -> একটি সুপার সোর্স এবং একটি সুপার সিংক বানিয়ে নিতে হবে। সুপার সোর্স এর সাথে মূল গ্রাফের সোর্সগুলোর এজ ক্যাপাসিটি হবে ইনফিনিটি এবং সুপার সিংক এর সাথে মূল গ্রাফের সিংক গুলোর এজ ক্যাপাসিটি হবে ইনফিনিটি। ফ্লো কল দিতে হবে সুপার সোর্স থেকে সুপার সিংক এ। (উদাঃ সুপার সোর্স ০ এবং সুপার সিংক n+1)
245  ২) নোড ক্যাপাসিটি দেয়া থাকলে কি করতে হবে?
246  -> নোডটা কে দুই ভাগে ভাগ করে ফেলবো। ভাগ দুইটি কে নতুন এজ দিয়ে সংযুক্ত করবো। যেই এজের ক্যাপাসিটি হবে ওই নোডের ক্যাপাসিটির সমান। যেমনঃ n সংখ্যক নোডের একটা গ্রাফে একটা নোড যদি I হয় এবং তার ক্যাপাসিটি যদি c হয় তবে নোডটাকে ভাঙলে দুইটা নোড পাবো I এবং n+I। এবং এদের এজ ক্যাপাসিটি হবে c।
247  আমরা A নোডটা A_in এবং A_out এই দুটি নোডে ভাগ করেছি। এখন আসল গ্রাফ যতগুলো এজ A তে প্রবেশ করেছে সেগুলো প্রবেশ করবে A_in এ এবং আসল গ্রাফে যতগুলো এজ A থেকে বইয়ে গিয়েছে সেগুলো এখন বইয়ে যাবে A_out থেকে। A_in থেকে A_out এ একটা এজ প্রবেশ করবে যেটার ক্যাপাসিটি হবে এজ এর ক্যাপাসিটির সমান।
248  একটা নোডকে স্প্লিট করলে তাদের মধ্যকার এজ অবশ্যই ডিরেক্টেড হবে।
249  গ্রাফ ডিরেক্টেড হোক কিংবা আনডিরেক্টেড হোক।
250

```

```

198  ll visited[100005], started[100005], parent[100005], low[100005];
199  vector< pair<ll,ll> > rasta; /// in this vector all bridges are stored from rasta[x].ff to rasta[x].ss
200  bool mark[100005];
201  ll Time;
202  void make_graph(ll u, ll v) {
203      graph[u].pb(v);
204  }
205  void DFS(ll u, ll p = -1) {
206      if(visited[u]==1) return ;
207      visited[u] = 1; ll child = 0;
208      parent[u] = p;
209      started[u] = (++Time); low[u] = started[u];
210      for(ll i = 0; i < graph[u].size(); i++){
211          if(visited[graph[u][i]]==0) {
212              child++; DFS(graph[u][i], u);
213              low[u] = min(low[u], low[graph[u][i]]);
214          }
215          if(low[graph[u][i]]>started[u]) {mark[u] = 1; if(u<graph[u][i]) rasta.pb(mp(u, graph[u][i])); else rasta.pb(mp(graph[u][i], u));}
216          else if(parent[u]!=graph[u][i]) low[u] = min(low[u], started[graph[u][i]]);
217      }
218  }
219  int main() {
220      int t,T; scin(T);
221      RUN_CASE(t,T) {
222          ll n, m;
223          ms(visited,0); ms(started,0);
224          ms(mark,0);
225          ms(parent,0); ms(low,0); scln(n);
226          ll u, v, sum = 0;
227          for(ll i = 0; i < n ; i++){

```

```

251  ৩) দুই বন্ধু একই নোড থেকে যাত্রা শুরু করে একই গন্তব্যে পৌঁছাতে চায় কিন্তু দুইজনেই চায় ভিন্ন ভিন্ন রাস্তা ব্যবহার করে যেতে, তারমানে একই এজ কখনো ২ জন ব্যবহার করতে পারবেনা। এধরনের পথকে এজ ডিসজয়েন্ট পথ বলে। তোমাকে বলতে হবে কোনো একটা গ্রাফে দুটি এজ ডিসজয়েন্ট পথ আছে নাকি?
252  -> সাধারণ ম্যাক্স-ফ্লো ব্যবহার করেই এজ ডিসজয়েন্ট পথ বের করা যায়। শুরুর নোডকে সোর্স এবং গন্তব্য নোডকে সিংক ধরবে। এবার সবগুলো এজ এর ক্যাপাসিটি বানিয়ে দাও ১ এর সমান। এখন যদি তুমি সোর্স থেকে সিংকে দুই ফ্লো পাঠাতে পারো সেটার মানে হলো দুটি ডিসজয়েন্ট পথ আছে। প্রতিটা এজের ক্যাপাসিটি ১ হওয়াতে ২ ফ্লো যে দুটি পথে গিয়েছে তাদের মধ্যে কমন এজ থাকা সম্ভব না।
253  ঠিক একই ভাবে তুমি একটা গ্রাফে সর্বোচ্চ কয়টা ডিসজয়েন্ট পথ থাকা সম্ভব অথবা দুই বন্ধুর জায়গায় K টা বন্ধু থাকলে কি হতো বের করে ফেলতে পারবে।
254  **/
255  #define sz 105
256  bool vis[sz];
257  int n,e,cap[sz][sz],flow[sz][sz],par[sz];
258  bool BFS(int src,int des) /** Return true if there is a path from source to destination.Also, fills parent[] to store the path **/
259  {
260      ms(vis,0); queue<int>q;
261      q.push(src); vis[src] = 1; par[src] = -1;
262      while(!q.empty()){ /** Standard BFS Code using adjacency matrix **/
263          int u = q.front();
264          q.pop();
265          for(int v=1; v<=n; v++){
266              if(!vis[v] && (cap[u][v]-flow[u][v])>0){
267                  q.push(v); par[v] = u;
268                  vis[v] = 1;
269              }
270          }
271      }
272      return vis[des];
273  }
274  int FordFulkerson(int src,int des){
275      int u,v,mxFlow=0;

```

```

274     while(BFS(src,des)){
275         int mnPathFlow = infinity;
276         for(v=des ; v!=src ; v=par[v]){
277             /** Traverse the path & find out minimum
278             edge capacity **/
279             u = par[v];
280             mnPathFlow = min(mnPathFlow ,
281             cap[u][v]-flow[u][v]);
282             for(v=des ; v!=src ; v=par[v]){
283                 /** Update the flow for every edge of the
284                 path **/
285                 u = par[v];
286                 flow[u][v] += mnPathFlow;
287                 flow[v][u] -= mnPathFlow;
288             }
289             mxFlow += mnPathFlow;
290         }
291         return mxFlow;
292     }
293     int main()
294     {
295         int i,j,k,u,v,c,src,des,ans;
296         ms(cap,0); ms(flow,0);
297         cin>>n>>src>>des>>e;
298         for(i=1 ; i<=e ; i++){
299             cin>>u>>v>>c; cap[u][v] += c;
300             cap[v][u] += c; /** This line
301             means that the graph is bidirectional.If
302             the graph is unidirectional & there're no
303             edge v-u the cap[v][u]=0 **/
304         }
305         ans = FordFulkerson(src,des);
306         cout<<ans<<endl;
307         return 0;
308     }
309     /** GT.10 - MaxFlow Template using
310     Adjacency List(Ford Fulkerson's Algorithm)
311     এডজাসেন্ট লিস্ট ব্যবহার করলে গ্রাফ ইউনিডিরেকশনাল হোক কিংবা
312     বাইডিরেকশনাল, এজ দুইদিকেই যুক্ত করতে হবে। কারণ, ম্যাক্স ফ্লো তে ব্যাক
313     এজ ব্যবহার করে অপটিমাল সল্যুশন বের করতে হয়। ইউনিডিরেকশনাল হলে
314     ক্যাপাসিটি এরে একমুখী হবে। তার মানে গ্রাফ একমুখী। এজন্য, লিস্টে দুইদিকে

```

```

330 }
331 int FordFulkerson(int src,int des){
332     int u,v,mxFlow=0;
333     while(BFS(src,des)){
334         int mnPathFlow = infinity;
335         for(v=des ; v!=src ; v=par[v]){
336             /** Traverse the path & find out minimum
337             edge capacity **/
338             u = par[v];
339             mnPathFlow = min(mnPathFlow ,
340             cap[u][v]-flow[u][v]);
341             for(v=des ; v!=src ; v=par[v]){
342                 /** Update the flow for every edge of the
343                 path **/
344                 u = par[v];
345                 flow[u][v] += mnPathFlow;
346                 flow[v][u] -= mnPathFlow;
347             }
348             mxFlow += mnPathFlow;
349         }
350         return mxFlow;
351     }
352     void addEdge(int u,int v){
353         graph[u].pb(v);
354         graph[v].pb(u);
355     }
356     int main()
357     {
358         int i,j,k,u,v,c,src,des,ans;
359         scin2(n,e); scin2(src,des);
360         for(i=1 ; i<=e ; i++){
361             cin>>u>>v>>c; cap[u][v] += c;
362             cap[v][u] += c; /** This line
363             means that the graph is bidirectional.If
364             the graph is unidirectional & there're no
365             edge v-u the cap[v][u]=0 **/
366             addEdge(u,v);
367         }
368         ans = FordFulkerson(src,des);
369         cout<<ans<<endl;
370         return 0;

```

```

304     এজ যুক্ত করলেও সমস্যা হবে না। বরং দুইদিকে যুক্ত না করলে কাজ করবে না।
305     Theory of BPM:
306     1. MaxBPM = MaxFlow
307     2. Max Size of Independent Set = Total
308     Node - Max BPM(here, edge of a graph is
309     the dependency)
310     3. Vertex Cover = Max BPM ( Vertex
311     Cover: You have to select minimum number
312     of vertex so that each & every edges r
313     connected to these vertex)
314     4. Edge cover = n - Max BPM ( Edge
315     cover: You have to select minimum number
316     of edges so that they cover all the vertex)
317     **/
318     #define sz 205
319     bool vis[sz];
320     int n,e,par[sz],cap[sz][sz],flow[sz][sz];
321     vector<int>graph[sz];
322     bool BFS(int src,int des) /** Return
323     true if there is a path from source to
324     destination.Also,fills parent[] to store
325     the path **/
326     {
327         ms(vis , 0); queue<int>q;
328         q.push(src); vis[src] = 1; par[src]
329         = -1;
330         while(!q.empty()){ /** Standard BFS
331         using adjacency list **/
332         int u = q.front();
333         q.pop();
334         for(int i=0 ;
335         i<(int)graph[u].size() ; i++){
336             int v = graph[u][i];
337             if(!vis[v] &&
338             (cap[u][v]-flow[u][v])>0){
339                 q.push(v);
340                 par[v] = u;
341                 vis[v] = 1;
342             }
343         }
344         return vis[des];

```

```

363 }
364 /** GT.11 - BPM using Kuhn's Algorithm ;
365 Complexity - O(VE) **/
366 int Left[1005],Right[1005],vis[1005];
367 vi graph[1005];
368 bool Kuhn(int u){
369     for(int i=0 ; i<(int)graph[u].size()
370     ; i++){
371         int v = graph[u][i];
372         if(vis[v])continue;
373         vis[v] = 1;
374         if(Right[v]==-1 || Kuhn(Right[v])){
375             Right[v] = u; Left[u] = v;
376             return true; ///Matching
377         }
378     }
379     return false; ///Matching not possible
380 }
381 int BPM(int setA){
382     ms(Left , -1); ms(Right , -1); int
383     m=setA; int cnt = 0;
384     for(int i=1 ; i<=m ; i++){ ///
385     m-total node of left part(set A) of
386     bipartite graph ; We can calculate the
387     number of node of set A by bi-coloring the
388     graph with two color white & black OR we
389     can put total number of node of the given
390     graph here
391     ms(vis , 0);
392     if(Kuhn(i)) cnt++;
393     }
394     return cnt;
395 }
396 int main()
397 {
398     int i,j,k,u,v,n,e,bpm;
399     scin2(n , e);
400     for(i=1 ; i<=e ; i++){
401         scin2(u , v); graph[u].pb(v);
402         graph[v].pb(u);
403     }

```

```

394     bpm = BPM(5);
395     cout<<bpm<<endl;
396     return 0;
397 }
398 /** DS.01 - Order Set Template */
399 #include <ext/pb_ds/assoc_container.hpp>
400 using namespace __gnu_pbds;
401 template<typename T> using orderset =
tree<T,null_type,less<T>,rb_tree_tag,tree_order_statistics_node_update>;
402 orderset<int> X ; orderset<int>::iterator
it ;
403 X.insert(1);
404 cout<<X.find_by_order(1)<<endl;
405 cout<<X.order_of_key(-5)<<endl;
406 /** DS.02 - DSU Template By Component Size
**/
407 #define sz 100005
408 int par[sz],Size[sz];
409 void makeset(int n){
410     for(int i=1 ; i<=n ; i++){
411         par[i] = i; Size[i] = 1;
412     }
413 }
414 int FindSet(int node){
415     if(par[node] == node) return node;
416     return par[node] = FindSet(par[node]);
417 }
418 void UnionSet(int nodeA,int nodeB){
419     int a = FindSet(nodeA);
420     int b = FindSet(nodeB);
421     if(a != b){
422         if(Size[a] < Size[b]) swap(a,b);
423         par[b] = a;
424         Size[a] += Size[b];
425     }
426 }
427 int main()
428 {
429     int i,j,k,n,e,u,v;
430     scin2(n,e); makeset(n);
431     for(i=1 ; i<=e ; i++){

```

```

469         for(int i=0 ; i<q ; i++){
470             while(r < query[i].r) Add(++r);
471             while(l < query[i].l)
Remove(l++);
472             while(r > query[i].r)
Remove(r--);
473             while(l > query[i].l) Add(--l);
474             ans[query[i].id] = sum;
475         }
476         for(int i=0 ; i<q ; i++)
477             cout<<ans[i]<<endl;
478         return 0;
479     }
480 /** DS.04 - Square Root Decomposition
Template(Point update ; Sum query) */
481 int n,numBlock,blockSize;
482 /** size of the blocks and the number of
blocks r equal*/
483 int ara[100005],block[320];
484 void Build(){
485     for(int i=0 ; i<n ; i++)
486         block[i/blockSize] += ara[i];
487 }
488 int Query(int lft,int rgt){
489     int
sum=0,sourceBlock,destBlock,nxtBlock,prevBlock;
490     sourceBlock = lft/blockSize;
491     destBlock = rgt/blockSize;
492     if(sourceBlock == destBlock) /** IF
left and right r existing in same block */
493     {
494         for(int i=lft ; i<=rgt ; i++)
495             sum += ara[i];
496     }
497     else{
498         nxtBlock = sourceBlock+1;
499         prevBlock = destBlock-1;
500         for(int i=lft ;
i<=nxtBlock*blockSize-1 ; i++)
501             sum += ara[i]; /** Add
1st partially relevant block */

```

```

432         scin2(u,v); UnionSet(u,v);
433     }
434     for(i=1 ; i<=n ; i++)
435         pf("\tParent of %d = %d\tSize of
this component =
%d\n",i,FindSet(i),Size[FindSet(i)]);
436     return 0;
437 }
438 /** DS.03 - MO's Algorithm */
439 #define sz 200005
440 int blocksz,ans[sz],ara[sz],sum;
441 struct info{
442     int id,l,r;
443     bool operator < (const info &p)const{
444         int blockno = l/blocksz , pblockno
= p.l/blocksz;
445         if(blockno == pblockno) return
r<p.r;
446         else return blockno<pblockno;
447     }
448 }query[sz];
449 void Add(int index){
450     sum += ara[index];
451 }
452 void Remove(int index){
453     sum -= ara[index];
454 }
455 int main()
456 {
457     int n,q;
458     scin(n);
459     blocksz = sqrt(n);
460     for(int i=0 ; i<n ; i++)scin(ara[i]);
461     scin(q);
462     for(int i=0 ; i<q ; i++){
463         scin2(query[i].l , query[i].r);
464         query[i].id = i;
465     }
466     sort(query , query+q);
467     int l=0,r=-1;
468     /** Calculate answer for all query in
OFFline */

```

```

502         for(int i=nxtBlock ; i<=prevBlock
; i++)
503             sum += block[i]; /** Add
all fully relevant block */
504         for(int i=destBlock*blockSize ;
i<=rgt ; i++)
505             sum += ara[i]; /** Add
last partially relevant block */
506     }
507     return sum;
508 }
509 void update(int id,int val){
510     int blockID = id/blockSize;
block[blockID] += (val - ara[id]);
ara[id] = val;
511 }
512 int main()
513 {
514     int i,j,k,temp,l,r,q,ans,type,id,val;
515     scin(n);
516     for(i=0 ; i<n ; i++)
517         scin(ara[i]);
518     numBlock = (int)sqrt(n*1.0)+1;
519     blockSize = numBlock;
520     Build();
521     scin(q);
522     for(i=1 ; i<=q ; i++){
523         scin(type);
524         if(type == 0){
525             scin2(id,val); update(id,val);
526         }
527         else{
528             scin2(l,r); ans = Query(l,r);
529             cout<<ans<<endl;
530         }
531     }
532     return 0;
533 }
534 /** DS.05 - Standard Segment Tree(Lazy) */
535 void push_down(ll node,ll b,ll e){
536     tree[node] += lazy[node];
537     if(b != e){

```

