Contest Notebook
of
MIST BEGINNERS

Table of Contents :

2-Sat :

```
//  1- based.....
struct two_sat{
int n,nn,m;    vector<int>G[maxm],GT[maxm],DAG[maxm],C[maxm],topo; // G= graph.. GT= transpose
graph.......
    int col[maxm],comp,comp_no[maxm],soln[maxm];
    // comp= total number component;
```

```
 // comp_no[i]= component no of node i
// soln[i]= truth symbol of node i;
two_sat(){}
void init(){
    for(int i=0;i<=n+n;i++){
        G[i].clear();
        GT[i].clear();
        DAG[i].clear();
                       C[i].clear();
    }
    topo.clear();
    memset(col,0,sizeof(col));
    memset(comp_no,0,sizeof(comp_no));
    memset(soln,-1,sizeof(soln));
              comp=0;
}
int inv(int no){
    if(no<=n) return no+n;
    return no-n;
}
void OR(int u,int v){
    G[inv(v)].push_back(u);
    G[inv(u)].push_back(v);
}
void AND(int u,int v){
    G[u].push_back(v);
    G[v].push_back(u);
}
void XOR(int u,int v){
    G[inv(v)].push_back(u);
    G[u].push_back(inv(v));
    G[inv(u)].push_back(v);
    G[v].push_back(inv(u));
}
void XNOR(int u,int v){
    G[u].push_back(v);
    G[v].push_back(u);
    G[inv(u)].push_back(inv(v));
    G[inv(v)].push_back(inv(u));
}
// problem  Dependent.......
void build_graph(){
    int u,v,op;
    nn=n+n;
    for(int i=0;i<m;i++){
        scanf("%d %d %d",&u,&v,&op);
        if(op==1) XNOR(u,v);
        else if(op==0) XOR(u,v);
    }
}
void make_reverse(){
    for(int i=1;i<=nn;i++){
        for(int j=0;j<G[i].size();j++){
            GT[G[i][j]].push_back(i);
        }
    }
}
int check_solution(){
    // Build  scc.........
    build_scc();
    for(int i=1;i<=n;i++){
        if(comp_no[i]==comp_no[inv(i)]) return 0;
```

```
         }
        return 1;
    }
    void find_solution(vector<int>&res){
        int i,j,i_p;
        for(i=1;i<=comp;i++){
            if(soln[i]==-1){
                soln[i]=0;
                i_p=comp_no[inv(C[i][0])];
                soln[i_p]=1;
                for(j=0;j<C[i_p].size();j++){
                    if(C[i_p][j]<=n) res.push_back(C[i_p][j]);
                }
            }
        }
    }

    void build_dag(){
        int i,j;
        for(i=1;i<=nn;i++){
            for(j=0;j<G[i].size();j++){
                if(comp_no[i]==comp_no[G[i][j]]) continue;
                DAG[comp_no[i]].push_back(comp_no[G[i][j]]);
            }
        }
    }
    void build_scc(){
        make_reverse();
        int i;
        for(i=1;i<=nn;i++){
            if(!col[i]) dfs(i);
        }
        for(i=topo.size()-1;i>=0;i--){
            if(!comp_no[topo[i]]){
                scc(topo[i],++comp);
            }
        }
    }
         void dfs(int s){
                  if(col[s]) return ;
                  col[s]=1;

                  for(int i=0;i<G[s].size();i++){
                           dfs(G[s][i]);
                  }
                  topo.push_back(s);
         }
         void scc(int s,int comp){
             if(comp_no[s]) return ;
             comp_no[s]=comp;
                  C[comp].push_back(s);

             for(int i=0;i<GT[s].size();i++){
                 scc(GT[s][i],comp);
             }
         }
};
two_sat T_sat;
vector<int>res;
int main(){
    int n,m;
    while(scanf("%d",&n)==1){
```

```
        scanf("%d",&m);
        T_sat.n=n; T_sat.m=m;
        T_sat.init();
        T_sat.build_graph();
        int ans=T_sat.check_solution();
        if(ans){
            res.clear();
            T_sat.find_solution(res);
            printf("%d\n",res.size());
            for(i=0;i<res.size();i++){
                if(i) printf(" ");
                printf("%d",res[i]);
            }
            puts("");
        }
        else{
            printf("Impossible\n");
        }
    }
    return 0;
}
```

Biconnected Component :

```
stack<pii>st_pii;
set<int>sets[maxm];
void bi_comp(int u,int v){
    while(!st_pii.empty()){
        pii now=st_pii.top(); st_pii.pop();
        sets[tot].insert(now.uu);
        sets[tot].insert(now.vv);

        if(now.uu==u && now.vv==v) break;
        if(now.uu==v && now.vv==u) break;
    }
    tot++;
}
void dfs(int s,int pre,int root){

    if(vis[s]) return;
    vis[s]=1;
    low[s]=dep[s]=tim++;
    // bi-connected with a single vertex
    if(G[s].size()==0){
        sets[tot++].insert(s);
        return ;
    }

    int i,j,k,c=0;
    for(i=0;i<G[s].size();i++){
        int d=G[s][i];
        if(d==pre) continue;
        if(vis[d] && dep[d]<dep[s]){
            st_pii.push(mp(s,d));
            low[s]=mini(low[s],dep[d]);
        }
        else if(!vis[d]){
            st_pii.push(mp(s,d));

            dfs(d,s,root); c++;
```

```
            if(low[d]>=dep[s]){
               bi_comp(s,d);
               if(s!=root){
                   is_cut[s]=1;
               }
           }
           low[s]=mini(low[s],low[d]);
       }
   }

   if(s==root && c>1){
       is_cut[s]=1;
   }
}
```

Floyd Warshall:

```
/*
w : edge weights
d : distance matrix
p : predecessor matrix
w[i][j] = length of direct edge between i and j
d[i][j] = length of shortest path between i and j
p[i][j] = on a shortest path from i to j, p[i][j] is the last node before j.
*/
// Initialization …..
. .. . . . . . ..
// Algorithm

for (k=0;k<n;k++) /* k -> is the intermediate point */
for (i=0;i<n;i++) /* start from i */
for (j=0;j<n;j++) /* reaching j */
     /* if i-->k + k-->j is smaller than the original i-->j */
     if (d[i][k] + d[k][j] < d[i][j]) {
          /* then reduce i-->j distance to the smaller one i->k->j */
          d[i][j] = d[i][k]+ d[k][j];
          /* and update the predecessor matrix */
          p[i][j] = p[k][j];
     }

void print_path (int i, int j) {
     if (i!=j) print_path(i,p[i][j]);
     print(j);
}
```

Bellman Ford :

```
struct edge{
       int u, v,cost;};
```

```
 edge edges[maxe]; int d[maxm],flag[maxm];
void bellman(int s,int n,int e){
        int i,j,k,l,u,v;
        for(i=1;i<=n;i++){
                flag[i]=0;
                d[i]=inf;
        }
        d[s]=0;
        for(i=1;i<=n+5;i++){
                 for(j=0;j<e;j++){
                         u=edges[j].u;
                         v=edges[j].v;
                         if(d[v]>d[u]+edges[j].cost){
                                 d[v]=d[u]+edges[j].cost;
                if(i>n){
                    // negative cycle .....
                    flag[v]=1; //  node v is in negative cycle
                }}}}
```

Shortest Path on A DAG :

```
Void relax(Node u,Node v,double w[][]){
      if d[v] > d[u] + w[u,v] then
      d[v]:=d[u] + w[u,v] // d= distance from source.
      pi[v]:=u     // pi[i]= parent of i'th node .
}
Void DAG_SHORTEST_PATHS(Graph G,double w[][],Node s){
      topologically sort the vertices of G  // O(V+E)
      initialize_single_source(G,s)
      for each vertex u taken in topologically sorted order
      for each vertex v which is Adjacent with u
relax(u,v,w)
}
```

Minimum Spanning Tree :

```
struct edge{
        int u,v,w;
};
edge edges[maxe]; int pre[maxm];
bool comp(edge a,edge b){
        return a.w>b.w;
}
int find(int x){
        if(pre[x]==x) return x;
        else return pre[x]=find(pre[x]);
}
        sort(edges,edges+m,comp);
        int sum=0;
        for(i=0;i<m;i++){
            k=find(edges[i].u); l=find(edges[i].v);
            if(k==l) continue;
            sum+=edges[i].w;
        }
        printf("%d\n",sum);
```

Euler Curcuit Print :

```
// A C++ program print Eulerian Trail in a given Eulerian or Semi-Eulerian Graph
// A class that represents an undirected graph
class Graph
{
  int V;    // No. of vertices
  list<int> *adj;    // A dynamic array of adjacency lists
public:
    // Constructor and destructor
  Graph(int V)  { this->V = V;  adj = new list<int>[V];  }
  ~Graph()      { delete [] adj;  }

  // functions to add and remove edge
  void addEdge(int u, int v) {  adj[u].push_back(v); adj[v].push_back(u); }
  void rmvEdge(int u, int v);

  // Methods to print Eulerian tour
  void printEulerTour();
  void printEulerUtil(int s);

  // This function returns count of vertices reachable from v. It does DFS
  int DFSCount(int v, bool visited[]);

  // Utility function to check if edge u-v is a valid next edge in
  // Eulerian trail or circuit
  bool isValidNextEdge(int u, int v);
};

/* The main function that print Eulerian Trail. It first finds an odd
   degree vertex (if there is any) and then calls printEulerUtil()
   to print the path */
void Graph::printEulerTour()
{
  // Find a vertex with odd degree
  int u = 0;
  for (int i = 0; i < V; i++)
      if (adj[i].size() & 1)
        {   u = i; break;   }

  // Print tour starting from oddv
  printEulerUtil(u);
  cout << endl;
}

// Print Euler tour starting from vertex u
void Graph::printEulerUtil(int u)
{
  // Recur for all the vertices adjacent to this vertex
  list<int>::iterator i;
  for (i = adj[u].begin(); i != adj[u].end(); ++i)
  {
      int v = *i;

      // If edge u-v is not removed and it's a a valid next edge
      if (v != -1 && isValidNextEdge(u, v))
      {
          cout << u << "-" << v << "  ";
          rmvEdge(u, v);
          printEulerUtil(v);
```

```
        }
    }
}

// The function to check if edge u-v can be considered as next edge in
// Euler Tout
bool Graph::isValidNextEdge(int u, int v)
{
  // The edge u-v is valid in one of the following two cases:

  // 1) If v is the only adjacent vertex of u
  int count = 0;  // To store count of adjacent vertices
  list<int>::iterator i;
  for (i = adj[u].begin(); i != adj[u].end(); ++i)
     if (*i != -1)
        count++;
  if (count == 1)
    return true;


  // 2) If there are multiple adjacents, then u-v is not a bridge
  // Do following steps to check if u-v is a bridge

  // 2.a) count of vertices reachable from u
  bool visited[V];
  memset(visited, false, V);
  int count1 = DFSCount(u, visited);

  // 2.b) Remove edge (u, v) and after removing the edge, count
  // vertices reachable from u
  rmvEdge(u, v);
  memset(visited, false, V);
  int count2 = DFSCount(u, visited);

  // 2.c) Add the edge back to the graph
  addEdge(u, v);

  // 2.d) If count1 is greater, then edge (u, v) is a bridge
  return (count1 > count2)? false: true;
}

// This function removes edge u-v from graph.  It removes the edge by
// replacing adjcent vertex value with -1.
void Graph::rmvEdge(int u, int v)
{
  // Find v in adjacency list of u and replace it with -1
  list<int>::iterator iv = find(adj[u].begin(), adj[u].end(), v);
  *iv = -1;

  // Find u in adjacency list of v and replace it with -1
  list<int>::iterator iu = find(adj[v].begin(), adj[v].end(), u);
  *iu = -1;
}
// A DFS based function to count reachable vertices from v
int Graph::DFSCount(int v, bool visited[])
{
  // Mark the current node as visited
  visited[v] = true;
  int count = 1;

  // Recur for all vertices adjacent to this vertex
  list<int>::iterator i;
```

```
    for (i = adj[v].begin(); i != adj[v].end(); ++i)
      if (*i != -1 && !visited[*i])
          count += DFSCount(*i, visited);

  return count;
}
// Driver program to test above function
int main()
{
  // Let us first create and test graphs shown in above figure
  Graph g1(4);
  g1.addEdge(0, 1);
  g1.addEdge(0, 2);
  g1.addEdge(1, 2);
  g1.addEdge(2, 3);
  g1.printEulerTour();

  Graph g2(3);
  g2.addEdge(0, 1);
  g2.addEdge(1, 2);
  g2.addEdge(2, 0);
  g2.printEulerTour();

  Graph g3(5);
  g3.addEdge(1, 0);
  g3.addEdge(0, 2);
  g3.addEdge(2, 1);
  g3.addEdge(0, 3);
  g3.addEdge(3, 4);
  g3.addEdge(3, 2);
  g3.addEdge(3, 1);
  g3.addEdge(2, 4);
  g3.printEulerTour();

  return 0;
}
```

Maximum Bipartite Matching :

```
vector<int>v[maxm];
int lefts[maxm],rights[maxm];
bool col[maxm];
// Number of bipartite matching …….
int match(int n){
        memset(lefts,-1,sizeof(lefts));
        memset(rights,-1,sizeof(rights));
        int i,j,k,l,done=0;
        do{
                memset(col,0,sizeof(col));
                done=1;
                for(i=1;i<=n;i++){
                        if(rights[i]==-1 &&dfs(i)) done=0;
                }

        }while(!done);
        k=0;
```

```
         for(i=1;i<=n;i++){
              if(rights[i]!=-1) k++;
        }
        return k;
}
bool dfs(int s){
        if(col[s]) return 0;
        col[s]=1;
        int i,j,k,l;
        for(i=0;i<v[s].size();i++){
              k=v[s][i];
              if(lefts[k]==-1){
                    rights[s]=k;
                    lefts[k]=s;
                    return 1;
              }
              else if(dfs(lefts[k])){
                    rights[s]=k;
                    lefts[k]=s;
                    return 1;
              }
        }
        return 0;}
```

Stable Marriage Problem :

```
/*
Problem : Loj 1400 ( Employment).
*/
vector<int>v[maxm];
int left[maxm],right[maxm],mat[maxm][maxm],matt[maxm][maxm],n,col[maxm];
// mat=left matrix , matt= right matrix ..
void match(int n){
        memset(col,0,sizeof(col));
        memset(left,-1,sizeof(left));
        memset(right,-1,sizeof(right));
        int i,j,k,done;
        do{
              memset(col,0,sizeof(col));
              done=1;
              for(i=1;i<=n;i++){
                    if(right[i]==-1&&dfs(i)) done=0;
              }
        }while(!done);

        for(i=1;i<=n;i++){
              printf(" (%d %d)",i,right[i]);
        }
        printf("\n");
}
bool dfs(int s){
```

```
          if(col[s]) return 0;
        col[s]=1;
        int i,j,k;
        for(j=0;j<v[s].size();j++){
                i=v[s][j];
                if(left[i]==-1){
                        left[i]=s;
                        right[s]=i;
                        return 1;
                }
                else{
                        k=left[i];
                        if(matt[i][k]>matt[i][s]){
                                right[k]=-1;
                                right[s]=i;
                                left[i]=s;
                                return 1;
                        }
                }
        }
        return 0;
}
```

Max - Flow :

Problem : Uva 10480 Sabotage .
Algo : Max-flow/Min-cut .

```
struct node{
        int no;
        int cost;
};
int n,m,tot,mat[maxm][maxm],pre[maxm],cap[maxm][maxm],col[maxm];
queue<int>q;
int in(int x){
        return 2*x-1;
}
int out(int x){
        return 2*x;
}
int comp(int x){
        if(x%2) return ((x+1)/2);
        else return x/2;
}
void ford(int s,int t);
int bfs(int s,int t);
int main(){
        int i,j,k,l,test,t=1;
        while(scanf("%d %d",&n,&m)==2){
                if(!n&&!m) break;
                memset(mat,0,sizeof(mat));
```

```
                memset(cap,0,sizeof(cap));
            for(i=1;i<=m;i++){
                scanf("%d %d %d",&k,&l,&j);
                mat[k][l]=mat[l][k]=j;
                cap[k][l]=cap[l][k]=1;
            }
            mat[2][n+1]=inf;
            mat[n+1][2]=inf;
            ford(1,n+1);
            printf("\n");
        }
        return 0;
}
int bfs(int s,int t){
        memset(pre,-1,sizeof(pre));
        memset(col,0,sizeof(col));
        int i,j,k,l;
        while(!q.empty()) q.pop();
        q.push(s);
        col[s]=1;
        while(!q.empty()){
            i=q.front(); q.pop();
            if(i==t) break;
            for(j=s;j<=t;j++){
                if(col[j]==0&&mat[i][j]>0){
                    pre[j]=i;
                    q.push(j);
                    col[j]=1;
                    if(j==t) break;
                }
            }
        }
        int wh,path,prev;
        path=inf;
        wh=t;
        while(pre[wh]!=-1){
            prev=pre[wh];
            path=mini(path,mat[prev][wh]);
            wh=prev;
        }
        wh=t;
        while(pre[wh]!=-1){
            prev=pre[wh];
            mat[prev][wh]-=path;
            mat[wh][prev]+=path;
            wh=prev;
        }
        if(path==inf )return 0;
        return path;
}
```

```
  void ford(int s,int t){
       int ret=0,i,j;
       while(1){
              int fl=bfs(s,t);
              if(fl) ret+=fl;
              else break;
       }
//     printf("ret - %d\n",ret);
       int flag[maxm][maxm];
       memset(flag,0,sizeof(flag));
       for(i=1;i<=n;i++){
              if(!col[i]) continue;
              for(j=1;j<=n;j++){
                     if(col[j]) continue;
                     if(cap[i][j])printf("%d %d\n",i,j);
              }
       }
}
```

Min-cost Flow :

```
/*
Problem :  Loj 1222 Gift Packing .
Algo    :  Min-cost Flow         .
*/
int in(int x){
       if(x%2) return x+1;
       return x-1;
};
struct node{
       int no;
       int cost;
};
struct edge{
       int u,v,cost,cap,next;
};
edge edges[maxe];
struct path{
       int a,b,c;
};
path paths[maxe];
priority_queue<node>pq;
int prev[maxm],pre[maxm],d[maxm],n,m,e,cas=1;
int mat[maxm][maxm];
bool operator<(const node &a,const node &b){
       return a.cost>b.cost;
}
void add(int u,int v,int cost,int cap){
       edges[e].u=u; edges[e].v=v;
       edges[e].cost=cost; edges[e].cap=cap;
```

```
          edges[e].next=prev[u];
        prev[u]=e++;
}
int mini(int a,int b){
        if(a<b) return a;
        return b;
}
bool dij(int s);
void ford(int s,int t);
int main(){
        int i,j,k,l,t=1,test,tot;
        scanf("%d",&test);
        while(test--){
                memset(prev,-1,sizeof(prev));
                scanf("%d",&n);
                tot=1;
                k=1;
                e=0;
                for(i=1;i<=n;i++){
                        for(j=1;j<=n;j++){
                                scanf("%d",&mat[i][j]);
                                paths[k].a=i; paths[k].b=n+j;
paths[k].c=mat[i][j];
                                k++;
                        }
                }
                tot=k;
                for(i=1;i<=n;i++){
                        add(0,i,0,1);
                    add(i,0,0,0);

                        add(n+i,2*n+1,0,1);
                add(2*n+1,n+i,0,0);
                }
                for(i=1;i<tot;i++){
                        k=paths[i].a; l=paths[i].b; j=paths[i].c;
                        add(k,l,-j,1);
                        add(l,k,j,0);
                }
                ford(0,2*n+1);
        }
        return 0;
}
bool dij(int s){
        int i,j,k,l;
        node temp,temp1;
        memset(pre,-1,sizeof(pre));
        for(i=0;i<=2*n+10;i++){
                d[i]=inf;
        }
```

```
          d[s]=0;
        temp.cost=0;
        temp.no=s;
        pq.push(temp);
        while(!pq.empty()){
                temp=pq.top(); pq.pop();
                j=temp.no;

                for(i=prev[j];i!=-1;i=edges[i].next){
                        k=edges[i].v;
                        if(edges[i].cap>0 && d[k]>d[j]+edges[i].cost){
                                d[k]=d[j]+edges[i].cost;
                                temp1.cost=d[k]; temp1.no=k;
                                pre[k]=i;
                                pq.push(temp1);
                        }

                }
        }
        return d[2*n+1]!=inf;
}
void ford(int s,int t){
        int i,j,k,l,wh,fl,ret,ans;
        fl=0;ans=0;
        while(dij(s)){
                wh=pre[t];
                ret=inf;
                while(wh!=-1){
                        ret=mini(ret,edges[wh].cap);
                        wh=pre[edges[wh].u];
                }
                wh=pre[t];
                while(wh!=-1){
                        edges[wh].cap-=ret;
                        edges[wh^1].cap+=ret;
                        wh=pre[edges[wh].u];
                }
                fl+=ret;
                ans+=(ret*d[t]);
        }
        printf("Case %d: %d\n",cas++,ans*-1);
}
```

Hierholzer's algorithm (Euler Cuircuit Print):

Hierholzer's 1873 paper provides a different method for finding Euler cycles that is more efficient than Fleury's algorithm:

- Choose any starting vertex *v*, and follow a trail of edges from that vertex until returning to *v*. It is not possible to get stuck at any vertex other than *v*, because the even degree of all vertices ensures that, when the trail enters another vertex *w* there must be an unused edge leaving *w*. The tour formed in this way is a closed tour, but may not cover all the vertices and edges of the initial graph.
- As long as there exists a vertex *v* that belongs to the current tour but that has adjacent edges not part of the tour, start another trail from *v*, following unused edges until returning to *v*, and join the tour formed in this way to the previous tour.

By using a data structure such as a doubly linked list to maintain the set of unused edges incident to each vertex, to maintain the list of vertices on the current tour that have unused edges, and to maintain the tour itself, the individual operations of the algorithm (finding unused edges exiting each vertex, finding a new starting vertex for a tour, and connecting two tours that share a vertex) may be performed in constant time each, so the overall algorithm takes linear time.

Erdos and Gallai Theorem:

```
// Given the degrees of the vertices of a graph, is it possible to construct
such graph Input - the deg[] array
int deg[MM], n, degSum[MM], ind[MM], minVal[MM];
bool ErdosGallai() { // 1 indexed
	bool poss = true;
	int i, sum = 0, j, r;
	for( i = 1; i <= n; i++ ) {
		if( deg[i] >= n ) poss = false;
		sum += deg[i];
	}
	//Summation of degrees has to be ODD and all degrees has to be < n - 1
	if( !poss || ( sum & 1 ) || ( n == 1 && deg[1] > 0 ) ) return false;
	sort( deg + 1, deg + n + 1, greater <int>() );
	degSum[0] = 0;
	j = n;
```

```
        for( i = 1; i <= n; i++ ) {
            degSum[i] = degSum[i-1] + deg[i];   //CONSTRUCTING: degSum
            for( ; j >= 1 && deg[j] < i; j-- );  //CONSTRUCTING: ind
            ind[i] = j+1;
        }
        //CONSTRUCTING : minVal
        for(r = 1; r < n; r++) {
            j = ind[r];
            if( j == n+1 ) minVal[r]=( n - r ) * r;
            else if( j <= r ) minVal[r] = degSum[n] - degSum[r];
            else {
                minVal[r] = degSum[n] - degSum[j-1];
                minVal[r] += (j-r-1)*r;
            }
        }
        //Checking : Erdos & Gallai Theorem
        for( r = 1; r < n; r++ ) if( degSum[r] > ( r * (r-1) + minVal[r] ) )
return false;
        return true;
}
```

Dynamic Programming

LIS(nlog(n)):

```
int in[maxim],L[maxim],p[maxim];
bool com(int a,int value)
{
     if(value>in[a])  return true;     //for strictly increasing LIS...ex - 1
2 2 ... ans - 2
     //if(value>=in[a])  return true;  //for non-decreasing LIS...ex - 1 2 2
... ans - 3
     return false;
}
void print_lis(int pos)
{
     if(p[pos]) print_lis(p[pos]);
     printf("%d\n",in[pos]);
}

int main()
{
     int n,i,l,pos;
     bool f;
     while(scanf("%d",&n)==1)
     {
          in[0]=-2147483648;
```

```
                l=1;
              L[0]=0;
              n++;
              rep(i,1,n)  scanf("%d",&in[i]);
              for(i=1;i<n;i++)
              {
                    pos = lower_bound(L,L+l,in[i],com) - L;
                    f = (pos==l);
                    if( f || in[ L[pos] ] > in[i] )
                    {
                          p[i] = L[pos-1];
                          L[pos] = i;
                          if(f) l++;
                    }
              }
              l--;
              printf("%d\n",l);
              printf("-\n");
              print_lis(L[l]);
       }
       return 0;
}
```

Convex Hull Trick 1:

```
/*
ID: brian_bi21
PROG: acquire (Usaco Mar 08).
Algo: Convex Hull Trick.
*/
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int pointer; //Keeps track of the best line from previous query
vector<long long> M; //Holds the slopes of the lines in the envelope
vector<long long> B; //Holds the y-intercepts of the lines in the envelope
//Returns true if either line l1 or line l3 is always better than line l2
bool bad(int l1,int l2,int l3)
{
       /*
       intersection(l1,l2) has x-coordinate (b1-b2)/(m2-m1)
       intersection(l1,l3) has x-coordinate (b1-b3)/(m3-m1)
       set the former greater than the latter, and cross-multiply to
       eliminate division
       */
       return (B[l3]-B[l1])*(M[l1]-M[l2])<(B[l2]-B[l1])*(M[l1]-M[l3]);
}
```

```
  //Adds a new line (with lowest slope) to the structure
void add(long long m,long long b)
{
	//First, let's add it to the end
	M.push_back(m);
	B.push_back(b);
	//If the penultimate is now made irrelevant between the antepenultimate
	//and the ultimate, remove it. Repeat as many times as necessary
	while (M.size()>=3&&bad(M.size()-3,M.size()-2,M.size()-1))
	{
		M.erase(M.end()-2);
		B.erase(B.end()-2);
	}
}
//Returns the minimum y-coordinate of any intersection between a given
vertical
//line and the lower envelope
long long query(long long x)
{
	//If we removed what was the best line for the previous query, then the
	//newly inserted line is now the best for that query
	if (pointer>=M.size())
		pointer=M.size()-1;
	//Any better line must be to the right, since query values are
	//non-decreasing
	while (pointer<M.size()-1&&
	  M[pointer+1]*x+B[pointer+1]<M[pointer]*x+B[pointer])
		pointer++;
	return M[pointer]*x+B[pointer];
}
int main()
{
	int M,N,i;
	pair<int,int> a[50000];
	pair<int,int> rect[50000];
	freopen("acquire.in","r",stdin);
	freopen("acquire.out","w",stdout);
	scanf("%d",&M);
	for (i=0; i<M; i++)
		scanf("%d %d",&a[i].first,&a[i].second);
	//Sort first by height and then by width (arbitrary labels)
	sort(a,a+M);
	for (i=0,N=0; i<M; i++)
	{
		/*
		When we add a higher rectangle, any rectangles that are also
		equally thin or thinner become irrelevant, as they are
		completely contained within the higher one; remove as many
		as necessary
		*/
```

```
                while (N>0&&rect[N-1].second<=a[i].second)
                    N--;
            rect[N++]=a[i]; //add the new rectangle
        }
        long long cost;
        add(rect[0].second,0);
        //initially, the best line could be any of the lines in the envelope,
        //that is, any line with index 0 or greater, so set pointer=0
        pointer=0;
        for (i=0; i<N; i++) //discussed in article
        {
            cost=query(rect[i].first);
            if (i<N)
                add(rect[i+1].second,cost);
        }
        printf("%lld\n",cost);
        return 0;
}
```

Divide and Conquer Optimization :

```
/*
Author : rng_58.
Sufficient Condition : pre[i][j]< pre[i][j+1] < pre[i][j+2].
Pre = Optimal path tracker .
*/

REP(i,N+1) dp[1][i] = get_cost(0, i);
for(i=1;i<K;i++) func(i, 0, N+1, 0, N);

void func(int d, int l, int r, int sepl, int sepr){
    int i;
    if(r-l == 1) return;
    int m = (l + r) / 2;
    int sep = -1;
    dp[d+1][m] = INF;
    for(i=sepl;i<=sepr;i++) if(i <= m){
        int tmp = dp[d][i] + get_cost(i, m);
        if(tmp < dp[d+1][m]){
            dp[d+1][m] = tmp;
            sep = i;
        }
    }

    func(d, l, m, sepl, sep);
    func(d, m, r, sep, sepr);
}
```

Knuth Optimization :

```
Let F[a][b] be the minimum cost to make all cuts from a to b inclusive.
In the standart n^3 solution :

F[a][b] = min(F[a][c-1] + F[c+1][b] + length(a, b)) - for every c from a to b;

Let P[a][b] be the c for which F[a][b] is minimized. It can be shown that:

F[a][b] = min(F[a][c-1] + F[c+1][b] + length(a, b)) - for every c from
                                          P[a][b - 1] to P[a + 1][b];
```

GEOMETRY

Macro :

```
//Macro........
#define ii int
#define maxm 100100
#define pi acos(-1.0)
#define eps 1e-9
#define sq(a) ((a)*(a))
#define dist(a,b) (sq(a.x-b.x) + sq(a.y-b.y))
#define iseq(a,b) (fabs(a-b)<eps)
#define eq(a,b) iseq(a,b)
#define area_t(x1,y1,x2,y2,x3,y3) ( x1*(y2-y3) + x2*(y3-y1) + x3*(y1-y2) )
#define spDist(lat1,long1,lat2,long2,r) ( r * acos( sin(lat1) * sin(lat2) +
cos(lat1) * cos(lat2) * cos(long1-long2) ) )

// Template........
template< class T > bool inside(T a, T b, T c) { return a<=b && b<=c; }
ii mini(ii a,ii b){
       if(a<b) return a; return b;
}
ii maxi(ii a,ii b){
       if(a>b) return a; return b;
}
```

Structure Declaration :

```
// Structure....
struct point { // Creates normal 2D point
    double x, y;
```

```
      point() {}
    point( double xx, double yy ) { x = xx, y = yy; }
    // Operator overloading......
    bool operator <(point b)const{
              if(!eq(x,b.x) )      return x < b.x;
              return y < b.y;
       }
       bool operator == (point b) const{
              if(eq(x,b.x) && eq(y,b.y))  return true;
              return false;
       }

};

struct point3D { // Creates normal 3D point
    double x, y, z;
};
struct line { // Creates a line with equation ax + by + c = 0
    double a, b, c;
    line() {}
    line( point p1,point p2 ) {
        a = p1.y - p2.y;
        b = p2.x - p1.x;
        c = p1.x * p2.y - p2.x * p1.y;
    }
};
struct circle { // Creates a circle with point 'center' as center and r as
radius
    point center;
    double r;
    circle() {}
    circle( point P, double rr ) { center = P; r = rr; }
};
struct segment { // Creates a segment with two end points -> A, B
    point A, B;
    segment() {}
    segment( point P1, point P2 ) { A = P1, B = P2; }
    bool operator < (const segment &a)const{
        return A<a.A;
    }
};
struct quad { // quadrilateral  with four points .. counterclock wise...
    point p[5];
    quad(){}
    quad(point a,point b,point c,point d){
        p[0]=a; p[1]=b; p[2]=c; p[3]=d;
    }
};
struct tri{ //  Triangle........... should be clock_wise...
    point p1,p2,p3;
```

```
      tri(){}
    tri(point _p1,point _p2,point _p3){
        p1=_p1; p2=_p2; p3=_p3;
    }
};
```

Function :

Essential Function

 # cross product = p0p1 * p0p2 :

```
//  cross product = p0p1 * p0p2..
inline double cross( point p0, point p1, point p2 ) {
    return( ( p1.x - p0.x ) * ( p2.y - p0.y ) - ( p2.x - p0.x ) * ( p1.y -
p0.y ) );
}
```

 # cross product p1*p2 :

```
inline double cross(point p1, point p2 ) {
    return( ( p1.x * p2.y ) - ( p2.x *  p1.y ) );
}
```

Find Distance

 # distance between point to point :

```
// distance between point to point...
inline double distancepp( point a, point b ) {
    return sqrt( ( a.x - b.x ) * ( a.x - b.x ) + ( a.y - b.y ) * ( a.y -
b.y ) );
}
```

 # distance between 3D point to point :

```
inline double distancepp( point3D a, point3D b ) {
    return sqrt( ( a.x - b.x ) * ( a.x - b.x ) + ( a.y - b.y ) * ( a.y - b.y )
+ ( a.z - b.z ) * ( a.z - b.z ) );
}
```

 # square distance between point to point :

```
// square distance between point to point.
inline double sq_distance( point a, point b ) {
    return ( a.x - b.x ) * ( a.x - b.x ) + ( a.y - b.y ) * ( a.y - b.y );
}
```

 # distance between point to line :

```
// distance between point to line....
inline double distancepl( point P, line L ) {
    return fabs( L.a * P.x + L.b * P.y + L.c ) / sqrt( L.a * L.a + L.b * L.b
);
}
```

#Distance - Point, Segment:

```
//Distance - Point, Segment:
inline double distanceps( point P, segment S ) {
       line L1 = line(S.A,S.B), L2; point P1;
       L2 = findPerpendicularLine( L1, P );
       if( intersection( L1, L2, P1 ) )
              if( eq ( distancepp( S.A, P1 ) + distancepp( S.B, P1 ),
distancepp( S.A, S.B ) ) )
                     return distancepl(P,L1);
       return mini ( distancepp( S.A, P), distancepp( S.B, P) );
}
```

Intersection

Intersection - Line, Line:

```
inline bool intersection( line L1, line L2, point &p ) {
       double det = L1.a * L2.b - L1.b * L2.a;
       if( eq ( det, 0 ) ) return false;
       p.x = ( L1.b * L2.c - L2.b * L1.c ) / det;
       p.y = ( L1.c * L2.a - L2.c * L1.a ) / det;
       return true;
}
```

Intersection - Segment, Segment:

```
inline bool intersection( segment L1, segment L2, point &p ) {
       if( !intersection( line( L1.A, L1.B ), line( L2.A, L2.B ), p) ) {
              return false; // can lie on another, just check their equations,
and check overlap
       }
       return(eq(distancepp(L1.A,p)+distancepp(L1.B,p),distancepp(L1.A,L1.B))
&&

       eq(distancepp(L2.A,p)+distancepp(L2.B,p),distancepp(L2.A,L2.B)));
}
```

Intersecting point between circle and line :

```
  inline bool intersectioncl(circle C,line L,point &p1,point &p2) {
	if( distancepl( C.center, L ) > C.r + eps ) return false;
	double a, b, c, d, x = C.center.x, y = C.center.y;
	d = C.r*C.r - x*x - y*y;
	if( eq( L.a, 0) ) {
		p1.y = p2.y = -L.c / L.b;
		a = 1;
		b = 2 * x;
		c = p1.y * p1.y - 2 * p1.y * y - d;
		d = b * b - 4 * a * c;
		d = sqrt( fabs (d) );
		p1.x = ( b + d ) / ( 2 * a );
		p2.x = ( b - d ) / ( 2 * a );
	}
	else {
		a = L.a *L.a + L.b * L.b;
		b = 2 * ( L.a * L.a * y - L.b * L.c - L.a * L.b * x);
		c = L.c * L.c + 2 * L.a * L.c * x - L.a * L.a * d;
		d = b * b - 4 * a * c;
		d = sqrt( fabs(d) );
		p1.y = ( b + d ) / ( 2 * a );
		p2.y = ( b - d ) / ( 2 * a );
		p1.x = ( -L.b * p1.y -L.c ) / L.a;
		p2.x = ( -L.b * p2.y -L.c ) / L.a;
	}
	return true;
}
```

//Intersection Area between Two Circles:

```
inline double intersectionArea2C( circle C1, circle C2 ) {
	C2.center.x = distancepp( C1.center, C2.center );
	C1.center.x = C1.center.y = C2.center.y = 0;
	if( C1.r < C2.center.x - C2.r + eps ) return 0;
	if( -C1.r + eps > C2.center.x - C2.r ) return pi * C1.r * C1.r;
	if( C1.r + eps > C2.center.x + C2.r ) return pi * C2.r * C2.r;
	double c, CAD, CBD, res;
	c = C2.center.x;
	CAD = 2 * acos( (C1.r * C1.r + c * c - C2.r * C2.r) / (2 * C1.r * c) );
	CBD = 2 * acos( (C2.r * C2.r + c * c - C1.r * C1.r) / (2 * C2.r * c) );
	res=C1.r * C1.r * ( CAD - sin( CAD ) ) + C2.r * C2.r * ( CBD - sin (
CBD ) );
	return .5 * res;
}
```

conversion

 # radian to degree :

```
double convrd(double theta){
    double ret=180; ret/=pi; return ret*theta;
}
```

degree to radian :

```
double convdr(double theta){
    double ret=pi; ret/=(double)180.0; return ret*theta;
}
```

convert spherical to cartesian co-ordinate......

```
// convert spherical to cartesian co-ordinate......
void sph_to_cartesian(double R,double lat,double lng,point3D &p){

    lat=convdr(lat);
    lng=convdr(lng);

    p.x=R*sin(lat)*cos(lng);
    p.y=R*sin(lat)*sin(lng);
    p.z=R*cos(lat);
}
```

convert longitude/latitude to cartesian co-ordinate......

```
// convert longitude/latitude to cartesian co-ordinate......
void earth_to_cartesian(double R,double lat,double lng,point3D &p){

    lat=convdr(lat);
    lng=convdr(lng);

    p.x=R*cos(lat)*cos(lng);
    p.y=R*cos(lat)*sin(lng);
    p.z=R*sin(lat);
}
```

convert cartesian co-ordinate to longitude/latitude

```
lat = asin(z / R)
lon = atan2(y, x)
```

Inside Function

```
//  check whether a  point inside a Segment ....
bool inside_segment(segment S,point P){
      if( eq ( distancepp( S.A, P ) + distancepp( S.B, P ), distancepp( S.A,
S.B ) ) ) return 1;
      return 0;
```

```
  }
//  check whether a  point inside a triangle ....
bool inside_tri(tri t,point p){

    point p1=t.p1,p2=t.p2,p3=t.p3;

    //  check  for boundary........
    if(iseq(cross(p,p1,p2),0) && inside_segment(segment(p1,p2),p)) return 1;
    if(iseq(cross(p,p2,p3),0) && inside_segment(segment(p2,p3),p)) return 1;
    if(iseq(cross(p,p1,p3),0) && inside_segment(segment(p1,p3),p)) return 1;
    // .........

    if(cross(p,p1,p2)*cross(p3,p1,p2)<0) return 0;
    if(cross(p,p2,p3)*cross(p1,p2,p3)<0) return 0;
    if(cross(p,p1,p3)*cross(p2,p1,p3)<0) return 0;

    return 1;

}
```

Point Inside a Convex Polygon(O(lgn)) :

```
/*C[] array of points of convex polygon in ccw order,  nc number of points in
C, p target points.
returns true if p is inside C (including edge) or false otherwise.  complexity
O(lg n) */
int triArea2(const point &a, const point &b, const point &c)   {
       return (a.x*(b.y-c.y) + b.x*(c.y-a.y) + c.x*(a.y-b.y));
}
bool inConvexPoly(point *C, int nc, const point &p)  {
       int st = 1, en = nc - 1, mid;
       while(en - st > 1) {
              mid = (st + en)>>1;
              if(triArea2(C[0], C[mid], p) < 0)       en = mid;
              else   st = mid;
       }
       //  for point in border............
       if(iseq(triArea2(C[0], C[1], p),0.0)) return false;
       if(iseq(triArea2(C[0], C[nc-1], p),0.0)) return false;
       if(iseq(triArea2(C[nc-1], C[nc-2], p),0.0)) return false;
       // finish.............
       if(triArea2(C[0], C[st], p) < 0 ) return false;
       if(triArea2(C[st], C[en], p) < 0 || iseq(triArea2(C[st], C[en], p) ,
0.0) ) return false;  // iseq() for border testing .....
       if(triArea2(C[en], C[0], p) < 0 ) return false;
       return true;
}
```

AREA

```
//  area of  polygon.........
```

```
  double areaPoly(point P[],int n){
    double area=0;
    for( int i = 0, j = n - 1; i < n; j = i++ ) area += P[j].x * P[i].y - 
P[j].y * P[i].x;
    return fabs(area)*.5;
}
```

Convex Hull :

```
//  convex Hull = graham scan O(nlgn)
bool sort_x(point a,point b){
    if(iseq(a.x,b.x)) return a.y<b.y;
    return a.x<b.x;
}
bool sort_y(point a,point b){
    if(iseq(a.y,b.y)) return a.x<b.x;
    return a.y<b.y;
}
point p[maxm]; // p=points for  convex hull...
bool normal(const point &a, const point &b) {   return (iseq(a.x,b.x) ? a.y < 
b.y : a.x < b.x);}
bool issame(const point &a, const point &b) {   return (iseq(a.x,b.x) && 
iseq(a.y,b.y));}
void makeUnique(point p[],int &np) {  sort(&p[0], &p[np], normal);    np = 
unique(&p[0], &p[np], issame) - p;}
//sort by polar angle>>>(convex_hull)
bool comp(point a,point b){
    double d = cross(p[0], a, b);
    if(d<0) return false;
    if(iseq(d,0) && dist(p[0], b) < dist(p[0], a)) return false;
    return true;
}
void convex_hull(point ans[],point p[],int &n,int &nc){
    makeUnique(p,n);
    int i,pos = 0;
    for(i=1; i<n; i++)
        if(p[i].y<p[pos].y || (p[i].y==p[pos].y && p[i].x<p[pos].x))
            pos = i;
    swap(p[0], p[pos]);
    sort(p+1, p+n, comp);

    ans[0] = p[0];
    if(n>=2)    ans[1] = p[1];
    for(i=nc=2; i<n; i++)
    {
        while(nc>=2 && cross(ans[nc-2], ans[nc-1], p[i])<0||iseq(cross(ans[nc-
2], ans[nc-1], p[i]),0)) nc--;
        ans[nc++] = p[i];
    }
    if(n==1)        nc=1;
    else if(nc==2)
```

```
    {
      if(p[0].x == p[1].x && p[0].y == p[1].y)   nc=1;
    }

}
```

Important Formulas

Area of a triangle :

Let K be the triangle's area and let a, b and c, be the lengths of its sides. By Heron's Formula , the area of the triangle is

K = sqrt(s * (s-a) * (s-b) * (s -c)).

where **S** is the semiperimeter .

$$s = \frac{1}{2}(a+b+c)$$.

length of median to side c = sqrt(2*(a*a+b*b)-c*c)/2

length of bisector of angle C = sqrt(ab[(a+b)*(a+b)-c*c])/(a+b) .

Radius of a In-cicle:

The radius of the incircle (also known as the **inradius**, r) is

$$r = \frac{2K}{P} = \sqrt{\frac{(s-a)(s-b)(s-c)}{s}}.$$

Thus, the area K of a triangle may be found by multiplying the inradius by the semiperimeter:

$$K = rs.$$

Regular Polygon :

a regular polygon is a Polygon that is equiangular (all angles are equal in measure) and equilateral (all sides have the same length).

Angle :

For a regular convex n-gon, each interior angle has a measure of:

$$(n-2) \times \frac{180}{n}$$ degrees .

Apothem: The **apothem** of a regular polygon is a line segment from the center to the midpoint of one of its sides. Equivalently, it is the line drawn from the center of the polygon that **is perpendicular to one of its sides.**

Circumradius:

The **circumradius** from the center of a regular polygon to one of the vertices is related to the side **length** ***s*** or to the **apothem** ***a*** by

$$r = \frac{s}{2\sin\frac{\pi}{n}} = \frac{a}{\cos\frac{\pi}{n}} .$$

Area :

The **area** ***A*** of a convex regular *n*-sided polygon having **Side** ***s***, **circumradius** ***r***, **apothem** ***a***, and **perimeter** ***p*** is given by

$$A = \tfrac{1}{2}nsa = \tfrac{1}{2}pa = \tfrac{1}{4}ns^2\cot\tfrac{\pi}{n} = na^2\tan\tfrac{\pi}{n} = \tfrac{1}{2}nr^2\sin\tfrac{2\pi}{n}$$

Centroid of a 2D polygon:

As in the calculation of the area above, xN is assumed to be x0, in other words the polygon is closed.

$$c_x = \frac{1}{6A}\sum_{i=0}^{N-1}(x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i) \qquad c_y = \frac{1}{6A}\sum_{i=0}^{N-1}(y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i)$$

Searching

Ternary Search :

```
double ts(){
	double min=0;
	double max=1;
	int c=100; //for higher precision have to increase
	double k,l,f,g;
	while(c--){

		f=min+(max-min)/(double)3.0;
		g=min+(double)2.0*((max-min)/(double)3.0);
		k=fun(f); l=fun(g);
		if(k<l){
			max=g;
		}
		else{
			min=f;
		}
	}

	return (min+max)/2.0 ;
}
// problem dependent . . . .
```

```
  double fun(double piv){
}
```

Game Theory

Nim - Game :

```
/*
Author  : Rashedul Hasan Rijul .
problem : Uva - 10165 ( stone Games) .
Algo    : Nim .
*/
#define ii long long int
int n;
int main(){
	int i,j,k,l,test,t=1;
	while(scanf("%d",&n)==1){
		if(!n) break;
		ii ans=0;
		for(i=1;i<=n;i++){
			scanf("%d",&k);
			ans=ans^k;
		}
		if(ans){
			printf("Yes\n");
		}
		else {
			printf("No\n");
		}
	}
	return 0;
}
```

Misere - Nim Game :

```
/*
Author  : Rashedul Hasan Rijul .
problem : Light OJ - 1253 ( Misere Nim) .
Algo    : Misere-Nim .
*/
#define maxm 1000
#define ii int
int a[maxm];
int main(){
	int i,j,k,l,test,t=1,n;
	scanf("%d",&test);
	while(test--){
		scanf("%d",&n);
		ii ans=0,ans1;
		bool fl=0;
		l=0;
```

```
            for(i=1;i<=n;i++){
                scanf("%d",&k);
                if(k==1){
                    l++;
                    ans^=1;
                }
                else{
                    ans^=k;
                    fl=1;
                }
            }
            // Alice play first....
            if(!fl){
                if(l%2==1) printf("Case %d: Bob\n",t++);
                else printf("Case %d: Alice\n",t++);
                continue;
            }
            if(ans) printf("Case %d: Alice\n",t++); // Alice play first....
            else  printf("Case %d: Bob\n",t++);
        }
        return 0;
}
```

Sprunge - Grundy Number :

```
problem : Light oj 1315 ( Game of Hyper Knight) .
int dp[maxm][maxm];
int dx[]={-1,-1,1,-2,-2,-3};
int dy[]={-2,-3,-2,-1,1,-1};
int cal(int i,int j){
        if(dp[i][j]!=-1) return dp[i][j];
        set<int>s;
        int ret=0,i1,k,l,j1,val;
        for(i1=0;i1<6;i1++){
                k=i+dx[i1]; l=j+dy[i1];
                if(k>=0&&l>=0){
                        s.insert(cal(k,l));
                }
        }
        while(s.find(ret)!=s.end()){
                ret++;
        }
        return dp[i][j]=ret;
}
```

Green Hackenbush :

```
/*
Author  : misof
Problem : ipsc 2003 G [hackenbush] (c) misof
Algo    : Green Hackenbush .
*/
```

```
#define min(x,y) ((x)<(y))?(x):(y)

int Cases,N,M;
vector< list<int> > G,G2;
vector<int> GV;
vector<int> visited,from,time_disc,time_up;
int DFStime;

void DFS_Visit(int v){
  int edges_to_parent=0;
  visited[v]=1; time_disc[v]=time_up[v]=++DFStime;
  for (list<int>::iterator start=G[v].begin();start!=G[v].end();start++) {
    if (!visited[*start]) { from[*start]=v; DFS_Visit(*start);
time_up[v]=min(time_up[v],time_up[*start]); }
    else {
      if ((*start)!=from[v]) { time_up[v]=min(time_up[v],time_disc[*start]); }
      else {
        if (edges_to_parent) { time_up[v]=min(time_up[v],time_disc[*start]); }
        edges_to_parent++;
      }
    }
  }
}

void FindBridges(void){
  time_disc.clear(); time_up.clear(); visited.clear(); from.clear();
  visited.resize(N+3,0); time_disc.resize(N+3,0); time_up.resize(N+3,0);
from.resize(N+3,0);
  from[1]=1; DFStime=0;
  DFS_Visit(1);
}

int IsBridge(int v_lo, int v_high) {
  if (v_high!=from[v_lo]) return 0;
  return ( time_disc[v_lo]==time_up[v_lo] );
}

void ContractGraph(void){
  vector<int> color(N+3,0);
  int colors=1;
  color[1]=1;

  list<int> Q;
  Q.clear(); Q.push_back(1);
  while (!Q.empty()) {
    int where=Q.front(); Q.pop_front();
    for (list<int>::iterator it=G[where].begin(); it!=G[where].end(); it++) if
(!color[*it]) {
```

```
        if (IsBridge(*it,where)) color[*it]=++colors; else
color[*it]=color[where];
      visited[*it]=1; Q.push_back(*it);
    }
  }

  G2.clear(); G2.resize(N+3);
  for (int i=1;i<=N;i++)
    for (list<int>::iterator it=G[i].begin(); it!=G[i].end(); it++)
      G2[color[i]].push_back(color[*it]);
}

int GrundyValue(int v){
  int loops=0,gv=0;

  if (GV[v]!=-1) return GV[v]; GV[v]=1000000000;

  for (list<int>::iterator start=G2[v].begin(); start!=G2[v].end(); start++) {
    if ((*start)==v) loops++; else if (GV[*start]!=1000000000)
gv^=(1+GrundyValue(*start));
  }
  loops/=2; if (loops%2) gv^=1;
  return GV[v]=gv;
}

int main(void){
  int v1,v2;

  //freopen("g1.in","r",stdin);
  //freopen("out.txt","w",stdout);

  cin >> Cases;
  while (Cases--) {
    // read graph dimensions
    cin >> N >> M;
    // read the graph
    G.clear(); G.resize(N+3);
    for (int i=0;i<M;i++) { cin >> v1 >> v2; G[v1].push_back(v2);
G[v2].push_back(v1); }
    // collapse all circuits in the graph
    FindBridges();
    ContractGraph();
    // compute the SG value
    GV.clear(); for (int i=0;i<=N;i++) GV.push_back(-1);
    int result=GrundyValue(1);
    if (result) cout << "Alice\n"; else cout << "Bob\n";    //cout << result
<< "\n";
  }
  return 0;
}
```

Red blue Hacken Bush (stalk Only):

```
/*
problem: codechef (chef game) .
Algo : red-blue hackenbush.
*/
#define MAXN 55
typedef long long int64;

/*
    Problem can be reduced to red-black hackenbush
    http://en.wikipedia.org/wiki/Hackenbush
    Each pile represent a hackenbush stalk
    Game value cooresponding to hackenbush stalk is easy to find.
    Please refer here : http://www.geometer.org/mathcircles/hackenbush.pdf.
    For hackebush games value of two disjoint game is equal to sum of individual game value.
    (http://www-math.mit.edu/~rstan/transparencies/games.pdf)

*/

int t,n,tcase;
int arr[MAXN];

int64 calculate(){
   int64 res = 0; int64 value = 1LL<<48;
   res = (arr[0]%2==0)?value:-value;
   bool is_changed = false;
   for(int i=1; i<n; ++i){
      assert(arr[i]!=arr[i-1]);
      if(arr[i]%2 != arr[i-1]%2){
          is_changed = true;
      }
      if(is_changed) value /= 2;
      res += (arr[i]%2==0)?value:-value;
   }
   return res;
}

int main(){
  for(scanf("%d",&tcase); tcase; tcase-=1){
     scanf("%d",&t);
     int64 res = 0;
     for(int i=0; i<t; ++i){
       scanf("%d",&n);
       for(int j=0; j<n; ++j) scanf("%d",&arr[j]);
       sort(arr,arr+n);
       res += calculate();
     }
```

```
      if(res > 0 ) printf("FIRST\n");
    else if(res < 0 ) printf("SECOND\n");
    else printf("DON'T PLAY\n");
  }
  return 0;
}
```

Matrix

Gaussian Elimination :

Problem : LOJ 1151 - Snakes and Ladders

```
#define maxm 110

int n,m;
int pos[maxm];

///Gaussian  Elimination......................................
#define eps (1e-9)
#define iseq(a,b) (fabs(a-b)<eps)

//  a1x1+a2x2+.....=b1 .....
double a[maxm][maxm],b[maxm],x[maxm];

void gauss(int r,int c){

    int i,j,k,l;
    double val;

    i=j=0;

    while(i<r&&j<c){

        for(k=i;k<r;k++){
            if(iseq(a[k][j],0.0)) continue;
            for(l=j;l<c;l++){
                swap(a[i][l],a[k][l]);
            }
            swap(b[i],b[k]);
            break;
        }
        if(k==r){
            j++; continue;
        }
        // Making jth col of every row  from (i+1)th to rth row  into
zero........
```

```
            for(k=i+1;k<r;k++){
              val=a[k][j]/a[i][j];
              for(l=j;l<c;l++){
                  a[k][l]-=(a[i][l]*val);
              }
              b[k]-=(b[i]*val);

          }

          i++; j++;
      }

      ///  Additional  information.............

      /*
      rep(k,i,r)
      {
      rep(j,0,c)        if(!(fabs(a[k][j])<eps))   goto stop ;

        if(!(fabs(b[k])<eps))               return -1 ;       // no solution

          stop : ;
          }

            if(i>c)           return -1 ; // no solution
            if(i==c)         return 0 ; // unique solution
            if(i<c)           return 1 ; // multiple solution
      */

      /// ................

      for(i=c-1;i>=0;i--){
          x[i]=b[i];
          for(k=i+1;k<c;k++){
              x[i]-=(a[i][k]*x[k]);
          }
          if(!iseq(a[i][i],0.0)) x[i]/=a[i][i];
      }
  }
  ///   Gaussian  Elimination   Finish.....................

  int main(){

      int i,j,k,l,test,t=1;
      scanf("%d",&test);
      while(test--){

          for(i=0;i<=100;i++){
              pos[i]=i;
          }
```

```
        scanf("%d",&m);

        for(i=1;i<=m;i++){
            scanf("%d %d",&k,&l);
            k--; l--;
            pos[k]=l;
        }


        n=100;
        memset(a,0.0,sizeof(a));

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

            a[i][i]=1.0;

            if(pos[i]!=i){
                a[i][pos[i]]=-1.0;
                b[i]=0.0;
                continue;
            }

            if(i==n-1){  b[i]=0;  continue; }
            else b[i]=1;

            // prob= probabilty.........
            double prob=(double) 1.0/ (double) 6.0;

            for(j=1;j<=6;j++){
                k=(i+j);
                if(k>99) k=i;
                k=pos[k];
                a[i][k]-=(prob);
            }

        }

        gauss(n,n);

        printf("Case %d: %.8lf\n",t++,x[0]);

    }

    return 0;
}
```

Matrix Exponentiation :

```
  Problem : Uva 12470 (Tribonacci).

#define maxm 10
#define ii long long int

ii n,mod;
ii base[3][3]={{1,1,1},{1,0,0},{0,1,0}};
ii unit[3][3]={{1,0,0},{0,1,0},{0,0,1}},res[3][3];

void cal(ii a[3][3],ii b[3][3]){

       ii ret[3][3]; int i,j,k;
       memset(ret,0,sizeof(ret));

       for(i=0;i<3;i++){
              for(j=0;j<3;j++){
                     for(k=0;k<3;k++){
                            ret[i][j]+=(a[i][k]*b[k][j]);
                            ret[i][j]%=mod;
                     }
              }
       }

       memcpy(a,ret,sizeof(ret));

}


void exp(ii r[3][3],ii n){

       ii b[3][3];
       memcpy(r,unit,sizeof(unit));
       memcpy(b,base,sizeof(base));

       while(n>0){
              if(n%2==1) cal(r,b);
              n/=2;
              cal(b,b);
       }
}

int main(){

       mod=1000000009;
              if(!n) break;
              if(n==1){
                     printf("0\n"); continue;
              }
              if(n==2){
                     printf("1\n"); continue;
```

```
            }
        if(n==3){
                printf("2\n"); continue;
        }

        exp(res,n-3);
        ii ans=0;
        ans+=(res[0][0]*2+res[0][1]*1);
        ans%=mod;

        printf("%lld\n",ans);
```

Memorization Technique in Matrix Expo :

```
memcpy(mem[0],base,sizeof(base));

for(i=1;i<=62;i++){
    cal(mem[i-1],mem[i-1],mem[i]);
}

void exp(ii r[4][4],ii n){

        ii b[4][4];
        memcpy(r,unit,sizeof(unit));
        memcpy(b,base,sizeof(base));

        int j=0;
        while(n>0){
                if(n%2==1) cal(r,mem[j],r);
                n/=2; j++;
                //cal(b,b);
        }
}
```

Number Theory

Prime Generation (Sieve) + Factoriation :

```
#define maxm 10000600
#define ii long long int

bool p[maxm];
int prie[664610],c,tot,totn;

void fact(ii n);
void take(int n);
```

```
  void gen(int n);
int main(){
	int i,j,k,l,test,t=1;
	gen(maxm-90);
	take(maxm-90);
	return 0;
}
// Factoriation . . . .
void fact(ii n){
	int i,j,k,l;
	node temp;
	ii sq;
	double nd=n;
	sq=sqrt(nd);
	v.clear();
	for(i=0;prime[i]<=sq;i++){
		if(n%prime[i]) continue;
		k=0;
		while(n%prime[i]==0){
			n/=prime[i];
			k++;
		}
		sq=sqrt(n);
		temp.count=k; temp.num=prime[i];
		v.push_back(temp);
		if(n==1) break;
	}

	if(n>1){
		temp.count=1; temp.num=n;
		v.push_back(temp);
	}
}
void take(int n){
	prime[c++]=2;
	for(int i=3;i<=n;i++){
		if(!p[i]) prime[c++]=i;
	}
	tot=c;
}
void gen(int n){
	int i,j,k,l,sq;
	p[0]=p[1]=1;
	sq=sqrt(n);
	for(i=4;i<=n;i+=2) p[i]=1;
	for(i=3;i<=sq;i+=2){
		if(p[i]) continue;
		for(j=i*i;j<=n;j+=(2*i)){
			p[j]=1;
		}
```

```
        }}
```

Segmented Seive :

```
/*
Author  : Rashedul Hasan Rijul
Problem : LOJ 1197 Help Hanzo
Algo    : Segmented Seive
*/
#define maxm 10111100
#define ii long long int

bool p[1000000+100],segment[maxm];
void gen(int n){
    int i,j,k,l,sq;
    sq=sqrt(n);
    p[0]=1; p[1]=1;
    for(i=4;i<=n;i+=2) p[i]=1;

    for(i=3;i<=sq;i+=2){
        if(p[i]) continue;
        for(j=i*i;j<=n;j+=(2*i)){
            p[j]=1;
        }
    }
}
int maxi(int a,int b){
    if(a>b) return a;
    return b;
}
int f(int l,int i){
    if(l%i==0) return maxi(l,i*i);
    return maxi(l+(i-(l%i)),i*i);
}

int main(){

    int i,j,k,l,test,t=1,h;

    freopen("in.txt","r",stdin);
    gen(1000000);
    scanf("%d",&test);
    while(test--){
        scanf("%d %d",&l,&h);
        memset(segment,0,sizeof(segment));
        if(l==1) segment[0]=1;
        int sq=sqrt(h);
        for(i=2;i<=sq;i++){
            if(p[i]) continue;
            for(j=f(l,i);j>=l&&j<=h;j+=i){
                segment[j-l]=1;
            }
```

```
            }

        int ans=0;
        ii i1;
        for(i1=l;i1<=h;i1++){
            if(!segment[i1-l]) ans++;
        }
        printf("Case %d: %d\n",t++,ans);
    }
}
```

Bitwise Seive :

```
#define maxm 100000000
int p[(maxm/32)+10],tot,prime[(maxm/10)+1000];
int on(int n,int k){
    return (n|(1<<k));
}
bool chck(int n,int k){
    return (bool)(n&(1<<k));
}
void gen(int n){

    int i,j,k,l,sq;

    sq=sqrt(n);

    for(i=3;i<=sq;i+=2){
        if(chck(p[i>>5],i&31)) continue;
        for(j=(i*i);j<=n;j+=(i<<1)){
            p[j>>5]=on(p[j>>5],j&31);
        }
    }

    // takine prime into array>>>>>>>>>
    prime[tot++]=2;
    printf("%d\n",2);
    for(i=3;i<=n;i+=2){
        if(!chck(p[i>>5],i&31)){
                    prime[tot++]=i;
                    //if((tot-1)%100==0) printf("%d\n",i);
              }
    }
    printf("%d\n",tot);
}
```

Euler Phi :

```
#define s 50100
double phi[s];
```

```
  bool prime[s];
void geneuler(int n){
       double temp;
              phi[0]=0;
       phi[2]=1;
       int sq=sqrt(n);
       int i,j;
       for(i=4;i<=n;i+=2){
              prime[i]=1;
              temp=i;
              phi[i]=temp*.5;
       }
       for(i=3;i<=n;i+=2) phi[i]=i;
       for(i=3;i<=n;i+=2){
              if(prime[i]==0){
                     phi[i]=i-1;
                     for(j=2*i;j<=n;j+=i){
                            prime[j]=1;
                            temp=i;
                            phi[j]*=((temp-1)/temp);
                     }
              }
       }

}
```

Primality Test:

```
/* this function calculates (a*b)%c taking into account that a*b might
overflow */
ii mulmod(ii a,ii b,ii c){
    ii x = 0,y=a%c;
    while(b > 0){
        if(b%2 == 1){
            x = (x+y)%c;
        }
        y = (y*2)%c;
        b /= 2;
    }
    return x%c;}

/* Miller-Rabin primality test, iteration signifies the accuracy of the test
*/
bool Miller(long long p,int iteration){
    if(p<2){
```

```
            return false;
        }
        if(p!=2 && p%2==0){
            return false;
        }
        long long s=p-1;
        while(s%2==0){
            s/=2;
        }
        for(int i=0;i<iteration;i++){
            long long a=rand()%(p-1)+1,temp=s;
            long long mod=big_mod(a,temp,p);
            while(temp!=p-1 && mod!=1 && mod!=p-1){
                mod=mulmod(mod,mod,p);
                temp *= 2;
            }
            if(mod!=p-1 && temp%2==0){
                return false;
            }
        }
        return true;
}
```

Big Mod :

```
#define ii __int64
ii  big_mod(int b,int p,int m){
    if(p==0) return 1;
    if(p==1) return b%m;
    ii ret;
    ret=big_mod(b,p/2,m);
    ret*=ret; ret%=m;
    if(p%2) ret*=b;
    ret%=m;
    return ret;
}
```

Modular Inverse :

```
// Extended Euclid .....  for  finding  Modular inverse
struct node{
       ii x,y,g;
       node(){};
       node(ii xx,ii yy,ii gg){ x=xx; y=yy; g=gg;};
};
//  ax+by=g where g=gcd(a,b)....
node euclid(ii a,ii b);
////////////////*******\\\\\\\\\\\\\\\\\\
node euclid(ii a,ii b){
       if(!b) return node(1,0,a);
       node r=euclid(b,a%b);
       return node(r.y,r.x-(a/b)*r.y,r.g);
```

```
  }
ii mod_inv(ii n,ii m){
    node t=euclid(n,m);
    if(t.g>1) return 0;
    ii ret=t.x%m;
    if(ret<0) ret+=m;
    return ret;
}
```

Extended Euclid :

```
/*
Problem : LOJ 1306 (Solutions to an Equation ).
Algo    : Extended Euclid ( Number of solution of a Linear Diaphontine
equation in a given range ).
*/

// Extended Euclid .....
struct node{
       ii x,y,g;
       node(){};
       node(ii xx,ii yy,ii gg){ x=xx; y=yy; g=gg;};
};
//  ax+by=g where g=gcd(a,b)....
node euclid(ii a,ii b){
       if(!b) return node(1,0,a);
       node r=euclid(b,a%b);
       return node(r.y,r.x-(a/b)*r.y,r.g);
}
//............//
ii A,B,C,xl,xh,yl,yh;
ii find_lo(ii x0,ii y0,ii ag,ii bg);
int valid_lo(ii x0,ii t,ii bg,ii lo,ii hi);
ii find_hi(ii x0,ii y0,ii ag,ii bg);
int valid_hi(ii x0,ii t,ii bg,ii lo,ii hi);
ii common(ii a,ii b,ii c,ii d){
    if(b<c || d<a) return 0;
    if(a>=c && b<=d) return (b-a+1);
    if(c>=a && d<=b) return (d-c+1);
    if(b>=c && a<=c) return (b-c+1);
    if(a<=d && b>d)  return (d-a+1);
    return 0;
}
ii find_ans(){
    node piv=euclid(A,B);
    if(!piv.g){
        if(C) return 0;
        return (xh-xl+1)*(yh-yl+1);
    }
```

```
      if(C%piv.g) return 0;

     ii x0=piv.x,y0=piv.y;
     x0*=(C/piv.g); y0*=(C/piv.g);

     ii ag=A/piv.g,bg=B/piv.g;

     // x= x0 - t*bg , y= y0 + t*ag;
     ii lo1=find_lo(x0,bg,xl,xh);
     ii lo2=find_lo(y0,-ag,yl,yh);

     ii hi1=find_hi(x0,bg,xl,xh);
     ii hi2=find_hi(y0,-ag,yl,yh);

     return common(lo1,hi1,lo2,hi2);
}

        scanf("%lld %lld %lld %lld %lld %lld %lld",&A,&B,&C,&xl,&xh,&yl,&yh);
         C=-C;
         printf("Case %d: %lld\n",t++,find_ans());

ii find_lo(ii x0,ii bg,ii lox,ii hix){
   // x= x0 - t*bg , y= y0 + t*ag;
   ii lo=-inf,hi=inf;
   ii mid;
   while(lo<hi){
       mid=lo+hi; mid/=2;
       if(valid_lo(x0,mid,bg,lox,hix)){
           if(hi==mid){
               if(valid_lo(x0,mid-1,bg,lox,hix)) return mid-1;
               return mid;
           }
           hi=mid;
       }
       else{
           lo=mid+1;
       }
   }
   return hi;
}
ii find_hi(ii x0,ii bg,ii lox,ii hix){
   // x= x0 - t*bg , y= y0 + t*ag;
   ii lo=-inf,hi=inf;
   ii mid;
   while(lo<hi){
       mid=lo+hi; mid/=2;
       if(valid_hi(x0,mid,bg,lox,hix)){
           if(lo==mid){
               if(valid_hi(x0,mid+1,bg,lox,hix)) return mid+1;
               return mid;
```

```
            }
            lo=mid;
        }
        else{
            hi=mid-1;
        }

    }
    return lo;
}

int valid_lo(ii x0,ii t,ii bg,ii lo,ii hi){
    // check increasing .....
    if(bg<0){
        if(x0-(t*bg)<lo) return 0;
        return 1;
    }
    else{
        if(x0-(t*bg)>hi) return 0;
        return 1;
    }
}
int valid_hi(ii x0,ii t,ii bg,ii lo,ii hi){
    // check increasing .....
    if(bg<0){
        if(x0-(t*bg)>hi) return 0;
        return 1;
    }
    else{
        if(x0-(t*bg)<lo) return 0;
        return 1;
    }
}
```

DigitCount of N!:

```
digitCountEfficient( N ) {
        double logVal = 0;
        for( i = 0; prime[ i ] <= N; i++ ){
                logVal += pPowers[ i ] * log10( prime[ i ] );
        }
        return ( int )logVal + 1;
}
```

Most Significant digit of N!

```
mostSignificantDigit( N ){
        double logVal = 0;
        for( i = 0; prime[ i ] <= N; i++ ){
                logVal += pPowers[ i ] * log10( prime[ i ] );
        }
        fractionPart = logVal - ( int )logVal; //get the fractional part

        return ( int )pow( 10, fractionPart);
```

```
  }
```

First k digits of N!, k<=12

```
firstKDigits( N , K ){
	long double logVal = 0; //long double precision
	for( i = 0; prime[ i ] <= N; i++ ){
		logVal += pPowers[ i ] * log10( prime[ i ] ); //after factoring
	}
	fractionPart = logVal - ( int )logVal + K - 1 ; //get the fractional part

printf( "%I64d\n" ,(__int64) powl ( 10.0, fractionPart ));
}
```

Number of rightMost zeros in N!

```
numberOfRightMostZeros( N ){
	return min( pPowers[0] , pPowers[2] ); //in array, prime[0]=2, prime[2]=5
}
```

Last non-zero digit of N!

```
lastNonZeroDigit( N ){
	__int64 prod = 1;
	for(int i = 1 ; i <= N; i++ ){
		__int64 f = i;
		while( f % 5 == 0 ){
			f /= 5;
			prod /= 2;
		}
		prod = (prod % 100000)* f;
	}
	return ( int )( prod % 10);
}
```

Data Structure

Segment Tree :

```
/*
Problem : LOJ 1183 ( Computing Fast Average) .
*/
struct tree{
	int sum,fl;
};
tree m[4*maxm];
int n,q;

int gcd(int a,int b){
	if(a%b==0) return b;
	return gcd(b,a%b);
}
```

```
 int query(int node,int b,int e,int x,int y){

	if(b==e){
		return m[node].sum;
	}

	int k=e-b+1,l;
	if(b==x&&e==y){
		return m[node].sum;
	}

	int left,right,mid,ret=0;

	left=node<<1; right=left+1; mid=b+e; mid/=2;

	if(m[node].fl==1){
		l=m[node].sum/k; m[node].fl=0;
		update(left,b,mid,b,mid,l);
		update(right,mid+1,e,mid+1,e,l);
	}

	if(y<=mid){
		return query(left,b,mid,x,y);
	}
	else if(x>mid){
		return query(right,mid+1,e,x,y);
	}
	else{
		ret+=query(left,b,mid,x,mid);
		ret+=query(right,mid+1,e,mid+1,y);
	}

	return ret;
}

void update(int node,int b,int e,int x,int y,int v){

	if(b==e){
		m[node].fl=0;
		m[node].sum=v;
		return ;
	}

	int k=e-b+1,l;
	if(b==x&&e==y){
		m[node].sum=k*v; m[node].fl=1;
		return ;
	}

	int left,right,mid;
```

```
         left=node<<1; right=left+1; mid=b+e; mid/=2;

        if(m[node].fl==1){
                l=m[node].sum/k; m[node].fl=0;
                update(left,b,mid,b,mid,l);
                update(right,mid+1,e,mid+1,e,l);
        }

        if(y<=mid){
                update(left,b,mid,x,y,v);
        }
        else if(x>mid){
                update(right,mid+1,e,x,y,v);
        }
        else{
                update(left,b,mid,x,mid,v);
                update(right,mid+1,e,mid+1,y,v);
        }
        m[node].sum=m[left].sum+m[right].sum;
}

void init(int node,int b,int e){

        if(b==e){
                m[node].fl=m[node].sum=0;
                return ;
        }

        int left,right,mid;

        left=node<<1; right=left+1; mid=b+e; mid/=2;

        init(left,b,mid);
        init(right,mid+1,e);

        m[node].fl=m[node].sum=0;
}
```

LCA (Lowest Common Ancestor) :

```
/*
Algo    : LCA O(sqrt) per query..
Problem: Min-Max Roads (light oj )
*/

#define maxm 100010
#define inf (1<<28)

struct node{
```

```
      int min1,max1;
    node(){}
    node(int a,int b){ min1=a; max1=b;}
};

// n=no of node, nr = sqrt of max heigt.....
int n,nr,T[maxm],L[maxm],costT[maxm],P[maxm],costP1[maxm],costP2[maxm];
// v for storing graph , w =weight of the edge ......
vector<int>v[maxm],w[maxm];

int mini(int a,int b){
    if(a<b) return a; return b;
}
int maxi(int a,int b){
    if(a>b) return a; return b;
}
// for calculate P .........
void dfs(int node,int val1,int val2);
// for calculate L and T.
void dfs1(int s,int lev,int pre);
// finding LCA .....
node lca(int x,int y);

int main(){

    int i,j,k,l,test,t=1;

   //freopen("in.txt","r",stdin);

    scanf("%d",&test);

    while(test--){

        scanf("%d",&n);

        for(i=0;i<=n;i++){
            v[i].clear();w[i].clear();
        }

        for(i=1;i<n;i++){
            scanf("%d %d %d",&k,&l,&j);
            v[k].push_back(l); w[k].push_back(j);
            v[l].push_back(k); w[l].push_back(j);
        }

        nr=0;
        dfs1(1,0,1);
        // fix height sqrt
        k=sqrt(nr); if(k*k!=nr) k++; nr=k;
```

```
         dfs(1,0,0);

        int q;
        scanf("%d",&q);
        printf("Case %d:\n",t++);
        for(i=1;i<=q;i++){
            scanf("%d %d",&k,&l);
            node ans=lca(k,l);
            printf("%d %d\n",ans.min1,ans.max1);
        }


    }

    return 0;
}
node lca(int x,int y){

    node ret=node(inf,-inf);

    while(P[x]!=P[y]){
        if(L[x]>L[y]){
            ret.min1=mini(ret.min1,costP1[x]);
            ret.max1=maxi(ret.max1,costP2[x]);
            x=P[x];
        }
        else{
            ret.min1=mini(ret.min1,costP1[y]);
            ret.max1=maxi(ret.max1,costP2[y]);
            y=P[y];
        }
    }
    while(x!=y){
        if(L[x]>L[y]){
            ret.min1=mini(ret.min1,costT[x]);
            ret.max1=maxi(ret.max1,costT[x]);
            x=T[x];
        }
        else{
            ret.min1=mini(ret.min1,costT[y]);
            ret.max1=maxi(ret.max1,costT[y]);
            y=T[y];
        }
    }

    // Lca node = x
       return ret;

}
```

```
  //val1= min cost of edge;
//val2= max cost of edge;
void dfs(int node,int val1,int val2){

    int i,j,k,l;

    if(L[node]<nr) P[node]=1;
    else{
        if(!(L[node]%nr)){
                     val1=val2=costT[node];
            P[node]=T[node];
        }
        else{
            P[node]=P[T[node]];
        }
    }
    costP1[node]=val1;
    costP2[node]=val2;

    for(i=0;i<v[node].size();i++){
        k=v[node][i];
        if(L[k]<=L[node]) continue;
        dfs(k,mini(val1,w[node][i]),maxi(val2,w[node][i]));
    }
}

void dfs1(int s,int lev,int pre){

    int i,j,k,l;
    T[s]=pre;
    L[s]=lev;
    nr=maxi(nr,lev);

    for(i=0;i<v[s].size();i++){
        k=v[s][i]; if(k==pre) continue;
        costT[k]=w[s][i];
        dfs1(k,lev+1,s);
    }

}
```

BIT :

```
/*
Author : Rashedul Hasan Rijul
Algo   : BIT
*/

#define ii long long int
```

```
  #define mod 1000000009
#define maxval 262150

struct BIT{

    ii tree[maxval+100];
    ii n;

    void init(int n1){
        n=n1;
        memset(tree,0,sizeof(tree));
    }
    void clear(){
        memset(tree,0,sizeof(tree));
    }

    ii read(int idx){
        ii sum=0;
        while(idx>0){
            sum+=tree[idx];
            idx-=(idx& -idx);
            sum%=mod;
        }
        return sum;
    }
    void update(int idx ,int val){

        while (idx <= n){
            tree[idx] += val;
            tree[idx]%=mod;
            idx += (idx & -idx);
        }
    }
    ii read(int beg,int end){
        ii ret=read(end)-read(beg-1);
        if(ret<0) ret+=mod;
        return ret;
    }
};


// BIT  finish ......

BIT bit;
```

2-D Bit :

```
/*
Author  : Rashedul Hasan Rijul (silent_coder).
```

```
  Problem : LOJ 1266 ( points in rectangle ).
Algo    : 2-D bit
*/

#define maxm 1010
#define max_v 1005
int tree[maxm][maxm];
bool fl[maxm][maxm];

void updatey(int x,int y,int val){
       while(y<=max_v){
              tree[x][y]+=val;
              y+=(y & -y);
       }
}
void updatex(int x,int y,int val){
       while(x<=max_v){
              updatey(x,y,val);
              x+=(x & -x);
       }
}
int ready(int x,int y){
       int ret=0;
       while(y>0){
              ret+=tree[x][y];
              y-=(y & -y);
       }
       return ret;
}
int readx(int x,int y){
       int ret=0;
       while(x>0){
              ret+=ready(x,y);
              x-=(x & -x);
       }
       return ret;
}

int main(){

       int i,j,k,l,k1,l1,test,t=1,q,ans;

       //freopen("in.txt","r",stdin);

       scanf("%d",&test);

       while(test--){

              memset(tree,0,sizeof(tree));
              memset(fl,0,sizeof(fl));
```

```
                  scanf("%d",&q);
                printf("Case %d:\n",t++);
                for(i=1;i<=q;i++){
                        scanf("%d",&j);
                        if(j==0){
                                scanf("%d %d",&k,&l);
                                k++; l++;
                                if(fl[k][l]) continue;
                                fl[k][l]=1;
                                updatex(k,l,1);
                        }
                        else{
                                scanf("%d %d %d %d",&k,&l,&k1,&l1);
                                k++; l++; k1++; l1++;
                                ans=readx(k1,l1);
                                ans-=readx(k1,l);
                                ans-=readx(k,l1);
                                ans+=readx(k,l);
                                for(j=k;j<=k1;j++){
                                        if(fl[j][l]) ans++;
                                }
                                for(j=l;j<=l1;j++){
                                        if(fl[k][j]) ans++;
                                }
                                if(fl[k][l]) ans--;
                                //ans-=readx(k,l-1);
                                //ans+=readx(k-1,l-1);
                                printf("%d\n",ans);
                        }
                }

        }

        return 0;
}
```

Heavy-Light Decomposition :

```
/*
Problem : Spoj - Query on a tree
*/
#define maxm 200100
#define lg_maxm 20

struct tree{
    int mx_cost;
};
tree seg_T[4*maxm];

int n,m;
```

```
  vector<int>G[maxm],W[maxm],edge_ind[maxm];
int L[maxm],T[maxm],P[maxm][lg_maxm],subtree_size[maxm];
int edge[maxm],ptr;

int chain_head[maxm],chain_ind[maxm],chain_no;
int pos_in_base[maxm],base_arr[maxm];

// fixing parent,size and level
void dfs(int s,int pre,int lev){

    T[s]=pre;
    L[s]=lev;
    subtree_size[s]=1;

    for(int i=0;i<G[s].size();i++){
        if(G[s][i]==pre) continue;
        edge[edge_ind[s][i]]=G[s][i];
        dfs(G[s][i],s,lev+1);
        subtree_size[s]+=subtree_size[G[s][i]];
    }

}

// Updating sparse table for lca . . .
void init_sparse(){

    int i,j;
    for(i=0;i<=n;i++){
        for(j=0;(1<<j)<n;j++){
            P[i][j]=-1;
        }
    }

    // the first ancestor ..
    for(i=1;i<=n;i++){
        P[i][0]=T[i];
    }

    // sparse table ..
    for(j=1;(1<<j)<n;j++){
        for(i=1;i<=n;i++){
            if(P[i][j-1]!=-1){
                P[i][j]=P[P[i][j-1]][j-1];
            }
        }
    }
}

/*
* Actual HL-Decomposition part
* Initially all entries of chainHead[] are set to -1.
* So when ever a new chain is started, chain head is correctly assigned.
* As we add a new node to chain, we will note its position in the baseArray.
* In the first for loop we find the child node which has maximum sub-tree size.
* The following if condition is failed for leaf nodes.
* When the if condition passes, we expand the chain to special child.
* In the second for loop we recursively call the function on all normal nodes.
* chainNo++ ensures that we are creating a new chain for each normal child.
*/
void heavy_light(int s,int pre,int curr_cost){

    if(chain_head[chain_no]==-1){
```

```
            chain_head[chain_no]=s; // Assign chain head
        }

        chain_ind[s]=chain_no;
        pos_in_base[s]=++ptr; // Position of this node in baseArray which we will use in Segtree
        base_arr[ptr]=curr_cost;

        int heavy_child=-1,heavy_cost=0,heavy_size=0,i;

        // Loop to find heavy child
        for(i=0;i<G[s].size();i++){
            if(G[s][i]==pre) continue;
            if(subtree_size[G[s][i]]>heavy_size){
                heavy_size=subtree_size[G[s][i]];
                heavy_child=G[s][i];
                heavy_cost=W[s][i];
            }
        }

        if(heavy_child!=-1){
            // Expand the chain
            heavy_light(heavy_child,s,heavy_cost);
        }

        for(i=0;i<G[s].size();i++){
            if(G[s][i]==pre || G[s][i]==heavy_child) continue;
            // light node . . . .
            chain_no++;
            heavy_light(G[s][i],s,W[s][i]);
        }

}

void init_segtree(int node,int b,int e){

        if(b>e) return ;
        if(b==e){
            seg_T[node].mx_cost=base_arr[b];
            return ;
        }

        int left=node<<1,right=left+1,mid=b+e;
        mid/=2;

        init_segtree(left,b,mid);
        init_segtree(right,mid+1,e);

        seg_T[node].mx_cost=maxi(seg_T[left].mx_cost,seg_T[right].mx_cost);
}

int seg_query(int node,int b,int e,int k,int l){

        if(b>e) return 0;

        if(b==k && e==l) return seg_T[node].mx_cost;

        int left=node<<1,right=left+1,mid=b+e;
        mid/=2;

        if(l<=mid) return seg_query(left,b,mid,k,l);
        else if(k>mid) return seg_query(right,mid+1,e,k,l);
        else{
```

```
        return maxi(seg_query(left,b,mid,k,mid),seg_query(right,mid+1,e,mid+1,l));
    }
}

void seg_update(int node,int b,int e,int ind,int v){

    if(b>e) return ;

    if(b==e){
        seg_T[node].mx_cost=v;
        return ;
    }

    int left=node<<1,right=left+1,mid=b+e;
    mid/=2;

    if(ind<=mid)  seg_update(left,b,mid,ind,v);
    else        seg_update(right,mid+1,e,ind,v);

    seg_T[node].mx_cost=maxi(seg_T[left].mx_cost,seg_T[right].mx_cost);
}


int lca(int p,int q){

     int log, i;

     //if p is situated on a higher level than q then we swap them
     if (L[p] < L[q])
         swap(p,q);

      //we compute the value of [log(L[p)]
      for (log = 1; 1 << log <= L[p]; log++);
      log--;

      //we find the ancestor of node p situated on the same level
      //with q using the values in P
      for (i = log; i >= 0; i--)
          if (L[p] - (1 << i) >= L[q])
              p = P[p][i];

      if (p == q)
          return p;

      //we compute LCA(p, q) using the values in P
      for (i = log; i >= 0; i--)
          if (P[p][i] != -1 && P[p][i] != P[q][i])
              p = P[p][i], q = P[q][i];

      return T[p];

}

/*
* query_up:
* It takes two nodes u and lc, condition is that lc is an ancestor of u
* We query the chain in which u is present till chain head, then move to next chain up
* We do that way till u and lc are in the same chain, we query for that part of chain and break
*/
int query_up(int u,int lc){

    if(u==lc) return 0;
```

```
      int u_chain,lc_chain,ret=-1;

    lc_chain=chain_ind[lc];
    while(1){

        if(u==lc) break;

        u_chain=chain_ind[u];

        if(u_chain==lc_chain){
           // Both u and lc are in the same chain, so we need to query from u to lc, update ret and
break.
           // We break because we came from u up till v, we are done
            ret=maxi(ret,seg_query(1,1,ptr,pos_in_base[lc]+1,pos_in_base[u]));

            return ret;
        }

        ret=maxi(ret,seg_query(1,1,ptr,pos_in_base[chain_head[u_chain]],pos_in_base[u]));
        // Above is call to segment tree query function. We do from chainHead of u till u. That is
the whole chain from
        // start till head. We then update the answer

        u=chain_head[u_chain]; // move u to u's chainHead
        u=T[u]; //Then move to its parent, that means we changed chains
    }

    return ret;
}


int query(int u,int v){

    int lc=lca(u,v);

    int ret=maxi(query_up(u,lc),query_up(v,lc));

    return ret;
}

void update(int u,int v){

    seg_update(1,1,ptr,pos_in_base[u],v);
}


int main(){

    int i,j,k,l,test,t=1;

    //freopen("in.txt","r",stdin);
    //freopen("out.txt","w",stdout);

    scanf("%d",&test);

    while(test--){

        scanf("%d",&n);

        // init_all
        ptr=0,chain_no=1;
        for(i=0;i<=n;i++){
```

```
            G[i].clear();
          W[i].clear();
          edge_ind[i].clear();
          chain_head[i]=-1;
      }


      for(i=1;i<n;i++){
          scanf("%d %d %d",&k,&l,&j);

          G[k].push_back(l);
          W[k].push_back(j);
          edge_ind[k].push_back(i);

          G[l].push_back(k);
          W[l].push_back(j);
          edge_ind[l].push_back(i);
      }

      // init . . .
      dfs(1,1,1);
      init_sparse();
      heavy_light(1,1,0);

      // seg-tree
      init_segtree(1,1,ptr);

      char qs[10];
      int u,v;
      while(1){
          scanf("%s",qs);
          if(qs[0]=='D') break;

          scanf("%d %d",&u,&v);
          if(qs[0]=='Q'){
              printf("%d\n",query(u,v));
          }
          else{
              u=edge[u];
              update(u,v);
          }
      }
  }

  return 0;}
```

String Algorithm

KMP:

S= string , p =pattern.

```
int kmp(){
        int i,j,k,l,ret=0;
        int n,m;
        n=strlen(s); m=strlen(p);
        prefix();
        k=0;
```

```
	for(i=1;i<=n;i++){
		while(k>0 && s[i-1]!=p[k]) k=pre[k];
		if(s[i-1]==p[k]) k++;
		if(k==m){
			ret++;
			k=pre[k];
		}
	}

	return ret;
}

void prefix(){
	int i,j,k,l;
	l=strlen(p);
	pre[1]=0;
	k=0;

	for(i=2;i<=l;i++){
		while(k>0 && p[k]!=p[i-1]) k=pre[k];
		if(p[k]==p[i-1]) k++;
		pre[i]=k;
	}

}
```

Aho-corasick :

```
/*
Problem : Beaver (Codeforces Round 71 - problem C ).
Algo    : Aho corasick , DP , Trie
*/
#define maxm 100100
#define inf (1<<29)
int maxi(int a,int b){
    if(a>b) return a;
    return b;
}
int mini(int a,int b){
    if(a<b) return a;
    return b;
}
/////****  Trie + Aho Corasick  ********///////

//  maxc= query...
#define maxc 15
//  maxl = length of query  string...
#define maxl 20
//  maxn =required trie  node ....
```

```
  #define maxn ((maxc*maxl)+10)
#define cn 64

struct trie{
    //vector<int>v; // for keeping track of patterns ends here ...
    int edges[cn+3],ind;
    int pat_no;  // highest lenght pattern ends at this node...
    int pat_len; // minimum lenght pattern ends at this node...
};
trie Tri[maxn],root;
int len[maxc]; // len[i]= length of pattern i ...
int tot,f[maxn],n,pos[maxc]; //  f=failure , pos[i]=position of ith pattern in
trie node .
int c[maxn]; // c[i] = (number of occurence) count of ith node of trie in
string s .
int fin[maxm]; // fin[i] = minimum lenght of pattern finish at pos i of string
s

int getid(char ch){
    if(ch>='a' && ch<='z') return ch-'a';
    else if(ch>='A' && ch<='Z') return ch-'A'+26;
    else if(ch>='0' && ch<='9') return ch-'0'+52;
    if(ch=='_') return cn-1;
    return ch-'a';
}

void init(trie *a,int ind){
    a->ind=ind;
    a->pat_no=0;
    a->pat_len=inf;
       //a->v.clear();
    memset(a->edges,-1,sizeof(a->edges));
}

void add(trie *a,char *s,int ind){

    int i,l,id;
    l=strlen(s);

    for(i=0;i<=l;i++){
        if(i==l){
            pos[ind]=a->ind;
            a->pat_no=ind;
            a->pat_len=mini(a->pat_len,len[ind]);
            //a->v.push_back(ind);
            continue;
        }
        id=getid(s[i]);
        if(a->edges[id]==-1){
            a->edges[id]=tot;
```

```
        init(&Tri[a->edges[id]],tot++);
      }
      a=&Tri[a->edges[id]];
    }

}
void build(){
    int i,j,piv;
    trie *a=&Tri[0];
    for(i=0;i<=cn;i++){
        if(a->edges[i]==-1) a->edges[i]=0;
    }
    //  Failure  Function .........
    queue<int>q;
    for(i=0;i<=cn;i++){
        if(a->edges[i]){
            f[a->edges[i]]=0;
            q.push(a->edges[i]);
        }
    }
    while(!q.empty()){
        int state=q.front(); q.pop();

              a=&Tri[state];

              //sort(a->v.begin(),a->v.end());
              //unique(a->v.begin(),a->v.end());

        for(i=0;i<=cn;i++){
            if(a->edges[i]==-1) continue;
            int failure=f[state];
            while(Tri[failure].edges[i]==-1){
                failure=f[failure];
            }
            failure=Tri[failure].edges[i];
            piv=Tri[state].edges[i];
            f[piv]=failure;

            Tri[piv].pat_len=mini(Tri[piv].pat_len,Tri[failure].pat_len);
            /*
            trie *a1=&Tri[failure];
            trie *a2=&Tri[piv];
            for (int ind=0; ind<a1->v.size(); ind++){
                a2->v.push_back(a1->v[ind]);
            }
            */
                    q.push(piv);
        }
    }
}
```

```
  void match(char *s);

//////**************   END  *************/////

char s[maxm];
char pat[maxl];
// Problem depenedent.....
int can[maxm]; // can[i] = minimum index of string s from pos i for which it
is valid ...
int dp[maxm];  // dp[i]  = store minimum index for which string S obtaining
from s[dp[i]] to s[i] is valid .
int main(){

    int i,j,k,l,test,t=1;
    scanf("%s",s);
    scanf("%d",&n);
    //Init.......
    c[0]=0;
    root=Tri[0];
    init(&Tri[0],tot++);
    for(i=1;i<=n;i++){
        scanf("%s",pat);
        len[i]=strlen(pat);
        //printf("%s\n",pat);
        add(&Tri[0],pat,i);
    }
    build(); // building automata
    int m;
    match(s); // match string

    queue<int>q;
    vector<int>v;
    q.push( 0 );
    while( !q.empty()){
        int now = q.front();
        q.pop();
        v.push_back(now);
        for( i=0;i<=cn;i++ ){
            if( Tri[now].edges[i]!=-1 && Tri[now].edges[i]!=0 )
q.push(Tri[now].edges[i]);
        }
    }
    for( i=v.size()-1;i>=0;i-- ){
        c[f[v[i]]] += c[v[i]];
    }
    /*

    for(i=1;i<=n;i++){
        printf("%d\n",c[pos[i]]);
    }
```

```
    for(i=0;s[i];i++){
        //if(fin[i]==inf) fin[i]=-1;
        printf("%2d ",i);
    }
    puts("");
    */
    int ans=0,mark=0,ans1;
    for(i=0;s[i];i++){
        //if(fin[i]==inf) fin[i]=-1;
        can[i]=maxi(i-fin[i]+2,0);
        dp[i]=can[i];
        if(i) dp[i]=maxi(dp[i],dp[i-1]);
        ans1=i-dp[i]+1;
        if(ans1>ans){
            ans=ans1;
            mark=dp[i];
        }
        //printf("%2d ",can[i]);
    }
    //puts("");
    /*
    for(i=0;s[i];i++){
        //if(fin[i]==inf) fin[i]=-1;
        printf("%2d ",dp[i]);
    }
    puts("");

    for(i=0;s[i];i++){
        //if(fin[i]==inf) fin[i]=-1;
        printf("%2c ",s[i]);
    }
    puts("");
    */
    printf("%d %d\n",ans,mark);
    return 0;
}
int find_next(int curr,char ch){
    int id=getid(ch);
    //printf("curr =%d %c-%d\n",curr,ch,id);
    while(Tri[curr].edges[id]==-1) curr=f[curr];
    //printf("curr =%d %c-%d %d\n",curr,ch,id,Tri[curr].edges[id]);
    return Tri[curr].edges[id];
}
void match(char *s){
    int i,j,l;
    l=strlen(s);
    int curr=0;
    for(i=0;i<l;i++){
        curr=find_next(curr,s[i]);
```

```
            c[curr]++;
        fin[i]=Tri[curr].pat_len;
        /*
        trie *a=&Tri[curr];
        for (it=a->v.begin(); it!=a->v.end(); ++it){
            c[*it]++;
        }
        */
    }
}
```

Suffix Array :

```
const int MAXN = 2005;
const int MAXL = 22;
int n ,stp,mv,suffix[MAXN],tmp[MAXN];
int sum[MAXN],cnt[MAXN],rank[MAXL][MAXN];
char str[MAXN];
int LCP(int u,int v){
        int ret=0,i;
        for(i = stp; i >= 0; i--){
                if(rank[i][u]==rank[i][v]){
                        ret += 1<<i;
                        u += 1<<i;
                        v += 1<<i;
                }
        }
        return ret;
}
bool equal(int u,int v){
        if(!stp)return str[u]==str[v];
        if(rank[stp-1][u]!=rank[stp-1][v]) return false;
        int a = u + mv < n ? rank[stp-1][u+mv] : -1;
        int b = v + mv < n ? rank[stp-1][v+mv] : -1;
        return a == b ;
}
void update(){
        int i;
        for(i = 0;i < n; i ++) sum[ i ] = 0;

        int rnk = 0;
        for(i = 0;i < n;i++){
                suffix[ i ] = tmp[ i ];
                if( i&&!equal(suffix[i],suffix[i-1])){
                        rank[stp][suffix[i]]=++rnk;
                        sum[rnk+1]=sum[rnk];
                }
                else rank[stp][suffix[i]]=rnk;
                sum[rnk+1]++;
        }
```

```
  }
void Sort(){
       int i;
       for(i = 0; i < n; i ++ ) cnt[ i ] = 0;
       memset(tmp,-1,sizeof tmp);
       for(i = 0 ; i < mv; i ++){
              int idx = rank[ stp - 1 ][ n-i-1 ];
              int x = sum[ idx ];
              tmp[ x + cnt[ idx ] ] = n-i-1;
              cnt[ idx ]++;
       }
       for(i = 0;i < n; i ++ ){
              int idx = suffix[ i ] - mv;
              if(idx<0)continue;
              idx = rank[stp-1][idx];
              int x = sum[ idx ];
              tmp[ x + cnt[ idx ] ] = suffix[ i ] - mv;
              cnt[idx]++;
       }
       update();
       return;
}
bool cmp(const int &a,const int &b){
       if(str[a]!=str[b]) return str[a]<str[b];
       return false;
}
int main(){
       scanf("%d", &n);
       scanf ( "%s", str  );
       int i;
       for(i = 0;i < n;i++) tmp[ i ] = i ;

       sort(tmp,tmp+n,cmp);
       stp = 0;
       update();
       ++stp;
       for( mv = 1; mv < n;  mv <<= 1){
              Sort();
              stp++;
       }
       stp--;
       for(i = 0;i<=stp; i++)      rank[ i ][ n ] = -1;
       int res=0;
       for(i = 1; i < n; i ++)
              res=max(res,LCP(suffix[i],suffix[i-1]));
       printf("%d\n",res);
       return 0;
}
```

Manachar's algorithm:

```
/*
Manacher algorithm implementation.
Application, largest palindromic substring, largest palindromic suffix
*/
int lengths[MAX<<1];
int manacher(char *buff, int len) {
    int i, k, pallen, found, d, j, s, e;
    k = pallen = 0;
    for(i = 0; i < len; ) {
        if(i > pallen && buff[i-pallen-1] == buff[i]) {
            pallen += 2, i++;
            continue;
        }
        lengths[k++] = pallen;
        s = k - 2, e = s - pallen, found = 0;
        for(j = s; j > e; j--) {
            d = j - e - 1;
            if(lengths[j] == d) {
                pallen = d;
                found = 1;
                break;
            }
            lengths[k++] = (d < lengths[j]? d : lengths[j]);
        }
        if(!found) { pallen = 1; i++; }
    }
    lengths[k++] = pallen;
    return lengths[k-1];
}
```

Miscellaneous

Fast Reader :

```
// Fast....... reader.............

const int BUFFSIZE = 10240;
char BUFF[BUFFSIZE + 1], *ppp = BUFF;
int RR, CHAR, SIGN, BYTES = 0;
#define GETCHAR(c) { \
        if(ppp-BUFF==BYTES && (BYTES==0 || BYTES==BUFFSIZE)) { BYTES =
fread(BUFF,1,BUFFSIZE,stdin); ppp=BUFF; } \
        if(ppp-BUFF==BYTES && (BYTES>0 && BYTES<BUFFSIZE)) { BUFF[0] = 0;
ppp=BUFF; } \
        c = *ppp++; \
```

```
  }

#define DIGIT(c) (((c) >= '0') && ((c) <= '9'))
#define MINUS(c) ((c)== '-')
#define GETNUMBER(n) { \
        n = 0; SIGN = 1; do { GETCHAR(CHAR); } while(!(DIGIT(CHAR) ||
MINUS(CHAR))); \
        if(MINUS(CHAR)) { SIGN = -1; GETCHAR(CHAR); } \
        while(DIGIT(CHAR)) { n = 10*n + CHAR-'0'; GETCHAR(CHAR); } if(SIGN ==
-1) { n = -n; } \
}
//////////*****************************//////////
```

Knight Distance (infinite Board) :

```
/*
NK is the size of grid you want to precalculate
NK/2,NK/2 will be considered origin
Calculates minimum knight distance from 0,0 to x,y
*/
const int KN = 101;
i64 dk[KN][KN];
int dx[] = {-1, -1, 1, 1, -2, -2, 2, 2};
int dy[] = {-2, 2, -2, 2, -1, 1, -1, 1};
void precalc() {
    int x, y, x1, y1, i;
    queue< int > Q;
    memset(dk, 0x3f, sizeof dk);
    x = y = (KN >> 1);
    dk[x][y] = 0;
    Q.push(x); Q.push(y);
    while(!Q.empty()) {
        x = Q.front(); Q.pop();
        y = Q.front(); Q.pop();
        for(i = 0; i < 8; i++) {
            x1 = x + dx[i], y1 = y + dy[i];
            if(0 <= x1 && x1 < KN && 0 <= y1 && y1 < KN) {
                if(dk[x1][y1] > dk[x][y] + 1) {
                    dk[x1][y1] = dk[x][y] + 1;
                    Q.push(x1); Q.push(y1);
                }
            }
        }
    }
}
i64 knight(i64 x, i64 y) {
    i64 step, res = 0;
    if(x < y) swap(x, y);
    while((x<<1) > KN) {
        step = x / 2 / 2; res += step;
```

```
        x -= step * 2; y -= step;
        if(y < 0) y = ((y % 2) + 2) % 2;
        if(x < y) swap(x, y);
    }
    res += dk[x+(KN>>1)][y+(KN>>1)];
    return res;
}
```

Compress a array :

```
int t[maxm];
int compress(int* a, int n){
  int i, m;
  for(i = 0; i < n; i++) t[i] = a[i];
  sort(t, t+n);
  m = unique(t,t+n)-t;
  for(i = 0; i < n; i++)
    a[i] = lower_bound(t,t+m,a[i])-t;
  return m;
}
```

Shank's Algorithm:

This algorithm finds **x** (0 <= x <= p - 2) **for** the equation

b = ax mod p where b, a, p are known

Using the fact that x can be expressed as jm + i, where 0 <= i <= m – 1, 0 <= j < p/m, and m = **ceil**(**sqrt**(p - 1))

So, the equation can be written as

b = amj+i mod p

b = amj ai mod p

ba-i = amj mod p

If two lists of ordered **pairs** (i, ba-i) **and** (j, amj), ordered by their second components are built, then it is possible to find one pair from each list that have equal second components. Then x = mj + i, where i and j are the first elements of the matching pairs.

Code:

```
/*
Shanks baby step giant step - discrete logarithm algorithm
for the equation: b = a^x % p where a, b, p known, finds x
works only when p is an odd prime
*/

int shank(int a, int b, int p) {
    int i, j, m;
    long long c, aj, ami;
    map< long long, int > M;
    map< long long, int > :: iterator it;
    m = (int)ceil(sqrt((double)(p)));
    M.insert(make_pair(1, 0));
```

```
      for(j = 1, aj = 1; j < m; j++) {
        aj = (aj * a) % p;
        M.insert(make_pair(aj, j));
    }
    ami = modexp(modinv(a, p), m, p);
    for(c = b, i = 0; i < m; i++) {
        it = M.find(c);
        if(it != M.end()) return i * m + it->second;
        c = (c * ami) % p;
    }
    return 0;
}
```

Negative Base:

```
string negaBase(int n,int b){
       int i,tmp;
       string a;
       for(i=0;n;i++){
              tmp=n%b; n=n/b;
              if(tmp<0) {   tmp+= (-b),   n++;          }
              a+='0'+tmp;
       }
       for(n=0;n<(i/2);n++) swap(a[n],a[i -n - 1]);
       if(i) return a;
       return "0";
}
```

Double Hashing :

```
/*
M > N and should be close, better both be primes.
M should be as much large as possible, not exceeding array size.
HKEY is the Hash function, change it if necessary.
*/

#define NIL -1
#define M 1021
#define N 1019
#define HKEY(x,i) ((x)%M+(i)*(1+(x)%N))%M

int a[M+1];

inline int hash(int key) {
    int i = 0, j;
    do {
        j = HKEY(key, i);
        if(a[j]==NIL) { a[j] = key; return j; }
```

```
        i++;
    } while(i < M);
    return -1;
}

inline int find(int key) {
    int i = 0, j;
    do {
        j = HKEY(key, i);
        if(a[j]==key) return j;
        i++;
    } while(a[j]!=NIL && i < M);
    return -1;
}
```

Joseph:

```
int joseph(int n,int k){
       if(n==1) return 0;
       return ((joseph(n-1,k)+k)%n);
}
```

GEO Template:

```
// Geometry Templates>>>>>>>>>>>
// Header File
//Macro........
// Structure....

// distance between point to point...
inline double distancepp( point a, point b ) {
    return sqrt( ( a.x - b.x ) * ( a.x - b.x ) + ( a.y - b.y ) * ( a.y - b.y )
);
}
// distance between point to point...
inline double distancepp( point3D a, point3D b ) {
    return sqrt( ( a.x - b.x ) * ( a.x - b.x ) + ( a.y - b.y ) * ( a.y - b.y )
+ ( a.z - b.z ) * ( a.z - b.z ) );
}
// square distance between point to point.
inline double sq_distance( point a, point b ) {
    return ( a.x - b.x ) * ( a.x - b.x ) + ( a.y - b.y ) * ( a.y - b.y );
}
// distance between point to Line....
inline double distancepl( point P, line L ) {
```

```
    return fabs( L.a * P.x + L.b * P.y + L.c ) / sqrt( L.a * L.a + L.b * L.b
);
}
//  cross product = p0p1 * p0p2..
inline double cross( point p0, point p1, point p2 ) {
    return( ( p1.x - p0.x ) * ( p2.y - p0.y ) - ( p2.x - p0.x ) * ( p1.y -
p0.y ) );
}
// cross product
inline double cross(point p1, point p2 ) {
    return( ( p1.x * p2.y ) - ( p2.x *  p1.y ) );
}

//Intersection - Line, Line:
inline bool intersection( line L1, line L2, point &p ) {
       double det = L1.a * L2.b - L1.b * L2.a;
       if( eq ( det, 0 ) ) return false;
       p.x = ( L1.b * L2.c - L2.b * L1.c ) / det;
       p.y = ( L1.c * L2.a - L2.c * L1.a ) / det;
       return true;
}

//Intersection - Segment, Segment:
inline bool intersection( segment L1, segment L2, point &p ) {
       if( !intersection( line( L1.A, L1.B ), line( L2.A, L2.B ), p) ) {
              return false; // can lie on another, just check their equations,
and check overlap
       }
       return(eq(distancepp(L1.A,p)+distancepp(L1.B,p),distancepp(L1.A,L1.B))
&&

       eq(distancepp(L2.A,p)+distancepp(L2.B,p),distancepp(L2.A,L2.B)));
}
//Perpendicular Line of a Given Line Through a Point:
inline line findPerpendicularLine( line L, point P ) {
       line res; //line perpendicular to L, and intersects with P
       res.a = L.b, res.b = -L.a;
       res.c = -res.a * P.x - res.b * P.y;
       return res;
}
//Distance - Point, Segment:
inline double distanceps( point P, segment S ) {
       line L1 = line(S.A,S.B), L2; point P1;
       L2 = findPerpendicularLine( L1, P );
       if( intersection( L1, L2, P1 ) )
              if( eq ( distancepp( S.A, P1 ) + distancepp( S.B, P1 ),
distancepp( S.A, S.B ) ) )
                     return distancepl(P,L1);
       return mini ( distancepp( S.A, P), distancepp( S.B, P) );
}
```

```
  //  area of  polygon.........
double areaPoly(point P[],int n){
    double area=0;
    for( int i = 0, j = n - 1; i < n; j = i++ ) area += P[j].x * P[i].y -
P[j].y * P[i].x;
    return fabs(area)*.5;
}
//  intersecting point between  circle and line...
inline bool intersectioncl(circle C,line L,point &p1,point &p2) {
       if( distancepl( C.center, L ) > C.r + eps ) return false;
       double a, b, c, d, x = C.center.x, y = C.center.y;
       d = C.r*C.r - x*x - y*y;
       if( eq( L.a, 0) ) {
              p1.y = p2.y = -L.c / L.b;
              a = 1;
              b = 2 * x;
              c = p1.y * p1.y - 2 * p1.y * y - d;
              d = b * b - 4 * a * c;
              d = sqrt( fabs (d) );
              p1.x = ( b + d ) / ( 2 * a );
              p2.x = ( b - d ) / ( 2 * a );
       }
       else {
              a = L.a *L.a + L.b * L.b;
              b = 2 * ( L.a * L.a * y - L.b * L.c - L.a * L.b * x);
              c = L.c * L.c + 2 * L.a * L.c * x - L.a * L.a * d;
              d = b * b - 4 * a * c;
              d = sqrt( fabs(d) );
              p1.y = ( b + d ) / ( 2 * a );
              p2.y = ( b - d ) / ( 2 * a );
              p1.x = ( -L.b * p1.y -L.c ) / L.a;
              p2.x = ( -L.b * p2.y -L.c ) / L.a;
       }
       return true;
}

//Find Points that are r1 unit away from A, and r2 unit away from B:
inline bool findpointAr1Br2(point A,double r1,point B, double r2,point
&p1,point &p2) {
       line L;
       circle C;
       L.a = 2 * (B.x - A.x );
       L.b = 2 * (B.y - A.y );
       L.c = A.x * A.x + A.y * A.y - B.x * B.x - B.y * B.y + r2 * r2 - r1 *
r1;
       C.center = A;
       C.r = r1;
       return intersectioncl( C, L, p1, p2 );
}
```

```
  //Intersection Area between Two Circles:
inline double intersectionArea2C( circle C1, circle C2 ) {
       C2.center.x = distancepp( C1.center, C2.center );
       C1.center.x = C1.center.y = C2.center.y = 0;
       if( C1.r < C2.center.x - C2.r + eps ) return 0;
       if( -C1.r + eps > C2.center.x - C2.r ) return pi * C1.r * C1.r;
       if( C1.r + eps > C2.center.x + C2.r ) return pi * C2.r * C2.r;
       double c, CAD, CBD, res;
       c = C2.center.x;
       CAD = 2 * acos( (C1.r * C1.r + c * c - C2.r * C2.r) / (2 * C1.r * c) );
       CBD = 2 * acos( (C2.r * C2.r + c * c - C1.r * C1.r) / (2 * C2.r * c) );
       res=C1.r * C1.r * ( CAD - sin( CAD ) ) + C2.r * C2.r * ( CBD - sin (
CBD ) );
       return .5 * res;
}

//Circle Through Thee Points:
circle CircleThrough3points( point A, point B, point C) {
       double den; circle c;
       den = 2.0 *((B.x-A.x)*(C.y-A.y) - (B.y-A.y)*(C.x-A.x));
       c.center.x =( (C.y-A.y)*(B.x*B.x+B.y*B.y-A.x*A.x-A.y*A.y)- (B.y-
A.y)*(C.x*C.x+C.y*C.y-A.x*A.x-A.y*A.y) );
       c.center.x /= den;
       c.center.y =( (B.x-A.x)*(C.x*C.x+C.y*C.y-A.x*A.x-A.y*A.y) - (C.x-
A.x)*(B.x*B.x+B.y*B.y-A.x*A.x-A.y*A.y) );
       c.center.y /= den;
       c.r = distancepp( c.center, A );
       return c;
}

// Rotating a Point anticlockwise by 'theta' radian w.r.t Origin:
inline point rotate2D( point P ,double theta) {
       point Q;
       Q.x = P.x * cos( theta ) - P.y * sin( theta );
       Q.y = P.x * sin( theta ) + P.y * cos( theta );
       return Q;
}


double ang(point a,point b,point c){     //returns angle <bac
    double absq = sq_distance(a , b);
    double bcsq = sq_distance(c , b), acsq = sq_distance(a , c);
    double cosp = (absq+acsq - bcsq)/(2.0*sqrt(absq * acsq) );
    return acos(cosp);
}
//  radian  to  degree.
double convrd(double theta){
    double ret=180; ret/=pi; return ret*theta;
}
// degree to radian...
```

```
  double convdr(double theta){
    double ret=pi; ret/=(double)180.0; return ret*theta;
}


//  check whether a  point lies inside a quad....
bool inside_quad(quad q,point p){

       double val=cross(q.p[0],q.p[1],p)*cross(q.p[3],q.p[2],p);
    if(val>0) return 0;

       val=cross(q.p[0],q.p[3],p)*cross(q.p[1],q.p[2],p);
    if(val>0) return 0;

       return 1;
}
//  check whether a point lies inside a segment....
bool inside_segment(segment S,point P){
    if( eq ( distancepp( S.A, P ) + distancepp( S.B, P ), distancepp( S.A, S.B
) ) )   return 1;
    return 0;
}

//  calculate  slope.......
double inline cal_slope(point p1,point p2){
    double num,den;
    num=p2.y-p1.y;
    den=p2.x-p1.x;
    if(iseq(den,0)) return inf;
    return num/den;
}

bool sort_x(point a,point b){
    if(iseq(a.x,b.x)) return a.y<b.y;
    return a.x<b.x;
}
bool sort_y(point a,point b){
    if(iseq(a.y,b.y)) return a.x<b.x;
    return a.y<b.y;
}

// newly added ..  ( need  modification..... ) ..........

bool is_square(segment l[5]){

    if(!iseq(sq_distance(l[0].A,l[0].B),sq_distance(l[1].A,l[1].B))) return 0;
    if(!iseq(sq_distance(l[1].A,l[1].B),sq_distance(l[2].A,l[2].B))) return 0;
    if(!iseq(sq_distance(l[2].A,l[2].B),sq_distance(l[3].A,l[3].B))) return 0;
    if(!iseq(sq_distance(l[3].A,l[3].B),sq_distance(l[0].A,l[0].B))) return 0;
```

```
      point p[5];
    p[0]=l[0].A; p[1]=l[0].B; p[2]=l[1].B; p[3]=l[2].B;

    int i,j,k;
    double com=pi;
    for(i=0,j=3;i<4;j=i++){
        k=i+1; k%=4;
        double val=ang(p[i],p[j],p[k]);
        val*=2.0;
        if(!iseq(val,pi)) return 0;
    }

  return 1;
}

bool is_rect(segment l[5]){

  if(!iseq(sq_distance(l[0].A,l[0].B),sq_distance(l[2].A,l[2].B))) return 0;
  if(!iseq(sq_distance(l[1].A,l[1].B),sq_distance(l[3].A,l[3].B))) return 0;

    point p[5];
    p[0]=l[0].A; p[1]=l[0].B; p[2]=l[1].B; p[3]=l[2].B;

    int i,j,k;
    double com=pi;
    for(i=0,j=3;i<4;j=i++){
        k=i+1; k%=4;
        double val=ang(p[i],p[j],p[k]);
        val*=2.0;
        if(!iseq(val,pi)) return 0;
    }
  return 1;
}

//  check  parallelogram............
bool is_para(segment l[5]){

  if(!iseq(sq_distance(l[0].A,l[0].B),sq_distance(l[2].A,l[2].B))) return 0;
  if(!iseq(sq_distance(l[1].A,l[1].B),sq_distance(l[3].A,l[3].B))) return 0;

    point p[5];
    p[0]=l[0].A; p[1]=l[0].B; p[2]=l[1].B; p[3]=l[2].B;

    int i,j,k;
    double com=pi;
    for(i=0,j=3;i<4;j=i++){
        k=i+1; k%=4;
        double val=ang(p[i],p[j],p[k]);
        val*=2.0;
        if(iseq(val,pi)) return 0;
```

```
        }
    return 1;
}

bool is_rhombus(segment l[5]){

    if(!iseq(sq_distance(l[0].A,l[0].B),sq_distance(l[1].A,l[1].B))) return 0;
    if(!iseq(sq_distance(l[1].A,l[1].B),sq_distance(l[2].A,l[2].B))) return 0;
    if(!iseq(sq_distance(l[2].A,l[2].B),sq_distance(l[3].A,l[3].B))) return 0;
    if(!iseq(sq_distance(l[3].A,l[3].B),sq_distance(l[0].A,l[0].B))) return 0;

       point p[5];
       p[0]=l[0].A; p[1]=l[0].B; p[2]=l[1].B; p[3]=l[2].B;

       int i,j,k;
       double com=pi;
       for(i=0,j=3;i<4;j=i++){
              k=i+1; k%=4;
              double val=ang(p[i],p[j],p[k]);
              val*=2.0;
              if(iseq(val,pi)) return 0;
       }

    return 1;
}

//  check  trapezium............
bool is_trap(segment l[5]){

       point p[5];
       p[0]=l[0].A; p[1]=l[0].B; p[2]=l[1].B; p[3]=l[2].B;

       int ans1=0,ans2=0;

       if(iseq(cross(point(p[1].x-p[0].x,p[1].y-p[0].y),point(p[2].x-
p[3].x,p[2].y-p[3].y)),0.0)) ans1=1;
    if(iseq(cross(point(p[1].x-p[2].x,p[1].y-p[2].y),point(p[0].x-
p[3].x,p[0].y-p[3].y)),0.0)) ans2=1;
    return ans2^ans1;
}
// convert spherical to cartesian co-ordinate......
void sph_to_cartesian(double R,double lat,double lng,point3D &p){

    lat=convdr(lat);
    lng=convdr(lng);

    p.x=R*sin(lat)*cos(lng);
    p.y=R*sin(lat)*sin(lng);
    p.z=R*cos(lat);
}
```

```
  // convert longitude/latitude to cartesian co-ordinate......
void earth_to_cartesian(double R,double lat,double lng,point3D &p){

     lat=convdr(lat);
     lng=convdr(lng);

     p.x=R*cos(lat)*cos(lng);
     p.y=R*cos(lat)*sin(lng);
     p.z=R*sin(lat);}
// geo  template   end>>>>>
```

Recently Added:

Meet in the middle + Ternary Mask:

```
struct ternary{
    ii pow3[maxm];

    void init(){
        init(maxm-1);
    }
    void init(int n){
        pow3[0]=1;
        for(int i=1;i<=n;i++){
            pow3[i]=pow3[i-1]*3;
        }
    }
    int get_bit(int mask,int k){
        ii tmp=mask; tmp/=pow3[k];
        return (tmp%3);
    }
    int set_bit(int mask,int k,int v){
        ii tmp=mask;
        tmp/=pow3[k];
        tmp%=3;
        mask-=(tmp*pow3[k]);
        mask+=(v*pow3[k]);
        return mask;
    }
};
ternary t_mask;

int n,req;
int a[maxm],b[maxm];

set<int>can_set;

int build(int mask,int a[],int n,int flag){

    int ret=0;

    for(int i=0;i<n;i++){
                  int bit_val=t_mask.get_bit(mask,i);

                  for(int j=1;j<=bit_val;j++){
                           ret+=a[i];
                           if(ret>req) return -1;
                  }
```

```
      }

         if(flag) can_set.insert(ret);
    return ret;
}

int main(){

    int i,j,k,l,test,t=1;

    t_mask.init();

    scanf("%d",&test);

    while(test--){

                 can_set.clear();

        scanf("%d %d",&n,&req);

        int n1=n/2,n2=n-n1;

        for(i=0;i<n1;i++){
            scanf("%d",&a[i]);
        }
                 for(i=n1,j=0;i<n;i++,j++){
                          scanf("%d",&b[j]);
                 }


        int tot=0;
        for(i=0;i<t_mask.pow3[n1];i++){
            build(i,a,n1,1);
        }

                 bool soln_found=false;
        for(i=0;i<t_mask.pow3[n2];i++){
            int now=build(i,b,n2,0);
                          if(now==-1) continue;
                          now=req-now;
                          if(can_set.find(now)!=can_set.end()){
                                   soln_found=true;
                                   break;
                          }
        }

        printf("Case %d: ",t++);
         if(soln_found==true){
            printf("Yes\n");
        }
        else{
            puts("No");
        }
    }
    return 0;
}
```

Treap :

```
  /*
Algo        : Treap (Balanced BST).
Problem     : Spoj - Yet another range difference query.
*/

//  TREAP  >>>>>>>>>>>>>>>>>>>>>>>>>

typedef int treap_type;

struct node{

    treap_type value,min_val,max_val,diff;
    ii priority;
    int cnt;
    node *left,*right;

    node(){}
    node(treap_type _value){
        cnt=1;
        value=min_val=max_val=_value;
        diff=inf;
        priority=rand();
        left=right=NULL;
    }
};

// 1-based . . . . . . . . . .
struct treap{

    node *root;

    void fix(node * &t){
        if(t==NULL) return ;
        t->cnt=get_count(t->left)+get_count(t->right)+1;
                  t->diff=inf;

        if(t->left){
            t->min_val=t->left->min_val;
            t->diff=mini(t->left->diff,t->value - t->left->max_val);
        }
                  else t->min_val=t->value;

        if(t->right){
            t->max_val=t->right->max_val;
            t->diff=mini( t->diff,mini( t->right->diff , t->right->min_val- t->value) );
        }
                  else t->max_val=t->value;
    }

    inline int get_count(node* t){
        return t ? t->cnt : 0;
    }

    inline void left_rotate(node* &t){
        node* tmp = t->left;
        t->left = tmp->right;
        tmp->right = t;
        t = tmp;
    }

    inline void right_rotate(node* &t){
        node* tmp = t->right;
```

```
        t->right = tmp->left;
      tmp->left = t;
      t = tmp;
  }

  bool insert(node * &t,treap_type value){

      if(t==NULL){
          t=new node(value);
          fix(t);
          return true;
      }

      if(t->value==value) return false;
      bool ret;

      if(value < t->value) ret=insert(t->left,value);
      else                 ret=insert(t->right,value);

      if(t->left && t->left->priority > t->priority){
          left_rotate(t);
      }
      else if(t->right && t->right->priority > t->priority){
          right_rotate(t);
      }

      if(t->left) fix(t->left);
      if(t->right) fix(t->right);
      fix(t);

      return ret;
  }

  bool insert(treap_type value){
      return insert(root,value);
  }

  inline ii get_priority(node* t){
      return t ? t->priority : -1;
  }

  bool erase(node* &t, treap_type val){

      if(!t) return false;

      bool ret;
      if(t->value != val){
          ret=erase(val < t->value ? t->left : t->right, val);
      }
      else{
          if(!t->left && !t->right){
              delete t;
              t = NULL;
          }else{
              if(get_priority(t->left) < get_priority(t->right))
                  right_rotate(t);
              else
                  left_rotate(t);

              ret=erase(t, val);
          }
      }
```

```
        if(t){
            if(t->left)  fix(t->left);
            if(t->right) fix(t->right);
            fix(t);
        }

        return ret;
    }

    bool erase(treap_type value){
                 return erase(root,value);
    }

    node* find(node * t,int value){

        if(t==NULL) return NULL;

                 node *ret;
        if(value > t->value){
            ret=find(t->right,value);
                          if(ret==NULL) return t;
            return ret;
        }
        else if(value==t->value){
            return t;
        }
        else{
            return find(t->left,value);
        }
    }

    node* find(int value){
        return find(root,value);
    }

    treap_type find_max(int beg,int end){
        if(beg>end) swap(beg,end);
        return 0;
    }

    treap_type find_min(int beg,int end){
        if(beg>end) swap(beg,end);
        return 0;
    }
};

// END >>>>>>>>>>>>>>>>>>

treap tree;

ii find_min(node *t,int left,int beg,int end){

    if(t==NULL) return inf;

    ii r_left=left;
    ii r_right=left+t->cnt-1;

    ii sz=tree.get_count(t->left)+1;
         ii curr_ind=left+tree.get_count(t->left);

    if(beg<=r_left && end>=r_right){
```

```
            return t->diff;
    }

    ii ret=inf;

    if(beg>curr_ind)              ret=mini(ret,find_min(t->right,curr_ind+1,beg,end));
    else if(end<curr_ind)         ret=mini(ret,find_min(t->left,left,beg,end));

    else                          ret=mini(mini(find_min(t->left,left,beg,end)
                                         ,find_min(t->right,curr_ind+1,beg,end)),ret);

    if(curr_ind>beg && curr_ind<=end && t->left){
        ret=mini(ret,t->value - t->left->max_val);
    }
    if(curr_ind<end && curr_ind>=beg && t->right){
        ret=mini(ret,t->right->min_val - t->value);
    }

    return ret;
}

ii find_min(int k,int l){

    if(l<=k) return -1;
    return find_min(tree.root,1,k,l);
}

ii find_kth(node *t,int kth){

    if(tree.get_count(t)<kth){
                  return -1;
    }

    int sz=tree.get_count(t->left)+1;

    if(sz==kth){
        return t->value;
    }

    if(kth<sz){
        return find_kth(t->left,kth);
    }
    else{
        return find_kth(t->right,kth-sz);
    }
}

ii find_max(int k,int l){

    if(l<=k) return -1;
    return find_kth(tree.root,l)-find_kth(tree.root,k);
}


int find_count(node *t,int value){

         if(t==NULL) return 0;

         if(value==t->value)   return tree.get_count(t->left);
         else if(value > t->value)  return tree.get_count(t->left)
                                                                       +find_count(t-
>right,value)+1;
```

```
            else return find_count(t->left,value);
}

int n,m;
char type[5];

int main(){

    int i,j,k,l,test,t=1,val;

    //freopen("in.txt","r",stdin);
    //freopen("out.txt","w",stdout);

    scanf("%d",&test);

    while(test--){

        scanf("%s",type);

        //printf("%s\n",type);

        if(type[0]=='I'){
            scanf("%d",&val);
            tree.insert(val);
        }

        if(type[0]=='D'){
            scanf("%d",&val);
            tree.erase(val);
        }

        if(type[0]=='N'){
            scanf("%d %d",&k,&l);
            printf("%d\n",find_min(k+1,l+1));
        }

        if(type[0]=='X'){
            scanf("%d %d",&k,&l);
            printf("%d\n",find_max(k+1,l+1));
        }

    }

    return 0;
}
```

Suffix Automation :

```
/*
Algo        : Suffix-Automation.
Problem      : Codeforces 235c- Cyclical Quest.
*/

int n;
char s[maxm];

// Suffix-Automation
struct state {
         int len, link;
         bool suffix;
         ii count;
```

```
            map<char,int> next;
};

const int MAXLEN = maxm+2;
state st[MAXLEN*2];
int sz, last;
pair<int, int> sorter[MAXLEN * 2 + 10];

inline void sa_init() {
        sz = last = 0;
        st[0].len = 0;
        st[0].link = -1;
        st[0].count = 0;
        st[0].suffix=0;
        ++sz;
}

inline void sa_extend (char c) {

        int cur = sz++;
        st[cur].len = st[last].len + 1;
        st[cur].suffix=0;
        st[cur].count=1;

        int p;
        for (p=last; p!=-1 && !st[p].next.count(c); p=st[p].link)
                st[p].next[c] = cur;
        if (p == -1)
                st[cur].link = 0;
        else {
                int q = st[p].next[c];

                if (st[p].len + 1 == st[q].len)
                        st[cur].link = q;
                else {
                        int clone = sz++;
                        st[clone].len = st[p].len + 1;
                        st[clone].next = st[q].next;
                        st[clone].link = st[q].link;

                        for (; p!=-1 && st[p].next[c]==q; p=st[p].link)
                                st[p].next[c] = clone;
                        st[q].link = st[cur].link = clone;
                }
        }
        last = cur;
}
//  Suffix-Automation End. ..

void post_process(){

    int i;

    for(i=0;i<sz;i++){
        sorter[i]=mp(st[i].len,i);
    }
    sort(sorter,sorter+sz);

    for(i=sz-1;i>=0;i--){
        int ind=sorter[i].vv;
        st[st[ind].link].count+=st[ind].count;
    }
```

```
    }

vector<pii>ans;
int pre[maxm];


void failure(char *p){

         int i,j,k,l;

         l=strlen(p);
         pre[1]=0;
         k=0;

         for(i=2;i<=l;i++){
                  while(k>0 && p[k]!=p[i-1]) k=pre[k];
                  if(p[k]==p[i-1]) k++;
                  pre[i]=k;
         }

}


ii cal(char *s,int lim){

    int i;

    int curr_st=0,len=0;
    ii ret=0;

    failure(s);

    /*for(i=0;s[i];i++){
        printf("%d ",pre[i+1]);
    }
    puts("");
    */

    for(i=0;s[i];i++){

        while (curr_st && !st[curr_st].next.count(s[i])) {
            curr_st = st[curr_st].link;
            len=st[curr_st].len;
        }

        if (st[curr_st].next.count(s[i])) {
            curr_st = st[curr_st].next[s[i]];
            len++;
        }
        while(st[st[curr_st].link].len>=lim){
            curr_st=st[curr_st].link;
        }

        if(len>=lim && pre[i+1]<lim){
            ret+=st[curr_st].count;
        }
    }

    return ret;
}

char tmp[maxm];
```

```
int main(){

    int i,j,k,l;

    //freopen("in.txt","r",stdin);
    //freopen("out.txt","w",stdout);

    scanf("%s",s);

    sa_init();

    for(i=0;s[i];i++){
        sa_extend(s[i]);
    }

    post_process();

    int q;
    int len;

    scanf("%d",&q);

    for(i=1;i<=q;i++){
        scanf("%s",tmp);
        len=strlen(tmp);
        strcpy(s,tmp);
        for(j=len,k=0;k<len-1;k++,j++){
            s[j]=tmp[k];
        }
        s[j]=0;
        //puts(s);
        printf("%I64d\n",cal(s,len));
    }

    return 0;
}
```

IDA*:

```
int ida_star(int puzzle[maxm][maxm]){

    int i,j;
    node root;

    for(i=1;i<=4;i++){
        for(j=1;j<=4;j++){
            root.puzzle[i][j]=puzzle[i][j];
        }
    }

    curr_node=root;

    int bound=mini(50,heuristic());

    solution="";
    while(true){
        curr_node=root;
        pii zero=find_pos(root.puzzle,0);
        int next_bound=ida_search(zero,bound,0);
        if(next_bound<=bound) return 1;
```

```
            next_bound=mini(55,next_bound);
        //if(next_bound<=bound) break;
        bound=next_bound;
    }

    return 1;
}

int ida_search(pii pos_zero,int bound,int d){

    int f=heuristic();
    if(f+d>bound) return f+d;

    if(!f){
        if(solution.size()==0 || solution.size()>d+1){
            soln[d]=0;
            solution=soln;
        }
        return f;
    }

    int ret=-1,x,y;
    for(int i=0;i<4;i++){
        if(d && soln[d-1]==move[3-i]){
            continue;
        }
        x=dirx[i],y=diry[i];
        pii pos=pos_zero;
        int ret1=ida_search(new_pos,bound,d+1);
        if(!ret1) return ret1;
         // move
         if(ret==-1) ret=ret1;
         ret=mini(ret,ret1);
         // reverse move

    }
    return ret;
}

int heuristic(){
    int i,j,ret=0;

    return ret;
}
```

Part -2: Prepared By Fahim Ahmed (xlr8)

```
/*<algorithm> handling*/
#include <bits/stdc++.h>
using namespace std;
int main()
{
        vector<int>V;

        V.push_back(23);
        V.push_back(21);
        V.push_back(53);
        V.push_back(53);
        V.push_back(53);
        V.push_back(59);
        V.push_back(25);
        V.push_back(48);

        //Pre-condition of Binary search is to sort the vector
        sort(V.begin(),V.end());
```

```
          for(int i=0;i<V.size();i++)
            cout<<V[i]<<" ";
        cout<<endl;
        if(binary_search(V.begin(),V.end(),53))
            cout<<"Found"<<endl;
        else
            cout<<"Not Found"<<endl;

        if(binary_search(V.begin(),V.end(),10))
            cout<<"Found"<<endl;
        else
            cout<<"Not Found"<<endl;

        //lower bound
        vector<int>::iterator it;

        it=lower_bound(V.begin(),V.end(),55);
        int pos=it-V.begin();
        int indx=pos-1;
        cout<<" Lowe bound of 55 is at position="<<pos<<" "<<V[indx]<<endl;

        //Upper bound
        it=upper_bound(V.begin(),V.end(),50);
        pos=it-V.begin();
        indx=pos-1;
        cout<<"Upper bound of 50 is at position="<<pos<<" "<<V[indx]<<endl;

        //Next permutation
        char arr[] = "ABC";
        do{
            puts(arr);
        }while(next_permutation(arr,arr+3));
        return 0;
}
```

```
/*Testing Divisibility of BIGINTEGERS USING Strings */
#include <bits/stdc++.h>
using namespace std;
#define li long long int

int main()
{
    string integer;
    int T;
    li mod;
    scanf("%d ", &T);
    for(int t = 1; t <= T; t++)
    {
        cin>>integer; scanf("%lld", &mod);
        li res = 0;
        if(integer[0] >= '0' && integer[0] <= '9')
        {
            for(int i = 0; i < integer.size(); i++)
            {
```

```
                  res = (res*10 + (integer[i] -  48)) % mod;
            }
        }
        else{
            for(int i = 1; i < integer.size(); i++)
            {
                res = (res*10 + (integer[i] -  48)) % mod;
            }
        }
        if(res == 0)
        {
            printf("Case %d: divisible\n", t);
        }
        else{
            printf("Case %d: not divisible\n", t);
        }
    }
    return 0;
}
```

```
/*Generic Binary  Search Tree*/
#include <bits/stdc++.h>
using namespace std;

template <typename T> struct node{
    T value;
    node *parent, *left, *right;
};

template <typename Tp> class BinTree
{
    node <Tp> *root;

public:
    BinTree(Tp rootvalue)
    {
        root = new node<Tp>;
        root->value = rootvalue;
        root->parent = root->left = root->right = NULL;
    }

    inline node<Tp>* getroot()
    {
        return root;
    }

    inline void setvalue(node<Tp> *temp, Tp key)
    {
        if(temp->value == key) return;
```

```
        else if(key < temp->value)
        {
            if(temp->left != NULL)
            {
                setvalue(temp->left, key);
                return;
            }
            else{
                node <Tp> *newleft = new node<Tp>;
                newleft->value = key;
                newleft->left = newleft->right = NULL;
                newleft->parent = temp;
                temp->left = newleft;
                return;
            }
        }
        else if(key > temp->value)
        {
            if(temp->right != NULL)
            {
                setvalue(temp->right, key);
                return;
            }
            else{
                node <Tp> *newright = new node<Tp>;
                newright->value = key;
                newright->left = newright->right = NULL;
                newright->parent = temp;
                temp->right = newright;
                return;
            }
        }
    }
    inline void setvalue(Tp key)
    {
        setvalue(root, key);
    }
    void Search(Tp key)
    {
        node<Tp>* temp = root;

        while(temp != NULL)
        {
            if(temp->value == key)
            {
                if(temp == root)
                {
                    cout<<key<<" is found in the ROOT\n";
                    return;
                }
                else if(temp->parent->left == temp)
                {
                    cout<<key<<" is found as the LEFT child of "<<temp->parent-
>value<<endl;
```

```
                        return;
                }
                else if(temp->parent->right == temp)
                {
                    cout<<key<<" is found as the RIGHT child of "<<temp->parent-
>value<<endl;
                    return;
                }
            }
            else if( key < temp->value)
            {
                temp = temp->left;
            }
            else if(key > temp->value)
            {
                temp = temp->right;
            }
        }
        cout<<key<<" is not in the TREE\n";
        return;
    }

    inline void inorder(node<Tp>* temp)
    {
        if(temp->left != NULL) inorder(temp->left);

        cout<<temp->value<<" ";

        if(temp->right != NULL) inorder(temp->right);

        return;
    }

    inline void preorder(node<Tp> *temp)
    {
        cout<<temp->value<<" ";

        if(temp->left != NULL) preorder(temp->left);

        if(temp->right != NULL) preorder(temp->right);

        return;
    }

    inline void postorder(node<Tp> *temp)
    {
        if(temp->left != NULL) postorder(temp->left);

        if(temp->right != NULL) postorder(temp->right);

        cout<<temp->value<<" ";
        return;
    }
```

```
    inline Tp minimum()
    {
        node<Tp>* temp = root;

        while(temp->left != NULL)
        {
            temp = temp->left;
        }

        return temp->value;
    }

    inline Tp maximum()
    {
        node<Tp>* temp = root;

        while(temp->right != NULL)
        {
            temp = temp->right;
        }

        return temp->value;
    }
    inline Tp prenode_inorder(Tp key)
    {
        node<Tp>* pointer = root;

        while(1)
        {
            if(pointer->value == key) break;
            else if(key < pointer->value)
            {
                pointer=pointer->left;
            }
            else if(key > pointer->value)
            {
                pointer = pointer->right;
            }
        }
        if(pointer->left != NULL)
        {
            pointer = pointer->left;
        }
        else
        {
            printf("NO PRENODE FOUND\n");
            return '\0';
        }
        while(1)
        {
            if(pointer -> right != NULL)
            {
                pointer = pointer->right;
            }
```

```
            else break;
        }
        return pointer->value;
    }
    inline Tp nextnode_inorder(Tp key)
    {
        node<Tp>* pointer = root;

        while(1)
        {
            if(pointer->value == key) break;
            else if(key < pointer->value)
            {
                pointer=pointer->left;
            }
            else if(key > pointer->value)
            {
                pointer = pointer->right;
            }
        }
        if(pointer->right != NULL)
        {
            pointer = pointer->right;
        }
        else
        {
            printf("NO PRENODE FOUND\n");
            return '\0';
        }
        while(1)
        {
            if(pointer -> left != NULL)
            {
                pointer = pointer->left;
            }
            else break;
        }
        return pointer->value;
    }

    inline Tp nextnode_preorder(Tp key)
    {
        node<Tp>* pointer = root;

        while(1)
        {
            if(pointer->value == key) break;
            else if(key < pointer->value)
            {
                pointer=pointer->left;
            }
            else if(key > pointer->value)
            {
                pointer = pointer->right;
```

```
            }
        }
        if(pointer->left != NULL)
        {
            return pointer->left->value;
        }
        else if(pointer->parent->right != NULL && pointer->parent->right != pointer)
        {
            return pointer->right->value;
        }
        else {
            printf("NO NEXTNODE in PREORDER FOUND\n");
            return NULL;
        }
    }
    inline Tp nextnode_postorder(Tp key)
    {
        node<Tp>* pointer = root;
        while(1)
        {
            if(pointer->value == key) break;
            else if(key < pointer->value)
            {
                pointer=pointer->left;
            }
            else if(key > pointer->value)
            {
                pointer = pointer->right;
            }
        }
        if(pointer == root)
        {
            cout<<"NO NEXTNODE FOUND IN POSTORDER\n";
            return NULL;
        }
        else if(pointer == pointer->parent->left)
        {
            if(pointer->parent->right != NULL)
            {
                node<Tp> *current = pointer->parent->right;
                while(1)
                {
                    if(current->left != NULL) current=current->left;
                    else if(current->right != NULL) current = current->right;
                    else break;
                }
                return current->value;
            }
            else{
                return pointer->parent->value;
            }
        }
        else{
            return pointer->parent->value;
```

```
        }
    }
};
```

/*Breadth fast search- cormen*/

```
#include <bits/stdc++.h>
using namespace std;
#define INF 2147483647

struct node{
    vector <int> adjacent;
    string color = "white";
    int depth = INF;
    int parent = NULL;
    int value;
};

void BFS(node *vertices, int source)
{
    vertices[source].color = "gray";
    vertices[source].depth = 0;
    queue <int> serial;
    serial.push(source);
    while(serial.empty() == false)
    {
        int u = serial.front();
        serial.pop();
        for(int i = 0; i < vertices[u].adjacent.size(); i++)
        {
            int v = vertices[u].adjacent[i];
            if(vertices[v].color == "white")
            {
                    vertices[v].color = "gray";
                    vertices[v].depth = vertices[u].depth+1;
                    vertices[v].parent = u;
                    serial.push(v);
            }
        }
        vertices[u].color = "black";
        cout<<vertices[u].value<<" ";
    }
    cout<<endl;
}


int main()
{
    int N;
    cout<<"TOTAL NODES = ";
    cin >> N;
    node vertices[N+1];
    int temp,source;
    cout<<"Which node is the root? ";
```

```
        cin >>source;
    cout<<"ROOTVALUE ? ";
    cin>>temp;
    vertices[source].value = temp;

    cout<<"PLEASE ENTER THE VALUE FOR THE REST "<<N-1<<" NODES:\n";
    for(int i = 1; i <= N; i++)
    {
        if(i == source) continue;

        int value; cout<<"VALUE for NODE-"<<i<<" : "; cin>>value;
        vertices[i].value = value;
    }

    int e;
    cout<<"TOTAL EDGES = ";
    cin >>e;

    for(int i = 1; i <= e; i++)
    {
        int x,y;
        cout<<"FOR EDGE "<<i<<endl;
        cout<<"BEGINNING NODE: ";
        cin>>x;
        cout<<"ENDING NODE: ";
        cin>>y;
        vertices[x].adjacent.push_back(y);
        vertices[y].adjacent.push_back(x);
    }

    BFS(vertices, source);
    return 0;
}

/*Test-case:
for a tree like this:

                            5
                           / \
                         /     \
                       /         \
                    6              1
                    /\            /\
                 /     \        /    \
              2         3     4       7
            /
          /
        8

              the total nodes = 8;
              root node = 5;
              total edges = 7;
              and edges are between: (5, 6), (5,1), (6,2), (6,3), (1,4), (1,7),
(2,8);
```

```
                 The level wise travarsel for this tree would be: 5 6 1 2 3 4 7 8

*/

/*Complex STL usage*/
#include <bits/stdc++.h>
using namespace std;

int main()
{
    map<string, list < int > > person;

    person["Fahim"].push_back(1);
    person["Fahim"].push_back(2);
    person["Fahim"].push_back(3);

    cout<<person["Fahim"].front()<<endl;
    cout<<person["Fahim"].back()<<endl;


}

                             /***DYNMAMIC PROGRAMMING***/
/*4 state knapsack from facebook hacker cup round1 */
#include <bits/stdc++.h>
using namespace std;

int n;
int gc,gf,gp;
int p[25],c[25],f[25];
 /*20 ta maximum menu, proti tay p,c,f er sum at most 1000*/
int p_memo[25][1001],c_memo[25][1001],f_memo[25][1001];




inline void reset()
{
    for(int i = 0; i < 25; i++)
    {
        for(int j = 0; j  <1001; j++)
        {
            p_memo[i][j] = -1;
            c_memo[i][j] = -1;
            f_memo[i][j] = -1;
        }
    }
}

inline int knapsack_calc(int serial, int p_be4, int c_be4, int f_be4)
{
    if(serial > n)
    {
```

```
            /*er mane already shob gula set check kora hoye gese, ekhon jodi bag e thaka p_be4 ba
        current protein er weight, carbohydrate er weight ar fat er weight ta gp,gc, gf er equal hoy
        tar mane shob gula set milaye gp,gc,gf milano possible, hence truth value ta hobe 1*/
        if(p_be4 == gp && c_be4 == gc && f_be4 == gf)
        {
            return 1;
        }
        else
        {
            return 0;
        }
    }
    /*jodi prottektar khetrei dekha jay je ei previos weight obosthay ei same serial er khetre
    oi serial er set ta neya jabe ki jabe na ta agei  ekbar decide kora hoisilo
    tahole notun kore hisab na kore table theke value ta niye boshaye dilei hobe*/
    else if(p_memo[serial][p_be4] != -1 && c_memo[serial][c_be4] != -1 && f_memo[serial][f_be4] != -1)
    {
        return p_memo[serial][p_be4];
    }

    /*serial tomo set ta niye ekta truthvalue pabo, na niye porer ta niye arekta truthvalue pabo,
    2 tai hishab korar jonno 2 ta variable nilam*/
    int trthval_with_i = 0, trthval_without_i = 0;

    /*jodi dekhi je knapsack er already thaka weight er sahthe serial tomo set er
    element gula add korle p,c,f 3 tai capacity er moddhe thake tahole amra dekhbo
    eita niye shamne agaile ki truthvalue paowa jay*/
    if(p_be4+p[serial] <= gp && c_be4+c[serial] <= gc && f_be4+f[serial] <= gf)
    {
        trthval_with_i = knapsack_calc(serial+1, p_be4+p[serial], c_be4+c[serial], f_be4+f[serial]);
    }

    /*eki shathe serial tomo set ta na niye er porer set ta niye agaile finally value ta ki ashe amra sheitao
    dekhbo*/
    trthval_without_i = knapsack_calc(serial+1, p_be4, c_be4, f_be4);

    /*final decision hobe obosshoi 2 ta truthvalue er moddhe better ta, mane finally serial tomo ta ke niye
    ba na niye shesh porjonto gp,gc,gf given set theke milano jay kina eitai set kora hobe final_truth
    er value te, ja hobe 2 ta truth value er moddhe maximum ta
    othoba 2 ta truth er moddhe or || korle jei value ta paowa jabe sheita*/
    int final_truth = max(trthval_with_i,trthval_without_i);

    /*ei serial er ei previous weight niye asha set er jonno prapto truth value ta ke amra memo te rekhe dei*/
```

```
        p_memo[serial][p_be4] = c_memo[serial][c_be4]=f_memo[serial][f_be4] =
final_truth;

    return final_truth;
}

int main()
{
    freopen("DP-2 Knapsack 4 state Hacker Cup input.txt","r",stdin);
    freopen("DP-2 Knapsack 4 state Hacker Cup output.txt", "w", stdout);
    int T,t;
    scanf("%d", &T);
    for(t = 1; t <= T; t++)
    {
        reset();
        scanf("%d %d %d", &gp, &gc, &gf);
        scanf("%d", &n);
        for(int i = 1; i <= n; i++)
        {
            scanf("%d %d %d", &p[i], &c[i], &f[i]);
        }

        /*amar quarry shuru hosse 1 no. serial er jinish ke niye jokhon p_be4 = 0,
c_be4 = 0, f_be4 = 0
        karon ekhono kono bostu e neya hoy nai knapsack e */
        if(knapsack_calc(1,0,0,0) == 1)
        {
        printf("Case #%d: yes\n",t);
        }

        else{
        printf("Case #%d: no\n",t);
        }
    }
    return 0;
}
```

```
/*This is a 2 state knapsack problem*/
#include <bits/stdc++.h>
using namespace std;

int knapsack_status[101][1001];
/*dhore nissi maximum weight 1000 er beshi hobe na
ar maximum element  100 tar beshi hobe na */
int capacity;
int cost[101],weight[101];  /*oi 100 ta object er jonno array jetay shobar cost ar
weight rakha hobe*/
int total_objcts;

inline int knapsack_calc(int i_to_calc, int w_before_i)
{
```

```
        if(i_to_calc > total_objcts)
    {
        return 0;
    }
    else if(knapsack_status[i_to_calc][w_before_i] != -1)
    {
        return knapsack_status[i_to_calc][w_before_i];
    }

    int profit_with_i = 0, profit_without_i = 0;

    /*jodi bag er ager ojon er shathe weight[i] jog korle ta capacity er theke beshi
    na hoy tahole oi obj er price ta profit + or porer bostu (i  +1) add korle profit
    joto hobe oitao profit with i er included*/

    if(w_before_i + weight[i_to_calc] <= capacity)
    {
        profit_with_i = cost[i_to_calc]  + knapsack_calc(i_to_calc + 1,
w_before_i+weight[i_to_calc]);
    }
    else
    {
        profit_with_i = 0;
    }

    /*eki shathe amra hishab kortisi oi ith object na niye direct (i+1)th  ta niye
dekhbo*/
    profit_without_i = knapsack_calc(i_to_calc+1,w_before_i);

    knapsack_status[i_to_calc][w_before_i] = max(profit_with_i,profit_without_i);
    return knapsack_status[i_to_calc][w_before_i];
}

inline void reset()
{
    for(int i = 0; i < 101; i++){
        for(int j = 0; j < 1001; j++){
            knapsack_status[i][j] = -1;}}}
int main()
{
    reset();
    printf("Total objects: ");
    scanf("%d", &total_objcts);
    printf("Max capacity: ");
    scanf("%d", &capacity);
    printf("Cost & Weight:\n");

    for(int i = 1; i <= total_objcts; i++)
    {
        scanf("%d %d", &cost[i], &weight[i]);
    }

    int profit = knapsack_calc(1,0);
    printf("%d", profit);
```

```
   }

/*using these DP technique in 2 ways we can find 2 things.
1. If we can make a certain amount of money with the use of the given coins in
coins[] array.
2. In how many ways that money can be made.
3. We can also do another thing, count the minimum number of coins made to make that
much money

***see Shafayet vaiya's book to get details.
*/
#include <bits/stdc++.h>
using namespace std;
#define li long long int

int coin[] = {1,2,5,10,50,100};
int maxcoins = 5;
int memo1[7][300000];
int memo2[7][300000];


int coinchange_if_can_be_made(int ith_coin, int sumbe4)
{
    if(ith_coin > maxcoins)
    {
        if(sumbe4 == 0) return 1;
        else return 0;
    }

    if(memo1[ith_coin][sumbe4] != -1) return memo1[ith_coin][sumbe4];

    int possibility_with_i = 0, possibility_without_i = 0;

    if(sumbe4 - coin[ith_coin] >= 0)
    {
        possibility_with_i = coinchange_if_can_be_made(ith_coin, sumbe4-
coin[ith_coin]);
    }

    possibility_without_i = coinchange_if_can_be_made(ith_coin + 1, sumbe4);

    memo1[ith_coin][sumbe4] = max( possibility_with_i, possibility_without_i );

    return memo1[ith_coin][sumbe4];
}

int coinchange_number_of_ways(int ith_coin, int sumbe4)
{
    if(ith_coin > maxcoins)
    {
        if(sumbe4 == 0) return 1;
        else return 0;
    }
```

```
        if(memo2[ith_coin][sumbe4] != -1) return memo2[ith_coin][sumbe4];

    int possibility_with_i = 0, possibility_without_i = 0;

    if(sumbe4 - coin[ith_coin] >= 0)
    {
        possibility_with_i = coinchange_number_of_ways(ith_coin, sumbe4-
coin[ith_coin]);
    }

    possibility_without_i = coinchange_number_of_ways( (ith_coin + 1), sumbe4);

    memo2[ith_coin][sumbe4] = possibility_with_i + possibility_without_i ;

    return memo2[ith_coin][sumbe4];
}

int main()
{
    memset(memo1, -1, sizeof(memo1));
    memset(memo2, -1, sizeof(memo1));

    int value_to_make;
    while(1)
    {
        scanf("%d", &value_to_make);
        if(value_to_make  <= 0) break;

        int possbile = coinchange_if_can_be_made(0,value_to_make);
        int ways = coinchange_number_of_ways(0, value_to_make);

        if(possbile == 1)
        {
            cout <<value_to_make<<" can be made in "<<ways<<" ways\n";
        }
        else{
            cout<<"Impossible\n";
        }
    }
}
```

/*To find out how many ways a value can be made from some given coins using each coin infinite amount of times*/

```
#include <bits/stdc++.h>
using namespace std;


int dp[10000];
vector <int> coins;

int main()
{
    coins.push_back(5);
    coins.push_back(1);
```

```
    coins.push_back(3);
    coins.push_back(2);

    int value;
    printf("Highest possible value: ");
    scanf("%d", &value);


    dp[0] = 1;
    for(int i = 0; i  < coins.size(); i++)
    {
        for(int j = coins[i]; j <= value; j++)
        {
            dp[j]+=dp[j-coins[i]];
        }
    }
    cout <<value <<" can be made in "<< dp[value]<<" ways\n";
}
```

/*To find out how many ways a value can be made from some given coins using each coin JUST ONCE*/

```
#include <bits/stdc++.h>
using namespace std;

int dp[10000];
vector <int> coins;

int main()
{
    coins.push_back(5); coins.push_back(1);
    coins.push_back(3); coins.push_back(2);

    int value;
    printf("Highest possible value: ");
    scanf("%d", &value);
    dp[0] = 1;
    for(int i = 0; i  < coins.size(); i++)
    {
        for(int j = value; j >= coins[i]; j--)
        {
            dp[j]+=dp[j-coins[i]];
        }
    }
    cout <<value <<" can be made in "<< dp[value]<<" ways\n";
}
```

/*lightoj - 1231 : coin change(I)*/
/*N coins would be given that we will be able to use at mose "times[i]" times (i = 1 to N, for all N coins)*/

```
#include <bits/stdc++.h>
using namespace std;
#define mod 100000007;

int main()
```

```
   {
    int dp[1005], coin[55], times[55];
    int T, n, value;
    scanf("%d", &T);
    for (int tc = 1; tc <= T; tc++)
    {
        scanf("%d %d", &n, &value);
        for (int i = 0; i < n; i++)
        {
            scanf("%d", &coin[i]);
        }
        for (int j = 0; j < n; j++)
        {
            scanf("%d", &times[j]);
        }
        memset(dp, 0, sizeof(dp));
        dp[0] = 1;

        for (int j = 0; j < n; j++)
        {
            for (int i = value; i >= 0; i--)
            {
                for (int k = 1; k <= times[j]; k++)
                {
                    if (i - k*coin[j] >= 0)
                    {
                        dp[i] += dp[i-coin[j]*k];
                        dp[i]%=mod;
                    }
                }
            }
        }
        /*to view the number of ways the other values can be made
        for(int i = 1; i <= value; i++) cout<<i <<" can be made "<<dp[i]<<" ways\n";
        */
        printf("Case %d: %d\n", tc, dp[value]);
    }
    return 0;
}
```

/*UVA- 11517, EXACT CHANGE*/
/*
EXPLANATION:

Use Dynamic Programming to determine whether there is a way to make a value V from the n bills.
Let dp[X] be the number of bills needed to make a value of X.
To fill in this DP table, first set dp[0] = 0 and set the rest to INFINITY.
For each bill with value C, update dp[v+C] = min(dp[v+C], dp[v]+1) for all value v where dp[v] is not INFINITY.
The answer is X and dp[X], where X >= P and dp[X] is not INFINITY and X is minimum.
To find such X, a simple iteration will do.

PSEUDOCODE:

```
   int dp[30001];
dp[0] = 0;
for (int i=1; i<=30000; i++)
    dp[i] = INFINITE;
for each coin C do
    for (int v = 30001 - C - 1; v >= 0; v--)
        if (dp[v] < INFINITE)
            dp[v+C] = min(dp[v+C], dp[v]+1);

*/
#include <bits/stdc++.h>
using namespace std;
#define INT_MAX 2147483647
#define MAX_POSSIBLE 10000
/*which is equal to 2^31 - 1; largest possible value for signed 32 bit integer*/


int main ()
{
    int testCase,bill,coinNumber, coins[102], dp[MAX_POSSIBLE + 10];
    scanf ("%d", &testCase);

    while ( testCase-- )
    {
        scanf ("%d", &bill);

        scanf ("%d", &coinNumber);

        for ( int i = 0; i < coinNumber; i++ )
        {
                scanf ("%d", &coins [i]);
        }

        for ( int i = 0; i < MAX_POSSIBLE + 10; i++ )
        {
             dp [i] = INT_MAX;
        }

        dp [0] = 0;
        for ( int i = 0; i < coinNumber; i++ )
        {
            /*MAX_POSSIBLE hosse maximum possible value for bill, hence amra
MAX_POSSIBLE porjonto dp table ready korbo*/
            for (int j = MAX_POSSIBLE; j >= 0; j-- )
            {
                /*initially loop ta analyze korle dekha jabe,
                when i == 0; only j = 0 er time e eshe she dekhbe
                dp[j] = dp[0] = 0;  whjch is != INT_MAX, && if 0+ coins[i] <=
MAX_POSSIBLE &&
                as all other dp[j] except dp[0] will be INT_MAX then,
                hence, dp[j + coins[i]] > dp[0] + 1, so tokhon for the first time
                dp table e value assigned hobe je
                **dp[j+coins[i]] = dp[j] + 1;
```

```
                   jeitar mane hoilo je (j + coins[i] ) taka banaite, amar j banaite
joto gula coin lagto tar
                theke just 1 ta coin beshi lagbe, taholei kono target value er jonno
minmum number of coins ber kora jabe
                dp[target] er theke
                */
                if ( dp [j] != INT_MAX && j + coins [i] <= MAX_POSSIBLE )
                {
                    dp [j + coins [i]] = min(( dp [j] + 1 ), dp [j + coins [i]]) ;
                }
            }
        }

        /*eitar mane hosse bill ba bill er closest greater value jeitar jonno minimun
number of notes/coins
        use kore value banaite amra jani oi value ta amra oi dp[bill/closest value]
ta coin diye pay korbo*/
        for ( int i = bill; i <= MAX_POSSIBLE; i++ )
        {
            if ( dp [i] != INT_MAX )
            {
                printf ("%d %d\n", i, dp [i]);
                break;
            }
        }
        /*jodi kisu print na hoy tar mane hosse oi value er ashe pashe kisu o minimum
number of coins diye
        ber kora possible na*/
    }
    return 0;
}


/*COIN CHANGE ITERATIVE VERSION -5
** Define C[j] to be the minimum number of coins we need to
make change for j cents.
** If we knew that an optimal solution for the problem of
making change for j cents used a coin of traceination di,
we would have:
C[j] = 1+C[j - di].

because, let c[4] banaite di = 2 takar ekta coin use korsi.. hence di taka ta to ekta
coin diyei banano jabe.. ar
baki [4-di] taka banaite koyta coin lage eita hishab kora lagbe
*/

#include <bits/stdc++.h>
using namespace std;
#define inf 2147483647

/*Using this We can store all the used coins in vector store. All these coins were
used to make the given value "value"*/
void trace_back_all(int *trace, int j, vector <int> *store)
{
```

```
      if(j > 0)
    {
        trace_back_all(trace, j - trace[j], store);
        store->push_back(trace[j]);
    }
}


int main()
{
   int value;
   scanf("%d", &value);

    int n; /*Number of coins*/
    scanf("%d", &n);

    int coins[n+1];
    for(int i = 1; i <= n; i++)
    {
        scanf("%d", &coins[i]);
    }

    int needed_coins[value+1];
    int trace[value+1]; /*trace[i] would keep trace of the largest coin that was used
to make the value i */

    for(int i = 0; i <= value; i++)
    {
        needed_coins[i] = inf;
    }

    needed_coins[0] = 0;

    for(int i = 1; i <= value; i++)
    {
        for(int j = 1; j <= n; j++)
        {
            if(i >= coins[j] && (1 + needed_coins[i - coins[j]] < needed_coins[i]))
            {
                needed_coins[i] = needed_coins[i - coins[j]] + 1;
                trace[i] = coins[j]; /* i value banaite coins[j] use kora hosse shei
trace rakha hosse*/
            }
        }
    }

    vector <int> store;
    trace_back_all(trace, value, &store); /*This would trace back for all the coins
that were used to make this value */

    cout<<needed_coins[value]<<endl;

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

```
            cout<<store[i]<<" ";
    }
    cout<<endl;
}
```

/*This problem illustrates basic LCS algorithm, to see the explanation, view Shafaet - DP 5*/

```
#include <bits/stdc++.h>
using namespace std;

char a[102],b[102];
int memo[101][101];

int lcs(int i, int j)
{
    if(i < 0 || j < 0) return 0;

    if(memo[i][j] != - 1) return memo[i][j];

    if(a[i] == b[j])
    {
        memo[i][j] = 1 + lcs(i-1, j-1);
        return memo[i][j];
    }
    else{
        memo[i][j] = max(lcs(i-1, j), lcs(i,j-1));
        return memo[i][j];
    }
}

int main()
{
    //freopen("in.txt", "r", stdin);
    int tcase = 0;
    while(1)
    {
        gets(a);
        if(a[0] == '#') break;

        gets(b);
        int lena = strlen(a);
        int lenb = strlen(b);
        tcase++;
        memset(memo, -1, sizeof(memo));
        int longest_common_subsequence = lcs(lena-1, lenb-1);
        printf("Case #%d: you can visit at most %d cities.\n", tcase,
longest_common_subsequence);
    }
    return 0;
}
```

/*Josephus problem with linked list*/

```
#include <bits/stdc++.h>
```

```
using namespace std;

struct node{
    int value;
    node *next = NULL;
    node *prev = NULL;
};

class doubly{
    node *head, *tail;
    int list_size;
public:
    doubly()
    {
        head = new node;
        tail = new node;
        head->next = head->prev = tail;
        tail->prev = tail->next = head;
        list_size = 0;
    }
    void insert_at_end(int key)
    {
        node *q = new node;
        q->prev = tail->prev;
        q->prev->next = q;
        q->next = tail;
        tail->prev = q;
        q->value = key;
        list_size++;
    }
    void delete_node(node *temp)
    {
        node *q = temp->next;
        q->prev = temp->prev;
        temp->prev->next = q;
        free(temp);
        list_size--;
    }
    int josephus(int distance)
    {
        node *start = head->next;
        while(list_size > 1)
        {
            int counter = 0;
            while(1)
            {
                if(start == tail) start = head->next;
                counter++;
                node *temp = start;
                start = start->next;
                if(counter == distance)
                {
                    delete_node(temp);
                    break;
```

```
                    }
            }
        }
        return head->next->value;
    }
};

int main()
{
    int TC,tcase;
    scanf("%d", &TC);

    for(tcase = 1; tcase <= TC; tcase++)
    {
        doubly a;
        int n,k;
        scanf("%d %d", &n, &k);
        for(int i = 1; i <= n; i++)
        {
            a.insert_at_end(i);
        }
        printf("Case %d: %d\n", tcase, a.josephus(k));
    }
    return 0;
}
```

```
/*Inverse Modulas, NCR, and Factorial*/
#include <bits/stdc++.h>
using namespace std;
#define li long long int
#define MOD 1000003

li facs[1000001];


li nCk( li n, li k )
{
    if (k > n)
    {
        return 0;
    }
    if (k * 2 > n)
    {
        k = n-k;
    }
    if (k == 0) return 1;

    li result = n;
    for( li i = 2; i <= k; ++i ) {
        result = (result*(n-i+1))/i;
        result = result%1000003;
    }
    return result;
}
```

```
li inv_modulo(li a, li b)
{
      li b0 = b, t, q;
      li x0 = 0, x1 = 1;
      if (b == 1) return 1;
      while (a > 1) {
            q = a / b;
            t = b, b = a % b, a = t;
            t = x0, x0 = x1 - q * x0, x1 = t;
      }
      if (x1 < 0) x1 += b0;
      return x1;
}

void factorial()
{
    li n = 1000000;
    facs[1] = 1;
    facs[0] = 0;
    li fac = 1;
    for(li i = 2; i <= n; i++)
    {
        fac = (fac*i) % MOD;
        facs[i] = fac;
    }
}

int main()
{
    factorial();
    li T,n,k;
    scanf("%lld", &T);
    for(li t = 1; t <= T; t++)
    {
        scanf("%lld %lld", &n, &k);
        if(k == 0)
        {
            printf("Case %lld: 1\n", t);
            continue;
        }

        li a = facs[n];
        li b = (facs[k] * facs[n-k]) % MOD;
        b = inv_modulo(b, MOD);
        a = (a * b) % MOD;
        printf("Case %lld: %lld\n", t, a);
    }
    return 0;
}
```

```
/*map and etc by badhon vaiya*/
#include <bits/stdc++.h>
#define pii pair <string,int>
using namespace std;


//map <char,pii > mp;



int main(){

    vector <int> v;
    v.push_back(1);
    v.push_back(4);
    v.push_back(5);
    v.push_back(7);

//  string s="asd";
//
//     mp['A']=make_pair(s,2);
//
//
//  map <char,pii> :: iterator it;
//
//
//  for(it=mp.begin();it!=mp.end();it++){
//
//     char ch=it->first;
//     pii x=it->second;
//
//     cout<<ch<<"  "<<x.first<<" "<<x.second<<endl;
//  }
//
//  cout<<mp.size()<<endl;


  vector <int> :: iterator it;
  it= upper_bound(v.begin(),v.end(),7);
  if(it!=v.end())
  {

     int indx=it-v.begin(); /*way to find the index of upper bound*/
     cout<<indx<<endl;
  }

  return 0;
}
/*MAP HANDLING*/
#include <bits/stdc++.h>
using namespace std;

int main()
```

```
{
      map<string,int>MP;

       //Initially everything is zero
      cout<<MP["Rafiq hasan"]<<endl;
      cout<<MP["Karim khan"]<<endl;
      cout<<MP["Khaled Milon"]<<endl;
      cout<<endl;

      MP["Rafiq hasan"]=10;
      MP["Karim khan"]=29;
      MP["Khaled Milon"]=30;

      cout<<MP.size()<<endl;
      //Use the key to access the value
      cout<<MP["Rafiq hasan"]<<endl;
      cout<<MP["Karim khan"]<<endl;
      cout<<MP["Khaled Milon"]<<endl;
      cout<<endl;


      //Solution of BUBT problem A
      map<string,string>MP2;
      MP2["ko te kader molla"]="tui rajakar tui rajakar";
      MP2["so te sayeedi"]="tui rajakar tui rajakar";
      MP2["tumi k ami k"]="bangali bangali";
      MP2["tomar amar thikana"]="podda meghna jomuna";

      cout<<MP2["ko te kader molla"]<<endl;
      cout<<MP2["so te sayeedi"]<<endl;
      cout<<MP2["tumi k ami k"]<<endl;
      cout<<MP2["tomar amar thikana"]<<endl;

      cout<<endl;


      //iterate through the MAP
      cout<<"ITERATING THE MAP: "<<endl;
      map<string,string>::iterator it;
      for(it=MP2.begin();it!=MP2.end();it++)
      {
              cout<<it->first<<" "<<it->second<<endl;
      }
      cout<<endl;

      //Finding key
      cout<<"Finding a key"<<endl;;
      it=MP2.find("ko te kader molla");

      if(it!=MP2.end())
          cout<<it->first<<" "<<it->second<<endl;
       else
          cout<<"Not found"<<endl;
```

```
           it=MP2.find("ko te");
        if(it!=MP2.end())
            cout<<it->first<<" "<<it->second<<endl;
        else
            cout<<"Not found"<<endl;
        cout<<endl;

        //Erasing elements

        MP2.erase("ko te kader molla");
//        cout<<MP["ko te kader molla"]<<endl;
//
         it=MP2.find("ko te kader molla");
         if(it!=MP2.end())
                cout<<it->first<<" "<<it->second<<endl;
        else
            cout<<"Not found"<<endl;

//pair in map
        map <int, pair <int ,int> > a;
        a[0] = make_pair(2,3);


        MP.clear();
        return 0;
}
```

/*QUEUE HANDLING*/

```
#include<iostream>
#include<queue>
using namespace std;


int main()
{
        queue<int>Q;

        for(int i=1;i<=10;i++)
            Q.push(i);

        cout<<"Size: "<<Q.size()<<endl;
        cout<<"Back: "<<Q.back()<<endl;
        cout<<"Popping elements: ";
        while(Q.empty()!=true)
        {
                cout<<Q.front()<<" ";
                Q.pop();
        }
        cout<<endl;

        return 0;
}
```

/*STACK HANDLING*/

```
   #include<iostream>
#include<stack>
using namespace std;


int main()
{
        stack<int>S;

        for(int i=1;i<=10;i++)
            S.push(i);

        cout<<"Size: "<<S.size()<<endl;

        cout<<"Popping elements: ";
        while(S.empty()!=true)
        {
                cout<<S.top()<<" ";
                S.pop();
        }
        cout<<endl;


        return 0;
}

/*SORT() HANDLING*/
#include <bits/stdc++.h>
using namespace std;

struct data{
    int x,y;

    /*this would sort the custom 'data' class increasingly. to sort decreasingly
depending on x, just write "return this-> x > b.x;"*/
    bool operator < (const data &b) const
    {
        return this->x < b.x;
    }
};

/*this is a method to decreasingly sort integer type data using std::sort*/
 /* how to use this method? : " sort(a.begin(), a.end(), decreasing); " */
bool decreasing (int a, int b)
{
    return a > b;
}

int main()
{
    vector<int> a;
    for(int i = 0; i < 5; i++)
    {
        a.push_back(i);
```

```
        }
    cout<<a[0]<<endl;
    sort(a.begin(), a.end(), decreasing);
    cout<<a[0]<<endl;
    /*again to reverse the array*/
    reverse(a.begin(), a.end());
    cout<<a[0]<<endl;
}

/*string class handling*/
#include <bits/stdc++.h>
using namespace std;


int main()
{
    string a = "ABCD E";
    string b,c,d;

    d = c = b = a;
    cout <<"INITIALLY THE STRINGS ARE :\n";
    cout<<a<<endl;
    cout<<b<<endl;
    cout<<c<<endl;
    cout<<d<<endl;

    /*now we learn to use 4 types of insert functions:*/

    /* string.insert(koto no. position e insert korte hobe, jei string ke insert korte hobe);
        string.insert(koto number position e insert korte hobe, kon string ke insert korte hobe, jei string ke insert korte hobe tar index 0 theke koto length porjonto insert korte hobe);
        string.insert(koto number position e insert korte hobe, kon string ke insert korte hobe, jei string ke insert korbo tar koy number index theke insert korbo ? ,jei string ke insert korte hobe tar oto no. index theke koto length porjonto insert korte hobe);
    */
    a.insert(3, "012345678");
    b.insert(3,"012345678",4);
    c.insert(3,"012345678",4, 3);
    cout<<"AFTER INDEXIVE INSERTION\n";
    cout<<a<<endl;
    cout<<b<<endl;
    cout<<c<<endl;

    /*to insert just a character*/
    d.insert(d.begin()+4, '#');
    cout<<d<<endl;

    /*now jodi chai je ei # jeikhane insert hobe oi jaygar iterator ta dhore rrakhte... tahole nicher way te kora lagbe*/

    auto  it  =  d.find('#'); // to be(,) not to be: that is the question
```

```
//convert int to string
/*
stringstream ss;
int p = 123;
ss << p;
string newst;
ss >> newst;
ss.clear(); //reset the stream
*/
```

```
   //    a.insert (a.end(),3,'.');       // to be, not to be: that is the
question(...)
//    a.insert (it+2,a.begin(),a.begin()+3); // (or )
    /*and oi iterator er porer jayga te ja iccha insert kora jay abar*/
    d.insert(it+1, "PAISI");
    cout<<d<<endl;
    /*abar tomra jodi chao je string:: iterator er kono position er pore n shonkhok
kono character boshabe tahole */

    string::iterator it2 = d.end();

    d.insert(it2, 4, '*');
//    it2 = d.end()-1;
//    d.insert(it2,"A");
    cout<<d<<endl;

    /*final surprise, ekta string er ekta segment er theke arekta segment insert
korte just jar ta copy
    korba tar starting ar ending iterator ta diye dao*/

d.insert(d.end(), a.begin(), a.begin() + 3);    cout<<d<<endl;

    /*Learning to create substring*/

    string tst = "0123456";

    cout<<tst.substr(0, 3) << " "<<tst.substr(5)<<endl;
    cout<<"thik length e substring null paowa jay as length tomo index e null
thake"<<"-> " <<tst.substr(tst.length())<<"<-kisu nai"<<endl;
}

/*Number of maximum nodes in a m-ary tree of hight h = (m^(h+1) - 1) / (m-1) [when, m
> 1] */


/*VECTOR LEARNING*/
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;

int main()
{
        vector<int>V;
        vector<int> first;
// empty vector of ints
        vector<int> second (4,100);                                          //
four ints with value 100
        vector<int> third (second.begin(),second.end());  // iterating through second
        vector<int> fourth (third);                                          //
a copy of third


        // push_back in the vector
```

```
            for(int i=0;i<10;i++)
             V.push_back(i);

        // access elements in the vector
        cout<<"After push_back V contains: ";
        for(int i=0;i<V.size();i++)
            cout<<" "<<V[i];
        cout<<endl;

        //reverse the whole vector
        cout<<"After reversal V contains: ";
        reverse(V.begin(),V.end());

        for(int i=0;i<V.size();i++)
            cout<<V[i]<<" ";
        cout<<endl;


        // sort the vector
        cout<<"After sorting V contains: ";
        sort(V.begin(),V.end());

        for(int i=0;i<V.size();i++)
            cout<<V[i]<< " ";
        cout<<endl;

        // erase the 6th element
        V.erase (V.begin()+5);

        cout << "After erasing 6th element V contains:";
        for (int i=0; i<V.size(); ++i)
                cout << " "<< V[i];
        cout <<endl;

        // erase the first 3 elements:
        V.erase (V.begin(),V.begin()+3);

        cout << "After erasing first 3 elements V contains:";
        for (int i=0; i<V.size(); ++i)
                cout << " "<< V[i];
        cout <<endl;

        //clear the vector
        V.clear();
        cout<<"Cleared: "<<V.size()<<endl;
        return 0;
}
```

/*BIPERTITE CHECKING USING BFS*/

```
#include <bits/stdc++.h>
using namespace std;
#define inf 2147483647

int n, Time;
```

```
struct node{
    string color;
    vector <int>adjacent;
    int depth;
    int parent;
    int id;
    node()
    {
        color = "white";
        depth = inf;
        id = 1;
    }
};


//void DFS_VIGIT(node *vertices, int u)
//{
//    Time++;
//    vertices[u].depth = Time;
//    vertices[u].color = "gray";
//    cout <<u<<" ";
//    for(int i = 0; i < vertices[u].adjacent.size(); i++)
//    {
//        int v = vertices[u].adjacent[i];
//        if(vertices[v].color == "white")
//        {
//            vertices[v].parent = u;
//            DFS_VIGIT(vertices, v);
//        }
//    }
//    vertices[u].color = "black";
//    Time++;
//    vertices[u].finishing_Time = Time;
//
//}
//
//void DFS(node *vertices)
//{
//    Time = 0;
//    for(int u = 1; u <= n; u++)
//    {
//        if(vertices[u].color == "white")
//        {
//            DFS_VIGIT(vertices, u);
//        }
//    }
//    cout<<endl;
//}

bool BFS(node *vertices, int source)
{
    bool flag = true;
    vertices[source].color = "gray";
```

```
    vertices[source].depth = 0;
    vertices[source].parent = NULL;
    queue<int> serial;

    serial.push(source);

    while(serial.empty() == false)
    {
        int u = serial.front();
        serial.pop();
        for(int i = 0; i < vertices[u].adjacent.size(); i++)
        {
            int v = vertices[u].adjacent[i];
            if(vertices[v].color == "white")
            {
                vertices[v].color = "gray";
                vertices[v].id = vertices[u].id*-1;
                serial.push(v);
            }
            else if(vertices[v].color == "gray")
            {
                if(vertices[v].id == vertices[u].id)
                {
                    flag = 0;
                    return flag;
                }
            }
        }
    }
    return flag;
}

int main()
{
    int edges,source;
    cout<<"NUMBER OF NODES: ";
    cin >>n;
    cout <<"NUMBER OF EDGES: ";
    cin>>edges;
    cout<<"SOURCE: "; cin>>source;

    node vertices[n+1];
    for(int i = 1; i <= edges; i++)
    {
        int x,y;
        cin>>x>>y;
        vertices[x].adjacent.push_back(y);
    }

//    DFS(vertices);
    bool decision = BFS(vertices, source);
    if(decision == true)
    {
        cout<<"BIPERTITE\n";
```

```
        }
    else{
        cout<<"NOT BIPERTITE\n";
    }
}
```

/*Efficient nPr, nCr*/

```
#include <bits/stdc++.h>
using namespace std;
#define li long long int
#define rl(n) scanf("%I64d", &n)
#define rll(m,n) scanf("%I64d %I64d", &m, &n)
#define rlll(m,n,o) scanf("%I64d %I64d %I64d", &m, &n, &o)
#define ri(n) scanf("%d", &n)
#define rc(n) scanf("%c", &n)
#define rs(n) gets(s)
#define rst(n) getline(cin,n)
#define rfile(a) freopen(a, "r", stdin)
#define pi acos(-1.00)
#define pb push_back
#define mp make_pair
#define Pr printf
#define For(i,a,b) for(int i = a; i < b; i++)

#define MOD 1000003
#define lim 1e250

map<double, string> converter;
string make_string(double a)
{
    if(a == INFINITY) return "inf";
    if(a == -1) return "MATH ERROR";
    string ret = converter[a];
    if(ret != "") return ret;

    string num;  int dig;char c;
    while(1)
    {
        if(1.00 > a)
        {
            if((int) a == 0) break;
        }
        double div = floor(a/10.00);
        div*=10.00;
        dig = a-div;
        c = char(dig+48);
        a = a/10.0;

        if(c >= 48 && c <= 57)
        {
            num.insert(num.begin()+0,c);
        }
        else{
```

```
                num.insert(num.begin()+0, 48);
        }
    }
    return num;
}

map<pair<double, double>, double> presult;
double nPr(double n, double r)
{
    if((li) r > 100000000) return INFINITY;
    if(r > n) return -1;
    if(r < 0.01 || n < 0.01) return -1;
    double m = 1;
    if(presult[mp(n,r)] != NULL)
    {
        return presult[mp(n,r)];
    }

    for(int i= 0; i < r && m != INFINITY; i++)
    {
        m = m*(n-i);
    }
    return presult[mp(n,r)] = m;
}


map<pair<double, double>, double> cresult;
double nCr(double n, double r)
{
    if((li) r > 100000000) return INFINITY;
    if(r > n) return -1;
    if(r < 0.01 || n < 0.01) return -1;
    double m = 1;
    if(cresult[mp(n,r)] != NULL)
    {
        return cresult[mp(n,r)];
    }

    for(int i= 0; i < r && m != INFINITY; i++)
    {
        m = m*((n-i)/(i+1));
    }
    return cresult[mp(n,r)] = m;
}




int main()
{
    int dec; double n,r;
    while(scanf("%lf %lf",&n, &r) != EOF)
    {
        cout<<"n = "<<n<<" :: r = "<<r<< "; nCr = ";
```

```
        ri(dec);
        if(dec == 1)
        {
            cout<<make_string(nCr(n,r))<<endl;
        }
        else if(dec == 2)
        {
            n = nCr(n,r);
            if(n == -1) Pr("MATH ERROR\n");
            else cout<<n<<endl;

        }
        Pr("\n");
    }
}
```

/*Hexagonal Intersection of Triangle, CF*/

```
#include <bits/stdc++.h>
using namespace std;

/*
total number of 1 unit triangles among the parallel lines =
(sum of length of 3 consecutive lines picking any one as the first line)^2 - [(first length)^2 + (third length)^2 + (fifth length)^2]
*/
int main()
{
    int a,b,c,d,e,f; cin>>a>>b>>c>>d>>e>>f;
    int ans = (a + b + c) * (a + b + c) - ((a*a) + (c*c) + (e*e));
    cout<<ans<<endl;
}
```

// C++ program to count trailing 0s in n!