```

538         lazy[node*2] += lazy[node];
539         lazy[node*2+1] += lazy[node];
540     }
541     lazy[node] = 0;
542 }
543 void Build(ll node,ll b,ll e){    /** Build
Tree for Range Minimum Query **/
544     if(b == e){
545         tree[node] = ara[b]; return;
546     }
547     ll mid = (b+e)/2;
548     Build(node<<1 , b , mid);
549     Build(1+(node<<1) , mid+1 , e);
550     tree[node] = min(tree[node<<1] ,
tree[1+(node<<1)]);
551 }
552 void Update(ll node,ll b,ll e,ll i,ll
j,ll val)    /** Increase all the element
in a range **/
553 {
554     if(lazy[node] != 0) push_down(node ,
b , e);
555     if(i>e || j<b) return;
556     if(b>=i && e<=j){
557         tree[node] += val;
558         if(b != e){
559             lazy[node*2] += val;
560             lazy[node*2+1] += val;
561         }
562         return;
563     }
564     ll mid = (b+e)/2;
565     Update(node<<1 , b , mid , i , j ,
val);
566     Update(1+(node<<1) , mid+1 , e , i ,
j , val);
567     tree[node] = min(tree[lft] ,
tree[rgt]);
568 }
569 ll Query(ll node,ll b,ll e,ll i,ll j)
/** Range Minimum Query **/
570 {

```

```

601     pii p1 = query(node<<1 , b , mid , i
, j);
602     pii p2 = query(right , mid+1 , e , i
, j);
603     return combine(p1 , p2);
604 }
605 void update(int node,int b,int e,int
pos,int val){
606     if(pos>e || pos<b) return;
607     if(b>=pos && e<=pos){
608         tree[node] = make_pair(val ,
1); return;
609     }
610     int mid = (b+e)/2;
611     update(node<<1 , b , mid , pos , val);
612     update(1+(node<<1) , mid+1 , e , pos
, val);
613     tree[node] = combine(tree[left] ,
tree[right]);
614 }
615 int main()
616 {
617     int i,j,n,lb,ub,pos,val,q,cmd;
618     pii ans;
619     scin2(n,q);
620     for(i=1;i<=n;i++)    scin(ara[i]);
621     init(1 , 1 , n);
622     for(i=1 ; i<=q ; i++){
623         scin(cmd);        /** 0-update;
1-query **/
624         if(cmd == 0){
625             scin2(pos , val);
626             update(1 , 1 , n , pos , val);
627         }
628         else{
629             scin2(lb , ub);
630             ans = query(1 , 1 , n , lb ,
ub);
631             pf("Maximum
value=%d\tFrequency=%d\n" , ans.first ,
ans.second);
632         }

```

```

571         if(i>e || j<b) return longlimit;
572         if(lazy[node] != 0) push_down(node ,
b , e);
573         if(b>=i && e<=j)    return tree[node];
574         ll mid = (b+e)/2;
575         ll p1 = Query(node<<1 , b , mid , i ,
j);
576         ll p2 = Query(1+(node<<1) , mid+1 , e
, i , j);
577         return min(p1 , p2);
578     }
579     /** DS.06 - Find the maximum value &
number of times it appears **/
580     int ara[sz];    pii tree[sz*4];
581     pii combine(pii a,pii b){
582         if(a.first > b.first)
583             return
/**
First-value , Second-frequency**/
584         if(b.first > a.first)
585             return b;
586         return make_pair(a.first ,
a.second+b.second);    /** When value is
same,then sum the frequency**/
587     }
588     void init(int node,int b,int e){
589         if(b == e){
590             tree[node] = make_pair(ara[b] ,
1); return;
591         }
592         int mid = (b+e)/2;
593         init(node<<1 , b , mid);
594         init(1+(node<<1) , mid+1 , e);
595         tree[node] = combine(tree[left] ,
tree[right]);
596     }
597     pii query(int node,int b,int e,int i,int
j){
598         if(i>e || j<b) return
make_pair(-infinity , 0);
599         if(b>=i && e<=j)    return tree[node];
600         int mid = (b+e)/2;
601
602
603         }
604         return 0;
605     }
606     /** DS.07 - Maximum Sub Array Sum with
point update **/
607     struct data
608     {
609         ll sum,pref,suff,ans;
610     }tree[4*sz];
611     int ara[sz];
612     data combine(data left,data right){
613         data res;
614         res.sum=left.sum+right.sum;
615
616         res.pref=max(left.pref,left.sum+right.pref);
617
618         res.suff=max(right.suff,right.sum+left.suff);
619
620         res.ans=max(max(left.ans,right.ans),left.su
ff+right.pref);
621         return res;
622     }
623     data make_data(int val){
624         data res;
625         res.sum=val;    res.pref=val;
626         res.suff=val;    res.ans=val;
627         /** You must take minimum one
element, thats why res.pref=val;res.suff=val;
628         res.ans=val; otherwise,
it will
629         res.pref=max(0,val);res.suff=max(0,val);res.
ans=max(0,val);**/
630         return res;
631     }
632     void init(int node,int b,int e){
633         if(b == e){
634             tree[node]=make_data(ara[b]);
635         }
636         return;
637     }
638     int mid=(b+e)/2;

```

```

664     init(node<<1 , b , mid);
665     init(1+(node<<1) , mid+1 , e);
666     tree[node] = combine(tree[node<<1] ,
667 }
668 void update(int node,int b,int e,int
idx,int val){
669     if(idx>e || idx<b) return;
670     if(b>=idx && e<=idx){
671         tree[node]=make_data(val); return;
672     }
673     int mid=(b+e)/2;
674     update(node<<1 , b , mid , idx , val);
675     update(1+(node<<1) , mid+1 , e , idx
, val);
676     tree[node] = combine(tree[node<<1] ,
tree[1+(node<<1)]);
677 }
678 data query(int node,int b,int e,int i,int
j){
679     if(i>e || j<b) return
make_data(-infinity);
680     if(b>=i && e<=j) return tree[node];
681     int mid=(b+e)/2;
682     data p1=query(node , b , mid , i , j);
683     data p2=query(1+(node<<1) , mid+1 , e
, i , j);
684     return combine(p1,p2);
685 }
686 int main()
687 {
688     int i,j,x,y,n,fg,q,val;
689     data res;
690     scin(n);
691     for(i=1;i<=n;i++) scin(ara[i]);
692     init(1,1,n); scin(q);
693     for(i=1;i<=q;i++){
694         scin(fg);
695         if(fg == 0){
696             scin2(x,val);
697         }

```

```

i , j);
727     pii ret2 = query(1+(node<<1) , mid+1
, e , i , j);
728     pii ret; ret.first =
min(ret1.first , ret2.first);
729     ret.second =
(ret.first==ret1.first)?ret1.second:ret2.sec
ond;
730     return ret;
731 }
732 int Calculate(int b,int e) /** BS for
calculating the area of simple histogram */
733 {
734     if(b > e)
735         return 0;
736     if(b == e)
737         return ara[b];
738     int ret1,ret2,ret3;
739     pii p = query(1,1,n,b,e);
740     ret1 = p.first*(e-b+1);
741     ret2 = Calculate(b , p.second-1);
742     ret3 = Calculate(p.second+1 , e);
743     return max(ret1 , max(ret2 , ret3));
744 }
745 int main()
746 {
747     int i,j,k,t,T,ans,cnt;
748     scin(T);
749     RUN_CASE(t,T){
750         ans=0; ms(ara,0); scin2(m,n);
751         for(i=1 ; i<=m ; i++)
752         {
753             sc("%s",s[i]);
754             for(j=0 ; j<n ; j++)
755                 ara[j+1] += 1;
756             /** Here,0 means space , 1 means block ;
so, we r trying to create histogram */
757             {

```

```

698         else{
699             scin2(x,y);
700             res=query(1,1,n,x,y);
701             pf("%lld\n",res.ans);
702         }
703         return 0;
704     }
705     /** DS.08 - Maximum SubRectangle Area
using segmentTree + BS ; We can calculate
maximum rectangle area using init,query &
calculate function*/
706     #define sz 2005
707     char s[sz][sz];
708     int n,m,ara[sz];
709     pii tree[3*sz]; /** 1st element-min
value of range ; 2nd element-index of min
value */
710     void init(int node,int b,int e){
711         if(b == e){
712             tree[node] = make_pair(ara[b] ,
b); return;
713         }
714         int mid=(b+e)/2;
715         init(node<<1 , b , mid);
716         init(1+(node<<1) , mid+1 , e);
717         tree[node].first =
min(tree[node<<1].first ,
tree[1+(node<<1)].first);
718         tree[node].second =
(tree[node<<1].first ==
tree[node].first)?tree[node<<1].second:tree[
1+(node<<1)].second;
719     }
720     pii query(int node,int b,int e,int i,int
j){
721         if(b>e || b>j || e<i)
722             return make_pair(infinity,infinity);
723         if(b>=i && e<=j)
724             return tree[node];
725         int mid=(b+e)/2;
726         pii ret1 = query(node<<1 , b , mid ,

```

```

758         else
759             ara[j+1] = 0;
760         /** Here,consecutive 0 from this row is 0
**/
761         init(1,1,n); /** Building
histogram from row 1 to i */
762         cnt = Calculate(1,n); /**
Maximum sub rectangle area of this
histogram */
763         ans = max(ans,cnt);
764     }
765     pf("Case %d: %d\n",t,ans);
766 }
767     return 0;
768 }
769 /** DS.09 - Segment Tree + Offline Query
How many horizontal lines are completely
lies in segment l to r?
771 You're given an array with n distinct
elements & q queries. In each query you'll
given l[i],r[i]. You have to print the
number of pairs of integers
q,w(l[i]<=q,w<=r[i]) such that ara[q] is
the divisor of ara[w].
772 Solution:
773 Suppose, we have a pair of index(i,j)
where ara[i] is the divisor of ara[j]. We
say it a segment on a 2D plane. Now we
have to count the number of segment those
are completely lies in range [x,y] on that
2D plane.
774 1. For each number between [1,n] we have
to store its multiples between [1,n].
Actually here in divisor's id, we stored
all its multiple's id.
775 2. Store the query according to l[i] of
each query. (see the solution for details)
776 3. Now, iterate from n to 1. For each i,
increase the value of endpoint of the
segment starting from i. Calculate the sum
of total segment lies in range i to

```



```

777 qry[i].first **/
778 int ara[sz],id[sz],tree[4*sz],ans[sz];
779 void Update(int node,int b,int e,int
780 pos,int val){
781     if(pos>e || pos<b) return;
782     if(b>=pos && e<=pos){
783         tree[node] += val; return;
784     }
785     int mid = (b+e)>>1;
786     if(pos <= mid)
787         Update(node<<1 , b , mid , pos ,
788 val);
789     else
790         Update(1+(node<<1) , mid+1 , e ,
791 pos , val);
792     tree[node] =
793     tree[node<<1]+tree[1+(node<<1)];
794 }
795 int Query(int node,int b,int e,int i,int
796 j){
797     if(i>e || j<b) return 0;
798     if(b>=i && e<=j) return tree[node];
799     int mid = (b+e)>>1;
800     return (Query(node<<1 , b , mid , i ,
801 j) + Query(1+(node<<1) , mid+1 , e , i ,
802 j));
803 }
804 vi segment[sz]; /** segment[i][j]
805 means i is the divisor's id & j is the
806 multiple's id **/
807 vector<pii>qry[sz]; /** qry[i][j] means
808 l=i && qry[i][j]={r , query's SL} **/
809 int main()
810 {
811     int
812     i,j,k,n,q,l,r,lft,rgt,x,y,totseg,cnt=0,bads
813     z=0;
814     scin2(n,q);
815     for(i=1 ; i<=n ; i++){
816         scin(ara[i]); id[ara[i]] = i;
817     }
818     /**ID of each distinct element
819     */

```

```

835 3. Then call query from each position to
836 calculate the number of segment started
837 on that position and have sum less
838 than val. **/
839 #define sz 200005
840 ll ara[sz],sum[sz];
841 vector<ll>tree[4*sz];
842 void Init(ll node,ll b,ll e){
843     if(b == e){
844         tree[node].pb(sum[b]); return;
845     }
846     ll mid = (b+e)>>1;
847     Init(node<<1 , b , mid);
848     Init(1+(node<<1) , mid+1 , e);
849     merge(tree[node<<1].begin() ,
850 tree[node<<1].end() ,
851 tree[1+(node<<1)].begin() ,
852 tree[1+(node<<1)].end() ,
853 back_inserter(tree[node]));
854 }
855 ll Query(ll node,ll b,ll e,ll i,ll j,ll
856 val){
857     if(i>e || j<b) return 0;
858     if(b>=i && e<=j)
859         return
860         upper_bound(tree[node].begin() ,
861 tree[node].end() ,
862 val-1)-tree[node].begin();
863     ll mid = (b+e)>>1;
864     ll p1 = Query(node<<1 , b , mid , i ,
865 j , val);
866     ll p2 = Query(1+(node<<1) , mid+1 , e
867 , i , j , val);
868     return p1+p2;
869 }
870 int main()
871 {
872     ll i,j,k,n,val,ans=0,x;
873     scin2(n , val);
874     rep(i,1,n)scin(ara[i]);
875     rep(i,1,n)sum[i] = sum[i-1]+ara[i];
876 }

```

```

805     for(i=1 ; i<=q ; i++){
806         scin2(lft,rgt);
807         qry[lft].pb(make_pair(rgt , i));
808     }
809     for(i=1 ; i<=n ; i++){
810         for(j=i ; j<=n ; j+=i){
811             x = id[i]; /**Divisor's ID
812             y = id[j]; /**Multiple's ID
813             segment[min(x,y)].pb(max(x,y));
814         }
815     }
816     /** How OFFline Query works
817     for(i=n ; i>=1 ; i--){
818         for(auto it : segment[i])
819             Update(1 , 1 , n , it ,
820 1); /**Increase the value of ending
821 point of the segment starting from this
822 node. Here, i is the divisor's address ,
823 it is the multiple's address.
824         for(auto it : qry[i])
825             ans[it.second] = Query(1 , 1
826 , n , i , it.first); /**Calculate the
827 sum of total segment completely lies in
828 range i to it.first
829     }
830     for(i=1 ; i<=q ; i++){
831         pf("%d\n",ans[i]);
832     }
833     return 0;
834 }
835 /** DS.10 - Merge Sort Tree Template
836 Problem:
837 You're given an array a with n integers.
838 You've to calculate the total number
839 of subArrays with sum less than val.
840 More formally, you are required to calculate
841 the number of pairs l,r (l<=r) such that
842 al+al+1+...+ar-1+ar<val.
843 Solution:
844 1. Calculate the cumulative sum of the
845 given array.
846 2. Build a merge sort tree on cumulative
847 sum array.

```

```

865 Init(1 , 1 , n);
866 for(i=1 ; i<=n ; i++){
867     x = Query(1 , 1 , n , i , n ,
868 sum[i-1]+val);
869     ans += x;
870 }
871 cout<<ans<<endl;
872 return 0;
873 }
874 /** DS.11 - 2D Segment Tree template
875 Sum query of a sub rectangle of the
876 matrices and update an element of given
877 matrices **/
878 int n,m,mat[sz][sz],tree[4*sz][4*sz];
879 void initY(int nodeX,int bX,int eX,int
880 nodeY,int bY,int eY){
881     if(bY == eY){
882         if(bX == eX) /** We r working
883 on a single row ; we can directly generate
884 segTree for row = bX **/
885         tree[nodeX][nodeY] =
886         mat[bX][bY];
887     }
888     else /** We r working
889 on multiple row ; We r creating parent
890 segTree from two child segTree which
891 contains sum from row(bX to eX) **/
892     tree[nodeX][nodeY] =
893     tree[nodeX*2][nodeY] +
894     tree[nodeX*2+1][nodeY];
895     return;
896 }
897 int midY = (bY+eY)/2;
898 initY(nodeX , bX , eX , 2*nodeY , bY
899 , midY);
900 initY(nodeX , bX , eX , 2*nodeY+1 ,
901 midY+1 , eY);
902 tree[nodeX][nodeY] =
903     tree[nodeX][nodeY*2] +
904     tree[nodeX][nodeY*2+1];
905 }
906 void initX(int nodeX,int bX,int eX){
907     if(bX == eX){

```

```

891     initY(nodeX, bX, eX, 1, 1, m);
892     return;
893 }
894 int midX=(bX+eX)/2;
895 initX(2*nodeX, bX, midX);
896 initX(2*nodeX+1, midX+1, eX);
897 initY(nodeX, bX, eX, 1, 1, m);
898 /** Merge segment trees of two rows to
899 create parent segment tree */
900 int sumQueryY(int nodeX, int nodeY, int
901 bY, int eY, int ly, int ry){
902     if(ly > ry) return 0;
903     if(ly==bY && ry==eY) return
904 tree[nodeX][nodeY];
905     int midY = (bY+eY)/2;
906     int ret1 = sumQueryY(nodeX, nodeY*2
907 , bY, midY, ly, min(ry, midY));
908     int ret2 = sumQueryY(nodeX,
909 nodeY*2+1, midY+1, eY, max(ly, midY+1)
910 , ry);
911     return ret1+ret2;
912 }
913 int sumQueryX(int nodeX, int bX, int eX, int
914 lx, int rx, int ly, int ry){
915     if(lx > rx) return 0;
916     if(lx==bX && rx==eX) return
917 sumQueryY(nodeX, 1, 1, m, ly, ry);
918     int midX = (bX+eX)/2;
919     int ret1 = sumQueryX(nodeX*2, bX,
920 midX, lx, min(rx, midX), ly, ry);
921     int ret2 = sumQueryX(nodeX*2+1,
922 midX+1, eX, max(lx, midX+1), rx, ly,
923 ry);
924     return ret1+ret2;
925 }
926 void updateY(int nodeX, int bX, int eX, int
927 nodeY, int bY, int eY, int x, int y, int val){
928     if(bY == eY){
929         if(bX == eX)
930             tree[nodeX][nodeY] = val;
931         else

```

```

951     for(i=1 ; i<=q ; i++){
952         scin(id);
953         if(id == 0){ ///Update
954             scin2(x1,y1); scin(val);
955             updateX(1, 1, n, x1, y1,
956 val);
957         }
958         else{ ///Sum Query
959             scin2(x1,y1); scin2(x2,y2);
960             ans = sumQueryX(1, 1, n,
961 x1, x2, y1, y2);
962             cout<<ans<<endl;
963         }
964     }
965     return 0;
966 }
967 /** DS.12 - Trie Template
968 ট্রাই এর কিছু ব্যবহার:
969 ১. একটা ডিকশনারিতে অনেকগুলো শব্দ আছে, কোনো একটা শব্দ আছে
970 নাকি নাই খুজে বের করতে হবে। এই প্রবলেমটা আমার উপরের কোডেই সলভ
971 করেছি।
972 ২. ধরো তোমার ৩ বন্ধুর টেলিফোন নম্বর হলো "৫৬৭৮", "৯০৩২২",
973 "৫৬৭"। তুমি যখন প্রথম বন্ধুকে ডায়াল করবে তখন ৫৬৭ চাপার সাথে সাথে
974 ৩য় বন্ধুর কাছে ফোন চলে যাবে কারণ ৩য় বন্ধুর নামের প্রথম জনের প্রিফিক্স!
975 অনেকগুলো ফোন নম্বর দেয়া আছে, বলতে হবে এরকম কোনো নামের আছে
976 নাকি যেটা অন্য নামের প্রিফিক্স! (UVA 11362)।
977 ৩. একটা ডিকশনারিতে অনেকগুলো শব্দ আছে। এখন কোনো একটা শব্দ
978 কয়বার "prefix" হিসাবে এসেছে সেটা বের করতে হবে। যেমন "al"
979 শব্দটা উপরের ডিকশনারিতে ৩বার প্রিফিক্স হিসাবে এসেছে (alga,
980 algea, also এই সবগুলো শব্দের প্রিফিক্স "al")। এটা বের করার
981 জন্য প্রতিটা নোডে একটা কাউন্টার ভ্যারিয়েবল রাখতে হবে, কোনো নোডে
982 যতবার যাবে ততবার কাউন্টারের মান বাড়িয়ে দিবে। সার্চ করার সময় প্রিফিক্সটা
983 খুজে বের করে কাউন্টারের মান দেখবে।
984 ৪. মোবাইলের ফোনবুকে সার্চ করার সময় তুমি যখন কয়েকটা লেটার লিখো
985 তখন সেই প্রিফিক্স দিয়ে কি কি নাম শুরু হয়েছে সেগুলো সাজেশন বক্সে
986 দেখায়। এটা তুমি ট্রাই দিয়ে ইমপ্লিমেন্ট করতে পারবে?
987 ৫. দুটি স্ট্রিং এর "longest common substring" বের করতে
988 হবে। (subsequence হলে ডিপি দিয়ে সহজে করা যায়, এখানে
989 substring চেয়েছি)।
990 (হিস্টস: একটা স্ট্রিং এর শেষ থেকে এক বা একাধিক ক্যারেকটার নেয়া হলে
991 সেটাকে স্ট্রিংটার সাফিক্স বলে, যেমন blog এর সাফিক্স
992 g, og, log, blog আর প্রতিটা substring ই কিন্তু কোনোনা কোনো

```

```

920     tree[nodeX][nodeY] =
921 tree[nodeX*2][nodeY] +
922 tree[nodeX*2+1][nodeY];
923     return;
924 }
925 int midY = (bY+eY)/2;
926 if(y <= midY)
927     updateY(nodeX, bX, eX, nodeY*2
928 , bY, midY, x, y, val);
929 else
930     updateY(nodeX, bX, eX,
931 nodeY*2+1, midY+1, eY, x, y, val);
932     tree[nodeX][nodeY] =
933 tree[nodeX][nodeY*2]+tree[nodeX][nodeY*2+1];
934     ///Merge two columns
935 }
936 void updateX(int nodeX, int bX, int eX, int
937 x, int y, int val){
938     if(bX == eX){
939         updateY(nodeX, bX, eX, 1, 1,
940 m, x, y, val);
941         return;
942     }
943     int midX = (bX+eX)/2;
944     if(x <= midX)
945         updateX(nodeX*2, bX, midX, x,
946 y, val);
947     else
948         updateX(nodeX*2+1, midX+1, eX,
949 x, y, val);
950     updateY(nodeX, bX, eX, 1, 1, m,
951 x, y, val);    ///Merge two rows
952 }
953 int main()
954 {
955     int i,j,k,q,x1,y1,x2,y2,ans,val,id;
956     scin2(n,m);
957     for(i=1 ; i<=n ; i++){
958         for(j=1 ; j<=m ; j++){
959             scin(mat[i][j]);
960         }
961     }
962     initX(1,1,n); scin(q);

```

সাফিক্স এর প্রিফিক্স!! তাই সবগুলো সাফিক্সকে ট্রাইতে ইনসার্ট করলে কাজটা সহজ হয়ে যায়।)

```

973 ৫. (অ্যাডভান্সড) সম্ভবত ২০১১তে ডেফোডিল ইউনিভার্সিটির ন্যাশনাল
974 কনটেস্টে এসেছিলো প্রবলেমটা। একটা ডিকশনারি ইনপুট দেয়া থাকবে।
975 প্রতিবার ডিকশনারির ২টা শব্দ কুয়েরি দিবে, বলতে হবে তাদের মধ্যে
976 common prefix এর দৈর্ঘ্য কত। যেমন algo আর algea এর কমন
977 প্রিফিক্স alg, দৈর্ঘ্য ৩। ট্রাইতে ডিকশনারিতে ইনসার্ট করে প্রতি কুয়েরিতে
978 শব্দদুটি এন্ড-মার্ক থেকে LCA (lowest common ancestor) বের
979 করে প্রবলেমটা সলভ করা যায়।
980 **/
981 struct node{
982     bool endmark;
983     node *next[26+1];
984     node(){
985         endmark = false;
986         for(int i=0 ; i<26 ; i++)
987             next[i] = NULL;
988     }
989 };
990 node *root;
991 void Insert(char* str, int len){
992     node* curr = root;
993     for(int i=0 ; i<len ; i++){
994         int id = str[i]-'a';
995         if(curr->next[id] == NULL)
996             curr->next[id] = new node();
997         curr = curr->next[id];
998     }
999     curr->endmark = true;
1000 }
1001 bool Search(char *str,int len){
1002     node* curr = root;
1003     for(int i=0 ; i<len ; i++){
1004         int id = str[i]-'a';
1005         if(curr->next[id] == NULL)
1006             return false;
1007         curr = curr->next[id];
1008     }
1009     return curr->endmark;
1010 }
1011 void Destroy(node *curr)    /** If we pass
1012 root then the whole tree will be deleted */

```



```

1006 {
1007     for(int i=0 ; i<26 ; i++){
1008         if(curr->next[i] != NULL)
1009             Destroy(curr->next[i]);
1010     }
1011     delete(curr);
1012 }
1013 int main()
1014 {
1015     int i,j,k,n,q;
1016     root = new node();
1017     pf("Enter the number of word: ");
1018     scin(n);
1019     for(i=0 ; i<n ; i++){
1020         char str[50];
1021         sc("%s",str);
1022         Insert(str , strlen(str));
1023     }
1024     pf("Enter the number of query: ");
1025     scin(q);
1026     for(i=1 ; i<=q ; i++){
1027         char str[50];
1028         sc("%s",str);
1029         if(Search(str , strlen(str)))
1030             pf("The string u searched for
1031 is found\n");
1032         else
1033             pf("The string u searched for
1034 is not found\n");
1035     }
1036     return 0;
1037 }
1038 /** DS.13 - LCA template using sparse
1039 table */
1040 /** complexity of build: O(nlogn) ;
1041 complexity of query: O(logn) */
1042 #define sz 100005
1043 bool vis[sz];
1044 int level[sz],parent[sz],st[sz][22];
1045 vi graph[sz];
1046 void BFS(int start) /** Pre-calculate
1047 the parent & level of each node */

```

```

1072 int LCA_Query(int n,int p,int q) /**
1073 Here, n is the number of node. p & q are
1074 two nodes whose LCA we've to found out */
1075 {
1076     if(level[p] < level[q])
1077         swap(p,q); /** p
1078 will always be more deeper node than q */
1079     int Log=1;
1080     while(1){
1081         int nxt = Log+1;
1082         if((1<<nxt) > level[p])
1083             break;
1084         Log++;
1085     }
1086     for(int i=Log ; i>=0 ; i--){ /**
1087 Took p & q in same level */
1088         if(level[p]-(1<<i) >= level[q])
1089             p = st[p][i];
1090     }
1091     if(p == q)
1092         return p; /** Here,q is the
1093 ancestor of p */
1094     for(int i=Log ; i>=0 ; i--){ /**
1095 Now,both nodes r in same level. Here we'll
1096 try to take both of them in a level
1097 together where parent of both of them will
1098 be equal */
1099     {
1100         if(st[p][i]!=-1 &&
1101 st[q][i]!=st[q][i]){
1102             p=st[p][i];
1103             q=st[q][i];
1104         }
1105     }
1106     return parent[p];
1107 }
1108 int main()
1109 {
1110     int i,j,k,n,e,u,v,q;
1111     scin2(n,e);
1112     rep(i,1,e){
1113         scin2(u,v);

```

```

1043 {
1044     ms(parent,-1); ms(level,0);
1045     ms(vis,0);
1046     queue<int>q;
1047     vis[start]=1; level[start]=0;
1048     q.push(start);
1049     while(!q.empty()){
1050         int u=q.front(); q.pop();
1051         for(int i=0 ; i<graph[u].size() ;
1052 i++){
1053             int v=graph[u][i];
1054             if(!vis[v]){
1055                 vis[v]=1; parent[v] =
1056 u; level[v] = 1+level[u]; q.push(v);
1057             }
1058         }
1059     }
1060 void LCA_Init(int n) /** Here,
1061 n is the number of node */
1062 {
1063     ms(st,-1); /**
1064 Initially, 2^j th parent of each node is
1065 -1 */
1066     for(int i=1 ; i<=n ; i++){
1067         st[i][0] = parent[i]; /** 2^0 th
1068 parent of each node is its original parent
1069 */
1070         for(int j=1 ; (1<<j)<=n ; j++){
1071             for(int i=1 ; i<=n ; i++){
1072                 if(st[i][j-1] != -1){
1073                     st[i][j] =
1074 st[st[i][j-1]][j-1]; /** konu node ar
1075 2^j th parent holo
1076 oi node ar 2^j-1 th parent
1077 ar 2^j-1 th parent. For example, 1th node ar
1078 2^4=16th parent holo tar
1079 2^3=8th parent ar 2^3=8th parent */
1080                 }
1081             }
1082         }
1083     }
1084 }
1085 }
1086 }

```

```

1104     graph[u].pb(v);
1105     graph[v].pb(u);
1106 }
1107 BFS(1);
1108 LCA_Init(n);
1109 scin(q);
1110 rep(i,1,q){
1111     scin2(u,v);
1112     pf("Lowest common ancestor of %d &
1113 %d = %d\n",u,v,LCA_Query(n,u,v));
1114 }
1115 return 0;
1116 }
1117 /** DS.14 - Path Cost In a Tree
1118 You're given a graph contains n node &
1119 m edges. You've to operate two different
1120 queries:
1121 q1. Update the cost of node x to v
1122 q2. Answer the total node cost between
1123 two nodes not necessary to be ancestor ,
1124 decendend
1125 You've to perform 10^5 queries
1126 */
1127 #define sz 30005
1128 bool vis[sz];
1129 int
1130 level[sz],parent[sz],sptable[sz][22],st[sz],
1131 ed[sz],cost[sz],ara[2*sz],tree[6*sz],curr_ti
1132 me;
1133 /** level-level of each node ;
1134 parent-parent of each node ; stable-sparse
1135 table ;
1136 st-discovering time of each node ;
1137 ed-finishing time of each node ;
1138 cost-number of genies in each node ;
1139 ara-reforming cost by discovering &
1140 finishing time
1141 tree-build segment tree */
1142 vi graph[sz];
1143 /** LCA part */
1144 void BFS(int start) /**
1145 Pre-calculate the parent & level of each

```

```

1132 node **/
1133 {
1134     ms(parent,-1); ms(vis,0); ms(level,0);
1135     queue<int>q;
1136     vis[start]=1; level[start]=0;
1137     q.push(start);
1138     while(!q.empty()){
1139         int u=q.front(); q.pop();
1140         for(int i=0 ; i<graph[u].size() ;
1141             i++){
1142             int v=graph[u][i];
1143             if(!vis[v]){
1144                 vis[v] = 1; parent[v] = u;
1145                 level[v] = 1+level[u];
1146             }
1147             q.push(v);
1148         }
1149     }
1150 void LCA_Init(int n) /** Here,
1151 n is the number of node **/
1152 {
1153     ms(sptable,-1); /**
1154     Initially, 2^j th parent of each node is
1155     -1 **/
1156     for(int i=1 ; i<=n ; i++){
1157         sptable[i][0] = parent[i]; /**
1158         2^0 th parent of each node is its original
1159         parent **/
1160         for(int j=1 ; (1<<j)<=n ; j++){
1161             for(int i=1 ; i<=n ; i++){
1162                 if(sptable[i][j-1] != -1){
1163                     sptable[i][j] =
1164                     sptable[sptable[i][j-1]][j-1]; /**
1165                     kono node ar 2^j th parent holo
1166                     ar node ar 2^j-1 th parent
1167                     ar 2^j-1 th parent. For example, 1ta node ar
1168                     2^4=16th parent holo tar
1169                     2^3=8th parent ar 2^3=8th parent **/
1170                 }
1171             }
1172         }
1173     }
1174 }

```

```

1191 int v=graph[start][i];
1192 if(!vis[v]) DFS(v);
1193 }
1194 ed[start] = ++curr_time;
1195 }
1196 void init(int node,int b,int e){
1197     if(b == e){
1198         tree[node] = ara[b]; return;
1199     }
1200     int mid=(b+e)/2;
1201     init(node<<1 , b , mid);
1202     init(1+(node<<1) , mid+1 , e);
1203     tree[node] =
1204     tree[node<<1]+tree[1+(node<<1)];
1205 }
1206 void update(int node,int b,int e,int
1207 i,int val){
1208     if(i>e || i<b) return;
1209     if(i==b && i==e){
1210         tree[node]=val;
1211         return;
1212     }
1213     int mid=(b+e)/2;
1214     update(node<<1 , b , mid , i , val);
1215     update(1+(node<<1) , mid+1 , e , i ,
1216 val);
1217     tree[node] =
1218     tree[node<<1]+tree[1+(node<<1)];
1219 }
1220 int query(int node,int b,int e,int i,int
1221 j){
1222     if(i>e || j<b) return 0;
1223     if(b>=i && e<=j) return tree[node];
1224     int mid=(b+e)/2;
1225     int p1 = query(node<<1 , b , mid , i
1226 , j);
1227     int p2 = query(1+(node<<1) , mid+1 ,
1228 e , i , j);
1229     return p1+p2;
1230 }
1231 void Clean(int n){
1232     curr_time=0;

```

```

1161 }
1162 int LCA_Query(int n,int p,int q) /**
1163 Here, n is the number of node. p & q are
1164 two nodes whose LCA we've to found out **/
1165 {
1166     if(level[p] < level[q])
1167         swap(p,q); /** P
1168 will always be more deeper node than q **/
1169     int Log=1;
1170     while(1){
1171         int nxt = Log+1;
1172         if((1<<nxt) > level[p])
1173             break;
1174         Log++;
1175     }
1176     for(int i=Log ; i>=0 ; i--){ /**
1177 Took p & q in same level **/
1178         if(level[p]-(1<<i) >= level[q])
1179             p = sptable[p][i];
1180     }
1181     if(p == q)
1182         return p; /** Here,q is the
1183 ancestor of p **/
1184     for(int i=Log ; i>=0 ; i--){ /**
1185 Now,both nodes r in same level. Here we'll
1186 try to take both of them in a level
1187 together where parent of both of them will
1188 be equal **/
1189     {
1190         if(sptable[p][i]!=-1 &&
1191 sptable[p][i]!=sptable[q][i]){
1192             p=sptable[p][i];
1193             q=sptable[q][i];
1194         }
1195     }
1196     return parent[p];
1197 } /** Segment Tree Part **/
1198 void DFS(int start){
1199     vis[start]=1;
1200     st[start] = ++curr_time;
1201     for(int i=0 ; i<graph[start].size() ;
1202         i++){
1203         for(int i=0 ; i<=n ; i++){
1204             graph[i].clear(); vis[i]=0;
1205             parent[i]=-1;
1206         }
1207     }
1208 int main()
1209 {
1210     int
1211     i,j,k,t,T,n,u,v,q,id,lca,tot,start,finish,p
1212     1,p2;
1213     scin(T);
1214     RUN_CASE(t,T){
1215         scin(n);
1216         rep(i,1,n)scin(cost[i]);
1217         rep(i,2,n){
1218             scin2(u,v);
1219             graph[u+1].pb(v+1);
1220             graph[v+1].pb(u+1);
1221         }
1222         DFS(1);
1223         for(i=1 ; i<=n ; i++){
1224             start = st[i]; finish = ed[i];
1225             ara[start] = cost[i];
1226             ara[finish] = -cost[i];
1227         }
1228         init(1,1,2*n); BFS(1);
1229     }
1230     LCA_Init(n);
1231     scin(q);
1232     pf("Case %d:\n",t);
1233     rep(i,1,q){
1234         sc("%d %d %d",&id,&u,&v);
1235         if(id){
1236             u += 1; cost[u] = v;
1237             update(1,1,2*n,st[u],v);
1238             update(1,1,2*n,ed[u],-v);
1239         }
1240         else{
1241             u+=1; v+=1;
1242             if(u > v)swap(u,v);
1243             lca = LCA_Query(n,u,v);
1244             start = min(st[u] ,
1245 st[lca]);

```