```
#include <iostream>
using namespace std;

// Function to return trailing 0s in factorial of n
int findTrailingZeros(int n)
{
    // Initialize result
    int count = 0;

    // Keep dividing n by powers of 5 and update count
    for (int i=5; n/i>=1; i *= 5)
          count += n/i;

    return count;
}

// Driver program to test above function
int main()
{
```

```
        int n = 100;
    cout << "Count of trailing 0s in " << 100
         << "! is " << findTrailingZeros(n);
    return 0;
}
```

/*Prime Factorization*/

```
#include <bits/stdc++.h>
using namespace std;

bool check (int i, int k)
{
    return (i>>k)&1;
}
int On(int i, int k)
{
    return i|(1 << k);
}
int Off(int i, int k)
{
    return (i-((check(i,k))<<k)  );
}

unsigned status[10000032/32] = {0};

vector<pair<int,int> > prime_factors[1000001];
void siv()
{
    int i, j, sq;
    sq = (int) sqrt(1000000);
    status[0/32] = On(status[0/32], 0%32);
    status[1/32] = On(status[1/32], 1%32);

    for(i = 3; i <= sq; i+=2)
    {
        if(check(status[i/32], i%32) == 0)
        {
            for(j = i * i; j <= 1000000; j+=2*i)
            {
                status[j/32] = On(status[j/32], j % 32);
            }
        }
    }
}

void prime_factorization()
{
    prime_factors[2].push_back({2,1});
    prime_factors[3].push_back({3,1});
    for(int i = 4; i <= 1000000; i++)
    {
        if(check(status[i/32],i%32) == 1 || (i % 2 == 0))
        {
```

```
            int sq = sqrt(i);
            for(int j = 2; j <= sq; j++)
            {
                int k = i/j;
                if(k * j == i)
                {
                    int x,y,a,b;
                    x = prime_factors[j].size();
                    y = prime_factors[k].size();

                    for(a = 0, b = 0; a < x || b < y; )
                    {
                        pair<int,int> p1,p2;
                        if(a < x && b < y)
                        {
                            p1 = prime_factors[j][a];
                            p2 = prime_factors[k][b];
                            if(p1.first == p2.first)
                            {
                                p1.second+=p2.second;
                                prime_factors[i].push_back(p1);
                                a++; b++;
                            }
                            else if(p1.first < p2.first)
                            {
                                prime_factors[i].push_back(p1);
                                a++;
                            }
                            else{
                                prime_factors[i].push_back(p2);
                                b++;
                            }
                        }
                        else if(a < x)
                        {
                            prime_factors[i].push_back(prime_factors[j][a++]);
                        }
                        else
                        {
                            prime_factors[i].push_back(prime_factors[k][b++]);
                        }

                    }
                    break;
                }
            }
        }
        else{
            prime_factors[i].push_back({i,1});
        }
    }
}

int main()
```

```
   {
    siv();
//    cout<<"2 is a prime\n";
//    for(int i = 3; i <= 30; i+=2)
//    {
//        if((check(status[i/32], i % 32) == 0))
//        {
//            cout<<i <<" is a prime\n";
//        }
//    }
    prime_factorization();
//    for(int i = 0; i < prime_factors[18].size(); i++)
//    {
//        cout<<prime_factors[18][i].first<<"^"<<prime_factors[18][i].second<<" ";
//    }
//    cout<<endl;
}
```

/*Modular Inverse*/

If you want to find **b^{-1} mod M** and

- If M is prime, then simply find **b^{M-2} mod M.**
- If M is not prime, then simply find **$b^{phi[M]-1}$ mod M , where phi[M] = Euler Totient of M.**

/*O(log n) Fibonacci*/

```
#include <iostream>
#include <unordered_map>
#define li long long int
using namespace std;

const li P = 1000000007;

/* Fast fibonacci with O(1) memory and O(lg n) time complexity. No cache. */

li fib_uncached (li n)
{
    /* find MSB position */
    li msb_position = 31;
    while (!((1 << (msb_position-1) & n)) && msb_position >= 0)
        msb_position--;

    li a=0, b=1;

    for (li i=msb_position; i>=0;i--)
    {
        li d = (a%P) * ((b%P)*2 - (a%P) + P),
            e = (a%P) * (a%P) + (b%P)*(b%P);
        a=d%P;
```

```
            b=e%P;

        if (((n >> i) & 1) != 0)
        {
            li c = (a + b) % P;
            a = b;
            b = c;
        }
    }
    return a;
}

/* Fast fibonacci using cache */
li fib (li n)
{
    static std::unordered_map<li,li> cache;

    if (cache.find(n) == cache.end())
    {
        li f;
        if (n==0)
            f = 0;
        else if (n < 3)
            f = 1;
        else if (n % 2 == 0)
        {
            li k = n/2;
            f = (fib(k) * (2*fib(k+1) - fib(k))) % P;
        }
        else
        {
            li k = (n-1)/2;
            f = (fib(k+1)*fib(k+1)+ fib(k) * fib(k)) % P;
        }
        if (f<0)
            f += P;

        cache[n] = f;
    }
    return cache.at(n);
}

int main ()
{
    li i ;
    cin >> i;
    cout << i << " : " << fib(i) << endl;
    return 0;
}
```

#PASSING ARRAY AS ARGUMENT

If you want to pass a single-dimension array as an argument in a function, you would have to declare function formal parameter in one of following three ways and all three declaration methods produce similar results because each tells the compiler that an integer pointer is going to be received. Similar way you can pass multi-dimensional array as formal parameters.

Way-1

Formal parameters as a pointer as follows. You will study what is pointer in next chapter.

```
void myFunction(int *param)
{
.
.
.
}
```

Way-2

Formal parameters as a sized array as follows:

```
void myFunction(int param[10])
{
.
.
.
}
```

Way-3

Formal parameters as an unsized array as follows:

```
void myFunction(int param[])
{
.
.
.
}
```

Example

Now, consider the following function, which will take an array as an argument along with another argument and based on the passed arguments, it will return average of the numbers passed through the array as follows:

```
double getAverage(int arr[], int size)
{
  int    i;
  double avg;
  double sum;

  for (i = 0; i < size; ++i)
  {
    sum += arr[i];
  }

  avg = sum / size;

  return avg;
}
```

Now, let us call the above function as follows:

```
#include <stdio.h>

/* function declaration */
double getAverage(int arr[], int size);

int main ()
{
   /* an int array with 5 elements */
   int balance[5] = {1000, 2, 3, 17, 50};
   double avg;

   /* pass pointer to the array as an argument */
   avg = getAverage( balance, 5 ) ;

   /* output the returned value */
```

```
    printf( "Average value is: %f ", avg );


  return 0;
}
```

When the above code is compiled together and executed, it produces the following result:

```
Average value is: 214.400000
```

As you can see, the length of the array doesn't matter as far as the function is concerned because C performs no bounds checking for the formal parameters.

There are three ways to pass a 2D array to a function:

1, The parameter is a 2D array

```
int array[10][10];
void passFunc(int a[][10])
{
    // ...
}
passFunc(array);
```

2, The parameter is an array containing pointers

```
int *array[10];
for(int i = 0; i < 10; i++)
    array[i] = new int[10];
void passFunc(int *a[10]) //array containing pointers
{
    // ...
}
passFunc(array);
```

3, The parameter is a pointer to a pointer

```
int **array;
array = new int *[10];
for(int i = 0; i <10; i++)
    array[i] = new int[10];
void passFunc(int **a)
{
    // ...
}
passFunc(array);
```

*Comparison Between Doubles

```
double a,b, eps = 1e-9;
if(a == b) -> if(fabs(a-b) <= eps)
if(a < b) -> if((a+eps) < b)
if(a < 0) -> if(a < eps)
```

/*ITERATIVE KNAPSACK 2 STATES Complete*/

```
#include <bits/stdc++.h>
using namespace std;
#define li long long int
#define rl(n) scanf("%I64d", &n)
#define rll(m,n) scanf("%I64d %I64d", &m, &n)
#define rlll(m,n,o) scanf("%I64d %I64d %I64d", &m, &n, &o)
#define ri(n) scanf("%d", &n)
#define rc(n) scanf("%c", &n)
#define rs(n) gets(s)
#define rst(n) getline(cin,n)
#define rfile(a) freopen(a, "r", stdin)
#define pi acos(-1.00)
#define pb push_back
#define mp make_pair
#define Pr printf
#define For(i,a,b) for(int i = a; i < b; i++)
#define MOD 1000003


int p[11][10001], d[11][10001];
int v[11], wt[11], n, W;

void knapsack()
{
    for(int w = 0; w <= W;w++) p[0][w] = 0;

    for(int i = 0; i <= n; i++)
    {
        p[i][0] = 0;
        for(int w = 0; w <= W; w++)
        {
            if(wt[i] <= w)
            {
                if(v[i] + p[i-1][w-wt[i]] > p[i-1][w])
                {
                    /*if taking this ith item would result in more profit
                    than we got without the ith item*/
                    p[i][w] = v[i] + p[i-1][w-wt[i]];
                    d[i][w] = 1;
                }
                else if(v[i] + p[i-1][w-wt[i]] == p[i-1][w])
                {
                    p[i][w] = p[i-1][w];
```

```
                        d[i][w] = 2;
                }
                else{
                    p[i][w] = p[i-1][w];
                    d[i][w] = 0;
                }
            }
            else{
                p[i][w] = p[i-1][w];
                d[i][w] = 0;
            }
        }
    }
}
vector<int> ans;
int solution_counter = 0;
void print_multiple_solution(int i, int j)
{
    if(i <= 0 || j <= 0)
    {
        Pr("Solution %d : ", ++solution_counter);
        int sz = ans.size();
        for(int i = sz-1; i >= 0; i--)
        {
            Pr("%d ", ans[i]);
        }
        Pr("\n");
        return;
    }

    if(d[i][j] == 1)
    {
        ans.push_back(i);
        print_multiple_solution(i-1, j - wt[i]);
        ans.pop_back();
    }
    else if(d[i][j] == 2)
    {
        print_multiple_solution(i-1,j);
        ans.push_back(i);
        print_multiple_solution(i-1, j - wt[i]);
        ans.pop_back();
    }
    else{
        print_multiple_solution(i-1,j);
    }
}


int main()
{
        /*Total Item*/
    ri(n);
    for(int i = 1; i <= n; i++)
```

```
        {
         ri(v[i]); ri(wt[i]);
     }
     /*Capacity*/
     ri(W);
     knapsack();
     solution_counter = 0;
     print_multiple_solution(n,W);
}
```

/*Number of Equilatarel Triangles from given point on a Circle Circumference*/

```
set<int> Gap;
bool valid(int i, int dist, int ttl)
{
    set<int>::iterator lim = Gap.end(), it1,it2;
    int firstpoint, secondpoint;
    firstpoint = (i + dist) % ttl;
    secondpoint = (i + 2*dist) % ttl;
    it1 = Gap.find(firstpoint); it2 = Gap.find(secondpoint);
    if(it1 == lim || it2 == lim) return 0;
    else return 1;
}

int main()
{
        int n,ans;
        while(ri(n) != EOF)
        {
         Gap.clear(); Gap.insert(0);
         vector<int> arr(n+1, 0), prefsum(n+1,0);

                int ttl = 0;
                For(i,1,n+1){ ri(arr[i]); ttl+=arr[i]; prefsum[i] = ttl; if(i < n)
        Gap.insert(ttl);}
                int avg_diff = (int) ttl / 3;
                ans= 0; For(i,1,n+1){if(valid(prefsum[i], avg_diff, ttl)) ans++;}
                Pr("%d\n", (int)ans/3);
        }
        return 0;
}
```

/*DISJOINT SET*/

```
int parents[100001],cost[100001],member[100001];
struct edge{
    int u,v,c;
    bool operator <(const edge& ot) const{
        return c < ot.c;
    }
};vector<edge> ed;
```

```
void makeset(int n){ed.clear(); For(i,1,n+1){ parents[i]=i; member[i] = 1;}}

int find_par(int a)
{
    if(parents[a] == a) return a;
    else return parents[a] = find_par(parents[a]);
}

void Union(int a,int b)
{
    int u=find_par(a);
    int v=find_par(b);
    if(u!=v)
    {
        if(member[u] > member[v])
        {
            parents[v] = u;
            member[u]++;
        }
        else
        {
            parents[u] = v;
            member[v]++;
        }
    }
}
```

/*RAFIUL’s PRIME FACTORIZATION*/

```
#include <bits/stdc++.h>
#define max 10000

using namespace std;

int List[100];  // saves the List
int listSize;   // saves the size of List
int status[10000];
int prime[1000];

void seive() {
    int i, j;
    // initially we think that all are primes, so change the status
    for( i = 2; i <= max; i++ )
    status[i] = 0;

    for( i = 2; i <= max; i++ ) {
        if( status[i] == 0 ) {
```

```
                // so, i is a prime, so, discard all the multiples
                // j = 2 * i is the first multiple, then j += i, will find the
                // next multiple
                for( j = 2 * i; j <= max; j += i )
                    status[j] = 1; // status of the multiple is 1
            }
        }
        // print the primes
        int k=0;
        for( i = 2; i <= max; i++ ) {
            if( status[i] == 0 ) {
                // so, i is prime
                prime[k]=i;
                k++;
            }
        }
    }

    void primeFactorize( int n ) {
        listSize = 0;   // Initially the List is empty, we denote that size = 0
        int sqrtN = int( sqrt(n) ); // find the sqrt of the number
        for( int i = 0; prime[i] <= sqrtN; i++ ) { // we check up to the sqrt
            if( n % prime[i] == 0 ) { // n is multiple of prime[i]
                // So, we continue dividing n by prime[i] as long as possible
                while( n % prime[i] == 0 ) {
                    n /= prime[i]; // we have divided n by prime[i]
                    List[listSize] = prime[i]; // added the ith prime in the list
                    listSize++; // added a prime, so, size should be increased
                }
                // we can add some optimization by updating sqrtN here, since n
                // is decreased. think why it's important and how it can be added
            }
        }
        if( n > 1 ) {
            // n is greater than 1, so we are sure that this n is a prime
            List[listSize] = n; // added n (the prime) in the list
            listSize++; // increased the size of the list
        }
    }
    int main() {
        int N;
        scanf("%d",&N);
        seive();
        primeFactorize(N);
```

```

    for( int i = 0; i < listSize; i++ ) // traverse the List array
            printf("%d ", List[i]);
        return 0;
    }
```

```
//BITSET LEARNING
#include <bits/stdc++.h>
using namespace std;
int main()
{
    bitset<16> a(15), b(0xf),c(string("11"));
    /**
        they will gibe equal to
        a = 0000000000001111
        b = 0000000000001111
        c = 0000000000000011
    **/

    /** Now suppose you want to
            set all the bits of 'b' to 1
            set all the bits of 'a' to 0
            set 5th bit of b to 0 (indexing starts from 0 by the way, and from left)
            set 6th bit of a to 1 (indexing starts from 0 by the way, and from left)
            then do the following works
    **/
        b.set();   // sets all the bits of b to 1
        a.reset(); // sets all the bits of a to 0
        b.reset(5); //resets 5th bit of b
        a.set(6,1); //set 6th bit of a to 1

    /**BINARY OPERATORS**/
    bitset<4> x(string("0001")), y (string("0011"));
    cout<<(x&y)<<endl; //operates x & y bit to bit
    cout<<(x|y)<<endl; //operates x or y bit to bit
    cout<<(x^y)<<endl; //operates x XOR y bit to bit
    cout<<x<< " after SHL 2 bits "<< (x<<2)<<endl; //shifts x LEFT
    cout<<x<< " after NOT x "<< (~x)<<endl; //INVERTS all the bits of X
```

```
/** [] operator **/
    x.set();
    x[0] = 0; //THIS WOULD DO THE SAME AS x.reset(0)

    /** SOME MORE FUNCTIONS
    x.filp(n) -> inverts the nth bit from left of x
    x.filp() -> equivalent to (~x)
    x.none() -> returns boolean value 1 or 0. 1 if any bits of x is 1. if (x == 0)
returns 0
    x.count() -> returns how many bits of x are set
    x.any() -> returns true if at least one bit is set
    x.size() -> returns the size of the bitset O(1)
    x.test(n) -> for (n = 0 to size-1) it returns 1 if that nth bit is 1, or 0
otherwise

    unsigned long int var1 = x.to_ulong(); -> it puts the unsigned long int value of
x inside var1
    string s1 = x.to_string() -> converts string form of x inside s1

    **/

}
```

প্রোগ্রামিং প্রতিযোগিতার শুরুর গল্প

Dawn of Programming Contest

মোঃ মাহবুবুল হাসান

জুলাই, ২০১৫

সূচীপত্র

চিত্র তালিকা

সারণী তালিকা

অধ্যায় ৩

Mathematics

৩.১ Number Theory

৩.১.১ Prime Number

Prime Number কে বাংলায় মৌলিক সংখ্যা বলে। একটি সংখ্যা n কে মৌলিক বলা হয় যদি ঐ সংখ্যাটি 1 এর থেকে বড় হয় এবং 1 বা n ছাড়া আর কোন ধনাত্মক সংখ্যা দ্বারা বিভাজ্য না হয়। এখন যদি একটি সংখ্যা n দিয়ে বলা হয় এটি Prime কিনা- তুমি কেমনে করবা? মোটামোটি সংজ্ঞা থেকেই বুঝা যায় কেমনে করা উচিত। অবশ্যই n এর থেকে কোন বড় সংখ্যা দিয়ে n কে ভাগ করা যায় না। সুতরাং যদি 2 হতে $n-1$ এর মাঝের কোন একটি সংখ্যা দ্বারা n নিঃশেষে বিভাজ্য হয় তাহলে n Prime না। সুতরাং আমরা এই Idea এর উপর ভিত্তি করে যদি Primality চেক করার জন্য একটি ফাংশন লিখি তা দাঁড়াবে কোড ৩.১ এর মত।

কোড ৩.১: isPrime1.cpp

```
//returns 1 if prime, otherwise 0
int isPrime(int n)
{
    if(n <= 1) return 0;
    for(int i = 2; i < n; i++)
        if(n % i == 0)
            return 0;

    return 1;
}
```

এখন এর Time Complexity কত? Worst case অর্থাৎ কোডটি সবচেয়ে বেশি সময় নিবে যদি এটি Prime হয়। সেক্ষেত্রে for loop টি $n-2$ বার চলবে, সুতরাং এটির Time Complexity $\mathrm{O}(n)$. তোমরা ভাবতে পার, আচ্ছা আমরা তো জানি, 2 বাদে কোন জোড় সংখ্যা Prime না। সুতরাং for loop টা তো শুধু বিজোড় সংখ্যার উপর দিয়ে চালালেই হয়! ভালো বুদ্ধি। তাহলে আমাদের run time কত হবে? $\mathrm{O}(n/2)$ আর আমরা বলেছি আমরা সকল constant coefficient term কে বাদ দেই। তাহলে এভাবে করলেও আমাদের run time $\mathrm{O}(n)$ ই থাকে। হ্যা, অর্ধেক হবে কিন্তু এটা আমাদের order notation এ কোন ব্যাপারই না। তাহলে আমরা কেমনে কমাবো? একটু চিন্তা করলে দেখবে যে, যদি এমন কোন d খুঁজে পাও যা n কে ভাগ করে তাহলে তুমি আরও একটি সংখ্যা কিন্তু খুঁজে বের করে ফেলেছ যেটা n কে ভাগ করেঃ n/d. অর্থাৎ কোন একটি সংখ্যার divisor

(গুণনীয়ক) গুলি সবসময় জোড়ায় জোড়ায় থাকে। যেমন $n = 24$ হলে এর divisor গুলি হচ্ছেঃ $1, 2, 3, 4, 6, 8, 12, 24$ এবং তারা 4টি জোড়ায় আছেঃ $(1, 24), (2, 12), (3, 8), (4, 6)$. একটু চিন্তা করলে দেখবে প্রতিটি জোড়ার ছোটটি সবসময় $\leq \sqrt{n}$ হবে। কেন? এটি direct প্রমাণ করা মনে হয় একটু কঠিন হবে, কিন্তু Proof by Contradiction এ কিন্তু খুবই সোজা। মনে কর ছোটটি $\sqrt{n}$ এর থেকেও বড়, তাহলে ঐ জোড়ার বড়টাতো বড় হবেই! আর জোড়াগুলি এমনভাবে বানানো হয়েছে যেন তাদের গুনফল n হয়। কিন্তু দুইটি $\sqrt{n}$ এর থেকে বড় সংখ্যার গুনফল কেমনে n হয়? অতএব জোড়ার ছোটটিকে অবশ্যই $\sqrt{n}$ এর সমান বা ছোট হতে হবে। এভাবে আমরা যদি কোড করি (কোড ৩.২) তাহলে আমাদের run time হবে $O(\sqrt{n})$. এখানে খেয়াল করতে পার যে আমরা আমাদের for loop এর condition টা $i * i \leq n$ লিখেছি, $i \leq sqrt(n)$ না। এর কিছু কারণ আছে। প্রথমত, বার বার sqrt হিসাব করা একটি costly কাজ। দ্বিতীয়ত, double ব্যবহার করলে কিন্তু precision loss হয়। এর ফলে sqrt(9) = 3 না হয়ে 2.9999999 বা 3.0000001 হলেও অবাক হবার কিছু নেই।[1] কিন্তু বার বার $i * i$ করাও কেমন জানি! তোমরা যা করতে পার তাহল loop শুরু হবার আগেই limit = sqrt(n + 1) করে নিতে পার। এর পর এই পর্যন্ত loop চালাবে। তাহলে বার বার sqrt ও করা লাগবে না গুন ও করা লাগবে না।

কোড ৩.২: isPrime2.cpp

```cpp
//returns 1 if prime, otherwise 0
int isPrime(int n)
{
    if(n <= 1) return 0;
    for(int i = 2; i*i <= n; i++)
        if(n % i == 0)
            return 0;

    return 1;
}
```

আরও কি উন্নতি করা যাবে run time? হ্যা যাবে, আসলে এটি $O(\log n)$ সময়েই করা যাবে! কারো যদি এতে আগ্রহ থাকে তাহলে internet এ সার্চ করে দেখতে পার।

Sieve of Eratosthenes

এটি Prime Number কে generate করার একটি দ্রুত উপায়। তোমরা লক্ষ্য করলে দেখবে যে পূর্বের $O(\sqrt{n})$ algorithm এ আমরা যা করেছি তা হল কোন একটি number নিয়ে তাকে কেউ ভাগ করে কিনা তা চেক করেছি। কিন্তু এর ফলে যা হয় তা হল, একটি সংখ্যা prime কিনা তা চেক করার জন্য অনেক সংখ্যা দ্বারা ভাগ করে দেখতে হয়। কিন্তু এই কাজটা যদি আমরা ঘুরিয়ে করি? অর্থাৎ কোন একটি সংখ্যাকে কে কে ভাগ করে এটা না দেখে বরং এই সংখ্যা কাকে কাকে ভাগ করে সেটা যদি দেখি তাহলেই আমাদের কাজ অনেক কমে যাবে। কারণ, এখানে আমরা শুধু মাত্র ঐসব সংখ্যার pair নিচ্ছি যারা একে অপরকে ভাগ করে। Sieve এর algorithm এ ঠিক এই কাজটাই করা হয়। এর মাধ্যমে তোমরা 1 হতে n এর মাঝের সব prime বের করে ফেলতে পারবে। শুধু তাই না, এই সীমার মাঝের কোন সংখ্যা দিলে সেটা prime কিনা সেটাও অনেক দ্রুত বলে দিতে পারবা। Algorithm টি কিন্তু খুবই সোজা! তুমি 2 হতে n পর্যন্ত সব সংখ্যা লিখ, এরপর প্রথম থেকে আসো, একটি করে সংখ্যা নিবা আর তার থেকে বড় তার যতগুলি multiple এখনও আস্ত আছে তা কেটে ফেল! এভাবে একে একে সব সংখ্যা নিয়ে কাজ করলে তোমার কাছে যেসব সংখ্যা অবশিষ্ট থাকবে সেগুলিই হল prime এবং এর বাইরে কিন্তু আর কোন prime নেই! তুমি কিন্তু এই কাজটা $\sqrt{n}$ পর্যন্তও করতে পার! আশা করি

[1] মজার ব্যপার হল double কে এমনভাবে represent করা হয় যেন sqrt ফাংশনটি integer উত্তর এর ক্ষেত্রে সবসময় সঠিক উত্তর দেয়!

এতক্ষণে বুঝতে পারছ কেন! $n = 10$ এর জন্য আমরা টেবিল ৩.১ এ এই algorithm টি simulate করে দেখালাম।

সারনী ৩.১: $n = 10$ এর জন্য sieve algorithm এর simulation

বিবরন	বর্তমান অবস্থা
initial অবস্থা	$2, 3, 4, 5, 6, 7, 8, 9, 10$
প্রথম uncut number = 2. আমরা 2 এর বড় সকল multiple কেটে দেব	$2, 3, \cancel{4}, 5, \cancel{6}, 7, \cancel{8}, 9, \cancel{10}$
পরবর্তী uncut number = 3. আমরা 3 এর বড় সকল multiple কেটে দেব	$2, \cancel{3}, \cancel{4}, 5, \cancel{6}, 7, \cancel{8}, \cancel{9}, \cancel{10}$
আর দরকার নেই, $5 > \sqrt{10}$	$2, 3, \cancel{4}, 5, \cancel{6}, 7, \cancel{8}, \cancel{9}, \cancel{10}$

তোমাদের সুবিধার জন্য এর implementation কোড ৩.৩ এ দেওয়া হল। এই algorithm এর run time $O(n \log \log n)$. এই ফাংশন শেষে তোমরা একটি prime number এর লিস্ট পাবা এবং mark array থেকে বলতে পারবে কোনটি prime আর কোনটি না।

কোড ৩.৩: sieve.cpp

```
int Prime[300000], nPrime;
int mark[1000002];

void sieve(int n)
{
    int i, j, limit = sqrt(n * 1.) + 2;

    mark[1] = 1;
    for(i = 4; i <= n; i += 2) mark[i] = 1;

    Prime[nPrime++] = 2;
    for(i = 3; i <= n; i += 2)
        if(!mark[i])
        {
            Prime[nPrime++] = i;

            if(i <= limit)
            {
                for(j = i * i; j <= n; j += i * 2)
                {
                    mark[j] = 1;
                }
            }
        }
}
```

sieve এর দুইটি variation নিয়ে কথা বলা যায়।

Memory Efficient Sieve এখানে খেয়াল করলে দেখবে যে n যত বড়, তত বড় array এর দরকার হয় mark এর জন্য। আমরা কিন্তু জানি 2 ছাড়া সব জোড় সংখ্যা এর mark এ 1 থাকবে সুতরাং এই জিনিস খাটিয়ে আমরা memory requirement অর্ধেক করে ফেলতে পারি। আবার mark array এর প্রতিটি জায়গায় আমরা কিন্তু 0 আর 1 ছাড়া আর কিন্তু কিছু রাখি

না। আমরা চাইলে, প্রতিটি int এ থাকা 32 bit কে কাজে লাগিয়ে একটা variable এ 32 টা information রাখতে পারি এবং memory requirement কে আরও 32 ভাগ করতে পারি।

Segmented Sieve অনেক সময় আমাদের 1 হতে n এর দরকার হয় না, a হতে b সীমার prime গুলির দরকার হয় যেখানে a, b হয়তো $10^{12} \sim 10^{14}$ range এর কিন্তু বাড়তি একটি শর্ত থাকে যে, $b - a \leq 10^6$. এসব ক্ষেত্রে আমরা $[a, b]$ range এ sieve চালাব। এর জন্য প্রথমেই আমাদের $\sqrt{b}$ পর্যন্ত সকল prime বের করে রাখা লাগতে পারে (prime দিয়ে করলে efficient হয় তবে চাইলে 2 হতে $\sqrt{b}$ পর্যন্ত সব সংখ্যা দিয়েও করতে পার।)

প্র্যাকটিস প্রবলেম

- একটি সংখ্যাকে prime factorize কর। অর্থাৎ এটি কোন কোন prime দ্বারা বিভাজ্য এবং সেই সব prime এর power গুলি বের কর।

৩.১.২ একটি সংখ্যার Divisor সমুহ

তুমি যদি কোন একটি সংখ্যার সকল divisor সমুহ বের করতে চাও তাহলে কোড ৩.২ এর মত $O(\sqrt{n})$ এ খুব সহজেই সকল divisor বের করে ফেলতে পার। কিন্তু আমরা কি sieve algorithm কে modify করে 1 হতে n পর্যন্ত প্রতিটি সংখ্যার সকল divisor বের করতে পারি? অবশ্যই! তবে এটির runtime $O(n \log n)$. কোড ৩.৪ এ এর কোডটি দেখানো হল। এখানে তেমন কিছুই না, শুধু প্রতিটি সংখ্যার জন্য আমরা এর multiple এর list গুলোতে তাকে insert করে দেব এখানে আমাদের কোন mark রাখার প্রয়োজন হয় না। তোমরা যদি মনে কর যে memory তো অনেক বেশি লেগে যাবে! না, n টি সংখ্যার divisor আসলে সর্বমোট $n \log n$ এর বেশি না। আমরা এই কোডে আমাদের সুবিধার জন্য STL এর vector ব্যবহার করেছি। vector না ব্যবহার করে Dynamic Linked List ব্যবহার করা যায়, কিন্তু জিনিসটা অনেক ঝামেলার হয়ে যায়।

কোড ৩.৪: all divisors.cpp

```
int mark[1000002];
vector<int> divisors[1000002];

void Divisors(int n)
{
    int i, j;
    for(i = 1; i <= n; i++)
        for(j = i; j <= n; j += i)
            divisors[j].push_back(i);
}
```

অনেক প্রবলেমে divisor এর লিস্ট হয়তো লাগে না, কিন্তু প্রতিটি সংখ্যার divisor সমুহের sum বা number of divisor এর দরকার হয়। আশা করি কেমনে করবে তা বুঝতে পারতেছ!

কোন একটি সংখ্যার divisor নিয়ে যখন problem থাকে তখন আরেকটি method বেশ কাজে লাগে। ধরা যাক, $n = p_1^{a_1} p_2^{a_2} \dots p_k^{a_k}$ এখানে p_i হল prime সংখ্যা। একে কোন একটি সংখ্যার prime factorization বলে। এখন চিন্তা করে দেখ, d কে যদি n এর divisor হতে হয় তাহলে তার কি কি বৈশিষ্ট্য থাকতে হবে। প্রথমত, d এর ভিতরে p_i ছাড়া আর কোন prime divisor থাকা যাবে না। দ্বিতীয়ত, p_i এর power কিন্তু a_i এর থেকে বেশি হতে পারবে না। অর্থাৎঃ $d = p_1^{b_1} p_2^{b_2} \dots p_k^{b_k}$ যেখানে $0 \leq b_i \leq a_i$. এই equation থেকে আমরা বলে দিতে পারি n এর divisor কয়টি আছে (NOD = Number of Divisor) বা তার divisor দের যোগফল কত (SOD = Sum of Divisor)!

$$
\begin{aligned}
NOD(n) &= (a_1+1)(a_2+1)\dots(a_k+1)\\
SOD(n) &= (1+p_1+p_1^2+\dots+p_1^{a_1})(1+p_2+p_2^2+\dots+p_2^{a_2})\dots(1+p_k+p_k^2+\dots+p_k^{a_k})\\
&= \frac{p_1^{a_1+1}-1}{p_1-1}\cdot\frac{p_2^{a_2+1}-1}{p_2-1}\cdot\ldots\cdot\frac{p_k^{a_k+1}-1}{p_k-1}
\end{aligned}
$$

৩.১.৩ GCD ও LCM

GCD এর পূর্ণ রূপ হলঃ Greatest Common Divisor বাংলায় গরিষ্ঠ সাধারণ গুণনীয়ক (গসাগু) আর LCM এর পূর্ণ রূপ হলঃ Least Common Multiple বাংলায় লঘিষ্ঠ সাধারণ গুণিতক (লসাগু)। যদি a এবং b দুইটি সংখ্যার গসাগু g এবং লসাগু l হয় আমরা বলতে পারিঃ $a \times b = g \times l$. সুতরাং আমরা যদি দুইটি সংখ্যার গসাগু বের করতে পারি তাহলে লসাগু খুব সহজেই বের হয়ে যাবে। প্রশ্ন হল আমরা গসাগু কেমনে বের করব? একটি উপায় হল a ও b এর মাঝে যেটি ছোট সেই সংখ্যা থেকে শুরু করে 1 পর্যন্ত দেখা, যেই সংখ্যা দিয়ে a এবং b উভয়েই প্রথম ভাগ যাবে সেটিই তাদের গসাগু। কিন্তু এর run time O(n). এর থেকে ভালো উপায় কিন্তু তোমরা জানো, ছোট বেলায় স্কুল এ থাকতে শিখেছ সেটি হল euclid এর পদ্ধতি। মনে কর আমাদের a আর b দিয়ে বলা হল এদের গসাগু বের করতে হবে, আমরা যা করব, a কে b দিয়ে ভাগ দিব। যদি নিঃশেষে ভাগ যায়, তাহলে b ই গসাগু কারন b এর থেকে বড় কোন সংখ্যা কিন্তু b কে ভাগ করে না (যদিও a কে ভাগ করতে পারে)। এখন যদি ভাগ না যায়, সেক্ষেত্রে আমরা ভাগশেষ c বের করব $a = k \cdot b + c$. এই c কিন্তু b এর থেকে ছোট হবে! ($a < b$ হলে কি হবে তা চিন্তা করে দেখতে পার!) এবং একটি সংখ্যা যদি a ও b কে ভাগ করে সেটা এই সমীকরণ অনুসারে c কেও করবে। সুতরাং আমরা এখন b ও c এর গসাগু বের করব। আগে ছিল a ও b এখন এদের একটি সংখ্যা ছোট হয়ে c হয়ে গেল। সুতরাং এই কাজটা যদি আমরা বার বার করতে থাকি এক সময় আমরা গসাগু পেয়ে যাব। তোমাদের মনে হতে পারে যে অনেক বার এই কাজ করতে হবে! কিন্তু আসলে কিন্তু তা না। এটার exact run time তোমাদেরকে বললে আপাতত বুঝবে না তবে এটুকু বিশ্বাস করতে পার যে long long এ যত বড় সংখ্যা ধরা সম্ভব তাদের যদি গসাগু বের করতে বলা হয় তাহলে $100 \sim 150$ ধাপের বেশি আসলে লাগবে না। [১] গসাগু নির্ণয়ের একটি recursive প্রোগ্রাম কোড ৩.৫ এ দেয়া হল।

কোড ৩.৫: gcd.cpp

```
int gcd(int a, int b)
{
    if(b == 0) return a;
    return gcd(b, a % b);
}
```

৩.১.৪ Euler এর Totient Function (ϕ)

শুরুতেই আমরা জেনে নেই Totient Function কি জিনিস।

$\phi(n)$ = n এর থেকে ছোট বা সমান এমন কতগুলি সংখ্যা আছে যা n এর সাথে coprime

[১]তোমাদের আগ্রহ থাকলে এই বিষয়ে wiki তে পড়তে পার। মজার ব্যপার হল এর runtime এর সাথে ফিবনাচি সংখ্যার একটা সম্পর্ক আছে!

coprime অর্থ হল তাদের কোন সাধারণ factor নেই। যেমন, $\phi(12) = 4$ কারণ $2, 3, 4, 6, 8, 9, 10, 12$ এর সাথে 12 এর কোন না কোন সাধারণ factor আছে। $1, 5, 7, 11$ এই চারটি সংখ্যার সাথে কোন common factor নেই। যদি $n = p_1^{a_1} p_2^{a_2} \dots p_k^{a_k}$ হয় তাহলেঃ

$$\phi(n) = n\left(1 - \frac{1}{p_1}\right)\left(1 - \frac{1}{p_2}\right)\dots\left(1 - \frac{1}{p_k}\right)$$

সুতরাং যদি কোন একটি সংখ্যার ϕ বের করতে হয় তাহলে তোমরা খুব সহজেই prime factorize করে বের করে ফেলতে পারবে। কথা হল prime বা divisor এর মত কি এক্ষেত্রেও 1 হতে n এর ϕ এর মান দ্রুত বের করা সম্ভব? অবশ্যই সম্ভব, প্রতিটি prime p এর জন্য আমরা এর multiple এ গিয়ে তাকে p দিয়ে ভাগ ও $p - 1$ দিয়ে গুন করলেই হবে। এভাবে 1 হতে n পর্যন্ত সব prime এর জন্য এই কাজ করলেই আমরা সব সংখ্যার ϕ এর মান পেয়ে যাব। (কোড ৩.৬)

কোড ৩.৬: sieve phi.cpp

```
int phi[1000006], mark[1000006];

void sievephi(int n)
{
    int i,j;

    for(i = 1; i <= n; i++) phi[i] = i;

    phi[1] = 1;
    mark[1] = 1;

    for(i = 2;i <= n;i += 2)
    {
        if(i != 2) mark[i] = 1;
        phi[i] = phi[i] / 2;
    }

    for(i = 3; i <=n; i += 2)
        if(!mark[i])
        {
            phi[i] = phi[i] - 1;

            for(j = 2 * i; j <= n;j += i)
            {
                mark[j] = 1;
                phi[j] = phi[j] / i * (i - 1);
            }
        }
}
```

৩.১.৫ BigMod

BigMod একটি অত্যন্ত গুরুত্বপূর্ণ method. ধরা যাক তোমার কাছে অনেক লাল বল এবং সাদা বল আছে। তুমি n বার বোলিং করলে, প্রতিটি বল হয় সাদা বল দিয়ে করবে না হয় লাল বল দিয়ে। তুমি

কত ভাবে বল করতে পার? এখানে প্রতিটি বল করতে পারছ 2 ভাবে, সুতরাং সর্বমোট 2^n ভাবে বল করতে পারবে। কিন্তু n এর বড় মানের জন্য এটা অনেক বড় সংখ্যা হয়ে দাঁড়ায়। সেজন্য প্রায় বেশির ভাগ সময়ে আমাদের exact উত্তর না চেয়ে mod 10^7 বা এরকম একটি সংখ্যা দিয়ে দেয়া হয় যার দ্বারা mod করে result চাওয়া হয়। যেমন ধর তোমাদের জিজ্ঞাসা করলাম, 2^{100} mod 7 কত? কি করবে? 2^{100} বের করে এর পর 7 দিয়ে ভাগ করে উত্তর দিবে? খেয়াল কর, 2^{100} কিন্তু long long এও ধরবে না। তাহলে উপায়? তুমি চাইলে প্রতিবার গুন করে সাথে সাথেই mod করতে পার, এতে করে উত্তরটা শুধু int ব্যবহার করেই পেয়ে যাবে! কিন্তু তোমাকে যদি 100 এর থেকে আরও বড়, অনেক বড় ধর প্রায় 10^{18} দেয়া হয়? তাহলে কি করা সম্ভব? হ্যা করা সম্ভব এবং idea টাও অনেক সহজ। মনে কর তোমাকে 2^{100} mod 7 বের করতে বলা হয়েছে। তুমি কি করবে, 2^{50} mod 7 বের করবে, ধর এটি a, তাহলে 2^{100} mod 7 হবে $(a \times a)$ mod 7. অর্থাৎ তুমি তোমার কাজ কে অর্ধেক করে ফেললে। তোমার এখন আর for loop চালিয়ে 100 টা 2 গুন করতে হবে না, 50 টা 2 গুন করে এই গুনফল কে তার সাথেই গুন দিলে তুমি result পেয়ে যাবে। কিন্তু একই ভাবে তোমার কিন্তু 50 বার গুন করার দরকার নেই, 25 বার গুন করে আবার সেই গুনফল কে তার সাথেই গুন করে গুনফল তুমি পেয়ে যাবে। কিন্তু এবার? বিজোড় সংখ্যার বেলায়? এবার তো আর অর্ধেক করতে পারবা না। তাতে কি যায় আসে! তুমি 2^{24} বের করে তার সাথে 2 কে গুন করতে পার। যেহেতু power বিজোড় বলে এই কাজ করছ সুতরাং এর পরের ধাপে power অবশ্যই জোড় হবে এবং আবার তুমি 2 দিয়ে ভাগ করতে পারবে। এভাবে চলতে চলতে এক সময় power যখন 0 হয়ে যাবে তখন আমরা জানি কিন্তু যে $2^0 = 1$ সুতরাং এবার আমরা আমাদের অন্য গুন গুলো করে আমাদের উত্তর বের করে ফেলতে পারব।

তাহলে এই জিনিস আমরা কোড করব কেমনে? আমরা ধরে নেই আমাদের কাছে একটা Black Box আছে যাকে a, b, M দিলে a^b mod M বের করে দেয়। সে যা করবে তা হল যদি b জোড় হয় তাহলে আবার সেই Black Box কে $a, b/2, M$ দিবে এবং সেই ফলাফল দিয়ে নিজের ফলাফল বের করবে। আর যদি বিজোড় হয় তাহলে সে Black Box কে $a, b-1, M$ দিবে এবং তার ফলাফল থেকে নিজের ফলাফল বানিয়ে নিবে। এবং এই ভাবে কতক্ষণ চলবে? যতক্ষণ না, $b = 0$ হয়। এখানে Black Box হল একটি function এবং এই ভাবে একটি function এর ভিতর থেকে একই function ব্যবহার করা কেই recursive function বলে। আমরা কোড ৩.৭ এ এই প্রোগ্রামটি দিলাম। তবে প্রোগ্রামটিতে b বিজোড় এর ক্ষেত্রে একটু অন্যভাবে করা হল, আশা করি বুঝতে অসুবিধা হবে না। এই algorithm এর run time হল $O(\log n)$.

কোড ৩.৭: bigmod.cpp

```
int bigmod(int a, int b, int M)
{
    if(b == 0) return 1 % M;
    int x = bigmod(a, b / 2, M);
    x = (x * x) % M;
    if(b % 2 == 1) x = (x * a) % M;
    return x;
}
```

এই ধরনের solving method কে divide and conquer বলা হয়ে থাকে। এখানে একটি বড় প্রবলেম কে ছোট ছোট ভাগে ভাগ করা হয় এবং তাদের সমাধান combine করে বড় প্রবলেম এর সমাধান বের করা হয়। আমি যখন BigMod দেখাই এর সাথে আরও একটি সমস্যা দেখাই। তাহল, $1 + a + a^2 + \ldots + a^{b-1}$ mod M বের করা। অবশ্যই এর আগের সমস্যার মত এখানেও b অনেক বড়। সুতরাং তুমি যে বার বার প্রতিটা term এর উত্তর বের করবে তা হবে না। এখানেও কিন্তু এর আগের মত বুদ্ধি খাটানো সম্ভব। আমরা $b = 6$ এর জন্য দেখি কেমনে একে দুই ভাগে ভাগ করা সম্ভব!

$$1 + a + a^2 + a^3 + a^4 + a^5 = (1 + a + a^2) + a^3(1 + a + a^2)$$

এভাবে যদি আমরা দুই ভাগ করি তাহলে আমাদের $bigmod(a, b/2, M)$ এর প্রয়োজন পরে। সর্বমোট $\log n$ ধাপ এবং প্রতি ধাপে আমাদের bigmod এর জন্য আরও $\log n$ সময় দরকার হয়, সুতরাং আমাদের run time $O((\log n)^2)$. এটা খুব একটা বড় cost না। কিন্তু তবুও আমরা এর $O(\log n)$ সমাধান শিখব। আমরা equation টিকে অন্যভাবে দুইভাগ করার চেষ্টা করি।

$$\begin{aligned}1 + a + a^2 + a^3 + a^4 + a^5 &= (1 + a^2 + a^4) + a(1 + a^2 + a^4)\\ &= (1 + (a^2) + (a^2)^2) + a(1 + (a^2) + (a^2)^2)\end{aligned}$$

অর্থাৎ আমরা আমাদের নতুন ফাংশন এর নাম যদি bigsum দেই তাহলে $bigsum(a, b, M)$ বের করতে আমরা $bigsum(a^2, b/2, M)$ বের করব। a যেন পরবর্তীতে overflow না করে সেজন্য আমরা আসলে a^2 mod M পাঠাবো। b বিজোড় হলে আশা করি বুঝতে পারছ যে কি করব? তাও দেখাইঃ

$$1 + a + a^2 + a^3 + a^4 = 1 + a(1 + a + a^2 + a^3)$$

আশা করি কোডটা নিজেরাই করতে পারবে। তোমাদের একই ভাবে সমাধান করা যাবে এমন আরেকটি সমস্যা দেই প্র্যাকটিস এর জন্য।

প্র্যাকটিস প্রবলেম

- $1 + 2a + 3a^2 + 4a^3 + 5a^4 + \ldots + ba^{b-1}$ mod M বের কর।

৩.১.৬ Modular Inverse

যেহেতু অনেক হিসাবের উত্তর অনেক বড় আসে সুতরাং প্রায় সময়ই আমাদের exact উত্তর না চেয়ে কোন একটি সংখ্যা দিয়ে mod করে উত্তর চায়। এখন সেই হিসাবে যদি যোগ বিয়োগ গুন থাকে তাহলে কোন সমস্যা হয় না, কিন্তু যদি ভাগ থাকে তাহলে বেশ সমস্যা হয়ে যায়, কারণ $\frac{a}{b}$ mod$M$ এবং $\frac{a\text{mod}M}{b\text{mod}M}$ এক কথা না। তুমি যদি b দ্বারা ভাগ করতে চাও তাহলে b^{-1}mod$M$ বের করতে হবে এবং এটি দিয়ে গুন করলেই b দ্বারা ভাগের কাজ হয়ে যাবে। M যদি prime হয় তাহলে, $b^{-1} \equiv b^{M-2}$ mod M. আর যদি তা না হয়, তাহলে $b^{-1} \equiv b^{\phi(M)-1}$ mod M কিন্তু সেক্ষেত্রে M এবং b কে coprime হতে হবে। এবং এই মান আমরা bigmod ব্যবহার করে খুব সহজেই বের করে ফেলতে পারি।

৩.১.৭ Extended GCD

যদি a ও b এর গসাগু g হয় তাহলে এমন x এবং y খুঁজে পাওয়া সম্ভব যেন $ax + by = g$ হয়। টেবিল ৩.২ এ আমরা $a = 10$ এবং $b = 6$ এর জন্য x ও y বের করে দেখালাম। অনেকেই এই টেবিল দেখে ভরকিয়ে যায়। আসলে ভয় পাবার কিছু নেই। এই টেবিল এর প্রথম কলাম হল সাধারণ euclid এর গসাগু বের করার পদ্ধতি হতে পাওয়া। এখন গসাগু করার সময় অবশ্যই ছোট সংখ্যাকে কোন একটি সংখ্যা দিয়ে গুন করে বড়টি হতে বাদ দিয়েই ভাগশেষ বের করা হয়। এই গুন ও বিয়োগ এর কাজটা আমাদের পরের দুই কলামেও করতে হবে। এরকম করলে আমরা যখন প্রথম কলামে গসাগু পাব তখন দ্বিতীয় ও তৃতীয় কলামে x ও y এর মান পেয়ে যাব। এক্ষেত্রে আমাদের $x = -1$ এবং $y = 2$.

আমরা Extended GCD কে সংক্ষেপে egcd বলে থাকি। এই প্রোগ্রামটির কোড ৩.৮ এ দেয়া হল যদিও কোডটায় আমরা ঠিক উলটো ভাবে x এবং y এর মান বের করেছি। এখানে দেয়া egcd ফাংশনটি call করার সময় দুইটি variable এর reference ও পাঠাতে হবে যেখানে x এবং y এর মান থাকবে। এই ফাংশনটি গসাগু return করে।

সারনী ৩.২: $a = 10$ ও $b = 6$ এর জন্য Extended GCD এর simulation

সংখ্যা	x	y	10x + 6y
10	1	0	10
6	0	1	6
4	1	-1	4
2	-1	2	2

কোড ৩.৮: egcd.cpp

```
int egcd (int a, int b, int &x, int &y)
{
    if (a == 0)
    {
        x = 0; y = 1;
        return b;
    }

    int x1, y1;
    int d = egcd (b%a, a, x1, y1);

    x = y1 - (b / a) * x1;
    y = x1;

    return d;
}
```

egcd ব্যবহার করেও আমরা Modular Inverse বের করতে পারি (তবে b ও M কে coprime হতে হবে)। কোন একটি b এর জন্য আমরা এমন একটি x বের করতে চাই যেন, $bx \equiv 1$ mod M. অর্থাৎ, $bx = 1 + yM$ যেখানে y হল একটি integer. এখন, $bx - yM = 1$. আমরা জানি b ও M coprime সুতরাং আমরা এমন একটি x ও y পাব যেন, $bx - yM = 1$ হয় বা, $bx \equiv 1$ mod M হয়। তবে egcd ফাংশন x এর মান হিসেবে ঋণাত্মক সংখ্যা দিতে পারে, এ জিনিসটা খেয়াল রাখতে হবে। সেক্ষেত্রে x কে সঠিক ভাবে[১] M দ্বারা mod করে এর অঋণাত্মক মানটা বের করতে হবে।

৩.২ Combinatorics

৩.২.১ Factorial এর পিছের 0

$100!$ এর পিছনে কতগুলি শূন্য আছে? - এটি খুবই common একটি প্রশ্ন। তোমরা হয়তো ইতোমধ্যেই এর সমাধান জানো। যারা জানো না, তাদের জন্য বলি আমাদের বসে বসে $100!$ এর মত এত বিশাল সংখ্যা বের করার দরকার নেই। শুধু আমাদের জানতে হবে যে কত গুলি 10 গুন করা হচ্ছে। কিন্তু একটু খেয়াল করলে দেখব যে, $5! = 120$. এখানে আমরা কোন 10 গুন করিনি, এর পরও একটি 0 চলে এসেছে। কেন? কারণ আমরা 5 আর 2 গুন করেছি, আর এরা আমাদের 10 দিয়েছে। অর্থাৎ আমাদের দেখতে হবে এই গুনফল এর মাঝে কত গুলি 2 আর কতগুলি 5 গুণিতক আকারে আছে। আসলে 2 কত বার আছে তা দেখার দরকার হয় না, কারণ 2 সবসময় 5 এর থেকে বেশি বার থাকবেই, সুতরাং আমাদের 5 গুনলেই চলবে। এখন আমরা চিন্তা করি, 5 কত বার আছে তা কেমনে বের করব। 1 হতে 100 এর মাঝে সর্বমোট $\lfloor 100/5 \rfloor = 20$ টি 5 এর গুণিতক আছে। এগুলি থেকে একটি করে 5 পাব।

[১] x = (x mod M + M) mod M

কিন্তু যেগুলি 25 দ্বারা ভাগ যায় তাদের থেকে কিন্তু আরও একটি করে 5 পাবো, আবার যেগুলো 125 দ্বারা বিভাজ্য তাদের থেকে আরও একটি করে পাবো। কিন্তু 125 কিন্তু আমাদের 100 থেকে বড় তাই আমাদের আর 5 এর বড় power গুলোকে দেখার প্রয়োজন নেই। সুতরাং আমাদের 5 এর মোট সংখ্যা $\lfloor 100/5 \rfloor + \lfloor 100/25 \rfloor = 20 + 4 = 24$. অর্থাৎ $100!$ এর পিছনে মোট 24 টা শূন্য থাকবে। আমরা যদি $n!$ এর ভিতরে একটি prime number p কতগুলি আছে সেটা বের করতে চাই তাহলে আমাদের সুত্র হচ্ছেঃ

$$\lfloor n/p \rfloor + \lfloor n/p^2 \rfloor + \lfloor n/p^3 \rfloor + \ldots \text{ যতক্ষণ না শূন্য হয়}$$

৩.২.২ Factorial এর Digit সংখ্যা

$n!$ এর পিছনের 0 এর সংখ্যা নাহয় বুদ্ধি করে বের করা গেল, কিন্তু $n!$ এ কতগুলি ডিজিট আছে তা কেমনে বের হবে? তোমরা যদি কোন একটি calculator এ $50!$ করে দেখ তাহলে হয়তো $3.04140932 \times 10^{64}$ এরকম সংখ্যা দেখতে পাবে। এখান থেকে বুঝতে পারছি যে 3 এর পরও আরও 64 টা সংখ্যা আছে। এখন আমাদের জানা এমন কি কোন ফাংশন আছে যেটা ঐ 10 এর উপরে থাকা power টা আমাদের বলে দেবে? আছে, log. তোমরা তোমাদের calculator এ যদি এই সংখ্যার log নাও তাহলে দেখবে $64.4830\ldots$ এরকম একটি সংখ্যা আসবে। সুতরাং আমরা যদি এর floor নিয়ে এক যোগ করি তাহলেই আমরা number of digits পেয়ে যাব।[১] আমরা যেকোনো সংখ্যার digit সংখ্যা বের করতে চাইলে এই পদ্ধতি কাজে লাগে। তোমাদের হয়তো কেউ কেউ ভাবছ, log নেবার জন্য তো আমাদের আগে $50!$ বের করতে হবে এর পর না log! না, log এর একটি সুন্দর বৈশিষ্ট্য আছে আর তা হলঃ $\log(a \times b) = \log a + \log b$ অর্থাৎ $\log 50! = \log 1 + \log 2 + \ldots \log 50$.

৩.২.৩ Combination: $\binom{n}{r}$

n টি জিনিস হতে r টি জিনিস কত ভাবে নির্বাচন করা যায় তার ফর্মুলা হলঃ $\binom{n}{r} = \frac{n!}{(n-r)!r!}$. অনেক সময়ই n ও r এর মান দেয়া থাকলে আমাদের $\binom{n}{r}$ এর মান নির্ণয় করার দরকার হয়। একটি উপায় হল factorial সমুহের মান আলাদা আলাদা করে নির্ণয় করে গুন ভাগ করা। কিন্তু এতে একটা সমস্যা হয় আর তা হল overflow. যেমন $n = 50, r = 1$ হলে $50!$ বের করতে গেলে আমাদের মানটা overflow করবে কিন্তু আমরা জানি $\binom{50}{1} = 50$. তোমরা যদি ভেবে থাক যে double ডাটা টাইপ ব্যবহার করবা, তাহলে সেটা সম্ভব না। কারণ double ডাটা টাইপ তোমাকে মানের একটা approximation মান দিবে, কখনই exact মান দিবে না।[২] অনেকে যা করে তা হল, $(n-r)!$ বা $r!$ এর মাঝে যেটা বড় তা দিয়ে $n!$ এর সাথে আগেই কাটাকাটি করে ফেলে, এর পর উপরের গুলো গুন এবং নিচের গুলো গুন করে এই দুইটি সংখ্যা ভাগ করে ফলাফল পাওয়া যায়। ফলে উপরের উদাহরন এ উপরে শুধু 50 থাকে আর নিচে থাকে 1 ফলাফল 50. কিন্তু এই method এও উপরেরটা overflow করে যেতেই পারে যেখানে হয়তো ভাগ দেবার পর উত্তরটা আর overflow করবে না। আরও একটি উপায় আছে আর তাহল, আমরা জানি যে $\binom{n}{r}$ সবসময় একটি পূর্ণ সংখ্যা। এর মানে নিচে যেসব সংখ্যা আছে তারা সবাই কাটাকাটির সময় কাটা পরবে। তোমরা উপরের সংখ্যা গুলিকে একটি array তে নাও। এর পর একে একে নিচের সংখ্যা নাও আর ঐ array এর প্রথম থেকে শেষ পর্যন্ত যাও, gcd নির্ণয় করবা আর এই দুইটি সংখ্যাকে কাটবা যতক্ষণ না নিচ থেকে নেয়া সংখ্যাটা 1 হয়ে যায়। এভাবে কাজ করলে তোমার উত্তর কখনই overflow করবে না। এভাবে নানা রকম ভাবে তোমরা $\binom{n}{r}$ এর মান নির্ণয় করতে পারবে, প্রতিটি পদ্ধতিরই সুবিধা অসুবিধা আছে, overflow, time complexity, memory complexity, implementation complexity ইত্যাদি। প্রবলেম এর উপর ভিত্তি করে তোমাদের বিভিন্ন পদ্ধতি অবলম্বন করার দরকার হতে পারে।

অনেক সময় আমাদের exact মান না চেয়ে কোন একটি সংখ্যা দ্বারা mod করার পরের মান চেয়ে থাকে। এক্ষেত্রেও আমাদের নানা রকম পদ্ধতি আছে। যদি mod করতে বলা সংখ্যাটি prime হয়

[১] ceiling নিলে যদি আমাদের সংখ্যাটি 10^x টাইপ এর সংখ্যা হয় তাহলে আমাদের উত্তরটি সঠিক আসবে না।

[২] যদি তোমাকে বলা হয় $\sqrt{2}$ বা π এর মান লিখ তুমি কি তা লিখতে পারবে? পারবে না, তুমি যাই লিখ না কেন সেটা আসলে আসল মানের একটি approximation মান।

তাহলে n! mod p, ModuloInverse(r! mod p, p) এবং ModuloInverse((n-r)! mod p, p) এই তিনটি সংখ্যা গুন করলেই আমারা আমাদের কাঙ্খিত সংখ্যা পেয়ে যাব। তবে এক্ষেত্রে অবশ্যই $p > n$ হতে হবে। আমরা facotiral এর mod মান আগে থেকেই precalculate করে রেখে মাত্র O$(\log n)$ এই এই মান নির্ণয় করতে পারি। mod যদি prime হয় কিন্তু ছোট হয় তাহলে অন্য একটি উপায় আছে এবং সেটি হল Lucas' Theorem. এই theorem বলেঃ

$$\binom{m}{n} \equiv \prod_{i=0}^{k} \binom{m_i}{n_i} (modp)$$

যেখানে, $m = m_k p^k + \ldots + m_1 p^1 + m_0 p^0$ এবং $n = n_k p^k + \ldots + n_1 p^1 + n_0 p^0$.

অর্থাৎ তোমাকে শুধু $\binom{a}{b}$ গুলি জানতে হবে যেখানে $a, b < p$ জিনিসটা কিন্তু তোমরা precalculate করে রাখতে পার।

যদি সংখ্যাটা prime না হয় বা আমাদের অনেক দ্রুত (O(1)) $\binom{n}{r}$ এর মান বের করতে হয় তাহলে আমরা অনেক সময় $\binom{n}{r}$ এর মান precalculate করে $n \times n$ array তে রেখে দেই। এটা তৈরি করতে আমাদের O(n^2) সময় লাগে। এই পদ্ধতির মূল ফর্মুলা হলঃ $\binom{n}{r} = \binom{n-1}{r} + \binom{n-1}{r-1}$[১] কোড ৩.৯ এ এই কোডটি দেয়া হল। তোমরা limncr এর মান পরিবর্তন করে দরকার মত $\binom{n}{r}$ এর মান generate করতে পার।

কোড ৩.৯: ncr.cpp