```

1261         finish = max(st[u] ,
1262         st[lca]);
1263         p1 =
1264         query(1,1,2*n,start,finish);
1265         start = min(st[v] ,
1266         st[lca]);
1267         finish = max(st[v] ,
1268         st[lca]);
1269         p2 =
1270         query(1,1,2*n,start,finish);
1271         tot = p1+p2-cost[lca];
1272         pf("%d\n",tot);
1273     }
1274     }Clean(n);
1275     }
1276     return 0;
1277 }
1278 /** DS.15 - LCA Problem - QTrEE2
1279 Problem: You r given a tree with n
1280 nodes & n-1 edges with w cost
1281 We will ask you to perfrom some
1282 instructions of the following form:
1283 DIST a b : ask for the distance
1284 between node a and node b
1285 or
1286 KTH a b k : ask for the k-th node on
1287 the path from node a to node b
1288 Solution: dis(a,b) =
1289 dis(root,a)+dis(root,b) -
1290 2*dis(root,LCA) */
1291 #define sz 100005
1292 bool vis[sz];
1293 ll
1294 level[sz],parent[sz],sptable[sz][25],dis[sz]
1295 ;
1296 vector<ll> graph[sz],cost[sz];
1297 void BFS(ll start) /** Pre-calculate the
1298 parent,level & distance of each node from
1299 root node */
1300 {
1301     ms(level,0);    ms(vis,0);
1302     ms(parent,-1);
1303
1304     if(level[p] < level[q])
1305         swap(p,q);
1306     /** P will always be more deeper node than
1307     q */
1308     ll Log=1;
1309     while(1){
1310         ll nxt=1+Log;
1311         if((1<<nxt) > level[p]) break;
1312         Log++;
1313     }
1314     for(ll i=Log ; i>=0 ; i--)
1315     /** Took p & q in same level */
1316     {
1317         if(level[p]-(1<<i) >= level[q]) p
1318         = sptable[p][i];
1319     }
1320     if(p == q) return p;
1321     for(ll i=Log ; i>=0 ; i--)
1322     /** Now,both nodes r in same level. Here
1323     we'll try to take both of them in a level
1324     together where parent of both of them will
1325     be equal */
1326     {
1327         if(sptable[p][i]!=-1 &&
1328         sptable[p][i]!=sptable[q][i]){
1329             p=sptable[p][i];
1330             q=sptable[q][i];
1331         }
1332     }
1333     return parent[p];
1334 }
1335 ll KthNodeInPath(ll n,ll u,ll v,ll
1336 k)
1337 /** We've to found out the k-th
1338 node on the path from node u to node v */
1339 {
1340     k-=1;
1341     /** u is the 1st node in the path which is
1342     already discovered,so decrease k by 1 */
1343     ll Log=0,dist,lca;
1344     lca =
1345     LCA_Query(n,u,v);
1346     /** To
1347
1348     queue<ll>q; vis[start] = 1;
1349     level[start] = dis[start] = 0;
1350     q.push(start);
1351     while(!q.empty()){
1352         ll u=q.front(); q.pop();
1353         for(ll i=0 ;
1354         i<(ll)graph[u].size() ; i++){
1355             int v=graph[u][i];
1356             if(!vis[v]){
1357                 vis[v] = 1; dis[v] =
1358                 cost[u][i]+dis[u];
1359                 level[v] = 1+level[u];
1360                 parent[v] = u; q.push(v);
1361             } } } }
1362
1363 void LCA_Init(ll n)
1364 /** Here, n is the number of node */
1365 {
1366     ms(sptable,-1);
1367     /** Initially, 2^j th parent of each node
1368     is -1 */
1369     for(int i=1 ; i<=n ; i++){
1370         sptable[i][0] = parent[i];
1371         /** 2^0 th parent of each node is its
1372         original parent */
1373         for(int j=1 ; (1<<j)<=n ; j++){
1374             for(int i=1 ; i<=n ; i++){
1375                 if(sptable[i][j-1] != -1){
1376                     sptable[i][j] =
1377                     sptable[sptable[i][j-1]][j-1]; /** kono
1378                     node ar 2^j th parent holo
1379                     ar node ar 2^j-1 th parent
1380                     ar 2^j-1 th parent. For example, 1ta node ar
1381                     2^4=16th parent holo tar
1382                     2^3=8th parent ar 2^3=8th parent */
1383                 } } } }
1384
1385 ll LCA_Query(ll n,ll p,ll q)
1386 /** Here, n is the number of node. p & q
1387 are two nodes whose LCA we've to found out
1388 */
1389 {
1390     move on the path from u to v we've to know
1391     the LCA for turning up */
1392     dist =
1393     abs(level[u]-level[lca]); /** From
1394     u to LCA how many level I can move */
1395     while(1){
1396         ll nxt=1+Log;
1397         if((1<<nxt) > dist) break;
1398         Log++;
1399     }
1400     for(ll i=Log ; i>=0 ; i--)
1401     { /**
1402     level[sptable[u][i]]>=level[lca] means
1403     that I can't go to upper level from LCA ;
1404     I've to turn to v from LCA */
1405         if((1<<i)<=k && sptable[u][i]!=-1
1406         && level[sptable[u][i]]>=level[lca])
1407         /** One kind of BS */
1408         { /** move u to
1409         maximum(LCA,kth node on the path) */
1410             k -= (1<<i);
1411             u = sptable[u][i];
1412         }
1413     }
1414     if(k > 0)
1415     { /** K>0 means we didn't go k th
1416     node yet ; So,we've to turn up through LCA
1417     and go deeper to node v */
1418         k = level[v]-k-level[lca]; /**
1419         We've to go upper of k level from node v
1420         for found out kth node from u */
1421         Log=0;
1422         dist =
1423         abs(level[v]-level[lca]); /** From v to
1424         LCA how many level I can move */
1425         while(1){
1426             ll nxt=1+Log;
1427             if((1<<nxt) > dist) break;
1428             Log++;
1429         }
1430         for(ll i=Log ; i>=0 ; i--)
1431         /** One kind of BS */

```

```

1365         {
1366             if((1<<i)<=k &&
sptable[v][i]!=-1 &&
level[sptable[v][i]]>=level[lca]){
1367                 k -= (1<<i);
1368                 v = sptable[v][i];
1369             }
1370         }
1371         return v;
1372     }
1373     else
1374         return u;
1375 }
1376 void Clean(ll n){
1377     for(ll i=0 ; i<=n ; i++){
1378         graph[i].clear();    cost[i].clear();
1379     }
1380 }
1381 int32_t main()
1382 {
1383     ll i,j,k,t,T,u,v,d,n,ans,lca;
1384     scln(T);
1385     RUN_CASE(t,T){
1386         scln(n);
1387         rep(i,2,n){
1388             scln2(u,v); scln(d);
1389             graph[u].pb(v); graph[v].pb(u);
1390             cost[u].pb(d); cost[v].pb(d);
1391         }
1392         BFS(1); LCA_Init(n);    string cmd;
1393         while(1){
1394             cin>>cmd;
1395             if(cmd == "DONE")
1396                 break;
1397             else if(cmd == "DIST"){
1398                 scln2(u,v);
1399                 lca = LCA_Query(n,u,v);
1400                 ans =
dis[u]+dis[v]-2*dis[lca];
1401                 pf("%lld\n",ans);
1402             }
1403             else{

```

```

1442         if(parent[i] != -1){
1443             sptable[i][0].lowest =
min(sptable[i][0].lowest,
cost[make_pair(i,parent[i])]);
1444             sptable[i][0].highest =
max(sptable[i][0].highest,
cost[make_pair(i,parent[i])]);
1445         }
1446     }
1447     for(int j=1 ; (1<<j)<=n ; j++){
1448         for(int i=1 ; i<=n ; i++){
1449             if(sptable[i][j-1].par != -1){
1450                 sptable[i][j].par =
sptable[sptable[i][j-1].par][j-1].par;
1451                 sptable[i][j].lowest =
min(min(sptable[i][j].lowest,sptable[i][j-1]
.lowest) ,
sptable[sptable[i][j-1].par][j-1].lowest);
1452                 sptable[i][j].highest =
max(max(sptable[i][j].highest,sptable[i][j-1]
.highest) ,
sptable[sptable[i][j-1].par][j-1].highest);
1453             }
1454         }
1455     }
1456 }
1457 pii LCA_Query(int n,int p,int q){
1458     if(p == q) return make_pair(0,0);
1459     pii ret = make_pair(infinity,0);
1460     if(level[p] < level[q]) swap(p,q);
1461     int Log=0;
1462     while(1){
1463         int nxt = Log+1;
1464         if((1<<nxt) > level[p])
1465             break;
1466         Log++;
1467     }
1468     for(int i=Log ; i>=0 ; i--){
1469         if(level[p]-(1<<i) >= level[q]){
1470             ret.first = min(ret.first ,
sptable[p][i].lowest);
1471             ret.second = max(ret.second ,

```

```

1404             scln2(u,v);
1405             scln(k);
1406             ans =
KthNodeInPath(n,u,v,k);
1407             pf("%lld\n",ans);
1408         }
1409     }
1410     pf("\n");    Clean(n);
1411 }
1412     return 0;
1413 }
1414 /** DS.16 - LCA problem
1415 You're given a rooted tree .Find the
length of the shortest and the length
of the longest road on the path between
the two given nodes. */
1416 #define sz 100005
1417 bool vis[sz]; int parent[sz],level[sz];
1418 vi graph[sz]; map<pii , int>cost;
1419 void BFS(int start)
1420 {
1421     //BFS FOR LCA
1422 }
1423 struct info
1424 {
1425     int par,lowest,highest;
1426     info() {}
1427     info(int a,int b,int c){
1428         par=a; lowest=b; highest=c;
1429     }
1430 }sptable[sz][20];
1431 void Init(int n){
1432     for(int i=0 ; i<=n ; i++){
1433         for(int j=0 ; j<20 ; j++){
1434             sptable[i][j] =
info(-1,infinity,0);
1435         }
1436     }
1437 }
1438 void LCA_Init(int n){
1439     Init(n);
1440     for(int i=1 ; i<=n ; i++){
1441         sptable[i][0].par = parent[i];

```

```

sptable[p][i].highest);
1472         p = sptable[p][i].par;
1473     }
1474 }
1475     if(p == q) return ret;
1476     for(int i=Log ; i>=0 ; i--){
1477         if(sptable[p][i].par!=-1 &&
sptable[p][i].par!=sptable[q][i].par){
1478             ret.first = min(ret.first ,
min(sptable[p][i].lowest ,
sptable[q][i].lowest));
1479             ret.second = max(ret.second ,
max(sptable[p][i].highest ,
sptable[q][i].highest));
1480             p=sptable[p][i].par;
1481             q=sptable[q][i].par;
1482         }
1483     }
1484     ret.first = min(ret.first ,
cost[make_pair(p,parent[p])]);
1485     ret.first = min(ret.first ,
cost[make_pair(q,parent[q])]);
1486     ret.second = max(ret.second ,
cost[make_pair(p,parent[p])]);
1487     ret.second = max(ret.second ,
cost[make_pair(q,parent[q])]);
1488     return ret;
1489 }
1490 int main()
1491 {
1492     int i,j,k,n,small,big,u,v,w,q;
1493     scin(n);
1494     rep(i,2,n){
1495         scin2(u,v); scin(w);
1496         graph[u].pb(v); graph[v].pb(u);
1497         cost[make_pair(u,v)]=w;
cost[make_pair(v,u)]=w;
1498     }
1499     BFS(1); LCA_Init(n); scin(q);
1500     rep(i,1,q){
1501         scin2(u,v);
1502         pii ans = LCA_Query(n,u,v);

```

```

1503         pf("%d %d\n",ans.first,ans.second);
1504     }
1505     return 0;
1506 }
1507 /** DS.18 - LCA Problem */
1508 /** Problem: you are given a rooted tree,
1509 where each node contains an integer value.
1510 And the value of a node is strictly
1511 greater than the value of its parent. Now
1512 you
1513 are given a node and an integer query. You
1514 have to find the greatest possible parent
1515 of this node (may include the node
1516 itself), whose value if greater than or
1517 equal to the given query integer. */
1518 bool vis[sz]; vi graph[sz];
1519 int
1520 level[sz],parent[sz],sptable[sz][25],cost[sz
1521 ];
1522 void BFS(int start)
1523 {
1524     ///BFS FOR LCA
1525 }
1526 void LCA_Init(int n){
1527     ms(sptable,-1);
1528     for(int i=1 ; i<=n ; i++)
1529         sptable[i][0]=parent[i];
1530     for(int j=1 ; (1<<j)<=n ; j++){
1531         for(int i=1 ; i<=n ; i++){
1532             if(sptable[i][j-1] != -1)
1533                 sptable[i][j] =
1534                     sptable[sptable[i][j-1]][j-1];
1535         }
1536     }
1537 }
1538 int LCA_Query(int node,int val){
1539     int Log=1,retnode=node;
1540     while(1){
1541         int nxt=Log+1;
1542         if((1<<nxt)>level[node]) break;
1543         Log++;
1544     }
1545 }

```

```

1546 Template(Iterative) */
1547 int
1548 n,CAP,weight[105],cost[105],dp[105][1005];
1549 /** State-1: Number of element ;
1550 State-2: Maximum size of knapsack */
1551 bool take[105];
1552 void KnapSack(){
1553     for(int i=1 ; i<=n ; i++){
1554         element **/ {
1555             dp[i][0]=0;
1556             for(int j=1 ; j<=CAP ; j++) /**
1557 Present knapsack size is j */ {
1558                 if(j >= weight[i])
1559                     dp[i][j] =
1560                         max(cost[i]+dp[i-1][j-weight[i]] ,
1561                             dp[i-1][j]);
1562                 else
1563                     dp[i][j] = dp[i-1][j];
1564             }
1565         }
1566     }
1567 }
1568 int main()
1569 {
1570     int i,j;
1571     ms(dp,0); scin2(n,CAP);
1572     for(i=1;i<=n;i++)
1573         scin2(weight[i],cost[i]);
1574     KnapSack();
1575     pf("\nOptimal answer is =
1576 %d\n",dp[n][CAP]);
1577 /** Identify the selected ITEM */
1578 j=CAP;
1579 for(i=n ; i>=1 ; i--){
1580     if(dp[i][j] == dp[i-1][j]) ;
1581     else{
1582         take[i]=1; j -= (weight[i]);
1583     }
1584 }
1585 /** Print the selected ITEM */
1586 for(i=1 ; i<=n ; i++){
1587     if(take[i] == 1) pf("ITEM-%d is
1588 selected\n",i);

```

```

1539     for(int i=Log ; i>=0 ; i--){ /**
1540 One kind of BS */
1541         int v=sptable[retnode][i];
1542         if(sptable[retnode][i]!=-1 &&
1543 cost[v]>=val)
1544             retnode=sptable[retnode][i];
1545     }
1546     return retnode;
1547 }
1548 void Clean(int n){
1549     for(int i=0 ; i<=n ; i++)
1550         graph[i].clear();
1551 }
1552 int main(){
1553     int i,j,k,t,T,n,q,u,v,ans,par,val;
1554     scin(T);
1555     RUN_CASE(t,T){
1556         scin2(n,q); cost[1]=1;
1557         rep(i,2,n){ /** Nodes denotes from
1558 1 to n */
1559             scin2(par,val);
1560             graph[i].pb(par+1);
1561             graph[par+1].pb(i); cost[i]=val;
1562         }
1563         BFS(1); LCA_Init(n);
1564         pf("Case %d:\n",t);
1565         rep(i,1,q){
1566             scin2(u,val); ans =
1567             LCA_Query(u+1,val);
1568             pf("%d\n",ans-1);
1569         }
1570         Clean(n);
1571     }
1572     return 0;
1573 }
1574 /** DP রদীক্ষা: 1.In DP, dp[2][12] is more
1575 efficient than dp[12][2]
1576 2. In DP, calculating from bigger to
1577 smaller state is more efficient. (In that
1578 case we haven't manet dp array in every
1579 case.) */
1580 /** DP.01 - 0-1 KnapSack

```

```

1601     }
1602     return 0;
1603 }
1604 /** DP.02 - Space Optimized knapsack */
1605 int n,cap,wt[505],val[505],dp[2][2000005];
1606 /** State-1: i th and (i-1)th element ;
1607 State-2: Maximum size of knapsack
1608 // Constraint: cap <= 2000000 ; n <= 500
1609 // val[i] <= 10^7 ; wt[i] <= 10^7
1610 void SpaceOptimizedKnapsack(){
1611     int present,past;
1612     for(int i=1 ; i<=n ; i++){
1613         if(i%2 == 0)
1614             present=0, past=1;
1615         else
1616             present=1, past=0;
1617         for(int j=1 ; j<=cap ; j++){
1618             if(wt[i] <= j)
1619                 dp[present][j] =
1620                     max(val[i]+dp[past][j-wt[i]] ,
1621                         dp[past][j]);
1622             else
1623                 dp[present][j] =
1624                     dp[past][j];
1625         }
1626     }
1627     pf("%d\n",dp[present][cap]);
1628 }
1629 int main()
1630 {
1631     scin2(cap , n);
1632     for(int i=1 ; i<=n ; i++)
1633         scin2(val[i] , wt[i]);
1634     SpaceOptimizedKnapsack();
1635     return 0;
1636 }
1637 /** DP.03 - LCS Template */
1638 string s1,s2,res;
1639 int len1,len2,dp[1005][1005]; /** State-1:
1640 Size of s1 ; State-2: Size of s2 */
1641 int LCS(int pos1,int pos2){
1642     if(pos1>=len1 || pos2>=len2)

```