```
ncr[0][0] = 1;
int limncr = 10;
for(i = 1; i <= limncr; i++)
    for(j = 0; j <= limncr; j++)
    {
        if(j > i) ncr[i][j] = 0;
        else if(j == i || j == 0) ncr[i][j] = 1;
        else ncr[i][j] = ncr[i - 1][j - 1] + ncr[i - ↩
            1][j];
    }
```

৩.২.৪ কিছু special number

কিছু কিছু special number আছে যা প্রবলেম সমাধান করার সময় প্রায়ই আমরা তাদের সম্মুখিন হই। অনেক সময় এসব মান নির্ণয়ের উপায় আমাদের অন্য সমস্যা সমাধানেও বেশ কাজে লাগে। আমরা এরকম কিছু সংখ্যা এই সেকশনে দেখব।

Derangement Number

একটি বক্সে n জন তাদের টুপি রাখল। এরপর প্রত্যেকে একটি করে টুপি ঐ বাক্স থেকে তুলে নিল। কত উপায়ে তারা এমন ভাবে টুপি তুলে নিবে যেন কেউই তাদের নিজেদের টুপি না পায়! যেমন যদি $n = 3$ হয় তাহলে উত্তর 2. BCA এবং CAB এই দুই উপায়েই হতে পারে (আশা করি বুঝতে পারছ যে প্রথম জনের টুপি A, দ্বিতীয় জনের টুপি B ও তৃতীয় জনের টুপি C)। প্রথমে এটি অনেক সহজ মনে হলেও আসলে এই সমস্যাটার সমাধান একটু জটিল। তোমরা চাইলে inclusion exclusion principle দিয়েও এটি সমাধান করতে পার কিন্তু এখানে তোমাদের recurrance এর মাধ্যমে সমাধান করে দেখান হল।

[১]চিন্তা করে দেখ base case কি হওয়া উচিত

মনে কর D_n হল n তম Derangement Number অর্থাৎ n জন মানুষের জন্য উত্তর। এখন এদের মধ্য থেকে প্রথম জনকে নাও। সে নিজের বাদে অন্য $n-1$ জনের টুপি পরতে পারে। ধরা যাক সে X এর টুপি পরেছে। এখন এই X সেই প্রথম জনের টুপি পরতে পারে আবার নাও পারে। যদি প্রথম জনের টুপি সে পরে তাহলে বাকি $n-2$ জন নিজেদের মাঝে D_{n-2} ভাবে টুপি আদান প্রদান করতে পারে সুতরাং এটি হতে পারেঃ $(n-1)D_{n-2}$ ভাবে। আর যদি X প্রথম জনের টুপি না নেয় তাহলে আমরা ধরে নিতে পারি যে ঐ টুপির মালিক এখন X এবং তার ঐ টুপি পরা যাবে না। অর্থাৎ এখন আমাদের কাছে $n-1$ টা মানুষ ও $n-1$ টা টুপি আছে যারা কেউই নিজেদের টুপি পরতে চায় না। এই ঘটনা ঘটতে পারে $(n-1)D_{n-1}$ উপায়ে। এখানে $n-1$ গুন হচ্ছে কারণ আমরা X কে $n-1$ উপায়ে নির্বাচন করতে পারি। সুতরাং আমাদের D_n এর ফরমুলা বা recurrence দাঁড়াচ্ছে,

$$D_n = (n-1)D_{n-2} + (n-1)D_{n-1}$$

Catalan Number

- $2n$ সাইজের কতগুলি Dyck Word আছে? $2n$ সাইজের Dyck Word এ n টি X ও n টি Y থাকে এবং এর কোন prefix এ Y এর সংখ্যা X এর থেকে বেশি নয়। যেমনঃ $n = 3$ এর জন্য Dyck Word গুলি হলঃ XXXYYY, XYXXYY, XYXYXY, XXYYXY এবং XXYXYY.
- nটি opening bracket ও n টি closing bracket ব্যবহার করে কতগুলি সঠিক parentheses expression বানান যায়? যেমনঃ $n = 3$ এর জন্যঃ ((())), ()(()), ()()(), (())() এবং (()()).
- n leaf আলা কয়টি complete binary tree আছে?
- n বাহু বিশিষ্ট একটি polygon কে কত ভাবে triangulate করা যায়?

এরকম অনেক প্রশ্নের উত্তর হল Catalan Number.[১] n তম Catalon Number কে আমরা C_n লিখে থাকি। এর ফর্মুলা হলঃ

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1}$$

একেও আমরা চাইলে recurrence আকারে লিখতে পারি, তবে এই ফর্মুলাটিই বেশি দরকারি। তোমরা যেকোনো discrete math বই এ এই ফর্মুলা কেমনে সঠিক তা দেখে নিতে পারো।

Stirling Number of Second Kind

nটা আলাদা জিনিসকে k ভাগে যত ভাগে ভাগ করা যায় তাই হল Stirling Number of Second Kind. একে $\left\{{n \atop k}\right\}$ দ্বারা প্রকাশ করা হয়। মনে কর $n = 3$ ও $k = 2$ এক্ষেত্রে উত্তর কিন্তু 3, $(AB, C), (AC, B), (BC, A)$. এখন এর recurrence বের করার জন্য আমরা কিছুটা Derangement Number বের করার মত করে চিন্তা করব। প্রথমে n টি জিনিস থেকে সর্বশেষ জিনিসটা নেই। এখন এই জিনিসটা একাই একটি ভাগে থাকতে পারে। সেক্ষেত্রে বাকি $n-1$ টা জিনিস $k-1$ ভাগে ভাগ করতে হবে (খেয়াল কর, আমরা শুধু মাত্র শেষটাই আলাদা করে নিয়েছি)। এটা সম্ভব $\left\{{n-1 \atop k-1}\right\}$ ভাবে। আবার এটাও সম্ভব যে, বাকি $n-1$ টা জিনিস k ভাগে আছে আর শেষ জিনিসটা এদেরই কোন একটায় আছে। এই কোন একটি k টার মাঝের কোন একটি। সুতরাং এটি হতে পারে $k\left\{{n-1 \atop k}\right\}$ ভাবে। অর্থাৎ,

$$\left\{{n \atop k}\right\} = \left\{{n-1 \atop k-1}\right\} + k\left\{{n-1 \atop k}\right\}$$

[১]তোমারা চাইলে http://en.wikipedia.org/wiki/Catalan_number হতে আরও interpretation গুলি পড়ে দেখতে পার।

এখন যেকোনো recurrance এর base case থাকে। $k = 1$ হলে সব জিনিসকে এক ভাগে কতভাবে ভাগ করা যায়? মাত্র এক ভাবেই তাই না? আর যদি $k = n$ হয়? অর্থাৎ n টি জিনিসকে n ভাগে কত ভাবে ভাগ করা যায়? এক ভাবেই। অর্থাৎ base case হলঃ

$$\left\{ {n \atop 1} \right\} = \left\{ {n \atop n} \right\} = 1$$

Stirling Number of First Kind

nটা আলাদা জিনিসকে k টি cycle এ যত ভাগে ভাগ করা যায় তাই হল Stirling Number of First Kind. একে $\left[{n \atop k} \right]$ দ্বারা প্রকাশ করা হয়। মনে কর $n = 4$ ও $k = 2$ এক্ষেত্রে উত্তর কিন্তু 11, (AB, CD), (AD, BC), (AC, BD), (A, BCD), (A, BDC), (B, ACD), (B, ADC), (C, ABD), (C, ADB), (D, ABC), (D, ACB)). এটা একটু জটিল লাগতে পারে তাই বলে নেই যে, AB আর BA কিন্তু একই cycle নির্দেশ করে, কারণ cycle এর আলাদা জায়গা থেকে শুরু করলে তুমি BA পাবে। আবার (AB, CD) আর (CD, AB) কিন্তু একই। কারণ cycle এর অর্ডার কোন ব্যপার না। যাই হোক, আমরা stirling number of first kind এর recurrence ও আগের মত একই ভাবে বের করতে পারি। প্রথমে n টি জিনিস থেকে সর্বশেষ জিনিসটা নেই। এখন এই জিনিসটা একাই একটি cycle এ থাকতে পারে। সেক্ষেত্রে বাকি $n-1$ টা জিনিস $k-1$ cycle এ ভাগ করতে হবে (খেয়াল কর, আমরা শুধু মাত্র শেষটাই আলাদা করে নিয়েছি)। এটা সম্ভব $\left[{n-1 \atop k-1} \right]$ ভাবে। আবার এটাও সম্ভব যে, বাকি $n-1$ টা জিনিস k cycle এ আছে আর শেষ জিনিসটা এই $n-1$টার মাঝে কোন একটির পর থাকে। সুতরাং এটি হতে পারে $(n-1)\left[{n-1 \atop k} \right]$ ভাবে। অর্থাৎ,

$$\left[{n \atop k} \right] = \left[{n-1 \atop k-1} \right] + (n-1)\left[{n-1 \atop k} \right]$$

এখন যেকোনো recurrance এর base case থাকে। $k = 1$ হলে সব জিনিসকে একটি cycle এ কত ভাগে ভাগ করা যায়? এটি একটি common সমস্যা এর উত্তর $(n-1)!$ আর যদি $k = n$ হয়? অর্থাৎ n টি জিনিসকে n cycle এ কত ভাবে ভাগ করা যায়? এক ভাবেই। অর্থাৎ base case হলঃ

$$\left[{n \atop 1} \right] = (n-1)!, \left[{n \atop n} \right] = 1$$

৩.২.৫ Fibonacci Number

ফিবনাচি নাম্বার এর সাথে তোমরা ইতোমধ্যেই পরিচিত হয়ে গেছো এবং হয়তো $n \leq 10^6$ এর জন্য ফিবনাচি নাম্বারও বের করে ফেলেছ। কিন্তু যদি আরও বড় বের করতে বলা হয়, ধর $n \leq 10^{18}$ এর জন্য (আমরা কিন্তু mod মান বের করব)? এর জন্য একটি সুন্দর method আছে। তোমরা যারা matrix জানো না তারা একটু matrix পড়ে নিতে পার। matrix ছাড়াও এটি করা যায় কিন্তু matrix ব্যবহার করে এই সমাধানটা একটি general method, সুতরাং তোমরা এই method ব্যবহার করে আরও অন্য অনেক problem সল্ভ করতে পারবে। Matrix ব্যবহার করে আমরা লিখতে পারিঃ

$$\begin{aligned}
\begin{bmatrix} F_2 \\ F_1 \end{bmatrix} &= \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F_1 \\ F_0 \end{bmatrix} \\
\begin{bmatrix} F_3 \\ F_2 \end{bmatrix} &= \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F_2 \\ F_1 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F_1 \\ F_0 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^2 \begin{bmatrix} F_1 \\ F_0 \end{bmatrix}
\end{aligned}$$

একই ভাবে,

$$\begin{bmatrix} F_4 \\ F_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^3 \begin{bmatrix} F_1 \\ F_0 \end{bmatrix}$$

সুতরাং আমরা লিখতে পারি,

$$\begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1} \begin{bmatrix} F_1 \\ F_0 \end{bmatrix}$$

তোমরা যদি ভেবে থাক ঐ matrix এর power তুলতে গেলে তো খবর হয়ে যাবে, তাহলে ভুল ভাববে। কিছুক্ষণ আগেই কিন্তু আমরা BigMod শিখে এসেছি, ওখানে ছিল একটি সংখ্যা, এখানে আছে matrix. সংখ্যা গুন করার পরিবর্তে তুমি শুধু matrix গুন করবে তাহলেই হবে। আমি তোমাদের পরামর্শ দিব তোমরা নিজেরাই এই জিনিসটা কোড কর। প্রথম দিকে যদিও কোড করতে একটু সমস্যা হবে কিন্তু দু একবার নিজে থেকে কোড করলে দেখবে অনেক সহজ হয়ে যাবে। Stirling Number ও এই পদ্ধতিতে বের করা যায়, তবে সেক্ষেত্রে $k \times k$ আকারের matrix লাগে। সুতরাং বুঝতেই পারছ যে k কে ছোট হতে হবে কিন্তু n অনেক বড় হতেই পারে। চেষ্টা করে দেখতে পার সমাধান করতে পার কিনা।

৩.২.৬ Inclusion Exclusion Principle

Counting এর ক্ষেত্রে অত্যন্ত গুরুত্বপূর্ণ একটি method হল Inclusion Exclusion Principle. মনে কর তোমাকে বলা হল, 1 হতে 100 পর্যন্ত কতগুলি সংখ্যা আছে যাদের 2 বা 3 দ্বারা ভাগ যায়? খেয়াল কর, সংখ্যা বিভিন্ন ধরনের হয়। কিছু সংখ্যা আছে শুধু 2 দ্বারা ভাগ যায় এমন, কিছু আছে শুধু 3 দ্বারা ভাগ যায়, কিছু আছে 2 আর 3 দুইটি দিয়েই যায়, আবার কিছু আছে যা কোনটা দিয়েই ভাগ যায় না। শুধু 2 দ্বারা কত গুলি ভাগ করা যায় তা বের করা কিন্তু খুব সহজ, $\lfloor 100/2 \rfloor = 50$. একই ভাবে শুধু 3 দ্বারা যায় এরকম আছে $\lfloor 100/3 \rfloor = 33$. এখন এদের যোগ করলেই কিন্তু তোমাদের উত্তর পেয়ে যাবে না। কারণ, প্রথম গ্রুপে $6, 12, 18 \ldots$ এরকম কিছু সংখ্যা আছে যারা দ্বিতীয় গ্রুপেও আছে। সুতরাং আমরা যদি এদের যোগ করে দেই এই জিনিস গুলো double count হয়ে যাবে। একটা সহজ উপায় হল এই double count বের করে তা বিয়োগ করে দেয়া। খেয়াল করলে দেখবে, এই double count গ্রুপ এ তারাই আছে যারা 2 ও 3 দুইটি দ্বারাই ভাগ যায় অর্থাৎ 6 দ্বারা ভাগ যায়। 6 দ্বারা ভাগ যায় এরকম মোট $\lfloor 100/6 \rfloor = 16$ টি সংখ্যা আছে। সুতরাং আমাদের ফলাফল হবেঃ $50 + 33 - 16 = 67$. এভাবে count করার method ই হল inclusion exclusion principle. যদি 2 টিরও বেশি সংখ্যা দেয়া থাকে তাহলে কি করতে হবে একটু চিন্তা করে দেখতে পার। মনে কর তোমাকে $2, 3, 5, 7$

এরকম চারটি সংখ্যা দেয়া হল। তাহলে একটি দ্বারা ভাগ যায় এরকম সংখ্যা যোগ করে, দুইটি দিয়ে ভাগ যায় এরকম সংখ্যা আবার বিয়োগ করবা, কিন্তু তিনটি সংখ্যা দ্বারা ভাগ যায় এরকম সংখ্যা আবার যোগ করবা এবং চারটি সংখ্যা দ্বারা ভাগ যায় সংখ্যা গুলি বিয়োগ করবা। অর্থাৎ, বিজোড় সংখ্যার সময় যোগ ও জোড় এর ক্ষেত্রে বিয়োগ করতে হয়। এই ধরনের সমস্যায় সাধারনত time complexity হয়ে থাকে $O(2^n)$. আরেকটা জিনিস তোমাদের বলে রাখি এসব ক্ষেত্রে bitmask ব্যবহার করে কোড করলে অনেক সহজেই কোড হয়ে যায়। যদি তোমাদের কাছে n টা সংখ্যা থাকে এবং কোনটা কোনটা দিয়ে ভাগ করবা এটা সিদ্ধান্ত নিতে চাও তাহলে তুমি n bit এর বিভিন্ন নাম্বার নিবা, কোন একটি bit এ 1 থাকার মানে ঐ সংখ্যা তুমি নিবা, 0 মানে নিবা না। এরকম করে খুব সহজে একটা loop দিয়েই তুমি এই নেয়া-না নেয়ার কাজটা করে ফেলতে পার।

৩.৩ সম্ভাব্যতা

কোন কারণে আমরা ছোট বেলা থেকে এই জিনিসের প্রতি একরকম ভীতি নিয়ে বেড়ে উঠি। কারণ এই জিনিস বুঝতে আমাদের কষ্ট হয় বা আমাদের হয়তো ভালো মত বুঝানো হয় না। এখানে আসলে খুব details এ তোমাদের এসব জিনিস দেখানো সম্ভব না কিন্তু খুব অল্প কথায় কিছু লিখার চেষ্টা করলাম।

৩.৩.১ Probability

মনে কর ক্রিকেট খেলা শুরুর আগে দুই ক্যাপ্টেইন টস করতে গেল। টস এর সময় Head পড়বে না Tail পড়বে? যেকোনো একটা পড়তে পারে! আমরা বলে থাকি chance 50-50. এখন মনে কর লুডু খেলায় একটা ছক্কা আছে, কিন্তু এই ছক্কায় কোন 5 নেই তার পরিবর্তে আরও একটি 6 আছে। এখন তোমাকে যদি জিজ্ঞাসা করা হয় এখানে কত পড়বে? তুমি কিন্তু নিশ্চিত ভাবে বলতে পারবে না যে এখানে অমুক সংখ্যাই পড়বে। আবার তুমি একটু যদি mathematical উত্তর দাও তাহলে বলবে এখানে 5 কখনই পড়বে না, 6 পড়ার সম্ভাবনা $1, 2, 3, 4$ পড়ার থেকে বেশি। আবার এও বলতে পার যে $1, 2, 3, 4$ পড়ার সম্ভাবনা একই। আমরা কিন্তু কোন হিসাব করে এ কথা বলতেসি না, শুধু common sense থেকেই এই কথা বললাম। তাই না? এখন যদি তোমাকে বলা হয় ঠিক মত হিসাব করে বের কর কোনটা পড়ার সম্ভাবনা কত? এই হিসাবটাই কেমনে করতে হয় তা এখন দেখা যাক।

সম্ভাব্যতা অংকের মূল নীতি হলঃ

$$\text{কোন ঘটনা ঘটার সম্ভাব্যতা} = \frac{\text{কত ভাবে এই ঘটনা ঘটতে পারে}}{\text{কত ভাবে সকল ঘটনা ঘটতে পারে}}$$

যেমন, আমাদের ক্রিকেট খেলার টস এ, Head পড়ার probability হলঃ $\frac{1}{2}$ কারণ আমাদের মাত্র দুইভাবেই coin পড়তে পারে Head ও Tail আর এদের মাঝে একটাই Head. এখন আমাদের কিছুক্ষণ আগের ছক্কার ক্ষেত্রে যদি আমরা হিসাব করতে চাই 5 পড়ার probability কত, তাহলে কিন্তু $\frac{0}{6} = 0$. অর্থাৎ 5 পাবই না! আবার $1, 2, 3$ বা 4 পড়ার probability হল $\frac{1}{6}$ আর 6 পড়ার probability হল $\frac{2}{6}$. অর্থাৎ 6 পড়ার সম্ভাবনা কিন্তু অন্য কোন একটি সংখ্যা পড়ার সম্ভাবনা থেকে বেশি।

তোমরা হয়তো π নির্ণয় করার নানা মজার মজার method দেখেছ। এমনই একটি method এখন বলব। তোমরা একটি বড় বর্গ আঁক। এর মাঝে একটি বৃত্ত আঁক যা চার বাহুকেই স্পর্শ করে। এখন তুমি ছোট ছোট কিছু জিনিস নাও, মনে কর n টা চুমকির মত জিনিস। এখন এগুলি বর্গক্ষেত্রের মাঝে randomly ছড়িয়ে দাও। এখন তুমি গুনে দেখ বৃত্তের মাঝে কত গুলি আছে। ধরা যাক, b টা। তাহলে $\frac{4b}{n}$ হবে প্রায় π এর সমান। অদ্ভুত না? এমন কেন হল? এর যুক্তি কিন্তু খুবই সহজ। বৃত্তের radius যদি r হয় তাহলে বৃত্তের ক্ষেত্রফল πr^2 আর বর্গের ক্ষেত্রফল $4r^2$. সুতরাং তুমি যদি কোন একটি চুমকি randomly ফেল এর মাঝে তাহলে বৃত্তের মাঝে পড়ার সম্ভাবনা $\frac{\pi r^2}{4r^2} = \frac{\pi}{4}$. আবার আমরা আমাদের experiment এ n টা বিন্দু randomly ফেলেছিলাম এবং বৃত্তের মাঝে পড়েছে b টা, সুতরাং আমরা experiment থেকে পাই বৃত্তের মাঝে একটি চুমকি পড়ার সম্ভাবনা $\frac{b}{n}$ অর্থাৎ

$\frac{b}{n} \approx \frac{\pi}{4}$ এখান থেকেই আমরা পাই, $\pi \approx \frac{4b}{n}$. তুমি n যত বড় নিবে এই experiment থেকে তত accurate π এর মান পাবে।

৩.৩.২ Expectation

মনে কর তোমাকে বলা হল একটি coin টস করার পর যদি head পড়ে তাহলে তুমি 0 টাকা পাবে কিন্তু tail পড়লে 100 টাকা পাবে। তুমি কত পাবার আশা কর? তুমি যদি বল যে আমি 100 টাকা পাবার আশা করি, তাহলে হল না, তুমি বেশি আশাবাদী হয়ে গেলে। কারণ তোমার 50% সম্ভাবনা আছে তুমি 0 টাকা জিতবে। আবার তুমি হতাশার সুরে যদি বল যে নাহ আমি একটা টাকাও পাবো না, আমার কপাল ভালো না। তাহলেও হল না। তুমি যদি একজন mathematician এর সুরে কথা বল তাহলে বলবে আমি 50 টাকা পাবার আশা করি। প্রথম প্রথম সবাই মনে করত পার- এটা কেমন কথা হল? হয় আমি 0 পাবো নাহয় 100 পাবো, 50 কই থেকে আসল? কাহিনী হল, তুমি যদি এই টস 10 বার কর, তাহলে এটা বলাই যায় যে 5 বার এর মত head পড়েছে আর বাকি 5 বার tail. অর্থাৎ তুমি 10 বারে মোট 500 টাকা পাবে, অর্থাৎ গড়ে তুমি প্রতিবার 50 টাকা পাবে। এই জন্যই এক্ষেত্রে তোমার expectation হল 50 টাকা। আরও একটি উদাহরন দেয়া যাক, ধর তুমি সাপ লুডু খেলতেছ তোমাকে যদি বলা হয় তোমার খেলা শেষ করতে কত চাল লাগতে পারে? তুমি কিন্তু হিসাব নিকাশ করে দেখাতে পারবে যে তোমার expected number of move কত। দেখা যাবে তুমি যদি বহুবার সাপ লুডু খেল তাহলে গড়ে ঐ সংখ্যক move লাগবে। কোন কিছুর expectation বের করার নিয়ম হচ্ছে যত রকম ঘটনা ঘটতে পারে তাদের probability গুন ঐ ঘটনা ঘটলে তোমার সেই কোন কিছু কত হবে। যেমন, coin টস এর ক্ষেত্রে তোমার দুইরকম ঘটনা ঘটতে পারে। head বা tail. head পড়ার probability 0.5 এবং এটি পড়লে তুমি 0 টাকা পাবে। আর যদি 0.5 probability তে tail পড়ে তাহলে তুমি পাবে 100 টাকা। অতএব তোমার টাকা পাবার expectation হবে $0.5 \times 0 + 0.5 \times 100 = 50$.

আরও একটি উদাহরন হিসাবে চিত্র ৩.১ এর ছোট পরিসরে সাপ লুডু বিবচনা করা যাক।

1	2	3	4	5

চিত্র ৩.১: একটি ছোট লুডু খেলা

মনে কর তুমি 1 এ আছ। তুমি যদি 5 এ যাও তাহলেই খেলা শেষ হয়ে যাবে। মাঝের গাঢ় রঙ আলা 3 এ তুমি যেতে পারবে না কখনই। তোমাকে একটি ছক্কা দেয়া হল যেটা ছুড়লে 1 হতে 6 এর মাঝের কোন একটি সংখ্যা পড়ে এবং তুমি ওত সংখ্যক ঘর সামনে যেতে পারবে। কিন্তু সেই ঘর যদি 3 হয় বা 5 পেরিয়ে যায় তাহলে আবারো তোমাকে চালতে হবে। তুমি যখন 5 এ পৌঁছাবে তখন খেলা শেষ হবে। Expected number of move কত?

ধরা যাক, T_i হল i এ থাকা কালীন সময়ে খেলা শেষ করার জন্য expected number of move. আমাদের T_1 বের করতে হবে। শেষ থেকে আসা যাক। $T_5 = 0$ কারণ তুমি যদি 5 এ থাকো তাহলে তো খেলা শেষ। কোন move না দিয়েই তোমার খেলা শেষ হয়ে যাবে। সে জন্য এটি শূন্য। এখন 4 এ আসা যাক। এখান থেকে খেলা শেষ করতে আমাদের লাগবে T_4 সংখ্যক move. খেয়াল কর, যদি 1 ব্যতিত কোন সংখ্যা পড়ে তাহলে কিন্তু তুমি 4 এই থাকবে, অর্থাৎ তোমার আরও T_4 সংখ্যক move লাগবে। ব্যপারটা আরও একটু পরিস্কার করা যাক, তুমি ধরেই নিয়েছ যে 4 থেকে খেলা শেষ করতে T_4 move লাগবে। এখন তোমার যদি এখানেই থাকতে হয় তাহলে তো T_4 move লাগবে তাই না? আর যদি 1 পড়ে তাহলে লাগবে T_5 move. অর্থাৎ $1/6$ probability তে লাগবে T_5 move আর $5/6$ probability তে লাগবে T_4 move. আর ভুলে যেও না এই মাত্র তুমি একটা চাল দিলে ছক্কা গড়িয়ে, সুতরাং $T_4 = 1 + \frac{1}{6}T_5 + \frac{5}{6}T_4$ এখান থেকে আমরা পাই, $T_4 = 6$. হিসাব টা কিন্তু ঠিকি আছে। ছক্কার ছয় দিক, আমরা আসা করতেই পারি যে 6 চালের মাঝে প্রতিটি সংখ্যা একবার না একবার আসবেই। সুতরাং আমাদের expectation 6. T_3 আমাদের লাগবে না, কারণ

আমরা এখানে কখনই আসব না। এখন আসা যাক, 2 এ। যদি $1/6$ probability তে 2 পড়ে তাহলে লাগবে T_4 move, যদি $1/6$ probability তে 3 পড়ে তাহলে T_5 move লাগবে, আর বাকি $4/6$ probability তে যা পড়বে তার জন্য আমাদের 2 এই থাকতে হবে অর্থাৎ T_2 move লাগবে। সুতরাং, $T_2 = 1 + \frac{1}{6}T_4 + \frac{1}{6}T_5 + \frac{4}{6}T_2$ অর্থাৎ $T_2 = 6$. আশা করি T_1 এর জন্য ফর্মুলা তুমি বুঝতে পারছ কি হবে, $T_1 = 1 + \frac{1}{6}T_2 + \frac{1}{6}T_4 + \frac{1}{6}T_5 + \frac{3}{6}T_1 \Rightarrow T_1 = 6$. অর্থাৎ expected number of move হবে 6.

৩.৪ বিবিধ

৩.৪.১ Base Conversion

আমরা যেই সংখ্যা পদ্ধতি ব্যবহার করে থাকি তাকে দশমিক বলে কারণ এর base হল 10. কম্পি-উটার যেই সংখ্যা পদ্ধতি ব্যবহার করে তার base হল 2 একে বলে বাইনারি। এরকম আরও কিছু বহুল প্রচলিত সংখ্যা পদ্ধতি আছে- অকটাল (base হল 8), হেক্সাডেসিমাল (base হল 16)। বলা হয়ে থাকে আমাদের হাতের দশ আঙ্গুলের জন্য আমাদের সংখ্যা পদ্ধতি হল Decimal বা দশমিক। কম্পিউটার এর জন্য 10 টি আলাদা আলাদা সংখ্যা নিয়ে হিসাব করা বেশ কষ্টকর এজন্য শুধু মাত্র voltage up down করে বুঝা যায় এমন একটি সংখ্যা পদ্ধতি ব্যবহার করা হয় কম্পিউটারে। এটিই হল বাইনারি। এই পদ্ধতিতে অঙ্ক আছে দুইটি 0 ও 1. আমরা কেমনে হিসাব করি একটু খেয়াল করঃ $0, 1, \ldots 9, 10, 11, 12, \ldots 19 \ldots$ অর্থাৎ আমাদের শেষ digit টা এক এক করে বাড়ে এর পর শেষ হয়ে গেলে এর বামের টা এক বাড়ে ওটাও শেষ হয়ে গেলে তারও বামের টা বাড়বে। বাইনারিও সে রকমঃ $0, 1, 10, 11, 100, 101, 110, 111, 1000 \ldots$। অন্যান্য number system এও একই রকম হয়ে থাকে।

এখন যদি একটু খেয়াল কর দশমিক number system এ 481 কে আমরা ভেঙ্গে লিখতে পারিঃ $4 \times 10^2 + 8 \times 10^1 + 1 \times 10^0$ একই ভাবে বাইনারি 1010 কেও আমরা এভাবে ভেঙ্গে লিখতে পারিঃ $1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$ যার মান দশমিক পদ্ধতি তে হবে 10. সুতরাং আমাদের যদি অন্য কোন number system এ একটি সংখ্যা দেয়া হয় তাকে আমরা খুব সহজেই আমাদের দশমিক number system এ পরিবর্তন করতে পারি। আমরা ডান থেকে iতম স্থান এ যাব এবং সেখানে থাকা অংককে $base^i$ দিয়ে গুন করে সবগুলি যোগ করলেই আমরা দশমিক পদ্ধতিতে সংখ্যাটি পেয়ে যাব।

কিন্তু আমরা যদি কোন একটি দশমিক সংখ্যাকে অন্য আরেকটি number system এ পরিবর্তন করতে চাই? ধরা যাক আমরা b base এ পরিবর্তন করতে চাই। তাহলে অবশ্যই আমাদের সংখ্যাটা হবে এরকমঃ $a_n \times b^n + \ldots + a_1 \times b^1 + a_0 \times b^0$ আমরা যদি এই সংখ্যা কে b দ্বারা ভাগ করি তাহলে ভাগশেষ হবে a_0 এবং ভাগ করার ফলে সব power গুলি কিন্তু 1 করে কমে গেছে, সুতরাং এর পরে আবারো b দ্বারা ভাগ করলে ভাগশেষ হবে a_1 এভাবে একে একে আমরা ডান থেকে বামের সব সংখ্যা পেয়ে যাব।

৩.৪.২ BigInteger

অনেক সময় দেখা যায় আমাদের প্রবলেম এ mod করতে বলা হয় না, আবার আমাদের উত্তরটাও বেশ বড় হয়। সেক্ষেত্রে আমাদের BigInteger ব্যবহার করতে হয়। যারা Java জানো তাদের জন্য এটা একটা advantage কারণ Java তে BigInteger নামে একটি library আছে। কিন্তু মাঝে মাঝে এটি বেশ slow হওয়ায় দেখা যায় TLE খেতে হয়। আমরা কিন্তু খুব সহজেই C তে নিজেদের BigInteger এর হিসাব নিকাশের জন্য function লিখে ফেলতে পারি। যোগ বিয়োগ ও গুন বেশ সহজ। ভাগ একটু কঠিন। খেয়াল করলে দেখবে যে, আমরা কাগজে কলমে যোগ বিয়োগ বা গুন সব সময় ডান দিক থেকে করি, এবং এই সময় যেই দুইটি সংখ্যা নিয়ে হিসাব করছি তাদের কে ডান দিকে একই বরাবর রাখা হয়, আমরা কিন্তু বাম দিকে একই বরাবর রাখি না, রাখলে ভুল হবে। কিন্তু আমরা যখন কম্পিউটারে array এর চিন্তা করছি তখন আমরা কল্পনা করি বাম দিক থেকে। এটা অনেক সময় সমস্যা হয়। যেমন মনে করা যাক আমরা 100 digit এর একটি সংখ্যার সাথে 50 digit এর একটি সংখ্যা যোগ করব। এখন এই দুইটি সংখ্যা যখন string আকারে input নিব তখন এটা বাম দিকে

align হয়ে থাকে। এই অবস্থায় কোন হিসাব করা বেশ কঠিন। এ জন্য যা করা উচিত তাহল, সংখ্যাকে উলটিয়ে নেয়া, এতে করে সখ্যার একক এর অংক সবসময় আমাদের বামে থাকে, আর যেহেতু আমরা কম্পিউটারে সংখ্যাকে বাম align করে চিন্তা করছি সেহেতু আর কোন সমস্যা হবে না। আমরা এখন বাম দিক থেকে ডান দিকে যাব আর হিসাব করব। এসময় আরও একটি গুরুত্বপূর্ণ জিনিস খেয়াল রাখলে ভালো হয় যে, আমরা যেহেতু জানি না যে আমাদের উত্তরটা কত গুলি digit হবে সেহেতু আমাদের result রাখার array কে 0 দ্বারা initialize করে নিতে হবে। এতে করে যোগ বিয়োগ বা গুন করতে কোন সমস্যা হবে না। এখন কথা হল কেমনে যোগ বিয়োগ গুন করব? খুবই সোজা, তুমি যেভাবে কাগজে কলমে কর ঠিক সেভাবে। তুমি নিজে একটু ভেবে দেখ কেমনে যোগ বিয়োগ কর? তুমি যোগ করার পর যোগফল 10 বা এর বড় হলে হাতে কিছু একটা থাকে, সেটা গিয়ে পরের ঘরের সাথে যোগ কর এভাবে চলতে থাকে। আবার বিয়োগ এর সময় তুমি কিছু ধার নাও যেটা পরে গিয়ে আবার শোধ করে দাও। ঠিক এই জিনিসগুলিই তোমাদের লজিকের মাধ্যমে লিখতে হবে। গুনের ক্ষেত্রে তোমাদের একটু ঝামেলা মনে হতে পারে। তোমরা একটু চিন্তা করে দেখ আমরা যে গুন করে সঙ্খ্যাগুলি পর পর লিখি এর পর যোগ করি তা না করে, যদি গুন করতে করতে যোগ করি? এটা আমাদের হাতে হাতে করতে বেশ কষ্ট হবে, কিন্তু কম্পিউটারে এই প্রোগ্রাম লিখা বেশ সহজ হবে। মোট কথা BigInteger এর যোগ, বিয়োগ, গুন এসব আসলে common sense থেকে করার বিষয়।

৩.৪.৩ Cycle Finding Algorithm

UVa 11036 প্রবলেমটা দেখতে বেশ কঠিন হলেও এর idea টা কিন্তু খুব সহজ। একটা x এর ফাংশন দেয়া থাকবে যেমন ধরা যাক, $f(x) = x * (x+1) \mod 11$, এখন একটি n এর মানের জন্য $f(n), f(f(n)), f(f(f(n)))\dots$ এই ধারার period বের করতে হবে। অর্থাৎ কত length বার বার repeat হবে। যেমন যদি $n = 1$ হয় তাহলে এই ধারার মান গুলি হবেঃ $2, 6, 9, 2, 6, 9, 2, 6, 9\dots$ এখানে তিনটি সংখ্যা বার বার পুনরাবৃত্তি করছে, সুতরাং এখানে period হবে 3. একটু চিন্তা করলে দেখবে যে এখানে আসলে কখনও যদি আগের একটি সংখ্যা আসে, তাহলে এর পরের সংখ্যা গুলি আগের মত আসতে থাকবে। যেমন উপরের ধারায় চতুর্থ পদে 2 চলে এসেছে যা প্রথম পদের সমান, সুতরাং এর পঞ্চম পদ হবে দ্বিতীয় পদের সমান এবং এভাবে পর পর একই সংখ্যা আসতে থাকবে। আমাদের আসলে বের করতে হবে প্রথম repeatation কখন হবে। এই জিনিসটা একটা array রেখে খুব সহজেই করা যায়। আমরা একটি একটি করে মান বের করব আর array তে দেখব যে এই জিনিসটা আগে এসেছিল কিনা। যদি না আসে তাহলে আমরা পরবর্তীতে ব্যবহারের জন্য array তে লিখে রাখব যে এই সংখ্যাটা ধারার অমুক position এ এসেছিল। আর যদি আগেই এসে থাকে তাহলে, আগে কোন জায়গায় এসেছিল তা আমরা array থেকেই দেখতে পারব, আর এখন কোন পদে আসলাম তাও আমরা জানি। এই দুই সংখ্যা থেকে আমরা এর period বের করে ফেলতে পারি। কিন্তু এভাবে array ব্যবহার করে করতে গেলে আমাদের memory complexity দাঁড়ায় $\mathrm{O}(N)$ যেখানে N হল সর্বোচ্চ সম্ভাব্য মান। যদিও আমাদের time complexity ও $\mathrm{O}(n)$ বা আরও সুনির্দিষ্ট ভাবে বলতে গেলে, $\mathrm{O}(\lambda + \mu)$ যেখানে $\mu =$ শুরু থেকে cycle এর শুরু পর্যন্ত দুরত্ব এবং $\lambda =$ cycle এর length. আমরা চাইলে memory complexity কমিয়ে $\mathrm{O}(1)$ এ নামাতে পারি এবং সেক্ষেত্রেও আমাদের time complexity একই থাকবে। এই method কে বলা হয় Floyd's Cycle Finding Algorithm.

এই algorithm টা বেশ মজার। আমরা যেই ধারার cycle বের করতে চাচ্ছি সেই ধারার শুরুতে একটি খরগোশ আর একটি কচ্ছপ রাখতে হবে। এর পর প্রতি ধাপে খরগোশ দুই ধাপ আর কচ্ছপ এক ধাপ করে যাবে (দুই ধাপ যাবে মানে $f(f(x))$ আর এক ধাপ যাওয়া মানে $f(x)$)। এক সময় না এক সময় দুজনে মিলিত হবেই। এখন খরগোশকে ঐ জায়গাতে রেখেই কচ্ছপ কে একধাপ এক ধাপ করে আগাতে হবে যতক্ষণ না সে আবার খরগোশ এর জায়গায় ফেরত আসে। যত ধাপে সে ফেরত আসে সেটাই হল period বা λ. এবার কচ্ছপ কে ঐ জায়গা তে রেখে খরগোশ কে আবার শুরুতে নিয়ে যাও তবে এবার খরগোশ আর কচ্ছপ দুজনই একধাপ একধাপ করে আগাবে যতক্ষণ না একত্র হয়। এটা খুব সহজেই প্রমাণ করা যায় যে, যে কয় ধাপে মিলিত হল সেটাই μ.

৩.৪.৪ Gaussian elimination

মনে করো তোমাকে দেয়া আছে $x+2y=5$ আর $3x+2y=7$. তোমাকে বলতে হবে x এবং y এর মান কত বা বলতে হবে এদের কোন সমাধান নাই বা অসীম সংখ্যক সমাধান আছে। দুইটি variable হলে তো জিনিসটা খুবই সহজ তাই না? হাতে হাতে করা যায়। কিন্তু যদি তোমাকে n টা variable আলা n টা equation দেয়? সত্যি কথা বলতে প্রথম যখন আমি নিজে এরকম সমস্যা সম্মুখীন হয়েছিলাম তখন বেশ ভয় লেগেছিল এবং বুঝছিলাম না যে এতো কঠিন সমস্যা কেমনে সমাধান করা যায়। যদি আমি ভুল না করে থাকি তাহলে সেটি BUET এ কোন এক practice কন্টেস্ট এ ছিল। আমি আর নাফি দুইজন একদলে ছিলাম আর অন্যান্য দলের মাঝে ছিল সেবারের BUET এর world finalist team, যত দূর সম্ভব ডলার ভাই, মাহমুদ ভাই এবং সানি ভাই এর দল। কন্টেস্টটায় কতগুলি সমস্যা ছিল ঠিক মনে নেই তবে উনারা $4-5$টি সমস্যার সমাধান করেছিলেন যার একটিও আমরা পারি নাই কিন্তু এই সমস্যাটার সমাধান আমরা করেছিলাম যা উনারা করেন নাই! সুতরাং বুঝতেই পারছ এই সমস্যাটা ওতটা কঠিন না। সত্যি কথা বলতে খুবই সহজ। আমাদের স্কুল বা কলেজের গণিতের জ্ঞান দিয়েই এর সমাধান সম্ভব। তুমি নিজে চিন্তা করো ধরো তোমাকে যদি 4 variable এর 4 টি equation দেয় তাহলে তুমি হাতে হাতে কেমনে সমাধান করবে? মনে করো variable গুলি হল x_0, x_1, x_2 এবং x_3. তুমি যা করবে তাহলো প্রথম সমীকরণ নিয়ে তাতে x_0 এর মান বের করে বাকি গুলিতে বসায়ে দাও, তাহলে কি হল? বাকি 3 টি equation এ 3 টি করে variable থাকবে। আবার তুমি এই কাজ করলে দুইটি equation এ দুইটি variable থাকবে, এবং আরেকবার করলে একটি variable ও একটি equation. এবার এই মান আগের equation গুলিতে বসাতে থাকলে একে একে সবগুলির মান পেয়ে যাবে। এটাই হল মূল idea. তবে এই জিনিস হাতে হাতে করা যত সহজ কোডে করা ওতটা সহজ হবে না। কিছু কিছু special case এসে পড়বে। এসব এড়িয়ে কেমনে সহজে এটা সমাধান করা যায় তা আমরা এখন দেখব।

প্রথমত n টি equation কে এভাবে কল্পনা করো যেন তাদের সব variable বাম দিকে আর constant ডান দিকে থাকে। এখন একটি $n\times n$ matrix A নাও আর আরেকটি n সাইজের অ্যারে B নাও। যেমন $x+2y=5$ আর $3x+2y=7$ এর ক্ষেত্রে A আর B হবে এরকমঃ

$$A=\begin{bmatrix}1 & 2\\3 & 2\end{bmatrix}, B=\begin{bmatrix}5\\7\end{bmatrix}$$

এখন একটি counter নিতে হবে ধরা যাক এর নাম e. এবার আমাদের একটা কাজ n বার করতে হবে। i তম বারে ($i=0\dots(n-1)$) আমরা e তম row তে যাব। এবার আমাদের $A[j][i]$ গুলি দেখতে হবে যেখানে $e\leq j<n$ এবং এদের মাঝে সেই j আমাদের নির্বাচন করতে হবে যেন $A[j][i]$ এর মান সর্বোচ্চ হয়। যদি সব $A[j][i]$ এর মান 0 হয় তাহলে কিছু না করে আমাদের i এর লুপ continue করতে হবে। ধরলাম সর্বোচ্চটি 0 না। এখন B এবং A উভয়ের e তম row এবং j তম row কে swap করতে হবে। এর পর উভয় matrix এর eতম row কে $A[e][i]$ দিয়ে ভাগ করতে হবে। তাহলে দেখবে অন্যান্য মান পরিবর্তনের সাথে সাথে $A[e][i]$ পরিবর্তন হয়ে 1 হয়ে যাবে। এবার যা করতে হবে তাহলো $0\leq j<n$ এবং $j\neq i$ এ প্রতিটি row এর জন্য আমরা $A[e]$ row কে $A[j][i]$ দিয়ে গুণ করে $A[j]$ হতে বিয়োগ করতে হবে। শুধু A এর row না এই কাজ কিন্তু B এর সাথেও করতে হবে। আমরা কিন্তু আসলে আগে বলা প্রসেস এর মত একটি variable কে eliminate করছি। এভাবে সব row হতে i তম variable eliminate করা হয়ে গেলে আমরা e এর মান এক বাড়িয়ে দেব এবং আমাদের i এর লুপ কে continue করব। এভাবে i এর লুপ শেষ হয়ে গেলে আমাদের দেখতে হবে e এর মান কত। যদি n নাহয় তাহলে বুঝতে হবে এর unique সমাধান নেই। খেয়াল করতে হবে A এর কোন কোন row সম্পূর্ণ 0 সেসব row এর B দেখতে হবে। যদি সেসব row এর কোন একটি B 0 নাহয় তাহলে এই equation গুলির কোন সমাধান থাকা সম্ভব না। আর যদি A এর শূন্য row এর জন্য B এর সেই row ও 0 হয় তার মানে বুঝতে হবে অসংখ্য সমাধান আছে। আর যদি $e=n$ হয় এর মানে আমাদের unique সমাধান আছে আর সেই সমাধানের variable গুলির মান B তে পাব। এটিই Gaussian elemination মেথড। এর time complexity হল O(n^3).

তোমাদের প্রতি পরামর্শ থাকবে তোমরা কাগজে কলমে একটি উদাহরণ করে দেখো। তাহলে আমরা কি করছি কেন করছি তা পরিষ্কার হয়ে যাবে।

৩.৪.৫ Matrix Inverse

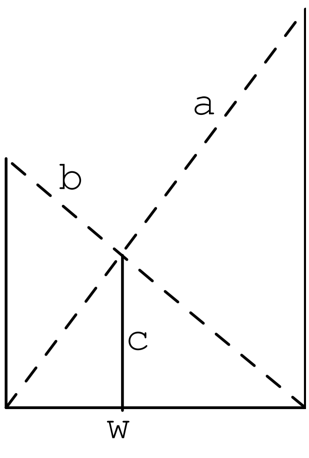
যদি Gaussian elimination বুঝে থাকো তাহলে matrix inverse করা তোমার কাছে খুবই সহজ হবে। মনে করো তোমাকে A matrix কে inverse করতে হবে। তাহলে তুমি আরেকটি matrix ধরো B নাও এবং এটি হতে হবে identity matrix অর্থাৎ এর main diagonal হবে 1 আর বাকি সব হবে 0. এখন উপরে যেমন কিছু operation (row swap, বিয়োগ, গুণ ইত্যাদি) করে করে A কে আমরা identity বানিয়েছিলাম। A এর উপর যখন operation গুলি করতে থাকবে ঠিক একই operation গুলি B এর উপর করতে থাকো। তাহলে A যখন identity matrix হয়ে যাবে তখন B হয়ে যাবে A^{-1}. যদি দেখো A কে identity বানানো সম্ভব না (0 row) তার মানে A কে inverse করাও সম্ভব না। সহজ না?

৪.২ Binary Search

ধর তুমি একটা Game Show তে আছ। তোমার সামনে মোট 100 টি বাক্স। এখন প্রতিটি বাক্সে একটি করে সংখ্যা থাকবে। তবে প্রথম বাক্সের সংখ্যা দ্বিতীয় বাক্সের সংখ্যার থেকে ছোট, দ্বিতীয় বাক্সের সংখ্যা তৃতীয় বাক্সের সংখ্যার থেকে ছোট এরকম করে আগের বাক্সের ভিতরে থাকা সংখ্যা পরের বাক্সের থেকে সবসময় ছোট হবে। এখন তোমাকে বের করতে হবে কোন বাক্সে 1986 আছে। এজন্য তুমি একটি একটি করে সব বাক্স খুলে দেখতে পার। কিন্তু এক্ষেত্রে তোমাকে অনেক বাক্স খুলতে হবে, তোমার সময়ও বেশি লাগবে। কিন্তু তুমি যদি একটু বুদ্ধি খাটাও তাহলে হয়তো সব বাক্স না খুলেও বের করতে পার যে কই 1986 আছে। তুমি ঠিক মাঝের বাক্সটা খুল। যদি দেখ এটাই 1986 তাহলে তো হয়েই গেল। আর যদি দেখ এখানে 1986 এর থেকে বড় সংখ্যা আছে তার মানে তোমার সংখ্যা বামের অর্ধেক এ আছে আর নাহলে ডানের অর্ধেকে আছে। খেয়াল কর, তুমি এক ধাক্কায় 100 বাক্স থেকে 50 বাক্স কে কিন্তু বাদ দিয়ে ফেলতে পারছ। একই ভাবে তুমি এই বাকি অর্ধেক কেও কিন্তু অর্ধেক করে ফেলতে পারবে। এরকম করতে করতে তুমি এক সময় 1986 খুব কম বাক্স খুলেই বের করে ফেলতে পারবে। তুমি কি বের করতে পারবে তোমার কত গুলি বাক্স খুলতে হবে? [১] এভাবে খুঁজার method কে আমরা binary search বলে থাকি। অনেক সময় এটি bisection method নামেও পরিচিত।

আরও একটি উদাহরন দেয়া যাক, মনে কর একটি array তে শুধু 0 ও 1 আছে। সব 0 সব 1 এর আগে থাকবে। তোমাকে প্রথম 1 খুঁজে বের করতে হবে। এটাও কিন্তু binary search দিয়ে করতে পার। তুমি মধ্য খানে গিয়ে দেখবে এটা 0 নাকি 1, যদি 0 হয় তাহলে তো এর ডানে থাকবে তোমার কাঙ্খিত জায়গা, আর যদি 1 হয় তাহলে এটা সহ বামে থাকতে পারে। এভাবে তুমি খুজতে থাকবে।

Binary Search ব্যবহার করে কিছু অদ্ভুদ সমস্যাও সমাধান করা যায়। অদ্ভুদ বললাম এই কারণে যে, প্রবলেম দেখে হয়তো কখনই মনে হবে না যে এখানে binary search ব্যবহার করা যায়, কিন্তু যায়! যেমন ৪.১ নং চিত্রে একটা w প্রস্থের রাস্তার দুদিকে দুইটি দালান আছে। এখন রাস্তার এক মাথায় একটা মই রেখে অপর মাথা রাস্তার অন্য পারের দালানের মাথায় রাখা হল, একই ভাবে রাস্তার অন্য পাশ থেকেও আরেকটি মই রাখা হল। মই দুটির দৈর্ঘ্য a ও b. মই দুটি রাস্তা থেকে c উচ্চতায় ছেদ করে। a, b এবং c এর মান দেয়া আছে, $w = ?$.



চিত্র ৪.১: w = ?

তোমরা যদি এখানে বিভিন্ন সুত্র খাটাও দেখবে খুব একটা সহজে কোন ফর্মুলা পাবে না w নির্ণয়ের জন্য। কিন্তু একটু অন্য ভাবে চিন্তা কর। তুমি যদি w এর মান জানো তাহলে কি তুমি c এর মান বের করতে পারবে? যেহেতু রাস্তার প্রস্থ দেয়া আছে আর আমরা মই এর দৈর্ঘ্য ও জানি সেহেতু আমরা প্রথমে দালান দুইটির উচ্চতা বের করি (পিথাগোরাস এর উপপাদ্য ব্যবহার করে)। ধরা যাক উচ্চতা দুইটি হল p ও q. এখন সদৃশকোনী ত্রিভুজের সুত্র খাটিয়ে আমরা দেখাতে পারি, $c = \frac{1}{\frac{1}{p}+\frac{1}{q}}$. অর্থাৎ আমরা

[১] উত্তর কিন্তু মাত্র 7টি বাক্স :)

যদি w এর মান জানি তাহলে c এর মান বের করে ফেলতে পারি। কিন্তু উল্টোটা কিন্তু কঠিন। ধরা যাক আমাদের দেয়া c এর মানের জন্য উত্তরটা হবে w'. যদি তুমি w এর মান হিসাবে w' এর থেকে বড় মান guess কর তাহলে, c এর মান প্রদত্ত মানের থেকে কম পাবে, আবার উল্টো ভাবে যদি তুমি w এর মান হিসাবে w' এর থেকে ছোট মান guess কর তাহলে c এর মান প্রদত্ত মানের থেকে বড় পাবে। কেবল w এর মান w' হলেই সঠিক দিবে। সুতরাং তোমরা w এর মানের উপর binary search চালাবে আর দেখবে c এর মান প্রদত্ত মানের থেকে বড় না ছোট সেই অনুসারে w এর মানের range ও পরিবর্তন করবে।

৪.৩ Backtracking

Backtracking কোন algorithm নয়, এটি একটি সাধারণ সল্ভিং method. আমরা যা সাধারণ ভাবে বুঝে থাকি তাই কোড করাটাই হল Backtracking. কিছু উদাহরন দিলে জিনিসটা পরিস্কার হবে।

৪.৩.১ Permutation Generate

মনে কর তোমাকে বলা হল 1 হতে n এর সকল permutation প্রিন্ট কর। যেমন $n = 3$ হলে তোমাকে $(1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2)$ এবং $(3, 2, 1)$ প্রিন্ট করতে হবে। এখন এই কাজ শুধু মাত্র for-loop দিয়ে করা কঠিন। খেয়াল কর, আমাদের n কিন্তু কত সেটা শুরুতে বলে দেয়া নাই। হয়তো বলা থাকবে যে, $n \leq 10$. যদি নির্দিষ্ট ভাবে বলা থাকত যে $n = 3$ বা $n = 4$ তাহলে হয়তো আমরা $3, 4$টি nested loop লিখে কাজটা করতাম, কিন্তু যখন n বড় হয়ে যায় তখন এত বড় জিনিস লিখা আমাদের জন্য কষ্টকর।[১] আবার লুপ এর মাধ্যমে করতে চাইলে প্রতিটি n এর জন্য আমাদের আলাদা আলাদা করে হয়তো কোড লিখতে হবে। শুধু loop দিয়ে করা একেবারে অসম্ভব না কিন্তু কষ্টকর। চিন্তা করে দেখ তোমাকে হাতে হাতে যদি এই কাজ করতে দেয়া হয় তুমি কেমনে করবে? যদি তুমি কোন systematically না করে একে একে লিখতে থাকো নিজের ইচ্ছা মত তাহলে n এর মান বড় হলে এক সময় দেখবে যে আর কি কি বাকি আছে তা বের করা বেশ কঠিন কাজ হয়ে যাবে। এখন systematic উপায়টা কি? মনে কর $n = 3$ এর জন্য তুমি যা করবে তা হল, তুমি দেখবে কোন কোন সংখ্যা এখনো বসানো হয় নাই, এদের মাঝে সবচেয়ে ছোটটা (1) নিয়ে প্রথম জায়গায় বসাবা। এখন বাকি সংখ্যাগুলি থেকে যেটি ছোট (2) সেটি দ্বিতীয় ঘরে বসাবা। একই ভাবে তৃতীয় ঘরেও বসাও (3). আমরা $(1, 2, 3)$ পেয়ে গেলাম। এখন এর immediate আগে যা বসিয়েছ তা মুছে ফেল, অর্থাৎ এর আগে যে আমরা 3 বসিয়ে ছিলাম তা সরাও। এখন দেখ next কোন সংখ্যা এখনো বসানো হয় নাই। আসলে এই ক্ষেত্রে 3 এর পর আর কোন সংখ্যা বাকি নেই, তাহলে এর আগে যেই সংখ্যা বসিয়েছিলে তা মুছো অর্থাৎ আমাদেরকে 2 মুছতে হবে। একই ভাবে আমরা দেখব 2 এর পর কোন সংখ্যাটা এখনো বসানো হয় নাই (3) তাকে বসিয়ে পরের ঘরে (তৃতীয় ঘরে) যাও। এখন দেখ কোন সবচেয়ে ছোট সংখ্যা এখনো বসানো হয় নাই (2) তাকে বসাও। আমরা শেষ প্রান্তে চলে এসেছি এবং আরও একটি permutation $(1, 3, 2)$ আমরা পেয়ে গেলাম। এবার আমরা আবার immediate আগের সংখ্যা মুছে ফেলি (2). এক্ষেত্রে আর বসানোর মত কোন সংখ্যা বাকি নেই, সুতরাং আরও একধাপ আগে চলে যাই আর 3 কে মুছে ফেলি। এখন 3 এর থেকে বড় কোন সঙ্খ্যাও বাকি নেই সুতরাং আরও এক ধাপ পিছে গিয়ে 1 কে মুছে next বড় সংখ্যা 2 বসাই। দ্বিতীয় ঘরে এসে সবচেয়ে ছোট সংখ্যা 1 বসাই ও তৃতীয় ঘরে এসে 3 বসিয়ে আমরা $(2, 1, 3)$ পাবো। এভাবে আমরা যদি করতে থাকি একে একে বাকি সব permutation ও পেয়ে যাব।

এখন এটা হাতে করা যতখানি সহজ কাজ কোডে করা ওত সহজ নাও মনে হতে পারে। সত্যি কথা বলতে হাতে করার থেকে এই জিনিসটা প্রোগ্রাম লিখা সহজ। প্রথমে খেয়াল কর আমরা প্রথমে প্রথম ঘরে সংখ্যা বসাব এর পর দ্বিতীয় ঘরে এর পর তৃতীয় ঘরে এরকম করে। সুতরাং তোমরা ভাবতে পার যে এই কাজ for loop দিয়ে করবা। কিন্তু একটু ভেবে দেখ, আমাদের systematic process এ কখনও

[১] হু, কষ্টকর কিন্তু অসম্ভব না। মনে হয় দুইটা লুপ দিয়ে যেকোনো n এর জন্য সকল permutation generate করা সম্ভব।

কখনও তুমি আবার পিছিয়ে আসো। এই পিছিয়ে আসার কাজ আর যাই হোক loop দিয়ে খুব একটা সহজে করতে পারবা না। আরেকটু ভালো করে চিন্তা করলে দেখবে যে আমরা যা করছি তাহল আমাদের কিছু সংখ্যা দেয়া আছে, আমরা সবচেয়ে ছোট সংখ্যা বসিয়ে বাকি সংখ্যা দিয়ে পরের অংশে permutation বানাচ্ছি। শেষ হয়ে গেলে next সংখ্যা বসিয়ে বাকি সংখ্যা দিয়ে আবার permutation বানাচ্ছি। অর্থাৎ জিনিসটা এমন- একটা black box আছে যাকে আমরা সংখ্যা দিলে সে permutation বানাবে। এর জন্য সে সবচেয়ে ছোট সংখ্যাকে রেখে দিবে এর পর বাকি সংখ্যা গুলিকে সে আবার একই ধরনের আরেক black box এ দিয়ে দেবে। ঐ অন্য black box এর কাজ হয়ে গেলে তুমি next বড় সংখ্যা রেখে দিয়ে বাকি সব সংখ্যা দিয়ে ঐ অন্য black box কে আবারো call করবা। আশা করি বুঝতে পারছ যে এই black box টা হল recursive function. কিন্তু এই recursive function টা Fibonacci বা Factorial এর মত সহজ নয়। যখনই একটা ফাংশন লিখবা আগে চিন্তা করে দেখ তোমার এই ফাংশনকে কি দিতে হবে, সে কি দিবে আর সে কি করবে? একে একে এই প্রশ্ন গুলির উত্তর দেয়া যাক। কি দিতে হবে? - যেসকল সংখ্যা এখনো বাকি আছে তাদের দিতে হবে, কি দিবে?- কিছুই দিবে না, কি করবে? - কিছু সংখ্যা ইতোমধ্যেই fix করা হয়েছে, বাকি সংখ্যা গুলি কে permute করে যেসব permutation হয় তাদের সবাই যেন প্রিন্ট হয় তা নিশ্চিত করতে হবে। একটু ভাবলে তোমরা বুঝবে যে যেসব সংখ্যা বসানো হয় নাই সেগুলো black box এ পাঠানোর সাথে সাথে তোমরা এখন পর্যন্ত কার পরে কাকে বসিয়েছ সেটাও পাঠাতে হবে। সুতরাং black box কে দুইটি জিনিস দিতে হবে, ১. এখন পর্যন্ত কোন সংখ্যা গুলো বসানো হয়েছে এবং কি order এ ২. কোন কোন সংখ্যা এখনো বসানো বাকি আছে। base case হল যখন সব সংখ্যা বসানো হয়ে যাবে তখন এবং সেই number sequence আমরা প্রিন্ট করব। এখন পর্যন্ত আমরা দেখেছি কেমনে একটি ফাংশনে একটি integer বা double পাঠানো যায় কিন্তু একটি array কেমনে পাঠাতে হয় তা আমাদের অজানা। standard নিয়ম হচ্ছে তুমি pointer ব্যবহার করে পাঠাবা কিন্তু তোমাদের বেশির ভাগই pointer ভয় কর। আরেকটা উপায় হচ্ছে vector ব্যবহার করা। তবে এটা বেশ slow. আরেকটা উপায় হল global array ব্যবহার করা। আমরা দুই ধরনের array রাখব। একটি array তে থাকবে কোন সংখ্যা ব্যবহার করা হয়েছে কোন সংখ্যা ব্যবহার করা হয় নাই তা (0 মানে ব্যবহার করা হয় নাই, 1 মানে ব্যবহার করা হয়েছে)। আরেকটা array তে এখন পর্যন্ত বসানো নাম্বারগুলি পর পর থাকবে। black box এর ভিতরে তুমি একে একে 1 হতে n পর্যন্ত নাম্বার গুলি চেক করবে যে আগে বসানো হয়েছে কিনা যদি না বসানো হয়ে থাকে তাহলে সেই সংখ্যা বসাবে এবং এটাও লিখে রাখবে যে এই নাম্বারটা ব্যবহার করা হয়ে গেছে। এবার পরবর্তী black box এর কাছে যাবে, সেও একই ভাবে কাজ করবে। কাজ শেষে সে যখন ফিরে আসবে, তখন দুইটা জিনিস করতে হবে তাহল বসানো সংখ্যাকে সরাতে হবে আর তুমি যে লিখে রেখেছ যে এই সংখ্যা ব্যবহার হয়েছে সেটা পরিবর্তন করে লিখতে হবে যে এটা এখনো ব্যবহার করা হয় নাই। এই কাজ যদি না কর তাহলে দেখবে মাত্র একটি permutation প্রিন্ট করেই তোমার প্রোগ্রাম শেষ হয়ে যাবে। তাহলে কি black box এ আমাদের কিছুই পাঠানোর দরকার নেই? আসলে দরকার নেই তবে আমরা আমাদের কোডকে সহজ করার জন্য একটি জিনিস পাঠাবো আর তা হল, আমরা কোন ঘরে এখন সংখ্যা বসাবো সেটা। যদিও এই জিনিস আমরা কোন কোন সংখ্যা ব্যবহার করা হয়েছে সেই array থেকে খুব সহজেই বের করতে পারি কিন্তু আমরা যদি এই সংখ্যা parameter হিসাবে পাঠাই তাহলে আমাদের কাজ অনেক সহজ হয়ে যাবে। আরও একটি জিনিস পাঠাতে হবে আর তাহল n এর মান, তবে এটি চাইলে global ও রাখতে পার। permutation প্রিন্ট করার প্রোগ্রামটা দেখানোর আগে আরেকটা জিনিস চিন্তা করে দেখতে পার- বসানো সংখ্যা কি সরানোর আদৌ দরকার আছে? শুধু কি এই সংখ্যা অব্যবহৃত আছে সেই কাজটা করাই কি যথেষ্ট নয়? তোমাদের জন্য permutation প্রিন্ট করার প্রোগ্রাম কোড ৪.৮ এ দেয়া হল।

কোড ৪.৮: permutation.cpp

```
int used[20], number[20];

//call with: permutation(1, n)
//make sure, all the entries in used[] is 0
void permutation(int at, int n)
{
```

```
    if(at == n + 1)
    {
        for(i = 1; i <= n; i++) printf("%d ", number[i]);
        printf("\n");
        return;
    }

    for(i = 1; i <= n; i++) if(!used[i])
    {
        used[i] = 1;
        number[at] = i;
        permutation(at + 1, n);
        used[i] = 0;
    }
}
```

৪.৩.২ Combination Generate

n টি সংখ্যা দেয়া থাকলে তাদের থেকে k টি করে সংখ্যা নিয়ে সকল combination প্রিন্ট করতে হবে। যেমন $n = 3$ ও $k = 2$ হলে আমাদের প্রিন্ট করতে হবেঃ $(1, 2), (1, 3)$ এবং $(2, 3)$. আমরা আগের মত যেমন তেমন ভাবে চিন্তা না করে systemetic চিন্তা করব। দুইভাবে আমরা এই প্রবলেম এর সমাধান খেয়াল করতে পারি। প্রথম উপায়টা হল, আমরা একে একে 1 থেকে n পর্যন্ত যাব আর ঠিক করব এই সংখ্যা কে নিব কি নিব না। একদম শেষে গিয়ে যদি আমরা দেখি যে আমরা k টা সংখ্যা নিয়ে ফেলেছি তাহলে তো হয়েই গেল। আর না হলে আমরা ফেরত যাবো এটা প্রিন্ট না করে। এখন খেয়াল কর, এই সমাধান সঠিক থাকলেও আমাদের সময় কিন্তু অনেক বেশি লাগবে। আমরা প্রতিটি সংখ্যার কাছে গিয়ে গিয়ে একবার নিচ্ছি আরেকবার নিচ্ছি না। সুতরাং মোট 2^n বার কাজ করে এর পর তার থেকে $\binom{n}{k}$ বার প্রিন্ট করছি। আমরা কি এই কাজ $\binom{n}{k}$ সময়ে করতে পারি না? পারি। মাত্র একটি লাইন লিখলেই আমাদের এই কাজ হয়ে যাবে। আমরা যদি কখনও দেখি যে আমাদের যেসব সংখ্যা নেবার ব্যপারে এখনো decision নেয়া বাকি আছে তাদের সবাইকে নিলেও যদি আমাদের k টা সংখ্যা না হয় তাহলে আর পরবর্তী ফাংশন call এর দরকার নেই। এখান থেকেই ফিরে গেলে হয়। আমাদের এভাবে সমাধান এর প্রোগ্রাম কোড ৪.৯ এ দেয়া হল। এই কোড এর লাইন 7 এর জন্য আমাদের প্রোগ্রাম $O(2^n)$ হতে $O(\binom{n}{k})$ হবে।

কোড ৪.৯: combination1.cpp

```
১ int number[20];
২ int n, k;
৩
৪ //call with: permutation(1, k)
৫ void combination(int at, int left)
৬ {
৭     if(left > n - at + 1) return;
৮
৯     //you can use left == 0 to make it a little bit more faster
১০     //in such case you dont need following if(left) condition
```

```
    if(at == n + 1)
    {
        for(i = 1; i <= k; i++) printf("%d ", number[i]);
        printf("\n");
        return;
    }

    if(left)
    {
        number[k - left + 1] = at;
        combination(at + 1, left - 1);
    }

    combination(at + 1, left);
}
```

দ্বিতীয় পদ্ধতির ক্ষেত্রে আমরা প্রত্যেক ঘরে যাবো, এরপর এখানে আগে বসানো হয় নাই এমন একটি সংখ্যা বসাবো এবং এভাবে একে একে k ঘরে যখন আমরা সংখ্যা বসিয়ে ফেলব তখন আমরা একটা number combination পেয়ে যাবো। তবে এক্ষেত্রে আমাদের $(1, 2)$ এর সাথে সাথে $(2, 1)$ ও প্রিন্ট হয়ে যাবে। তোমরা যদি n টা সংখ্যা থেকে k টা করে সংখ্যা নিয়ে তাদের permutation প্রিন্ট করতে চাও তাহলে এভাবে করলেই হবে। কিন্তু যদি combination প্রিন্ট করতে চাও তাহলে আরও একটা কাজ করতে হবে আর তা হল, তুমি নতুন যেই সংখ্যা বসাবে সেটা যেন আগের সংখ্যা থেকে বড় হয়। এই কাজ করার জন্য সবচেয়ে ভালো উপায় হল তুমি parameter হিসাবে সর্বশেষ বসানো সংখ্যাটা পাঠিয়ে দাও এরপর যখন তুমি loop চালাবে তখন 1 থেকে না চালিয়ে এই সংখ্যা থেকে চালালেই হবে। এই সমাধানটা আমাদের প্রথম সমাধান এর থেকে একটু হলেও better বলা যায়। কারণ, আমাদের আগের সমাধান worst case এ recursion এ n depth পর্যন্ত যায়, কিন্তু আমাদের দ্বিতীয় সমাধান k depth পর্যন্ত যাবে। আমাদের দ্বিতীয় সমাধানের প্রোগ্রামের কোড ৪.১০ এ দেয়া হল। আশা করি 14 নাম্বার লাইন বাদে বাকি কোডটুকু বুঝতে তোমাদের সমস্যা হবে না। আমরা যেভাবে এর আগের বার একটা if দিয়ে $O(2^n)$ হতে $O(\binom{n}{k})$ আনা হয়েছে এখানেও সেরকম কাজ করা হয়েছে। তবে পার্থক্য হল আমরা এর আগে আলাদা ভাবে condition চেক করে ছিলাম, এবার for loop এর upper bound হিসাবে এই কাজটা করেছি। এই দুই উপায়ের মাঝে তেমন কোন পার্থক্য নেই, আমরা দুই কোডে দুইভাবে করে দেখালাম তোমরা যাতে দুই উপায়ের সাথে পরিচিত থাক সেজন্য। তোমরা ভেবে দেখতে পার $i \leq n - k + at$ এই condition টা কই থেকে এল? আমরা যদি i কে at এ বসাই তাহলে আমাদের সংখ্যা বাকি থাকে $n - i$ টি আর আমাদের ঘর বাকি থাকে $k - at$ টি। ঘরের থেকে সংখ্যা কম হওয়া যাবে না, তাই $k - at \leq n - i \rightarrow i \leq n - k + at$.

কোড ৪.১০: combination2.cpp

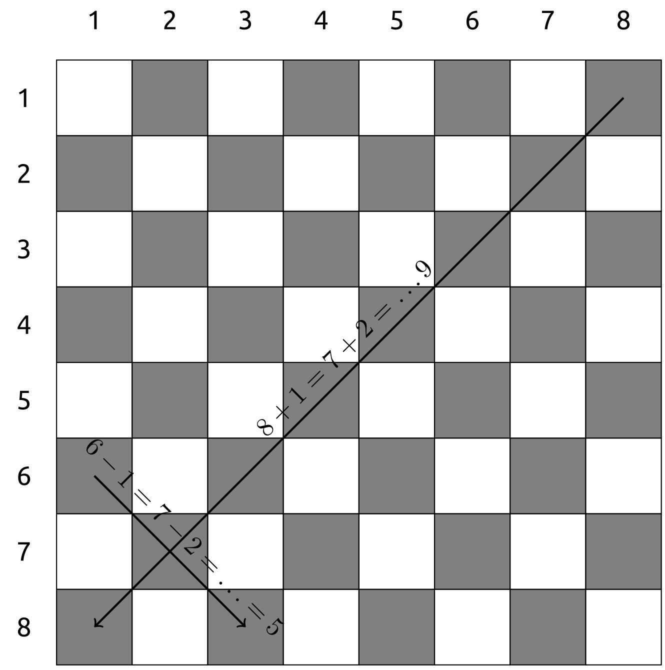
```
১ int number[20];
২ int n, k;
৩
৪ //call with: permutation(1, 0)
৫ void combination(int at, int last)
৬ {
৭     if(at == k + 1)
৮     {
৯         for(i = 1; i <= k; i++) printf("%d ", number[i]);
```

```
        printf("\n");
        return;
    }

    for(i = last + 1; i <= n - k + at; i++)
    {
        number[at] = i;
        combination(at + 1, i);
    }
}
```

৪.৩.৩ Eight Queen

এটি একটি বিখ্যাত সমস্যা। তোমরা যারা দাবা খেলা জানো, আশা করি তাদের বুঝতে কোন সমস্যা হবে না। সমস্যাটা হল একটি দাবার বোর্ডে (দাবা বোর্ড 8×8 হয়ে থাকে) 8 টা queen (আমরা অনেক সময় বাংলায় এদের মন্ত্রী বলে থাকি) কে কত ভাবে বসানো যায় যেন কোন queen ই অন্য কোন queen কে attack না করে। একটি queen অপর আরেকটি queen কে attack করতে পারবে যদি তারা একই row বা একই column বা একই diagonal বরাবর থাকে। সমস্যাটা খুব একটা কঠিন না। আমাদের যা করতে হবে তা হল প্রত্যেক row তে গিয়ে গিয়ে একটা করে queen বসাতে হবে, সব row তে queen বসানো শেষ হয়ে গেলে আমরা দেখব কোন একটি queen অপর আরেকটি queen কে attack করে কিনা। যেহেতু আমরা প্রতি row তে একটি করে queen বসাচ্ছি সেহেতু কোন দুইটি queen একই row তে আছে কিনা তা দেখার দরকার নেই। শুধু একই column এ আছে কিনা তা দেখতে হবে আর একই diagonal এ আছে কিনা তা। একই column এ আছে কিনা এটা চেক করা খুব সহজ, কিন্তু একই diagonal এ আছে কিনা সেটা দেখা বেশ tricky. দুই ধরনের diagonal হতে পারে। এক ধরনের diagonal উপরের বাম দিক থেকে শুরু করে নিচের ডান দিকে যায় অন্য diagonal গুলি উপরের ডান দিক থেকে নিচের বাম দিকে যায়। মনে করি আমাদের দাবা বোর্ড এর row গুলি উপর থেকে নিচে 1 হতে 8 পর্যন্ত নাম্বার করা এবং column গুলি বাম থেকে ডান দিকে 1 হতে 8 পর্যন্ত নাম্বার করা (চিত্র ৪.২)। এখন একটু খেয়াল করলে দেখবে যেসকল diagonal উপরের বাম দিক থেকে নিচের ডান দিকে যায় তাদের row ও column এর বিয়োগফল একই হয় এবং যেসকল diagonal উপরের ডান দিক থেকে নিচের বাম দিকে যায় তাদের row ও column এর যোগফল একই হয়।

তাহলে আমরা সব row তে queen বসানোর পর দুইটি দুইটি করে queen নিয়ে দেখব যে তাদের column বা diagonal একই কিনা। এরকম করে করলে একটা সমস্যা হল, এমনও হতে পারে যে আমরা প্রথম দুইটি queen কে একই column এ বসিয়ে ফেলেছি এর পর বাকি 6 টি queen কে আমরা কিন্তু অনেক ভাবে বসাতে পারি, যেভাবেই বসাই না কেন আমরা কোন valid placement পাবো না। সুতরাং আমরা প্রতিবার queen বসানোর আগে বা পরে চেক করে দেখতে পারি যে এখন পর্যন্ত বসানো queen গুলো কেউ কারোS সাথে attacking position এ আছে কিনা। একটু চিন্তা করলে দেখবে, আসলে সব queen pair চেক করার দরকার নাই। শুধু মাত্র নতুন বসানো queen এর সাথে আগের বসানো queen গুলোকে চেক করলেই হয়। আরও একটু চিন্তা করলে দেখবে এখানে আমাদের ধরে ধরে আগে বসানো প্রতিটি queen এর সাথে চেক করার দরকার হবে না, যদি আমরা এমন কিছু array রাখি যারা বলে দিবে যে অমুক column বা অমুক diagonal এ কোন queen আছে কিনা। অর্থাৎ আমরা কোন একটি queen বসানোর সময় array গুলিতে লিখে দিব যে অমুক column, অমুক diagonal এ queen বসেছে। তাহলে নতুন queen বসানোর আগে শুধু আমরা চেক করে দেখব যে যেই column বা diagonal এ আমরা queen বসাতে চাচ্ছি তা আদৌ ফাঁকা আছে কিনা। অর্থাৎ আমাদের কোন loop লাগবে না, শুধু if-else দিয়েই চেক হয়ে যাবে যে আমরা যেই column বা diagonal এ queen বসাতে চাচ্ছি তা ফাঁকা আছে কিনা। খেয়াল কর আমরা কিন্তু ধীরে ধীরে আমাদের সমাধানকে যতটুকু সম্ভব optimized করছি। আমরা আমাদের প্রাথমিক সমাধান এর থেকে অনেক দূরে চলে এসেছি ঠিকি কিন্তু আমরা এমন সব কিছু করেছি যাতে করে আমাদের প্রোগ্রাম



চিত্র ৪.২: দাবা বোর্ড

আগের তুলনায় অনেক অনেক কম সময় নেয়। তোমরা যখন এই জিনিস কোড করে দেখবে তখন প্রতিটি improvement যোগ করার আগে ও পরে দেখবে তোমাদের কোড কত সময় নেয়। এতে করে তোমরা বুঝবে এসব optimization দেখতে অনেক সহজ হলেও এরা performance এর দিক থেকে অনেক অনেক এগিয়ে দেয় তোমাকে। হয়তো 8×8 বোর্ড এর জন্য এসব optimization এর প্রভাব তুমি নাও বুঝতে পার। তোমরা চাইলে $9 \times 9, 10 \times 10$ এসব বোর্ড এও চেক করে দেখতে পার। আমরা এখানে আলোচনা করা সকল optimization ব্যবহার করে লিখা প্রোগ্রাম কোড ৪.১১ এ দেখালাম। যদি তোমরা $n = 8$ দাও তাহলে 8×8 বোর্ড হবে, বা তোমরা চাইলে অন্যান্য মাপের বোর্ড এর জন্যও এই প্রোগ্রাম রান করে দেখতে পার।

কোড ৪.১১: nqueen.cpp

```
int queen[20]; //queen[i] = column number of queen at ↩
    ith row
int column[20], diagonal1[40], diagonal2[40]; //arrays ↩
    to mark if there is queen or not

//call with nqueen(1, 8) for 8 queen problem
//make sure column, diagonal1, diagonal2 are all 0 ↩
    initially
void nqueen(int at, int n)
{
    if(at == n + 1)
    {
        printf("(row, column) = ");
        for(i = 1; i <= n; i++) printf("(%d, %d) ", i, ↩
```

```c
            queen[i]);
        printf("\n");
        return;
    }

    for(i = 1; i <= n; i++)
    {
        if(column[i] || diagonal1[i + at] || diagonal2[n + i - at]) continue;
        queen[at] = i;
        //note that, i - at can be negative and we cant have array index negative
        //so we are adding offset n with this.
        column[i] = diagonal1[i + at] = diagonal2[n + i - at] = 1;
        nqueen(at + 1, n);
        column[i] = diagonal1[i + at] = diagonal2[n + i - at] = 0;
    }
}
```

তোমরা যদি এতটুকুতেই হাঁফ ছেড়ে মনে কর যাক optimization শেষ হল, তাহলে বলে রাখি আরও একটি খুব সহজ optimization আছে যার ফলে তোমরা তোমাদের run time কে একদম অর্ধেক করতে পারবে। চিন্তা করে দেখ সেই optimization টা কি! আসলে optimization এর শেষ নেই। Backtracking এর ক্ষেত্রে যে যত optimization যোগ করতে পারবে তার কোড তত ভালো কাজ করবে। তবে খেয়াল রাখতে হবে সেই সব optimization এর জন্য আবার না অনেক বেশি সময় লেগে যায়! যেমন আমরা যদি বসানোর সময় শুধু array তে না দেখে আগের সব queen এর সাথে যদি চেক করতে যাই তাহলে দেখা যাবে অনেক বেশি সময় লেগে যাবে। সুতরাং আমাদের এই জিনিসও খেয়াল রাখতে হবে।

৪.৩.৪ Knapsack

মনে কর এক চোর চুরি করতে গিয়েছে। তার কাছে একটা থলে আছে যাতে খুব জোর W ওজনের জিনিস নেয়া যাবে। এখন সেই চোর চুরি করতে গিয়ে n টা জিনিস দেখতে পেল। প্রতিটি জিনিসের ওজন w_i এবং ঐ জিনিস বিক্রি করলে সে v_i টাকা পাবে। এখন সবচেয়ে বেশি কত টাকার জিনিস তুমি চুরি করতে পারবে যেন সেসব জিনিসের মোট ওজন W এর থেকে বেশি না হয়? এক্ষেত্রে limit গুলি খুব গুরুত্বপূর্ণ। $n \le 50, w_i \le 10^{12}$ এবং $v_i \le 10^{12}$. তাহলে আমরা কি করব? আগের মত একে একে 1 থেকে n পর্যন্ত যাবো, কোন জিনিস নিব, কোন জিনিস নিবো না শেষে গিয়ে দেখব যে ওজন W এর থেকে বেশি হয়েছে কিনা। বেশি হলে এটা সমাধান হবে না, আর তা না হলে আমরা এরকম সকল সমাধান এর মাঝে যেক্ষেত্রে দাম সবচেয়ে বেশি হয় সেটাই সমাধান হবে। বুঝতেই পারছ এত সহজ সমাধান হলে এখানে আর আলোচনার কিছু থাকত না! তাহলে এই সমাধানের ঝামেলা কোথায়? প্রথমে হিসাব করে দেখ এই সমাধানের run time কত? $O(2^n)$. অবশ্যই $n \le 50$ এর জন্য এটা একটা বিশাল মান। তাহলে আমরা কেমনে সমাধান করতে পারি? আমাদেরকে আগের Eight queen problem এর মত কিছু optimization বের করতে হবে। প্রথমত আগের মত আমরা প্রতিটি জিনিস নেবার পরেই দেখব এর ওজন W এর থেকে বেশি হয়ে গেছে কিনা, তা হয়ে গেলে পরের জিনিস গুলো দেখার কোন মানে নেই, এখান থেকেই ফিরত যাওয়া উচিত। আরও কি কি optimization থাকতে পারে? খেয়াল কর যেগুলো বাকি আছে তাদের সবার ওজন মিলেও যদি আমাদের সর্বমোট ওজন W এর থেকে বেশি না হয় তাহলে বাকি সবগুলো নিয়ে ফেলাই বুদ্ধিমানের মত কাজ। আবার দেখো, যেসব ওজন বাকি আছে তাদের মাঝে সবচেয়ে যেটি ছোট তাকে নিলেই যদি আমাদের মোট ওজন W এর থেকে বেশি

হয়ে যায় তাহলে আমাদের আর এগিয়ে লাভ নেই। ওজন নিয়ে বেশ অনেক optimization ই হয়ে গেছে। দাম দিয়ে কি কিছু optimization করা যায়? যায়, ধর এখন পর্যন্ত আমরা যেসব সমাধান বের করেছি তার মাঝে সবচেয়ে ভালো যেই সমাধান তা থেকে আমরা V টাকা পাই। এখন আমরা সমাধান করার মাঝামাঝি পর্যায়ে যদি দেখি আমরা ইতোমধ্যে যত টাকা পেয়ে গেছি আর এখনও যত জিনিস বাকি আছে তাদের দাম সহ যদি V এর থেকে বেশি না হয় তার মানে এখান থেকে আরও এগিয়ে কোন লাভ নেই। এরকম নানা optimization প্রয়োগ করলে আমাদের এই প্রোগ্রাম এর run time অনেক কমে যায়।

```
Q.size();                          //size of the queue
Q.empty();                         //returns true if empty
```

৫.৪ Graph এর representation

Graph হল সহজ অর্থে সম্পর্ক। অনেক ধরনের entity এর মাঝে সম্পর্ক বুঝাতে আমরা graph ব্যবহার করে থাকি। যেমন কিছু আগেই আমরা দেখে এসেছি যে facebook এ কতগুলি মানুষের মাঝে বন্ধুত্ব এর সম্পর্ক বুঝানর জন্য আমরা graph ব্যবহার করতে পারি। এই গ্রাফ কিন্তু আমাদের লেখ চিত্রের গ্রাফ না। তবে এখানেও আঁকার জিনিস আছে তবে ছক কাগজের দরকার নেই! তুমি যেসব মানুষের মাঝে সম্পর্ক নির্ণয় করবা তারা এক এক জন একটি করে node বা vertex। আমরা node বা vertex বুঝাতে একটি বিন্দু একে থাকি। এখন দুজন মানুষের মাঝে বন্ধুত্ব আছে এটা নির্দেশ করার জন্য তাদের মাঝে লাইন টেনে থাকি একে edge বলা হয়। তোমরা লক্ষ্য করলে দেখবে একটা map এ বিভিন্ন শহরের মাঝে রাস্তা রেললাইন এসব জিনিস দাগ কেটে দেখানো থাকে। এসব ক্ষেত্রে আমরা শহর গুলিকে vertex ও রাস্তা গুলিকে edge হিসাবে কল্পনা করতে পারি আর তাহলে আমাদের এই বিশাল ম্যাপ একটা গ্রাফ হয়ে যায় যা বিভিন্ন শহরের মাঝে রাস্তার সম্পর্ক দেখায়।

এখন গ্রাফ এর সমস্যা সমাধানের সময় আমাদের এই গ্রাফ কে মেমরি তে রাখতে হবে। আমরা তো আর কম্পিউটারে ছবি একে রাখতে পারি না, আমাদের কে এই vertex গুলির একটি করে number দিতে হয় আর কোন নাম্বার এর সাথে কোন নাম্বার vertex এর সম্পর্ক আছে তা adjacency list বা adjacency matrix এর সাহায্যে রাখতে হয়। আমরা কিন্তু ইতোমধ্যেই এই দুইটির নাম আর কেমনে করতে হয় তা জেনে ফেলেছি!

আমাদের এই facebook এর উদাহরনে friendship কিন্তু mutual বা bidirectional অর্থাৎ A যদি B এর বন্ধু হয় তাহলে B ও A এর বন্ধু হবে। কিন্তু অনেক সময় এই সম্পর্ক bidirectional না হয়ে directional হয়ে থাকে। যেমন facebook এর follower. A যদি B কে follow করে এর মানে এই না যে B ও A কে follow করছে। অর্থাৎ এখানে সম্পর্ক এক তরফা। আমরা আগের গ্রাফ কে বলে থাকি undirected graph আর follower এর গ্রাফ হল directed graph. প্রথম ক্ষেত্রে A ও B এর মাঝে যদি undirected edge থাকে তাহলে A এর লিস্টে B কে এবং B এর লিস্টে A কে রাখতে হয়। আর যদি directed edge হয় তাহলে শুধু A এর লিস্টে B কে রাখলেই হবে যদি A এর থেকে B এর দিকে edge হয়।

৫.৫ Tree

Tree একটি special ধরনের গ্রাফ যেখানে n টি vertex এর জন্য $n-1$ টি edge থাকে এবং পুরো গ্রাফ connected থাকে। Connected থাকার অর্থ হল ঐ গ্রাফের এক node থেকে অপর node এ তুমি এক বা একাধিক edge ব্যবহার করে যেতে পারবে। যদি আমরা node কে শহর আর edge কে রাস্তা মনে করি তাহলে বলা যায়, প্রতিটি শহর থেকেই অন্য সকল শহরে যাওয়া যায়। এই গ্রাফের কিছু properties আছে। যেমন এই গ্রাফে কোন cycle নাই, অর্থাৎ তুমি যদি কোন node থেকে শুরু কর তাহলে কোন edge দুইবার ব্যবহার না করে কোন মতেই ঐ node এ ফিরতে পারবে না। তুমি কোন একটি node থেকে অপর node এ সবসময় uniquely যেতে পারবে মানে তোমার যাবার রাস্তা একটাই থাকবে (অবশ্যই তুমি এক রাস্তা একাধিক বার ব্যবহার করবা না)। অনেক সময় tree তে একটি special node দেয়া থাকে যাকে বলা হয় root. এক্ষেত্রে tree এর edge গুলিকে directed ভাবে কল্পনা করা হয়। edge এর direction হবে root থেকে কোন node এ যেতে হলে ঐ edge দিয়ে তুমি যেদিকে যাবা সেদিক। তোমরা চাইলে root কে ধরে ঐ tree কে ঝুলিয়ে দিতে পার। তাহলে উপর থেকে নিচের দিকে হবে ঐ edge গুলি। কোন edge এর উপরের node কে parent বলা হয়, আর নিচের node কে ঐ parent এর child বলা হয়। কোন node এর parent এর অন্যান্য child কে এই node এর sibling বলে। যদি কোন node থেকে তুমি উপরে root এর দিকে যেতে থাকো তাহলে পথে যেসব node পাবা তাদের ancestor বলে। কোন node থেকে

উপরে না উঠে শুধু নিচে নামতে থাকলে যেসব node পাওয়া যায় তাদের descendant বলে। tree এর ক্ষেত্রে level নামে একটা টার্ম আছে, এই টার্ম থেকে বুঝা যায় কোন একটি node আমাদের root থেকে কত গভীরে আছে। আমরা বলে থাকি root আছে level 0 তে, এর এক ধাপ নিচের গুলি level 1 এ। আমরা অনেক সময় level এর পরিবর্তে depth ও বলে থাকি। একটি tree এর সবচেয়ে গভীরের node যদি $h-1$ depth এ থাকে তাহলে আমরা সেই tree এর height h বলে থাকি।

যেহেতু tree এক ধরনের গ্রাফ সেহেতু তুমি একটি গ্রাফ কে adjacency matrix বা list এর মাধ্যমে যেভাবে represent করেছো সেভাবে করা যায়। কিন্তু tree এর আলাদা বৈশিষ্ট্য এর জন্য একে অন্য আরও ভাবেও প্রকাশ করা যায়।

Child List এটা কিছুটা directed graph এ adjacency list এর মত। আমরা প্রতিটি node এর child লিস্ট রাখব। আমরা চাইলে যেকোনো node থেকে শুরু করে শুধু নিচের দিকে যেতে পারি। এই representation এর ক্ষেত্রে বেশির ভাগ সময় আমরা root থেকে শুরু করে নিচের দিকে যেতে থাকি। একে আমরা top down representation বলতে পারি।

Parent link এক্ষেত্রে আমরা প্রতিটি node এর parent রেখে থাকি। এই representation এর ক্ষেত্রে আমরা উপর থেকে নিচে যেতে পারি না। কিন্তু কোন একটা node থেকে উপরে উঠতে পারি। একে আমরা bottom up representation বলতে পারি।

অনেক সময় আমাদের Child list ও Parent link দুইটিই একই সাথে দরকার হয়ে থাকে। যদি প্রতিটি node এর খুব জোর দুইটি child থাকে তাহলে সেই tree কে binary tree বলা হয়। আমরা ছবি আঁকার সময় যে child কে বাম দিকে রাখি তাকে left child ও অপরটিকে right child বলি। এছাড়াও tree সম্পর্কিত আরও অনেক term আছে আমরা ধীরে ধীরে সেসব জানব।

৫.৬ Binary Search Tree (BST)

এই binary tree এর প্রতিটি node এ একটি করে মান থাকে। এখন মান গুলি এমন ভাবে থাকে যেন এর left subtree এর সকল মান [১] এই node এ থাকা মান থেকে ছোট হয় আর right subtree এর সকল মান এর থেকে বড় হয়। আমরা link list এর মত করে এই জিনিস বানাতে পারি। সেক্ষেত্রে প্রতিটি node এ আমাদের দুইটি link এর দরকার হবে, একটি left child এর জন্য অপরটি right child এর জন্য। অনেক সময় parent এর জন্যও আলাদা link রাখা হয়। একটি Binary Search Tree তে কোন একটি সংখ্যা আছে কিনা তা খুঁজে বের করা বেশ সহজ। তুমি কোন একটি node এ গিয়ে দেখবা যে তুমি যেই সংখ্যা খুঁজছ সেটা এখানে থাকা সংখ্যার থেকে ছোট না বড়। যদি সমান হয় তাহলে তো পেয়েই গেলা, আর যদি ছোট হয় তাহলে বাম দিকে যাবা আর বড় হলে ডান দিকে যাবা। এরকম করে তুমি insert ও করতে পারবা। delete করা একটু কঠিন। অনেক সময় প্রোগ্রামিং কন্টেস্টে আমরা সত্যিকার ভাবে delete না করে প্রতিটি node এ একটি করে flag রাখি। delete করলে সেই flag কে আমরা off করে দিলেই হয়।

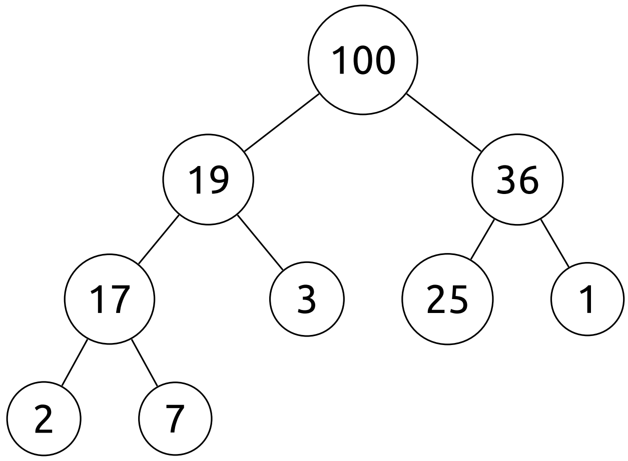
এখন কথা হল, একটা array তে সংখ্যা না রেখে আমরা এরকম tree আকারে সংখ্যা রাখলে লাভ কি? খেয়াল কর, একটি সংখ্যা খুঁজার সময় আমরা যখন একটি node এ থাকা সংখ্যার সাথে আমার সংখ্যাকে তুলনা করি তখন সেই তুলনার ভিত্তিতে আমরা একদিক বাদ দিয়ে আরেক দিকে যেতে পারি। এখন এই ভাগা ভাগি যদি ঠিক অর্ধেক হয় তাহলে আমরা প্রতিবার অর্ধেক সংখ্যা বাদ দিতে পারি। ঠিক আমাদের শিখে আশা binary search এর মত। তাহলে আমরা যদি ঠিক অর্ধেক অর্ধেক করে রাখতে পারি তাহলে আমরা $O(\log n)$ এই সার্চ করতে পারব। তাহলে আমাদের binary search এর সাথে এর পার্থক্য কই? খেয়াল কর, binary search এ আমরা কিন্তু কোন একটি সংখ্যা কে insert বা delete করতে পারি না। কিন্তু আমরা আমাদের এই BST তে কোন সংখ্যা insert বা delete করতে পারি। একটু ভাবলে বুঝবা যে সাধারণ ভাবে সংখ্যা গুলিকে প্রবেশ করালে কিন্তু আমাদের BST অনেক লম্বা হয়ে যেতে পারে, যেমন 1 এর ডানে 2, 2 এর ডানে 3 এরকম করে n পর্যন্ত যদি সংখ্যা থাকে তাহলে কিন্তু $O(n)$ সময় লেগে যাবে। যাতে এরকম সমস্যা না হয় সেজন্য আমাদের BST কে balance করে

[১] subtree হল মূল tree এর একটি অংশ যা নিজেও tree.

নিতে হয় যেন tree এর height বেশি বড় না হয়। এরকম ধরনের কিছু ডাটা স্ট্রাকচার আছে যেমন AVL Tree, Red Black Tree, Treap ইত্যাদি। তোমরা চাইলে এসব জিনিস internet এ দেখতে পার। তবে বেশির ভাগ সময় আমরা STL এর Map বা Set ব্যবহার করে BST সংক্রান্ত অনেক কাজ করে ফেলতে পারি।

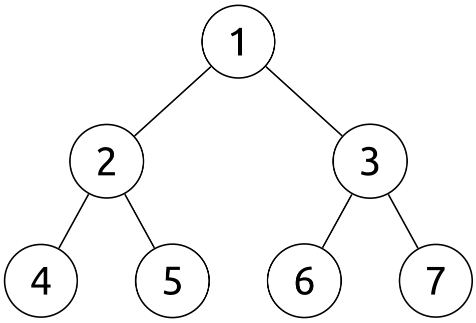
৫.৭ Heap বা Priority Queue

এটিও এক ধরনের binary tree. আরও শুদ্ধ ভাবে বলতে গেলে complete binary tree. এই tree এর শেষ level বাদে প্রতিটি level এ সর্বোচ্চ সংখ্যক node থাকবে। শুধু শেষ লেভেলটি পূর্ণ নাও হতে পারে, তবে সেক্ষেত্রেও বাম থেকে ডান দিকে node গুলি সাজানো থাকে। চিত্র ৫.২ এ তোমাদের জন্য একটি heap দেয়া আছে।



চিত্র ৫.২: Heap

Heap দুই রকম হতে পারে, Max Heap, Min Heap. Max Heap এর বৈশিষ্ট্য হচ্ছে কোন node এ থাকা মান তার যেকোনো descendant এর থেকে বড় হবে। অর্থাৎ root এ এই heap এর সবচেয়ে বড় মান থাকবে, তার left child এ থাকবে left subtree এর মাঝের সবচেয়ে বড় মান। এরকম করে পুরো heap বানানো হয়। আশা করি বুঝতেই পারছ Min Heap কি রকম হয়। যদি কোন heap এ n টি node থাকে তাহলে সেই heap এর height $\log(n)$ হয়। আমরা এই ডাটা স্ট্রাকচারটি তখন ব্যবহার করে থাকি যখন আমাদের অনেকগুলি মান একে একে আসতে থাকে এবং আমাদের মাঝে মাঝে সবচেয়ে বড় মানটি দরকার হয় এবং এই বড় মানটি সরিয়ে ফেলতে হয়। ফলে পরে যখন আবারো সবচেয়ে বড় মান এর দরকার হয় তখন এর পরবর্তী বড় মানটি দিতে হয়। যেমন চিত্র ৫.২ এ আমাদের সবচেয়ে বড় মান চাইলে 100 দিতে হবে, এর পরে আবারো বড় মান চাইলে 36 দিতে হবে এরকম। একটু মনে মনে ভাব তো তোমাদের যদি একটি heap বানাতে বলি কেমনে কোড করবে? যদি ভেবে থাকো link list এর মত করে link রেখে রেখে- তাহলে তোমরা ঠিক ভেবেছ। কিন্তু এর থেকেও সহজ উপায় আছে (এবং এর drawback ও আছে!)। তোমরা যদি আগের মত left child link ও right child link রেখে রেখে কর এবং dynamically memory assign কর তাহলে আগে থেকে আমাদের বড় array declare করার দরকার হয় না। কিন্তু যদি আমরা বড় array declare করে করতে চাই তাহলে একটা সহজ উপায় আছে। চিত্র ৫.৩ এ আমরা heap এর জন্য array কেমনে indexing করলে সহজ হয় তা দেখালাম। খেয়াল করলে দেখবে, কোন একটি node এর index যদি i হয় তাহলে এর left child এর index হবে $2i$ এবং এর right child এর index হবে $2i + 1$. তাহলে দেখ সুন্দর করে পর পর level by level আমাদের index হয়ে যাবে। যদি কোন node i এ থেকে তার parent এ যেতে চাও তা অনেক সহজে $i/2$ করে যেতে পারবে, মনে রেখ এখানে integer division হচ্ছে।



চিত্র ৫.৩: Heap array numbering

Heap এ insert করা খুব সহজ। যদি heap এ ইতোমধ্যে n টা সংখ্যা থাকে তাহলে নতুন সংখ্যা তোমরা $n+1$ এ বসাও। এর পর তুমি parent দিয়ে root পর্যন্ত যেতে থাকবে, যদি দেখ parent তোমার থেকে ছোট তাহলে swap করবে, এরকম যতক্ষণ না তোমার parent তোমার থেকে বড় না হয় ততক্ষণ এই কাজ করলেই হবে। যেহেতু আমাদের height $\log(n)$ সুতরাং আমাদের insertion এ সময় লাগবে $O(\log n)$. যদি Max Heap হতে এর root অর্থাৎ সবচেয়ে বড় সংখ্যা remove করতে চাও তাহলে যা করতে হবে তা হল এর শেষ সংখ্যা কে এনে root এ বসাতে হবে। এর পর দেখতে হবে তোমার left child বড় না right child. ওদের যেটি বড় তা যদি আবার তোমার থেকেও বড় হয় তাহলে তার সাথে swap কর এবং এভাবে নিচে নামতে থাকো। এভাবে $O(\log n)$ এ আমরা সবচেয়ে বড় সংখ্যা কে remove করতে পারি। একটু চিন্তা করলে তোমরা কোন একটি node কে modify বা remove ও করতে পারবে। কিন্তু একটা জিনিস খেয়াল রাখবে, এখানে কিন্তু কোন একটি সংখ্যা খুব দ্রুত খুঁজে পাওয়া সম্ভব না। তোমাকে কোন সংখ্যা খুজতে হলে সবগুলো node এ তোমাকে চেক করতে হতে পারে worst case এ।

Heap কে আমরা কখনও কখনও priority queue বলে থাকি। আমরা queue এর উদাহরন দিতে বাস এর লাইন এর কথা বলেছিলাম, এখন মনে কর, বাসের লাইন ওরকম আগে আসলে আগে যাবেন এরকম না হয়ে কে কত গন্যমান্য ব্যক্তি তার উপর ভিত্তি করে হবে। অর্থাৎ এটা হল priority queue. যত জন মানুষ আছে তাদের মাঝে সেই যাবে যার priority সবচেয়ে বেশি। এই জিনিসই কিন্তু আমাদের heap. STL এ priority queue বানিয়ে দেয়া আছে। কোড ৫.৪ এ তোমাদের এই STL এর ব্যবহার দেখানো হল। তোমরা চাইলে শুধু int না, যেকোন structure এরও priority queue বানাতে পার তবে সেক্ষেত্রে তোমাদের operator overload করতে হবে।

কোড ৫.৪: priority queue.cpp

```
#include<priority_queue>
using namespace std;

priority_queue<int> PQ; //declare a max heap
PQ.push(4);             //insert
PQ.top();               //maximum element
PQ.pop();               //pop max
PQ.size();              //returns size of heap
PQ.empty();             //returns 1 if heap is empty
```

৫.৮ Disjoint set Union

মনে কর তোমাকে কিছু কোম্পানির নাম দেয়া আছে। প্রথমে সব কোম্পানির মালিক আলাদা আলাদা। এর পর একে একে বলা হবে যে অমুক কোম্পানির মালিক অমুক কোম্পানি কিনে নিয়েছে। মাঝে মাঝে প্রশ্ন করা হবে যে, এই কোম্পানির মালিক কে? বা এই কোম্পানির মালিক আসলে কত গুলি কোম্পানির মালিক? বা সে যত কোম্পানি ক্রয় করেছে তাদের মাঝে সবচেয়ে বেশি লোকজন কাজ করে কোন কোম্পানিতে এরকম নানা প্রশ্ন করা হতে পারে। এই সব ক্ষেত্রে Disjoin Set Union ডাটা স্ট্রাকচার ব্যবহার করে সমাধান করা সম্ভব। একে অনেকে Union Find ও বলে থাকে।

এই ডাটা স্ট্রাকচার এর জন্য আমাদের একটি মাত্র array দরকার, ধরা যাক তা হল p. $p[i]$ এর মানে হল i কোম্পানির মালিক হল $p[i]$ কোম্পানির মালিক। প্রথমে সকল i এর জন্য $p[i] = i$. এর পরে কখনও যদি তোমাকে বলে a কোম্পানির মালিক কে? তখন তুমি এর $p[a]$ দেখবে যদি এটি a এর সমান হয় তাহলে তো হয়েই গেল আর না হলে তার $p[]$ দেখবে, এরকম করে চলতে থাকবে। এখন কথা হল এতে তো অনেক সময় লাগার কথা, যদি আমাদের $p[]$ এর array টা এমন থাকে যে, $p[1] = 2, p[2] = 3, \ldots p[n-1] = n$ তাহলে যদি $a = 1$ হয় তাহলে প্রতিবার $O(n)$ সময় লাগবে। এখন খেয়াল কর তুমি যদি একবার 1 এর জন্য বুঝে যাও যে n হল আসল মালিক তাহলে কি তুমি $p[1] = n$ লিখতে পার না? একই ভাবে, তুমি 1 এর মালিক খুঁজার সময় $2, 3, \ldots n-1$ এর ভিতর দিয়ে গিয়েছ এবং সব শেষে তুমি জেনেছ যে তোমাদের সবার মালিক হল n সুতরাং এখন তুমি চাইলে সবার মালিক পরিবর্তন করে n করে দিতে পার। এতে করে কোন ক্ষতি নাই, বরং তুমি এতক্ষণ যে অনেক বড় chain পার করে যে আসল উত্তর বের করছ এখন আর ওত বড় chain ডিঙ্গাতে হবে না। একে আমরা Find বলে থাকি। Find এর কোড ৫.৫ এ দেখতে পার।

কোড ৫.৫: union find.cpp