```

1637     return 0;
1638     if(dp[pos1][pos2] != -1) return
dp[pos1][pos2];
1639     else{
1640         if(s1[pos1] == s2[pos2])
            return dp[pos1][pos2] =
1+LCS(pos1+1,pos2+1);
1641         else{
1642             int res1=LCS(pos1,pos2+1);
1643             int res2=LCS(pos1+1,pos2);
1644             return dp[pos1][pos2] =
max(res1,res2);
1645         }
1646     }
1647 }
1648 void Print(int pos1,int pos2) /** Print
the Longest Common Subsequence */{
1649     if(pos1>=len1 || pos2>=len2) return;
1650     if(s1[pos1] == s2[pos2]){ /** Take
the character */
1651         res += s1[pos1] ;
1652         Print(pos1+1,pos2+1);
1653     }
1654     else{
1655         int res1=dp[pos1][pos2+1];
1656         int res2=dp[pos1+1][pos2];
1657         if( res1 >= res2 ) Print( pos1 ,
pos2 + 1 ) ; /** Go,where the result is
max */
1658         else Print( pos1 + 1 , pos2 ) ;
1659     }
1660 }
1661 void PrintAll(int pos1,int pos2) /**
Print all LCS */{
1662     if(pos1>=len1 || pos2>=len2){
1663         cout<<res<<endl; return;
1664     }
1665     if(s1[pos1] == s2[pos2]){
1666         res += s1[pos1];
1667         PrintAll(pos1+1,pos2+1);
1668         res.erase(res.end()-1); /**
Delete the last Character */

```

```

1707 int longest(int u){
1708     if(dp[u] != -1)
1709         return dp[u];
1710     int maxi=0;
1711     for(int v=u+1;v<=n;v++){
1712         if(value[v] > value[u]){
1713             if(longest(v) > maxi){
1714                 maxi=longest(v);
1715                 dir[u]=v;
1716             }
1717         }
1718     }
1719     return dp[u]=1+maxi;
1720 }
1721 int main()
1722 {
1723     ms(dp,-1); ms(dir,-1);
1724     int LIS=0,start,i;
1725     for(int i=1;i<=n;i++){
1726         if(longest(i) > LIS){
1727             LIS=longest(i);
1728             start=i;
1729         }
1730     }
1731     pf("LIS = %d , starting point
%d\n\n",LIS,start);
1732     solution(start);
1733     return 0;
1734 }
1735 /** DP.05 - LIS Template - O(n log (k)) */
1736 const int inf = 2000000000; ///Infinity
1737 int
n,ara[2000005],L[2000005],I[2000005],sequenc
e[2000005];
1738 void takeInput(){
1739     scin(n);
1740     for(int i=0 ; i<n ; i++)
1741         scin(ara[i]);
1742 }
1743 int LisNlogK(){
1744     int i;
1745     I[0]=-inf;

```

```

1669     }
1670     else{
1671         int res1=dp[pos1][pos2+1];
1672         int res2=dp[pos1+1][pos2];
1673         if(res1 > res2)
1674             PrintAll(pos1,pos2+1);
1675         else if(res1 < res2)
1676             PrintAll(pos1+1,pos2);
1677         else /** If both call return
same ans,then go both side */
1678         {
1679             PrintAll(pos1,pos2+1);
1680             PrintAll(pos1+1,pos2);
1681         }
1682     }
1683 }
1684 int main()
1685 {
1686     ms(dp,-1);
1687     int i,j,ans;
1688     cin>>s1; cin>>s2;
1689     len1=s1.size(); len2=s2.size();
1690     ans = LCS(0,0);
1691     pf("Size of LCS : %d\n",ans);
1692     pf("LCS:\n"); res = "" ;
Print(0,0) ;
1693     cout<< res << endl ;
1694     pf("ALL LCS:\n"); res = "" ;
PrintAll(0,0);
1695     return 0;
1696 }
1697 /** DP.04 - LIS Template - O(n^2) */
1698 int n=7,value[]={-1000000,5,0,9,2,7,3,4};
1699 int dp[1000],dir[1000];
1700 /**Printing Soln*/
1701 void solution(int start){
1702     while(dir[start] != -1){
1703         pf("Index %d ; value =
%d\n",start,value[start]);
1704         start=dir[start];
1705     }
1706 }

```

```

1746     for(i=1 ; i<=n ; i++){
1747         I[i] = inf; L[i]=n;
1748     }
1749     int LisLen=0; /** Keeps the maximum
position where a data is inserted */
1750     for(i=0 ; i<n ; i++){
1751         int lo=0,hi=LisLen,mid;
1752         while(lo <= hi) /** Find the
suitable position for ara[i] */
1753         {
1754             mid=(lo+hi)/2;
1755             if(ara[i] > I[mid])
1756                 lo=mid+1;
1757             else
1758                 hi=mid-1;
1759         }
1760         /** Observe the BS carefully ,
when the BS ends low>hi and we put our
item in I[lo] */
1761         I[lo]=ara[i]; L[i]=lo;
1762         if(LisLen < lo) /** LisLen
contains maximum position */
1763             LisLen=lo;
1764     }
1765     return LisLen;
1766 }
1767 void findSequence(int mxlen){
1768     int i=0,j;
1769     /** at first find the position of the
item whose L[] is maximum */
1770     for(j=1 ; j<n ; j++){
1771         if(L[j] > L[i]) i=j;
1772     }
1773     /** initialize the position in
sequence where the items can be added ;
observe that the
data r saving from right to left */
1774     int top=L[i]-1;
1775     /** insert the item in ith position to
sequence */
1776     sequence[top--] = ara[i];
1777     for(j=i-1 ; j>=0 ; j--){
1778

```



```

1779         if(ara[j]<ara[i] && L[j]==L[i]-1){
1780             /** we have found a valid
item. so we'll save it */
1781             i=j;     sequence[top--]=ara[i];
1782         }
1783     }
1784     pf("LIS is : \n");
1785     for(i=0 ; i<mxlen ; i++)
1786         pf("%d ",sequence[i]);
1787     pf("\n");
1788 }
1789 int main()
1790 {
1791     takeInput();     int res=LisNlogK();
pf("The LIS length is %d\n",res);
1792     findSequence(res);
1793     return 0;
1794 }
1795 /** DP.06 - Matrix Chain Multiplication
Template */
1796 int row[105],col[105],dp[105][105];
1797 bool vis[105][105];
1798 int FuN(int BEG,int END){
1799     if(BEG >= END) return 0;
1800     if(vis[BEG][END]) return dp[BEG][END];
1801     int ret=1<<30;
1802     for(int MID=BEG ; MID<END ; MID++){
1803         int LeftRet = FuN(BEG , MID);
1804         int RightRet = FuN(MID+1 , END);
1805         int MulLeftRight =
row[BEG]*col[MID]*col[END];
1806         int tot =
LeftRet+RightRet+MulLeftRight;
1807         ret = min(ret , tot);
1808     }
1809     vis[BEG][END]=1;
1810     return dp[BEG][END] = ret;
1811 }
1812 int main()
1813 {
1814     int n;   cin>>n;
1815     for(i=0 ; i<n ; i++)

```

```

1849     ms(vis , 0);     scln2(n , k);
1850     rep(i , 1 , n)   scln(ara[i]);
1851     ans = FuN(1 , 1 , 0);
cout<<ans<<endl;
1852     return 0;
1853 }
1854 /** DP.08 - Digit DP-1 : How many zeros?
1855 Problem: Jimmy writes down the decimal
representations of all natural numbers
between and including m and n, (m ≤ n).
How many zeros will he write down? */
1856 ll dp[2][2][15][15];
1857 /** State-1: is_less_than_given_value ;
State-2:
is_current_position_is_starting_position
1858 State-3:
current_position_from_left_side ; State-4:
total_zero_so_far_in_this_created_number
*/
1859 vector<int>num;
1860 ll DP(ll isSmall , ll isStart , ll pos ,
ll totZero){
1861     if(pos >= (ll)num.size())
1862         return totZero;     /** Return
number of zero used in this number so far
*/
1863     if(dp[isSmall][isStart][pos][totZero]
!= -1)
1864         return
dp[isSmall][isStart][pos][totZero];
1865     ll ret=0LL,can_be_taken=0;
1866     if(isSmall)
1867         can_be_taken=9; /** If it already
small,then any digit can be taken */
1868     else
1869         can_be_taken=num[pos]; /** If its
prefix is equal then digit can be taken
till num[pos] */
1870     if(!isStart){     /** If its not a
starting digit , then 0 can be taken */
1871         for(int i=0 ; i<=can_be_taken ;
i++){

```

```

1816         cin>>row[i]>>col[i];
1817         cout<<FuN(0 , n-1)<<endl;
1818         return 0;
1819     }
1820     /** DP.07 - Maximum Sum of K-subArrays
Problem: You've given an array of n
elements. You've to select k non-empty
subArrays so that total sum of all sub
arrays will be maximize. */
1822 ll n,k,ara[5005],dp[2][5005][5005];
1823 bool vis[2][5005][5005];
1824 ll FuN(bool isStart,ll pos,ll subArrayNo){
1825     if(pos > n){
1826         if(subArrayNo == k) return 0LL;
1827         else return -longlimit;
1828     }
1829     if(vis[isStart][pos][subArrayNo])
1830         return dp[isStart][pos][subArrayNo];
1831     ll
ret1=-longlimit,ret2=-longlimit,ret3=-longli
mit;
1833     if(isStart){
1834         ret1 = ara[pos] + FuN(0 , pos+1 ,
subArrayNo+1);
1835         ret3 = ara[pos] + FuN(1 , pos+1 ,
subArrayNo+1);
1836         ret2 = FuN(1 , pos+1 , subArrayNo);
1837     }
1838     else{
1839         ret1 = ara[pos] + FuN(0 , pos+1 ,
subArrayNo);
1840         ret2 = FuN(1 , pos+1 , subArrayNo);
1841         ret3 = ara[pos] + FuN(1 , pos+1 ,
subArrayNo);
1842     }
1843     vis[isStart][pos][subArrayNo] = 1;
1844     return dp[isStart][pos][subArrayNo] =
max(ret1 , max(ret2 , ret3));
1845 }
1846 int main()
1847 {
1848     ll i,j,ans;

```

```

1872         ret += DP(isSmall |
(i<num[pos]) , 0 , pos+1 ,
(i==0)+totZero); /** If i<num[pos] then
isSmall OR 1 = 1 and if i==0 the number
of totZero in this number will increased
by 1*/
1873     }
1874 }
1875     else{
1876         for(int i=1 ; i<=can_be_taken ;
i++){     /** we can't put zero in starting
position */
1877             ret += DP(isSmall |
(i<num[pos]) , 0 , pos+1 , (i==0)+totZero);
1878         }
1879         ret += DP(1 , 1 , pos+1 ,
totZero); /** IF its a starting position ,
then we can put nothing in that position.
1880         Then the position will be starting
position again , and it will be smaller
than max range */
1881     }
1882     return
dp[isSmall][isStart][pos][totZero] = ret;
1883 }
1884 ll Calculate(ll x){
1885     if(x < 0)
1886         return 0LL;
1887     else if(x <= 9)
1888         return 1LL;
1889     num.clear();     ms(dp , -1);
1890     while(x){
1891         num.pb(x%10);     x /= 10;
1892     }
1893     reverse(num.begin() , num.end());
1894     return DP(0 , 1 , 0 , 0) + 1; /** 0
can't be taken ; So,add 1 by default */
1895 }
1896 int main()
1897 {
1898     ll t,T,n,m,ret1,ret2,ans;
1899     scln(T);

```

```

1900     RUN_CASE(t,T){
1901         scln2(m,n);
1902         ans = Calculate(n) -
Calculate(m-1);
1903         pf("Case %lld: %lld\n",t,ans);
1904     }
1905     return 0;
1906 }
1907 /** DP.09 - Digit DP-2: Digit Sum
1908 Problem: Given the numbers a and b,
calculate the digit sum(sum of all digit
between a and b) of [a,b]. **/
1909 ll dp[2][2][16][16][145];
1910 /** State-1: Is this number smaller than
the range
1911 State-2: Is this position is starting
position or not
1912 State-3: Position
1913 State-4: Total number of digit used in
this number
1914 State-5: Sum of digit of this
number **/
1915 vector<int>num;
1916 ll FuN(bool isSmall,bool isStart,int
pos,int tot_digit,int sum_digit){
1917     if(pos >= (int)num.size()){
1918         return (ll)sum_digit;
1919
1920     if(dp[isSmall][isStart][pos][tot_digit][sum_
digit] != -1)
1921         return
dp[isSmall][isStart][pos][tot_digit][sum_dig
it];
1922     ll ret=0,can_be_taken=0;
1923     if(isSmall)
1924         can_be_taken=9;
1925     else
1926         can_be_taken=num[pos];
1927     if(!isStart){
1928         for(int i=0 ; i<=can_be_taken ;
i++){
1929             ret +=
FuN(isSmall|(i<num[pos]) , 0 , pos+1 ,
tot_digit+1 , sum_digit+i);
1930         }
1931     }
1932     else{
1933         for(int i=1 ; i<=can_be_taken ;
i++){
1934             ret +=
FuN(isSmall|(i<num[pos]) , 0 , pos+1 ,
tot_digit+1 , sum_digit+i);
1935             ret += FuN(1 , 1 , pos+1 , 0 , 0);
1936         }
1937     }
1938     return
dp[isSmall][isStart][pos][tot_digit][sum_dig
it] = ret;
1939 }
1940 ll Calculate(ll x){
1941     if(x <= 0) return 0;
1942     num.clear(); ms(dp , -1);
1943     while(x){
1944         num.pb(x%10); x /= 10;
1945     }
1946     reverse(num.begin() , num.end());
1947     return FuN(0 , 1 , 0 , 0 , 0);
1948 }
1949 int main()
1950 {
1951     ll t,T,a,b,ans;
1952     scln(T);
1953     RUN_CASE(t,T){
1954         scln2(a,b);
1955         if(a > b) swap(a,b);
1956         ans = Calculate(b)-Calculate(a-1);
1957         pf("%lld\n",ans);
1958     }
1959     return 0;
1960 }
1961 /** DP.10 - Digit DP 3
1962 Problem: Count the number of integers
in the range [A, B] which are divisible by
K and the sum of its digits is also
divisible by K. **/
1963 int dp[2][2][15][85][85];
1964
1965 /** State-1: is_less_than_given_value ;
1966 State-2:
1967 is_current_position_starting_position
1968 State-3: current_position_from_left_side ;
1969 State-4: remainder_of_value_i_have_made ;
1970 State-5: remainder_of_sum_of_digit **/
1971 vector<int>num; int K;
1972 int FuN(int isSmall , int isStart , int
pos , int val_remainder , int sum_remainder)
1973 {
1974     if(pos >= (int)num.size()){
1975         if(isStart)
1976             return 0; /** If its a
1977 starting position even now! the value is
1978 zero which is invalid **/
1979         if(val_remainder%K==0 &&
sum_remainder%K==0) return 1;
1980         else return 0;
1981     }
1982     if(dp[isSmall][isStart][pos][val_remainder][
sum_remainder] != -1)
1983         return
dp[isSmall][isStart][pos][val_remainder][sum_
remainder];
1984     int ret=0 , can_be_taken=0;
1985     if(isSmall)
1986         can_be_taken = 9; /** If the
1987 number already small,then any digit can be
1988 taken **/
1989     else
1990         can_be_taken = num[pos]; /** If
1991 its prefix is equal then digit can be
1992 taken till num[pos] **/
1993     if(!isStart){
1994         for(int i=0 ; i<=can_be_taken ;
i++){
1995             /** If its not a starting digit
1996 , then 0 can be taken **/
1997             ret +=
FuN(isSmall|(i<num[pos]) , 0 , pos+1 ,
(val_remainder*10+i)%K ,
(sum_remainder+i)%K);
1998         }
1999     }
2000     else{
2001         for(int i=1 ; i<=can_be_taken ;
i++){
2002             /** we can't put zero in
2003 starting position **/
2004             ret +=
FuN(isSmall|(i<num[pos]) , 0 , pos+1 ,
(val_remainder*10+i)%K ,
(sum_remainder+i)%K);
2005             ret += FuN(1 , 1 , pos+1 , 0 ,
0); /** IF its a starting position , then
2006 we can also put nothing in that position.
2007 Then the position will be starting
2008 position again , and it will be smaller
2009 than max range **/
2010         }
2011     }
2012     return
dp[isSmall][isStart][pos][val_remainder][sum_
remainder] = ret;
2013 }
2014 int Calculate(int x)
2015 {
2016     if(x <= 0) return 0;
2017     num.clear(); ms(dp , -1);
2018     while(x){
2019         num.pb(x%10);
2020         x /= 10;
2021     }
2022     reverse(num.begin() , num.end());
2023     return FuN(0 , 1 , 0 , 0 , 0);
2024 }
2025 int main(){
2026     int t,T,ans,A,B;
2027     scln(T);
2028     RUN_CASE(t,T){
2029         sc("%d %d %d",&A,&B,&K);
2030         if(A>B) swap(A,B);
2031         if(K == 1) ans = B-A+1; /**
2032 All number in the range will fulfill the
2033 given condition **/

```

```

2012     else if(K > 81) ans = 0;
2013     /** None of the number will fulfill the
2014     condition digit_sum%K==0; because maximum
2015     digit sum for 9 position can be 81 */
2016     else ans =
2017     Calculate(B)-Calculate(A-1);
2018     pf("Case %d: %d\n",t,ans);
2019     return 0;
2020 }
2021 /** DP.11 - BitMask template ; complexity
2022 (2^n)*(n^2) */
2023 int Set(int N,int pos){return N=N |
2024 (1<<pos);}
2025 int reset(int N,int pos){return N= N &
2026 ~(1<<pos);}
2027 bool check(int N,int pos){return (bool)(N
2028 & (1<<pos));}
2029 int w[20][20],n,dp[(1<<15)+2];
2030 int call(int mask){
2031     if(mask==(1<<n)-1) return 0;    /**
2032     Bought everything */
2033     if(dp[mask]!=-1) return dp[mask];
2034     int mn=1<<28;
2035     for(int i=0; i<n; i++){
2036         if(check(mask,i)==0){    /** ith
2037         item haven't bought yet */
2038             int price=w[i][i];    /** Base
2039             price of ith item */
2040             for(int j=0; j<n; j++){
2041                 if(i!=j and
2042                 check(mask,j)!=0){    /** Increase the price
2043                 if I've already bought jth item */
2044                     price+=w[i][j];
2045             }
2046             int
2047             ret=price+call(Set(mask,i));
2048             mn=min(mn,ret);
2049         }
2050     }
2051     return dp[mask]=mn;

```

```

2073 int FuN(int isParTaken,int node){
2074     if(dp[isParTaken][node] != -1)
2075         return dp[isParTaken][node];
2076     int ret1=0,ret2=0;
2077     if(isParTaken || par[node]==0){
2078         ///IF parent of this node is taken,then we
2079         can both take this node or ignore this node
2080         ret1 = 0;
2081         for(int i=0 ;
2082         i<(int)graph[node].size() ; i++){
2083             int v = graph[node][i];
2084             if(v != par[node])
2085                 ret1 += FuN(0 , v);
2086         }
2087         ret2 = 1;
2088         for(int i=0 ;
2089         i<(int)graph[node].size() ; i++){
2090             int v = graph[node][i];
2091             if(v != par[node])
2092                 ret2 += FuN(1 , v);
2093         }
2094         return dp[isParTaken][node] =
2095         min(ret1 , ret2);
2096     }
2097     else{    ///IF parent of this node
2098     was not taken, then we have to take this
2099     node
2100         ret2 = 1;
2101         for(int i=0 ;
2102         i<(int)graph[node].size() ; i++){
2103             int v = graph[node][i];
2104             if(v != par[node])
2105                 ret2 += FuN(1 , v);
2106         }
2107         return dp[isParTaken][node] = ret2;
2108     }
2109 }
2110 int main()
2111 {
2112     int i,j,k,n,u,v,ans;
2113     scin(n);
2114     for(i=1 ; i<n ; i++){

```

```

2040 }
2041 int main()
2042 {
2043     mem(dp,-1);
2044     cin>>n;
2045     for(int i=0; i<n; i++){
2046         for(int j=0; j<n; j++){
2047             scanf("%d",&w[i][j]);
2048         }
2049     }
2050     int ret=call(0);    printf("%d\n",ret);
2051     return 0;
2052 }
2053 /** DP.11 - Tree DP Example
2054 Problem: Vertex cover with tree dp
2055 You are given an unweighted, undirected
2056 tree. Write a program to find a vertex set
2057 of minimum size in this tree such that
2058 each edge has as least one of its
2059 end-points in that set.
2060 Solution:
2061 1. A parent makes an edge with a child.
2062 2. In every child, if we take its
2063 parent,then we can take this node also or
2064 can leave this node.
2065 3. IF we have not taken its parent, then
2066 we must have to take this child.    */
2067 #define sz 200005
2068 int vis[sz];
2069 vector<int> graph[sz];
2070 int par[sz],dp[5][sz];    ///State-1:Is
2071 parent taken? State-2:node no.
2072 void DFS(int start){
2073     vis[start] = 1;
2074     for(int i=0 ;
2075     i<(int)graph[start].size() ; i++){
2076         int v = graph[start][i];
2077         if(!vis[v]){
2078             par[v] = start; DFS(v);
2079         }
2080     }
2081 }

```

```

2107     scin2(u,v);
2108     graph[u].pb(v); graph[v].pb(u);
2109 }
2110 ms(vis , 0);    par[1] = 0; DFS(1);
2111 ms(vis , 0);    ms(dp , -1);
2112 ans = FuN(0 , 1);    cout<<ans<<endl;
2113     return 0;
2114 }
2115 /** DP.12 - Graph + DP - ICPC,19
2116 Problem: You're given a wighted tree. A
2117 tree is said to be beautiful if the
2118 summation of all pair distance of the
2119 vertex of the tree is non-negative(if n=5
2120 , then
2121 d(1,2)+d(1,3)+d(1,4)+d(1,5)+d(2,3)+d(2,4)+d(
2122 2,5)+d(3,4)+d(3,5)+d(4,5) >= 0 )
2123 You have to determine whether the given
2124 tree is beautiful or not? IF the tree is
2125 not beautiful, you have to perform a
2126 series of operating to make the given tree
2127 beautiful.
2128 The operation is: Select an edge whose
2129 weight is negative and increase its weight
2130 by 1.
2131 Now, u have to determine the minimum
2132 number of times u have to perform the
2133 operation to make the given tree
2134 beautiful.    */
2135 bool vis[sz]; vi graph[sz];
2136 int dp[sz],level[sz];
2137 void make_graph(int u,int v){
2138     graph[u].pb(v); graph[v].pb(u);
2139 }
2140 void DFS(int start){
2141     vis[start] = 1;
2142     int sum = 0;
2143     for(int i=0 ;
2144     i<(int)graph[start].size() ; i++){
2145         ll v = graph[start][i];
2146         if(!vis[v]){
2147             level[v] = 1+level[start];
2148             DFS(v); sum += (1+dp[v]);

```

```

2132     }
2133 }
2134 dp[start] = sum;
2135 }
2136 pii edges[sz];
2137 map<pii, int> edgcost, repeat;
2138 struct info{
2139     int u,v,w,rept;
2140     info(){}
2141     info(int a,int b,int c,int d){
2142         u=a; v=b; w=c; rept=d;
2143     }
2144 };
2145 vector<info> q;
2146 bool cmp(info p1, info p2){
2147     return p1.rept<p2.rept;
2148 }
2149 ll BS(ll sum){
2150     if(sum >= 0LL)
2151         return 0LL;
2152     sort(q.begin(), q.end(), cmp);
2153     ll
2154     i,j,k,u,v,w,tempsum=fabs(sum),cnt=0LL,lo,hi
2155     ,mid,res,bad;
2156     while(!q.empty() && tempsum>0){
2157         info top = q.back(); q.pop_back();
2158         u = top.u; v = top.v; w =
2159         fabs(top.w);
2160         if(level[v] > level[u]) swap(u,v);
2161         lo=0;hi=fabs(w);res=-1;
2162         while(lo <= hi){
2163             mid = (lo+hi)/2;
2164             bad = top.rept*mid;
2165             if(bad >= tempsum){
2166                 res = mid; hi = mid-1;
2167             }
2168             else
2169                 lo = mid+1;
2170         }
2171         if(res == -1){
2172             tempsum -= (top.rept * w);
2173             cnt += w;

```

```

2209         repeat[make_pair(u,v)] = temp;
2210         if(edgcost[make_pair(u,v)] <
2211         0)
2212             q.push_back(info(u, v,
2213             edgcost[make_pair(u,v)], temp));
2214             sum +=
2215             (temp*edgcost[make_pair(u,v)]);
2216         }
2217         ans = BS(sum); pf("Case %d:
2218         %lld\n",t,ans); Clean(n);
2219     }
2220     return 0;
2221 }
2222 /** DP.13 - Palindrome Partitioning
2223 Problem: A palindrome partition is the
2224 partitioning of a string such that each
2225 separate substring is a palindrome.
2226 Count minimum possible number of
2227 substrings in a palindrome partition of a
2228 string */
2229 int minPalPartition(char str[]){
2230     int n=strlen(str),i,j,k,L;
2231     bool isPal[n][n]; /**
2232     isPal[i][j]=true if str[i..j] is a
2233     palindrome */
2234     int dp[n]; /** Minimum number of
2235     cuts needed for palindrome partitioning of
2236     substring str[0..i] */
2237     for(i=0 ; i<n ; i++){
2238         isPal[i][i]=1; /** Every one
2239         length substring is a palindrome */
2240         for(L=2 ; L<=n ; L++){ /** Build
2241         solution for L size substring */
2242             for(i=0 ; i<n-L+1 ; i++){ /**
2243             From index i,take a L length substring */
2244                 j=i+L-1; /** Here j is the
2245                 ending index */
2246                 if(L == 2)
2247                     isPal[i][j] =
2248                     (str[i]==str[j]); /** If two character
2249                     matched,then its a palindrome */
2250                 else

```

```

2170     }
2171     else{
2172         tempsum -= (top.rept * res);
2173         cnt += res;
2174         break;
2175     }
2176 }
2177 return cnt;
2178 }
2179 void Clean(ll n){
2180     for(ll i=0 ; i<=n ; i++){
2181         vis[i] = 0; dp[i] = 0;
2182         level[i] = 0; graph[i].clear();
2183     }
2184     q.clear(); edgcost.clear();
2185     repeat.clear();
2186 }
2187 int main(){
2188     int i,j,k,n,u,v,w,t,T;
2189     ll ans,sum,temp;
2190     scin(T);
2191     RUN_CASE(t,T)
2192     {
2193         sum = 0LL;
2194         scin(n);
2195         for(i=1 ; i<=n-1 ; i++){
2196             scin2(u,v);
2197             scin(w);
2198             edges[i] = make_pair(u,v);
2199             edgcost[make_pair(u,v)] = w;
2200             edgcost[make_pair(v,u)] = w;
2201             make_graph(u,v);
2202         }
2203         DFS(1);
2204         for(i=1 ; i<=n-1 ; i++){
2205             u = edges[i].first; v =
2206             edges[i].second;
2207             if(level[v] > level[u])
2208                 swap(u,v);
2209             ll under = dp[u];
2210             ll upper = n-(dp[u]+1);
2211             temp = (under+1)*upper;

```

```

2233         isPal[i][j] =
2234         (str[i]==str[j]) && isPal[i+1][j-1]; /**
2235         IF str[i+1..j-1] is a palindrome and
2236         str[i]==str[j] */
2237     }
2238     }
2239     for(i=0 ; i<n ; i++){
2240         if(isPal[0][i] == true)
2241             dp[i]=0;
2242         else{
2243             dp[i] = infinity;
2244             for(j=0 ; j<i ; j++){
2245                 /** If str[j+1..i] is a
2246                 palindrome & min cut of str[0..j]+1 < min
2247                 cut of str[0..i] then update min cut of
2248                 str[0..i] */
2249                 if(isPal[j+1][i]==true &&
2250                 1+dp[j]<dp[i])
2251                     dp[i] = 1+dp[j];
2252             }
2253         }
2254     }
2255     return 1+dp[n-1]; /** Return the min
2256     cut value for complete string */
2257 }
2258 int main()
2259 {
2260     int t,T,ans;
2261     scin(T);
2262     RUN_CASE(t,T){
2263         char s1[1005]; sc("%s",s1);
2264         ans = minPalPartition(s1);
2265         pf("Case %d: %d\n",t,ans);
2266     }
2267     return 0;
2268 }
2269 /** DP.14 - Longest Palindromic
2270 Subsequence */
2271 int dp[1005][1005]; string s;
2272 int LPS(int pos1,int pos2){
2273     if(pos1 == pos2) /** If there is
2274     only one character */

```

```

2265         return 1;
2266         if(s[pos1]==s[pos2] &&
pos1+1==pos2) /** If there r only two
characters & both r same */
2267         return 2;
2268         if(dp[pos1][pos2] != -1) return
dp[pos1][pos2];
2269         else{
2270             if(s[pos1] == s[pos2]){
2271                 return dp[pos1][pos2] =
2+LPS(pos1+1, pos2-1);
2272             }
2273             else
2274                 return dp[pos1][pos2] =
max(LPS(pos1+1,pos2), LPS(pos1,pos2-1));
2275         }
2276         /** Print the solution */
2277         vector<char>v1,v2;
2278         void Print(int pos1,int pos2){
2279             if(pos1 == pos2){
2280                 v1.pb(s[pos1]); return;
2281             }
2282             if(s[pos1]==s[pos2] && pos1+1==pos2){
2283                 v1.pb(s[pos1]); v2.pb(s[pos2]);
2284                 return;
2285             }
2286             if(s[pos1] == s[pos2]){
2287                 v1.pb(s[pos1]); v2.pb(s[pos2]);
2288                 Print(pos1+1, pos2-1);
2289             }
2290             else{
2291                 if(dp[pos1+1][pos2] >
dp[pos1][pos2-1]) Print(pos1+1,pos2);
2292                 else Print(pos1,pos2-1);
2293             }
2294         }
2295         int main()
2296         {
2297             int i,j,lps,len;
2298             cin>>s; ms(dp,-1);
2299             lps = LPS(0,s.size()-1);
2300             cout<<lps<<endl;
2301
2302             mxlen=high-low+1;
2303             }
2304             --low; ++high;
2305         }
2306         /** Find longest even length
palindrome with center point i */
2307         low=i-1; high=i+1;
2308         while(low>=0 && high<len &&
str[low]==str[high]){
2309             if(high-low+1 > mxlen){
2310                 start=low;
2311                 mxlen=high-low+1;
2312             }
2313             --low; ++high;
2314         }
2315         PrintSubstring(str,start,start+mxlen-1);
2316         return mxlen;
2317     }
2318     int main()
2319     {
2320         int mx;
2321         string s1;
2322         cin>>s1;
2323         mx = LongestPalSubstr(s1);
2324         pf("Length of longest palindromic
substring is = %d\n",mx);
2325         return 0;
2326     }
2327     /** DP.16 - DP on grid (Baker Bro) */
2328     int m,n,grid[105][105],dp[105][105][105][2];
2329     /** Approach: এই প্রব্লেমে ধরে নেই যে দুইজন লোক গ্রিডের ১ম
সেল থেকে শেষ সেলে যাবে এবং পয়েন্টগুলো কালেক্ট করবে। মনে রাখতে
হবে, এই দুইজন লোকের
2330     রাস্তা শুধুমাত্র গ্রিডের ১ম এবং শেষ সেলেই মিলিত হতে পারবে। আর
উভয় ব্যক্তিই হয় নিচে নাহয় ডানে মুভ করতে পারবে। মানে রো এবং কলাম এর
মান শুধু বাড়তেই পারবে।
2331     আমাদের এপ্রোচ হচ্ছে , দুইজন লোক একইসাথে নতুন কলামে যেতে
পারবে। এবং একজন একজন করে রো
2332     পরিবর্তন করতে পারবে। ১ম ব্যক্তি সর্বদা ২য় ব্যক্তির থেকে নিচের কোন
রোতে থাকবে। এক্ষেত্রে এমন হতে পারে তারা ২য় কলামের ১ম এবং ২য় রোতে
তারা আছে। এখন ১ম ব্যক্তি ২য় রো থেকে একে একে ৫ম রোতে যাওয়ার পর

```

```

2300         /** Solution Print */
2301         Print(0,s.size()-1);
2302         reverse(v2.begin(), v2.end());
2303         for(i=0 ; i<v1.size() ; i++)
2304             cout<<v1[i];
2305         for(i=0 ; i<v2.size() ; i++)
2306             cout<<v2[i];
2307         cout<<endl;
2308         return 0;
2309     }
2310     /** DP.15 - Longest Palindromic Substring
Template
2311     We can find the longest palindrome
substring in (n^2) time with O(1) extra
space. The idea is to generate all even
length and odd length palindromes and keep
track of the longest palindrome seen so far.
2312     Step to generate odd length palindrome:
2313     Fix a centre and expand in both
directions for longer palindromes.
2314     Step to generate even length palindrome
2315     Fix two centre ( low and high ) and
expand in both directions for longer
palindromes. */
2316     void PrintSubstring(string str,int
low,int high){
2317         pf("Longest palindromic substring : ");
2318         for(int i=low ; i<=high ; i++)
2319             cout<<str[i];
2320         cout<<endl;
2321     }
2322     int LongestPalSubstr(string str){
2323         int
len=str.size(),start=0,low,high,mxlen=1;
2324         for(int i=1 ; i<len ; i++){
2325             /** Find longest even length
palindrome with center point i-1 and i */
2326             low=i-1; high=i;
2327             while(low>=0 && high<len &&
str[low]==str[high]){
2328                 if(high-low+1 > mxlen){
2329                     start=low;
2330                     mxlen=high-low+1;
2331                 }
2332                 --low; ++high;
2333             }
2334             /** Find longest even length
palindrome with center point i */
2335             low=i-1; high=i+1;
2336             while(low>=0 && high<len &&
str[low]==str[high]){
2337                 if(high-low+1 > mxlen){
2338                     start=low;
2339                     mxlen=high-low+1;
2340                 }
2341                 --low; ++high;
2342             }
2343             PrintSubstring(str,start,start+mxlen-1);
2344             return mxlen;
2345         }
2346     }
2347     int main()
2348     {
2349         int mx;
2350         string s1;
2351         cin>>s1;
2352         mx = LongestPalSubstr(s1);
2353         pf("Length of longest palindromic
substring is = %d\n",mx);
2354         return 0;
2355     }
2356     /** DP.16 - DP on grid (Baker Bro) */
2357     int m,n,grid[105][105],dp[105][105][105][2];
2358     /** Approach: এই প্রব্লেমে ধরে নেই যে দুইজন লোক গ্রিডের ১ম
সেল থেকে শেষ সেলে যাবে এবং পয়েন্টগুলো কালেক্ট করবে। মনে রাখতে
হবে, এই দুইজন লোকের
2359     রাস্তা শুধুমাত্র গ্রিডের ১ম এবং শেষ সেলেই মিলিত হতে পারবে। আর
উভয় ব্যক্তিই হয় নিচে নাহয় ডানে মুভ করতে পারবে। মানে রো এবং কলাম এর
মান শুধু বাড়তেই পারবে।
2360     আমাদের এপ্রোচ হচ্ছে , দুইজন লোক একইসাথে নতুন কলামে যেতে
পারবে। এবং একজন একজন করে রো
2361     পরিবর্তন করতে পারবে। ১ম ব্যক্তি সর্বদা ২য় ব্যক্তির থেকে নিচের কোন
রোতে থাকবে। এক্ষেত্রে এমন হতে পারে তারা ২য় কলামের ১ম এবং ২য় রোতে
তারা আছে। এখন ১ম ব্যক্তি ২য় রো থেকে একে একে ৫ম রোতে যাওয়ার পর
2362     ২য় ব্যক্তি ২য়,৩য় রো
ভিজিট করছে। মানে ১ম ব্যক্তির ভিজিট করা পথেই যাচ্ছে। যা অসম্ভব।
এজন্য একটা স্টেট রাখতে হবে যে বর্তমান কলামে ১ম ব্যক্তি ১ বারও
নিচে গেছে কি না? ১ম ব্যক্তি যদি নিচে যায়,তাহলে আমি আর জানি না সে এই
কলাম এর কোন কোন
সেল ভিজিট করেছে।সে, ২য় ব্যক্তি নিচে যেতে পারবে না এই কলাম
বরাবর।
আর যদি ১ম ব্যক্তি এখনো নিচে যায় নি এই কলাম বরাবর, তাহলে ২য়
ব্যক্তি প্রথম ব্যক্তির রো এর আগের রো পর্যন্ত নিচে যেতে পারবে ১ রো ১ রো করে।
সর্বশেষে, আমরা জানি যে গুরুত্ব এবং শেষের সেলের পয়েন্ট দুইবার
যোগ হয়েছে। তাই ১বার বিয়োগ করে দিতে হবে। স্টেটগুলো হবে:
State-1: বর্তমান কলাম বরাবর ১ম ব্যক্তি কি নিচের দিকে মুভ
করছে?
State-2: ১ম ব্যক্তি বর্তমানে কোন রোতে আছে?
State-3: ২য় ব্যক্তি বর্তমানে কোন রোতে আছে?
State-4: উভয়ে বর্তমানে কোন কলামে আছে?
বেইস কেস কি হবে নিজে চিন্তা করে বের করা. */
2363     int FuN(int row1,int row2,int col,int
p1_not_moved_in_this_col){
2364         if(row1!=m && col==n) return
-infinity;
2365         else if(row1==m && row2==m && col==n
&& p1_not_moved_in_this_col)
2366             return
grid[row1][col]+grid[row2][col];
2367         if(dp[row1][row2][col][p1_not_moved_in_this_col]
!= -1)
2368             return
dp[row1][row2][col][p1_not_moved_in_this_col];
2369         int ret1=0,ret2=0,ret3=0;
2370         if((col==n && row2<row1) || (col<n &&
row2+1<row1 && p1_not_moved_in_this_col &&
col>1))
2371             ret1 = grid[row2][col] +
FuN(row1,row2+1,col,p1_not_moved_in_this_col);
2372         if(row1<m)
2373             ret3 = grid[row1][col] +
FuN(row1+1,row2,col,0);
2374         if(col<n && row1!=row2)

```