```
int p[100]; //initially p[i] = i;

int Find(int x)
{
    if(p[x] == x) return x;
    return p[x] = Find(p[x]);
}

void Union(int a, int b)
{
    p[Find(b)] = Find(a);
}
```

এখন আশা যাক, a এর মালিক যদি b কোম্পানিকে কিনে নেয় তাহলে কি করবে? যদি ভেবে দেখ যে, $p[b] = a$ করবে তাহলে ভুল। কারণ b কিন্তু কোম্পানির মালিক না। b কোম্পানির মালিক কে? এই যে কিছু ক্ষণ আগে বের করা হলঃ $Find(b)$. সুতরাং আমরা যা করব তা হল, $p[Find(b)] = a$ এর মানে হল b এর মালিক এখন a এর দ্বারা নিয়ন্ত্রিত। তোমরা চাইলে $p[Find(b)] = Find(a)$ ও করতে পার। একেই Union বলে। এর কোডও ৫.৫ এ আছে। অনেক সময় আমাদের জানার দরকার হতে পারে যে কোন মালিকের অধীনে কতগুলি কোম্পানি আছে বা যেসব কোম্পানি আছে তাদের মাঝে কোনটিতে সবচেয়ে বেশি মানুষ কাজ করে। এসব ক্ষেত্রে আমাদের যা করতে হবে তাহল $p[]$ ছাড়াও আমাদের $total$ বা max এর তথ্য রাখতে হবে এবং union-find এর সময় এই তথ্য গুলি আমাদের যথাযথ ভাবে update করতে হবে।

৫.৯ Square Root segmentation

একটা ছোট সমস্যা দিয়ে শুরু করি। মনে কর 0 হতে $n-1$ পর্যন্ত nটি দান বাক্স আছে। প্রথমে প্রতিটিতে 0 টাকা করে আছে। একজন করে আসে আর সে i তম বাক্সে t টাকা দান করে চলে যায়। মাঝে মাঝে তোমাকে জিজ্ঞাসা করা হবে যে i হতে j পর্যন্ত বাক্সগুলিতে মোট কত টাকা আছে। তুমি কত efficiently এই সমস্যা সমাধান করতে পারবে? এখন খুব সাধারণ একটি সমাধান হল i বাক্সে টাকা রাখতে হলে ঐ বাক্সের টাকার পরিমাণ বাড়িয়ে দেবঃ $amount[i]+=t$ আর query করলে i হতে j পর্যন্ত $amount$ যোগ করব। কিন্তু এখানে update অপারেশন মাত্র $\mathrm{O}(1)$ সময় নিলেও query অপেরেশন worst case এ $\mathrm{O}(n)$ সময় নিবে। তাহলে আমাদের এক্ষেত্রে query এর জন্য সময় update এর সময় থেকে বেশি। এখন সব গুলি সংখ্যা না যোগ করে কেমনে আমরা অনেক সংখ্যার যোগফল বের করতে পারি? যদি আমরা 0 হতে x বাক্সতে থাকা টাকার পরিমাণ $total[x]$ এ রাখি তাহলে খুব সহজেই $total[j]-total[i-1]$ করে i হতে j বাক্সে থাকা মোট টাকার পরিমাণ পেয়ে যেতে পারি ($i=0$ এর ক্ষেত্রে একটু সাবধানতা অবলম্বন করতে হবে), এক্ষেত্রে আমাদের query হয়ে যায় $\mathrm{O}(1)$. কিন্তু এই যে $total[x]$ এটা নির্ণয় এর জন্য আমাদের i এ t টাকা update এর সময় i হতে n পর্যন্ত $total$ এর পরিমাণ t করে বৃদ্ধি করতে হবে। অর্থাৎ এক্ষেত্রে আমাদের update হয়ে যাবে $\mathrm{O}(n)$. আমাদের আসলে এর মাঝামাঝি কোন একটি পদ্ধতি অবলম্বন করতে হবে, যেন কোনটিই খুব বড় না হয়ে যায়। প্রথম পদ্ধতিতে আমাদের update এ অনেক কম সময় লেগেছে কারণ আমরা খুব ছোট একটি জায়গায় পরিবর্তন করেছি, আবার দ্বিতীয় পদ্ধতিতে আমাদের query করতে কম সময় লেগেছে কারণ, অনেক গুলি সংখ্যার যোগফল আমরা এক জায়গায় রেখেছিলাম। আমরা যেটা করতে পারি তা হল, 0 হতে x পর্যন্ত সকল সংখ্যার যোগফল একত্র করে না রেখে কিছু কিছু করে সংখ্যার যোগফল একত্র করে রাখতে পারি। ধরা যাক এই কিছুর পরিমাণ হল k. অর্থাৎ, প্রথম k টি সংখ্যা (0 হতে $k-1$ বাক্সের টাকার পরিমাণ) একত্রে $sum[0]$ এ থাকবে, দ্বিতীয় k টি সংখ্যার যোগফল (k হতে $2k-1$ বাক্সের টাকার পরিমাণ) একত্রে $sum[1]$ এ থাকবে এরকম করে প্রতি k টি করে সংখ্যার যোগফল একত্রে থাকবে। তুমি যদি একটু ভালো করে চিন্তা কর তাহলে দেখবে i তম স্থানের সংখ্যা আসলে $sum[i/k]$ এ থাকে। সুতরাং update অপারেশনের সময় তোমাকে $amount[i]$ বৃদ্ধির সাথে সাথে $sum[i/k]$ কেও বাড়াতে হবে। অতএব আমাদের update হয় $\mathrm{O}(1)$ সময়ে। query এর সময় আমরা আলাদা আলাদা করে যোগ না করে বেশির ভাগ স্থান গুচ্ছ গুচ্ছ করে যোগ করব। ধরা যাক আমাদের বলা হল i হতে j পর্যন্ত যোগ করতে হবে। এখন i আছে $x=i/k$ তে আর j আছে $y=j/k$ এ। এখন যদি দেখা যায়, $x=y$ তাহলে আমরা i হতে j পর্যন্ত একটি loop চালাব, যদি তারা আলাদা হয় তাহলে, i হতে x range এর শেষ পর্যন্ত যোগ করব, j range এর শুরু হতে j পর্যন্ত যোগ করব আর $x+1$ হতে $y-1$ sum গুলি যোগ করব। কোন একটি range p এর শুরুর মাথার ফর্মুলা হল kp এবং শেষ মাথার ফর্মুলা হল $k(p+1)-1$. এই ফর্মুলা দুইটি ব্যবহার করে আমরা আমাদের যোগফল এর প্রথম দুই অংশ লুপ চালিয়ে বের করে ফেলব। এই কাজ করতে আমাদের খুব জোর $2k$ অপারেশন লাগবে, আর মাঝের sum গুলি যোগ করতে আমাদের n/k টি যোগ করতে হতে পারে, কারণ যেহেতু প্রতিটি range এর সাইজ k সুতরাং আমাদের মোট range এর সংখ্যা n/k. তাহলে আমাদের query এর জন্য সময় লাগবে $\mathrm{O}(k+n/k)$. তোমরা যদি ক্যালকুলাস জেনে থাকো বা inequality নিয়ে একটু ঘাটাঘাটি করে থাকো তাহলে জানো এই মানটি সর্ব নিম্ন হবে যদি $k=\sqrt{n}$ হয়। এবং এই ক্ষেত্রে query অপারেশনের জন্য $\mathrm{O}(\sqrt{n})$ সময় লাগে। $\mathrm{O}(n)$ এর তুলনায় $\mathrm{O}(\sqrt{n})$ কিন্তু অনেক কম! এই পদ্ধতিকেই Square Root Segmentation বলা হয়।

তোমরা চিন্তা করে দেখতে পার, আমাদের update অপারেশনে শুধু i কে t পরিমাণ না বাড়িয়ে যদি বলা হয় i হতে j পর্যন্ত সবাইকে t পরিমাণ বাড়াতে হবে তাহলে কেমন করে সমাধান করতে? এই সমস্যার সমাধানও আগের সমস্যার মতই তবে প্রতি range এর জন্য এক্ষেত্রে আলাদা আরেকটা variable রাখতে হবে যা নির্দেশ করবে এই range এর সকল $amount$ এর সাথে অতিরিক্ত কত যোগ করলে তুমি আসল টাকার পরিমাণ পাবে। আশা করি এতক্ষণে বুঝতে পারছ যে update এর সময় range গুলির ভিতরে ভিতরে গিয়ে প্রতিটিকে না বাড়িয়ে তুমি ঐ নতুন variable এর মান শুধু বাড়িয়ে দিলেই হয়ে যাবে! এক্ষেত্রে তোমার update এর জন্যও $\mathrm{O}(\sqrt{n})$ সময় লাগবে।

৫.১০ Static ডাটায় Query

এর আগে যা আলোচনা করলাম তাতে আমরা query করেছি, updateও করেছি। কিন্তু যদি কোন update না করা লাগে? অর্থাৎ প্রথমেই সকল সংখ্যা বা information দিয়ে দেয়া হবে তোমাকে, এর পরে query এর উত্তর দিতে হবে। আগের ডাটায় কোন রকম পরিবর্তন হবে না। এটা যদি sum এর জন্য query হয় তাহলে তো খুবই সোজা, সকল i এর জন্য তোমরা 1 হতে i পর্যন্ত যোগফল বের করে রাখবে (1-indexing মনে করি), এর পর তোমাকে যদি বলে i হতে j এর যোগফল কত? তাহলে 1 হতে j এর যোগফল থেকে 1 হতে $i-1$ এর যোগফল বাদ দিলেই $O(1)$ সময়ে উত্তর দিতে পারবে প্রতি query এর। এবং এর জন্য preprocessing সময় লাগবে $O(n)$.

কিন্তু আমাদের query যদি sum না হয়ে maximum বা minimum হয়? একটি উপায় হল আগের মত Square Root Segmentation ব্যবহার করা। সেক্ষেত্রে আমাদের preprocessing সময় লাগবে $O(n)$ আর query এর জন্য সময় লাগবে $O(\sqrt{n})$. Tarjan এর একটি বিখ্যাত research আছে এই ব্যপারে, সে preprocessing $O(n)$ সময়ে এবং query $O(1)$ সময়ে কর-তে পারে, তবে সেই method টি বেশ complex. তোমরা চাইলে পড়ে দেখতে পার এই ব্যপারে internet এ। আমরা এখন যেই method দেখব তাতে আমাদের preprocessing সময় লাগবে $O(n \log n)$ আর query সময় লাগবে $O(1)$. যেহেতু maximum এবং minimum বের করার method প্রায় একই আমরা এখানে maximum বের করব। সংক্ষেপে এই পদ্ধতিটি হবে এরকমঃ

১. প্রথমে আমরা প্রতি 1 সাইজের segment এর maximum বের করবঃ $[1,1], [2,2], [3,3], \ldots$

২. এর পর আমরা প্রতি 2 সাইজের segment এর maximum বের করবঃ $[1,2], [2,3], [3,4], [4,5] \ldots$

৩. এর পর আমরা প্রতি 4 সাইজের segment এর maximum বের করবঃ $[1,4], [2,5], [3,6], [4,7] \ldots$

৪. এর পর আমরা প্রতি 8 সাইজের segment এর maximum বের করবঃ $[1,8], [2,9], [3,10], [4,11] \ldots$

৫. এভাবে i তম ধাপে আমরা 2^i সাইজের segment এর maximum বের করবঃ $[1, 2^i], [2, 2^i + 1] \ldots$.

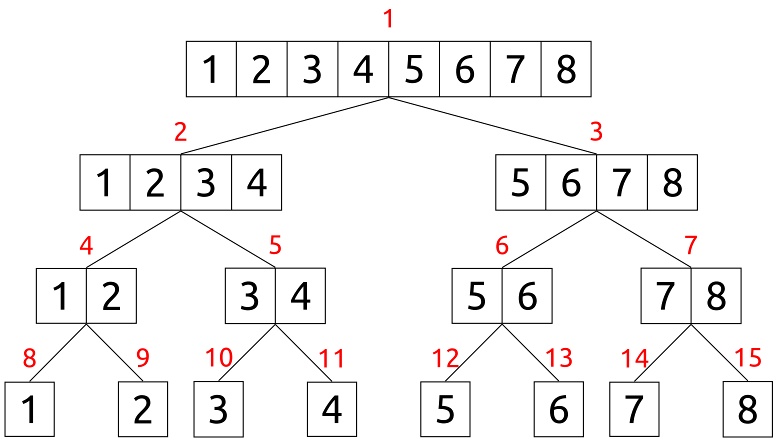
এভাবে চলতে থাকবে যতক্ষণ $2^i \le n$ হয় অর্থাৎ $i \le \log(n)$. এই preprocessing এর সময় আমরা x হতে শুরু করে 2^i সাইজের maximum কেমনে বের করব? খুব সহজ, x হতে শুরু করে 2^i সাইজের segment এর maximum হবে x হতে শুরু করে 2^{i-1} সাইজের segment এর maximum এবং $x + 2^{i-1}$ হতে শুরু করে 2^{i-1} সাইজের segment এর maximum- এই দুইটি সংখ্যার maximum এর সমান। অর্থাৎ, $table[i][x] = max(table[i-1][x], table[i-1][x + (1 << (i-1))]$. প্রতি ধাপে আমাদের $O(n)$ সময় লাগছে। যেহেতু সর্বমোট $O(\log n)$ টি ধাপ আছে সুতরাং আমাদের মোট সময় লাগবে $O(n \log n)$. এখন তোমাকে যদি জিজ্ঞাসা করে i হতে j এর max কত? তোমাকে সবচেয়ে ছোট x বের করতে হবে যেন $2^x \le j - i + 1$ হয়। এটি তুমি চাইলে একটি loop চালিয়ে বের করতে পার সেক্ষেত্রে $O(\log n)$ সময় লাগবে, তবে তোমরা প্রথমেই যদি একটি array বানাতে পার যেখানে প্রতিটি সংখ্যার জন্য এই মান বের করা থাকে, তাহলে সেই array ব্যবহার করে তোমরা খুব সহজেই $O(1)$ এ x এর মান নির্ণয় করতে পারবে। তাহলে $max(table[x][i], table[x][j - (1 << x) + 1])$ ই হল আমাদের কাঙ্ক্ষিত মান।

৫.১১ Segment Tree

Binary Search Tree কোড করা বেশ কষ্টকর ব্যাপার। সে তুলনায় এরই জাত ভাই Segment Tree অনেক ভদ্র। জাত ভাই এই অর্থে যে এখানেও BST এর মতই insert, delete, update ইত্যাদি অপারেশন করা যায় তবে এক্ষেত্রে আমাদের সংখ্যাগুলি 1 হতে n এর মাঝে সীমাবদ্ধ থাকে। শুধু সংখ্যা insert বা delete না, কোন একটি index এ চাইলে আমি কিছু সংখ্যা replace করতে পারি বা একটি range এ আমরা query ও করতে পারি। তবে হুট করে দুইটি index এর মাঝে নতুন

একটি index বসিয়ে দিতে পারব না। অর্থাৎ তুমি যদি চাও যে 1 আর 2 এর মাঝে নতুন একটি জিনিস বসাবে তা হবে না। তাহলে এর সাহায্যে কি কি করা যায়? একটা ছোট খাটো তালিকা বানানো যাকঃ কোন একটি range এ query যেমনঃ সংখ্যা গুলির যোগফল, সবচেয়ে বড় সংখ্যা, জোড় সংখ্যা গুলির যোগফল ইত্যাদি; কোন একটি সংখ্যাকে পরিবর্তন করা; কোন একটি range এর প্রতিটি সংখ্যাকে update করা যেমনঃ নির্দিষ্ট সংখ্যা যোগ করা ইত্যাদি। এটি কোড করা তুলনামুলকভাবে অনেক সহজ। সাধারনত Segment Tree ব্যবহার করে আমরা যেসব সমস্যা সমাধান করতে পারি Square Root Segmentation ব্যবহার করেও করতে পারি। তবে Square Root Segmentation এর ক্ষেত্রে আমাদের complexity হয় $O(\sqrt{n})$ আর Segment Tree এর ক্ষেত্রে হয় $O(\log n)$। Square Root Segmentation এর ক্ষেত্রে আমরা একটি সমস্যা নিয়ে আলোচনা করছিলামঃ update হল array এর একটি স্থানে একটি সংখ্যা যোগ করা আর query হল array এর কোন একটি range এর যোগফল প্রিন্ট করা। আমরা এই সেকশনে দেখব কেমনে এই সমস্যায় update ও query দুটিই Segment Tree ব্যবহার করে $O(\log n)$ সময়ে করা যায়।

৫.১১.১ Segment Tree Build করা



চিত্র ৫.৪: Segment Tree Build

$n = 8$ এর জন্য Segment Tree চিত্র ৫.৪ এ দেখানো হল। আপাতত লাল সংখ্যাগুলিকে বাদ দাও। আমাদের কাছে মোট 8 টি জায়গা আছে $[1, 8]$। আমরা যা করব তা হল এই জায়গা গুলিকে সমান দুই ভাগে ভাগ করবঃ $[1, 4]$ এবং $[5, 8]$. যদি আমাদের কাছে $[L, R]$ এরকম একটি range থাকে তাহলে একে দুই ভাগ করলে দাঁড়াবে $[L, mid]$ এবং $[mid + 1, R]$ যেখানে $mid = (L + R)/2$ (এখানে কিন্তু integer division হচ্ছে)। এখন এভাবে আমরা সকল range কে দুই ভাগে ভাগ করতে থাকব যতক্ষণ না আমাদের segment এ একটি মাত্র সংখ্যা থাকে, অর্থাৎঃ $L = R$. মনে কর না যে আমাদের দেখানো segment tree তে n একটি 2 এর power বলে এটা সম্ভব হয়েছে। যদি $n = 3$ হয় তাহলে আমাদের প্রথম segment $[1, 3]$ কে ভাঙ্গলে আমরা পাবো $[1, 2]$ ও $[3, 3]$ এবং $[1, 2]$ কে ভাঙ্গলে $[1, 1]$ ও $[2, 2]$। এখন কথা হল, আমরা তো খুব সুন্দর করে কাগজে কলমে ছবি একে ফেললাম কিন্তু এটা কোড এ করব কেমনে? এবার লাল সংখ্যা গুলি খেয়াল কর। আমরা প্রতিটি segment এর একটি করে নাম্বার দিয়েছি। ছবির মত করে নাম্বার দেয়ার একটা বিশেষত্ব আছে। খেয়াল করলে দেখবে, কোন নাম্বার x এর বামে নিচে (left child) সবসময় $2x$ এবং ডানে নিচে সবসময় $2x + 1$ থাকে। আবার তার উপরে (parent) $x/2$ হয়। এই ট্রিক খাটিয়ে আমরা খুব সহজেই একটা segment tree বানাতে পারি। কোড ৫.৬ এ কিভাবে একটি Segment Tree বানানো যায় তা দেখানো হল। এখানে আমাদের প্রয়োজন মত কোড পরিবর্তন করতে হবে। যেমন যদি বলা থাকে যে আমাদের পুরো segment প্রথমে ফাঁকা তাহলে আমরা 0 দ্বারা initialize করব। আবার অনেক সময়

আমাদের বলা থাকে কোন ঘরে কত সংখ্যা আছে, সেক্ষেত্রে আমরা একদম শেষ segment এ গিয়ে সঠিক সংখ্যা বসাব বা ফিরে এসে সঠিক যোগফল রাখব (sum[at] = sum[at * 2] + sum[at * 2 + 1]). খেয়াল করলে দেখবে আমাদের tree এর প্রথম level এ আছে মাত্র 1 টি node, দ্বিতীয় level এ আছে মাত্র 2 টি, এর পরে 4 টি, এরকম করে $8, 16 \ldots n$ টি node, যা যোগ করলে দাঁড়ায় $2n$. সুতরাং আমাদের time complexity $\mathrm{O}(n)$.

কোড ৫.৬: segmentTreeBuild.cpp

```
void build(int at, int L, int R)
{
    //do initialization like: sum[at] = 0
    if(L == R)
    {
        //might need to do, something like: sum[at] = num[L]
        return;
    }
    int mid = (L + R)/2;
    build(at * 2, L, mid);
    build(at * 2 + 1, mid + 1, R);
    //do initialization like: sum[at] = sum[at * 2] + sum[at * 2 + 1] etc.
}
```

Segment Tree এর ক্ষেত্রে Time Complexity এর থেকেও important বলা যায় Space Complexity. কারন অনেকেই ভুল সাইজের array declare করার জন্য Run Time Error পেয়ে থাকে। সবসময় মনে রাখবে, তোমার n যত ঠিক তার 4 গুন বা তার বেশি সাইজের array declare করতে হবে। এর কারন হল n কিন্তু সবসময় এরকম 2 এর power এ থাকবে না। 2 এর power এ থাকলে এটি দুই গুন। কিন্তু 2 এর power এ না থাকলে আসলে এই memory সাইজ accurately বের করা একটু কঠিন হয়। আমরা জানি x এবং $2x$ এর মাঝে অবশ্যই একটি 2 এর power আছে। আর আমরা জানি 2 এর power এর ক্ষেত্রে দ্বিগুণ লাগে, সুতরাং আমরা 4 গুন সাইজ declare করে থাকি। অনেকে n এর পরের 2 এর power এর দ্বিগুন declare করে। তাহলেও হবে, কিন্তু সেক্ষেত্রে একটু হিসাব নিকাশ করতে হবে। সুতরাং আমরা চোখ বন্ধ করে চারগুণ declare করে থাকি।

৫.১১.২ Segment Tree Update করা

প্রথমে update দিয়ে শুরু করা যাক। আমরা কোন একটি সংখ্যাকে বাড়াতে চাই। আমরা root থেকে শুরু করব। আমাদের root হল $[1, 8]$ এবং ধরা যাক আমরা 3 কে update করতে চাই। আমরা যা করব তাহল আমাদের সংখ্যাটা যেদিকে আছে সেদিকে যাব অর্থাৎ, root হতে $[1, 4]$ এ যাব, এর পর $[3, 4]$ এবং সবশেষে $[3, 3]$। এবং ফেরার পথে আমরা build এর সময় যেভাবে sum এর array কে populate করেছিলাম ঠিক সেভাবে আমরা sum এর array কে update করব। অর্থাৎ আমাদের update ফাংশন দেখতে ৫.৭ এর মত হবে। যেহেতু আমাদের tree এর height $\log(n)$ সুতরাং আমাদের update এর time complexity ও হবে $\mathrm{O}(\log n)$।

কোড ৫.৭: segmentTreeQuery.cpp

```
void update(int at, int L, int R, int pos, int u)
{
```

```
৩     //sometimes instead of using if−else in line 11 and
          12
৪     //you can use: if(at < L || R < at) return;
৫     if(L == R)
৬     {
৭         sum[at] += u;
৮         return;
৯     }
১০
১১    int mid = (L + R)/2;
১২    if(pos <= mid) update(at * 2, L, mid, pos, u);
১৩    else update(at * 2 + 1, mid + 1, R, pos, u);
১৪
১৫    sum[at] = sum[at * 2] + sum[at * 2 + 1];
১৬ }
```

৫.১১.৩ Segment Tree তে Query করা

এখন আসা যাক query তে। আমরা জানতে চাই $[l, r]$ এই range এ থাকা সংখ্যা গুলির যোগফল কত। আমরা আগের মত tree এর root হতে শুরু করে ধীরে ধীরে নিচের দিকে যেতে থাকব। যদি কখনও দেখি আমরা এখন যেই node এ আছি তার range আমাদের query range এর বাইরে তাহলে তো আর এখান থেকে নিচে যাবার দরকার নেই, তাই না? সুতরাং আমরা এখান থেকেই বলব যে এই range এর জন্য উত্তর 0. যদি আমরা এমন একটি node এ থাকি যা পুরোপুরি আমাদের query range এর ভিতরে তাহলেও কিন্তু নিচে যাবার দরকার নেই, সেক্ষেত্রে আমরা আমাদের এই node এর sum এর মান return করব। যদি এই দুই case এর কোনটিই না হয় এর মানে দাঁড়ায় যে আমাদের query range আসলে এই node এর দুই child এই কিছু কিছু করে আছে। সুতরাং আমরা দুইদিকেই যাব এবং দুই দিক থেকে আসা sum কে যোগ করে return করব। এর কোড তোমরা কোড ৫.৮ তে দেখতে পাবে।

কোড ৫.৮: segmentTreeQuery.cpp

```
int query(int at, int L, int R, int l, int r)
{
    if(r < L || R < l) return 0;
    if(l <= L && R <= r) return sum[at];

    int mid = (L + R)/2;
    int x = query(at * 2, L, mid, l, r);
    int y = query(at * 2 + 1, mid + 1, R, l, r);

    return x + y;
}
```

এখন কথা হল এর time complexity কত! আমাদের মনে হতে পারে এর time complexity অনেক বেশি! কেউ কেউ ভাবতে পারে যেহেতু আমাদের tree তে $O(n)$ সংখ্যক node আছে তাই এর time complexity ও $O(n)$. না! খেয়াল কর যদি কখনও $[1, 1]$ ও $[2, 2]$ আমাদের query range এর মাঝে থাকে তার মানে দাঁড়ায় আমরা আসলে $[1, 2]$ থেকেই ফিরে যাব। অর্থাৎ তুমি যদি আসলে অনেক বেশি range কে cover করতে চাও তাহলে একটা বড় range থেকেই তুমি ফিরত

যাবে। তা নাহয় বুঝা গেল কিন্তু complexity টা আসলে কত? একটু চিন্তা করে দেখ, আমরা কখন কাজ করতেসি? যখন নিচে নামতেসি। কখন নিচে নামতেসি? যখন আমাদের node এর range আমাদের query range এর সাথে partially overlap করে। খেয়াল কর, আমাদের tree এর কোন level এ কিন্তু দুইটার বেশি partially overlap করা node থাকবে না, তাই না? বাকি গুলো হয় বাইরে নাহয় একদম ভিতরে হবে। আমরা তখনই নিচে নামি যখন partially overlap হয়। যেহেতু প্রতি level এ partially overlap এর সংখ্যা 2 আর আমাদের level আছে $\log n$ টি সুতরাং আমাদের time complexity হবে $O(\log n)$. আমরা হয়ত এই জিনিস অন্য ভাবে প্রমান করতে পারতাম, কিন্তু আমি এখানে এভাবে দেখালাম। কারন এখানে time complexity যে আসলে অনেক বেশি না সেটা আমরা tree এর structure দেখে প্রমান করলাম। এরকম প্রমান আরো পাবে। তোমরা যদি কখনও Link Cut Tree নিয়ে পড়ার সুযোগ পাও তখন সেখানে এরকম প্রমান দেখতে পাবে। এবং সেই প্রমান আমার কাছে অনেক amazing লেগেছিল!

৫.১১.৪ Lazy without Propagation

মনে কর তোমাদেরকে বলা হল যে n টি বাল্ব পর পর আছে এবং শুরুতে তারা সবাই off. এখন একটি অপারেশনে i হতে j পর্যন্ত সকল বাল্ব toggle করতে বলা হতে পারে। [১] আবার তোমাকে কখনও কখনও জিজ্ঞাসা করা হতে পারে যে q তম বাল্বটি on আছে নাকি off? তুমি এই সমস্যার সমাধান Segment Tree ব্যবহার করে $O(\log n)$ এ করতে পারবে। এক্ষেত্রে idea হল যখন তুমি i হতে j পর্যন্ত বাল্বকে toggle করবে তখন কিন্তু তুমি এই সীমার মাঝে প্রতিটি বাল্বকে update করতে পারবে না, তোমাকে গুচ্ছ ধরে update করতে হবে। ধর তোমার কাছে $n = 8$ টি বাল্ব আছে আর তোমাকে 1 হতে 4 পর্যন্ত বাল্ব update করতে বলা হল। তুমি যা করবে তাহল Segment Tree এর শুধু $[1, 4]$ এর segment এ লিখে রাখবে যে এই সীমার প্রতিটি বাল্ব toggle করা হয়েছে। চিত্র ৫.৪ এর সাথে তুলনা করতে পার। যদি তোমাকে বলে 1 হতে 3 পর্যন্ত toggle করতে হবে, তাহলে তুমি $[1, 2]$ এবং $[3, 3]$ এই সীমা দুইটি update করবে। update এর সময় শুধু তুমি লিখে রাখবে যে এই সীমাটি কত বার toggle হয়েছে। লাভ কি? ধর তোমাকে জিজ্ঞাসা করল 3 এর অবস্থা কি? তুমি যা করবে, root থেকে $[3, 3]$ পর্যন্ত যাবে এবং গুনবে এটি যেই যেই সীমা দিয়ে যায় সেসব সীমা কতবার করে toggle হয়েছে। এথেকেই তুমি তোমার উত্তর পেয়ে যাবে। তাহলে query যে মাত্র $O(\log n)$ এ হচ্ছে তাতো খুব সহজেই বুঝা যায়, কিন্তু update? আমরা কিন্তু ইতোমধ্যেই সাবসেকশন ৫.১১.৩ তে এরকম কিছু একটা প্রমান করে এসেছি। সুতরাং আমাদের update ও $O(\log n)$. কোড ৫.৯ এ query ও update এর কোড দেয়া হল। আমাদের এই সমাধানে আমরা যে একদম নিচ পর্যন্ত না গিয়ে উপরেই কিছু একটা লিখে রেখে শেষ করে ফেলেছি update এর কাজ, একেই lazy বলা হয়। আমরা এখানে lazy কে কিন্তু ভেঙ্গে নিচে নামায় নাই, সে জন্য একে without propagation বলে। ভেঙ্গে নিচে নামানোর মানে কি তা পরবর্তী সাবসেকশনেই পরিষ্কার হয়ে যাবে।

কোড ৫.৯: lazyWithoutPropagation.cpp

```
void update(int at, int L, int R, int l, int r)
{
    if(r < L || R < l) return;
    if(l <= L && R <= r) {toggle[at] ^= 1; return;}

    int mid = (L + R)/2;
    update(at * 2, L, mid, l, r);
    update(at * 2 + 1, mid + 1, R, l, r);
}

//returns 1 if ON, 0 if OFF
```

[১] toggle অর্থ হল on থাকলে off করা বা off থাকলে on করা।

```
int qurery(int at, int L, int R, int pos)
{
    if(pos < L || R < pos) return 0;
    if(L <= pos && pos <= R) return toggle[at];

    int mid = (L + R)/2;
    if(pos <= mid) return query(at * 2, L, mid, pos) ^ 
        toggle[at];
    else return query(at * 2 + 1, mid + 1, R, pos) ^ 
        toggle[at];
}
```

৫.১১.৫ Lazy With Propagation

মনে করা যাক উপরের সমস্যায় আমাদের কোন একটি বাল্ব সম্পর্কে না জিজ্ঞাসা করে জিজ্ঞাসা করা হবে যে l হতে r এর মাঝে কত গুলি বাল্ব on আছে! বলে রাখা ভাল যে এই সমস্যাও একটু চিন্তা করলে without propagation এ সমাধান করা সম্ভব। কিন্তু আমরা এখানে দেখাব কেমন করে এই সমস্যা with propagation এ সমাধান করা যায়। সমাধানে যাবার আগে আমাদের একটু চিন্তা করা দরকার আমাদের এই সমস্যার সমাধানের জন্য tree এর প্রতি node এ কি কি জিনিস দরকার! প্রথমত Lazy দরকার, অর্থাৎ এই node এর প্রতিটি বাল্ব কি toggle করা হয়েছে কি হয় নাই এবং আরও দরকার এই range এর কতগুলি বাল্ব এখন on আছে। off এর সংখ্যা কিন্তু দরকার নাই, কারন তুমি যদি on এর সংখ্যা জানো তাহলে off এর সংখ্যা এমনিতেই বেরিয়ে আসবে। সুতরাং build পর্যায়ে আমাদের প্রতি node এ লিখতে হবে $toggle = 0$ এবং $on = 0$. এখন আসা যাক আমরা কেমনে update করব। আগের মতই আমরা দেখব যদি আমাদের বর্তমান node এর সীমা যদি query range এর সম্পূর্ণ বাইরে হয় তাহলে কিছু করব না, যদি partially ভিতরে হয় তাহলে সেভাবেই আমরা ডানে বা বামে যাব (বা উভয় দিকে)। এখন আসা যাক যদি সম্পূর্ণ ভাবে ভিতরে হয় তাহলে কি করব। খুব সহজ, $toggle = toggle \wedge 1$ করব, এবং $on = R - L + 1 - on$ করব। আশা করি বুঝা যাচ্ছে এই দুই লাইন এ আসলে কি করা হচ্ছে। কিন্তু শুধু এটুকু করলে কিন্তু হবে না। কেন? একটু দুরের চিন্তা কর। মনে কর তুমি $[1, 4]$ কে এভাবে update করলে। এর পর যদি তোমাকে বলে $[1, 2]$ কে update করতে হবে। তুমি কি করবে? ঐ node এ গিয়ে একই ভাবে update করে আসবে তাই না? কিন্তু যদি এর পরে তোমাকে query করে $[1, 4]$ এ কতগুলি on আছে, তখন তুমি কেমনে উত্তর দিবে? তুমি কিন্তু নিচে $[1, 2]$ তে পরিবর্তন করে এসেছ, সুতরাং তুমি $[1, 4]$ থেকেই উত্তর দিতে পারবে না এখন, কারন $[1, 2]$ এর পরিবর্তন $[1, 4]$ এ কিন্তু নেই।[১] তাহলে উপায় কি? আগে দেখ আমাদের সমস্যাটা কি! আমরা যে $[1, 4]$ এর update এর সময় সেখান থেকেই ফিরে গিয়েছি সেটা সমস্যা। আমরা যদি তা না করে একদম নিচ পর্যন্ত নামতাম এবং node গুলির on ঠিক মত update করতাম তাহলেই হয়ে যেত। কিন্তু তা করলে আমাদের time complexity বেড়ে যাবে। তাহলে আমরা কি করব? উপায় হল, তুমি এখানেই lazy রেখে যাবে, কিন্তু যদি কখনও এর থেকে নিচে নামতে হয় তাহলে তখন তুমি এই lazy কে এক ধাপ নামিয়ে দিবে। অর্থাৎ, আমার যতক্ষণ না দরকার পরবে ততক্ষণ আমরা lazy নামাব না। এই যে lazy কে দরকার এর সময় নিচে নামান একেই বলে propagation. আমরা $[1, 4]$ এর update এর পর যখন $[1, 2]$ কে update করব তার আগেই অর্থাৎ যখন আমরা $[1, 4]$ থেকে নিচে নামতে চাইব তখন আমরা দেখব $[1, 4]$ এ কোন lazy আছে কিনা, যদি থাকে তাহলে তাকে আগে propagate করব, এর পর নিচে নামব। সেখানে দেখব lazy আছে কিনা, থাকলে তা propagate করে আবার নিচে নামব। এরকম করে চলতে থাকবে। একই ভাবে query এর সময় ও আমরা যদি কোন node দিয়ে নিচে নামতে চাই, নামার আগেই আমাদের দেখে নিতে হবে এখানে কোন lazy আছে কিনা এবং সেই অনুসারে তাকে দরকার হলে নিচে নামাতে হবে।

এখন আমরা lazy কে কেমনে নামাতে পারি? তার আগে চিন্তা করে দেখ, একটা lazy কে নামালে কে কে পরিবর্তন হতে পারে? আমার node এর toggle ও on, আমার left ও right child এর

[১]একটু চিন্তা করলে তোমরা without propagation এ তাহলে কি করতে হবে তা বের করে ফেলতে পারবে

toggle ও on. Lazy থাকা মানে হল $toggle = 1$. একে নিচে নামান মানে, আমার left ও right child এর $toggle = toggle \wedge 1$ হবে এবং একই সাথে on ও পরিবর্তন হবে। আর আমার বর্তমান node এর $toggle = 0$ হবে, কিন্তু on পরিবর্তন হবে না (কারন আমরা যখন toggle করে ছিলাম তখনই আসলে on পরিবর্তন করেছিলাম)। একটু চিন্তা করলে দেখবে যে, যদি পর পর দুইবার তোমাকে $[1, 4]$ এ toggle করতে বলে তাহলে কিন্তু তোমাকে এর মাঝে propagation করার দরকার নেই। কারন তুমি নিচে নামছ না, শুধু দুইবার $toggle = toggle \wedge 1$ এবং $on = R - L + 1 - on$ করতে হবে। এবং এর ফলে প্রথম বারে যেই lazy জমতো সেটা পরের update এর কারনে cancel হয়ে যাচ্ছে। অর্থাৎ আমাদের প্রব্লেম এ আসলে lazy cancel ও হতে পারে। আসলে cancel হল কি হল না তা নিয়ে তোমাকে চিন্তা করতে হবে না, যদি দেখ $toggle = 1$ এর মানে তোমার এখানে lazy আছে, শেষ! তাহলে এবার এর কোড দেখা যাক। কোড ৫.১০ এ এই কোড দেয়া হল।

কোড ৫.১০: lazyWithPropagation.cpp

```
void Propagate(int at, int L, int R)
{
    int mid = (L + R)/2;
    int left_at = at * 2, left_L = L, left_R = mid;
    int right_at = at * 2 + 1, right_L = mid + 1, right_R = R;

    toggle[at] = 0;
    toggle[left_at] ^= 1;
    toggle[right_at] ^= 1;

    on[left_at] = left_R - left_L + 1 - on[left_at];
    on[right_at] = right_R - right_L + 1 - on[right_at];
}

void update(int at, int L, int R, int l, int r)
{
    if(r < L || R < l) return;
    if(l <= L && R <= r) {toggle[at] ^= 1; on[at] = R - L + 1 - on; return;}

    if(toggle[at]) Propagate(at, L, R);

    int mid = (L + R)/2;
    update(at * 2, L, mid, l, r);
    update(at * 2 + 1, mid + 1, R, l, r);

    on[at] = on[at * 2] + on[at * 2 + 1];
}

int qurery(int at, int L, int R, int l, int r)
{
    if(r < L || R < l) return;
    if(l <= L && R <= r) return on[at];

```

```
    if(toggle[at]) Propagate(at, L, R);

    int mid = (L + R)/2;
    int x = query(at * 2, L, mid, l, r);
    int y = query(at * 2 + 1, mid + 1, l, r);

    return x + y;
}
```

৫.১২ Binary Indexed Tree

সংক্ষেপে একে BIT বলা হয়। এটা Segment Tree এর মতই একটি ডাটা স্ট্রাকচার তবে এটি একটু জটিল, কিন্তু মজার ব্যপার হল এর কোড খুবই ছোট। তুমি পুরো ডাটা স্ট্রাকচার না বুঝেও ব্যবহার করতে পারবে। সত্যি কথা বলতে আমি নিজেও এই ডাটা স্ট্রাকচার খুব ভাল মত বুঝি না। কিন্তু এটা ব্যবহার করতে আমার খুব একটা কষ্ট হয় না। এটা ঠিক যে, BIT দিয়ে তুমি যা যা করতে পারবা Segment Tree দিয়েও প্রায় সবই তুমি করতে পারবা, তবে Segment Tree দিয়ে এমন কিছু করা যায় যা আসলে তুমি BIT দিয়ে করতে পারবা না। কিন্তু BIT এর সুবিধা হল, এটি অনেক ছোট কোড, এর জন্য n সাইজের মেমরি লাগে এবং এটি অনেক fast. তোমরা এর সম্পর্কে আরো বিস্তারিত জানতে চাইলে টপকোডার এর article পড়তে পার।

খুব সংক্ষেপে বলতে হলে বলা যায়, BIT এ তোমরা দুই ধরনের operation করতে পার।

১. কোন একটি স্থান idx কে v পরিমান বৃদ্ধি update(idx, v)

২. শুরু হতে idx পর্যন্ত যোগফল বের করা read(idx)

আরও বেশ কিছু operation করা যায় যা আসলে অত বহুল ব্যবহৃত না। তোমরা topcoder এর tutorial এ দেখতে পার।

কোড ৫.১১ এ read এবং update দেখানো হল। এখানে MaxVal হল n এর মান। অর্থাৎ তোমার array যত বড় আর কি! আর BIT এ তোমাকে 1-indexing ব্যবহার করতে হবে।

কোড ৫.১১: bit.cpp

```
int read(int idx)
{
    int sum = 0;

    while (idx > 0)
    {
        sum += tree[idx];
        idx -= (idx & -idx);
    }

    return sum;
}

void update(int idx ,int val)
{
    while (idx <= MaxVal)
    {
```

```
        tree[idx] += val;
        idx += (idx & -idx);
    }
}
```

অধ্যায় ৬

Greedy টেকনিক

Greedy মানে তো সবাই বুঝো? এর মানে হল লোভী। যেমন ধর তোমাকে একটা buffet তে নিয়ে গিয়ে ছেড়ে দিলে কি করবে? তুমি হাপুস হুপুস করে খাওয়া শুরু করে দেবে তাই না? যদি একটু বুদ্ধিমান হউ তাহলে হয়তো সবচেয়ে দামি খাবার বেশি বেশি করে খাবা! কারন যার দাম কম তা হয়তো তুমি পরে কিনে খেতেই পারবে! যেমন যদি buffet তে গলদা চিংড়ি থাকে আর জিলাপি থাকে, তাহলে নিশ্চয় জিলাপি খেয়ে পেট ভরানোর থেকে চিংড়ি খেয়ে পেট ভরানো বুদ্ধিমানের মত কাজ হবে? Greedy মানে যে সবসময় বেশি বেশি করে নেয়া তা কিন্তু না। অনেক সময় কম কম নেয়াও লাভ জনক। যেমন তোমাকে বলা হল একটি গাড়ি কিনতে। এখন গাড়ি গুলো এককেটা একেক পরিমান তেল খায়! নিশ্চয় যেই গাড়ি সবচেয়ে কম তেল খায় সেটা কেনাই বুদ্ধিমানের মত কাজ তাই না? যদিও বাস্তব জীবনে আরও অনেক factor আছে! যাই হক, তো Greedy মানে হল অন্য কিছু না দেখে যার মান কম বা বেশি তাকে সবসময় বাছাই করা।

৬.১ Fractional Knapsack

Greedy algorithm এর জন্য এটি খুবই common সমস্যা। মনে কর একটা চোর একটি মুদি দোকানে ঢুকেছে চুরি করতে। সেখানে চাল আছে, ডাল আছে, চিনি, লবন এরকম নানা জিনিস আছে। এখন সে সব জিনিস চুরি করতে পারবে না। কারন তার কাছে যেই থলে আছে তার ধারন ক্ষমতা ধরা যাক 20 kg. তাহলে সে কিভাবে চুরি করলে সবচেয়ে বেশি লাভবান হবে? খুবই সহজ। যেই জিনিসটার দাম সবচেয়ে বেশি তুমি সেই জিনিস আগে নেয়া শুরু করবে। যদি দেখ ঐ জিনিস নেয়া শেষ এবং এখনও থলে তে কিছু জায়গা বাকি আছে তাহলে তুমি পরবর্তী দামি জিনিস নেয়া শুরু করবে। এরকম করে যতক্ষণ না তোমার থলের ধারন ক্ষমতা শেষ হচ্ছে তুমি নিতে থাকবে। এখানে খেয়াল কর, দাম বেশি মানে কিন্তু প্রতি kg এর দাম। ধর চাল আছে 1 kg আর দাম 100 টাকা, আর ডাল আছে 500g কিন্তু এর দাম 60 টাকা তাহলে কিন্তু ডাল নেয়া লাভজনক হবে কারন, ডাল এর দাম প্রতি kg তে 120 টাকা!

এই সমাধান ঠিক থাকবে যদি তুমি কোন জিনিসের যেকোনো পরিমান নিতে পার। সমস্যাটা যদি চাল ডাল না হয়ে electronics এর দোকান হয় তাহলে তুমি আর এভাবে সমাধান করতে পারবে না। তুমি তো আর একটা tv ভেঙ্গে এর অর্ধেক চুরি করবে না তাই না? tv হোক laptop হোক আর mobile ই হোক তুমি যাই নিতে চাও না কেন পুরোটাই নিতে হবে। এই ক্ষেত্রে কিন্তু আমাদের greedy মেথড কাজ করবে না। উদাহরন দেয়া যাক, মনে কর 1 টা tv এর দাম 15000 টাকা এবং ওজন 15kg, দুইটা monitor আছে যাদের ওজন 10kg করে এবং প্রতিটার দাম 9000 টাকা। তোমার কাছে 20 kg জিনিস নেবার থলে আছে। তুমি কি করবে? tv নেয়া কিন্তু বোকামি হবে যদিও এর প্রতি kg তে দাম বেশি তাও তোমার দুইটা monitor নিলে লাভ হবে সবচেয়ে বেশি। সুতরাং এটা মনে করার কিছু নাই যে Greedy মেথড সবসময় কাজ করবে। যদি তুমি জিনিসের চাইলে "কিছু" অংশ নিতে পার তাহলে সেই সমস্যাকে বলা হয় Fractional Knapsack আর যদি তোমাকে পুরোপুরি নিতে হয় তাহলে সেই সমস্যাকে বলা হয় 0-1 knapsack (এটি পরবর্তী অধ্যায় এ আমরা দেখব কেমনে সমাধান করতে

হয়)।

৬.২ Minimum Spanning Tree

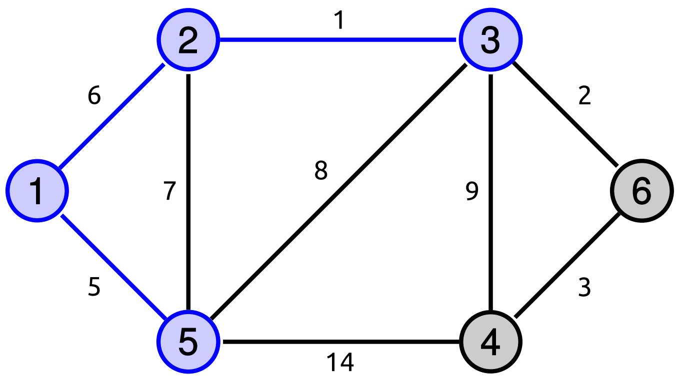
এই সেকশন এর নাম দেখে ভয় পাবার কিছু নেই। খুবই সহজ জিনিস। আমরা আগেই জেনে এসেছি Tree কাকে বলে, এখন দেখে নেয়া যাক Minimum Spanning Tree কি জিনিস। মনে কর আমাদের একটা weighted graph দেয়া আছে (weight গুলি ধনাত্মক) অর্থাৎ কিছু vertex, কিছু edge এবং সেই edge গুলির weight. তোমাকে এখন এদের মাঝ থেকে কিছু edge বাছাই করতে হবে যেন তাদের weight এর যোগফল সর্বনিম্ন হয় এবং সকল vertex যেন connected হয়। এটা নিশ্চয় বুঝতে পারছ যে তোমার যদি n টি vertex থাকে তাদের connected করতে আসলে তোমার সর্বনিম্ন $n-1$ টা edge লাগবেই। এবং তুমি যদি তোমার বাছাই করা edge গুলির weight এর যোগফল সর্বনিম্ন করতে চাও তাহলে অবশ্যই $n-1$ টার বেশি edge নিবে না। আর n vertex এবং $n-1$ edge ওয়ালা একটি connected graph হল tree. অর্থাৎ আমাদের সকল vertex কে connected করতে আমরা যেসকল edge নির্বাচন করব তাদের weight এর যোগফল সর্বনিম্ন করতে চাইলে যেই graph টি দাঁড়ায় সেটিই হল Minimum Spanning Tree (সংক্ষেপে MST). এখানে Minimum আর Tree শব্দ দুইটি তো বুঝছই? Spanning অর্থ connected মনে করতে পার। এখানে আমরা MST বের করার জন্য দুইটি algorithm এর কথা বলব। তোমরা কেউ কেউ মনে করতে পার যে হয়তো এই সেকশনটি গ্রাফ এর অধ্যায়ে থাকলে ভাল হত। কিন্তু আমরা যেই দুইটি algorithm আলোচনা করব তারা আসলে Greedy টাইপ বলা যায়। আর তোমরা কেমনে একটি গ্রাফ কে represent করা যায় তাতো শিখেই ফেলেছ! সুতরাং চিন্তা কি! আমাদের পরবর্তী দুইটি সেকশনের জন্য ধরে নেই যে আমাদের প্রদত্ত গ্রাফে n টি vertex ও m টি edge আছে।

৬.২.১ Prim's Algorithm

যেহেতু আমাদের সবগুলি vertex কে connected করতে হবে সুতরাং আমরা যেকোনো vertex থেকে শুরু করতে পারি। এখন আমরা দেখব, এই vertex এর সাথে যেই যেই edge আছে তাদের মাঝে কার weight সবচেয়ে কম। যার সবচেয়ে কম সেই edge আমরা নিব এবং তাহলে আমাদের এখন দুইটা vertex ও একটি edge হয়ে গেল। এখন দেখব, এই দুইটি vertex থেকে যেসব edge বের হয়েছে তাদের মাঝে কার weight সবচেয়ে কম তাকে নিব। এভাবে নিতে থাকব যতক্ষণ না আমাদের সব vertex নেয়া হয়ে যায়। এটা আশা করি বুঝছ যে যখন এই সবচেয়ে কম weight এর edge নিচ্ছ তখন সেই edge এর এক মাথা তোমার ইতমধ্যে বানানো tree এর ভিতরে যেন থাকে এবং অপর মাথায় যেন আমরা এখনও নির্বাচন করি নাই এরকম vertex থাকে। দুই মাথাই যদি আমাদের tree এর মাঝে থাকে তাহলে কিন্তু লাভ নেই! কারন তারা তো ইতোমধ্যেই connected, শুধু শুধু এই edge নিয়ে weight এর যোগফল বাড়ানোর কি কোন মানে আছে?

একটা উদাহরণ দেয়া যাক। মনে করো চিত্র ৬.১ এ আমরা 1 নোড হতে prim এর algorithm শুরু করেছি। তাহলে প্রথমে আমরা $1-5$ edge নির্বাচন করব, এরপর $1-2$, এরপর $2-3$. কেমন করে আমরা এই edge নির্বাচন করছি? চিত্রের এই state থেকে আমরা তা দেখি। চিত্রে নীল নোড গুলি হল ইতোমধ্যেই নির্বাচিত এবং কালোগুলি এখনও নির্বাচন করা হয় নাই। এখন সেসব edge দেখো যাদের এক মাথা নীল নোডে এবং অপর মাথা কালো নোডে। এরকম edge গুলি হল $3-6$, $3-4$ এবং $5-4$. এদের মাঝে সবচেয়ে কম weight এর edge হল $3-6$. সুতরাং আমাদের পরের নির্বাচিত নোড হবে 6.

এখন কথা হল এই algorithm এর time complexity কত! প্রথমত তুমি n বার এই নতুন vertex নির্বাচন করার কাজ করছ। এবং প্রতিবার হয়তো তুমি সব edge দেখছ। সুতরাং তোমার complexity দাঁড়ায় $O(nm)$. একে তুমি খুব সহজেই $O(n^2)$ করতে পার। মনে কর তুমি প্রথমে a নামক vertex নিয়েছিলে এবং আমাদের বর্তমান process এ প্রতিবার a এর সাথে লাগান সব edge প্রতি বার চেক করছ। কিন্তু প্রতিবার চেক করার কি দরকার আছে? তুমি যখন একটা নতুন vertex আমাদের tree তে অন্তর্ভুক্ত করবে তখন এর সাথে লাগান সব edge দেখবে, দেখে সেই



চিত্র ৬.১: Prim's algorithm

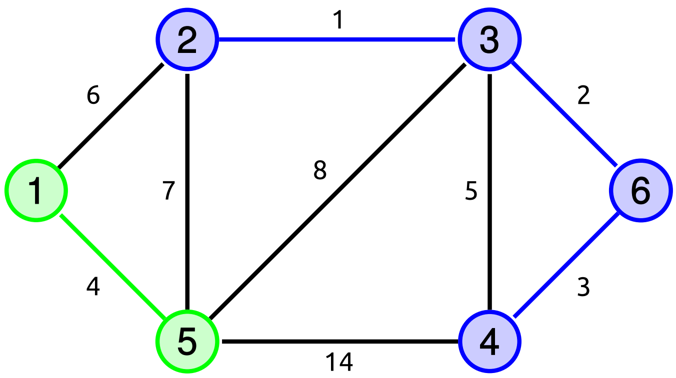
edge এর অপর প্রান্ত কত কম খরচে আমাদের tree তে নেয়া যায় সেটা update করবে। অর্থাৎ আমাদের প্রতিটি নোডে একটি করে মান থাকবে যা নির্দেশ করবে কত কম খরচে সেই নোড আমাদের tree তে অন্তর্ভুক্ত করা যায়। চিত্র ৬.১ এ খেয়াল করো, এই অবস্থায় নোড 4 এ থাকা মান হবে 9 এবং নোড 6 এ থাকা মান হবে 2. সুতরাং আমরা নির্বাচন করব নোড 6. এই নোড নির্বাচন করার পর আমরা এর সাথে লাগানো সব edge দেখব এবং অপর মাথা দরকারে আপডেট করব। নোড 6 এর সাথে লাগানো একটি edge হল $6-4$ এবং এর weight হল 3. সুতরাং আমরা নোড 4 এর আগের cost 9 আপডেট করে 3 করে দেব। একটু অন্য ভাবে বলি। মনে কর, আমরা নোড 3 যখন নিয়েছি তখন দেখেছি যে নোড 4 কে আমরা 9 cost এ অন্তর্ভুক্ত করতে পারি, কিন্তু নোড 6 অন্তর্ভুক্ত হয়ে যাবার পর দেখলাম যে নোড 4 কে আরও কম খরচে তুমি অন্তর্ভুক্ত করতে পারবে! তখন তুমি নোড 4 এর cost কে update করবে। এটা গেল ভিতরের কাজ, আমাদের কিন্তু বাহিরে একটা লুপ n বার চলছে যেটি প্রতিবার একটি একটি করে নোড নির্বাচন করছে। এই নোড কেমনে নির্বাচন করা হচ্ছে তা মোটামোটি বলেছি তাও আরেকবার বলি, প্রতিবার আমাদের দেখতে হবে যে কোন কোন vertex এখনও নির্বাচন করা হয় নাই এবং তাদের মাঝ হতে সবচেয়ে কম খরচের vertex কে তোমাকে নির্বাচন করতে হবে। তাহলে কি দাঁড়ালো? তুমি মোট n বার কাজ করবে, প্রতিবার সব অনির্বাচিত vertex যাচাই করে দেখবে যে কোনটি সবচেয়ে কম, তাকে নিবে। এর পর এর সাথে লাগান সব edge বরাবর গিয়ে দেখবে যে তার অপর প্রান্তে কোন অনির্বাচিত vertex আছে কিনা, থাকলে তার cost কে update করবে। তাহলে আমাদের complexity কত? n বার কাজ করছি আর প্রতিবার n টা vertex আমরা check করে দেখছি আর আমরা সর্বমোট খুব জোর $2m$ বার edge চেক করছি, সুতরাং আমাদের complexity দাঁড়ায় $\mathrm{O}(n^2+m)$ যা আসলে $\mathrm{O}(n^2)$ বলা যায় কারন $m < n^2$ তাই না?

তোমরা কিন্তু চাইলে একে আরও improve করতে পারবে। আমরা যে প্রতিবার n টা vertex ঘুরে ঘুরে দেখছি কোনটার cost সবচেয়ে কম তা না করে তুমি যদি একটা Min-Heap রাখ (C++ এ priority queue বা set) তাহলে তোমার এই algorithm আসলে $\mathrm{O}(m\log n)$ এ কাজ করবে। বুঝতেই পারছ এই পদ্ধতি তখনই ভাল কাজ করবে যখন m তোমার n^2 এর থেকে বেশ ছোট হবে। আরও ভাল heap ব্যবহার করে এর complexity আরও কমান যায়। তোমরা যদি interested হউ একটু internet বা বই ঘেটে ঘুটে দেখতে পার।

৬.২.২ Kruskal's Algorithm

এটিও বেশ সহজ algorithm. কোন কারনে যখনি MST আমাকে কোড করতে হয় আমি Kruskal's algorithm ই implement করে থাকি। হয়তো আমার কাছে এটি সহজ লাগে সেজন্য! এই algorithm বুঝানো খুবই সহজ, কোড করাও অনেক সহজ কিন্তু যেভাবে কোড করতে হবে সেটা বুঝানো একটু কষ্টকর। এই algorithm এ তুমি যা করবা তাহল সবচেয়ে কম weight এর edge নিবা, দেখবা এর দুই মাথার vertex দুইটি ইতোমধ্যেই একই tree বা component এ আছে কিনা, থাকলে এই edge

নিবা না। না থাকলে নিবা। শেষ! এখন প্রশ্ন হচ্ছে কেমনে বুঝবা যে দুইটি vertex একই tree তে আছে কিনা! উত্তরঃ Disjoint Set Union. প্রথমে সকল vertex কে আলাদা আলাদা set আকারে কল্পনা করো। আমরা যখনি একটি edge নিচ্ছি তখন দুইটি set কে জোড়া লাগানর চেষ্টা করছি। এবং সেজন্য চেক করছি যে, এই দুইটি vertex একই set এ আছে কিনা!



চিত্র ৬.২: Kruskal's algorithm

একটি উদাহরণ দেখা যাক। চিত্র ৬.২ এ আমরা প্রথমে $2-3$ কে জোড়া দিয়েছি। এরপর $3-6$, $6-4$, $1-5$. আমাদের পরের edge হল $3-4$ কিন্তু এই দুইটি নোডই একই tree তে আছে সুতরাং আমরা আর এই edge জোড়া লাগাব না। এর পরের edge হল $1-2$ এবং এটি দুইটি আলাদা tree কে জোড়া লাগায় সুতরাং আমরা এই edge নিবো এবং tree দুইটিকে জোড়া লাগাব। আশা করি বুঝতে পারছ যে আমরা edge এর cost এর increasing order এ edge গুলিকে consider করছি।

এখন প্রশ্ন হল এই algorithm এর time complexity কত? সহজ, edge গুলিকে weight অনুযায়ী sort করতে $O(m \log m)$ এবং প্রতি edge এর জন্য আমরা find করছি বা দুইটি set কে union করছি যাদের complexity আমরা $O(1)$ ধরে নিতে পারি। সুতরাং $O(m \log m + m)$ = $O(m \log m)$.

খেয়াল কর আমরা কিন্তু এই দুই algorithm এই greedily সবচেয়ে কম খরচের vertex বা edge নির্বাচন করে পুরো প্রব্লেম সমাধান করে ফেলেছি। তোমাদের যদি এই algorithm দুটির কোনটিতে সন্দেহ হয় তাহলে প্রমান করে দেখতে পার কেন এই greedy ভাবে edge নির্বাচন করলে সঠিক উত্তর দিবে।

৬.৩ ওয়াশিং মেশিন ও ড্রায়ার

মনে কর তুমি একটি কাপড় কাঁচার কোম্পানি চালাও। তোমার কাছে কিছু সেট কাপড় আছে এবং সেই সাথে একটি ওয়াশিং মেশিন ও একটি ড্রায়ার আছে। তুমি প্রতিটি সেট কে প্রথমে ওয়াশিং মেশিনে দিবে এবং এর পর ড্রায়ার এ দিবে। তুমি কিন্তু প্রথমে ড্রায়ার পরে ওয়াশিং মেশিনে দিতে পারবে না। এখন প্রতি সেট এর জন্য তোমার জানা আছে যে সেটি ওয়াশিং মেশিন এ কত সময় নিবে এবং ড্রায়ার এ কত সময় নিবে। অবশ্যই একই সাথে কয়েক সেট কাপড় তুমি একই মেশিনে দিতে পারবে না কিন্তু একই সাথে দুই সেট কাপড় আলাদা ভাবে দুই মেশিন এ দিতে পারবে। তুমি কত কম সময়ে সব কাপড় পরিস্কার করে ফেলতে পারবে?

সমস্যাটা যত না সুন্দর এর সমাধান তার থেকে বেশি সুন্দর। মনে কর i তম সেটের জন্য a_i হল ওয়াশিং মেশিন এ দরকারি সময় আর b_i হল ড্রায়ার এ দরকারি সময়। এখন মনে কর optimal order এ দুইটি পাশাপাশি সেট হল i আর j. অর্থাৎ তুমি চাইতেছ যে ঠিক i কাজ এর পর পর j কাজ করবা তাহলে এই দুইটি কাজ করতে তোমার কত সময় লাগবে? $a_i + max(b_i, a_j) + b_j$। আর যদি j কাজ

আগে করতে তাহলে তোমার সময় লাগত $a_j + max(b_j, a_i) + b_i$। অবশ্যই $a_i + max(b_i, a_j) + b_j \le a_j + max(b_j, a_i) + b_i$. এখন মনে কর k হল আরেক সেট কাজ এবং $a_j + max(b_j, a_k) + b_k \le a_k + max(b_k, a_j) + b_j$ অর্থাৎ k সেট j সেট এর পর করা ভাল। এই দুইটি ইকুয়েশন যদি সত্যি হয় তাহলে প্রমান করা যায় যে $a_i + max(b_i, a_k) + b_k \le a_k + max(b_k, a_i) + b_i$ অর্থাৎ i কাজের পর k কাজ করা ভাল (এটি তোমরা proof by contradiction এর মাধ্যমে একটু খেটেখুটে করতে পার)। তাহলে কি দাঁড়ালো? আমরা কাজ গুলোকে আসলে একটা order এ সাজাতে পারি। তবে কোন কাজের আগে কোন কাজ আসবে সেটা নির্ণয় করার জন্য আমাদের উপরের ইকুয়েশন এর সাহায্য নিয়ে দেখতে হবে যে কোন সেট আগে করলে কম সময় নেয়। এভাবে কাজ গুলোকে সাজালে আমরা optimal order পাবো।

এই সমাধান বের করা মোটেও সহজ নয়, সুতরাং তোমরা যদি একবার পড়ে এই সমাধান না বুঝো তাহলে আরও কয়েকবার পড়ে দেখ। একটু দুই একটা উদাহরন হাতে হাতে করে দেখ।

৬.৪ Huffman Coding

Huffman coding জানার আগে তোমাদের জানতে হবে prefix free coding কি জিনিস। Coding হল alphabet এর বিভিন্ন character কে অন্য কিছু দ্বারা প্রকাশ করা। আমাদের এই প্রবলেমে আমরা english alphabet এর কিছু character কে 0 আর 1 ব্যবহার করে এক একটি সংখ্যা দ্বারা প্রকাশ করব। যেমন আমরা হয়তো a কে প্রকাশ করব 001 দ্বারা, b কে প্রকাশ করব 110 দিয়ে ইত্যাদি। Prefix free coding হল কোন কোড ই অপর কোড এর prefix হতে পারবে না। Prefix মানে হল শুরুর অংশ। যেমন 01 হল 0110 এর একটি prefix. সুতরাং এই দুইটি একই সাথে code হতে পারবে না। মজার ব্যাপার হল এরকম 0 আর 1 দ্বারা প্রকাশিত যেকোনো coding তুমি চাইলে একটি binary tree দিয়ে প্রকাশ করতে পারো। মনে করো কোন নোড থেকে বাম দিকে যেই edge যায় তার label হল 0 আর ডান দিকে যেই edge যায় তার label হল 1. তাহলে root থেকে শুরু করে 0 আর 1 অনুসারে ডানে বামে যাও এবং কোড এর শেষ মাথায় এলে সেই নোডকে মার্ক করে ফেল। এটিই হল আমাদের coding এর binary tree তে representation. তাহলে একটু চিন্তা করে দেখো তো prefix free coding এর representation কেমন হবে? আগের মতই হবে শুধু মাত্র একটি অতিরিক্ত বৈশিষ্ট্য থাকবে আর তাহলো মার্ক করা নোড গুলি কেউ কার ancestor বা predecessor হবে না। কোন code যদি prefix free হয় তাহলে কি সুবিধা? সুবিধা হল তুমি কোন space ছাড়াই তাদের decode করতে পারবা। একটা উদাহরণ দেয়া যাক, মনে করো $a = 01, b = 0, c = 1$. এখানে কোড কিন্তু prefix free না। এখন যদি তোমাদের 01 দেয় তাহলে decode করলে কি হবে? bc নাকি a? দুইটিই হতে পারে। কিন্তু যদি prefix free হয় তাহলে কিন্তু কোনই মাথা ব্যাথা নাই, একে একভাবেই decode করা যায় আর decode করাও খুব সহজ, তুমি tree এর root থেকে traverse করতে থাকবে যতক্ষণ না কোন মার্ক করা নোড এ না পৌঁছাও। এরকম কোন নোড পেলে তুমি সেই character লিখে রেখে আবার root থেকে traverse করা শুরু করবে। এভাবেই তুমি decode করতে পারবে।

যাই হোক এখন মূল সমস্যায় আসা যাক। তোমাকে কিছু character দেয়া থাকবে এবং সেই character গুলি কত বার করে একটি text এ আসবে তা বলা আছে। তোমাকে এই text এর এমন একটি prefix free coding বের করতে হবে যেন পুরো text এর encoded দৈর্ঘ্য সবচেয়ে কম হয়। একটি উদাহরণ দেয়া যাক, মনে করো তোমাকে 3 টি character এর frequency দেয়া আছেঃ $(a, 10), (b, 4), (c, 8)$, ধরা যাক আমরা এদের কোড করলাম এভাবেঃ $(a, 01), (b, 1), (c, 00)$ তাহলে encoded text এর দৈর্ঘ্য হবে $10 \times 2 + 4 \times 1 + 8 \times 2 = 40$. এর থেকেও যে ভাল করা সম্ভব সেটা তোমরা বুঝতেই পারছ। এখন কথা হল এর সমাধান কেমনে করবে? সমাধান খুব সহজ। তুমি প্রতিটি character কে একটি নোড হিসাবে কল্পনা করো আর কোন character এর frequency হল সেই নোড এর cost। এর পর সবচেয়ে কম cost এর দুইটি নোড নাও তাদের merge করে ফেল। merge করার মানে হল তুমি নতুন একটি নোড নিবে, এদের দুজন কে দুইটি child হিসাবে দিবে আর নতুন নোড এর cost হবে সেই দুইটি নোড এর cost এর যোগফল। আগের লিস্ট এ সেই দুইটি নোড আর থাকবে না বরং এই merge কৃত নোড থাকবে। এভাবে যতক্ষণ না মাত্র একটি নোড থাকে ততক্ষণ এই কাজ করতে থাকো। তাহলেই তোমার ট্রি তৈরি হয়ে যাবে এবং সেই সাথে প্রতিটি

character এর code ও। এই algorithm একটি heap বা priority queue বা set ব্যবহার করে খুব সহজেই $O(n \log n)$ এ করে ফেলা যায়।

৭.২.৫ Variant 5

আমরা variant 2 তে $1 + 2 + 1$ এবং $2 + 1 + 1$ কে আলাদা ভেবেছিলাম কিন্তু যদি আলাদা না হয়? ধরা যাক $way[n][i]$ হল প্রথম i টি কয়েন ব্যবহার করে n কে কত ভাবে বানানো যায়। এখন n বানানোর পথে আমরা প্রথমে $coin[i]$ ব্যবহার করতেও পারি নাও পারি। যদি ব্যবহার করি তাহলে মোট $way[n - coin[i]][i]$ উপায় আর যদি না করি তাহলে $way[n][i-1]$ উপায়। শেষ :)

৭.৩ Travelling Salesman Problem

মনে কর তুমি একদিন রাজশাহী বেড়াতে গেলে। সেখানে তোমার n জন বন্ধুর বাড়ি। তুমি একে একে তাদের সবার বাড়ি যেতে চাও। তাদের সবার বাড়ির দুরত্ব তুমি জানো। তুমি প্রথমে গিয়ে তোমার সবচেয়ে ভাল বন্ধু 1 এর বাসায় যাবে এর পর একে একে সবার বাসা ঘুরে আবারও 1 এর বাসায় ফেরত আসবে। সবচেয়ে কম মোট কত দুরত্ব অতিক্রম করে তুমি সবার বাসা ঘুরতে পারবে? এটি হল Travelling Salesman Problem. আমরা এতক্ষণ একটি প্রব্লেমকে DP ভাবে সমাধান করার জন্য যা করেছি তাহল বড় একটি সমস্যাকে ছোট সমস্যা দ্বারা সমাধান করেছি। আরেকটি উপায় হল একই রকম জিনিস খুজে বের করা। যেমন আমাদের এই সমস্যার ক্ষেত্রে খেয়াল কর, তুমি মনে কর $1 - 2 - 3 - 4$ এই ভাবে চার জন বন্ধুর বাসা ঘুরেছ বাকি আছে $5 \ldots n$ বন্ধুরা। এই বাকি বন্ধুদের বাসা ঘুরতে তোমার যেই সবচেয়ে কম খরচ সেটা $1 - 3 - 2 - 4$ ঘুরার পর বাকি বন্ধুদের বাসা ঘুরে ফেলার জন্য সবচেয়ে কম খরচের সমান। অর্থাৎ, কোন এক সময় তোমাকে শুধু জানতে হবে তুমি কোন কোন বন্ধুর বাসা ঘুরে ফেলেছ এবং এখন তুমি কই আছ। বিভিন্ন ভাবে আমরা একই state এ আসতে পারি যেমন উপরের উদাহরনে আমরা প্রথম চার জন বন্ধুর বাসা দুই ভাবে ঘুরে এখন 4 এর বাসায় আছি। অর্থাৎ তোমার state হল তুমি কার কার বাসা ঘুরে ফেলেছ$(1, 2, 3, 4)$ আর এখন কই আছ(4)। এখন কই আছি সেটা শুধু একটা নাম্বার, কিন্তু তুমি কই কই ঘুরে ফেলেছ এই জিনিস অনেক গুলি নাম্বারের সেট। আমরা DP এর সময় state কে array এর parameter হিসাবে লিখি। এই ক্ষেত্রে আমরা একটি সেট কে কেমনে নাম্বার আকারে লিখতে পারি? খেয়াল কর, আমাদের মোট n জন বন্ধু আছে, কার কার বাসায় গিয়েছি তাদের 1 আর কার কার বাসায় এখনও যাওয়া হয় নাই তাদের 0 দ্বারা লিখতে পারি। তাহলে n টা $0 - 1$ দ্বারা আমরা কার কার বাসায় গিয়েছি সেটা বানিয়ে ফেলতে পারি। কিন্তু তুমি যদি array এর dimension n নিতে চাও তাহলে নিশ্চয় কোড করা খুব একটা সুখকর হবে না? এখানে একটা মজার tricks আছে তাহল তুমি এই $0 - 1$ সংখ্যাকে binary ফর্ম এ ভাবতে পার। যেমনঃ তোমার যদি $1, 2, 4$ নাম্বার বন্ধুর বাসা ঘুরা হয়ে থাকে তাহলে তোমার নাম্বার হবেঃ $0000 \ldots 1011 = 7$. এখন এই সংখ্যা কত বড় হতে পারে? 2^n কারন একটি বন্ধু থাকতে পারে নাও পারে। তাহলে আমাদের state কত বড়? $n \times 2^n$ এবং প্রতি state এ গিয়ে তুমি অন্যান্য সবার বাসায় যাবার চেষ্টা করবে (n ভাবে)। সুতরাং আমাদের time complexity হবে $O(n^2 2^n)$. কোড ৭.৬ এ দেয়া হল।

কোড ৭.৬: tsp.cpp

```
int dp[1<<20][20]; //Assume that there are 20 friends
//mask = friends i visited, at = last visited friend
int DP(int mask, int at)
{
    int& ret = dp[mask][at];
    //Assume that we initialized dp with -1
    if(ret != -1) return ret;

    ret = 1000000000; //initialize ret with infinity
    //dist contains distance between every two nodes
    for(int i = 0; i < n; i++) //n = number of friend
        if(!(mask & (1<<i)))
```

```
            ret = MIN(ret, DP(mask | (1<<i), i) + dist[at][i]);

    return ret;
}
```

৭.৪ Longest Increasing Subsequence

সংখ্যার একটি sequence আছে। এই sequence থেকে কিছু সংখ্যা (হয়তো একটিও না) মুছে ফেলতে হবে যেন বাকি সংখ্যা গুলি increasing order এ থাকে। আমাদের লক্ষ্য হল সবচেয়ে দীর্ঘ increasing subsequence[১] বানানো। আমরা যদি এটি DP এর মাধ্যমে সমাধান করতে চাই তাহলে ভাবতে হবে আমরা কোন ছোট সমস্যা সমাধান করতে পারি। ধরা যাক আমাদের কে n টি সংখ্যা দেয়া আছে। এর LIS (Longest Increasing Subsequence) বের করতে হবে। আমরা যদি প্রথম $n-1$ টির LIS জানি তাহলে কি কোন লাভ আছে? চিন্তা করে দেখা যাক। আমরা nতম সংখ্যা কে কার পিছে বসাব? এমন একটি সংখ্যার পিছে যা nতম সংখ্যার থেকে ছোট, ধরা যাক $a[i]$ (a হল মূল sequence এবং $i < n$). এখন এরকম তো অনেক $a[i]$ আছে যেন $a[i] < a[n]$ কিন্তু কোনটির পিছনে? যে সবচেয়ে ছোট তার পিছনে? না, কারনঃ $1, 2, 3, 4$ এদের মাঝে কার পিছনে আমরা 5 কে বসাতে চাইব? নিশ্চয় 1 না। আমরা 4 এর পিছনে বসাতে চাইব কারন প্রথমত 5 এর থেকে 4 ছোট আর দ্বিতীয়ত এই 4 পর্যন্তই সবচেয়ে বড় LIS আছে। সুতরাং n তম সংখ্যাকে নিয়ে প্রথম n টি সংখ্যার LIS ($LIS[n]$) হল, $LIS[i]+1$ এর মাঝে সবচেয়ে বড় মান যেখানে $a[i] < a[n]$। এই পদ্ধতির time complexity $O(n^2)$. আমরা খুব সহজেই Segment Tree ব্যবহার করে এটিকে $O(n \log n)$ করতে পারি। আমরা n এ এসে $1 \ldots a[n]-1$ পর্যন্ত query করব আর শেষে $a[n]$ এ আপডেট করব।

তবে সাধারণত আমরা অন্য আরেকভাবে $O(n \log n)$ এ LIS বের করে থাকি। মনে করা যাক আমাদের ইনপুট এর অ্যারে হল a আর আমাদের কাছে একটা auxiliary অ্যারে আছে তা ধরা যাক b. প্রথমে b ফাঁকা থাকবে। এখন আমরা যা করব তাহলো a এর শুরু থেকে শেষ পর্যন্ত যাব আর এই সংখ্যাগুলি নিয়ে কিছু একটা কাজ করব। কাজটা সংক্ষেপে বললে বলতে হয় যে b তে আমাদের বর্তমান সংখ্যা $a[i]$ কে এমন জায়গায় বসাব যেন b sorted থাকে। যদি আমরা $a[i]$ নিয়ে দেখি আমাদের b ফাঁকা (এটি কেবল মাত্র প্রথম element এর ক্ষেত্রে ঘটতে পারে) তাহলে তো এই $a[i]$ কে b তে ঢুকিয়ে দিব। আর যদি তা না হয় তাহলে আমরা b এর ভিতরে সবচেয়ে ছোট এমন একটি সংখ্যা খুঁজে বের করব যেন সেই সংখ্যাটি আমাদের সংখ্যার থেকে বড় হয়। আর যদি সেরকম কোন সংখ্যা না পাওয়া যায় তাহলে b এর শেষে। কিছুটা insertion sort এর মতও চিন্তা করতে পারো তবে মূল পার্থক্য হল এই নতুন $a[i]$ কিন্তু insert হবে না, বরং replace হবে। কিছু উদাহরণ দেয়া যাক।

যদি $a[i] = 10$ হয় এবং $b = 2, 4, 8, 12, 15$ হয় তাহলে 10 কে বসাতে হবে 12 তে, কারণ এটি সবচেয়ে ছোট সংখ্যা যা 10 এর থেকে বড়। সুতরাং আমাদের b পরিবর্তন হয়ে, হয়ে যাবে $2, 4, 8, 10, 15$. এখন মনে করো $a[i] = 18$. তাহলে কই বসাবে? উত্তর সহজ 15 এর পরে অর্থাৎ $2, 4, 8, 10, 15, 18$. এখন যদি $a[i]$ হয় 1 তাহলে b পরিবর্তন হয়ে দাঁড়াবে $1, 4, 8, 10, 15, 18$. এভাবে আমাদের b এর অ্যারে পরিবর্তন হতে থাকবে। সকল a consider হয়ে গেলে b এর length ই হবে আমাদের LIS এর length. কিন্তু এটা ভেবে বসো না যে b হবে LIS. যেমন $a = 2, 3, 1$ হলে b হবে $1, 3$. কিন্তু $1, 3$ কিন্তু LIS না, তাহলে কি করতে হবে যদি আমরা পুরো sequence বের করতে চাই? খুব সহজ, যখন কোন একটি সংখ্যা বসাবে তখন তার immediate আগের সংখ্যা কত তা লিখে রাখবে। আসলে সেই সংখ্যাটা লিখে রাখলেই যে সবসময় হবে তা না। তোমরা equal নাম্বার এর ক্ষেত্রে কি হবে তা একটু চিন্তা করে দেখতে পারো। আবার অনেক সময় প্রবলেম ভেদে খেয়াল রাখতে হয় যে strictly increasing চেয়েছে নাকি non decreasing চেয়েছে। Strictly increasing হল $1, 5, 6, 7$ এরকম আর non decreasing হল $1, 2, 2, 3, 3, 3, 4, 5, 5$ এরকম। তবে সমাধানের মূল idea একই থাকবে। শুধু একটু টুকটাক পরিবর্তন করতে হবে।

[১]একটি sequence থেকে কিছু সংখ্যা মুছে ফেললে যা বাকি থাকে তাই subsequence. এখানে কিন্তু বাকি সংখ্যা গুলির order একই থাকতে হবে।

এখন কথা হল এই algorithm এর time complexity কত? প্রথমত আমাদের n বার কাজ করতে হচ্ছে। যদি আমরা লুপ চালাই b এর অ্যারে তে তাহলে প্রতিবার n সময় লাগবে। কিন্তু আমরা যদি লুপ না চালিয়ে binary search করি তাহলেই কিন্তু আমরা এই কাজ $O(\log n)$ সময়ে করতে পারি। এবং এভাবে আমাদের runtime হবে $O(n \log n)$. এখন একটা প্রশ্ন করি, আমরা insertion sort এ তাহলে binary search চালিয়ে runtime কমালাম না কেন? কারণ হল, insertion sort এ আমাদের শুধু সঠিক জায়গা খুঁজে বের করলেই চলবে না তাকে insert করতে হবে। replace করতে কিন্তু $O(1)$ কাজের প্রয়োজন হয় কিন্তু insert করতে worst case এ $O(n)$ পরিমাণ কাজ করতে হতে পারে। আবার তোমরা যদি মনে করো linked list ব্যবহার করে insert তুমি $O(1)$ সময়ে করতে পারো, তাহলে সমস্যা হল linked list এ তুমি binary search করতে পারবা না। কারণ binary search করতে হলে একটি নির্দিষ্ট index এ যাওয়ার দরকার হয় যা linked list এ সম্ভব না।

৭.৫ Longest Common Subsequence

দুইটি string: S এবং T দেয়া থাকবে, আমাদের এমন একটি string বের করতে হবে যা S এবং T উভয়েরই subsequence হয় এবং longest হয়। এই প্রব্লেম এর ক্ষেত্রে আমাদের state হবেঃ যদি আমাদের S এবং T সম্পূর্ণ ভাবে না দিয়ে S এর প্রথম s টি এবং T এর প্রথম t টি letter দেয়া হয় তাহলে Longest Common Subsequence (LCS) কত? যদি S[s] = T[t] হয় (1 indexing ধরে) তাহলে কিন্তু আমাদের উত্তর হল S এর প্রথম $s-1$ এবং T এর $t-1$ এর যত উত্তর তার থেকে এক বেশি। আর যদি $S[s] \neq T[t]$ হয় তাহলে S এর প্রথম $s-1$ ও T এর প্রথম t letter এর ক্ষেত্রে উত্তর আর S এর প্রথম s ও T এর $t-1$ letter এর ক্ষেত্রে উত্তর এর মাঝে যেটি বড় সেটি। এখন যদি আমাদের শুধু উত্তরের দৈর্ঘ্য নয় সেরকম একটি string ও প্রিন্ট করতে বলে তাহলে আমরা প্রথমে dp table অর্থাৎ উত্তরের table বানিয়ে নেব। এর পর দুইটি string এরই শেষ থেকে আসতে হবে। যদি শেষ character দুইটি একই হয় তাহলে আমরা সেটি নেবই। আর না হলে আমরা dp টেবিল থেকে দেখব যে কোন string থেকে শেষ character বাদ দেওয়া উচিত। খেয়াল করে দেখ, এভাবে করলে সমস্যা হল আমরা string টা উলটো দিক থেকে তৈরি করতেসি। সুতরাং যেহেতু আমাদের string কে সামনের দিক থেকে প্রিন্ট করতে হবে সেহেতু হয় আমাদের কোন একটি স্ট্রিং এ character গুলি নিয়ে পরে reverse করে প্রিন্ট করতে হবে অথবা এই পুরো কাজটা recursively করতে হবে। আরেকটা বুদ্ধি হল আমরা যদি সামনের দিক থেকে dp না করে পিছন থেকে dp করি তাহলে আর string উলটো করার ঝামেলা থাকে না। সাধারন একটা while লুপ দিয়েই আমরা পুরো স্ট্রিং প্রিন্ট করে ফেলতে পারি। খেয়াল কর, আমি এখানে অনেক উপায়ে স্ট্রিং প্রিন্ট [১] করার পদ্ধতি বললাম, একেক সমস্যার ক্ষেত্রে একেক উপায়ে path print করা সহজ হয়।

৭.৬ Matrix Chain Multiplication

যেকোনো দুইটি সংখ্যা আমরা চাইলেই গুন করতে পারি। কিন্তু দুইটি matrix কিন্তু চাইলেই গুন করা যায় না। আমরা $A(p \times q)$ এবং $B(r \times s)$ সাইজের দুইটি matrix গুন করতে পারব যদি $q = r$ হয়। অর্থাৎ A এর কলাম সংখ্যা যদি B এর রো সংখ্যার সমান হয় তাহলেই আমরা $A \times B$ করতে পারব ($B \times A$ আর $A \times B$ কিন্তু matrix এর ক্ষেত্রে আলাদা কথা)। এখন মনে করো আমাদের অনেক গুলি matrix পর পর আছেঃ $A_1, A_2, \ldots A_n$. এদের পর পর গুন করা যাবে যদি এদের dimension এরকম হয়ঃ $A_1(p_1 \times p_2), A_2(p_2 \times p_3), A_3(p_3 \times p_4) \ldots A_n(p_n \times p_{n+1})$. দুইটি matrix $A(p \times q)$ এবং $B(q \times r)$ গুন করার cost হল $p \times q \times r$. কেন? কারণ আমাদের এই দুইটি matrix গুন করতে চাইলে তিনটি লুপ p, q এবং r পর্যন্ত চালাতে হবে। এখন একটা জিনিস খেয়াল

[১]যেকোনো dp সমস্যায় শুধু মান না, মান টা কেমনে হয় সেটাও চেয়ে থাকে, একে আমরা path printing বলে থাকি।

করো $A_1 \times (A_2 \times A_3)$ এর cost আর $(A_1 \times A_2) \times A_3$ এর cost কিন্তু আলাদা হতে পারে[১]। একটা উদাহরণ দেখা যাক। মনে করো $A_1 = 2 \times 3$, $A_2 = 3 \times 5$ এবং $A_3 = 5 \times 4$. তাহলে $A_1 \times (A_2 \times A_3)$ এর cost হবে $3 \times 5 \times 4 + 2 \times 3 \times 4 = 60 + 24 = 84$ আর $(A_1 \times A_2) \times A_3$ এর cost হবে $2 \times 3 \times 5 + 2 \times 5 \times 4 = 30 + 40 = 70$. যদি n টি matrix থাকে তাহলে তাদের অনেক ভাবে গুন করা যায় (আরও নির্দিষ্ট ভাবে বললে এটি catalan নাম্বার এর সমান)। সুতরাং n এর বড় মান, ধরা যাক 100, এর জন্য তুমি সব ভাবে চেষ্টা করতে পারবে না। তাহলে উপায় কি? প্রথমত খেয়াল করো তুমি কিন্তু লাফ দিয়ে A_1 এর সাথে A_5 এর গুন দিতে পারবে না। তোমাকে সবসময় পরপর গুন করতে হবে। optimal গুন কেমন হবে তুমি একটু কল্পনা করো। তুমি পাশাপাশি দুইটি দুইটি করে গুন করছ যতক্ষণ না তোমার কাছে একটি matrix বাকি থাকে। তুমি শেষ গুণটা খেয়াল করো। শেষ গুণের সময় matrix দুইটা হবে কিছুটা এরকমঃ $(A_1 \times A_2 \times \ldots A_i) \times (A_{i+1} \times \ldots A_n)$. অর্থাৎ শুরুর দিকের কিছু matrix একত্রে "কোনভাবে" গুণ হবে, শেষের দিকের বাকি matrix কোনভাবে গুণ হবে এবং এরা দুইজন আবার গুণ হবে। এদের দুই জনের গুণের খরচ কিন্তু তুমি জানো কারণ তোমার প্রথম গুণফলের matrix এর dimension হবে $p_1 \times p_{i+1}$ এবং দ্বিতীয় গুণফলের matrix এর dimension হবে $p_{i+1} \times p_n$. সুতরাং এদের গুণের cost হবে $p_1 \times p_{i+1} \times p_n$. এখন যেটা জানিনা তাহলো এই দুইটি অংশের cost. এটাই কিন্তু DP. আমরা 1 হতে n পর্যন্ত গুণ করার cost বের করার জন্য ছোট দুইটি range এর cost জানতে চাচ্ছি। সুতরাং আমাদের একটি recursive ফাংশন থাকবে যাকে বলব আমাকে A_i হতে A_j পর্যন্ত সকল matrix এর গুণফলের cost বল। সে ভিতরে যা করবে তাহলে সে A_i হতে A_k এবং A_{k+1} হতে A_j পর্যন্ত গুণ করার cost কে recursively বের করবে। এর সাথে ঐ দুই অংশের গুণফলের matrix গুণ করার cost যোগ করবে। এভাবে প্রতি $i \le k < j$ এর জন্য আমাদের cost বের করতে হবে। এই সকল cost এর মাঝে যেটি সবচেয়ে কম সেটিই A_i হতে A_j পর্যন্ত optimally গুণ করার cost. প্রশ্ন হল base case কি? দুইভাবে চিন্তা করতে পারো। এক, তুমি ভেবে দেখো কত ছোট কাজ তুমি এমনিই করে ফেলতে পারবে। সহজ, তোমাকে যদি দুইটি matrix দেয় তাহলে তুমি জানো যে তুমি একভাবেই গুণ করতে পারবে আর গুণের cost এতো। দুই, তুমি এভাবেও চিন্তা করতে পারো যে কত পর্যন্ত ভাগ করা যায়। তোমাকে যদি একটি matrix দেয় অর্থাৎ $i = j$ যদি হয় তাহলে কিন্তু $i \le k < j$ এই সমীকরণ অনুসারে কোন k পাবে না অর্থাৎ ভাঙ্গা যাবে না। এখন তোমাকে একটা ম্যাট্রিক্স দিয়ে যদি বলে এদের গুণ করার cost কত? কি উত্তর হবে? 0, তাই না? কারণ এখানে গুণ করার কিছু নেই। এতো গেল base case. এখন চিন্তা করো এর time complexity কত? এজন্য দেখো তোমার DP এর parameter কয়টা? দুইটা, i এবং j অর্থাৎ $O(n^2)$ এর মত। i, j parameter এর জন্য তোমাকে $i \le k < j$ এর মাঝের একটি k এর লুপ চালাতে হবে। অর্থাৎ মোট $O(n^3)$. যেহেতু সরাসরি আমরা n পর্যন্ত লুপ চালাচ্ছি না তাই তোমরা ভাবতে পারো এটাতো n^3 নাও হতে পারে। যারা বিশ্বাস করছ না তারা একটু হিসাব করলেই দেখতে পারবা যে এটা আসলেই $O(n^3)$. আর হ্যা আশা করি memoization এর কথা ভুল নাই। Memoize না করলে কিন্তু তোমাদের algorithm আর $O(n^3)$ হবে না বরং এটা হয়ে যায় backtrack. অর্থাৎ অন্যভাবে বলা যায় backtrack এ state কে memoize করলেই DP হয়ে যায়। কিন্তু সমস্যা হল অনেক সময় এই state এতো বড় হয়ে যায় যে memoize করা অসম্ভব হয়ে যায়।

যাই হোক, এতক্ষণ আমি recursively উত্তর বের করার কথা বললাম। Iteratively ও কিন্তু এই সমাধান করা যাবে। তবে এতে i, j এর লুপ না চালিয়ে প্রথমে length এর লুপ l চালাতে হবে এর পর শুরুর মাথার লুপ i. তাহলে শেষ মাথা $j = i + l - 1$. এখন তুমি k এর লুপ চালাবে। কেন আমরা এভাবে করলাম? খেয়াল করো, তুমি যদি প্রথমে i, j এর লুপ চালাতে এবং এর ভিতরে k তাহলে তুমি $dp[i][k]$ এবং $dp[k+1][j]$ এর মান জানতে চাইবে। প্রথমটার মান ইতোমধ্যেই বের করে ফেলেছ কিন্তু i এখনও k পর্যন্ত যায় নাই! সুতরাং এভাবে করলে হবে না। Idea হল ছোট থেকে আশা। তুমি যদি ছোট এর উত্তর জানো তাহলেই বড় এর উত্তর বের করতে পারবে। এখানে ছোট বা বড় কিসের সাপেক্ষে? length. এ জন্যই আমরা আগে length এর লুপ চালিয়েছি।

[১]গুন করা যে যাবে সেটা নিয়ে কোন সন্দেহ নেই, কারণ তুমি A_i হতে A_j পর্যন্ত যেভাবেই গুন করো না কেন এর dimension হবে $p_i \times p_{j+1}$

৭.৭ Optimal Binary Search Tree

Binary search tree কি জিনিস তা তো তোমাদের ইতোমধ্যেই বলেছি। এটি এমন একটি ট্রি যার প্রতি নোড এ একটা করে সংখ্যা থাকে। তোমাকে কোন query দিলে সেই নাম্বার খুঁজে বের করতে হয়। আমরা সবসময় root থেকে শুরু করি এবং যদি দেখি আমাদের বর্তমানের নোড আমরা যেই সংখ্যা খুঁজছি তার সমান তাহলে তো হয়েই গেল, আর যদি তা নাহয় তাহলে দেখব এই নোড এর সংখ্যার থেকে আমাদের সংখ্যা ছোট নাকি বড়, ছোট হলে বামে যাব আর বড় হলে ডানে। এভাবে যতক্ষণ না খুঁজে পাচ্ছি ততক্ষণ এই কাজ চলতেই থাকবে। আমাদের এই সমস্যায় মনে করব সবসময় উত্তর খুঁজে পাওয়া যাবে মানে ট্রি তে যেসব সংখ্যা আছে শুধু তাদেরকেই query করা হবে। যাই হোক, এখন যদি আমাদের যে কোনো query আসা equi-probable হয় তাহলে আমাদের balanced binary search tree বানাতে হবে। কিন্তু যদি query গুলি সমসম্ভাব্য না হয়? যদি আমাদের বলা থাকে কোন query আসার সম্ভাবনা কত তাহলে? এখানে কিন্তু huffman tree এর টেকনিক খাটবে না কারণ আমাদের সংখ্যা গুলি অবশ্যই sorted আকারে থাকতে হবে tree তে[১]। এখন মনে করো তোমার কাছে n টি সংখ্যা আছে 1 হতে n পর্যন্ত আর i তম সংখ্যা query হবার সম্ভাবনা p_i. তাহলে কোন একটি ট্রি এর মোট cost বা expected cost হবে কোন সংখ্যার query হবার সম্ভাবনা আর সেই সংখ্যার ট্রি তে depth এর গুণফল সমূহের যোগফল। আসলে তুমি যদি অন্য algorithm এর বই দেখো বা নেট দেখো তাহলে দেখবে যে optimal binary search tree প্রবলেম এটা না। ওটা আরেকটু জটিল তবে মূল theme একই এবং সমাধানের ধরনও একই। যাই হোক, এখন তুমি চিন্তা করো এই n টি সংখ্যাকে নিয়ে কেমনে ট্রি বানাবে? অবশ্যই যদি একটি নোড বাকি থাকে তাহলে তাকে নিয়ে ট্রি বানানোর খরচ 0. কিন্তু যদি একাধিক থাকে তাহলে? তাহলে যেসব সংখ্যা বাকি আছে তাদের একটি হবে root, ধরা যাক i হতে j পর্যন্ত সংখ্যা বাকি আছে আর আমরা k কে root হিসাবে নির্বাচন করলাম যেখানে $i \leq k \leq j$. তাহলে এ জন্য আমাদের cost কত হবে? প্রথমত k এর জন্য cost 0. বাকি $[i, k-1]$ যাবে left এ আর $[k+1, j]$ যাবে right এ। এই configuration এ cost হবে $dp[i, k-1] + dp[k+1, j] + (p_i + p_{i+1} + \ldots + p_{k-1}) + (p_{k+1} + \ldots p_j)$. কারণ প্রথম দুইটি হল root এর দুই পাশের ট্রিতে cost আর বাকিটুকু হল এতো probablity তে আমাদের নিচে নামতে হতে পারে বা এতো probability তে আমাদের আরও 1 cost দিতে হতে পারে। এই সমাধান আমাদের matrix chain multiplication এর মত। এর time complexity ও যে $\mathrm{O}(n^3)$ তা নিয়ে কোন সন্দেহ নাই। তবে খুব ছোট একটা optimization করে আমরা একে $\mathrm{O}(n^2)$ করে ফেলতে পারি। প্রথমেই বলে নেই এই রকম optimization কিন্তু সব প্রবলেম এর ক্ষেত্রে খাটবে না। কিছু special properties থাকলেই কেবল হবে। তবে সেই special properties টা কি সেটা আমি নিজেও ভাল মত জানি না। শুধু জানি অন্তত এই প্রবলেমে এই optimization কাজ করবে। Optimization টা এখন বলি। মনে করো $[i, j]$ এর জন্য optimal k হল $P[i, j]$. তাহলে আমরা লিখতে পারি $P[i+1, j] \leq P[i, j] \leq P[i, j-1]$. অর্থাৎ তুমি যখন $[i, j]$ তে আসবে তখন k এর লুপ i হতে j না চালিয়ে $P[i+1, j]$ হতে $P[i, j-1]$ চালাবে। এই কাজ করলেই আমাদের runtime $\mathrm{O}(n^2)$ এ নেমে আসবে। কেন কেমনে এসবের উত্তর আমি নিজেও খুব ভাল মতো জানি না। সুতরাং যাদের আগ্রহ আছে তারা নেট এ ঘেটেঘুটে জেনে নিও।

[১] sorted বলতে কি বুঝাচ্ছি আশা করি বুঝা যাচ্ছে? কঠিন ভাবে বলতে গেলে বলতে হয় in-order traversal করলে sorted সংখ্যা পাবা

```
                }
            }
        }
}
```

৮.৩ DFS ও BFS এর কিছু সমস্যা

৮.৩.১ দুইটি node এর দুরত্ব

মনে কর একটি গ্রাফ আর দুইটি নোড দিয়ে বলা হল যে তাদের দুরত্ব বের কর। দুরত্ব বলতে কয়টি edge পার করে যেতে হয় সেটা বুঝানো হচ্ছে এখানে। কেমনে করবে? এটি কিন্তু BFS দিয়ে খুব সহজেই সমাধান করা যায়। খেয়াল করে দেখ, BFS এর ক্ষেত্রে কিন্তু শুরুর নোড থেকে যেই পথে অন্য একটি নোড এ যাওয়া হয় তা কিন্তু shortest path। সুতরাং যখন আমরা কোন নোড থেকে আরেকটি নতুন নোড এ যাব তখন আমরা বলতে পারি নতুন নোড এর দুরত্ব যেখান থেকে আসা হচ্ছে তার থেকে এক বেশি। সুতরাং আমাদের যেই দুইটি নোড দেয়া আছে তাদের একটি থেকে BFS শুরু করলে আমরা অন্য নোড এ যাবার দুরত্ব পেয়ে যাব।

খেয়াল কর আমরা কিন্তু এই সমস্যা DFS দিয়ে সমাধান করতে পারব না। কারন DFS দিয়ে সব-সময় shortest path এ যাওয়া হয় না। মনে কর A, B ও C তিনটি নোড। প্রত্যেকটি থেকে অন্য দুইটি নোড এ যাওয়া যায়। এখন A হতে DFS শুরু করলে ধরা যাক আমরা প্রথমে B তে যাব এর পর C তে যাব। খেয়াল কর, A থেকে C তে এক ধাপে যাওয়া গেলেও আমরা DFS করে গেলে দুই ধাপে যাচ্ছি।

এখন যদি আমাদের এই দুইটি নোডের মাঝে shortest path টাও প্রিন্ট করতে বলে? path printing টেকনিক মোটামোটি সব ক্ষেত্রেই একই রকম। তুমি কোন নোডে যাবার সময় লিখে রাখবে এখানে কেমনে এসেছ। যেমন BFS এর ক্ষেত্রে তুমি কোন নতুন নোডে আসার সময় লিখে রাখবে কোন নোড থেকে এখানে এসেছ। তাহলেই হবে।

৮.৩.২ তিনটি গ্লাস ও পানি

খুব কমন একটি পাজল হল, তোমাকে 3 ও 5 লিটারের দুইটি ফাঁকা গ্লাস আর 8 লিটারের একটি পানি ভর্তি গ্লাস দেয়া থাকলে তুমি 4 লিটার পানি আলাদা করতে পারবে কিনা। যদি শুধু প্রশ্ন হয় পারবে কিনা তাহলে DFS করেই করতে পারবে। আর যদি চায় সবচেয়ে কম কতবার ঢালাঢালি করে? তাহলে তোমাকে BFS করতে হবে। এটাতো বললাম যে এটা BFS দিয়ে সমাধান করা যাবে কিন্তু এখানে নোডই বা কই আর edge ই বা কই? আসলে BFS করতে যে vertex বা edge লাগবে এই ধারনা ঠিক না। আমাদের যা জানতে হবে তাহল আমরা কোথায় আছি আর এখান থেকে আমরা কোথায় কোথায় যেতে পারি। কিছুটা DP এর মত চিন্তা করতে পারো। আমরা যেখানে আছি সেটাকে একটা state আকারে represent করতে হবে- এটাই মূল জিনিস। যেমন আমাদের এই প্রবলেম এর state হতে পারে তিনটি পাত্রে কত খানি করে পানি আছে। সুতরাং আমরা 1-dimensional array তে visited না রেখে একটা 3-dimensional array তে visited রাখতে পারি আর queue তে একটি নাম্বার না রেখে তিনটি নাম্বার একত্রে structure করে রাখতে পারি (structure এর queue). এভাবে BFS করলেই এই সমস্যা সমাধান হয়ে যাবে।

এই সাবসেকশন শেষ করার আগে আরেকটি জিনিস, তাহল তোমরা চাইলে কিন্তু এই state কে দুইটি নাম্বার দিয়ে represent করতে পারো। প্রথম দুই পাত্রে কত খানি করে পানি আছে। কারন 8 থেকে ঐ পরিমান বাদ দিলে তুমি তৃতীয় পাত্রের পানির পরিমান পেয়ে যাবে। এই optimization এর ফলে তোমার run time এ কোন পরিবর্তন হবে না তবে memory কম লাগবে। তোমরা চাইলে queue তে তিনটি নাম্বার ই রাখতে পারো এতে করে কষ্ট করে 8 থেকে বিয়োগ করার কাজ করতে হবে না। আর সেই সাথে আমরা visited রাখার সময় প্রথম দুইটি সংখ্যা ব্যবহার করব যাতে আমাদের

memory কম লাগে। এই tricks প্রায়ই কাজে লাগে। যাতে বেশি computation এর দরকার না হয় সেজন্য আমরা queue তে সব জিনিসই রেখে দেব কিন্তু visited বা memoization এর জন্য যাতে কম জায়গা লাগে সেজন্য আমরা ছোট visited matrix ব্যবহার করব।

৮.৩.৩ UVa 10653

তোমাকে একটি গ্রিড দেয়া থাকবে। সেই সাথে তোমার শুরুর জায়গা আর শেষ গন্তব্য দেয়া থাকবে। তোমাকে সবচেয়ে কম কত সময়ে গন্তব্যে পৌঁছান যায় তা বলতে হবে। এটা খুব ভাল মতই বুঝা যাচ্ছে যে গ্রিড এর একেকটি সেল হল একেকটি নোড। তোমরা যারা প্রথম প্রথম প্রোগ্রামিং করতেছ তারা হয়তো প্রতি সেল কে $1, 2, \ldots RC$ এভাবে নাম্বার দিবে কিন্তু এর থেকে সুবিধা হবে তুমি যদি নোডকে (r, c) ভাবে represent কর। আর ৮.৪ এর মত দুইটি matrix রাখ।

কোড ৮.৪: cellbfs.cpp

```
int dr[] = {-1, 0, 1, 0};
int dc[] = {0, 1, 0, -1};

int valid(int r, int c)
{
    return r >= 0 && r < R && c >=0 && c < C;
    // also may be check if (r, c) is empty.
    // you may also check if the cell is visited by bfs.
}
```

তাহলে (r, c) থেকে নতুন যেসব cell এ যেতে পারবে সেসব হল $(r + dr[i], c + dc[i])$ (i এর একটি লুপ 0 হতে 3 পর্যন্ত চালাও) আর এই নতুন cell টি আদৌ valid কিনা তা জানার জন্য ৮.৪ কোড এর valid ফাংশনকে call করে দেখ।

৮.৩.৪ UVa 10651

এটি তুমি BFS বা DFS যেকোনোটি দিয়েই সমাধান করতে পারবে। কারন এখানে সর্বনিম্ন কয়টি peeble থাকবে অর্থাৎ কোন কোন game configuration এ যাওয়া যাবে সেটিই মূল জিনিস।

৮.৩.৫ 0 ও 1 cost এর গ্রাফ

ধরা যাক তোমাকে একটি weighted গ্রাফ দেয়া হল যার edge cost হয় 0 নাহয় 1. এই ক্ষেত্রে এক নোড থেকে আরেক নোড এ যাবার shortest path বের করার একটি সহজ উপায় হল BFS এর মত কাজ করা। তুমি যখনি 0 দিয়ে যেতে চাইবে তখন queue এর শুরুতে add করবা, আর 1 দিয়ে যেতে চাইলে queue এর শেষে। আর নেবার সময় সবসময় সামনে থেকে নিবা। আসলে দুই দিকে add করতে পারলে সেটা আর queue থাকে না, এর আরেক নাম হল deque. STL এ deque বলে built-in ডাটা স্ট্রাকচার আছে। এক্ষেত্রে deque এর সাইজ প্রায় $2n$ এর সমান হয়ে যেতে পারে। তবে এ জন্য তোমাদের একটি $dist$ এর অ্যারে নিয়ে তাতে দূরত্ব রাখতে হবে এবং তখনই তুমি deque এ insert করবে যখন তোমার এখনকার cost, $dist$ অ্যারেতে থাকা cost এর থেকে কম হয়।

৮.৪ Single Source Shortest Path

Shortest Path প্রবলেম হল, কোন একটা weighted graph এ এক নোড থেকে আরেক নোড এ যাবার সর্বনিম্ন cost বের করার প্রবলেম। সাধারনত আমরা দুই ধরনের Shortest Path প্রবলেম দেখে থাকি। Single source shortest path এবং All pair shortest path. Single source shortest path প্রবলেম এ আমরা এক নোড থেকে অন্য সকল নোড এ যাবার সর্বনিম্ন cost বের করে থাকি আর All Pair Shortest Path প্রবলেম এ আমাদের প্রতিটি নোড থেকে অন্য সকল নোড এ যাবার cost বের করতে হয়। তোমরা হয়তো ভাবতে পারো যে তাহলে Single Source Single Destination Shortest Path বলে আরও একটা কিছু বলছি না কেন? কারন হল, Single Source Shortest Path এর মাধ্যমে এই Single Destination এর variation টা সল্ভ করা যায় আর তাছাড়া মূল কারন হল, Single Destination এর variation সমাধান করার জন্য আসলে simpler কোন algorithm নেই। খেয়াল কর আমি কিন্তু simpler বলেছি। আমি এখানে Single Source Shortest Path এর সাথে তুলনা করছি। অর্থাৎ, আমরা Single Source Shortest Path এর জন্য যেসকল algorithm দেখব, Single Destination এর জন্য তার থেকেও efficient algorithm আসলে আমার জানা নেই। তবে হ্যা, হয়তো খুবই সামান্য optimization করতে পারবে কিন্তু আসলে worst case এ একই time complexity পাবে। এসব কথা এখন বুঝতে না পারলেও সমস্যা নেই, নিচের algorithm গুলি বুঝে এসে এই কথা গুলো পড়লে আসা করি বুঝতে পারবে আমি কোন optimization এর কথা বলছি, বা কেন বলছি যে Single Destination এর variation এ আমরা Single Source Shortest Path এর algorithm ই ব্যবহার করব।

Single Source Shortest Path এর জন্য দুইটি algorithm খুব বেশি ব্যবহার করা হয়। আসলে এই দুইটির বাইরে আমার জানাও নেই। একটি হল Dijkstra's Algorithm[১] আর আরেকটি হল BellmanFord Algorithm.

৮.৪.১ Dijkstra's Algorithm

সাধারনত এই algorithm ব্যবহার করা হয় যদি সব edge এর cost অঋণাত্মক (non-negative) হয়। একে বিভিন্ন ভাবে implement করলে বিভিন্ন time complexity পাওয়া সম্ভব। আমরা $O(n^2)$ দিয়ে শুরু করি।

- প্রথমে একটি n সাইজের array নেই, ধরা যাক এর নাম dist (distance এর সংক্ষিপ্ত রূপ) এবং এর প্রতিটি element কে ইনফিনিটি cost দেই। অনেকে ইনফিনিটি হিসাবে খুব বড় সংখ্যা যেমন $1'000'000'000$ ব্যবহার করে থাকে। অনেক সময় কেউ কেউ -1 কে ব্যবহার করে থাকতে পারে। তুমি যাই ব্যবহার কর না কেন তোমার বাকি কোডটা সেই অনুসারে লিখলেই হবে।

- যেহেতু আমরা Single Source Shortest Path সমাধান করতেছি, সুতরাং আমাদের কাছে একটি source বা যেখানে থেকে আমাদের যাত্রা শুরু সেই নোড আছে। ধরে নেই সেটা s। তাহলে আমরা উপরের array তে s এর পজিশনে 0 বসাবো। এর মানে হল, s এ পৌঁছানর cost হল 0.

- আরও একটা n সাইজের array নেই যার নাম ধরা যাক visited এবং এর প্রতিটি স্থান 0 দ্বারা initialize করি।

- এখন আমাদের এই ধাপটা বার বার করতে হবে। এই ধাপে আমরা দেখব কোন কোন নোড এর visited এ 0 আছে, তাদের মাঝ থেকে যেই নোডের cost, dist array তে সবচেয়ে কম তাকে select করি। ধরা যাক সেই নোডটা হল u. এখন visited array তে u এর পজিশনে

[১]আমি আসলে জানি না আসল উচ্চারন কি! অনেকে অনেক ভাবে উচ্চারন করে। আমিও বিভিন্ন বয়সে বিভিন্ন উচ্চারন করতাম, ছোট বেলায় ডিজিকস্ট্রা বড় হয়ে ডায়াকস্ট্রা। মাঝে মনে হয় আরও অনেক কিছুই বলতাম। তবে মোটামোটি সবাই ডায়াকস্ট্রা ই বলে অভ্যস্ত।

1 বসিয়ে দেই। এবার আমরা দেখব u থেকে কোথায় কোথায় যাওয়া যায়? ধরা যাক, u থেকে v তে যাবার জন্য একটি edge আছে যার cost হল c. আমরা দেখব কোনটি ছোট $dist[v]$ নাকি $dist[u] + c$? অর্থাৎ আমরা দেখতে চাচ্ছি যে v তে already যেভাবে যাওয়া যায় সেটা ভাল নাকি আমরা যদি প্রথমে u তে এসে এর পর $u - v$ edge ব্যবহার করে যাই তাহলে সেটা ভাল হবে। যদি $dist[u] + c$ কম হয় তাহলে $dist[v]$ কে এই মান দিয়ে update করি।[১] এভাবে আমরা একে একে u এর সাথে লাগান সব edge চেক করব।[২] এভাবে যতক্ষণ না আমাদের সব নোড visited হয়ে যায় (অর্থাৎ visited array তে সবাই 1 হওয়া পর্যন্ত) ততক্ষণ আমরা এই প্রসেস চালাতে থাকব।

- এখন তুমি dist এর array তে s থেকে সকল নোড এর সর্বনিম্ন distance পেয়ে যাবে।

এখানে আমাদের চতুর্থ ধাপ কিন্তু চলবে n বার। এবং প্রতিবার আমরা সব নোড পর্যবেক্ষণ করে বের করছি আমাদের ঐ ধাপের u কে হবে। সুতরাং আমরা n বার n সমান কাজ করছিঃ $O(n^2)$. এর পরে প্রতি u এর জন্য আমরা এর সাথে লাগান edge গুলি চেক করছি, এর মানে হল প্রতিটা edge আসলে খুব জোর 2 বার চেক হবে। সুতরাং আমাদের complexity হবে $O(n^2 + m)$ বা $O(n^2)$, কারন $m \le n^2$. কারণ তা না হওয়া মানে হল আমাদের গ্রাফে multi-edge আছে, আর multi-edge থাকলে আমাদের সবচেয়ে কম খরচের edge রেখে দিলেই হয়, সকল parallel edge না রাখলেও চলে।

এখন আমরা এই complexity কে চাইলেই কমিয়ে $O(m \log m)$ করতে পারি। খেয়াল করে দেখ আমরা একটি ধাপে লুপ চালিয়ে কোন unvisited নোড মিনিমাম সেটা বের করেছিলাম। তা না করে আমরা চাইলে STL এর priority queue ব্যবহার করতে পারি। যা করতে হবে তা হল, একটি structure এর priority queue বানাতে হবে। এর পর যখনি কোন নোড এর cost আপডেট করা হবে তখনই সেই নোডকে cost সহ priority queue তে পুশ করে দিতে হবে। আর প্রতিবার মিনিমাম cost এর নোড সিলেক্ট করার সময় priority queue থেকে মিনিমাম cost এর struct এর object নিয়ে দেখতে হবে সেখানে যে নোড আছে সেটা কি visited কিনা। যদি visited হয়ে থাকে তাহলে এটা নিয়ে আর কাজ করার দরকার নেই। খেয়াল কর আমরা এই method এ কিন্তু একটি নোড একাধিকবার পুশ করছি। আমরা ধরে নিতে পারি প্রতি edge এর জন্য খুব জোর একটি নোড পুশ হয়। সুতরাং সর্বোচ্চ m বার পুশ হয় m সাইজের heap বা priority queue তে, সুতরাং আমাদের complexity $O(m \log m)$.

যারা মনোযোগ দিয়ে পড়ছ আসা করি বুঝতে পারছ যে আমাদের আরও optimization এর সুযোগ আছে। আমরা যদি priority queue তে বার বার পুশ না করে আগের পুশ করা নোড এর cost আপডেট করতে পারতাম তাহলে কিন্তু complexity কমে যেতো। সুতরাং তোমাদের যদি আরও efficient করার প্রয়োজন হয় তাহলে নিজেরা heap বানিয়ে করতে পারো কিন্তু এতে আরও অনেক কোড করতে হয় বলে আমরা সহজে নিজে থেকে heap বানাই না।

যারা STL এর set সম্পর্কে জানো তারা হয়তো মনে করতে পারো priority queue ব্যবহার না করে set ব্যবহার করলে তো complexity আরও কমতে পারে! কারন set এ চাইলে remove করা যায়, সুতরাং আমরা আমাদের complexity $O(m \log n)$ তে নামিয়ে ফেলতে পারি। কিন্তু সমস্যা হল, set এর internal algorithm অনেক কমপ্লেক্স আরও definitely বলতে গেলে বলতে হয়, priority queue আসলে একটা heap আর set আসলে একটা red black tree. Red black tree এর internal structure অনেক কমপ্লেক্স বিধায় এদের দুজনের মোটামোটি সব অপারেশন এর complexity $O(\log n)$ হলেও set এর constant factor আমার জানা মতে অনেক বেশি। সুতরাং বেশির ভাগ সময়েই দেখা যায়, priority queue, set এর থেকে dijkstra algorithm এ ভাল perform করছে। তবে যদি কখনও তোমরা খুব dense গ্রাফ এর সম্মুখিন হউ অর্থাৎ $m \approx n^2$ তাহলে priority queue না ব্যবহার করে set ব্যবহার করলে ভাল performance পাবা।

এখন আশা করি নিজেরাই বুঝতে পারছ কেন এই algorithm ঋণাত্মক edge cost এর ক্ষেত্রে কাজ করবে না। ঋণাত্মক edge cost থাকলে বড় সমস্যা হল এই algorithm অনুসারে গ্রাফে প্রসেস

[১]update করা মানে হল পরিবর্তন করা, বা ভাল মান দিয়ে পরিবর্তন করা।

[২]খেয়াল কর, আমাদের মূল লক্ষ্য হল u থেকে যেই যেই edge দিয়ে যাওয়া যায় তাদের update করা। সুতরাং আমাদের গ্রাফ directed বা undirected যাই হোক না কেন সেইভাবে কাজ করলেই এই algorithm কাজ করবে।

করতে থাকলে এক সময় দেখা যাবে visited নোড এরও cost কমবে কিন্তু আমরা এই algorithm এ visited নোড কে দুই বার প্রসেস করি না। যদি আমরা বার বার প্রসেস করতাম আর গ্রাফে negative cycle না থাকত তাহলে এই algorithm ই negative edge cost এও কাজ করত তবে সে ক্ষেত্রে আমাদের complexity আসলে খুব একটা ভাল হবে না, আমি নিজেও নিশ্চিত না complexity কত হবে, মনে হয় exponential এর মত কিছু হবে। তবে এভাবে যে negative edge cost ওয়ালা গ্রাফে shortest path বের করা সম্ভব তা জেনে রাখা ভাল। যদি আমার দুর্বল স্মৃতিশক্তি আমার সাথে দুষ্টুমি না করে তাহলে আমার মনে হয় আমাকে এভাবেও কিছু প্রবলেম সল্ভ করতে হয়েছিল।

৮.৪.২ BellmanFord Algorithm

এটিও single source shortest path বের করার একটি algorithm এবং এটি dijkstra এর তুলনায় অনেক সহজ এবং ঋণাত্মক edge cost এ এটি কাজ করে। তাহলে আমরা dijkstra শিখ-লাম কেন? কারন এর complexity $O(mn)$ যা dijkstra এর তুলনায় অনেক বেশি। যদি গ্রাফে negative cycle ও থাকে তাহলে এই algorithm তা বুঝতে পারে। negative cycle হল গ্রাফের এমন একটি cycle যেখানে edge cost এর sum ঋণাত্মক হয়। অর্থাৎ তুমি একটা নোড থেকে শুরু করে বিভিন্ন edge হয়ে আবার শুরুর নোড এ ফিরে আসবে আর দেখতে পাবে যে তোমার edge cost এর sum ঋণাত্মক হয়ে গেছে। এটি কেন সমস্যার কারন বুঝতে পারছ তো? কারন হল, negative cycle এ আছে এমন একটি নোড এ তুমি যদি যেতে পারো তাহলে সেই নোড এ পৌঁছানর খরচ তুমি কিন্তু negative cycle ব্যবহার করে কমাতেই থাকতে পারো। সুতরাং ঐ সকল নোড এর minimum cost আসলে undefined বা negative infinity বা এরকম অনেক কিছুই বলতে পারো। অনেক প্রবলেমেই আছে যেখানে তোমাকে বলতে বলবে কোন কোন নোড এরকম negative cycle এ আছে বা শুরুর নোড থেকে কোন কোন নোড এ যাওয়া যায় যারা negative cycle এ আছে। এসব ক্ষেত্রে আমরা BellmanFord algorithm ব্যবহার করতে পারি। খেয়াল কর, negative cycle এ থাকা মানেই শুরুর নোড থেকে negative infinity cost এ পৌঁছান না!!

এই algorithm কে দুইটি অংশে ভাগ করা যায়।

প্রথম অংশে আমরা shortest path বের করব। প্রথমত আমাদের একটি n সাইজের $dist$ এর অ্যারে নিতে হবে যার সকল element হবে infinity কেবল source হবে 0. এখন আমাদের এক-টি কাজ n বার করতে হবে। কাজটি হল, সব edge একে একে নিতে হবে,ধরা যাক একটি edge হল a থেকে b তে এবং তার cost হল c (যদি গ্রাফটি bidirectional হয় তাহলে অন্য দিকের edge টাও আলাদা ভাবে consider করতে হবে)। এখন তোমাকে দেখতে হবে, $dist[b]$ বড় না-কি $dist[a] + c$ বড়। সেই অনুসারে আপডেট করতে হবে। এই ধাপটা n বার চালালেই তোমাদের shortest path বের হয়ে যাবে। খেয়াল কর, আমরা বাইরের লুপ চালাচ্ছি n বার আর ভিতরের edge এর লুপ চলছে m বার সুতরাং আমাদের complexity হবে $O(mn)$. তোমরা চাইলে এখানে একটা ভাল optimization করতে পারো এবং এটি প্রায়ই কাজে লাগে বিশেষ করে যখন mincost maxflow তে আমরা bellman-ford ব্যবহার করে থাকি[১]। optimization টা হল, আমরা যখনি দেখব n এর লুপের ভিতরের edge এর লুপে কোন edge এ কোন আপডেট ঘটে নাই, তাহলে n এর লুপ কে break করে ফেলো। কারন পরের অন্য কোন লুপে আর কোন আপডেট হবে না। আর আরেকটা কাজও করতে পারো, সেটা হল, bellman ford চালানর আগে edge গুলোর order তু-মি randomize করে ফেলো। এতে সুবিধা হল কেউ যদি bellman ford এর জন্য বাজে case বানায়ও, তুমি edge এর order পরিবর্তন করে ফেলায় সেটা আর থাকবে না।

এখন দ্বিতীয় অংশে আশা যাক। দ্বিতীয় অংশে আমরা বের করব গ্রাফে negative cycle আছে কিনা। এটা করার জন্য যা করতে হবে তা হল, আমাদের আগের n এর লুপ এর ভিতরের অংশ আরেক-বার চালাতে হবে। যদি দেখ এই $n + 1$ তম বারে আবারও কোন edge দ্বারা নোড এর cost আপডেট করা যায় তাহলেই বুঝবা যে তোমার গ্রাফে negative cycle আছে।

---

[১]যাক আমার দুর্বল স্মৃতিশক্তি আমার সাথে দুষ্টামি করে নাই! আমি mincost maxflow তে negative cost এর edge এর জন্য dijkstra করেছি বহুবার।

৮.৫ All pair shortest path বা Floyd Warshall Algorithm

আমাদের dijkstra algorithm এর complexity ছিল $O(m \log n)$ এর মত। আমরা যদি সব নোড থেকেই dijkstra চালাতাম তাহলে all pair shortest path বের করতে সময় লাগত প্রায় $O(nm \log n)$ এর মত। যদি গ্রাফ টা খুব একটা dense ($m \approx n^2$) না হয় তাহলে n বার dijkstra চালানই ভাল। সত্যি কথা বলতে যেখানে floyd warshall চালাতে হবে সেখানে বার বার dijkstra চালালেই হয়ে যাবার কথা। তাহলে আমরা floyd warshall শিখব কেন? এর একমাত্র কারন হল এর কোড খুবই ছোট ও সহজ। মাত্র পাঁচ লাইন আর সেই পাঁচ লাইনের মাঝে তিনটি for-loop। এটি মনে রাখাও খুব সহজ। প্রথমেই আমরা algorithm টা দেখে নেইঃ

কোড ৮.৫: apsp.cpp

```
for(k = 1; k <= n; k++)
  for(i = 1; i <= n; i++)
      for(j = 1; j <= n; j++)
          if(w[i][j] > w[i][k] + w[k][j])
              w[i][j] = w[i][k] + w[k][j];
```

শুধু মনে রাখতে হবে যে, প্রথমে k এর লুপ এর পর i আর j. এখানে শেষ দুই লাইনে কি করা হচ্ছে তাতো বুঝতে পারছ? দেখা হচ্ছে যে, i থেকে j তে যাবার cost কি k হয়ে যাওয়ার cost থেকে ভাল না খারাপ। এর পর আমরা ভাল cost দিয়ে আপডেট করে দেব। তোমরা যারা এখনও ভাবছ যে w এর array তে কি আছে তাদের জন্য বলছি, এই array এর initial ভ্যালু হবে infinity. যদি তোমার গ্রাফে i ও j এর মাঝে কোন edge থাকে তাহলে তার cost হবে $w[i][j]$ এর মান। যদি একাধিক edge থাকে তাহলে সর্বনিম্নটা নিবে। যদি গ্রাফ টা undirected হয় তাহলে একই সাথে $w[j][i]$ তেও সেই মান দিয়ে দিবে।

এখন প্রশ্ন হল এর complexity কত? খুবই সহজ $O(n^3)$.

আরও একটি প্রশ্ন হল, ঋণাত্মক edge cost এ floyd warshall কি কাজ করবে? হ্যা করবে। শুধু তাই না, গ্রাফে কোন কোন নোড দিয়ে negative cycle যায় তাও বের করা যাবে। তুমি শুরুতে সকল $w[i][i]$ এ 0 নিবে। এর পর floyd warshall চালানর পর যদি দেখ যে কোন একটি $w[i][i]$ এ negative মান এর মানে হল ঐ নোড দিয়ে একটি negative cycle গিয়েছে।

৮.৬ Dijkstra, BellmanFord, Floyd Warshall কেন সঠিক?

Dijkstra কেন সঠিক এটা আসলে ব্যখ্যা করার কিছু নেই। তুমি প্রতিবার সবচেয়ে কম cost এ যাওয়া যায় এমন vertex কে visited করছ আর যেহেতু তোমার edge cost অঋণাত্মক সেহেতু আগের visited কোন নোডে আসলে আরও কম খরচে তুমি যেতে পারবে না।

BellmanFord এ ভিতরের লুপ এ কি করছি তাতো বুঝা যায়, যেটা বুঝা যায় না সেটা হল কেন সেই কাজ n বার করলেই আমরা shortest পাথ পাবো! খেয়াল কর, তুমি যদি s হতে shortest path বের করতে চাও সব জায়গার তাহলে প্রতিটি জায়গায় তুমি n এর থেকে কম edge দিয়ে পৌছা-তে পারবে। এখন ভিতরের লুপ দিয়ে কিন্তু আমরা এই কাজ টাই করছি। যদি মনে করে থাকো একবার চালালেই তো সব বের হয়ে যাওয়া উচিত। না, এখানে কিন্তু edge এর order টা খুব important. যদি কেও চায় তাহলে সে edge এর order এমন ভাবে দিতে পারে যে একবার লুপ ঘুরলেই সব জায়গার shortest path বের হয়ে যাবে, আবার কেউ যদি চায় তাহলে n বারই ঘুরাতে পারবে।[১]

Floyd warshall কেন সঠিক এটা বুঝা একটু ঝামেলা। এর জন্য যেটা বুঝতে হবে সেটা হল k এর লুপটা বাইরে কেন[২]? এখানের k এর লুপকে Bellman Ford এর বাইরের লুপ এর মত ভাবলে

[১]আসলে $n-1$ বার ঘুরালেই হয়। কোন এক ঐতিহাসিক কারনে আমরা সবসময় n বার বলে থাকি।

[২]আগে আমি বেশ কয়েকবার এই ভুল করতাম, প্রায় সময় k এর লুপ ভিতরে দিয়ে থাকতাম ভাবতাম একই তো কথা! কিন্তু এক কথা না।

হবে না। ভিতরের দুইটি লুপ n বার ঘুরান এর উদ্দেশ্য না, এর উদ্দেশ্য হল i হতে j তে যাবার সময় যদি k দিয়ে যাওয়া হয় তাহলে সেটা ভাল হয় কিনা এটা বুঝা। আরও ভাল করে বলতে, যদি বাইরের লুপ k বার ঘুরে এর মানে হবে, i হতে j পর্যন্ত শুধু $1, 2, \ldots k$ দিয়ে গেলে সবচেয়ে কম যত cost এ যাওয়া যায় তা $w[i][j]$ তে থাকবে। সুতরাং আমরা যদি n পর্যন্ত লুপ চালাই তাহলে আসলে shortest path পেয়ে যাব।

এখন অনেকে ভাবতে পারে যে- বুঝলাম $k-i-j$ কেন সঠিক কিন্তু $i-k-j$ বা $i-j-k$ কেন সঠিক না। এই দুইটি কেন সঠিক না সেটার জন্য আমি case দিচ্ছি। তোমরা নিজেরা চিন্তা করে দেখবে কেন এই দুইটি উদাহরণে $i-k-j$ বা $i-j-k$ সঠিক ভাবে কাজ করে না। ধরা যাক আমাদের গ্রাফে দুইটি shortest path হলঃ $1-5-3-2$ এবং $5-4-3-2$, তাহলে এই দুই case এ আমাদের পরিবর্তিত সমাধান কাজ করবে না।

৮.৭ Articulation vertex বা Articulation edge

একটি undirected গ্রাফ এ যদি কোন নোড কে মুছে ফেললে গ্রাফটা disconnected হয়ে যায় তাহলে তাকে articulation vertex বলে। একই ভাবে যদি কোন edge কে মুছে ফেললে গ্রাফটা disconnected হয়ে যায় তাহলে তাকে articulation edge বা articulation bridge বলে। DFS ব্যবহার করে খুব সহজেই articulation vertex বা edge বের করে ফেলা যায়। DFS এর একটা powerful প্রয়োগ হল এটি। এটা করার জন্য আমাদের কয়েকটি জিনিসের সাথে পরিচিত হতে হবেঃ dfsStartTime, dfsEndTime এবং low. Articulation vertex বা edge বের করতে এদের সবগুলিই যে দরকার তা নয়, কিন্তু এদের ব্যবহার করে আমরা বেশ জটিল জটিল প্রবলেম সমাধান করে ফেলতে পারি। বিশেষ করে Informatics Olympia লেভেলে এধরনের অনেক প্রবলেম দেখা যায়। যতদূর মনে পরে 2006 সালের IOI এ এরকম একটি প্রবলেম ছিল। যাই হোক, dfsStartTime ও dfsEndTime খুবই সহজ জিনিস। তুমি dfsTime বলে একটা variable রাখবে যার initial মান হবে 0. এর পর তোমরা dfs করার সময় যখনি কোন একটা নতুন unvisited নোডে আসবে তখনই dfsStartTime এ dfsTime এর বর্তমান সময় নোট করবে আর কোন নোডের সকল child এর visit শেষ হয়ে গেলে সেই time টা dfsEndTime এ মার্ক করে রাখবে। আর প্রতিবার নতুন vertex কে visit করার সময় dfsTime কে এক করে বাড়াবে। এতো গেল dfsStartTime আর dfsEndTime. low একটু জটিল জিনিস। আমরা তো জানি dfs পুরো গ্রাফে একটা tree এর মত করে আগায়। মানে কোন নোডের যদি unvisited adjacent vertex থাকে তাহলে আমরা সেটা visit করি (নিচে নামি) আর যদি কোন unvisited adjacent vertex না থাকে তাহলে ফেরত যাই (parent এ ফেরত যাই)। যদি সেই নোড থেকে কোন visited নোড এ যাওয়া যায় তা অবশ্যই এর ancestor হবে অর্থাৎ ঐ নোড থেকে root এর path এর মাঝেই থাকবে (চিন্তা করে দেখ) অথবা তার subtree তে থাকবে। যদি কোন নোড থেকে তার ancestor এ যাওয়া যায় তাহলে সেই edge কে আমরা back edge বলি। low[u] হল u নোড বা u এর subtree তে থাকা নোডগুলি থেকে সবচেয়ে উপরে (root এর কাছের) যেই নোড এ যাওয়া যায় তার dfsStartTime.

low[u] বের করার জন্য যা করতে হবে তা হলঃ ধরা যাক v হল u এর adjacent কোন vertex. যদি v ইতোমধ্যেই visited হয়ে যায় তাহলে হয় v হবে u এর parent অথবা ancestor। যদি ancestor হয় তাহলে তার dfsStartTime দিয়ে low[u] কে update করতে হবে। আর যদি parent হয় এবং তুমি যদি articulation edge বের করতে চাও তাহলে একটু সতর্ক হতে হবে। parent থেকে যেই edge দিয়ে আমরা u তে এসেছি যদি সেই edge হয় এটা তাহলে আমরা কোন কিছু করব না, আর যদি এটা ভিন্ন edge হয় তাহলে আগের মত update করতে হবে। যদি আমাদের গ্রাফ multi edge গ্রাফ না হয় তাহলে আমাদের এটা নিয়ে ভাবার কিছু নেই। এখন যদি আমাদের v নোডটা আগে থেকে visited না হয় তাহলে তার dfs করতে হবে recursively এবং low[v] দিয়ে low[u] কে update করতে হবে। এখানে update করা মানে হল minimum value টা বের করা। এই low[] ভ্যালু কিন্তু কোন একটি নোড এর dfsStartTime, আমাদের এই time যত কম হবে ততই সেই নোড root এর কাছাকাছি হবে।

এখন আমাদের সব দরকারি value বের করা হয়ে গেছে। এই value গুলো দেখে আমরা বলতে পারব কোন কোনটা articulation vertex আর কোন কোনটা articulation edge। নোড u, articulation vertex হবে ১. যদি এটি root হয় এবং এর একাধিক child থাকে অথবা ২. এটি root না হয় এবং $low[u] \geq dfsStartTime[u]$ হয়। এখন আশা করি তোমরা একটু চিন্তা করলেই বুঝবে কেন এই দুইটি condition এর একটি সত্য হলে সেই নোডটি articulation vertex হবে বা কোন নোড articulation vertex হলে কেন এই দুই condition এর একটি সত্য হবে।

যদি উপরের condition দুইটা বুঝে থাকো তাহলে আশা করি $u - v$ edge কখন articulation edge হবে তাও বের করে ফেলতে পারবে। condition টা হল, u যদি v এর parent হয় তাহলে $low[v] > dfsStartTime[u]$ হতে হবে। খেয়াল কর, কোন back edge কিন্তু কখনই articulation edge হতে পারবে না। সুতরাং আমাদের শুধু dfs tree এর edge গুলি check করলেই হবে।

৮.৮ Euler path এবং euler cycle

আমরা প্রথমে শুধু undirected graph নিয়ে ভাবব। Euler[১] path হল কোন একটি গ্রাফে যদি একটি vertex থেকে যাত্রা শুরু করে প্রতিটি edge, exactly একবার করে ঘুরে কোন একটি vertex এ যাত্রা শেষ করা যায় তাহলে তাকে euler path বলে। আর যদি শুরু ও শেষের vertex একই হয় তাহলে তাকে euler cycle বা euler circuit বলে। কোন একটি গ্রাফে euler path বা cycle আছে কিনা তা বের করা খুবই সহজ। প্রথম শর্ত হল গ্রাফ কে connected হতে হবে। এখন যদি সব গুলি নোড এর degree জোড় হয় তাহলে গ্রাফ এ euler cycle আছে (euler cycle থাকা মানে কিন্তু euler path ও থাকা, কিন্তু উল্টোটা সত্য নয়)। আর যদি এই গ্রাফ এর শুধুমাত্র দুইটি নোড odd degree ওয়ালা হয় তাহলেও গ্রাফটাতে euler path থাকবে তবে সেক্ষেত্রে আমাদের অবশ্যই ঐ দুইটি নোড এর কোন একটি থেকে যাত্রা শুরু করতে হবে। এরকম হবার কারন হল, শুরু আর শেষের নোড বাদে বাকি সব নোড এর ক্ষেত্রে আমরা কিন্তু একবার ঢুকলে বের হতে হয় সুতরাং আমাদের edge গুলি জোড়ায় জোড়ায় থাকে বা বলতে পারি মাঝের সব vertex গুলির degree হবে জোড়। এখন যদি cycle হয় তাহলে যেখান থেকে শুরু করেছি সেখানেই শেষ করেছি সুতরাং সেই নোড এর degree ও জোড় হবে। কিন্তু যদি এটা cycle না হয়ে path হয় তাহলে দেখ শুরু আর শেষ vertex আলাদা এবং তাদের degree হবে বিজোড়। এটা তো আমরা প্রমান করলাম যে euler path বা cyle হলে এরকম property থাকবে। কিন্তু এরকম property থাকলেই যে euler path বা cycle হবে তা কিন্তু প্রমান করি নাই। সেটা প্রমান করাও কিন্তু খুব একটা কঠিন না। তোমরা induction ব্যবহার করে খুব সহজেই প্রমান করতে পারবে।

এখন কোড করে কেমনে আমরা euler path বা cycle বের করতে পারব? এটাও খুব সহজ, dfs এর মত তুমি কোন একটি vertex থেকে যাত্রা শুরু কর। তবে এখানে আমাদের vertex এর জন্য কোন visited থাকবে না, থাকবে edge এর জন্য। খেয়াল রেখো কোন একটা edge কিন্তু দুই দিকের কোন একদিক থেকেই visit করা যায়। এখন কোন একটি vertex এ আমরা দাঁড়িয়ে দেখব যে এর থেকে বের হওয়া কোন কোন edge এখনও visited হয় নাই, যদি এমন কোন edge বাকি থাকে তাহলে সেটা দিয়ে বের হয়ে যাব এবং আগের মতই ঘুরতে থাকব। আর যদি দেখি এই vertex এর সাথে লাগান সব edge ই visited হয়ে গেছে তাহলে এই vertex কে print করে দেব। খেয়াল কর এই প্রসেস এ কিন্তু একটা vertex কিন্তু অনেক বার visit হতে পারে।

এবার দেখা যাক directed গ্রাফ এ কেমনে আমরা euler path বা cycle বের করতে পারি। আসলে আমি এই paragraph লিখার আগে এই জিনিস এর সম্মুখিন হই নাই বা হলেও মনে নেই। সুতরাং আমি একটু internet ঘেঁটে ঘুটে যা দেখলাম তাহল directed গ্রাফ এর euler path বা cycle বের করা প্রায় পুরোপুরি undirected গ্রাফ এর মত। আমরা কোন একটা নোড থেকে শুরু করব, এর outgoing কোন edge দিয়ে বের হব যতক্ষণ কোন না কোন outgoing edge বাকি থাকে। যখনি শেষ হয়ে যাবে আমরা print করে দিব এবং আগের নোড এ ফিরে যাব, ঠিক আগের dfs এর মত। আশা করি বুঝতে পারছ যে আমাদের euler path বা cycle এর ঠিক উলটো order প্রিন্ট হয়েছে! আরেকটা জিনিস, তাহল path print করার আগে প্রতিটি নোড এর outdegree আর indegree একটু দেখে নিতে হবে। প্রতিটি নোড এর indegree আর outdegree সমান হতে হবে কেবল একটি নোড এর indegree, outdegree হতে এক বেশি হতে পারবে এবং একটি নোড এর

[১] Euler এর উচ্চারন অয়লার।

outdegree, indegree এর থেকে এক বেশি হতে পারবে। তাহলে তারা যথাক্রমে path এর শেষ ও শুরু হবে। কিন্তু সবার যদি in আর out degree সমান হয় তাহলে যেকোনো নোড থেকে শুরু করে সেখানে ফেরত আশা যাবে। তবে আগের মতই connected ব্যপার টা একবার দেখে নিতে হবে। খেয়াল রাখতে হবে যেন, শুরুর নোড থেকে যেন সব জায়গায় যাওয়া যায় আর শেষের নোড এ যেন সব জায়গা থেকে আসা যায়।

৮.৯ টপলজিকাল সর্ট (Topological sort)

একটি directed গ্রাফে নোডগুলিকে এমন ভাবে অর্ডার করতে হবে যেন, যদি u থেকে v তে কোন directed edge থাকে তাহলে সর্টেড অ্যারেতে u, v এর আগে থাকে। এটা তখনি সম্ভব যখন ঐ গ্রাফে কোন directed cycle থাকবে না। cycle থাকলে তো ঐ cycle এর নোড গুলিকে তুমি কোন ভাবেই অমন অর্ডার দিতে পারবে না তাই না? আমরা এই অ্যালগোরিদম ব্যবহার করে কোন directed গ্রাফে cycle আছে কিনা তাও বের করে ফেলতে পারব।

আমরা দুই ভাবে topological sort করতে পারি। একটি হল BFS দিয়ে আরেকটি DFS দিয়ে। আমার কাছে BFS দিয়ে তুলনামুলক সহজ মনে হয়, এছাড়াও এখানে stack overflow নিয়ে মাথা ঘামাতে হয় না। কিন্তু DFS একটু তুলনামুলক ভাবে ছোট হয়ে থাকে। প্রথমে দেখা যাক আমরা BFS দিয়ে কেমনে সমাধান করতে পারি।

প্রথমে আমাদের একটি indegree এর অ্যারে লাগবে যেখানে প্রতিটি নোড এর indegree লিখা থাকবে। এখন একটি queue তে সেসব নোড রাখতে হবে যাদের indegree শূন্য। এবার queue থেকে একে একে নোড তুলার পালা। একটা করে নোড তুলবো আর তার থেকে যেসব edge বের হয়ে গেছে তাদের দেখে দেখে অন্য প্রান্তের নোড এর indegree কমায়ে দিব। যদি অন্য প্রান্তের indegree শূন্য হয়ে যায় তাহলে তাকে queue তে ঢুকিয়ে দিব। এভাবে যতক্ষণ না queue ফাঁকা হয়ে যায় ততক্ষণ চলতে থাকবে। queue তে নোড যেই order এ ঢুকেছে সেটাই কিন্তু topological sort এর অর্ডার। তবে একটা জিনিস, যদি queue ফাঁকা হয়ে যাবার পরেও দেখ যে কোন নোড এর indegree এখনও শূন্য হয় নাই তার মানে গ্রাফটায় cycle আছে। এটি কিন্তু খুবই intuitive একটা অ্যালগোরিদম। কেন কাজ করছে তা খুব সহজেই বুঝা যায়।

এবার আশা যাক DFS দিয়ে কেমনে এটা সমাধান করা যায়। ধরা যাক T হল আমাদের রেজাল্ট এর একটি অ্যারে, আর visited আরেকটি অ্যারে যার initial মান 0. এখন আমরা একে একে প্রতিটি নোড দেখব আর যদি সেই নোড এর visited এর মান এখনও 2 না হয়ে থাকে তাহলে তার DFS কল করব। DFS এর প্রথমেই আমরা যা করব তাহল এর visited এর মান 1 করে দেব। এর পর এখান থেকে যেই যেই directed edge বের হয়েছে তাদের দেখব। যদি অন্য প্রান্তের নোড visited এর মান 1 হয়ে থাকে এর মানে হল আমরা একটা cycle পেয়ে গেছি সুতরাং নোড গুলির কোন topological order নেই। যদি visited এর মান 2 হয় তাহলে আমাদের করার কিছু নেই। আর যদি 0 হয় তাহলে আমরা ঐ নোড এর জন্য DFS কল করব। এভাবে প্রতিটি নোড এর জন্য প্রসেসিং শেষ হলে আমরা সেই নোড এর visited কে 2 করে দেব এবং তাকে T তে ঢুকিয়ে দেব এবং DFS থেকে return করব। এভাবে সব নোড এর জন্য DFS শেষ হয়ে গেলে আমরা T তে topological sort উলটো অর্ডারে পাবো।

৮.১০ Strongly Connected Component (SCC)

একটি directed গ্রাফে যখন একটি নোড u থেকে v তে যাওয়া যায় এবং একই সাথে v থেকে u নোড এ যাওয়া যায় তখন আমরা বলব ঐ দুইটি নোড একই SCC তে আছে। খেয়াল করে দেখ, যদি u আর v একই SCC তে থাকে আর v আর w ও একই SCC তে থাকে তাহলে u আর w ও একই SCC তে থাকবে কারন তুমি w থেকে v তে যেতে পারো আর v থেকে u তে যেতে পারো, একই ভাবে u থেকেও v হয়ে তুমি w তে যেতে পারো। সুতরাং u আর w একই SCC তে। একটু চিন্তা করলে বুঝবে যে এর মানে দাঁড়ায় তুমি পুরো গ্রাফকে আসলে অনেকগুলি SCC তে ভাংতে পারবে যেন কোন একটি নোড কোন একটি এবং কেবল মাত্র একটি SCC এর অংশ। যেমন ধর, যদি আমাদের edge

গুলি হয়ঃ $(u, v), (v, w), (w, u)$ তাহলে এখানে কেবল মাত্র একটি SCC: $\{u, v, w\}$. আবার ধরা যাক আমাদের edge গুলি হলঃ $(u, v), (w, v)$ তাহলে কিন্তু তিনটি SCC: $\{u\}, \{v\}, \{w\}$. আবার $(u, v), (v, u), (w, u), (w, v)$ হলে দুইটি SCC: $\{u, v\}, \{w\}$.

এখন আমরা SCC বের করার অ্যালগোরিদম শিখব। এর জন্য দুইটি বহুল প্রচলিত অ্যালগোরিদম আছে। একটি হল Kosaraju's algoritm (2-dfs অ্যালগোরিদম) আরেকটা হল Tarjan's algorithm. আমি আসলে শুধু প্রথমটাই জানি। এটা করা বা বলা সহজ, কিন্তু আমার কাছে মনে রাখা বেশ কষ্টকর মনে হয় এবং বুঝাও একটু কষ্টকর মনে হয়। প্রথমে একটি visited এর অ্যারে নাও এবং সব নোড কে unvisited করে দাও। এখন একে একে নোড গুলি চেক কর, যদি কোন unvisited নোড পাও তাহলে তার জন্য dfs কল কর। dfs এর ভিতরে যা করবা তাহল, ঐ নোড কে visited করে দিবা আর ঐ নোড থেকে যদি কোন unvisited নোড এ যাওয়া যায় তাহলে তার dfs কল করবা। adjacent সব নোড visited হয়ে গেলে dfs থেকে return করার আগে একটা লিস্টের শেষে (ধরা যাক তার নাম L) ঐ নোডকে পুশ করে যাবা। তাহলে সব নোড visited হয়ে গেলে কিন্তু আমাদের ঐ L লিস্টে সব নোড থাকবে। এবার যেটা করতে হবে তাহল ঐ গ্রাফের সব edge কে উলটো করে দিতে হবে (আসলে তুমি শুরুতেই দুইটি adjacency list বানায়ে নিবা একটা ঠিক দিকে আরেকটা উলটো দিকে)। তুমি যেই নোড এর লিস্ট L বানায়ে ছিলা ওটাকেও উলটো করতে হবে। এবার আমরা আরও একটা DFS করব। আমাদের আবার সব নোড কে unvisited করে দিতে হবে। এখন আমরা L এর সামনের দিক থেকে একটা একটা করে নোড নিব এবং সে যদি visited না হয়ে থাকে তার জন্য dfs কল করব। খেয়াল রেখো, এবার dfs এ কিন্তু আমরা উলটো গ্রাফ ব্যবহার করছি। এই dfs এর সময় যেই যেই নোড visited হবে তারা একটা SCC [১]. এভাবে আমরা সব SCC পেয়ে যাব।

এই বই লিখতে গিয়ে আমি Tarjan এর SCC অ্যালগোরিদম দেখলাম। খুব একটা কঠিন না। এই অ্যালগোরিদমটা কিছুটা Articulation Bridge বা Vertex বের করার মত। তবে মনে রাখতে হবে এবার আমরা directed গ্রাফ নিয়ে কাজ করছি। সবসময়ের মত প্রতিটি নোড visited না হওয়া পর্যন্ত আমরা প্রতিবার একটি একটি করে unvisited নোড নিয়ে তার জন্য dfs কল করব। dfs এর শুরুতে আমরা তার startTime আর low কে বর্তমান time এর মান দ্বারা আপডেট করে time এর মান এক বাড়ায়ে দেব। সেই সাথে এই নোডকে একটা stack এ পুশ করব। এবার এই নোড থেকে যেখানে যেখানে যাওয়া যায় তাদের দেখার পালা। যদি অপর নোডটি আগে থেকেই visited হয় তাহলে আমরা বর্তমান নোড এর low কে ওপর নোড এর startTime দিয়ে আপডেট করব (minimum নিব) আর যদি unvisited হয় তাহলে তার জন্য dfs কল করব। এভাবে সব neighbor এর জন্য প্রসেসিং শেষ হলে আমাদের দেখতে হবে যে তার low এর মান startTime এর মানের সমান কিনা, অর্থাৎ আমরা তার neighbor দিয়ে তার থেকেও আগের কোন নোড এ যেতে পারি কিনা। যদি যেতে পারি (low এর মান startTime এর থেকেও কম) তাহলে এখানে আর কিছু করার নেই। আর যদি সমান হয়, তাহলে আমাদের stack থেকে নোড তুলতেই থাকব যতক্ষণ না আমাদের বর্তমান নোড পাই। এই সব নোডগুলি হল একটি SCC.

৮.১১ 2-satisfiability (2-sat)

(a or b) and (!b or c) and (!a or !c) এই equation এর একটি সমাধান হতে পারেঃ a = 1, b = 0, c = 0. কিন্তু অনেক সময় কোন সমাধান নাও থাকতে পারে, যেমনঃ (a or b) and (a or !b) and (!a or b) and (!a or !b). Formally বলতে গেলে a, b, c এগুলোকে variable বলা হয় আর দুইটি করে variable বা তাদের not নিয়ে or করে যে এক একটি pair বানানো হয় তাদের clause বলে। অনেক গুলি clause এর and করে বড় equation বা statement তৈরি করা হয়। আমাদের লক্ষ্য হল, variable গুলিতে এমন মান assign করা যায় কিনা যেন আমাদের statement টা সঠিক হয়। যেহেতু প্রত্যেকটা clause এ দুইটি করে term থাকে সেজন্য একে 2-sat প্রবলেম বলা হয়। তোমাদের জানার জন্য বলে রাখি 3-sat প্রবলেম হল NP আর দুনিয়ার মোটামোটি অনেক প্রবলেম এদিক ওদিক করে 3-sat এ কনভার্ট করা যায়।

[১]তোমরা চাইলে dfs এর ফাংশন কে একটি number দিয়ে দিতে পারো যে এটা হল SCC এর নাম্বার এটা দিয়ে visited কে মার্ক করতে, বা চাইলে একটা লিস্ট ও parameter এ দিয়ে দিতে পারো যেন visited নোড গুলি ঐ লিস্ট এ রাখা হয়।

যদি আমাদের statement এ n টি variable থাকে তাহলে আমাদেরকে একটি 2n নোড এর directed গ্রাফ বানাতে হবে। x দিয়ে যদি একটি variable থাকে তাহলে x এর জন্য একটা নোড আরেকটা !x এর জন্য। এখন ধরা যাক (!x or y) হল একটি clause, তাহলে একে আমরা দুই ভাবে লিখতে পারিঃ $x \rightarrow y$ আর $!y \rightarrow !x$.[১] বা (!x or y) কে এভাবে তুমি interpret করতে পারো যদি !x মিথ্যা হয় তাহলে y সত্য হবে, অথবা যদি y মিথ্যা হয় তাহলে !x সত্য হবে। আমরা একে গ্রাফে directed edge দিয়ে প্রকাশ করব। (!x or y) এর ক্ষেত্রে আমাদের edge হবে x থেকে y এর দিকে আর !y থেকে !x এর দিকে। অন্যভাবে বলতেঃ যদি x সত্য হয় তাহলে y ও সত্য হবে, আর যদি !y সত্য হয় তাহলে !x ও সত্য হবে। এখন আমরা এভাবে প্রত্যেকটা clause এর জন্য দুইটি করে edge দিব। সব edge আঁকা শেষে আমাদের দেখতে হবে যে, x আর !x (এখানে x হল যেকোনো variable, অর্থাৎ তোমাকে n টি variable এর জন্য চেক করে দেখতে হবে) এর জন্য x থেকে !x এ আর !x থেকে x এ দুইদিকেই path আছে কিনা। যদি থাকে তাহলে আমাদের 2-sat সমাধান করা যাবে না। আর যদি এমন কোন variable খুজে পাওয়া না যায় তাহলে সমাধান করা যাবে। আমরা দুইটি নোড থেকে একে অপরের দিকে যাওয়া যায় কিনা কেমনে সহজে বের করতে পারি? SCC! যদি আমাদের গ্রাফকে SCC তে ভাঙ্গার পরে দেখি x ও !x একই component এ তাহলে বুঝবো একে ওপরের দিকে যাওয়া যায়, আর না হলে যাবে না।

এখন কথা হল একই component এ হলে সমস্যা কই? খেয়াল কর, যদি কখনও x থেকে y এ যাওয়া যায় এর মানে দাঁড়াবে, x সত্য হলে y সত্য হবে। তাহলে যদি x থেকে !x এ পাথ থাকে তার মানে দাঁড়াবে x সত্য হলে !x সত্য হবে। অর্থাৎ x কে অবশ্যই মিথ্যা হতে হবে। এটা সমস্যা না। সমস্যা হবে যদি একই সাথে !x থেকেও x এ পাথ থাকে। তাহলে x কে অবশ্যই সত্য হতে হবে। x যেহেতু একই সাথে সত্য আর মিথ্যা হতে পারবে না সেহেতু আমাদের 2-sat এরও সমাধান থাকবে না।

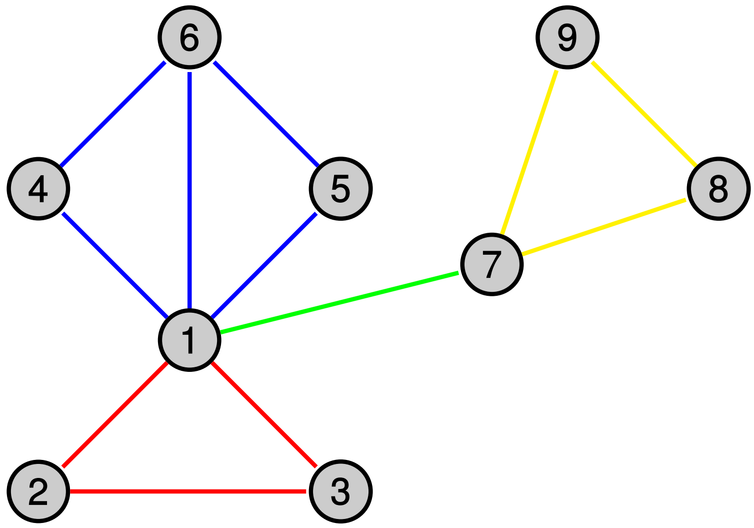
কথা হল আমরা বের করলাম কেমনে 2-sat সমাধানযোগ্য কিনা তা বের করা যায়। সমাধান কেমনে বের করা যায়? এর উপায় হল, তুমি scc এর প্রত্যেক component কে contract (সংকুচিত) করে একটি নোড বানাও। এই পরিবর্তিত গ্রাফ অবশ্যই একটি DAG হবে (DAG = Directed Acyclic Graph) অর্থাৎ এই directed গ্রাফে কোন cycle নাই। যেহেতু কোন cycle নাই তাই এর topological order আছে। আমাদের যা করতে হবে, এই অর্ডার এর শেষ থেকে আসতে হবে আর প্রত্যেক component কে true দেবার চেষ্টা করতে হবে। যদি দেখ তোমার এই component এ এমন একটা variable আছে যার মান আগে থেকেই assign করা হয়ে গেছে আর তোমার এই এখন true করতে চাওয়া মান এর সাথে conflict করছে তাহলে false দিবা। তুমি চাইলে চিন্তা করে দেখতে পারো বা প্রমানও করতে পারো কেন এই greedy প্রসেসটা ঠিক ভাবে মান assign করছে।

৮.১২ Biconnected component

সত্যি কথা বলতে আমি এই অ্যালগোরিদম নিজে থেকে কখনও implement করি নাই। আমার কাছে বেশ কঠিন লাগে বা বলতে পারো সময় সাপেক্ষ লাগে। Strongly connected component এ আমরা নোড গুলিকে বিভিন্ন ভাগে ভাগ করেছিলাম এবং এদেরকে আমরা SCC বলেছিলাম। Biconnected component বা BCC ও কিছুটা একই রকম। আমরা একটি undirected graph এর vertex সমূহ কে BCC তে ভাগ করতে পারি। Biconnected graph মানে হল সেই গ্রাফের যদি কোন vertex কে আমরা মুছে ফেলি তাহলে সেই গ্রাফ disconnected হবে না। যেমন ধরা যাক আমাদের তিন নোড এর একটি গ্রাফ আছে এবং এদের মাঝের edge গুলি হল $1-2, 2-3$. এটি কিন্তু biconnected graph না কারণ এই গ্রাফ থেকে আমরা যদি 2 মুছে ফেলি তাহলে বাকি নোড দুইটি disconnected হয়ে যায়। কিন্তু এই গ্রাফটায় যদি আরও একটি edge $1-3$ থাকত তাহলে কিন্তু এটি biconnected graph হতো। আমরা যেকোনো গ্রাফ কে কিছু সংখ্যক biconnected subgraph বা biconnected component এ ভাগ করতে পারি যেন সব edge কোন না কোন ভাগে পরে। খেয়াল করো, একটি edge একাই কিন্তু একটি BCC হতে পারে কারণ এর এক মাথার নোড মুছে ফেললে কিন্তু কেউ disconnected হয় না। আমরা চিত্র ৮.১ তে একটি গ্রাফকে BCC তে ভাঙ্গিয়ে দেখালাম।

[১]যারা implication sign এর মানে জানো না তাদের জন্য বলি, $a \rightarrow b$ কেবল মাত্র (a = 1, b = 0) এর জন্য false এছাড়া সবসময় true. অর্থাৎ একে এভাবে ভাবতে পারঃ a সত্য হলে b ও সত্য হবে, a মিথ্যা হলে b যা খুশি তাই হোক যায় আসে না।

প্রতিটি রং একেকটি component. এখানে খেয়াল করো একটি নোড কিন্তু একাধিক component এর অংশ হতে পারে। কেন? কারণ দেখো নীল রং এর component এ তুমি যদি 1 নোডটি মুছে ফেল তাহলে বাকি নোডগুলি connected থাকে। আবার লাল অংশেও একই কথা সত্য। তবে তুমি যদি লাল আর নীল এই দুই অংশকে একত্র করে যদি বলতে এটা একটা BCC তাহলে তা কিন্তু সত্য হতো না কারণ আমরা 1 কে মুছে ফেললে গ্রাফটি disconnected হয়ে যেতো। এর মানে একটি নোড একাধিক BCC এর অংশ হতে পারে কিন্তু একটি edge কেবল মাত্র একটি BCC এরই অংশ। আশা করি এটা বলার অপেক্ষা রাখে না যে আমরা প্রতিটি component কে যত বড় করা সম্ভব তত বড় করতে চেষ্টা করি। তোমরা চিত্র ৮.১ এ যদি বলতে প্রতিটি edge আলাদা আলাদা component, হ্যা কথা ঠিক কিন্তু এই যে বললাম প্রতিটি component কে আমরা বড় করার চেষ্টা করি, সে জন্য আমাদের BCC হবে চিত্রের মত।



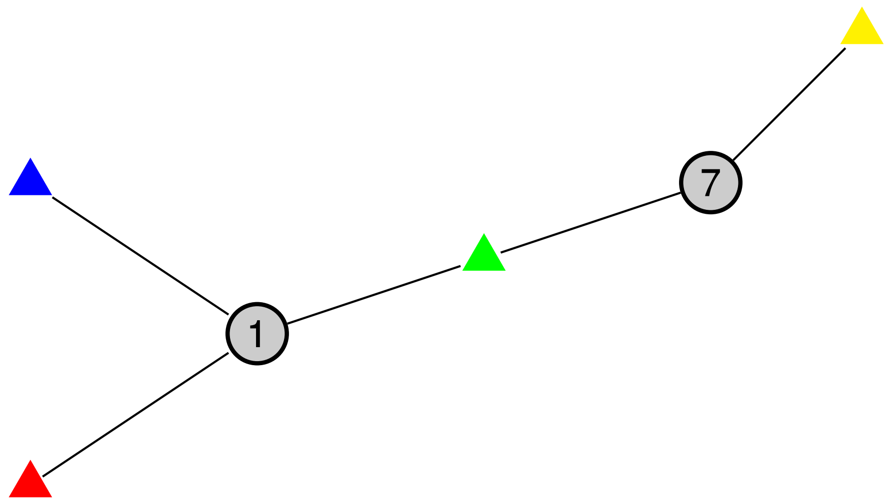
চিত্র ৮.১: Biconnected algorithm

অনেক সময় সমস্যা ভেদে তোমাকে হয়তো গ্রাফকে component এ ভাগ করতে হয় যেন একই component এর কোন edge মুছে ফেললে গ্রাফটি disconnected না হয়ে যায়। সেক্ষেত্রে আমাদের এতো কষ্ট করতে হবে না, শুধু articulation bridge বা edge বের করে একটি BFS বা DFS চালিয়ে দিলেই আমাদের সব component বের হয়ে যাবে। আমরা প্রতিটি unvisited নোড এর জন্য BFS বা DFS চালাব এবং articulation bridge ব্যতিত অন্য সকল edge দিয়ে আমরা traverse করব।

BCC এর সাথে সম্পর্কিত আরেকটি জিনিস আছে আর তাহলো Block cut vertex tree. প্রতিটি undirected graph কে যেমন আমরা BCC তে ভাগ করতে পারি ঠিক তেমনি সেই BCC কে আমরা একটা tree আকারে সাজাতে পারি যেখানে tree এর নোডগুলি হল BCC এর cut vertex (articulation node) সমূহ এবং block বা component গুলি। যদি কোন একটি cut vertex একটি block এর অংশ হয় তাহলে তাদের মাঝে edge থাকবে। তাহলে যেকোনো connected undirected graph এর জন্য আমরা একটি tree পাব। যেমন চিত্র ৮.১ এর block cut vertex tree হবে চিত্র ৮.২ এর মত। বেশ কিছু সমস্যায় আমরা দেখব যে এই tree বেশ কাজে লাগে।

৮.১৩ Flow সম্পর্কিত অ্যালগরিদম

নিঃসন্দেহে flow একটি কঠিন টপিক। এর কোড বেশ সহজ কিন্তু এটি ঠিক মত বুঝা বা একে রপ্ত করা খুবই কঠিন। দেখা যাক তোমাদের এর মূল concept টা বুঝাতে পারি কিনা।



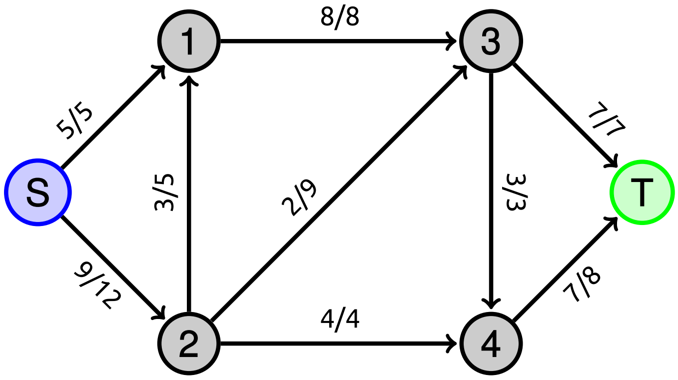
চিত্র ৮.২: Biconnected algorithm

৮.১৩.১ Maximum flow

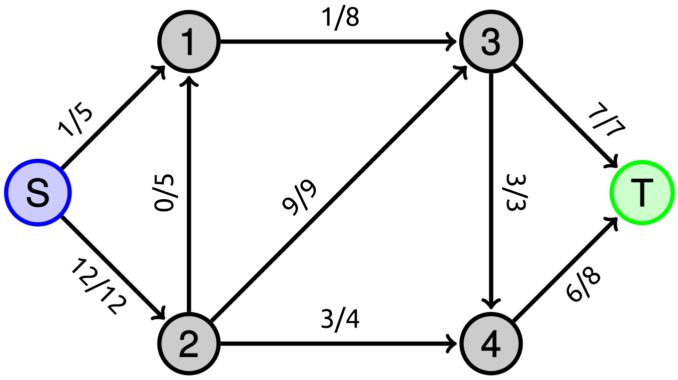
এই প্রবলেমে কিছু নোড এবং কিছু directed edge থাকবে। নোড সমূহের মাঝে দুইটি বিশেষ নোড থাকবে। একটি হল source যা সাধারণত আমরা S দিয়ে প্রকাশ করি আর আরেকটি হল sink যা আমরা সাধারণত T দ্বারা প্রকাশ করি। Directed edge সমূহ এর সাথে একটি সংখ্যা থাকবে, একে আমরা weight বলব না, একে আমরা বলব capacity. এখন মনে করো source এ unlimited তরল পদার্থ আছে, আর sink এ যেকোন সময়ে unlimited তরল প্রবেশ করতে পারে। অন্যান্য যে-সব edge এর কথা বললাম তাদের একেকটি পাইপ মনে করো যাদের capcacity দেয়া আছে অর্থাৎ কোন একক সময়ে সর্বোচ্চ কত তরল ঐ পাইপ দিয়ে প্রবাহিত হতে পারে তা দেয়া আছে। তোমা-দের বলতে হবে কোন একক সময়ে কি পরিমাণ তরল source হতে sink এ প্রবাহিত হতে পারে? তোমরা মনে করতে পারো যে, কোন পাইপ দিয়ে তরল যেতে কোন সময় লাগে না তবে একক সম-য়ে তার capacity এর থেকে বেশি তরল প্রবাহিত যেন না হয়। এই সমস্যাটাই হল maximum flow বা সংক্ষেপে maxflow. কিছু উদাহরণ দেখা যাক। ধরা যাক, S হতে A তে একটি edge আছে যার capacity 10, আর A হতে T তে একটি edge আছে যার capacity 20. তাহলে এই গ্রাফে maximum flow হবে 10. যদি আগের গ্রাফে capacity ঠিক উলটো হতো তাহলেও কিন্তু maximum flow হতো 10. এবার একটু বড় উদাহরণ দেখা যাক চিত্র ৮.৩ এ। খেয়াল করলে দেখবে যে এখানে edge এ f/c আকারে কিছু লিখা আছে। এখানে প্রথম সংখ্যা অর্থাৎ f/c এর f হল কত flow হচ্ছে, আর c হল কত capacity. এখন চিত্রের মত ছাড়া আর অন্য কোন ভাবে flow দিলেও তুমি 14 এর থেকে বেশি flow দিতে পারবে না।

তাহলে আশা করি maxflow কি জিনিস তা বুঝা গেছে? এখন আসো কঠিন অংশে। কেমনে maxflow সমস্যা সমাধান করা যায়? সহজ হতে শুরু করা যাক। একটা খুব সাধারণ উপায় হল যতক্ষণ আমরা S হতে T তে এমন একটা path পাব যা দিয়ে কিছু পরিমাণ flow দেয়া যায় সেই path এ flow দেয়া। এবং যখন আর এমন কোন path পাবা না তাকে maximum flow বলা। কিন্তু এটা কাজ করবে না। চিত্র ৮.৩ এর গ্রাফে আমরা যদি অন্য ভাবে flow দেই তাহলেই দেখতে পারবে যে মোট flow 14 না অথচ আর কোন flow দিতে পারছ না। মনে করো, $S-2-3-T$ দিয়ে 7, $S-2-3-4-T$ দিয়ে 2, $S-1-3-4-T$ দিয়ে 1 এবং $S-2-4-T$ দিয়ে 3 flow যদি দ্বাও তাহলে মোট flow হয় $7+2+1+3=13$ এবং আমাদের গ্রাফ দেখতে চিত্র ৮.৪ এর মত হবে। এখানে দেখো আর কোন path পাবে না যা দিয়ে তুমি আরও flow দিতে পারো কিন্তু এর flow হল 13. অর্থাৎ এটাকে বলতে পারো এই মেথডের একটি local maxima কারণ আমরা চিত্র ৮.৩ এ দেখে এসেছি যে অন্তত 14 পাওয়া যায়।

তাহলে কেমনে আমরা maxflow সমস্যা সমাধান করব? এই সমাধান বুঝতে হলে আমাদের



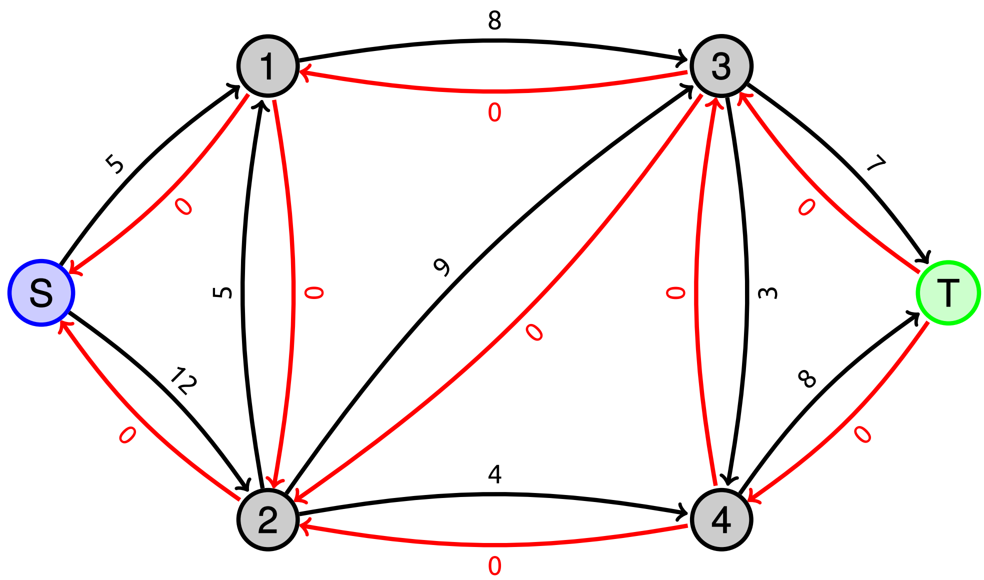
চিত্র ৮.৩: Maximum flow



চিত্র ৮.৪: Maximum flow: local maxima কিন্তু maximum নয়

গ্রাফে কিছু পরিবর্তন করতে হবে। প্রথম পরিবর্তন হবে গ্রাফের representation এ। এতক্ষণ কোন edge এ f/c বলতে আমরা flow ও capacity বুঝিয়েছি। এখন থেকে আমরা দুইটি সংখ্যা দিয়ে f/c আকারে প্রকাশ না করে মাত্র একটি সংখ্যা cf দিয়ে প্রকাশ করব। cf হল residual capacity বা গাণিতিক ভাবে $c - f$. Residual capacity মানে হল আরও কত flow যেতে পারে। যেমন চিত্র ৮.৩ এ 2 হতে 3 নোডের মাঝের edge এ আছে $2/9$ এটি নতুন representation এ হবে $9 - 2 = 7$ আবার S হতে 1 এর মাঝের representation $5/5$ হতে পরিবর্তন হয়ে হবে 0. দ্বিতীয় পরিবর্তন হল গ্রাফের প্রতিটি edge এর জন্য আমাদের আরও একটি edge থাকবে যার direction হবে সম্পূর্ণ উলটো। অর্থাৎ আমাদের গ্রাফে যদি 1 হতে 3 এ কোন edge থাকে তাহলে আমাদের 3 হতে 1 এর দিকে একটি নতুন edge যোগ করতে হবে। এখন প্রশ্ন হল এই edge এর initial cf কি হবে? তার আগে সকল মূল edge এর initial cf কি হবে তা পরিষ্কার করি। এটি হবে c, কারণ প্রথমে কোন flow থাকবে না তাই $f = 0$ এবং $cf = c - f = c - 0 = c$. এখন এর বিপরীত edge এর কথা ভাবা যাক, এর initial লেবেল হবে 0. কারণ আমাদের এর ভিতর দিয়েও কোন initial flow যাবে না এবং এর capacity ও 0. তাহলে ফট করে আমরা চিত্র ৮.৫ এ আমাদের গ্রাফ এর initial রূপ পরিবর্তিত representation এ দেখি। আশা করি বুঝতে পারছ যে লাল edge গুলি হল উলটো edge.

এখন যা করতে হবে তাহলো এমন একটি path খুঁজে বের করতে হবে যার প্রতিটি edge এই কিছু পরিমাণ residual capacity থাকে। অর্থাৎ তোমরা S হতে শুরু করবা এবং একটি BFS বা DFS করে T পর্যন্ত যাবার চেষ্টা করবা সেসব edge ব্যবহার করে যাদের $cf > 0$. এই path কে বলা হয় augmenting path. এখন যা করতে হবে তাহলো এই path এর সব edge এর cf এর মাঝে



চিত্র ৮.৫: Maximum flow: পরিবর্তিত representation এ initial রূপ

minimum cf বের করতে হবে। আমরা এই পুরো path দিয়ে এই পরিমাণ flow করাবো। মনে করো এই minimum cf হল x. তাহলে যা করতে হবে তাহলো এই path এর সকল edge এর cf কে x পরিমাণ কমাতে হবে। কারণ cf বলে কি পরিমাণ আরও flow করানো যাবে আর যেহেতু আমরা x পরিমাণ flow করছি সেহেতু residual capacity x পরিমাণ কমাতে হবে। এটুকু তো বুঝা গেল কিন্তু এর পর যা বলব সেটাই হল মূল সমস্যা। সেটা হল, আমরা যেই যেই edge এর cf কে x কমিয়েছি তার উলটো edge এর cf কে x বাড়াতে হবে। অর্থাৎ যদি augmenting path কোন edge দিয়ে যায় তাহলে তার উলটো edge এর cf বাড়াতে হবে। এই পুরো প্রসেস কে augment করা বলে। এখন কথা হল আমরা কেন উলটো দিকের edge গুলোর cf বাড়াবো? এটা আসলে বলে অতটা ভাল করে বুঝানো সম্ভব না, এটা কিছুটা অনুভব এর বিষয়। তবুও বলি, যদি আমরা a হতে b এর দিকে flow দেই এর মানে হল S হতে কোন ভাবে আমরা a তে এসেছি এর পর $a - b$ edge দিয়ে আমরা b তে এসেছি, এর পর কোন ভাবে b হতে T তে গিয়েছি। ধরা যাক এই path এর নাম P. এখন কথা হল আমরা তো চাইলে $a - b$ না গিয়ে $a - c$ ও যেতে পারতাম তাই না? এবং হয়তো ঐভাবে গেলে আমরা optimal সমাধান পেতাম। কিন্তু আমরা তো $a - b$ দিয়ে চলে এসেছি। কি করা যায়? আচ্ছা মনে করো পরের ধাপে আমরা S হতে কোন ভাবে b তে এসেছি। এখন যদি আমরা a হতে T পর্যন্ত কোন path পাই তাহলে এই নতুন path আর P মিলে একটা নতুন flow দিতে পারি। কেমনে? আমরা নতুন path এ S হতে b পর্যন্ত আসব, এর পর P যে path এ b হতে T তে গিয়েছে সেই path এ এই নতুন augmenting path যাবে, আর আগের path S হতে a তে এসে $a - b$ এর মধ্য দিয়ে না গিয়ে a হতে T পর্যন্ত যাবার যেই নতুন path আমরা পেয়েছি সেই path এ যাবে। অর্থাৎ আগে আমরা a হতে b তে যেই flow দিয়েছিলাম সেটা আমরা বলতে পারো cancel করতেছি। বা এভাবেও ভাবতে পারো যে S হতে b হয়ে a তে গিয়ে এর পর T তে flow দিলাম। এটা সম্ভব হবে যদি উলটো দিকের cf বাড়ানো হয়। তাহলে এটাই আমাদের অ্যালগোরিদম। তবে একটা জিনিস তুমি যদি augmenting এর জন্য DFS ব্যবহার করো তাহলে আসলে অনেক সময় লেগে যাবে। এটি maxflow এর Ford Fulkerson অ্যালগোরিদম নামে পরিচিত। এবং এতে তুমি চাইলে এমন একটি গ্রাফ বানাতে পারো যেন তোমার মোট flow এর সমান বার augment করতে হতে পারে। ফলে যদি নোড সংখ্যা কমও হয় কিন্তু capacity এর উপর নির্ভর করবে এর runtime. এর time complexity $O(E \times answer)$. তবে তুমি যদি BFS ব্যবহার করো augmenting path বের করার জন্য তাহলে এর runtime হবে $O(VE^2)$ যেখানে V হল vertex এর সংখ্যা আর E হল

edge এর সংখ্যা। আর একে Edmonds Karp অ্যালগোরিদম বলা হয়।

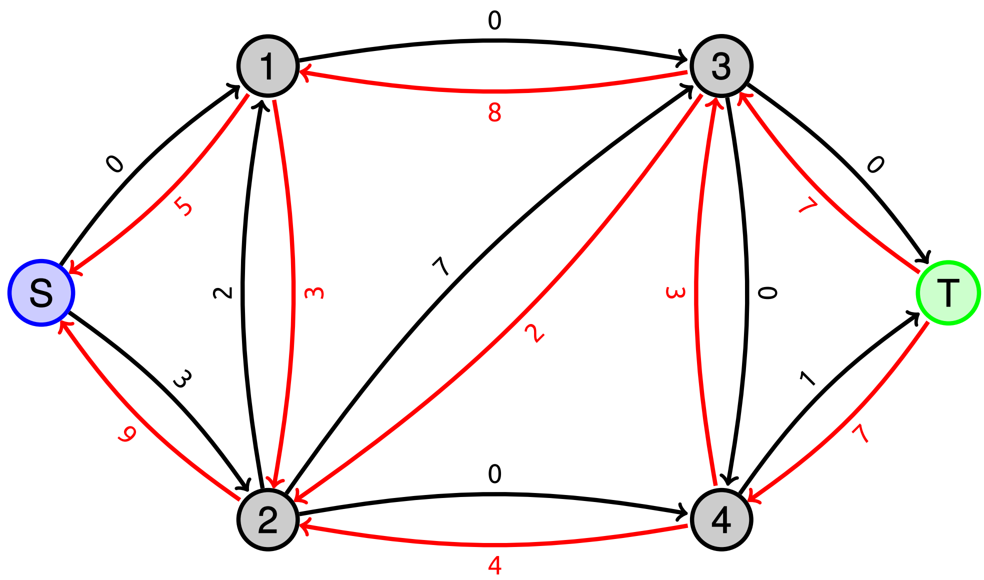
একটা ছোট কাজ করলেই আমরা এর runtime প্রায় $O(V^2E)$ করতে পারি। আমাদের যা করতে হবে তাহলো, প্রথমে একটি পূর্ণ BFS চালাতে হবে[১]। পূর্ণ BFS চালানোর সময় BFS এর layer to layer এর information রাখতে হবে। অর্থাৎ BFS কে তো একটি ট্রি হিসাবে কল্পনা করা যায়। কোন নোড কোন layer এ আছে সেই information রাখতে হবে। এর পর একটি backtrack বা DFS এর মত কাজ করতে হবে। আমরা S হতে শুরু করব এবং প্রতিবার পরের layer এর কোন নোডে যাবার চেষ্টা করব (যদি residual capacity ধনাত্মক হয়)। যদি আমরা T তে পৌছাই তাহলে এই path টা augment করব। এভাবে চলতে থাকবে। একে Dinic's algorithm বা Dinic's blocking algorithm বলা হয়। এর থেকেও ভাল অ্যালগরিদম সাধারণত দরকার হয় না।

আরও কিছু বলার আগে ছোট খাটো দুই একটা কথা জেনে রাখা ভাল। প্রথমত আমি এখানে বুঝানোর সুবিধার জন্য ইচ্ছা করে directed graph নিয়েছিলাম। কিন্তু তুমি যদি বই পড় বা প্রবলেম দেখো তাহলে দেখবে বেশির ভাগ সময় undirected edge এ capacity দেয়া থাকে। এসময় যা করতে হয় তাহলো দুই দিকের জন্য দুইটি directed edge তোমাকে লাগাতে হবে। তুমি চাইলে এই দুইটি দিয়েই কাজ সাড়তে পারো বা এই দুইটির উলটো দিকে আরও দুইটি edge লাগাতে পারো। আরেকটি বিষয় হল তুমি কেমনে এই গ্রাফের edge বা cf রাখবে? তুমি চাইলে adjacency matrix রাখতে পারো। তাতে সমস্যা হল BFS এর সময় আর তোমার time complexity $O(()$E) থাকবে না $O(V^2)$ হয়ে যাবে। আবার তুমি যদি adjacency list করো তাহলে উলটো দিকের cf পরিবর্তন করা আবার আরেক ঝামেলা। তুমি চাইলে একই সাথে adjacency list ও matrix রাখতে পারো। তবে আমি vector দিয়ে adjacency list বানিয়ে এই কোড করে থাকি। মনে করো তোমাকে x হতে y এর দিকে একটি edge দেয়া হল। প্রথমে x আর y এর adjacency list এ কতগুলি element আছে তা দেখে নাও। ধরা যাক এই দুইটি সংখ্যা যথাক্রমে $size_x$ এবং $size_y$. এর মানে তুমি x হতে y এই edge টা যখন আমাদের data structure এ রাখবা একটি edge থাকবে x এর list এ sz_x তম স্থানে আর তার উলটো edge থাকবে y এর sz_y স্থানে। তোমাকে যা করতে হবে তাহলো একটি edge যখন insert করবে তখন 3 টি information রাখবেঃ other end, residual capacity এবং index of the reverse edge. শেষ। এখন তুমি কোন edge এর cf কমানোর সময় খুব সহজেই তার উলটো দিকের edge এ গিয়ে তার cf এর মান পরিবর্তন করে ফেলতে পারো। তাহলে চিত্র ৮.৫ এর maxflow রূপটা চিত্র ৮.৬ এ দেখানো হল।

৮.১৩.২ Minimum cut

যেকোনো optimization সমস্যার একটি dual প্রবলেম থাকে। অর্থাৎ যদি একটি প্রবলেম থাকে যেখানে আমাদের কিছু maximize করতে হবে তাহলে আমরা সেই সমস্যাকে অন্যভাবে দেখতে পারি যেখানে কোন কিছুকে minimize করতে হয়। প্রধান সমস্যা বা primal এর যা উত্তর হয় dual এরও একই উত্তর হয়। আমরা কিছুক্ষণ আগে maxflow প্রবলেম দেখলাম। সেখানে আমরা flow কে maximize করেছিলাম। এখন কথা হল এর dual প্রবলেম কি? এটাই হল minimum cut বা সংক্ষেপে mincut. Mincut প্রবলেমটা হল, আগের মতই source বা sink থাকবে এবং edge সমূহে capacity থাকবে। তোমাকে কিছু edge ডিলিট করে source এবং sink কে disconnected করতে হবে যেন source এর component থেকে sink এর component এ যাওয়া edge সমূহের capacity এর যোগফল বিয়োগ sink এর component থেকে source এর component এ যাওয়া edge সমূহের capacity এর যোগফল সর্বনিম্ন হয়। খেয়াল করো আমি বলেছি capacity এর যোগফল residual capacity এর না কিন্তু! যেমন চিত্র ৮.৫ এ যদি আমরা $(S,1,2)$ কে একটা component আর $(3,4,T)$ কে আরেকটা component ধরি তাহলে এই cut এর cost হবে $8+4+9=21$. যদি $(S,1,2,3,4)$ এক component আর বাকি (T) এক component হয় তাহলে cost হবে $7+8=15$. যদি $(S,1)$ একটি আর $(2,3,4,T)$ আরেকটি হয় তাহলে cost হয় $8-5+12=15$. Optimal cut টা হবে $(S,1,2,3)$ এবং $(4,T)$ এর ক্ষেত্রে কারণ এর cost $7+3+4=14$ যা maxflow এর সমান। তুমি কাগজে কলমে maxflow বা mincut বের করার

[১]পূর্ণ বলার কারণ হল এর আগের বার আমরা S হতে BFS শুরু করতাম আর T তে পৌঁছে গেলেই চলত। কিন্তু এবার যতক্ষণ না সকল vertex ই visited হচ্ছে ততক্ষণ আমাদের BFS চালাতে হবে।



চিত্র ৮.৬: Maximum flow: maxflow তে গ্রাফ এর ছবি

সময় নিশ্চিত হবার জন্য এই দুইটিই বের করে দেখতে পারো, যদি তারা সমান হয় তার মানে তোমার উত্তর ঠিক আছে।

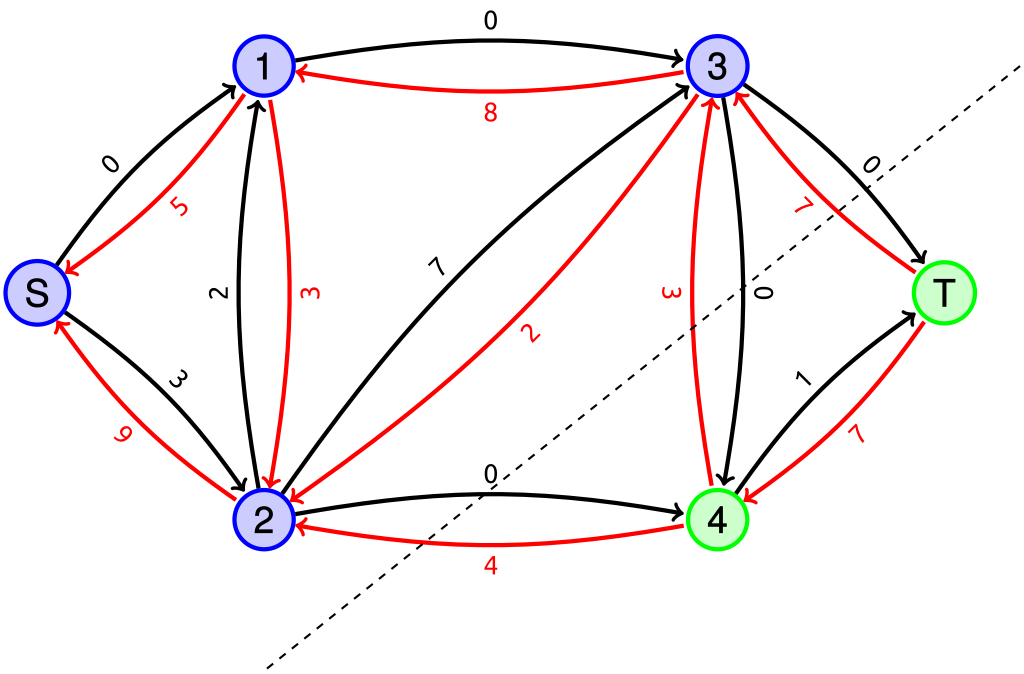
প্রশ্ন হল mincut কেমনে বের করা যায়? মানে mincut এর উত্তর তো তোমরা maxflow এর উত্তর থেকেই পেতে পারো কিন্তু কিভাবে দুইটি component এ ভাগ করলে তুমি mincut পাবে তা কেমনে বের করা যায়? সহজ, maxflow শেষে তোমরা source হতে শুরু করে একটি BFS চালাবে, BFS এর সময় তুমি সেইসব edge দিয়ে যাবে যাদের এখনও কিছু residual capacity বাকি আছে। ব্যাস তাহলে visited node গুলি একটি component আর unvisited node গুলি অপর component তৈরি করবে। যেহেতু maxflow শেষে আমরা এই কাজ করছি সেজন্য অবশ্যই আমরা T কে visit করতে পারব না (করা গেলে তো আবার flow করা যেতো)। চিত্র ৮.৬ এ এই BFS চালালে visited হওয়া নোড সমূহ কে চিত্র ৮.৭ এ দেখানো হল। এটা আশা করি বুঝা যাচ্ছে যে নীল নোডগুলি একদিক এবং সবুজ নোডগুলি আরেক দিক। যদি তুমি এই দুই সাইডের মাঝে capacity যোগ বিয়োগ করো তাহলে পাবে $7 + 3 + 4 = 14$. যদিও এই চিত্রে capacity দেখানো নেই কিন্তু তুমি আগের চিত্র ৮.৩ এর সাথে তুলনা করে edge গুলির capacity দেখে নিতে পারো।

৮.১৩.৩ Minimum cost maximum flow

অনেক সময় edge সমূহের cost ও দেয়া থাকে এবং আমাদের কাছে চাওয়া হয় যে সবচেয়ে কম খরচে সবচেয়ে বেশি সংখ্যক flow কেমনে দেয়া যায়। এর সমাধান maxflow এর মতই। শুধু পার্থক্য হল এখানে আমরা S হতে T তে যাবার path, BFS বা DFS করে বের করব না। বরং bellman ford বা repeated dijkstra করে বের করব। আর প্রতিবার flow এর cost গুলি যোগ করব। তাহলেই mincost maxflow সমাধান হয়ে যাবে।

৮.১৩.৪ Maximum Bipartite Matching

প্রথমে আমাদের জানতে হবে bipartite graph কি জিনিস। এটি এমন একটি গ্রাফ যেখানে নোডগু-লিকে দুই ভাগে ভাগ করা যায় যেন প্রতিটি ভাগের নোডগুলির মাঝে কোন edge না থাকে। যা edge আছে তা হবে এই দুই ভাগের মাঝে। এর বাস্তব উদাহরণ এরকম হতে পারে, মনে করো তোমার কাছে



চিত্র ৮.৭: Minimum cut

কিছু বল আর কিছু বাক্স আছে। যদি বল আর বাক্সকে তুমি নোড দিয়ে প্রকাশ করো তাহলে তুমি সেই সব বল আর বাক্স এর মাঝে edge দিবে যেন সেই বল সেই বাক্স এর মাঝে fit করে। এই গ্রাফটি অবশ্যই একটি bipartite graph কারণ এখানে দুইটি বল বা দুইটি বাক্সের মাঝে কোন edge নেই। এখন যদি তোমাকে জিজ্ঞাসা করা হয় তুমি সর্বোচ্চ কতগুলি বল কোন বাক্সতে ভড়তে পারবে? মানে অবশ্যই তুমি একটি বলকে দুইটি বাক্সে বা কোন বাক্সে দুইটি বল রাখতে পারবে না। এই সমস্যাটাই হল maximum bipartite matching. অর্থাৎ তুমি সর্বোচ্চ কত গুলি edge নির্বাচন করতে পারবে যেন কোন নোডে একাধিক নির্বাচিত edge না থাকে।

Maxflow ব্যবহার করে এই সমস্যা খুব সহজেই সমাধান করা যায়। মনে করো নোডগুলি যেই দুই ভাগে বিভক্ত তাদের নাম L ও R (left, right এর সংক্ষিপ্ত রূপ)। এখন তুমি একটি source S নাও এবং S হতে L এর সকল নোডের 1 capacity এর edge দাও। একই ভাবে একটি sink T নাও এবং R এর সকল নোড হতে T তে 1 capacity এর edge দাও। আর L ও R এর মাঝের edge গুলির 1 capacity করে দিতে হবে। তাহলে এই গ্রাফের maxflow ই হল maximum bipartite matching. কেন এটা সঠিক তা আশা করি বলতে হবে না।

যদিও আমরা maxflow ব্যবহার করে সমাধান করতে পারি কিন্তু আসলে আমরা এসব অতিরিক্ত নোড না লাগিয়েই সমাধান করতে পারি। আমরা এখন যেই সমাধান এর কথা বলব সেটি আসলে Edmond Karp অ্যালগোরিদম এর bipartite graph ভার্সন মনে করতে পারো। প্রথমে দুইটি অ্যারে নাও $matchL$ এবং $matchR$ এবং এদের -1 দিয়ে initialize করো। যদি L এ থাকা একটি নোড i হয় তাহলে $matchL[i]$ হল সেই নোড এর সাথে match হওয়া R সাইড এর নোড। $matchR$ একই রকম জিনিস। এখন একে একে L থেকে একটি করে নোড নাও (ধরো নোডটি হল x) আর একটা কাজ করো। কাজটা চাইলে BFS এর মত করে করতে পারো বা DFS এর মত করে করতে পারো। আমি DFS এর মত করে করি কারণ এতে কোড বেশ ছোট হয়। আমি এখানে DFS করে কেমনে করতে হবে সেটা বর্ণনা দিচ্ছি। যেহেতু আমরা DFS করব তাই তোমাদের একটি visited এর অ্যারে নিতে হবে এবং একে 0 দ্বারা initialize করতে হবে (আমরা কিন্তু এই initialization এবং DFS L এর প্রতিটি vertex এর জন্য করব)। এখন তুমি x হতে DFS শুরু করো। তুমি যেই নোড এর জন্য DFS এ আছো (ধরা যাক y, অর্থাৎ প্রথমে y এর মান হবে x) তার neighbor দের একে একে দেখো। আমরা এমন ভাবেই কাজ

করব যেন y সবসময় L এর হয়ে থাকে। সুতরাং এর neighbor হবে R এর, ধরা যাক এরকম একটি neighbor হল z. এখন তোমাকে দেখতে হবে $matchR[z]$ কি -1 কিনা। যদি -1 না হয় তাহলে $matchR[z]$ এর DFS কল করতে হবে, তবে কল করার আগে দেখে নিয়ো যে এই নোডটি আগেই visited কিনা। যদি visited হয় তাহলে আর কল করার দরকার নেই। আর যদি -1 হয় এর মানে একটি augmenting path পেয়েছ। তখন তোমাকে $matchL[y] = z$ এবং $matchR[z] = y$ করতে হবে এবং 1 return করতে হবে। কিছু ক্ষণ আগে কিন্তু তুমি $matchR[z]$ -1 নাহলে DFS কল করেছিলে। যদি এই DFS তোমাকে 1 return করে তার মানে তুমি augmenting path পেয়েছ এবং আগের মতই $matchL[y] = z$ এবং $matchR[z] = y$ করবে এবং 1 return করবে। আর যদি দেখো y এর জন্য সব z শেষ কিন্তু তাও তুমি augmenting path পাও নাই তাহলে 0 return করো। এই DFS এর কোডটা দেখতে কোড ৮.৬ এর মত হবে।

কোড ৮.৬: bpm dfs.cpp