```

2383         ret2 = grid[row1][col] +
grid[row2][col] + FuN(row1,row2,col+1,1);
2384         return
dp[row1][row2][col][p1_not_moved_in_this_col
] = max(ret1, max(ret2,ret3));
2385     }
2386     int main()
2387     {
2388         int i,j,t,T,ans;
2389         scin(T);
2390         RUN_CASE(t,T){
2391             scin2(m,n);
2392             for(i=1 ; i<=m ; i++){
2393                 for(j=1 ; j<=n ; j++){
2394                     scin(grid[i][j]);
2395                 }
2396                 ms(dp,-1); ans = FuN(1,1,1,1);
2397                 ans -= (grid[1][1]+grid[m][n]);
2398                 pf("Case %d: %d\n",t,ans);
2399             }
2400             return 0;
2401         }
2402         /** DP.17 - Problem: LOJ - 1092 - Lighted
Panels
2403         এই প্রবলেম এ আমাদের একটা লাইট প্যানেল এর অবস্থা দেওয়া হয়েছে,
কোন অবস্থার '*' এর মানে হচ্ছে লাইট জ্বলে আছে, '.' মানে হচ্ছে লাইট
নিভে আছে। আমাদেরকে মিনিমাম মুভে প্যানেল এর সবগুলো লাইট জ্বলাতে
হবে। এইখানে যখন কোন পয়েন্ট toggle করা হয় ( মানে এইটা যে
অবস্থায় আছে জ্বলা থাকলে নিভা, নিভা থাকলে জ্বলে উঠবে ) ঐ পয়েন্ট এর
সাথে adjacent যে পয়েন্টগুলো থাকে (diagonal সহ) তারাও
toggle করবে।
2404         row and column of the grid <= 8
2405         Solution:
2406         1. এই প্রবলেমটা যদি দেখি আমরা যখন কোন পয়েন্ট toggle করি
তাহলে কি হবে আমরা যে row তে আছি তার আগের row এবং পরের
row এর আমরা যে column এ আছি তার আগের column এবং পরের
column নিয়ে কাজ করব।
2407         2. আমি এখন যে row তে আসি তার কি অবস্থা ( মানে কোন কোন পয়েন্ট
জ্বলে আসে কি নিভে আসে ) যে row থেকে আসলাম তার কি অবস্থা ছিল
এবং যে row তে যাব তার কি অবস্থা আছে আমাদের তা জানা থাকা দরকার।
2408         3. State-1: number_of_row ; State-2:
bit_representation_of_previous_row ;

```

```

//Check all possible combination by
subset mask
2431         if(Cheek(i , j) == 0)
2432             continue; //No need to
toggle
2433         //Need to toggle
2434         cnt++;
2435         for(int k=0 ; k<3 ; k++){
2436             //Toggle the jth light of
prevrow,currow,nxtrow
2437             rowrep[k] ^= (1<<j);
2438             if(j+1 < col){ //Toggle the
(j+1)th light of prevrow,currow,nxtrow
2439                 for(int k=0 ; k<3 ; k++){
2440                     rowrep[k] ^=
(1<<(j+1));
2441                 }
2442                 if(j-1 >= 0){ //Toggle the
(j-1)th light of prevrow,currow,nxtrow
2443                     for(int k=0 ; k<3 ; k++){
2444                         rowrep[k] ^=
(1<<(j-1));
2445                     }
2446                 }
2447                 if(idx == 0) //IF its 1st
row,then go to 2nd row. Because,if any
light isn't on yet, we can on that light
from next row.
2448                 ret = min(ret , cnt+FuN(idx+1
, rowrep[2] , rowrep[1]));
2449                 else if(rowrep[0] ==
(1<<col)-1) //Only IF all the lights
are on in our previous row,then we can go
to next row
2450                 ret = min(ret , cnt+FuN(idx+1
, rowrep[2] , rowrep[1]));
2451             }
2452             vis[idx][currmask][prevmask] = 1;
2453             return dp[idx][currmask][prevmask] =
ret;
2454         }

```

```

2409         State-3: bit representation of current row
4. আমাদের ক্যালকুলেশন এর জন্য আমাদের পরের row ও দরকার হবে
এইটা আমরা কোথায় পাব। আমরা প্রথমেই ইনপুট নেওয়ার সময় একটা
array তে রেখে দিতে পারি কোন কোন লাইট এখন জ্বলে আসে।
2410         5. যে row তে আসি এর প্রতিটা combination করে আমরা
আমাদের store value গুলো change করে দেখব। এর জন্য আমরা
subset mask use করতে পারি। যেহেতু column এর
highest limit 8 .তাই (1<<8) == 256 খুব সহজেই আমরা
আমাদের dp এর ভিতর তা চালাতে পারি।
2411         6. যখন আমরা নেস্টেড row তে যাব আমাদের sure করতে হবে
আমাদের আগের row এর সবগুলো লাইট জ্বলানো আছে ( যদি না থাকে
তাহলে আর কোন ভাবেই ঐ লাইটকে আর জ্বলানো সম্ভব হবে না।) /**
2412         int Set(int N, int pos) {return N = N |
(1<<pos);}
2413         int Reset(int N, int pos) {return N = N
& ~(1<<pos);}
2414         bool Cheek(int N, int pos) {return
(bool)(N & (1<<pos));}
2415
2416         bool vis[10][(1<<8)+5][(1<<8)+5];
2417         int
row,col,rowcondition[10],dp[10][(1<<8)+5][(1
<<8)+5];
2418         char grid[10][10];
2419         int FuN(int idx,int currmask,int prevmask){
2420             if(idx >= row){
2421                 if(prevmask == (1<<col)-1) return 0;
2422                 else return infinity;
2423             }
2424             if(vis[idx][currmask][prevmask])
2425                 return dp[idx][currmask][prevmask];
2426             int ret = infinity;
2427             for(int i=0 ; i <= (1<<col)-1 ; i++){
2428                 int cnt = 0;
2429                 int rowrep[3] = {prevmask ,
currmask , rowcondition[idx+1]}; //
manipulated bit representation of previous
row , manipulated bit representation of
current row(manipulated from previous row)
, original bit representation of next row
**/
2430                 for(int j=0 ; j<col ; j++){

```

```

2455         int main(){
2456             int i,j,k,t,T,ans,mask;
2457             scin(T);
2458             RUN_CASE(t,T){
2459                 scin2(row , col);
2460                 for(i=0 ; i<row ; i++){
2461                     sc("%s",grid[i]);
2462                     mask = 0;
2463                     for(j=0 ; j<col ; j++){
2464                         if(grid[i][j] == '*')
2465                             mask = Set(mask , j);
2466                     }
2467                     rowcondition[i] = mask;
2468                 }
2469                 ms(vis , 0); ans = FuN(0 ,
rowcondition[0] , 0);
2470                 if(ans < infinity)
2471                     pf("Case %d: %d\n",t,ans);
2472                 else
2473                     pf("Case %d: impossible\n",t);
2474             }
2475             return 0;
2476         }
2477         /** NT.01 - Segmented Sieve Template */
2478         vector<int>Prime;
2479         bool mark[10000009];
2480         void sieve(int n){
2481             u know it well--
2482         }
2483         ll segmentedSieve(ll L,ll R){
2484             ll cnt=0;
2485             bool isPrime[R-L+2];
2486             for(int i=0 ; i<=R-L+1 ; i++){
2487                 isPrime[i]=true; //
Initially mark all as prime **/
2488             if(L == 1)
2489                 isPrime[0]=false;
2490             for(int i=0 ; i<Prime.size() &&
Prime[i]*Prime[i]<=R ; i++){
2491                 ll cutPrime=Prime[i];
2492                 ll base=cutPrime*cutPrime;
2493                 if(base < L){

```



```

2494         base =
((L+cutPrime-1)/cutPrime)*cutPrime; /**
Increase base if it already cut by
normal sieve **/
2495     }
2496     for(ll j=base ; j<=R ; j+=cutPrime)
2497         isPrime[j-L]=false; /**
Mark the index as composite **/
2498     }
2499     for(int i=0 ; i<=R-L ; i++){
2500         if(isPrime[i] == true){
2501             // cout<<L+i<<endl;
2502             /**Print the Prime Numbers **/
2503             cnt += 1;
2504         }
2505     }
2506     return cnt;
2507 }
2508 int main(){
2509     sieve(10000000); segmentedSieve(1e9
, 1e9+1e5);
2510     return 0;
2511 }
2512 /** NT.02 - Large BigMOD **/
2513 ll BigMul(ll a,ll b,ll m){
2514     if(b == 0) return 0;
2515     if(b == 1) return a%m;
2516     if(b%2 == 0){
2517         ll temp=b/2;
2518         ll res=BigMul(a,temp,m);
2519         return ((res%m)+(res%m))%m;
2520     }
2521     else{
2522         ll temp=BigMul(a,b-1,m);
2523         return ((a%m)+(temp%m))%m;
2524     }
2525 }
2526 ll BigMod(ll a,ll b,ll m){
2527     if(b == 0) return 1%m;
2528     if(b%2 == 0){
2529         ll temp=BigMod(a,b/2,m);
2530         return BigMul(temp,temp,m)%m;

```

```

2568 void getFactor(int n){
2569     Factor.clear();
2570     while(n != 1){
2571         Factor.push_back(SF[n]);
2572         n/=SF[n];
2573     }
2574 }
2575 /** NT.05 - nCr in nlogn **/
2576 /** nCr in nlogn complexity **/
2577 int SF[1000005];
2578 void Smallest_Factor(){
2579     U know me--
2580 }
2581 ll Power(ll a,ll n){
2582     ll ret=1;
2583     for(ll i=1 ; i<=n ; i++)
2584         ret = ((ret%MOD)*(a%MOD))%MOD;
2585     return ret;
2586 }
2587 map<int,int>nom,denom;
2588 map<int,int>::iterator it;
2589 void FactorizeNom(int n){
2590     while(n != 1){
2591         nom[SF[n]]++;
2592         n /= SF[n];
2593     }
2594 }
2595 void FactorizeDenom(int n){
2596     while(n != 1){
2597         denom[SF[n]]++;
2598         n /= SF[n];
2599     }
2600 }
2601 ll nCr(ll n,ll r)
2602 {
2603     nom.clear();
2604     denom.clear();
2605     ll mn=min(r, n-r), mx=max(r, n-r), i,
temp , ans=1 , x , pow;
2606     for(i=mx+1 ; i<=n ; i++)
2607         FactorizeNom(i);
2608     for(i=2 ; i<=mn ; i++)

```

```

2531     else{
2532         ll temp=BigMod(a,b-1,m);
2533         return BigMul(temp,a,m)%m;
2534     }
2535 int main()
2536 {
2537     cout<<BigMod(a , b , m)<<endl;
2538     return 0;
2539 }
2540 /** NT.03 - SievePhi **/
2541 int phi[1000006], mark[1000006];
2542 void sievephi(int n){
2543     int i,j;
2544     for(int i=1;i<=n;i++) phi[i] = i;
2545     phi[1] = 1; mark[1] = 1;
2546     for(int i=2;i<=n;i++){
2547         if(!mark[i]){
2548             for(j=i;j<=n;j+=i){
2549                 mark[j] = 1;
2550                 phi[j] = phi[j] / i *(i-1);
2551             }
2552         }
2553     }
2554 }
2555 /** NT.04 - Prime Factorization in nlog(n)
**/
2556 int SF[1000005]; /** Here we store
smallest factor for each number **/
2557 void Fun(){
2558     for( int i = 2 ; i * i <= 10000000 ;
i ++ ){
2559         if( !SF[i] ){
2560             for( int j = i ; j <=
10000000 ; j += i ){
2561                 if( !SF[j] ) SF[j] = i ;
2562             }
2563         }
2564     }
2565     for( int i = 2 ; i <= 1e7 ; i ++ )
if( !SF[i] ) SF[i] = i ;
2566 }
2567 vector<int>Factor;

```

```

2608     FactorizeDenom(i);
2609     for(it=nom.begin() ; it!=nom.end() ;
it++){
2610         x = it->first; pow =
nom[x]-denom[x];
2611         temp = Power(x , pow); ans *= temp;
2612     }
2613     return ans;
2614 }
2615 int main()
2616 {
2617     Smallest_Factor();
2618     cout<<nCr(10 , 1)<<endl;
2619     return 0;
2620 }
2621 /** Z-algorithm Template **/
2622 vi z_fun(string s){
2623     int i , l , r , n = s.size();
2624     vi z(n);
2625     for(i=1,l=0,r=0 ; i<n ; i++){
2626         if(i <= r)
2627             z[i] = min(r-i+1 , z[i-l]);
2628         while(i+z[i]<n &&
s[z[i]]==s[i+z[i]])
2629             ++z[i];
2630         if(i+z[i]-1 > r){
2631             l=i;
2632             r=i+z[i]-1;
2633         }
2634     }
2635     return z;
2636 }
2637 int main()
2638 {
2639     int i,j,n;
2640     string s;
2641     cin>>s;
2642     vi Z = z_fun(s);
2643     for(i=0;i<Z.size();i++)
2644         pf("%d ",Z[i]);
2645     pf("\n");
2646     return 0;

```

```

2647     }
2648     /** String Hashing Problem:
2649     You are given a string M and N other
    strings smaller in length than M. You have
    to find whether each of these N strings is
    a substring of M. All strings consist of
    only alphanumeric characters. */
2650     #define szz 100005
2651     string s,sub;    const int p=331;
2652     ll p_pow[szz],inv[szz],hash_sum[szz];
2653     void powER(){
2654         p_pow[0] = 1;
2655         for(ll i=1 ; i<szz ; i++)
2656             p_pow[i] = (p_pow[i-1]*p)%MOD;
2657     }
2658     void modINV(){
2659         inv[0] = 1;
2660         inv[1] = BigMod(p , MOD-2);
2661         for(ll i=2 ; i<szz ; i++)
2662             inv[i] = (inv[i-1]*inv[1])%MOD;
2663     }
2664     void Hash_Table(){
2665         ll len = (ll)s.size();
2666         for(ll i=0 ; i<len ; i++)
2667             hash_sum[i+1] = (hash_sum[i] +
2668             (s[i]-'0'+1)*p_pow[i])%MOD;
2669     }
2670     ll compute_hash(){
2671         ll hash_value=0LL , len =
2672         (ll)sub.size();
2673         for(ll i=0 ; i<len ; i++){
2674             hash_value = (hash_value +
2675             (sub[i]-'0'+1)*p_pow[i])%MOD;
2676         }
2677         return hash_value;
2678     }
2679     map<ll,ll>st[2005];
2680     void subStringStore(ll len){
2681         ll lb,ub,val,sz = (ll)s.size();
2682         for(ll i=0 ; i<=sz-len ; i++){
2683             lb = i+1;
2684             ub = i+len;

```

```

2722         out.clear(); vis = false; cnt = 0;
2723     }
2724     ~NODE(){
2725         for(int i = 1; i < goes; i++)
2726             if(next[i]!=NULL && next[i]
2727             != this)
2728                 delete next[i];
2729     };
2730     NODE *root;
2731     void buildtrie(){
2732         root = new NODE();
2733         for(int i = 0; i < n; i++){
2734             NODE *p = root;
2735             for(int j = 0; dictionary[i][j];
2736             j++){
2737                 char c = dictionary[i][j]; ///
2738                 - 'a' + 1; ///uncomment it for only
2739                 lowercase letters
2740                 if(!p->next[c])
2741                     p->next[c] = new NODE();
2742                 p = p->next[c];
2743             }
2744             queue<NODE *> q;
2745             for(int i = 0; i < goes; i++){
2746                 if(!root->next[i]) root->next[i]
2747                 = root;
2748                 else{
2749                     q.push(root->next[i]);
2750                     root->next[i]->next[0] =
2751                     root; /// next[0] is back pointer
2752                 }
2753                 while(!q.empty()){
2754                     NODE *u = q.front(); q.pop();
2755                     for(int i = 1; i < goes; i++){
2756                         if(u->next[i]){
2757                             NODE *v = u->next[i];
2758                             NODE *w = u->next[0];
2759                             while(!w->next[i])
2760                                 w = w->next[0];

```