```
int dfs(int y) {
    for (int i = 0; i < V[y].size(); i++) {
        int z = V[y][i];
        if (matchR[z] == -1 || (!visited[matchR[z]] && ↩
            dfs(matchR[z]))) {
            matchL[y] = z;
            matchR[z] = y;
            return 1;
        }
    }
    return 0;
}
```

একদম শুরুতে তো x এর জন্য DFS কল করেছিলে। যদি এই কল 1 return করে এর মানে তোমার matching 1 বেড়েছে বা তুমি চাইলে $matchL$ এ দেখতে পারো কত গুলির জন্য -1 নেই এভাবে তুমি maximum matching কয়টা তা বের করতে পারো আর $matchL$ দেখে এও বলতে পারো যে কার সাথে কে match করেছে। এর time complexity হল $O(VE)$ কারণ তুমি V বার DFS চালাচ্ছ।

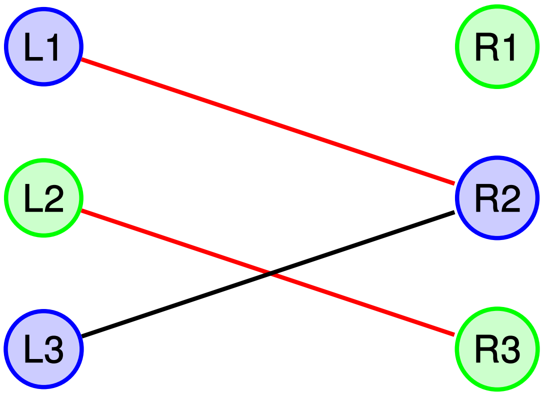
তবে তুমি যদি এই algorithm কে একটু modify করো তাহলে $O(E\sqrt{V})$ time complexity এর অ্যালগরিদম পেয়ে যাবে যার নাম Hopcroft Karp অ্যালগোরিদম। যা করতে হবে তাহলো এক-টি একটি করে নোড নিয়ে DFS না করে, L এর সব unmatched নোড নিয়ে প্রথমে BFS করো এবং BFS tree তে L এর সকল নোড এর depth লিখে রাখ। এর পর একটি DFS চালিয়ে শুধু মাত্র পরের layer এর নোড এ যেতে চেষ্টা করো এবং দেখো একটা augmenting path পাও কিনা। কিছুটা dinic এর মত। এটিই তোমাকে $O(E\sqrt{V})$ time complexity দিবে।

৮.১৩.৫ Vertex cover ও Independent set

Maximum bipartite matching প্রবলেম এর সাথে জড়িত দুইটি সমস্যা হল vertex cover এবং independent set. Vertex cover বলে এমন কিছু নোড select করতে যেন সকল edge এর কোন একটি মাথা যেন অবশ্যই নির্বাচিত vertex গুলির মাঝে একটি হয়। যেহেতু আমরা চাইলেই সকল নোড নির্বাচন করতে পারি তাই আমাদের লক্ষ্য হল সবচেয়ে কম সংখ্যক নোড নির্বাচন করা। এ জন্য এই সমস্যাকে বলা হয় minimum vertex cover. Independent set ঠিক এর উলটো সমস্যা। এই সমস্যায় বলে এমন কিছু নোড নির্বাচন করতে যেন তাদের মাঝে কোন edge না থাকে। যেহেতু আমরা চাইলেই মাত্র একটি নোড নির্বাচন করতে পারি সেহেতু আমাদের লক্ষ্য হল সবচেয়ে বেশি সংখ্যক নির্বাচন করা, সেজন্য একে বলা হয় maximum independent set. কথা হল general graph এ এই দুইটি সমস্যা NP অর্থাৎ আমরা জানি না এদের ভাল কোন সমাধান আছে কিনা। তবে

bipartite graph এ এদের খুব সুন্দর সমাধান আছে। সেজন্য আমরা এখানে bipartite graph এ এই দুইটি সমস্যা দেখব।

প্রথমত বলে নেই যে maximum bipartite matching এবং minimum vertex cover সমস্যা হল dual. অর্থাৎ দুইটির উত্তর সমান হবে। এখন কথা হল আমরা এরকম একটি vertex set কেমনে বের করব? খুব একটা কঠিন না। আমার কাছে সহজ লাগে এই ক্ষেত্রে প্রবলেমটিকে maxflow এবং mincut এর দৃষ্টিকোণ থেকে দেখলে। ঠিক mincut এর মত তুমি S অংশ এবং T অংশ বের করো। অবশ্যই L এর কিছু অংশ হবে S এর অংশে এবং বাকি অংশ T এর অংশে। মনে করি এরা হল যথাক্রমে L_S এবং L_T. একই ভাবে আমরা R_S ও R_T পাব। তাহলে vertex cover হবে $L_T \cup R_S$. অদ্ভুত লাগলেও এটি সত্য। একটা উদাহরণ দেখা যাক।



চিত্র ৮.৮: Bipartite matching, Vertex cover ও Independent set

চিত্র ৮.৮ এ লাল edge দিয়ে matching edge বুঝানো হয়েছে। এখন তুমি mincut এর অ্যালগরিদম চালালে নীল নোড গুলি S অংশ আকারে চিহ্নিত হবে এবং সবুজ নোড গুলি T অংশ হিসাবে। সুতরাং আমাদের vertex cover হবে $L_T \cup R_S = \{L2, R2\}$. তোমরা হয়তো জিজ্ঞাসা করতে পারো এই S ও T অংশ বের করার কি কোন সহজ উপায় নেই? Mincut ই করতে হবে? না অবশ্যই সহজ উপায় আছে। কল্পনা করো এটি যদি flow এর গ্রাফ হতো তাহলে source S হতে BFS করলে কোন কোন নোড এ যেতে? L এর সেসব নোড এ যাদের matching নেই। সুতরাং BFS এর জন্য একটি queue তে এই সব নোড ঢুকিয়ে ফেল। এখন চিন্তা করো mincut এর BFS এর সময় তুমি কখন L হতে R এ যেতে? যখন এই edge টা matching এ থাকবে না তখনই তার $cf = 1$ হবে এবং তুমি সেই edge ব্যবহার করতে। সুতরাং আমাদের এখনকার BFS এর সময় L এর কোন নোড এ থাকলে তার non-matching সব neighbor এ যাও। এখন আবার দেখো mincut এ R এর কোন নোড এ থাকলে কি করতে? এটি L সাইডে যার সাথে match করা সেখানে যেতে, সুতরাং এখনকার BFS এও তুমি তাই করবে। এভাবে BFS করলেই হবে।

এখন একটা মজার জিনিস খেয়াল করো, vertex cover হল এমন কিছু নোড এর সেট যেখানে সকল edge এর অন্তত একটি মাথা আছেই। তাহলে এমন কোন edge নেই যাদের দুইটি মাথাই vertex cover এর বাহিরে তাই না? অর্থাৎ vertex cover এর ঠিক complement হল independent set. যদি আমরা vertex cover কে minimize করি তাহলেই আমরা independent set কে maximize করতে পারব। অর্থাৎ আমাদের উপরের উদাহরণে $\{L2, R2\}$ ছিল minimum vertex cover, তাহলে $\{L1, L3, R1, R3\}$ হবে maximum independent set. শেষ!

৮.১৩.৬ Weighted maximum bipartite matching

Maxflow এর যেমন weighted ভার্সন ছিল (mincost maxflow) ঠিক তেমনই maximum bipartite matching এরও weighted ভার্সন আছে। এটিই weighted maximum bipartite matching. আমাদেরকে matching edge সমূহের cost কে maximum বা minimum করতে

বলে। দুইটিই একই কথা। যদি আমরা weight গুলিকে -1 দ্বারা গুণ করি বা একটি বড় সংখ্যা ধরো M থেকে সব edge এর cost বিয়োগ করি তাহলেই maximize প্রবলেম minimize প্রবলেম এ পরিবর্তিত হয়ে যায়। এখন আমরা এই minimum weighted maximum bipartite matching প্রবলেম কেমনে সমাধান করতে পারি? একটি সহজ উপায় হল একে mincost maxflow দিয়ে সমাধান করা। আরেকটি উপায় হল hungarian algorithm. এটি বেশ কঠিন মনে হয় আমার কাছে, আমি নিজেই এটা পারি না, তবে এর উপর topcoder এ আর্টিকেল আছে। তোমরা চাইলে সেই আর্টিকেল পড়ে দেখতে পারো। এর রান টাইম $\mathrm{O}(V^3)$.

অধ্যায় ৯

কিছু Adhoc টেকনিক

Adhoc প্রবলেম বলতে আমরা বুঝি আমাদের জানা কোন নির্দিষ্ট ক্যাটাগরিতে না পড়া সমস্যা। একই ভাবে adhoc টেকনিক হল এমন কিছু টেকনিক যা নির্দিষ্ট ভাবে কোন ভাগে ফেলা যায় না বা ফেলা কঠিন হয়। প্রোগ্রামিং কন্টেস্ট করতে গিয়ে প্রায়ই ব্যবহার করতে হয় এমন কিছু adhoc টেকনিক নিয়েই আমাদের এই চ্যাপটার। অনেকে হয়তো এখানে আলোচিত প্রবলেমগুলিকে বিভিন্ন ক্যাটাগরিতে(dp, greedy, math) ভাগ করতে চাইবে, কিন্তু আমি তা না করে adhoc সেকশনে রাখলাম।

৯.১ Cumulative sum টেকনিক

Cumulative sum মানে হল পর পর অনেক জিনিসের যোগফল। ধরা যাক তোমার কাছে একটি n সাইজ এর অ্যারে আছে। আমরা আমাদের সুবিধার জন্য অ্যারে কে 1-index ধরব। আসলে ঠিক 1-index না, কারণ আমরা মনে করব যে 0-index বলে একটা জায়গা আছে যা আপাতত অব্যবহৃত। অর্থাৎ mathematically লিখতে গেলে আমাদের অ্যারের সংখ্যা গুলি হল $A[1 \dots n]$ এবং $A[0]$ আপাতত অব্যবহৃত আছে। আমাদের এই A অ্যারে এর জন্য cumulative sum এর অ্যারে হবে S, যেখানে $S[i]$ হল A অ্যারে এর 1-index হতে i-index পর্যন্ত সংখ্যাগুলির sum. যেহেতু আমাদের A অ্যারে এর সাইজ n সেহেতু স্বাভাবিক ভাবেই S অ্যারে এর সাইজও $n.$ এখন কথা হল আমরা কিভাবে বা কত দ্রুত এই cumulative sum এর অ্যারে বের করতে পারব? সবচেয়ে সহজ উপায় মনে হয়, আমরা S এর প্রত্যেক index এ যাব (n বার) এবং i তম index এর জন্য cumulative sum বের করব। এভাবে করতে গেলে আমাদের time complexity দাঁড়াবে O(n^2). একটু চিন্তা করলে দেখবে যে আমরা কিছু কাজ বার বার করছি। যেমন ধরা যাক, আমরা $S[i-1]$ জানি অর্থাৎ আমরা $A[1 \dots i-1]$ এর যোগফল জানি। এখন এর পরে যখন $S[i]$ বের করতে যাব তখন কিন্তু আমাদের $A[1 \dots i]$ এর যোগফল পুরাটা নতুন করে বের করার দরকার নেই। বরং আগের $A[1 \dots i-1]$ এর যোগফল অর্থাৎ $S[i-1]$ এর সাথে শুধু $A[i]$ যোগ করলেই কিন্তু $S[i]$ পেয়ে যাবে। এভাবে আমরা মাত্র O(n) সময়েই পুরো cumulative sum এর অ্যারে বের করে ফেলতে পারব। অর্থাৎ আমরা $i = 1$ হতে $i = n$ পর্যন্ত লুপ চালাব এবং $S[i] = S[i-1] + A[i]$ করব। একটু খেয়াল করলে দেখবে যে $S[1]$ বের করার সময় আমরা $S[0]$ ব্যবহার করছি, সুতরাং আমাদেরকে $S[0] = 0$ সেট করতে হবে সবার শুরুতে। আমরা চাইলে লুপ 1 থেকে শুরু না করে 2 থেকে শুরু করতে পারতাম এবং $S[1] = A[1]$ সেট করতে পারতাম। তবে মনে হয় $S[0] = 0$ সেট করা অনেক বেশি intuitive এবং সহজবোধ্য। এই অ্যারে ব্যবহার করে কিন্তু আমরা খুব সহজেই অ্যারে এর a হতে b পর্যন্ত যোগফল বের করে ফেলতে পারি। আমাদের ফর্মুলা হবেঃ $S[b] - S[a-1]$.

এইতো গেল 1-dimension এ cumulative sum. আমরা চাইলে 2-dimension এও এই টেকনিক ব্যবহার করতে পারি। 2-dimension এ আমাদের প্রবলেম দাঁড়াবে কিছুটা এরকম- আমাদের কাছে একটি 2-dimension matrix থাকবে। আমাদেরকে (a, b) হতে (c, d) পর্যন্ত বিস্তৃত আয়ত-ক্ষেত্রের ভিতরের সংখ্যা সমূহের যোগফল বের করতে হবে। আগের মতই আমরা matrix এর প্রত্যেক

স্থান (i,j) এর জন্য $(1,1)$ হতে (i,j) পর্যন্ত আয়তক্ষেত্রের ভিতরের সংখ্যার যোগফল বের করব। যদি আমাদের এই যোগফল রাখার অ্যারে হয় S এবং আমাদের মুল অ্যারে হয় A তাহলে আমরা লিখতে পারিঃ $S[i][j] = S[i-1][j] + S[i][j-1] - S[i-1][j-1] + A[i][j]$. তুমি একটি ছবি আঁকলেই বুঝতে পারবে কেন এই ফর্মুলা সঠিক। এই ফর্মুলা ব্যবহার করে আমরা সকল prefix sum (আসলে prefix sum বলা ঠিক হচ্ছে না কিন্তু তোমরা আশা করি বুঝতে পারছ আমি কি বুঝাতে চাচ্ছি) মাত্র $\mathrm{O}(n^2)$ সময়েই বের করে ফেলতে পারব (ধরে নেই আমাদের মুল matrix টি $n \times n$ সাইজের)। এখন প্রশ্ন হল আমরা (a,b) হতে (c,d) পর্যন্ত বিস্তৃত আয়তক্ষেত্রের ভিতরের সংখ্যা সমূহের যোগফল কেমনে বের করব? খুব সহজঃ $S[c][d] - S[a-1][d] - S[c][b-1] + S[a-1][b-1]$. আশা করি বুঝতে পারছ যে, 3-dimension এর ক্ষেত্রে ফর্মুলাগুল দেখতে কেমন হবে!

৯.২ Maximum sum টেকনিক

৯.২.১ One dimensional Maximum sum problem

মনে কর তোমাদের কাছে একটা সংখ্যার অ্যারে আছে। তোমাকে এই অ্যারে থেকে maximum sum sub-array বের করতে বলা হল। অর্থাৎ তোমাকে এমন একটি sub-array বের করতে হবে যাতে করে সেই sub-array তে থাকা সংখ্যাগুলির যোগফল অন্যান্য যেকোনো sub-array তে থাকা সংখ্যাগুলির যোগফল থেকে বেশি হয়। মনে করা যাক আমাদের অ্যারে হলঃ 5, -10, 6, -2, 4, -10, 1. তাহলে আমাদের উত্তর হবে 8 যা 6, -2, 4 এই sub-array নিলে পাওয়া যাবে।[১] খুবই সহজ পদ্ধতি হতে পারে আমরা সকল sub-array এর জন্য একটি লুপ চালিয়ে তার যোগফল বের করব। কিন্তু এই পদ্ধতিতে আমাদের সময় লাগবে $\mathrm{O}(n^3)$. কারণ আমাদের মোট sub-array আছে $\mathrm{O}(n^2)$ টি এবং প্রতিটির যোগফল বের করতে সময় লাগবে $\mathrm{O}(n)$. আমরা চাইলেই এই ভিতরের যোগফল বের করার লুপ বাদ দিয়ে দিতে পারি। এর থেকে বরং প্রথম লুপ এর ভিতরে একটি ভ্যারিয়াবল ব্যবহার করে দ্বিতীয় লুপ এর ভিতর সেই ভ্যারিয়াবল এ সংখ্যার মান একে একে যোগ করতে থাকলেই আমাদের আর তৃতীয় লুপ এর দরকার পড়বে না। সুতরাং আমাদের সময় লাগবে $\mathrm{O}(n^2)$. আমরা কি কোন ভাবে consecutive sum এই টেকনিকটা ব্যবহার করতে পারি? খেয়াল করো, আমাদের consecutive sum এর অ্যারে তৈরি করতে সময় লাগবে $\mathrm{O}(n)$ আর প্রতিটি sub-array এর উপর লুপ চালিয়ে তার যোগফল আমাদের consecutive sum এর অ্যারে থেকে বের করতে সময় লাগবে $\mathrm{O}(n^2)$. সুতরাং আমাদের খুব একটা লাভ হচ্ছে না।

একটু চিন্তা করে দেখ আমরা consecutive sum এর অ্যারে ব্যবহার করে কেমনে কোন একটি sub-array এর মান বের করতে পারি? আমাদের i হতে j পর্যন্ত যোগফল হবে $S[j]-S[i-1]$. কোড এর কথা চিন্তা করলে জিনিসটা এমন যে, বাহিরে j এর লুপ চলবে, ভিতরে 1 হতে j পর্যন্ত i এর লুপ চলবে আর এই দুই লুপ এর মাঝে আমরা $S[j]-S[i-1]$ এর মান বের করব আর তাদের সবার মাঝে সর্বোচ্চটা বের করব। এখন বাহিরের লুপ j এর জন্য ভিতরের লুপ i এর $[1,j]$ এর মাঝে কোন মানের জন্য আমরা সর্বোচ্চ মানটা পাব? আমাদের এই চিন্তা ধারাটা একটু ভাল করে দেখ। আমরা এখানে j কে নির্দিষ্ট করে ফেলছি এবং জানতে চাচ্ছি যে i এর কোন মানের জন্য আমাদের উদ্দেশ্য সফল হবে। এখন তোমরাই বল যদি $S[j]-S[i-1]$ এ $S[j]$ কে নির্দিষ্ট করা হয় তাহলে $S[j]-S[i-1]$ কে সর্বোচ্চ করার জন্য তোমরা কোন $S[i-1]$ কে নিবে? অবশ্যই সর্বনিম্ন $S[i-1]$ কে বা $S[0\ldots j-1]$ এর মাঝের সর্বনিম্ন মান কে। আমরা কিন্তু চাইলেই বাহিরের লুপ j এর মাঝেই $1\ldots j-1$ এর সর্বনিম্ন মান বের করে ফেলতে পারি, আলাদা করে ভিতরের লুপ এর দরকার নেই। আসলে খেয়াল করলে দেখবে তুমি যদি $S[1\ldots j]$ এর মাঝের সর্বনিম্নটা বের করো তাহলে ক্ষতি নেই। সুতরাং আমরা এভাবে খুব সহজেই $\mathrm{O}(n)$ এ আমাদের maximum sum sub-array বের করে ফেলতে পারছি।

উপরে দেখান পদ্ধতিতে maximum sum sub-array বের করা অনেক বেশি intuitive মনে হয় আমার কাছে। তবে তোমরা অনেকেই হয়তো ক্লাসে বা অন্যান্য অ্যালগোরিদম এর বই এ অন্য একটি পদ্ধতি দেখেছ। সেই পদ্ধতি বলা অনেক সহজ কিন্তু সেটা কেন সঠিক সেটা বুঝা ওতটা সহজ মনে হয় না, সত্যি কথা বলতে সেটা কেন সঠিক এটা আমি নিজেকে কেমনে convince করব এটা

[১]অনেকে sub-array কে contiguous subsequence নামে চিনে।

চিন্তা করতে বেশ খানিকটা সময় ব্যয় করে ফেলেছি। যাই হক আগে পদ্ধতিটা বলে নেই। তোমরা একটি ভ্যারিয়েবল রাখবে ধরা যাক তার নাম $sum.$ এখন অ্যারে এর শুরু থেকে শেষ পর্যন্ত যাও। প্রত্যেক জায়গায় সেখানের মান sum এ যোগ করো। যদি দেখ sum এর মান negative হয়ে গেছে তাহলে sum কে 0 করে দাও। এভাবে প্রত্যেক জায়গায় sum এর যেই মান পাচ্ছ তাদের মাঝে maximum টাই আমাদের উত্তর। আমরা এই maximum বের করার জন্য max নামের একটি ভ্যারিয়েবল ব্যবহার করব এবং সেখানে এ পর্যন্ত পাওয়া সর্বোচ্চ sum রাখব। এখন কথা হল এটা কেন সঠিক? প্রথমত প্রতিটি স্থানে পৌঁছে sum এর মান হবে ওই পর্যন্ত শেষ হওয়া সকল sub-array এর maximum. induction এর মত করে চিন্তা করো। এই কথাটা 0 তে থাকা অবস্থায় সঠিক ছিল কারণ sum কে আমরা 0 দ্বারা initialize করেছি (এই কথা আগে বলি নাই কিন্তু এতক্ষণে তোমাদের এটা বুঝে যাবার কথা)। এখন মনে করো $i-1$ পজিশনে $i-1$ এ শেষ হওয়া sub-array দের মাঝে সর্বোচ্চ যোগফল sum এ আছে। তাহলে আমরা যখন i এ যাব তখন কি আমাদের পদ্ধতিতে sum এ i পর্যন্ত শেষ হওয়া sub-array দের মাঝে maximum টা কি পাব? অবশ্যই, কারণ i এ শেষ হতে গেলে আমাদের i তম element কে নিতে হবে এবং $i-1$ এ শেষ হওয়া maximum sub-array কে নিতে হবে। তবে এই সর্বোচ্চটা যদি আবার negative হয় তাহলে কিন্তু এটা না নেওয়াই ভাল অর্থাৎ i পর্যন্ত শেষ হওয়া ফাঁকা অ্যারে আমাদের সর্বোচ্চ মান দিবে এরকম চিন্তা করতে পার। একটা কথা বলে রাখা যেতে পারে যে, আমাদের এই পদ্ধতিতে উত্তর কিন্তু কমপক্ষে 0 হবে। এর যুক্তি হল, আমরা যদি empty sub-array নেই তাহলেই 0 পাওয়া সম্ভব। তবে যদি এটা বলা থাকে যে sub-array এর সাইজ কমপক্ষে 1 হতে হবে তাহলে তোমাদের এই পদ্ধতিকে সামান্য পরিবর্তন করতে হবে। কি পরিবর্তন করতে হবে এটা তোমরা নিজেরা বের করে নিও।

৯.২.২ Two dimensional Maximum sum problem

তোমাকে 2D একটি অ্যারে দেয়া আছে, তোমাদের এমন একটি আয়তক্ষেত্র বের করতে হবে যার মাঝে থাকা সকল সংখ্যার যোগফল সর্বোচ্চ হয়। আশা করি তোমরা এই প্রবলেম এর naive সমাধান বের করে ফেলেছ যার order $O(n^6)$. এই সমাধানে আমরা আয়তক্ষেত্রের দুই কোনা চারটি লুপ চালিয়ে বের করব এবং আরও দুইটি লুপ দিয়ে এই আয়তক্ষেত্রের ভিতরের সংখ্যাগুলিকে যোগ করব। আমরা চাইলে প্রতি column বা row তে consecutive sum টেকনিক ব্যবহার করে order কিছুটা কমিয়ে ফেলতে পারি। ধরা যাক আমরা প্রতিটি কলামে consecutive sum এর অ্যারে রেখেছি। এখন ধরি আমরা আয়তক্ষেত্রের দুই কোনা চারটি লুপ চালিয়ে fix করেছি এবং তারা হলঃ (a, b) এবং (c, d) যেখানে $a \leq c, b \leq d$ এবং a ও c ধরা যাক row নাম্বার এবং b ও d কলাম নাম্বার। তাহলে আমরা এই চারটি লুপ এর ভেতরে আরও একটি লুপ চালাব $[b, d]$ range এ এবং প্রতি কলামে আমরা সেই কলামের consecutive sum এর অ্যারে ব্যবহার করে a হতে c পর্যন্ত সংখ্যাগুলির যোগফল বের করে ফেলতে পারি। এই পদ্ধতিতে আমাদের order হবে $O(n^5)$. আমরা চাইলে 2-dimension এ consecutive sum টেকনিক ব্যবহার করে $O(n^4)$ এ সমাধান করতে পারি। দুই কোনা select করার পর তো আমাদের আগের সেকশনের $S[c][d]-S[a-1][d]-S[c][b-1]+S[a-1][b-1]$ ফর্মুলা ব্যবহার করে ভিতরের সব সংখ্যার যোগফল বের করে ফেলতে পারি।

আমার জানা এই প্রবলেম এর সবচেয়ে ভাল সমাধান হল $O(n^3)$. এই সমাধান খুব একটা কঠিন না। আমাদেরকে প্রত্যেক কলাম এর জন্য consecutive sum এর অ্যারে তৈরি করে রাখতে হবে। এবার দুইটা লুপ চালিয়ে আয়তক্ষেত্রের উপরের আর নিচের দুই row কে fix করে ফেলব। এখন প্রত্যেক কলাম এর জন্য fix করা দুই row এর মাঝের অংশ এর যোগফল বের করব। তাহলে আমরা আসলে একটি 1-dimension অ্যারে পাব। একটু চিন্তা করে দেখ তো আমরা এখন যদি 1-dimension এ maximum sum problem সমাধান করি তাহলেই 2-dimension এর প্রবলেমটা সমাধান হয়ে যায় কিনা!

৯.৩ Pattern খোঁজা

৯.৩.১ LightOJ 1008

আমাদের টেবিল ৯.১ এর মত একটি ছক থাকবে। বলতে হবে n সংখ্যাটি কোন row এবং কোন column এ আছে। n এর মান সর্বোচ্চ 10^{15} হতে পারে।

সারনী ৯.১: LightOJ 1008 সমস্যার টেবিল

10	11	12	13
9	8	7	14
2	3	6	15
1	4	5	16

n এর এতো বড় মান দেখে বুঝাই যাচ্ছে যে আমরা কোন ভাবেই এতো বড় টেবিল তৈরি করতে পারব না। এসব সমস্যার ক্ষেত্রে প্রধানত যা করতে হয় তাহলো এই টেবিল কে ভাল মত পর্যবেক্ষণ করতে হয়। খেয়াল করলে দেখবে যে নিচের বাম কোনা হতে x সাইজের বর্গে 1 হতে x^2 পর্যন্ত সকল সংখ্যা থাকে। এই সাবসেকশনে আমরা এখন থেকে বর্গ বলতে নিচের বাম কোনা থেকে শুরু হওয়া বর্গ বুঝব। সুতরাং আমরা যদি কোন একটি সংখ্যা, ধরা যাক 85 নেই তাহলে এটি 10 সাইজের বর্গের ভেতরে থাকবে কারণ $9^2 = 81$ আর $10^2 = 100$. তাহলে কোন একটি নাম্বার n দেয়া থাকলে সে কত সাইজের বর্গে থাকবে তা আমরা কেমনে বের করতে পারি? Square root করে। কিন্তু square root করলে তো fractional নাম্বার আসতে পারে। তাহলে? উপায় হল আমাদের হয় floor নিতে হবে অথবা ceiling নিতে হবে। কিন্তু কোনটা? এক্ষেত্রে আমাদের উদাহরণ নিয়ে কাজ করতে হবে। ধরা যাক, $n = 3$, তাহলে $\sqrt{3} = 1.732\ldots$. কিন্তু আমরা দেখতে পারছি এটা 2 সাইজের বর্গে আছে, সুতরাং আমাদের ceiling নিতে হবে। বা অন্যভাবে বলতে এমন একটি সর্বনিম্ন মান x খুঁজে বের করতে হবে যেন $x^2 \geq n$ হয়। তোমরা চাইলে binary search করে এই x বের করতে পার বা math.h এর sqrt ফাংশন ব্যবহার করতে পার। তবে sqrt ফাংশন ব্যবহার করতে চাইলে পূর্ণ বর্গ সংখ্যা এর sqrt বের করার সময় একটু খেয়াল রাখতে হবে। এসব ক্ষেত্রে সাবধানতার জন্য আমি যা করি তাহলো নিজে একটি sqrt ফাংশন লিখি যেখানে math.h এর sqrt ফাংশন কল করি এবং তার ceiling নেই বা int এ cast করি। ধরা যাক এই মান হল y. এখন আমি $y-1$ থেকে $y+1$ পর্যন্ত একটা লুপ চালাই। এবং এই লুপ চালিয়ে decision নেই যে কোন মানটা আসলে সঠিক। আরেকটি জিনিস, তাহলো এইযে আমরা বের করলাম যে আমাদের ceiling নিতে হবে বা floor নিতে হবে এসব ফর্মুলা বের করার সময় boundary case দিয়ে ফর্মুলা ভাল করে চেক করে দেখতে হবে। যেমন আমাদের এই প্রবলেম এ boundary case হবে $1, 2, 4, 5, 9, 10, 16\ldots$. অনেক সময় দেখা যায় তুমি যেই ফর্মুলা বের করেছ তা boundary case এ কাজ হয় না। যাই হক, আমরা তাহলে পেয়ে গেলাম কোন square এ আমাদের সংখ্যা আছে। যদি $x = \lceil\sqrt{n}\rceil$ হয় তাহলে n হয় x-তম row তে নাহয় x-তম কলামে আছে (ধরে নিলাম আমাদের টেবিলটা নিচ হতে উপরে এবং বাম হতে ডান দিকে 1 indexed)। কিন্তু কোনটা সত্য? যদি টেবিল এর দিকে তাকাও তাহলে দেখবে যে x যদি জোড় হয় তাহলে আমাদের square এর শেষ band টা বাম হতে নিচে আসে, আর যদি বিজোড় হয় তাহলে নিচ থেকে বামে যায়। আমাদের এই band এর সাইজ x. সুতরাং আমরা যদি জানি যে n এই শেষ band এ কত তম সংখ্যা তাহলেই আমাদের সমাধান হয়ে যায়। খেয়াল করো আমরা যদি x তম band এ থাকি তাহলে এর আগের আগের band এর শেষ সংখ্যা হল $(x-1)^2$ সুতরাং আমাদের সংখ্যাটি শেষ band এর $n-(x-1)^2$ তম সংখ্যা। এখন দেখতে হবে x জোড় না বিজোড়। ধরা যাক বিজোড়। তাহলে দেখতে হবে $y = n-(x-1)^2$ কি x এর থেকে বড় নাকি না। যদি বড় না হয় তাহলে এর $row = y$ এবং $column = x$ আর যদি বড় হয় তাহলে $row = x$ এবং $column = 1 + x^2 - n$. আশা করি কেন কলাম এর ফর্মুলা এরকম হল তা বুঝাতে হবে না। একই ভাবে x জোড় হলে row বা column এর ফর্মুলা কি হবে তা বের করে নিতে পারবে।

৯.৩.২ Josephus Problem

এই প্রবলেম এর একটি গল্প আছে। গল্পটা এরকম এক দিন জসেফাস তার 40 জন সৈন্য সহ একটি গুহায় আটকা পড়ে। গুহার মুখে রোমান সৈন্যরা ছিল। সুতরাং তারা সিদ্ধান্ত নেয় যে তারা একে একে আত্মহত্যা করবে। এজন্য তারা বৃত্তাকার ভাবে দাঁড়ায়। বৃত্তের এক স্থান থেকে শুরু হয়। প্রথম জন আত্মহত্যা করে এর পর পরের জনকে বাদ দিয়ে এর পরের জন আত্মহত্যা করে, এর পর এক জনকে বাদ দিয়ে তার পরের জন এভাবে চলতে থাকে। এভাবে চলতে চলতে একসময় মাত্র দুইজন বাকি থাকে, তাদের একজন ছিল জসেফাস। তারা দুইজন আর আত্মহত্যা না করে রোমানদের হাতে আত্মসমর্পণ করে।

যাই হোক, আমরা এই দুঃখের গল্প মাথা থেকে সরিয়ে ফেলি এবং চিন্তা করি n জন মানুষ থাকলে একজন কোথায় দাঁড়ালে সে last man standing হবে। যেহেতু এই সেকশনটি pattern খোঁজার সুতরাং আমরা pattern এর মাধ্যমে এই সমস্যা সমাধানের চেষ্টা করব। আমরা বিভিন্ন n এর মানের জন্য last man standing এর index বের করি (ধরে নেই 1-indexed). টেবিল ৯.২ এ $n = 1$ হতে 15 এর জন্য last man standing এর index দেয়া হল।

সারনী ৯.২: Josephus problem এ n এর বিভিন্ন মানে last man standing এর index

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
index	1	1	3	1	3	5	7	1	3	5	7	9	11	13	15

আশা করি তোমরা pattern টা বুঝতে পারছ? হয়তো কেউ কেউ বুঝতে পারছ যে pattern কিছু একটা আছে কিন্তু সেটাকে হয়তো নির্দিষ্ট করে বুঝতে পারছ না। যাই হোক, তাহলে খেয়াল করো, প্রত্যেক 2 এর power আমাদের উত্তর হয় 1 আর এর পর পরবর্তী 2 এর power পর্যন্ত উত্তর 2 করে বাড়তে থাকে। অর্থাৎ n দিলে আমাদের প্রথম কাজ হল এমন একটা সবচেয়ে বড় x বের করা যেন $2^x \leq n$ হয়। তাহলে $2(n - 2^x) + 1$ হবে উত্তর।

তোমরা চাইলে একে recursion এর মাধ্যমেও সমাধান করতে পারো। মনে করো তুমি জানতে চাইছ যে n জনের জন্য উত্তর কত। তাহলে আগে $n - 1$ এর জন্য উত্তর বের করে নাও। এখন তুমি n জনের প্রথম জন কে কাটো এবং 3 নাম্বার জনের কাছে যাও। তুমি কিন্তু জানো একে যদি 1 ধর তাহলে কোন জন বেঁচে যায়, তাহলে এ যদি 3 নাম্বার হয় তাহলে কত তম জন বেঁচে যাবে সেটা আশা করি বের করতে পারবে? এই পদ্ধতি n এর ছোট মান এ কাজ করবে। এছাড়াও যদি "এক জনকে বাদ দিয়ে পরের" জন না বলে "k জন কে বাদ দিয়ে পরের জন" বলত তাহলেও কাজে দিবে। তোমরা চাইলে knuth এর concrete mathematics বই এ এই নিয়ে আরও বিস্তর পড়াশুনা করতে পারো।

৯.৪ একটি নির্দিষ্ট রেঞ্জ এ Maximum element

৯.৪.১ 1 dimension

মনে করো তোমাকে একটি 1-dimension এর অ্যারে দেয়া আছে। এখন তোমাকে যেকোনো h সাইজের একটি সাব অ্যারে দিয়ে জিজ্ঞাসা করা হবে এই রেঞ্জ এর মাঝে সর্বোচ্চ সংখ্যা কত? সব সময়ই তোমাকে h সাইজের সাব অ্যারেই জিজ্ঞাসা করা হবে তবে সেই সাব অ্যারে বিভিন্ন জায়গায় শুরু হতে পারে। তুমি কেমনে খুব দ্রুত এই সমস্যা সমাধান করতে পারবে? যদি অ্যারে এর সাইজ n হয় তাহলে তো আমরা $O(nh)$ এ সমাধান করতে পারি। আমরা $O(nh)$ এ সকল স্থানের জন্য উত্তর বের করে একটি অ্যারে তে রেখে দিব এবং এর পর তুমি যেই সাব অ্যারে এর কথাই জিজ্ঞাসা করো না কেন তুমি সেই উত্তর এর অ্যারে দেখে বলে ফেলতে পারবে। কিন্তু $O(nh)$ খুব বেশি হয়ে যায়। তোমরা চাইলে square root segmentation বা segment tree ব্যবহার করে একে $O(n\sqrt{n})$ বা $O(n \log n)$ এও সমাধান করতে পারো। তবে মজার ব্যাপার হল আমরা একে $O(n)$ এ সমাধান করতে পারি। তোমরা যারা RMQ এর Tarjan এর linear time solution জানো তাদের বলে রাখি যে, সেই সমাধান এর

constant factor অনেক বেশি। সুতরাং আমরা সেই সমাধান বলব না। বরং আমাদের যে বলা আছে যে আমাদের query সাব অ্যারে সবসময় h সাইজের হবে সেই information কাজে লাগাব।

প্রথমে অ্যারে এর প্রতিটি ইনডেক্স কে $mod\ h$ ভাবে কল্পনা করো। এখন আমাদের দুইটি অ্যারে এর দরকার হবে। ধরা যাক একটি হল A এবং অপরটি B. $A[i]$ এ থাকবে i থেকে শুরু করে i বা i এর পরের $h-1$ mod ওয়ালা index পর্যন্ত সাব অ্যারে এর maximum. আর $B[i]$ এ থাকবে i বা এর আগের 0 mod ওয়ালা index পর্যন্ত সাব অ্যারে এর maximum. যেমন $h = 4$ এর ক্ষেত্রে চিত্র ৯.৩ এ আমরা দুইটি row তে A এবং B এর রেঞ্জ দেখালাম। A আর B এর অ্যারে বের করতে কিন্তু আমাদের $O(n)$ সময় লাগবে। B বের করার জন্য তুমি মূল অ্যারে এর সামনে থেকে পিছনে লুপ চালাবে। যদি দেখো তুমি 0 mod ওয়ালা index এ আছ তাহলে তো মূল অ্যারে এর সংখ্যাই তোমার maximum, আর যদি তা নাহয় তাহলে মূল অ্যারে এর এই index এর সংখ্যা আর B অ্যারে এর আগের index এর সংখ্যার maximum ই B এর এই index এর সংখ্যা হবে। একই ভাবে তুমি মূল অ্যারে এর পিছন থেকে আসলে A কে linear time এ বের করতে পারবে। এখন কথা হল তোমাকে (a, b) রেঞ্জ এর জন্য query করা হল ($b - a + 1 = h$) তাহলে কেমনে এই রেঞ্জ এর maximum বের করবে? খুব সহজ, max(A[a], B[b]) হল উত্তর। কেন? কারণ a হতে তুমি পরের $h - 1$ mod ওয়ালা index এর maximum পাচ্ছ A হতে আর b হতে এর আগের 0 mod ওয়ালা index এর maximum পাচ্ছ B থেকে। এটি দিয়েই তোমার পুরো রেঞ্জ কাভার হয়ে যাচ্ছে। যেহেতু তোমার query সাব অ্যারে এর সাইজ সবসময় h সুতরাং এই সাব অ্যারে এর মাঝে সবসময় একটা (এবং কেবল মাত্র একটা) 0 mod ওয়ালা index পাওয়া যাবেই। তাই এই পদ্ধতি সবসময় কাজ করবেই। সুতরাং আমরা এভাবে $O(n)$ preprocessing এ $O(1)$ সময়ে query এর উত্তর দিতে পারি।

সারনী ৯.৩: একটি নির্দিষ্ট রেঞ্জ এ maximum element বের করার জন্য $h = 4$ এ A ও B এর অ্যারে

index	0	1	2	3	4	5	6	7
index mod 4	0	1	2	3	0	1	2	3
A	[0, 3]	[1, 3]	[2, 3]	[3, 3]	[4, 7]	[5, 7]	[6, 7]	[7, 7]
B	[0, 0]	[0, 1]	[0, 2]	[0, 3]	[4, 4]	[4, 5]	[4, 6]	[4, 7]

৯.৪.২ 2 dimension

এবার 2D তে একই সমস্যা কেমনে সমাধান করা যায় দেখা যাক। মনে করো $n \times m$ সাইজের একটি অ্যারে আছে, তোমাকে প্রতিবার কোন একটি $p \times q$ সাইজের sub-array এর জন্য minimum বা maximum জিজ্ঞাসা করা হবে। কেমনে সমাধান করবা? খুব একটা কঠিন না। তুমি আগে প্রত্যেক row এর জন্য আগের উপায়ে q সাইজের sub-array এর minimum বা maximum বের করে ফেল। তাহলে তুমি একটি $n \times (m - q + 1)$ সাইজের একটি sub-array পাবা। এবার প্রত্যেক কলাম এর জন্য p সাইজের sub-array এর minimum বা maximum বের করো। তাহলে পাবা $(n-p+1) \times (m-q+1)$ সাইজের sub-array. এটিই তোমাকে শেষ উত্তর দিচ্ছে। অর্থাৎ এই পরিবর্তিত অ্যারে এর কোন একটি পজিশন (a, b) তোমাকে $(a, b) - (a + p - 1, b + q - 1)$ সাব অ্যারে এর maximum বা minimum দিবে।

৯.৫ Least Common Ancestor

মনে করো একটি tree দেয়া আছে। এখন দুইটি নোড দিয়ে জিজ্ঞাসা করা হবে তাদের least common ancestor কে? Leacent common ancestor হল সবচেয়ে নিচের এমন একটি নোড যেটি query এর দুই নোড এরই ancestor. আমাদের বর্ণনা এর সুবিধার জন্য ধরে নেই এই query এর দুইটি নোড হল x এবং y. $O(n)$ এ সমাধান করা তো খুব সহজ কিন্তু আমরা চাই আরও দ্রুত এই

query এর উত্তর দিতে। আমরা এখানে কেমনে $O(n\log(n))$ এ preprocess করে $O(\log(n))$ এ এই query এর উত্তর দেয়া যায় তা দেখব। ধরে নেই $\lceil\log(n)\rceil = 18$ তাহলে আমরা $parent[18][n]$ সাইজের একটি অ্যারে নিবো। এখন প্রতিটি নোড i এর জন্য আমরা $parent[0][i]$ এ i এর parent রাখব। root এর ক্ষেত্রে আমরা চাইলে root কেই রাখতে পারি বা কোন একটি sentinel যেমন -1 ব্যবহার করতে পারি। তবে আমার মতে root কে রাখাই ভাল, অর্থাৎ 0 যদি root হয় তাহলে $parent[0][root = 0] = 0(root)$.

এখন $parent[j][i]$ তে থাকবে i এর 2^j তম parent. এই জিনিস populate করার উপায় হল তুমি DFS করো। DFS এর সময় যেই নোড এ আসবা সেই নোড এর $parent[0\dots17][at]$ পূরণ করতে হবে। এটা পূরণ করার উপায় হল $parent[j][at] = parent[j-1][parent[j-1][at]]$. মানে at এর 2^{j-1} তম parent এর 2^{j-1} তম parent. এভাবে তুমি সব নোড এর জন্য $parent$ এর অ্যারে $O(n\log(n))$ সময়ে পূরণ করে ফেলতে পারবে। এবার আসা যাক query এর সময় কি করতে হবে সেই বিষয়ে। প্রথম কাজ হল x এবং y এর মাঝে যেটি বেশি depth এ আছে তাকে x এ নাও। এখন x কে y এর depth এর ancestor এ আনো। আনার উপায় সহজ, তুমি আগে $parent[17][x]$ দেখো, যদি এটি y এর depth এর থেকে কম depth এ হয় তাহলে এর পরের parent অর্থাৎ $parent[16][x]$ চেষ্টা করো, নাহলে $x = parent[17][x]$ করো। এভাবে 17 থেকে 0 পর্যন্ত লুপ চালালে x এবং y একই depth এ চলে আসবে এবং এ জন্য তোমার সময় লাগবে $O(\log(n))$. এবার আবার 17 হতে 0 পর্যন্ত ধরা যাক i এর লুপ চালাও এবং দেখো $parent[i][x]$ আর $parent[i][y]$ একই কিনা। যদি একই হয় তাহলে i এর লুপ continue করো আর যদি নাহয় তাহলে $x = parent[i][x]$ এবং $y = parent[i][y]$ করো। এভাবে 0 পর্যন্ত লুপ চালানোর পর, উত্তর হবে x বা y এর direct parent বা $parent[0][x]$. এই পদ্ধতির main theme হল $1 + 2^0 + 2^1 + \dots 2^{n-1} = 2^n$.

অধ্যায় ১০

Geometry এবং Computational Geometry

Geometry এর মানে হল জ্যামিতি। Computational Geometry হল জ্যামিতি বিষয়ক অ্যালগরিদম এর পাঠ। আমরা এই চ্যাপটার এ এই দুই বিষয় নিয়ে দেখব।

১০.১ Basic Geometry ও Trigonometry

খুব সাধারণ জ্যামিতির জ্ঞান আমাদের প্রায়ই প্রবলেম সমাধান করতে যেয়ে দরকার হয়। যেমন ত্রিভুজের ক্ষেত্রফলের ফর্মুলা, বর্গক্ষেত্রের ক্ষেত্রফল, আয়তক্ষেত্রের ক্ষেত্রফল, গোলক এর আয়তন ইত্যাদি নানা ফর্মুলা আমাদের প্রায়ই কাজে লাগে। যদি এসব ফর্মুলা তোমাদের মনে না থাকে তাহলে এখনি ছোট ক্লাসের বই দেখে নাও। এছাড়াও HSC তে পড়ে আসা ত্রিকোণমিতির ফর্মুলা আমাদের প্রায়ই কাজে লাগে। আমরা এখানে এদের মাঝে গুটিকয়েক ফর্মুলা দেখব।

মনে করো আমাদের কাছে একটি ত্রিভুজের তিন বাহু দেয়া আছে এবং তারা হলঃ $a, b, c.$ তাহলে সেই ত্রিভুজের ক্ষেত্রফল কত? এর ফর্মুলা হলঃ $\sqrt{s(s-a)(s-b)(s-c)}$ যেখানে $s = (a+b+c)/2$. একে হিরণের ফর্মুলা বলা হয়ে থাকে। অনেক সময় আমাদের বলে যে "যদি কোন উত্তর সম্ভব না হয় তাহলে impossible প্রিন্ট করো"। জ্যামিতির সমস্যার ক্ষেত্রে কোথাও এরকম কথা বললে তোমরা যা করবা তাহলো ফর্মুলা এর দিকে তাকাবা আর চিন্তা করবা এই ফর্মুলা কখন অসম্ভব হবে। যেমন উপরের ফর্মুলাটা অসম্ভব হবে যদি sqrt এর ভিতরের মান ঋণাত্মক হয়। যেমন $a = 10, b = 1, c = 1$ হলে উপরের ফর্মুলায় square root এর ভিতরের সংখ্যা ঋণাত্মক হবে। এর মানে হল এই তিনটি মান দিয়ে কোন ত্রিভুজ বানানো সম্ভব না। এভাবে অন্যান্য সমস্যার ক্ষেত্রেও এই ট্রিকটা তোমাদের কাজে লাগতে পারে। একই ভাবে যদি উপরের ফর্মুলা শূন্য মান দেয় এর মানে হবে ত্রিভুজের তিনটি বিন্দুই একই রেখায় আছে।

এখন ধরা যাক আমাদের দরকার হল যে a এর বিপরীত দিকের কোন A বের করতে হবে। কেমনে? তোমাদের অনেকের হয়তো ত্রিকোণমিতিতে পড়া ত্রিভুজের ক্ষেত্রফলের ফর্মুলা মনে আছেঃ $\Delta = \frac{1}{2}bc\sin A$ যেখানে Δ হল ত্রিভুজের ক্ষেত্রফল। আমরা যদি উপরের হিরণের ফর্মুলা ব্যবহার করে Δ এর মান বের করি তাহলে $\sin^{-1}$ করে A এর মান খুব সহজেই বের করে ফেলতে পারব। আসলেই কি পারব? খেয়াল করো, $\sin X = \sin(180^\circ - X)$ সুতরাং আমরা যদি $\sin^{-1}$ করি আমরা বুঝব না যে এটা কি X নাকি $180^\circ - X$. হ্যা আমরা জানি $\sin X = \sin(360^\circ + X)$ ও কিন্তু যেহেতু আমাদের ত্রিভুজের কোন কোণ 180° অপেক্ষা বড় নয় সেহেতু সেটা নিয়ে মাথা ব্যাথাও নাই। কিন্তু X নাকি $180^\circ - X$ তাতো বুঝা যাবে না। সুতরাং sin inverse এর সূত্র এক্ষেত্রে ব্যবহার করা যাবে না (হয়তো যাবে কিন্তু এর সাথে আরও কিছু চেক করতে হবে সেক্ষেত্রে)। আমরা যদি cos inverse ব্যবহার করতে পারি তাহলে কিন্তু এই সমস্যা হবে না, কারণ 0° হতে 180° প্রত্যেক ডিগ্রী এর জন্য cos

এর মান আলাদা। সুরতাং আমাদের এমন একটি ত্রিকোণমিতির সূত্র ব্যবহার করতে হবে যেন তাতে cos থাকে। এবং সেই সূত্র হলঃ $a^2 = b^2 + c^2 - 2bc\cos A$.

তবে এই যে square root বা sin inverse বা cosine inverse এসব ফাংশন কল করার আগে একটু সাবধান হতে হবে। আগেই বলেছি double বা float কিন্তু তোমার পুরো মান রাখতে পারে না, অনেক সময় দেখা যায় 2 এর জায়গায় $1.99999\ldots$ আছে আবার দেখা যায় 2.0001 আছে। এসবের জন্য square root বা sin inverse বা cosine inverse ফাংশন কল করার আগে একটু সাবধান হতে হয়। মনে করো square root এর ভিতরের মান আসার কথা 0 কিন্তু আসল -0.00001 বা sine inverse এর ভেতরে মান আসার কথা 1 আসল 1.00001 এসবক্ষেত্রে তুমি যদি ওই ফাংশন ডেকে বস তাহলে কিন্তু runtime error হয়ে যাবে। সেজন্য আমি যেটা করি তাহলো নিজের square root বা sin/cosine inverse ফাংশন লিখি। যেখানে দেখা যায় $sqrt(abs())$ কে কল করি বা sin/cosine inverse এর ভেতরে চেক দেই যে যেই মান পেলাম তা 1 এর থেকে বড় হলে কত return করব আর -1 এর থেকে ছোট হলে কত return করব।

আর দুইটি floating point নাম্বার সমান কিনা এটা চেক করার জন্য $a == b$ করলে কিন্তু হয় না। যা করতে হবে তা হলঃ $abs(a-b) <= eps$ কিনা যেখানে $eps = 10^{-7}$ এর মত কোন ছোট সংখ্যা। সমস্যা ভেদে দেখা যায় eps এর মান পরিবর্তন করতে হয়। একই ভাবে তুমি যদি চেক করতে চাও যে $a \le b$ কিনা তাহলে চেক করবাঃ $a <= b + eps$.

আমরা মাত্র দুইটি জিনিস দেখলামঃ ১- কিভাবে ত্রিভুজের তিন বাহু দেয়া থাকলে তার ক্ষেত্রফল বের করতে হয়, ২- কিভাবে ত্রিভুজের কোন কোণ বের করতে হয়। এরকম অনেক অনেক ছোট ছোট প্রবলেম আছে যা তোমরা আসলে তোমাদের কমন সেন্স প্রয়োগ করলেই সমাধান করতে পারবা অথবা খুব জোড় তোমাদের SSC বা HSC এর বই খুলে দেখতে হবে। আর internet তো আছেই। এরকম কিছু প্রবলেম আমি এখানে লিস্ট আকারে দিচ্ছিঃ

- ত্রিভুজের তিন বাহু দেয়া আছে, ত্রিভুজের ...
  - পরিবৃত্তের ব্যাসার্ধ বের করো (radius of circumcircle)
  - অন্তঃবৃত্তের ব্যাসার্ধ বের করো (radius of innercircle)
  - তিনটি মধ্যমা এর দৈর্ঘ্য বের করো (median)
  - তিনটি উচ্চতা বের করো (height)
  - তিনটি কোণ explicitly বের না করে তুমি কি বলতে পারবে কোন ত্রিভুজ সূক্ষ্মকোণী (acute), সমকোণী (right) বা স্থুলকোণী (obtuse) কিনা? এটা প্রায়ই দরকার হয়। কারণ আমরা যতটা সম্ভব floating point calculation কম করার চেষ্টা করি। যদি ত্রিভুজের তিন বাহু integer এ দেয়া থাকে তাহলে আমরা floating point calculation না করে বলে দিতে পারি ত্রিভুজটা সূক্ষ্মকোণী/সমকোণী/স্থুলকোণী কিনা। বা আসলে কোন একটি কোণ সূক্ষ্মকোণ/সমকোণ/স্থুলকোণ কিনা। উপায় হল, আমরা জানি পিথাগোরাসের ফর্মুলা হল $a^2 = b^2 + c^2$ যেখানে a হল সমকোণের বিপরীত বাহু বা অতিভুজ। এখন তুমি যদি $=$ এর পরিবর্তে $>$ বা $<$ বসাও তাহলেই তুমি বলে ফেলতে পারবে যে তারা সূক্ষ্মকোণ/স্থুলকোণ কিনা।
- ধর একটি r ব্যাসার্ধের বৃত্ত আছে। এখন এর কেন্দ্র হতে d দূরত্ব দূরে একটি লাইন টেনে বৃত্তের একটি অংশ কেটে ফেলে দেয়া হল। বলতে হবে বাকি অংশের ক্ষেত্রফল কত? বাকি অংশের পরিধিই বা কত? এটি যদি বৃত্ত না হয়ে একটি গোলক (sphere) হতো তাহলে ফর্মুলাগুলি কেমন হতো?

১০.২ Coordinate Geometry এবং Vector

আমরা আগের সেকশনে জ্যামিতির খুবই basic কিছু হিসাব নিকাশ দেখলাম। এখন আমরা কিছু coordinate geometry আর vector দেখব। আমরা হয়তো এক ফাঁকে complex number

ব্যবহার করে কেমনে vector এবং coordinate geometry এর কিছু কিছু হিসাব নিকাশ আরও সহজে করা যায় তাও দেখব।

2D coordinate geometry তে আমরা একটি বিন্দুকে (x, y) দ্বারা প্রকাশ করি। অর্থাৎ আমরা একটি structure ব্যবহার করে খুব সহজেই point বানিয়ে ফেলতে পারি। একই ভাবে সেই একই structure ব্যবহার করে আমরা 2d vector এর কাজও করে ফেলতে পারি। সুতরাং দেখা যায় যে, point structure এর জন্য addition, subtraction, scalar/dot product, cross product ইত্যাদি ফাংশন বা operator overload করার প্রয়োজন হয়। Line আবার বিভিন্ন রকম হতে পারে। আমরা অনেক ছোট বেলাতেই পড়েছিঃ (সরল)রেখা, রেখাংশ আর রশ্মি। রেখা হল যার দুই দিকেই কোন সীমা নেই, রেখাংশ হল যার দুই দিকেই সীমা আছে আর রশ্মি হল যার এক দিকে সীমা। একটি রেখাকে আমরা বিভিন্ন ভাবে represent করতে পারি। এদের একটি হল $y = mx + c$। এই representation এর সমস্যা হল y-axis এর parallel কোন রেখাকে আমরা represent করতে পারি না। এছাড়াও এভাবে representation এর জন্য বেশির ভাগ সময়ই আমাদের floating point number এর দরকার হয়। কারণ দুইটি point (x_1, y_1) এবং (x_2, y_2) দেয়া থাকলে $m = \frac{y1-y2}{x1-x2}$ এবং c বের করার জন্য আরও একটু জটিল হিসাব নিকাশ করতে হয় (তুমি m এর মান আর যদি $x1, y1$ বসাও তাহলেই c এর মান পেয়ে যাবে)। এর থেকে ভাল উপায় আমার মনে হয়ঃ $ax + by = c$. কারণ এভাবে কোন ধরনের line কে represent করা সমস্যা না এবং আগের মত floating point number ব্যবহার না করলেও চলে। এসব equation এর সুবিধা হল তুমি তৃতীয় কোন বিন্দু নিয়ে খুব সহজেই চেক করে দেখতে পারবে যে সেটি তোমাদের line এর উপর আছে কিনা বা লাইনের কোন দিকে আছে। যদি তুমি রেখাংশ বা রশ্মি represent করতে চাও তাহলে তুমি একটি বা দুইটি বিন্দু আর একটি সরল রেখার representation দিয়েই করতে পার। রেখাকে represent করার আরেকটি উপায় হল parametric form এবং এটি বেশ কাজের। ধর তোমাকে দুইটি বিন্দু $A(x_a, y_a)$ এবং $B(x_b, y_b)$ দেয়া আছে। তুমি এদের ভিতর দিয়ে যায় এরকম একটি সরল রেখার parametric representation চাও। এটি হবেঃ $A + t(B - A)$. এখানে $B - A$ কে একটি vector এর মত ভাবতে পার। এই representation এর অনেক সুবিধা আছে। এখানে তোমার floating point number এর দরকার হয় না- যদি A এবং B দুইটি integer point হয়। আবার খেয়াল করো t এর মান যদি $[0, 1]$ এ সীমাবদ্ধ হয় তাহলে এটি একটি রেখাংশ represent করে। যদি $[0, \infty]$ এ সীমাবদ্ধ হয় তাহলে এটি হবে রশ্মি, আর যদি পুরো রেখাকে represent করতে চাও তাহলে হবে $[-\infty, \infty]$. জিনিসটা এমন যে তুমি A থেকে শুরু করে B এর দিকে তিলে তিলে যাবে। $t = 0$ এর মানে তুমি A তেই থাকবে, $t = 1$ এর মানে তুমি B তে, এর মাঝে মান মানে তুমি A আর B এর মাঝে আছ। এমনকি তুমি $t = 1/2$ বা $t = 1/3$ বসিয়ে ঠিক মাঝের বা ঠিক এক তৃতীয়াংশ দূরের বিন্দুও বের করে ফেলতে পারবে। আবার যদি $t < 0$ হয় এর মানে হবে তুমি উলটো দিকে যাচ্ছ। কোন একটি বিন্দু $C(x_c, y_c)$ দিয়ে যদি বলা হয় এটি এই রেখার উপরে আছে কিনা তাহলেও এটি বের করা বেশ সহজ। তোমাকে $A + t(B - A) = C$ এই সমীকরণ সমাধান করতে হবে। এখানে x এর জন্য একটি এবং y এর জন্য আরেকটি সমীকরণ পাবা। তুমি কিন্তু কোন floating point calculation না করেই বলে ফেলতে পারবে এই সমীকরণ এর সমাধান আছে কিনা। তবে কোন দুইটি বিন্দু দিয়ে যদি বলে এটি AB রেখার একই দিকে আছে কিনা তাহলে মনে হয় এভাবে সম্ভব না। সেক্ষেত্রে আমি যা করি তাহলো coordinate geometry এর ত্রিভুজের ক্ষেত্রফল বের করার ফর্মুলা ব্যবহার করি:

$$\begin{vmatrix} x_A & y_A & 1 \\ x_B & y_B & 1 \\ x_C & y_C & 1 \end{vmatrix}$$

এই determinant এর ফর্মুলা ব্যবহার করে আমরা যেমন ABC ত্রিভুজের ক্ষেত্রফল বের করতে পারি ($\frac{1}{2}$ দিয়ে গুন আর পরম মান অর্থাৎ absolute value নিতে ভুলে যেয়ো না) ঠিক তেমনি A, B, C একই সরলরেখায় কিনা (determinant এর মান শূন্য হলে) বা ABC কি clockwise(cw) আছে নাকি anti-clockwise বা counter-clockwise(ccw) আছে তাও বের করা যায়। A, B, C যদি ccw তে থাকে তাহলে determinant এর মান positive আর cw তে থাকলে negative হয়। তোমার যদি মনে না থাকে cw হতে গেলে positive না negative হতে হয় তাহলে তুমি ফট করে

$(0,0)$, $(0,1)$ আর $(1,0)$ বসিয়ে determinant এর মান বের করে দেখো। যেহেতু বেশির ভাগ মানই শূন্য সেহেতু তোমাকে খুব কম হিসাব করতে হবে।

এখন চিন্তা করে দেখো তোমাকে যদি একটা বৃত্ত (কেন্দ্র ও ব্যাসার্ধ) আর সরলরেখা(দুইটি বিন্দু) দিয়ে যদি বলে তাদের ছেদ বিন্দু বের করো কেমনে করবা? সহজ উপায় হল প্রথমে তুমি পুরো co-ordinate কে বৃত্তের কেন্দ্রের perspective এ চিন্তা করো অর্থাৎ coordinate translate করে নাও। তাহলে বৃত্তের সমীকরণ আসবে $x^2 + y^2 = r^2$ (কারণ বৃত্তের কেন্দ্র এখন $(0,0)$). এবার সরলরেখার parametric equation বের করো আর তা বৃত্তের সমীকরণে বসিয়ে t এর মান সমাধান করো। তুমি t দিয়ে একটি quadratic equation পাবে। যদি দেখো এই সমীকরণের সমাধান নেই অর্থাৎ $b^2 - 4ac < 0$ এর মানে সরলরেখা বৃত্তকে ছেদ করে না। যদি $b^2 - 4ac = 0$ হয় অর্থাৎ t এর একটি মাত্র সমাধান থাকে তাহলে তারা পরস্পর কে স্পর্শ করে (tangent) আর যদি দুইটি সমাধান পাওয়া যায় এর মানে তারা দুইটি জায়গায় ছেদ করে। যদি এটা সরলরেখা না হয়ে রেখাংশ হতো তাহলেও কিন্তু তুমি t এর মান দেখে বলে দিতে পারতে যে ছেদ বিন্দুটি রেখাংশ এর উপর আছে না বাহিরে। যদি তোমার ছেদ বিন্দুই লাগে তাহলে coordinate আবারো translate করে নিতে ভুলে যেয়ো না। একটু চিন্তা করলে দেখবে এটি শুধু 2D তে না 3D তেও সমান ভাবে কাজ করে। তোমাকে যদি 3D তে দুইটা বিন্দু দিয়ে যদি বলে এই দুইটি বিন্দু দিয়ে যায় এরকম একটি সরল রেখার সাথে একটি গোলক (sphere) এর ছেদ বিন্দু বের করতে তাহলেও কিন্তু তুমি একই পদ্ধতি অনুসরণ করতে পারতে।

বৃত্ততে যখন চলেই আসলাম তখন বৃত্ত বৃত্ত এর ছেদ বিন্দু কেমনে বের করে এটাও দেখা যাক। বৃত্তের সাধারণ সমীকরণ হল $(x-a)^2 + (y-b)^2 = r^2$ ধরনের। অর্থাৎ বৃত্তের উপরের যেকোনো (x,y) বিন্দু যদি তুমি এই সমীকরণে বসাও তাহলে দুই পক্ষ সমান হবে। এখন তুমি যদি বৃত্তের দুইটি সমীকরণকে একটি থেকে আরেকটি বিয়োগ দাও তাহলে মনে হয় একটি সরলরেখার সমীকরণ পাবে। মনে হয় বলছি একারণে যে আমি এরকম করে করেছি কিনা মনে পড়ছে না। এসব কোড কয়েকবার করার পর বার বার করার আর মানে থাকে না, তখন এগুলোকে library আকারে রেখে দিতে হয় যাতে প্রয়োজন মত শুধু copy-paste করা লাগে। তবে একদম না বুঝে library তে রাখার কোন মানে নেই। যাই হোক, তাহলে আমরা যেই সরলরেখার সমীকরণ পেলাম সেটা কিসের? ধর $E1$ আর $E2$ হল দুইটি বৃত্তের সমীকরণ আর $E1 - E2$ যদি আরেকটি সমীকরণ হয় তাহলে যেসকল সমাধান $E1$ ও $E2$ দুইটিকেই সমাধান করে তারা $E1 - E2$ কেও সমাধান করবে। অর্থাৎ বৃত্তের ছেদবিন্দু দিয়ে যায় এরকম একটি সরল রেখার সমীকরণ হল আমাদের বিয়োগ করে পাওয়া সমীকরণ। যেহেতু বৃত্তের সমীকরণ জানো আর ছেদবিন্দু দিয়ে যাওয়া সরলরেখার সমীকরণও জানো তাহলে এখন তুমি কিছু সমীকরণ সমাধান করলেই ছেদবিন্দু গুলি পেয়ে যাবে। তবে আগের পদ্ধতিতে আর করতে পারবে না কারণে এখানে সরল রেখার সমীকরণ হল $ax + by = c$ ধরনের। এই পদ্ধতি অবলম্বন করার আগে তোমরা দেখে নিবে যে বৃত্ত দুইটি আসলেই ছেদ করে কিনা, নাহলে এই হিসাব নিকাশ করার সময় বিপদে পড়তে পার। বিপদ মানে, শেষ পর্যায়ে এসে তোমাকে যখন দ্বিঘাত সমীকরণ সমাধান করতে হবে তখন তোমাকে চিন্তা করতে হবে এই সমীকরণের আদৌ সমাধান আছে কিনা ইত্যাদি। এসব বাড়তি চিন্তা থেকে মুক্ত হবার জন্যই আগেই দেখে নিতে হবে বৃত্ত ছেদ করে কিনা। ছেদ বিন্দু বের করার চেয়ে বৃত্ত দুইটি ছেদ করে কিনা সেটা বের করা সহজ। যদি কেন্দ্রদের দূরত্ব d হয় তাহলে দুইটি বৃত্ত ছেদ করে যদি $abs(r_1 - r_2) \le d \le r_1 + r_2$ হয়। আরও একটি কথা, এখানে খেয়াল করো d এর মান বের করতে আমাদের square root ব্যবহার করতে হয়। কিন্তু আমরা চাইলে সবগুলি মানকে square করে square root ব্যবহার না করেও বৃত্ত দুইটি ছেদ করে কিনা তা বলে ফেলতে পারি। এটা খুব দরকারি কারণ আমাদের উচিত যতটুকু পারা যায় double এ calculation কে এড়িয়ে চলা।

এতক্ষণ মূলত coordinate geometry আর parametric equation নিয়ে কথা বললাম। parametric equation এর কথা বলতে গিয়ে একটু vector এর কথাও বলেছি। এবার vector কে আরও একটু ভালমতো দেখা যাক। মনে করো একটি রেখা আর একটি বিন্দু দিয়ে বলা হল এই বিন্দু থেকে ঐ সরলরেখার উপর লম্ব টানতে হবে। লম্ব দূরত্ব কিন্তু বের করা বেশ সহজ কিন্তু লম্বের পাদ বিন্দু বের করা একটু কঠিন। একটি উপায় হল রেখার উপর একটি বিন্দু A নাও এর পর রেখার parametric equation বের করো। ধরে নাও t তে পাদবিন্দু। তাহলে A, পাদবিন্দু আর দেয়া বিন্দু এদের নিয়ে

একটি পিথাগোরাস এর ফর্মুলা লিখে ফেল তাহলেই t এর মান সমাধান করে তুমি পাদবিন্দু বের করে ফেলতে পারবে। তবে আরেকটি উপায় হল ধরে নাও পাদবিন্দু B, তাহলে B হতে A এবং প্রদত্ত বিন্দুর vector দের dot product নিলে তা শূন্য হবার কথা। এই সমীকরণ কে সমাধান করলেই তুমি B এর coordinate পেয়ে যাবে। একই ভাবে তোমাকে যদি বলে একটি রেখার একটি বিন্দুতে লম্ব এর সমীকরণ বের করতে হবে আশা করি তুমি তা বের করতে পারবে।

এখন মনে করো তোমাকে দুইটি বিন্দু দিয়ে বলা হল এই দুইটি বিন্দু যদি কোন সমবাহু ত্রিভুজ এর vertex হয় তাহলে ওপর বিন্দু বের করো। তাহলে কেমনে করবে? তুমি চাইলে ধরে নিতে পার যে ওপর বিন্দু (x, y) এবং এর পর এটি হতে ওপর দুইটি বিন্দুর দূরত্ব "এতো" হবে এই বলে দুইটি সমীকরণ দাঁড় করিয়ে তাদের সমাধান করতে পার। তবে এটি আমার কাছে একটু পেঁচানো লাগে। যারা coordinate কে rotate করাতে পারো না তারা এভাবেই করতে পারো শেষ অবলম্বন হিসাবে। কিন্তু যারা rotate করাতে পারো তারা হয়তো আরও সহজে করে ফেলতে পারবা। মুল বিন্দুর সাপেক্ষে coordinate θ কোনে ঘুরানোর উপায় হল $(x, y) \mapsto (x\cos\theta - y\sin\theta, x\sin\theta + y\cos\theta)$. এই কথা বলার পর তোমাদের উচিত আমাকে বকা বকি করা। এই হড়বড়ে ফর্মুলা কেমনে মনে রাখা সম্ভব! অনেকে একে matrix form এ মনে রাখেঃ

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix} \tag{১০.১}$$

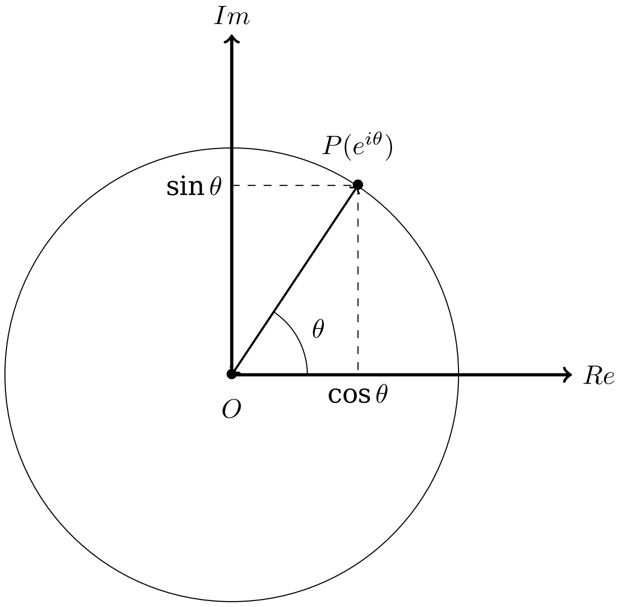
এটা বলার অপেক্ষা রাখে না যে, এটাও কোন অংশে কম কঠিন নয়! তাহলে কি কোন সহজ উপায় নেই? আছে, তবে এজন্য তোমাদের complex number সম্পর্কে basic জ্ঞান দরকার। আমি জানি না এখন HSC তে complex number আছে কিনা বা থাকলেও অয়লার এর সুন্দর ফর্মুলাটি দেখানো হয় কিনা। মনে হয় না আমাদের সময়ও অয়লার এর ফর্মুলা দেখানো হতো। অয়লার এর ফর্মুলাটি হল $e^{i\theta} = \cos\theta + i\sin\theta$. অনেকে মনে করে $\theta = \pi$ বসালে পৃথিবীর সবচেয়ে সুন্দর ফর্মুলা $e^{i\pi} + 1 = 0$ পাওয়া যায় যেখানে বিশ্বের সব থেকে গুরুত্বপূর্ণ constant গুলি আছেঃ $0, 1, \pi, e$. যাই হোক, চিত্র ১০.১ এ 1 unit ব্যাসার্ধের একটি বৃত্ত নেয়া হয়েছে এবং এই বৃত্তের উপর x-axis হতে θ কোণ দূরত্বে একটি বিন্দু নেয়া হয়েছে ধরা যাক এর নাম P. এই বিন্দুটি euler এর representation অনুসারে $e^{i\theta}$. যদি বৃত্তের ব্যাসার্ধ r হতো তাহলে এটি হতো $re^{i\theta}$. খেয়াল করলে দেখবে P বিন্দু এর coordinate হল $(\cos\theta, \sin\theta)$ অর্থাৎ জটিল সংখ্যায় $a + ib$ ফর্ম এ যদি আমরা লিখি তাহলে দাঁড়াবে $\cos\theta + i\sin\theta$.

এতো কিছু বলার কারণ হল, তোমরা যদি কোন একটি vector কে $e^{i\theta}$ দিয়ে গুন করো তাহলে সেই vector মূলবিন্দু সাপেক্ষে counter clockwise দিকে θ কোণে ঘুরে যাবে। ধরা যাক আমাদের vector টি হল OA যেখানে A এর coordinate হল (x, y). একে complex number এ লিখলে পাব $x + iy$. এখন একে আমরা যদি $e^{i\theta}$ দিয়ে গুন করি করি তাহলে আমরা পাবঃ

$$\begin{aligned} e^{i\theta} \times (x + iy) &= (\cos\theta + i\sin\theta)(x + iy) \\ &= (x\cos\theta - y\sin\theta) + i(x\sin\theta + y\cos\theta) \end{aligned}$$

অর্থাৎ গুন করার পর coordinate দাঁড়ায় $(x\cos\theta - y\sin\theta, x\sin\theta + y\cos\theta)$.

শেষ করব vector কত powerful হতে পারে তার একটি উদাহরণ দিয়ে। ইচ্ছা করেই আমি এই প্রবলেম এর 2D variant টা উদাহরণ হিসাবে না নিয়ে 3D variant টা নেবো, উদ্দেশ্য তোমাদের দেখানো যে dimension বাড়লেও vector computation এর জটিলতা ওতটা পরিবর্তন হয় না। আমাদের প্রবলেম হল, O কেন্দ্র বিশিষ্ট r ব্যাসার্ধের একটি গোলক আছে। গোলকের উপর কোন রশ্মি পড়লে তা প্রতিফলিত হয়। A হতে AB নির্গত হয় যা ধরা যাক 1 second এ 1 unit দূরত্ব যায়। বলতে হবে t সময় পরে A হতে AB এর দিকে নির্গত রশ্মি কোথায় গিয়ে পৌঁছায়। যদি AB রশ্মি গোলককে ছেদ না করে (parametric equation ব্যবহার করে ছেদ করে কিনা তাতো বের করতে পারবাই) তাহলে তো এটা বের করা ব্যাপারই না! কথা হল ছেদ করলে কি হবে। আমরা প্রথমে parametric equation এর সাহায্যে ছেদ বিন্দু বের করি। ধরা যাক ছেদ বিন্দু হল P. এখন আমাদের বের করতে



চিত্র ১০.১: জটিল সংখ্যার euler এর representation

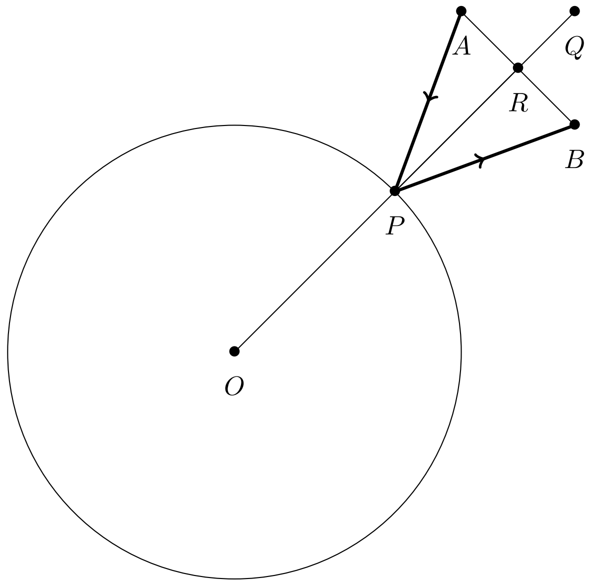
হবে AP রশ্মি প্রতিফলিত হয়ে কোন দিকে যাবে সেই দিক। এখন OP vector আঁকি, এটি আলোর প্রতিফলনের জন্য লম্ব হিসাবে কাজ করবে। আমরা OP কে Q পর্যন্ত বাড়িয়ে দেই। যদি আমাদের প্রতিফলিত রশ্মি PB হয় (ধরি $|PB| = |AP|$) তাহলে আমরা বলতে পারি $\angle APQ = \angle BPQ$. তোমাদের সুবিধার জন্য চিত্র ১০.২ এ সব কিছু আঁকা আছে।

এখন A হতে PQ এর উপর AR লম্ব টানি (BR ও তাহলে QR এর উপর লম্ব হবে)। আমরা লিখতে পারিঃ $\vec{AP} = \vec{AR} + \vec{RP}$. একই ভাবে $\vec{BP} = \vec{BR} + \vec{RP}$. তাই আমরা লিখতে পারি $\vec{AP} - \vec{AR} = \vec{BP} - \vec{BR}$. কিন্তু $\vec{AR} = -\vec{BR}$. সুতরাং $\vec{BP} = \vec{AP} - 2\vec{AR}$. আমরা $\vec{AP}$ জানি কারণ A ও P উভয়ই আমাদের জানা। কিন্তু AR জানি না। আমরা চাইলে A হতে PQ এর উপর লম্ব AR এর vector ফর্মুলা আগের বলে দেয়া নিয়মে বের করতে পারি কিন্তু তা না করে আমরা আরও একটু ভেক্টরিয়ো ভাবে সমাধান করব। আমরা যদি কোনভাবে $\vec{RP}$ বের করতে পারি তাহলেও কিন্তু হবে। এখন $\vec{RP}$ হোল $\vec{AP}$ এর $\vec{QP}$ বরাবর component, যেটা আমরা $\vec{QP}$ বরাবর unit vector এর সাথে dot product করলেই পেতে পারি। কিন্তু আমরা কিন্তু Q একটি arbitrary বিন্দু ধরেছিলাম তবে $\vec{QP}$ এর দিকে $\vec{OP}$ এর দিকের সম্পূর্ণ বিপরীত দিক আর আমরা O আর P দুটিরই coordinate জানি। ব্যাস শেষ!

১০.৩ কিছু Computational Geometry এর অ্যালগোরিদম

১০.৩.১ Convex Hull

তোমাকে 2d coordinate এ n টি বিন্দু দেয়া আছে, তোমাকে এর convex hull বের করতে হবে। Convex hull কি? প্রদত্ত বিন্দু গুলিকে bound করে সবচেয়ে ছোট যে convex polygon বানানো যায় তাই এই সকল বিন্দুর convex hull. Convex polygon হল এমন একটি polygon যার প্রতিটি কোণ $180°$ এর সমান বা ছোট। যদি convex hull এর কোন তিনটি বিন্দুকে একই রেখার উপর না দেখতে চাও তাহলে সবসময় $180°$ এর ছোট নিতে পারো। একে এভাবেও চিন্তা করতে পারো- প্রদত্ত n বিন্দুতে তুমি পেরেক পুঁতো, এর পর একটি রাবার ব্যান্ড কে প্রসারিত করে সকল পেরেককে