```

2682         val =
2683         (hash_sum[ub]-hash_sum[lb-1]+MOD)%MOD;
2684         val = (val*inv[i])%MOD;
2685         st[len][val]++;
2686     }
2687 }
2688 bool vis[2005];
2689 int main(){
2690     ll i,j,k,n,val,len,sz;
2691     powER();
2692     modINV();
2693     cin>>s;
2694     sz = (ll)s.size();
2695     Hash_Table();
2696     scln(n);
2697     rep(i,1,n){
2698         cin>>sub;
2699         len = (ll)sub.size();
2700         val = compute_hash();
2701         if(!vis[len]){
2702             subStringStore(len);
2703             vis[len] = 1;
2704         }
2705         if(st[len][val] > 0)
2706             pf("Y\n");
2707         else
2708             pf("N\n");
2709     }
2710     return 0;
2711 }
2712 /** Aho Corasick Template - SPOJ-SUB_PROB */
2713 const int MX = 2005;
2714 int n; /// n is the number of queries
2715 char text[1000006],dictionary[MX][MX];
2716 const int goes = 127; /// If MLE comes,
2717 set it to 27 for only lowercase letters
2718 struct NODE{
2719     int cnt; bool vis; NODE *next[goes];
2720     vector<NODE *> out;
2721     NODE(){
2722         for(int i = 0; i < goes; i++)
2723             next[i] = NULL;
2724     }
2725     v->next[0] = w =
2726     w->next[i];
2727     w->out.pb(v); q.push(v);
2728     }
2729     }
2730 }
2731 void aho_corasick(){
2732     NODE *p = root;
2733     for(int i = 0; text[i]; i++){
2734         char c = text[i]; /// - 'a' + 1;
2735         ///uncomment it for only lowercase letters
2736         while(!p->next[c])
2737             p = p->next[0];
2738         p = p->next[c];
2739         p->cnt++;
2740     }
2741 }
2742 int DFS(NODE *p){
2743     if(p->vis) return p->cnt;
2744     for(int i = 0; i <
2745     (int)p->out.size(); i++)
2746         p->cnt += DFS(p->out[i]);
2747     p->vis = true;
2748     return p->cnt;
2749 }
2750 int main(){
2751     sc("%s",text); scin(n);
2752     for(int i = 0; i < n; i++){
2753         sc("%s",dictionary[i]);
2754     }
2755     buildtrie(); aho_corasick();
2756     for(int i = 0; i < n; i++){
2757         NODE *p = root;
2758         for(int j = 0; dictionary[i][j];
2759         j++){
2760             char c = dictionary[i][j];
2761             p = p->next[c];
2762         }
2763         int x = DFS(p);
2764         if(x==0) pf("N\n");
2765         else pf("Y\n");

```

```

2795     }
2796     delete root;
2797     return 0;
2798 }
2799 /** Ternary Search Template */
2800 double ternary_search(double l, double r) {
2801     double eps = 1e-9; ///set the error
2802     limit here
2803     while (r - l > eps) {
2804         double m1 = l + (r - l) / 3;
2805         double m2 = r - (r - l) / 3;
2806         double f1 = f(m1); ///evaluates
2807         the function at m1
2808         double f2 = f(m2); ///evaluates
2809         the function at m2
2810         if (f1 < f2)
2811             l = m1;
2812         else
2813             r = m2;
2814     }
2815     return f(l); ///return the maximum
2816 of f(x) in [l, r]
2817 }
2818 /** Binary Search + Meet In the Middle +
2819 Backtracking */
2820 /** For each coin we've three coins. We
2821 can take this coin for 0,1 or 2 times */
2822 /** So,we'll divide the coins into two
2823 portion & generating all possible
2824 permutation of both portion */
2825 /** Then,sort 2nd portion B. And search
2826 for every x-A[i] in the 2nd portion. If
2827 yes the ans will be yes or No otherwise */
2828 int coin[20]; vi A;
2829 map<int,int>B; int n;
2830 void SubSet(int idx,int val){
2831     if(idx == (n/2)){
2832         B[val]++;
2833         return;
2834     }
2835     SubSet(idx+1,val);
2836     SubSet(idx+1,val+coin[idx]);

```

```

2827     SubSet(idx+1,val+(2*coin[idx]));
2828 }
2829 void SubSet2(int idx,int val){
2830     if(idx == n){
2831         A.pb(val);
2832         return;
2833     }
2834     SubSet2(idx+1,val);
2835     SubSet2(idx+1,val+coin[idx]);
2836     SubSet2(idx+1,val+(2*coin[idx]));
2837 }
2838 int main(){
2839     int i,j,k,t,T,mx,temp,cnt;
2840     scin(T);
2841     RUN_CASE(t,T){
2842         A.clear(); B.clear(); scin2(n,k);
2843         for(i=0; i<n; i++) scin(coin[i]);
2844         SubSet(0,0); SubSet2(n/2,0);
2845         bool fg=0;
2846         for(i=0; i<A.size() && !fg; i++){
2847             temp = k-A[i];
2848             if(B[temp] > 0) fg=1;
2849         }
2850         if(fg)
2851             pf("Case %d: Yes\n",t);
2852         else
2853             pf("Case %d: No\n",t);
2854     }
2855     return 0;
2856 }
2857 /** Matrix Exponential Template */
2858 struct Matrix{
2859     ll v[5][5]; /** Sizes may vary */ ll
2860     row, col;
2861     Matrix operator = (const Matrix X){
2862         row = X.row; col = X.col;
2863         for(ll i = 0; i < row; i++){
2864             for(ll j = 0; j < col; j++){
2865                 v[i][j] = X.v[i][j];
2866             }
2867         }
2868     }; ll mod;

```

```

2868 Matrix multiply(const Matrix &A, const
2869 Matrix &B){
2870     assert(A.col==B.row);
2871     Matrix R;
2872     R.row = A.row; R.col = B.col;
2873     for(ll i = 0; i < R.row; i++){
2874         for(ll j = 0; j < R.col; j++){
2875             ll sum = 0;
2876             for(ll k = 0; k < A.col; k++){
2877                 sum += (A.v[i][k] *
2878                     B.v[k][j]);
2879             }
2880             sum %= mod;
2881             R.v[i][j] = sum;
2882         }
2883     }
2884     return R;
2885 }
2886 Matrix power(const Matrix &M, ll p){
2887     assert(p >= 1);
2888     if(p == 1) return M;
2889     if(p%2 == 1) return multiply(M,
2890 power(M, p-1));
2891     Matrix ret = power(M, p / 2);
2892     Matrix bet = multiply(ret,ret);
2893     return bet;
2894 }
2895 int main(){
2896     int t,T;
2897     Matrix base;
2898     base.row = 2LL; base.col = 2LL;
2899     base.v[0][0] = base.v[0][1] =
2900     base.v[1][0] = 1LL;
2901     base.v[1][1] = 0; scin(T);
2902     RUN_CASE(t,T){
2903         ll a,b,n,m;
2904         scln(a); scln(b); scln(n); scln(m);
2905         mod = POW(10LL,m);
2906         Matrix MAT, Res, M;
2907         MAT.row = 2LL; MAT.col = 1LL;
2908         MAT.v[0][0] = b; MAT.v[1][0] = a;
2909         M = power(base, n - 1); Res =

```

```

2906 multiply(M, MAT);
2907     CASE(t); pf("%lld\n",Res.v[0][0]);
2908 }
2909 return 0;
2910 }
2911 /** Solve of a given mathematical
2912 expression */
2913 map<char, int> M;
2914 int main(){
2915     M['-'] = 1; M['+'] = 1; M['*'] = 3;
2916     M['/'] = 4; M['#'] = 0;
2917     int t,T; scin(T);
2918     RUN_CASE(t,T){
2919         string str,tmp; cin>>str;
2920         str+="#"; tmp = "";
2921         stack<char> chr; stack<ll> num;
2922         str = "0" + str;
2923         stringstream ss;
2924         int n = (int)str.size();
2925         for(int i = 0; i < n; i++){
2926             if(str[i]=='+'||str[i]=='-'||str[i]=='*'||st
2927             r[i]=='/'||str[i]=='#'){
2928                 ss<<tmp; ll x; ss>>x; tmp
2929                 = ""; ss.clear();
2930                 if(chr.empty() ||
2931                 M[str[i]] > M[chr.top()]){
2932                     num.push(x);
2933                     chr.push((char)str[i]);
2934                 }
2935                 else{
2936                     while(!chr.empty() &&
2937                     M[str[i]] <= M[chr.top()]){
2938                         ll y = num.top();
2939                         int z = M[chr.top()];
2940                         char ch = chr.top();
2941                         num.pop();
2942                         chr.pop();
2943                         if(ch=='-') y = y-x;
2944                         else if(ch=='+') y
2945                         = y+x;
2946                         else if(ch=='*') y

```

```

2936     = x * y;
2937     else if(ch=='/') y
2938         = y/x;
2939         x = y;
2940         }
2941         num.push(x);
2942         chr.push((char)str[i]);
2943         }
2944         else tmp += ((char)str[i]);
2945         }
2946         CASE(t); printf("%lld\n", num.top());
2947         return 0;
2948         /** 10+2*3,100*2+500/10,10+20/2-5 */
2949         /** MOD INVERSE Template */
2950         #define x first
2951         #define y second
2952         pii egcd(int a,int b)
2953         {
2954             if(b == 0) return pii(1,0);
2955             else{
2956                 pii d = egcd(b , a%b);
2957                 return pii(d.y , d.x-d.y*(a/b));
2958             }
2959         }
2960         int modInv(int a,int b){
2961             pii ret = egcd()
2962             return ;
2963         }
2964         /**COIN CHANGE 1 */
2965         /**In a strange shop there are n types of
2966         coins of value A1, A2 ... An. C1, C2, ...
2967         Cn denote the number of coins of value A1,
2968         A2 ... An respectively. You have to find
2969         the number of ways you can make K using
2970         the coins.**/
2971         int main()
2972         {
2973             int n,current_limit, ans,coin[105],
2974             t, T, k, limit[105],tmp,tmp1;

```

```

3000             scin(coin[i]);
3001             for(int j = coin[i]; j <= k ;
3002             j++){
3003                 if(coin[i]<=j){
3004                     tmp = combination[ j
3005                     - coin[i] ]%MOD;
3006                     tmp1 = combination[j]
3007                     + tmp;
3008                     combination[j]
3009                     =tmp1%MOD;
3010                 }}}
3011             CASE(t);
3012             pf("%lld\n", combination[k]);
3013             ms(combination,0);
3014             }return 0;
3015             /**COIN CHANGE 3 */
3016             /**In a strange shop there are n types of
3017             coins of value A1, A2 ... An. C1, C2, ...
3018             Cn denote the number of coins of value A1,
3019             A2 ... An respectively. You have to find
3020             the number of different values (from 1 to
3021             m), which can be produced using these
3022             coins**/
3023             int main()
3024             {
3025                 int n,current_limit, ans;
3026                 int coin[105],cnt[100005];
3027                 int t, T, k, limit[105],tmp,tmp1;
3028                 scin(T);int combination[100005];
3029                 RUN_CASE(t,T){
3030                     ms(coin,0); ms(limit,0);
3031                     scin(n); scin(k);
3032                     for(int i=0; i<n; i++)
3033                     scin(coin[i]);
3034                     for(int i=0; i<n; i++)
3035                     scin(limit[i]);
3036                     CASE(t);
3037                     combination[0] = 1; ans = 0;
3038                     for(int i=0; i<n; i++){
3039                         current_limit = limit[i];
3040                         ms(cnt,0);

```

```

2969         scin(T);int combination[1005];
2970         RUN_CASE(t,T){
2971             ms(coin,0); ms(limit,0);
2972             scin(n); scin(k);
2973             for(int i=0; i<n; i++)
2974             scin(coin[i]);
2975             for(int i=0; i<n; i++)
2976             scin(limit[i]);
2977             CASE(t);
2978             ms(combination,0); combination[0]
2979             = 1;
2980             for(int i=0; i<n; i++){
2981                 current_limit = limit[i];
2982                 for(int j = k; j >= 0; j--){
2983                     for(int x = 1; x <=
2984                     current_limit; x++){
2985                         if((j-x*coin[i])>=0)
2986                         combination[j]+= combination[ j -
2987                         (x*coin[i])];
2988                     }
2989                     for(int j=0;j<=k;j++)
2990                     combination[j]%MOD;
2991                 }
2992                 pf("%d\n",combination[k]);
2993                 }return 0;
2994             /**COIN CHANGE 2 */
2995             /**In a strange shop there are n types of
2996             coins of value A1, A2 ... An. You have to
2997             find the number of ways you can make K
2998             using the coins. You can use any coin at
2999             most K times.**/
3000             ll combination[10002] ,tmp, tmp1;
3001             int main()
3002             {
3003                 int t, T, n, k, coin[102];
3004                 scin(T);
3005                 RUN_CASE(t,T){
3006                     scin(n);scin(k);
3007                     combination[0] = 1;
3008                     for(int i = 0; i < n; i++){

```

```

3029             for(int j = coin[i]; j <= k;
3030             j++){
3031                 if(cnt[j-coin[i]<current_limit &&
3032                 combination[j]==0 &&
3033                 combination[j-coin[i]]){
3034                     ans++;
3035                     cnt[j] =
3036                     cnt[j-coin[i]]+1;
3037                     combination[j] = 1;
3038                 }}}
3039                 pf("%d\n",ans);
3040                 ms(combination,0);
3041                 }return 0;
3042             /**nCr using inverse mod */
3043             /** x must be
3044             relatively prime to p
3045             const int Maxn = 1e5 + 1;
3046             template<typename T, typename T1>
3047             T mod(T x, T1 p){
3048                 x %= p;
3049                 if (x < 0) x += p;
3050                 return x;
3051             }
3052             int fact[Maxn], inv[Maxn], ifact[Maxn];
3053             void factorial(){
3054                 fact[0] = 1;
3055                 for(int i = 1; i < Maxn; i++){
3056                     fact[i] = 1LL * fact[i - 1] * i %
3057                     MOD;
3058                 }
3059                 template<typename T>
3060                 T inverse(T x, T p){
3061                     x = mod(x, p);
3062                     if (x == 1) return x;
3063                     return mod((1LL * (-p / x) * (inv[p %
3064                     x] % p)), p);
3065                     /** Since inverse of p % x is already
3066                     calculated.
3067                 }
3068                 void inverse_fact(){

```

```

3062     ifact[0] = 1;
3063     for(int i = 1; i < Maxn; i++){
3064         inv[i] = inverse(i, MOD);
3065         ifact[i] = (1LL * ifact[i - 1] *
3066             inv[i]) % MOD;
3067     }
3068     int nCr(int n, int r){
3069         int ret = (1LL * ifact[n - r] *
3070             ifact[r]) % MOD;
3071         ret = (1LL * ret * fact[n]) % MOD;
3072         return ret;
3073     }
3074     int main(){
3075         factorial(); inverse_fact();
3076         while(t--){
3077             cin >> n >> r;
3078             cout << nCr(n, r) << endl;
3079         }
3080         /** Default Template **/
3081         #define Input
3082         freopen("in.txt", "r", stdin)
3083         #define Output
3084         freopen("out.txt", "w", stdout)
3085         #define ll long long int
3086         #define ull unsigned long
3087         #define pii pair<int, int>
3088         #define pll pair<ll, ll>
3089         #define sc scanf
3090         #define scin(x) sc("%d", &(x))
3091         #define scin2(x, y) sc("%d
3092             %d", &(x), &(y))
3093         #define scln(x) sc("%lld", &(x))
3094         #define scln2(x, y) sc("%lld
3095             %lld", &(x), &(y))
3096         #define pf printf
3097         #define all(a) (a.begin(), a.end())
3098         #define UNIQUE(X)
3099             (X).erase(unique(all(X)), (X).end())
3100
3101         {-2, -2, -1, -1, +1, +1, +2, +2}; //Knight's move
3102         ///const int fx[] =
3103         {-1, +1, -2, +2, -2, +2, -1, +1}; //Knight's move
3104
3105

```

```

3106     #define SORT_UNIQUE(c)
3107         (sort(c.begin(), c.end()),
3108         c.resize(distance(c.begin(), unique(c.begin()
3109             , c.end()))))
3110     #define ms(a, b)
3111         memset(a, b, sizeof(a))
3112     #define pb(a) push_back(a)
3113     #define mp make_pair
3114     #define db double
3115     #define ff first
3116     #define ss second
3117     #define sqr(x) (x)*(x)
3118     #define vi vector<int>
3119     #define vl vector<ll>
3120     #define CIN
3121         ios_base::sync_with_stdio(0); cin.tie(0); cout
3122             .tie(0)
3123     #define RUN_CASE(t, T) for(__typeof(t)
3124         t=1; t<=T; t++)
3125     #define CASE(t) printf("Case %d:
3126         ", t)
3127     #define CASE1(t) printf("Case
3128         %d:\n", t)
3129     #define intlimit 2147483690
3130     #define longlimit 92233720368547758
3131         (1<<28)
3132     #define gcd(a, b) __gcd(a, b)
3133     #define lcm(a, b) ((a)*(b))/gcd(a, b)
3134     #define mx 123456789
3135     #define PI 2*acos(0.0)
3136     #define rep(i, a, b) for(__typeof(i)
3137         i=a; i<=b; i++)
3138     using namespace std;
3139     ///-----Graph
3140     Moves-----
3141     ///const int fx[] = {+1, -1, +0, +0};
3142     ///const int fy[] = {+0, +0, +1, -1};
3143     ///const int fx[] =
3144         {+0, +0, +1, -1, -1, +1, -1, +1}; //King's move
3145     ///const int fy[] =
3146         {-1, +1, +0, +0, +1, +1, -1, -1}; //King's move
3147     ///const int fx[] =

```