চিত্র ১০.২: গোলকে প্রতিফলন

cover করে ছেঁড়ে দাও, তাহলে দেখবে তোমার রাবার খুব tight ভাবে পেরেক গুলি দিয়ে যায় এবং একটি convex polygon এর রূপ নেয়। এটিই convex hull. প্রদত্ত point সমূহ কে cover করে যেসব convex polygon আঁকা যায় তাদের মাঝে সবচেয়ে ছোট ক্ষেত্রফল এবং পরিসীমা এই convex hull এর।

এখন প্রশ্ন হল আমরা কি ভাবে convex hull বের করতে পারি? Convex hull বের করার জন্য অনেকগুলি অ্যালগরিদম আছে। সবচেয়ে জনপ্রিয় হল Graham's scan. প্রথমে আমাদের দেয়া বিন্দু গুলির মাঝে সবচেয়ে কম y ওয়ালা বিন্দু বের করতে হবে, যদি এরকম অনেক গুলি থাকে তাহলে তাদের মাঝে সবচেয়ে কম x ওয়ালা বিন্দু নিব। ধরা যাক এটি হল O. এখন এই বিন্দুকে কেন্দ্র করে অন্যান্য বিন্দু গুলিকে আমরা ccw এ সর্ট করব। এক্ষেত্রে দুইটি জিনিস খেয়াল রাখতে পারো, একঃ তোমরা চাইলে পরবর্তী calculation গুলি সহজে করার জন্য O কে মুল বিন্দুতে translate করে নিতে পারো, দুইঃ atan2 ব্যবহার করে তুমি সর্ট করার আগেই সব বিন্দুর O এর সাপেক্ষে কোণ বের করে নিতে পারো, বার বার comparision ফাংশন এর ভিতরে না বের করে আগে থেকে বের করে রাখলে সময় কম লাগে। তবে আমরা চাইলে atan2 ব্যবহার না করেও সর্ট করতে পারি। ধরা যাক comparison ফাংশন এ A ও B দুইটি বিন্দু দিয়ে বলা হল কোনটি আগে হবে? যদি OAB ccw হয় তাহলে A আগে হবে নাহলে পরে। এখন এই সর্টেড বিন্দু গুলিকে একে একে নিতে হবে আর একটি stack এ রাখতে হবে। যদি stack ফাঁকা হয় তাহলে সরাসরি stack এ ঢুকিয়ে দাও আর যদি তা নাহয় তবে O, stack এর সবচেয়ে উপরে থাকা বিন্দু আর বর্তমান বিন্দু এদের চেক করে দেখতে হবে যে এরা কি cw নাকি ccw. যদি cw হয় তাহলে stack এর উপরের বিন্দুকে ধরে ফেলে দিতে হবে এবং এবারের stack এর উপরের বিন্দুকে নিয়ে আবার একই চেক করতে হবে। এভাবে একে একে সব বিন্দু চেক করা শেষ হয়ে গেলে stack এ convex hull পাওয়া যাবে।

যদিও Graham's scan বেশ সহজ এবং জনপ্রিয় কিন্তু এটি numerically ওতটা stable না। অর্থাৎ তোমার coordinate যদি floating point এ থাকে তাহলে মাঝে মাঝে ঝামেলা পাকতে পারে floating point calculation এর instability এর জন্য। সেজন্য যেই অ্যালগরিদম ব্যবহার করা উচিত তাহলো Monotone chain convex hull algorithm. এটিও বেশ সহজ। প্রথমে আমাদের বিন্দু গুলিকে coordinate অনুসারে সর্ট করতে হবে অর্থাৎ প্রথমে x অনুযায়ী এবং তাদের x সমান হলে y অনুযায়ী। এবার আগের মত সর্ট করা বিন্দুগুলিকে একে একে নিব। stack এর উপরের

2 টি বিন্দু এবং বর্তমান বিন্দু নিয়ে চেক করে যদি দেখি ccw তাহলে stack এর উপরের বিন্দুকে ফেলে দিব (এখন কিন্তু আমরা বাম দিক থেকে ডান দিকে যাচ্ছি এবং আমরা শুধু উপরের hull বানাচ্ছি)। এই প্রসেস শেষ হয়ে গেলে আমরা সর্টেড লিস্ট এর শেষ থেকে শুরুর দিকে যাব এবং আগের মত যদি stack এর উপরের দুইটি বিন্দুর সাথে বর্তমান বিন্দু ccw এ থাকে তাহলে stack এর উপরের বিন্দুটি ফেলে দিব। এভাবে একবার বাম থেকে ডানে এবং আরেকবার ডান থেকে বামে গেলে আমরা উপরের hull এবং নিচের hull তৈরি করে ফেলতে পারব। এই অ্যালগরিদম আগের থেকে stable কারণ এখানে কোণ অনুসারে সর্ট করার কোন ব্যাপার নেই। তোমরা চাইলে wikipedia তে দেয়া implementation টা দেখে নিতে পারো।

১০.৩.২ Closest pair of points

2d coordinate এ n টি বিন্দুর coordinate দেয়া আছে। তোমাকে এদের মাঝের closest pair distance বের করতে হবে অর্থাৎ যেই দুইটি বিন্দুর মাঝের দূরত্ব সবচেয়ে কম সেই দুইটি বিন্দু বের করতে হবে বা সেই দূরত্ব বের করতে হবে। এখানে দূরত্ব মাপতে আমরা euclidean distance ব্যবহার করব। দুইটি বিন্দুর coordinate যদি (x_1, y_1) এবং (x_2, y_2) হয় তাহলে তাদের মাঝের euclidean distance হবে $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. একে straight line distance ও বলা যায়। এই সমস্যা সমাধানের জন্য আমাদের প্রথমে যা করতে হবে তাহল বিন্দু গুলিকে x অনুযায়ী সর্ট করতে হবে (ছোট থেকে বড়)। এরপর আমরা এই বিন্দুগুলিকে দুই ভাগে ভাগ করব, এক ভাগে (বাম ভাগে) থাকবে ছোট x আরেক ভাগে বড় গুলি। মোটামোটি সমান দুই ভাগে ভাগ করতে হবে। অর্থাৎ বিজোড় এর ক্ষেত্রে একদিকে একটি বেশি থাকতে পারে আর কি! এখন তোমাদের recursively দুই দিকের জন্য closest pair বের করার ফাংশন call করতে হবে। closest pair ফাংশন দুইটি জিনিস দিবে- একঃ তার পাওয়া বিন্দুগুলির মাঝে closest pair distance এবং দুইঃ তার পাওয়া বিন্দুগুলির y অনুযায়ী sorted list (বড় থেকে ছোট). যদি তোমার ফাংশন মাত্র একটি বিন্দু পায় (base case) তাহলে ধরা যাক সে ∞ বলবে closest pair distance হিসাবে আর একটি বিন্দুর জন্য তো সর্ট করার কিছু নাই। এখন যদি একের বেশি বিন্দু হয় তাহলে তো আমরা দুই ভাগে ভাগ করে recursive কল করেছিলাম। ধরা যাক আমরা দুই দিক থেকে closest pair distance পেয়েছি d_1 এবং d_2. আমরা $d = min(d_1, d_2)$ নিয়েই শুধু আগ্রহী। আরও মনে করা যাক, দুই দিকের y অনুসারে sorted list হল P_1 এবং P_2. এখন আশা করি বুঝতে পারছ কেমনে এদের মিলিত y অনুসারে সর্ট করা লিস্ট পাওয়া যাবে? ঠিক merge sort এর মত। দুই লিস্ট এর মাথা দেখবা এবং যার y বেশি তাকে নিবা এভাবে চলতে থাকবে (আমরা কিন্তু বড় থেকে ছোট সর্ট করছি)। এখন মনে করো বামের ভাগ এর সবচেয়ে বড় x হল $x_{divider}$. এটা recursive কল করার আগে x অনুযায়ী sorted লিস্ট হতে বের করে একটি local variable এ রাখতে পারো। খেয়াল করো, recursive call করার পর কিন্তু সেই লিস্ট y অনুযায়ী sorted হয়ে যাবে। সুতরাং আমাদের আগেই এই $x_{divider}$ বের করে রাখতে হবে (অথবা আরও অনেক trick খাটানো যায় যা আমরা পরে সংক্ষেপে বলব)। recursive কল শেষে পাওয়া y অনুযায়ী সর্ট করা বিন্দু গুলিকে ব্যবহার করে এদের মাঝের closest pair distance বের করতে পারি। প্রথমে খেয়াল করো, কোন বিন্দুর x যদি $x_{divider} - d$ এর থেকে ছোট হয় বা $x_{divider} + d$ এর থেকে বড় হয় তাহলে সেই বিন্দুকে আমরা বিবেচনা না করলেও পারি (তবে sorted list পাবার জন্য আমাদের সব বিন্দুই বিবেচনা করতে হবে)। এখানে $[x_{divider} - d, x_{divider} + d]$ এর বাহিরে হলে বিবচনা করবোনা বলতে বুঝাচ্ছি যে closest pair distance বের করার জন্য বিবেচনা না করা। এখন মনে করো P_1 থেকে আমরা যাদের বিবেচনা করব তারা হল P_1' এবং একই ভাবে P_2 হতে P_2' এবং এরা y অনুযায়ী sorted. এখন আমাদের দুইটি pointer লাগবে যা দুই লিস্ট এর দুইটি বিন্দুকে point করবে। শুরুতে এরা list এর শুরুর element গুলিকে point করবে। এখন দেখো যদি P_2' লিস্ট এর পয়েন্টকৃত বিন্দুর y যদি P_1' এর পয়েন্টকৃত বিন্দুর y থেকে বেশি হয় তাহলে P_2' এর pointer কে ততক্ষণ এগিয়ে নিতে হবে যতক্ষণ না P_2' লিস্ট এর বিন্দুর y ছোট হয় বা লিস্টটা শেষ না হয়ে যায়। এখন তোমাকে কয়েকটা চেক করতে হবে, চেকগুলি হল P_1' লিস্ট এর পয়েন্টকৃত বিন্দুর সাথে P_2' লিস্ট এর পয়েন্টকৃত বিন্দু এবং এর কিছু আগে ও পরের বিন্দু। কত আগে বা পরে? সেটা

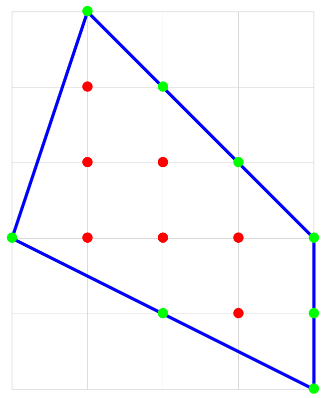
নির্ভর করবে বিন্দু গুলির y এর উপর। অর্থাৎ যদি P_1' এর বিন্দুটির y coordinate y' হয় তাহলে $[y'-d, y'+d]$ রেঞ্জ এর ভিতরে y থাকা সকল P_1' এর বিন্দুকে চেক করতে হবে। এটা দেখানো যায় যে এরকম বিন্দু আসলে 6 টার বেশি হবে না। সুতরাং এভাবে $O(n\log n)$ এ আমরা closest pair of points বের করতে পারি। কিছুক্ষণ আগে কিছু ট্রিক বলব বলেছিলাম। ট্রিকটা হল তুমি মূল অ্যারে সর্ট না করে একটা index এর অ্যারে কে সর্ট করতে পারো। আবার তুমি চাইলে একটি auxiliary array নিয়ে তাতে সর্ট করতে পারো (মানে আরেকটা লিস্ট আর কি)।

তোমরা চাইলে stl এর সাহায্যে আরও সহজে এই প্রবলেম সমাধান করতে পারো। এজন্য দুইটি set এর প্রয়োজন, একটি সেট এ বিন্দুগুলি x অনুযায়ী সাজানো থাকবে অপরটি y অনুযায়ী। আমরা তাদের নাম দেই যথাক্রমে X এবং Y. আর এখন পর্যন্ত প্রাপ্ত closest pair distance ধরা যাক d তে থাকবে যাতে শুরুতে ∞ মান থাকবে। এখন প্রথমে আমাদের দুইটি সেটই ফাঁকা এবং আমাদের কাছে x অনুযায়ী সর্ট করা বিন্দুদের একটি লিস্ট আছে। এখন আমরা এই লিস্ট থেকে একে একে বিন্দু গুলিকে নিব। ধরা যাক এই বিন্দুটি হল (x, y), তাহলে আমাদের প্রথমে X এ দেখতে হবে $x-d$ এর থেকেও ছোট x আলা কোন বিন্দু আছে কিনা থাকলে তাকে X এবং Y উভয় থেকেই মুছতে হবে। এরকম সব বিন্দু মুছা হয়ে গেলে আমাদের Y নিয়ে কাজ করতে হবে। আমরা Y এ lower bound ব্যবহার করে $y-d$ থেকে $y+d$ এর ভিতরে y এর মান আলা যত বিন্দু Y এ আছে তাদের সাথে বর্তমান বিন্দুর distance বের করে closest pair distance d কে আপডেট করব এবং সেই সাথে X ও Y এ বর্তমান বিন্দু ঢুকিয়ে দেব। সত্যি কথা বলতে আমাদের X set এর দরকার নেই, তোমরা x অনুযায়ী sorted অ্যারেতে দুইটা হাত রেখেই এই কাজ করতে পার।

১০.৩.৩ Line segment intersection

ধরা যাক তোমাদের অনেক গুলি line segment দেয়া আছে, বলতে হবে এদের মাঝে কতগুলি ছেদ বিন্দু আছে, অর্থাৎ কতগুলি pair of line segments পরস্পরকে ছেদ করে। যাতে floating point calculation ঝামেলা না পাকাতে পারে সেজন্য অনেক সময় বলা হয় দুই এর বেশি line segment পরস্পরকে একই বিন্দুতে ছেদ করে না। এই সমস্যা সমাধানের জন্য আমাদের একটি balanced binary search tree এবং একটি priority queue বা heap দরকার। heap এ অনেকগুলি event থাকবে এবং সবচেয়ে কম x এর event টি top এ থাকবে। প্রথমে সকল line segment এর শুরু এবং শেষ প্রান্ত heap এ প্রবেশ করাই। এবার আমরা heap থেকে সবচেয়ে কম x ওয়ালা event তুলব। যদি এটি কোন line segment এর শুরুর প্রান্ত হয় তাহলে আমরা এই line segment কে binary search tree (BST) তে প্রবেশ করাব আর যদি এটি শেষ প্রান্ত হয় তাহলে সেই segment টি আমরা BST থেকে সরিয়ে নিব। BST টা হবে line segment গুলির উপর হতে নিচে order এ। তবে তোমরা হয়তো এটা বুঝতে পারতেছো যে এই order আসলে কোন x এর জন্য y দেখা হবে তার উপর নির্ভর করে। মনে করো দুইটি segment পরস্পরকে ছেদ করে তাহলে, ছেদ করার আগে একটা উপরে থাকে আর ছেদ করার পরে আরেকটা। যাই হোক আগের কথাই ফিরে যাই। খেয়াল করো, যখন তুমি কোন একটি নতুন segment প্রবেশ করাবে তখন এর ঠিক উপরে এবং ঠিক নিচে একটি segment থাকবে (BST যেহেতু y এর অর্ডারে থাকে সেহেতু বলা যায় segment এর ঠিক আগের ও ঠিক পরের দুইটি segment)। তোমাকে এই দুইটি segment এর সাথে এই নতুন segment এর intersection গুলি বের করে তা event আকারে heap এ ঢুকাতে হবে এবং ঐ দুই পাশের দুই segment এর মাঝের intersection এর event টি remove করে ফেলতে হবে। অর্থাৎ আমাদের তিন ধরনের event আছে, একটি হল segment এর শুরু একটি শেষ আরেকটি হল intersection. শুরু আর শেষে কি করতে হবে তা জানি, কিন্তু intersection এ কি করব? intersection এ আমাদের ঐ দুইটি segment এর অর্ডার পালটিয়ে ফেলতে হবে, অর্থাৎ আগের উপরের segment এবার নিচে চলে যাবে আর নিচেরটা উপরে চলে যাবে। এই হল পুরো সমাধান কিন্তু এই জিনিসটা BST দিয়ে implement করা আসলেই অনেক কষ্টকর। তবে আমরা খুব সহজেই stl এর set ব্যবহার করে এই প্রবলেম সমাধান করে ফেলতে পারি।

প্রথমে যা করতে হবে হবে তাহলো segment এর একটি set. যেহেতু set সেহেতু আমাদের একটি comparison ফাংশন লিখতে হবে যা দুইটি segment এর মাঝে তুলনা করবে। তবে এই



চিত্র ১০.৩: Pick's theorem

তুলনা করার জন্য আমাদের একটি x দরকার। কোন x এর সাপেক্ষে আমরা এই compare করব? ধরা যাক এই x একটি global variable এ থাকবে এবং আমাদের comparison ফাংশনকে যখন দুইটি segment দিয়ে জিজ্ঞাসা করা হবে কোনটি ছোট আর কোনটি বড়? তখন আমরা এই x এ segment দুইটির y বের করে বলে দেব কোনটি বড় আর কোনটি ছোট। সুতরাং set এ modification এর আগে আমাদের খেয়াল রাখতে হবে x এর মান যেন সঠিক হয়। যেমন, তুমি কোন একটি segment শুরু event এর সাপেক্ষে set এ কোন একটি segment ঢুকাতে চাও তাহলে তোমাকে x কে সেই event এর x এর সমান করে নিতে হবে insert এর আগে। একই ভাবে যখন segment শেষ এর event পাবে তখন segment কে remove এর আগে x কে সেই event এর x এর সমান করে নিতে হবে। এখন যদি কোন intersection পাও তাহলে তো আমাদের swap করতে হবে তাই না? এটা কেমনে করা যায়? খুব সহজ, মনে করো intersection এর event এর x হল x'. তাহলে প্রথমে $x = x' - \epsilon$ সেট করো এবং সেই দুইটি segment কে set হতে remove করো, এর পর $x = x' + \epsilon$ সেট করে তাদের আবার insert করো, তাহলেই তারা swap হয়ে যাবে, এখানে ϵ হল খুব ছোট মান। insertion বা removal এর পর কোন একটি segment নিয়ে তার lower bound বা upper bound করে আমরা খুব সহজেই তার ঠিক আগে এবং পরের segment গুলি বের করে ফেলতে পারি। এভাবেই আমরা $\mathrm{O}(I \log n)$ এ এই সমস্যার সমাধান করতে পারি যেখানে I হল segment গুলির মাঝের number of intersections.

১০.৩.৪ Pick's theorem

Pick's theorem বলে 2d grid এ যদি আমরা কোন simple polygon আঁকি অর্থাৎ এমন একটি পলিগন যেখানে কোন বাহু ওপর বাহুর সাথে ছেদ করে না বা স্পর্শও করে না তাহলে $A = i + \frac{b}{2} - 1$ হবে (পলিগন এর সকল শীর্ষ বিন্দুকে অবশ্যই integer coordinate এ থাকতে হবে)। এখানে A হল পলিগন এর ক্ষেত্রফল, i হল পলিগন এর অভ্যন্তরে(internal) কতগুলি integer coordinate আছে এবং b হল পলিগন এর boundary এর উপরে কতগুলি integer coordinate আছে। যেমন চিত্র ১০.৩ এ লাল বিন্দু গুলি হল অভ্যন্তরের বিন্দু $i = 7$, সবুজ বিন্দুগুলি boundary এর উপরের বিন্দু $b = 8$ এবং পুরো পলিগনের ক্ষেত্রফল $A = 10$. এখানে বলে রাখা যায় যে, চিত্রের পলিগন এর জন্য ক্ষেত্রফল বের করা কিন্তু খুব একটা কঠিন না, তুমি কল্পনা করো $y = 2$ এই লাইনের উপরে একটি ত্রিভুজ এবং নিচে একটি ত্রিভুজ। আর ত্রিভুজের ক্ষেত্রফল তো বের করা খুবই সোজা। যাই হোক, সুতরাং আমরা এখন A, i এবং b এর মান জানি, pick's formula তে বসিয়ে দেখি এটা কাজ করে কিনা! $10 = 7 + \frac{8}{2} - 1$ একদম সঠিক!

এখন একটা সমস্যা দেখা যাক। মনে করো তোমাদের 2d coordinate এ একটি ত্রিভুজ দেয়া আছে। এই ত্রিভুজের উপর (বাহুর উপর) এবং এই ত্রিভুজের অভ্যন্তরে কতগুলি বিন্দু (integer বিন্দু বা

integer coordinate বা lattice point) আছে তা বের করতে হবে। কিছুটা pick's theorem এর গন্ধ আছে। ফর্মুলার দিকে তাকালে আমরা বুঝতে পারব যে শুধু মাত্র এই ত্রিভুজের ক্ষেত্রফল আমাদের জানা। আমরা যদি কোনভাবে ভিতরে কয়টি বিন্দু আছে সেটা বা বাহুর উপরে কয়টা বিন্দু আছে সেটা বের করতে পারি তাহলে ওপরটাও বের হয়ে যাবে। বাহু গুলির উপরে কয়টা বিন্দু আছে তা বের করা সহজ। আমরা যদি একটি বাহুর উপরে কয়টা বিন্দু আছে তা বের করতে পারি তাহলে ত্রিভুজের তিনটি বাহুর উপরে কয়টা বিন্দু আছে তাও বের করে ফেলতে পারব। ধরা যাক আমরা বের করতে চাই $(x_1, y_1) - (x_2, y_2)$ এই বাহুর উপরে কয়টা বিন্দু আছে। আমরা এখন একটু একটু করে প্রবলেমটাকে সহজ করব, প্রথমে দেখো $(x_1, y_1) - (x_2, y_2)$ সেগমেন্ট দেখা আর $(0, 0) - (x_1 - x_2, y_1 - y_2)$ এই সেগমেন্ট দেখা একই কথা। আবার $(0, 0) - (|x_1 - x_2|, |y_1 - y_2|)$ দেখাও কিন্তু একই কথা। সুতরাং আমাদের আসলে বের করতে হবে $(0, 0) - (x, y)$ বাহুর উপরে কয়টা বিন্দু আছে যেখানে $x, y \geq 0$. এর উত্তর হল $gcd(x, y) + 1$. কেন? ধর g হল তাদের gcd, তাহলে এর উপরে থাকা বিন্দু গুলি হলঃ $(x = g \times x/g, y = g \times y/g), ((g - 1) \times x/g, (g - 1) \times y/g), \ldots (0 \times x/g, 0 \times y/g)$. আশা করি বাকিটুকু নিজেরাই করতে পারবে!

১০.৩.৫ Polygon সম্পর্কিত টুকিটাকি

একটি পলিগন অর্থাৎ বহুভুজ দিয়ে যদি বলে এই পলিগনের ক্ষেত্রফল বের করতে হবে কেমনে করবে? আলোচনার সুবিধার জন্য ধরে নিলাম আমাদের পলিগন হল $P_0P_1P_2 \ldots P_{n-1}$. অনেকে মনে করতে পারো যে $\triangle P_0P_1P_2, \triangle P_0P_2P_3, \ldots \triangle P_0P_{n-2}P_{n-1}$ এই ত্রিভুজগুলির ক্ষেত্রফল যোগ করলেই তো পুরো পলিগন এর ক্ষেত্রফল বের হয়ে যাবে। না তা ঠিক না। এটা ঠিক হবে সকল convex polygon এর জন্য। Concave polygon এর জন্য এটা সত্য নাও হতে পারে। একটি পলিগনের কোন কোণই যদি $180°$ এর চেয়ে বড় না হয় তাহলে তাকে convex polygon বলে। আর যদি কোন একটি $180°$ এর থেকে বড় হয় তাহলে তাকে concave polygon বলে। তাহলে আমরা কেমনে বের করতে পারি? উত্তর হল signed area ব্যবহার করে। signed area কি? আমরা এই অধ্যায়ের শুরুর দিকে ত্রিভুজের ক্ষেত্রফল বের করার জন্য determinant ব্যবহার করেছিলাম। সেভাবে বের করার জন্য বলেছিলাম যে absolute value নিতে। কিন্তু তুমি যদি absolute value না নাও এবং উপরের মত $\triangle P_0P_1P_2, \triangle P_0P_2P_3, \ldots \triangle P_0P_{n-2}P_{n-1}$ এর এই signed value গুলি যোগ করো তাহলেই তোমরা পুরো পলিগনের signed ক্ষেত্রফল পেয়ে যাবে। অর্থাৎ এই signed মান গুলি যোগ করে যদি তুমি absolute value নাও তাহলে তুমি ক্ষেত্রফল পাবে। তুমি চাইলে এই কাজ P_0 কে কেন্দ্র করে না করে মূলবিন্দুকে কেন্দ্র করেও করতে পারো। তবে আমি এই ভাবে করি না। আমি যা করি তাহলো $\sum (x_iy_{i+1} - y_ix_{i+1})$ (i এর মান $n - 1$ হলে $i + 1 = 0$ ধরতে হবে কিন্তু!)। এটা আমার জন্য মনে রাখা সহজ তবে উপরের মেথডটা কিন্তু একটা বেশ দরকারি মেথড। অন্যান্য অনেক কাজেই এই signed মান ব্যবহার করে অনেক সহজে প্রবলেম সমাধান করা যায়।

মনে করো তোমাদের একটি পলিগন আর একটি বিন্দু দিয়ে বলল যে এই বিন্দুটি পলিগনের ভিতরে আছে না বাহিরে? কেমনে করবে? যদি এটি convex polygon এর ক্ষেত্রে হয় তাহলে কিন্তু ব্যপারটা বেশ সহজ। মনে করো তোমাদের বিন্দুটি হল (x', y') তাহলে $y = y'$ এ পলিগনের দুইটি ছেদবিন্দু বের করতে হবে। যদি তোমার x' এই দুই ছেদবিন্দুর x এর মাঝে থাকে তাহলে বলা যাবে যে আমাদের বিন্দু পলিগনের ভিতরে আছে। পলিগনের সাথে কোন একটি $y = y'$ লাইনের ছেদবিন্দু বের করা কিন্তু কঠিন কিছু না। তুমি সকল বাহু দেখবে এবং তাদের জন্য সমাধান করবে, যদি কোন শীর্ষ বিন্দুতে ছেদ করে তাহলে একটু সাবধান হতে হবে আর কি! যাই হোক, আমাদের এই মেথড কিন্তু concave polygon এ কাজ করবে না। আমরা বিভিন্ন ভাবে এই সমস্যা সমাধান করতে পারি। একটি উপায় হল প্রদত্ত বিন্দু থেকে যেকোনো একদিকে রশ্মি টান। খেয়াল রাখতে হবে যেন এই রশ্মি পলিগনের কোন শীর্ষ দিয়ে যেন না যায়। তুমি এইকাজ খুব সহজে একটি random দিক নির্বাচন করে করতে পারো। যদি সেই দিকে কোন শীর্ষ বিন্দু থাকে তাহলে আরেকটা random দিক নিবা, এভাবে চলতে থাকবে। আসলে দুই একবারের বেশি লাগার কথা না। এখন তোমাকে দেখতে হবে যে এই রশ্মি তোমার পলিগন কে কয় জায়গায় ছেদ করে। যদি এই সংখ্যা বিজোড় হয় তার মানে তুমি পলিগনের ভিতরে আছ আর যদি জোড় সংখ্যক হয় তার মানে তুমি বাহিরে আছ। এর থেকেও সহজ উপায় আছে সমাধান করার। সেটা হল winding number. তোমার বিন্দুর চারিদিকে পলিগন কয় পাক মারে সেটাই হল

winding number. তোমার বিন্দু সাপেক্ষে সব গুলি বাহুর জন্য signed angle এর যোগফল বের করলেই তুমি সমাধান করে ফেলতে পারবে। কিন্তু এটা একটু ঝামেলা, এর থেকে এর আগের মেথড ray shooting এর মত করে আমি সমাধান করে থাকি। আমরা ঠিক ডান দিকে একটি রশ্মি টানি। যেহেতু তোমার query বিন্দু এর সমান y আলা একটি বিন্দু পলিগনের শীর্ষ বিন্দু হতে পারে সেহেতু আমরা আমাদের প্রদত্ত বিন্দুটির y coordinate কে $y - \epsilon$ হিসাবে ভাবতে পারি, অর্থাৎ (x, y) যদি ভিতরে থাকে তাহলে $(x, y - \epsilon)$ ও ভিতরে থাকবে, সুবিধা হল $y - \epsilon$ এ আঁকা x অক্ষের সমান্তরাল রেখা কোন শীর্ষ বিন্দু দিয়ে যায় না। এখন প্রতিটি বাহু নিতে হবে আর দেখতে হবে এটি এই রশ্মিকে ছেদ করে কিনা। যদি করে তাহলে সেই বাহুটি নিচ থেকে উপরে যাচ্ছিলো নাকি উপর থেকে নিচে? ধরা যাক এই দুইটি ক্ষেত্রে যথাক্রমে আমরা 1 যোগ ও বিয়োগ করি (আসলে এক অর্থে এটা জোড় বিজোড় বের করা)। তাহলে সকল বাহু বিবেচনা করার পর যদি দেখি আমাদের যোগফল শূন্য তাহলে আমরা বুঝব যে আমাদের বিন্দু বাহিরে আছে, আর যদি শূন্য না হয় এর মানে এই বিন্দু ভিতরে আছে।

মনে করো তোমাদের কিছু পলিগন দেয়া আছে, বলা হল এদের union এর ক্ষেত্রফল বের করতে। union বলতে আমরা বুঝি যদি কোন জিনিস একাধিক বার কাভার হয়ে থাকে তবুও সেই জিনিসকে আমরা একবারই বিবেচনা করব। যেমন ধর দুইটি বর্গক্ষেত্র নেয়া হল যেন একটি আরেকটিকে পুরপুরি ভাবে কাভার করে। তাহলে তাদের union এর ক্ষেত্রফল হবে বড়টির ক্ষেত্রফলের সমান। এখন এই সমস্যার সমাধান কি? প্রথমে সব পলিগন গুলিকে একদিকে orient করে নাও। একদিকে orient করা মানে হল হয় সবাই cw অথবা ccw. concave polygon এর cw বা ccw বলে কিন্তু কোন কথা নেই বলতে পারো। কিন্তু তুমি চাইলে signed ক্ষেত্রফল বের করে কোন পলিগন cw না ccw তা বলতে পারো। সুতরাং এভাবে তোমাকে সকল পলিগন কে একই দিকে orient করে নিতে হবে। এরপর আমাদের যা করতে হবে তাহল প্রতিটি পলিগনের প্রতিটি বাহু নিতে হবে এবং আমাদের অন্যান্য সকল বাহু দ্বারা একে ছেদ করার চেষ্টা করতে হবে, এভাবে আমাদের বিবেচিত বাহু এর উপর অন্যান্য সকল বাহুর ছেদ বিন্দু বের করি। যদি আমরা parametric equation ব্যবহার করি তাহলে খুব সহজেই এই ছেদ বিন্দু গুলিকে sort করে আমরা কতিপয় segment পাব। আমাদের বের করতে হবে এই সেগমেন্ট গুলির কোন কোন গুলি অন্য পলিগনের ভিতরে আছে। এটা বের করার সহজ উপায় হল এই সেগমেন্ট এর মধ্যবিন্দু নিয়ে অন্যান্য পলিগনের ভিতরে আছে কিনা তা চেক করা। যেহেতু আগেই আমরা বাহুকে সেগমেন্ট এ ভাগ করে ফেলেছি সুতরাং এমন হবে না যে কোন সেগমেন্ট partially কোন পলিগন এর ভিতরে আছে। যদি কোন সেগমেন্ট অন্য কোন পলিগন এর ভিতরে থেকে থাকে তাহলে তাকে ধরে ফেলে দাও! এখন বেঁচে থাকা segment গুলি নিয়ে যেকোনো বিন্দু (ধরা যাক মূলবিন্দু) এর সাপেক্ষে ক্ষেত্রফল বের করো, আশা করি এটা বলতে হবে না যে ক্ষেত্রফলটি signed হবে। খেয়াল করো তুমি কিন্তু প্রথমেই পলিগনকে orient করে নিয়েছিলে, সুতরাং প্রতিটি segment এর একটি দিক আছে, সেই দিক ব্যবহার করে তোমাদের signed area বের করতে হবে। এভাবে সকল signed area যোগ করলে তুমি area এর union পেয়ে যাবে। আমি যেই সমাধান এখানে বলেছি, মনে হয় তার থেকেও ভাল সমাধান আছে। কিন্তু এই মুহূর্তে ঠিক বের করতে পারছি না। কিন্তু এই সমাধানটা খুবই চমৎকার একটি সমাধান যা তোমাদের জেনে রাখা উচিত।

১০.৩.৬ Line sweep এবং Rotating Calipers

Line sweep বা Rotating calipers কোন অ্যালগরিদম না, বরং একটি solving technique. যেমন আমরা closest pair of points প্রবলেমে যেই দ্বিতীয় সমাধান দেখেছিলাম সেটাকে line sweep বলা যায় কারণ আমরা 2d coordinate system এ একদিক হতে আরেকদিকে গিয়েছিলাম। Rotating calipers হল কিছুটা line sweep এর মত, তবে প্রধান পার্থক্য হচ্ছে line sweep এ আমরা একটি linear দিকে move করি আর rotating calipers এ আমরা angular sweep করে থাকি। অনেক সময় শুধু sweep না, একটি window রেখে sweep করা হয়ে থাকে। আমরা এই সংক্রান্ত কিছু সমস্যা এখন দেখব।

ধরা যাক একটি convex polygon দিয়ে বলা হল এর সব থেকে বড় diagonal বের করতে হবে। যেহেতু পলিগনটি convex সুতরাং এটা বলাই যায় যে এর যেকোনো diagonal সম্পূর্ণ ভাবে পলিগনের ভিতরে থাকবে। এখন আমরা এই সমস্যা কে একটু ভেঙ্গে ভেঙ্গে দেখি। মনে করো আমরা যদি প্রতিটি শীর্ষবিন্দু থেকে বের হওয়া diagonal এর মাঝে সবচেয়ে বড়টা বের করতে পারি তাহলে

তাদের মাঝে সবচেয়ে বড়টিই আমাদের উত্তর। ধরা যাক, i তম শীর্ষের জন্য $f(i)$ হল ওপর শীর্ষ, তাহলে একটু খেয়াল করলে বুঝবে যে $i+1$ জন্য ওপর শীর্ষটা আসলে $f(i)$ বা এর কাছাকাছি কোন একটা। অন্যভাবে বলা যায়, i থেকে cw দিকে গিয়ে যদি তুমি $i+1$ পাও তাহলে $f(i)$ বা $f(i)$ এর থেকে কিছুটা cw গেলে তুমি $f(i+1)$ পাবে। সুতরাং আমাদের যা করতে হবে তাহলো প্রথমে $i=0$ নিতে হবে এবং এর জন্য আমাদের $j=f(0)$ বের করতে হবে। বের করার জন্য যা করবে তাহলো, $j=i=0$ ধরবে এবং দেখবে যে (i,j) diagonal বড় নাকি $(i,j+1)$ diagonal বড়। যদি পরেরটা বড় হয় তাহলে j কে এক বাড়িয়ে দিবে এবং পুনরায় একই চেক করবে। এভাবে করতে করতে দেখবে এক সময় আর j কে বাড়ান যাবে না কারণ (i,j) diagonal $(i,j+1)$ থেকে বড়। তোমাদের এই মানই হবে $f(0)$. এবার তুমি i কে এক বাড়াও, এবং কিছু ক্ষণ আগে যেই কাজ করেছি সেই কাজ আবার করব। j এর মান যা ছিল সেখান থেকেই শুরু হবে, 0 করা বা i এর সমান করার দরকার নেই। এভাবে দরকার মত j বাড়িয়ে $f(i)$ বের করে ফেলবে এবং এই কাজ $i=0$ হতে $n-1$ পর্যন্ত করবে (n হল পলিগনের শীর্ষ সংখ্যা) অর্থাৎ $f(0)\dots f(n-1)$ এর মানগুলি বের করবে। আসলে এই ভাবে সব $f(i)$ এর মান দরকার নেই, তুমি প্রতিবার যদি সর্বোচ্চ diagonal এর মান আপডেট করো তাহলেই হবে। এখন কথা হল এই সমাধানের complexity কত? মাত্র $\mathrm{O}(n)$. কারণ একটু চিন্তা করলে দেখবে যে j কে $2n$ বার এর বেশি বাড়ানোর দরকার হবে না। এখানে একটা জিনিস বলে নেই তাহলো, যদি প্রবলেম এ বলা থাকে যে পলিগনের যেকোনো বাহুও একটি diagonal তাহলে এই সমাধান ঠিক আছে। তবে যদি বলে যে diagonal টি কোন বাহু হতে পারবে না, তাহলে j এর মান fix করার আগে একটু ভালমতো দেখতে হবে। যেমন i কে বাড়ানোর পর পরই দেখতে হবে যে j কি $i+1$ এর থেকে ছোট? তাহলে j কে $i+2$ করে দিতে হবে। আবার j কে বাড়ানোর পর দেখতে হবে যে এটি কি $i-1$ এর সমান হয়ে গেছে কিনা তাহলে আর হবে না, $i-2$ তে পরিবর্তন করে দিতে হবে। তোমরা আশা করি বুঝতে পারছ যে $j=n-1$ কে যদি এক বাড়াতে হয় তাহলে কি করবে? তোমরা চাইলে if-else লাগাতে পারো অথবা mod অপারেশন ব্যবহার করতে পারো। তবে n এর মান খুব ছোট হলে কিছুক্ষণ আগে যে বললাম যে "বাড়ানো কমানোর সময় খেয়াল রাখতে হবে" এই জিনিসটা হয়তো তোমাদের চিন্তা করতে একটু কষ্ট হবে, সেজন্য যেটা সহজ বুদ্ধি তাহলো $n<10$ হলে সাধারণ $\mathrm{O}(n^2)$ পদ্ধতি ব্যবহার করা। তাহলে আরও এই বাড়ানো কমানো নিয়ে আলাদা চিন্তা করতে হবে না।

এখন উপরের সমস্যাকে একটু পরিবর্তন করা যাক। উপরের সমস্যায় বড় diagonal বের করতে না দিয়ে যদি বলত convex polygon টির ভিতরে থাকে এরকম সবচেয়ে বড় line segment বের করতে হবে, তাহলে কি করতে? একটু চিন্তা করে দেখো, এই বড় line segment কে অবশ্যই একটি diagonal হতে হবে। যদি তা নাহয় তাহলে একটু কল্পনা করো, তোমার বের করা line segment এর দুই মাথাকে অবশ্যই পলিগনের উপরে হতে হবে কারণ তা না হলে তুমি ঐ line segment কে বড় করতে পারবে। এখন এই segment এর এমন একটি মাথা নাও যেটি পলিগনের শীর্ষে নেই। যেহেতু এটি একটি বাহুর উপরে আছে সুতরাং তুমি সেই বাহু দিয়ে ওপর প্রান্তকে দুইদিকের যেকোনো একদিকে slide করলে অবশ্যই বড় segment পাবে। কেন? কারণ খুব সহজ, তোমাকে যদি জিজ্ঞাসা করা হয় একটি বিন্দু আর একটি line দেয়া আছে তোমাকে ঐ বিন্দু হতে ঐ line এর উপর সবচেয়ে ছোট segment আঁকতে হবে তুমি কি করবে? লম্ব টানবে। এই লম্ব থেকে যেদিকেই যাও সেদিকেই বাড়বে। অর্থাৎ আমাদের ক্ষেত্রে আমরা বাহুর উপরের বিন্দুকে আমরা যদি লম্বর বিপরীত দিকে slide করতে থাকি তাহলে আমাদের line segment এর দৈর্ঘ্য বাড়তে থাকবে।

এখন আরও একটি সমস্যা দেখা যাক। মনে করো এবার তোমাদের বলা হল একটি convex polygon এর ভিতরে সবচেয়ে বড় ক্ষেত্রফলের ত্রিভুজে বের করতে হবে। যদি আমি বলি যে ত্রিভুজটির তিনটি শীর্ষই polygon এর কোন না কোন শীর্ষে থাকবে তাহলে আশা করি খুব একটা অবাক হবে না। এর কারণটাও বেশ সহজ। ত্রিভুজের এমন একটি শীর্ষ নাও যেটা পলিগনের শীর্ষে নেই। ওপর দুই শীর্ষকে যথাস্থানে রাখ। এখন যেই দুই শীর্ষ যথাস্থানে আছে তাদের মাঝের বাহুকে ত্রিভুজের ভূমি মনে করো, তাহলে ত্রিভুজের ক্ষেত্রফলের সূত্র অনুসারে ত্রিভুজটির উচ্চতা যদি বাড়ে ক্ষেত্রফলও তাহলে বাড়বে। এখন তুমি এই ভূমির সমান্তরাল কিছু line টান। সবচেয়ে দূরের যেই সমান্তরাল লাইন পলিগন কে ছেদ করে সেই ছেদবিন্দুতেই তুমি সবচেয়ে বড় ক্ষেত্রফলের ত্রিভুজ পাবে। আর এটা বলার অপেক্ষা রাখে না যে সেই ছেদবিন্দুগুলির একটি অবশ্যই পলিগনের শীর্ষ হবে। তাহলে কেমনে সমাধান হবে আশা করি বুঝতে পারছ? প্রথমে $i=0, j=1, k=2$ নাও। এবার k কে বাড়াতে থাক যতক্ষণ

$i - j - k$ ত্রিভুজের ক্ষেত্রফল বাড়তে থাকে। এখন j কে বাড়াও দেখো k কে বাড়ালে ক্ষেত্রফল বাড়ে কিনা, যদি বাড়ে তাহলে আবার j কে বাড়ানোর চেষ্টা করো, এভাবে কিছুক্ষণ j আর k কে বাড়াতে থাকলে দেখবে আর বাড়ানো যাচ্ছে না। তখন তুমি i কে বাড়াবে এবং একই ভাবে আবারো $j - k$ কে বাড়ানোর চেষ্টা করবে। এই সমাধানও $\mathrm{O}(n)$. তবে এটা কেন সঠিক তা প্রমাণ করা একটু কঠিন। তোমরা চাইলে নিজেরা প্রমাণ করে দেখতে পারো অথবা internet এ খুঁজে দেখতে পারো। সত্যি বলতে আমার নিজেরও প্রমাণটা জানা নেই তবে এটুকু জানি যে এটা সঠিক সমাধান!

এতক্ষণ polygon এর ভিতরের জিনিস নিয়ে সমস্যা দেখেছি এবার একটু বাহিরের জিনিস নিয়ে দেখা যাক। মনে করো অনেক গুলি বিন্দু দেয়া আছে, তোমাদের সবচেয়ে ছোট ক্ষেত্রফলের আয়তক্ষেত্র বের করতে হবে যেন সকল বিন্দু এই আয়তক্ষেত্রের ভিতরে থাকে। তুমি যদি internet এ সার্চ করো তাহলে wikipedia তে দেখবে minimum enclosing rectangle বা এরকম কোন নামে একটি আর্টিকেল আছে এবং এতে বলা আছে যে এই আয়তক্ষেত্রটির একটি বাহু প্রদত্ত বিন্দু গুলি দিয়ে তৈরি convex hull এর কোন একটি বাহু দিয়ে যায়। মনে করো আমরা convex hull বের করে ফেলেছি এবং এর একটি বাহু $(i, i{+}1)$ দিয়ে আয়তক্ষেত্রটি যায়। তাহলে optimal আয়তক্ষেত্রের ক্ষেত্রফল কত (যেন এই বাহু দিয়ে যায়)? আমরা কিন্তু খুব সহজেই আয়তক্ষেত্রের উচ্চতা বের করতে পারি কিছুক্ষণ আগে শেখা উপায়ে। কিন্তু প্রস্থ বা দৈর্ঘ্য কেমনে বের করতে পারি? এটাও খুব একটা কঠিন না, উচ্চতা বের করার মত করে আমরা বাম দিকের boundary আর ডান দিকের boundary বের করতে পারি। কোন একটা বিন্দু থেকে আমরা ঐ বাহুর উপর লম্ব টানব। এই লম্ব ডানে আর বামে যত দূরে নেয়া যায় নিব (লুপ চালিয়ে যেভাবে উচ্চতা বাড়তে থাকা পর্যন্ত আমরা index বাড়িয়েছি সেভাবে)। তাহলেই আমরা আয়তক্ষেত্রের প্রস্থও পেয়ে যাব। সুতরাং $(i, i + 1)$ দিয়ে যাওয়া optimal আয়তক্ষেত্রের ক্ষেত্রফল আমরা জেনে গেলাম। একই ভাবে আমরা i কে বাড়াব এবং উচ্চতা, ডান আর বাম দিকের vertex এর pointer কে আমরা আপডেট করতে থাকব। এভাবে আমরা $\mathrm{O}(n \log n)$ এ convex hull বের করার পর $\mathrm{O}(n)$ এ optimal আয়তক্ষেত্র বের করতে পারি।

এবার Line sweep এর কিছু সমস্যা দেখার আগে এর কিছু মূল জিনিস দেখে নেয়া যাক। 2d grid এ line sweep এর সময় আমরা মূলত যা করি তাহলো একটি অক্ষ বরাবর এক দিক থেকে আরেকদিকে ধীরে ধীরে যাই (sweep) এবং ওপর অক্ষ বরাবর একটি data structure রেখে তাকে ধীরে ধীরে আপডেট করি। প্রথম অক্ষ বরাবর যে ধীরে ধীরে যাই এর মানে হল আমরা কেবল মাত্র আমাদের জরুরি স্থান গুলিতেই থামব, যেমন closest pair of points সমস্যায় কিন্তু আমরা শুধু মাত্র সেসব x এই থেমেছিলাম যেখানে কোন বিন্দু ছিল। কোন জায়গা জরুরি বা next কোন জায়গা জরুরি তা বের করার জন্য আমাদের আরেকটি data structure ব্যবহার করতে হয় সাধারণত। বিভিন্ন সমস্যায় বিভিন্ন data structure ব্যবহার করতে হয়, তবে মূল theme একই হয়ে থাকে। এখন কিছু সমস্যা দেখা যাক।

প্রথম সমস্যা হল union of rectangles. মনে করো তোমাকে অনেকগুলি আয়তক্ষেত্র দেয়া হল আর বলা হল এদের union এর ক্ষেত্রফল বের করতে। কিছুক্ষণ আগেই আমরা দেখেছি কেমনে অনেকগুলি convex পলিগনের union এর ক্ষেত্রফল বের করা যায়। কিন্তু সেই সমাধানের time complexity অনেক ছিল। যেহেতু এই প্রবলেমে আয়তক্ষেত্র দেয়া আছে (যা convex পলিগনের তুলনায় অনেক বেশি সহজ structure) সেহেতু আমরা আশা করতে পারি যে এর আগের সমাধানের তুলনায় এই সমস্যার একটি সহজ সমাধান থাকবে। ইনপুট হিসাবে আয়তক্ষেত্রের উপরের ও নিচের y দেয়া আছে আর ডান ও বামের x দেয়া আছে। চল আমরা সমাধানটা দেখে নেই। আমরা x অক্ষের বাম থেকে ডান দিকে sweep করব এবং y অক্ষ বরাবর একটি segment tree রাখব। আরও বেশি দূর যাবার আগে আমাদের আরেকটা common technique জানতে হবে। সেটা হল coordinate compression. Geometry সমস্যা সমাধানের সময় এই টেকনিক প্রায়ই ব্যবহার হয়ে থাকে। এই টেকনিক এর মূল কথা হল সব coordinate সবসময় দরকার হয় না, যেসব coordinate লাগবে শুধু তাদের নিয়েই কাজ করা হল coordinate compression. আমাদের যা করতে হবে তাহলো আয়তক্ষেত্র গুলির ডান মাথা আর বাম মাথা (দুইটি করে x coordinate) একটি লিস্ট এ নিয়ে তাদের সর্ট করতে হবে। এর পর একটি লুপ চালিয়ে শুধু মাত্র distinct x coordinate গুলি বের করতে হবে। আমরা চাইলে এই distinct coordinate গুলি ঐ একই লিস্ট এ রাখতে পারি বা আলাদা লিস্ট এ নিতে পারি। তোমরা চাইলে এই কাজ set ব্যবহার করেও করতে পারো। একই ভাবে y এর

distinct coordinate সমূহও বের করতে হবে। ধরা যাক x এর জন্য যে লিস্ট পাওয়া গেছে তাহলঃ $x[1], x[2] \dots x[nx]$ আর y এর জন্য যে লিস্ট পাওয়া গেছে তাহলোঃ $y[1], y[2] \dots y[ny]$. এখন আমাদের প্রতিটি আয়তক্ষেত্রের coordinate সমূহ আপডেট করতে হবে। যদি কোন আয়তক্ষেত্রের উপরের বা নিচের y যদি হয় লিস্ট এর i তম element অর্থাৎ, $y[i]$ তাহলে তাকে i করে দাও। একই ভাবে x coordinate গুলিকে একই ভাবে পরিবর্তন করে দাও। এবার একটি vector এর অ্যারে নাও যার সাইজ হবে nx. এবার আয়তক্ষেত্রের উপর লুপ চালাও এবং প্রতিটির বামের x এর জন্য তার vector এ লিখে রাখ যে এই x থেকে অমুক আয়তক্ষেত্রের boundary শুরু হয়েছে, একই ভাবে অমুক x এ গিয়ে অমুক আয়তক্ষেত্র শেষ হয়েছে। তোমরা চাইলে এই কাজ vector রেখে না করে একটি heap বা priority queue রেখেও করতে পারতে। অথবা একটি vector এও event গুলি রেখে সর্ট করতে পারতে। যাই হোক, অন্যদিকে y অক্ষ এর জন্য আমাদের একটি segment tree বানাতে হবে যার সাইজ হবে $ny-1$. Segment tree এর নোড গুলি হবেঃ $(1, 2), (2, 3) \dots (ny-1, ny)$. আরও ভাল মত বলতে গেলে এটা হবেঃ $(y[1] \sim y[2]), (y[2] \sim y[3]) \dots (y[ny-1] \sim y[ny])$. এখন আমাদের x অক্ষ বরাবর ধীরে ধীরে আগাতে হবে। প্রতিটি x এ যাব আর তার vector এ থাকা সব event কে আমাদের execute করতে হবে। বুঝাই যাচ্ছে যে event দুই রকম হতে পারে, এক আয়তক্ষেত্রের শুরু আরেকটা হল আয়তক্ষেত্রের শেষ। আয়তক্ষেত্র শুরুর সময় আমাদের যা করতে হবে তাহলো ঐ আয়তক্ষেত্রের উপরের আর নিচের y যদি হয় যথাক্রমে $y[hi]$ এবং $y[lo]$ তাহলে $(y[lo] \sim y[lo+1]), (y[lo+1] \sim y[lo+2]) \dots (y[hi-1] \sim y[hi])$ এই সেগমেন্ট এর প্রতিটি স্থানে 1 যোগ করতে হবে বা সংক্ষেপে $[lo, hi-1]$ এই সেগমেন্ট এর প্রতিটি স্থানে 1 যোগ করতে হবে। তাহলে আশা করি বুঝা যাচ্ছে যে আয়তক্ষেত্রের শেষ মাথার event এ তোমাকে 1 করে বিয়োগ করতে হবে। এখন একটা জিনিস খেয়াল করো, তুমি যদি $x[i]$ এ যেসব event আছে সেসব প্রসেস করা শেষে যদি segment tree তে যেসব জায়গায় non-zero মান আছে সেসব জায়গার মান (s এ non-zero থাকলে $y[s] - y[s-1]$ যোগ হবে) যোগ করো এবং তাকে $(x[i+1] - x[i])$ দিয়ে গুন করো তাহলে $x[i]$ থেকে $x[i+1]$ এর ভিতরে আয়তক্ষেত্রের union পাবা। এভাবে প্রত্যেক x যদি প্রসেস করো এবং যোগ করো তাহলেই তুমি তোমার কাঙ্ক্ষিত ক্ষেত্রফলের union পেয়ে যাবে। segment tree তে query কেমনে করবা তা তোমাদের জন্য রেখে দেয়া হল।

এবার একটা সহজ সমস্যা দেখা যাক। মনে করো তোমাকে 2d coordinate এ অনেক গুলি axis parallel line segment দেয়া আছে। তোমাকে বলতে হবে তাদের মাঝে কতগুলি intersection আছে। মনে করো line segment এর সংখ্যা প্রায় $n \leq 100,000$ টি এবং coordinate গুলি সর্বোচ্চ 10^9 হতে পারে। আশা করি বুঝা যাচ্ছে যে প্রথমে তোমাদের coordinate compress করে ফেলতে হবে। এর পরে x axis বরাবর একটি সেগমেন্ট ট্রি নাও এবং y axis বরাবর sweep করো। মনে করা যাক উপর হতে নিচে sweep করা হচ্ছে। sweep করার মানে হচ্ছে event process করা। যদি y axis এর সমান্তরাল একটি সেগমেন্ট এর শুরুর মাথা পাও তাহলে সেগমেন্ট ট্রি তে ঐ জায়গায় 1 বাড়াও। যদি শেষ মাথা পাও তাহলে 1 কমাও। আর যদি x axis এর সমান্তরাল একটি লাইন পাও যার x এর বিস্তার $[x_1, x_2]$ তাহলে সেগমেন্ট ট্রি তে এই রেঞ্জ এর জন্য একটা query করে ঐ রেঞ্জ এর যোগফল বের করতে হবে। এই যোগফল তোমার উত্তর এর সাথে যোগ করতে থাক। তাহলেই তুমি মোট উত্তর পেয়ে যাবে। সহজ না?

এবার একটা IOI এর সমস্যা দেখা যাক। মনে করো n টি axis parallel rectangle দেয়া আছে। তোমাকে এদের boundary এর length বের করতে হবে। সমস্যাটা খুব একটা কঠিন না, একটু বুদ্ধি খাটালে সমস্যাটা সহজ হয়ে যাবে। এখানে চার ধরনের boundary হতে পারে। উপরে, নিচে, ডানে আর বামে। আবার উপরের boundary কিন্তু নিচের সমান। একই ভাবে ডানেরটা বামের সমান। সুতরাং আমাদের দুইটা বের করলেই চলে। আবার যদি আমরা উপরেরটা বের করতে পারি তাহলে x আর y swap করে একই ভাবে ডানেরটা বের করতে পারি। সুতরাং আমরা এখন শুধু উপরেরটা কেমনে বের করে তা দেখব। এখন আগের মতই x axis বরাবর সেগমেন্ট ট্রি আর y axis বরাবর sweep করব আমরা। আর উপরের boundary বের করার জন্য কিন্তু শুধু x axis এর parallal line segment ই লাগবে y axis এর parallal গুলির কোন প্রয়োজন নেই। তবে অবশ্যই segment গুলির সাথে একটা information লাগবে তাহলো এই সেগমেন্টটি আয়তক্ষেত্রের উপরের মাথা নাকি নিচের মাথা। এখন sweep করার সময় তুমি যদি উপরের মাথা অর্থাৎ শুরুর মাথা পাও তাহলে ট্রি তে query করো যে ঐ রেঞ্জ এ কতগুলি শূন্য আছে অর্থাৎ এই সেগমেন্ট এর কত খানি অংশ ঢেকে নেই। সেটা উত্তর এর সাথে যোগ করো। এই query শেষে তোমাকে এই পুরো রেঞ্জ এর সকল সংখ্যায় 1

যোগ করে দিবে। আর নিচের মাথা পেলে কি করতে হবে তা বলার দরকার দেখি না। যদি coordinate খুব বড় হয় তাহলে coordinate compress করে নিবে- এই কথাও এতক্ষণে ডাল ভাত হয়ে যাবার কথা।

এবার মনে করো তোমাদের 2d coordinate এ কিছু বিন্দু দেয়া আছে এবং কিছু axis parallel rectangle দেয়া আছে। তোমাদের বলতে হবে প্রতিটি আয়তক্ষেত্রের ভিতরে কতগুলি বিন্দু আছে অর্থাৎ প্রতিটি আয়তক্ষেত্রের জন্য আলাদা আলাদা ভাবে তোমাকে উত্তর দিতে হবে। যদি তোমরা ইতোমধ্যেই 2d segment tree এর নাম শুনে থাকো আর ভাব যে ঐ কঠিন ডাটা স্ট্রাকচার ব্যবহার করা লাগবে তাহলে ভুল ভেবে থাকবে। এটা আসলে আমাদের এতক্ষণ শেখা মেথড এই সমাধান করা যাবে। মনে করো আমরা x অক্ষ বরাবর sweep করছি আর y অক্ষ বরাবর সেগমেন্ট ট্রি আছে। যখন আমরা কোন একটি বিন্দু পাব তখন আমাদের সেগমেন্ট ট্রি তে সেই y এ 1 যোগ করতে হবে। যদি আয়তক্ষেত্রের বামের বাহু পাও তাহলে ঐ রেঞ্জ এ query করে সেই সংখ্যা মনে রাখ। একই ভাবে ডান মাথা পেলেও একই ভাবে query করে মনে রাখতে হবে। এখন প্রতিটি আয়তক্ষেত্রের জন্য ডানের মাথার জন্য পাওয়া মান থেকে বামের মাথার জন্য পাওয়া মান বিয়োগ করলে আমরা ঐ আয়তক্ষেত্রের ভিতরে কতগুলি বিন্দু আছে তা পেয়ে যাব।

মনে করো তোমাদের কিছু বিন্দু দেয়া হল আর বলা হল তুমি $w \times h$ সাইজের একটি আয়তক্ষেত্র কে এমন ভাবে বসাও যেন সবচেয়ে বেশি সংখ্যক বিন্দু এর ভিতরে থাকে। কেমনে করবে? সত্যি কথা বলতে এই সমস্যাটা শুনে একটু কঠিন কঠিনই লাগে। কিন্তু আমি যদি বলতাম, তোমাকে $w \times h$ সাইজের বেশ কিছু আয়তক্ষেত্র দেয়া আছে, তোমাকে এমন একটি বিন্দু বের করতে হবে যা সবচেয়ে বেশি সংখ্যক আয়তক্ষেত্রের ভিতরে থাকে। এই সমস্যা কিন্তু তুলনামূলক সোজা লাগার কথা। কারণ এই ক্ষেত্রে তুমি sweep করবে এবং sweep এর সময় কোন আয়তক্ষেত্র এর এক মাথা সেগমেন্ট ট্রি তে ঢুকানোর পর দেখবে সেই রেঞ্জ এ থাকা সকল সংখ্যার মাঝে সর্বোচ্চটি কত। ব্যাস শেষ! এখন কথা হল মূল সমস্যা কে এই সমস্যায় পরিণত করা যায় কেমনে? এটাও বেশ সহজ, মনে করো তোমাকে যদি (x, y) বিন্দু দেয় তাহলে তুমি মনে করো তোমাকে $(x - w, y - h) \sim (x, y)$ আয়তক্ষেত্র দিয়েছে। শেষ! কেন এমন করলাম? কারণ চিন্তা করে দেখো এই আয়তক্ষেত্রের মাঝে যদি আমরা $w \times h$ সাইজের একটি আয়তক্ষেত্র এর নিচের বাম কোণা বসাই তাহলে তা ঐ বিন্দুকে কাভার করে। বা অন্যভাবে বলা যায় আমরা সেই বিন্দুটি বের করছি যেখানে $w \times h$ আয়তক্ষেত্রের নিচের বামের বিন্দু বসালে সবচেয়ে বেশি বিন্দু পাওয়া যাবে।

কিন্তু উপরের সমস্যায় যদি দুইটি disjoint আয়তক্ষেত্র নির্বাচন করতে বলত যাতে দুইটি দ্বারা কাভার করা বিন্দুর সংখ্যা সবচেয়ে বেশি হয় তাহলে কি করতে? একটা জিনিস খেয়াল করো দুইটি আয়তক্ষেত্র যেভাবেই বসাও না কেন তুমি তাদের হয় horizontally নাহয় vertically একটি লাইন টেনে আলাদা করতে পারবে। অর্থাৎ তুমি আগের মত sweep করবে এবং প্রতি x এ লিখে রাখবে যে এই x এ যদি তুমি তোমার আয়তক্ষেত্রের নিচের বাম বিন্দু বসাতে তাহলে তুমি কতগুলি বিন্দু কাভার করতে পারবে। এবার এই পাওয়া মানগুলির উপর একটি খুব সহজ dp চালিয়ে তুমি উত্তর বের করে ফেলতে পারবে। এটা গেল যদি vertically ভাগ করলে, কিন্তু যদি তোমাকে horizontally ভাগ করতে হয় optimal উত্তর পাবার জন্য? সহজ, সব বিন্দুর x ও y coordinate swap করে দিয়ে উপরের কাজ আবার করো এবং দুইটি উত্তর থেকে যেটি বেশি ভাল সেইটি হবে তোমার আসল উত্তর।

১০.৩.৭ কিছু coordinate সম্পর্কিত counting

মনে করো একটি $n \times n$ grid (অর্থাৎ প্রতি side এ n টি করে lattice point থাকবে) দিয়ে বলা হল এতে কতগুলি বর্গক্ষেত্র আঁকা যায় যেন এর সকল শীর্ষ একটি lattice point হয়। খেয়াল করো, এখানে কিন্তু বলা হয় নাই বর্গ গুলি axis parallel হবে। সুতরাং $(1, 0), (0, 1), (-1, 0), (0, -1)$ ও একটি valid বর্গ। এই ধরনের সমস্যার ট্রিক হল তোমাকে একটি bounding box ধরতে হবে এবং এর ভিতরে ঠিক ঠিক ফিট করে এরকম কতগুলি বর্গ পাওয়া সম্ভব তা বের করতে হবে। এর পর এই সাইজের bounding box পুরো গ্রিড এ কতগুলি থাকতে পারে তা বের করে হিসাব নিকাশ করলেই আমাদের উত্তর বের হয়ে আসবে। ধরা যাক আমরা বের করতে চাই $m \times m$ গ্রিড এ fit করে এরকম কতগুলি বর্গ আছে? উত্তর সহজ, m টি। যদি তুমি $(i, 0)$ কে একটি শীর্ষ ধর তাহলে

$(m,i),(m-i,m),(0,m-i)$ হবে ওপর তিন শীর্ষ। এভাবে $i = 0 \ldots m-1$ এর জন্য তুমি একটি করে বর্গ পাবে যা $m \times m$ গ্রিড এ ফিট করে। একটু চিন্তা করে দেখতে পারো কোন $p \times q$ গ্রিডে কোন বর্গ ফিট করে না, সুতরাং তোমাকে $m = 1 \ldots n$ এর জন্য $m \times m$ গ্রিড বিবেচনা করতে হবে এবং প্রতিটিতে কতগুলি করে বর্গ সম্ভব তা বের করতে হবে। এখন প্রশ্ন হল $n \times n$ গ্রিডে কতগুলি $m \times m$ sub-grid আছে? সহজ $(n-m+1) \times (n-m+1)$ টি। বাকিটুকু কিছু সাধারণ গণিত। তুমি চাইলে এই পুরো calculation $\mathrm{O}(1)$ এ করতে পারো।

একই রকম আরও একটি সমস্যা দেখা যাক। আগের মতই তোমাদের $n \times m$ সাইজের একটি গ্রিড দেয়া আছে ($n, m \leq 1000$)। তোমাদের বের করতে হবে এর ভিতরে কতগুলি ত্রিভুজ আছে? এইবার সমস্যাটা একটু কঠিন লাগার কথা। যেহেতু সর্বমোট $n \times m$ টি বিন্দু আছে সুতরাং আমরা $\binom{nm}{3}$ ভাবে তিনটি বিন্দু নির্বাচন করতে পারি। সমস্যা হল যদি তিনটি বিন্দু একই রেখায় থাকে তাহলে সেটি valid ত্রিভুজ হবে না। এখন কত ভাবে তিনটি বিন্দু নির্বাচন করা যায় যেন তারা সরল রেখায় থাকে এই সংখ্যা বের করতে পারলে আমাদের সমাধান হয়ে যাবে। প্রথমত যেকোনো রেখার একটি bounding box থাকবে। সুতরাং আমাদের bounding box এর সাইজের উপর একটি লুপ চালাতে হবে। এখন যদি box এর কোনো একটি dimension যদি 1 হয় তাহলে আমরা এই count খুব সহজেই বের করে ফেলতে পারব। যদি 1 নাহয় তাহলে অবশ্যই রেখাটি কোন একটি কর্ণ বরাবর থাকবে। যেহেতু আয়তক্ষেত্রে দুইটি কর্ণ আছে সুতরাং একটি কর্ণের জন্য উত্তর বের করে দুই দিয়ে গুন করে দিলেই হয়ে যাবে। আর আমরা কিছুক্ষণ আগেই দেখেছি একটি কর্ণের জন্য উত্তর আমরা gcd ব্যবহার করে বের করে ফেলতে পারি।

যদি সামান্তরিক (parallelogram) বলত? তাও সহজ। দুই ধরনের case হতে পারে। চার কোনা bounding box এর চার বাহুর উপরে, অথবা দুই কোনা bounding box এর দুই কোনার উপর। যদি সামান্তরিক এর চার কোণা বক্স এর চার বাহুতে থাকে তাহলে একটা pattern দেখতে পারবা। সেটা হল- মনে করো bounding box এর উপরের বাহুতে সামান্তরিকের যেই শীর্ষ আছে তা বাহুকে $a : b$ অনুপাতে বিভক্ত করে তাহলে নিচের বাহুকে নিচের শীর্ষ $b : a$ অনুপাতে বিভক্ত করবে। একই কথা ডান ও বামের বাহুর উপরেও খাটবে। কিন্তু ডান-বামের অনুপাত উপর-নিচ এর অনুপাতের উপর নির্ভর করে না। সুতরাং আমরা এই pattern এ একটি (w,h) সাইজের আয়তক্ষেত্র থেকে প্রায় $(h-1) \times (w-1)$ টি সামান্তরিক পাব, আমি প্রায় শব্দ বললাম কারণ এটা নির্ভর করবে তুমি (w,h) বলতে কি বুঝাচ্ছ তার উপর, তুমি চাইলে w বলতে দৈর্ঘ্য ও বুঝাতে পারো আবার চাইলে দৈর্ঘ্য বরাবর কতগুলি lattice point আছে তাও বুঝাতে পারো। এতো details এ দেখানোর আমার ইচ্ছা নেই, বরং তোমাদের মূল idea দেখানোই আমার মূল উদ্দেশ্য। যাই হোক, এখন যদি সামান্তরিক এর দুই শীর্ষ যদি bounding box এর বিপরীত দুই শীর্ষে থাকে তাহলে? তাহলে ঐ দুই কোণা দিয়ে যেই diagonal যায় সেটা বাদে যেকোনো বিন্দুকে একটা শীর্ষ হিসাবে পছন্দ করলে বাকি শীর্ষ এমনিই পাওয়া যাবে। অর্থাৎ এখানে আমাদের gcd এর ফর্মুলা খাটাতে হবে। একটু সাবধানে এই দুই কেস সামলালেই তোমরা পুরো উত্তর পেয়ে যাবে যা $\mathrm{O}(n^2 \log n)$ এ বের হয়ে যাবে।

অধ্যায় ১১

String সম্পর্কিত ডাটা স্ট্রাকচার ও অ্যালগোরিদম

১১.১ Hashing

যদিও hashing ঠিক string সম্পর্কিত টেকনিক না কিন্তু hashing মনে হয় string এর সমস্যাতে বেশি ব্যবহার হয়ে থাকে। Hashing এর মূল theme হল তোমাকে একটা জিনিস দিবে, তোমার কাজ হল এই জিনিস কে একটি সংখ্যাতে পরিবর্তন করা। তবে এই পরিবর্তন এর প্রসেস এমন হতে হবে যেন এই জিনিসটা যত বারই দিক না কেন, আমাদের সংখ্যা বা hash value যেন একই হয়। এই যে সংখ্যায় পরিবর্তন করার যেই প্রসেস একে hashing বলে। এখন কেমনে কোন জিনিসকে একটি সংখ্যায় রূপান্তর করবে তার কোন নির্দিষ্ট উপায় নেই। তুমি যেভাবে খুশি পরিবর্তন করতে পারো। তবে এই পরিবর্তন এর উপর তুমি কি পেতে চাচ্ছ তা অনেক পরিমাণ নির্ভর করে। সাধারণত আমাদের প্রবলেম গুলি এরকম হয়ে থাকে, তোমাকে কিছু জিনিস দেয়া হতে থাকবে এবং তোমাকে মাঝে মাঝে জিজ্ঞাসা করা হবে অমুক জিনিস তোমার কাছে আছে কিনা। এজন্য তোমাকে কিছু list নিতে হবে, এদেরকে bucket বলা হয়। মনে করো তোমার কাছে n টি bucket আছে। এখন তুমি যা করবা তাহলো যখন তোমাকে সংরক্ষণের জন্য কোন জিনিস দিবে তুমি তাকে hash করে সেই hash value কে n দিয়ে mod করো এবং সেই bucket এ গিয়ে এই জিনিস রেখে দাও। এবার তোমাকে যখন কোন query দেয়া হবে তখন সেই query এর hash value বের করে সেই bucket এ যাও এবং সেই bucket এর জিনিসগুলি একে একে চেক করে দেখো যে তাদের কোনটি তোমার জিনিস কিনা। যদি তুমি n কে অনেক বড় নাও এবং তোমার hash function যদি ভাল হয় তাহলে খুব কম খুঁজেই তুমি তোমার query এর উত্তর দিতে পারবে। ধরা যাক আমি বললাম যে আমি তোমাকে কিছু 1000 ডিজিট এর সংখ্যা দিব আর মাঝে মাঝে জিজ্ঞাসা করব আমি তোমাকে অমুক সংখ্যা দিয়েছিলাম কিনা। তুমি মনে করলে আচ্ছা আমার hashing function হবে সংখ্যা গুলির যোগফল। তাহলে কিন্তু এটা খুব একটা ভাল hashing function হবে না। প্রথমত এক্ষেত্রে hashing function এর সর্বোচ্চ মান হবে $1000 \times 9 = 9000$ কিন্তু মনে করো তোমার bucket আছে 100000 টি। সুতরাং বুঝতেই পারছ যে এই hashing function এর ক্ষেত্রে অনেক bucket অব্যবহৃত থাকবে। তাহলে তোমার ব্যবহৃত bucket গুলিতে গড়ে বেশি বেশি সংখ্যা থাকবে আর এতে তোমার query সময় ও বাড়বে। তাহলে কি করা যায়? গুণফল? হ্যা গুণফল অনেক বড় সংখ্যা হবে, একে mod করলে আমরা 100000 টি bucket ই ব্যবহার করতে পারব। কিন্তু খেয়াল করো যদি আমাদের সংখ্যার কোন একটি ডিজিট 0 হয় তাহলে সেটি সবসময় 0 তম bucket এ পড়বে। এখন তোমাদের ইনপুট এ যেসব সংখ্যা দেয়া থাকবে মনে করো তাদের সবার 0 ডিজিট থাকবে। তাহলে তোমার query করতে অনেক বেশি সময় লাগবে তাই না? সুতরাং এই hashing function ও খুব একটা ভাল না।

তাহলে ভাল ফাংশন কেমন হয়? আগেই বলেছি hashing function তোমার ইচ্ছা মত করতে পারো তবে উপরের দুইটি জিনিস খেয়াল রাখতে হবে, এক যেন অনেক সংখ্যা একটা bucket এ

গিয়ে জড়ো নাহয় আর সব bucket যেন সমান ভাবে ব্যবহার হয়। এই দুইটি দিক খেয়াল করে একটি বহুল ব্যবহৃত hashing function হল polynomial hashing function. একটা polynomial দেখতে কেমন হয় তাতো জানো? $a_0 + a_1x^1 + a_2x^2 + \ldots$. মনে করো $a_0, a_1 \ldots$ এগুলি হল তোমাকে দেয়া সংখ্যার ডিজিট বা তোমাকে দেয়া জিনিসের ক্ষুদ্র ক্ষুদ্র অংশ। যেমন একটি string এর ক্ষেত্রে এর প্রতিটি character এর ascii value. আর x হিসাবে একটি prime সংখ্যা নেয়া ভাল। এখন তুমি এই hash এর মান বের করো। আর n টি bucket এ distribute হবার জন্য এই মানকে n দিয়ে mod করো। সাধারণত এই n কেও অন্য কোন prime নেয়া হয়ে থাকে। এটি একটি ভাল hash function বলা চলে। তাহলে তুমি hash value বের করার পর সেই bucket এ গিয়ে তোমার জিনিস store করবে এবং কোন জিনিস খুঁজতে বললে তুমি তার hash value এর bucket এ গিয়ে প্রতিটির সাথে তুলনা করে দেখবে তোমার সংখ্যা এখানে আছে কিনা। যদি অনেক গুলি bucket নাও তাহলে এই খোঁজার পরিমাণ অনেক অনেক কমে যাবে। এটাই হল hashing.

এখন অনেক সময় সংখ্যা না দিয়ে একটি সেট দিয়ে বলে যে এই সেটটি আগে এসেছিল কিনা। অর্থাৎ $\{1, 2\}$ আর $\{2, 1\}$ কে একই জিনিস হিসাবে বিবেচনা করতে হবে। এক্ষেত্রে যেটা করলে ভাল হয় তাহলো তুমি তোমার সেট এর element গুলিকে sort করে নাও। এর পর একে একে hash করো। বা তুমি সেট এর element গুলিকে আলাদা আলাদা ভাবে hash করে এর পর তাদের যোগ করো বা xor করো। কারণ এতে order কোন ব্যপার হয় না। এভাবে প্রবলেম ভেদে টুকটাক টেকনিক খাটিয়ে hash করলে ভাল ফল পাবা।

অনেক সময় দুইটি বড় string দিয়ে বলা হয় একটি আরেকটির ভিতরে substring আকারে আছে কিনা (subsequence না কিন্তু)। ধরা যাক আমাদের কে বলা হয়েছে S এর ভিতরে T খুঁজতে। স্বাভাবিক idea হল তুমি S এর প্রতিটি জায়গায় গিয়ে $|T|$ পরিমাণ substring নিয়ে তাকে hash করে T এর hash এর সাথে তুলনা করা। কিন্তু এতে $|S| \times |T|$ সময় লেগে যাবে। এর থেকে ভাল উপায় আছে। খেয়াল করো S এর 0 থেকে $|T|$ সাইজের substring এর polynomial hash কেমন? $H_0 = s_0 + s_1P^1 + \ldots s_{n-1}P^{n-1}$ তাই না? আবার 1 থেকে? $H_1 = s_1 + s_2P^1 + \ldots s_nP^{n-1}$. এই দুইটি সংখ্যার পার্থক্য কিন্তু খুব একটা বেশি না। আমরা কিন্তু লিখতে পারি $H_0 = H_1 \times P + s_0 - s_nP^n$. অর্থাৎ আমরা যদি H_1 জানি তাহলে খুব সহজে H_0 বের করে ফেলতে পারি। এর মানে আমরা পিছন থেকে hash function এর ভ্যালু বের করতে থাকলে খুব কম সময়েই সব জায়গার hash ভ্যালু বের করতে পারি। অথবা তুমি চাইলে polynomial কে উলটিয়ে দিতে পারো অর্থাৎ $a_0P^{n-1} + a_1P^{n-2} + \ldots$ তাহলে তুমি সামনে থেকেই যেতে পারবে। তাহলে এভাবে তোমার S এর সব জায়গার জন্য hash value বের করতে সময় লাগবে মাত্র $|S|$ সময়। আবার একটা জিনিস খেয়াল করো এখানে T কিন্তু fixed, তাই একে কিন্তু কোন একটা bucket এ ফেলা জরুরি না। তুমি চাইলে mod না করেই এই কাজ করতে পারো, মানে তুমি long বা long long এ যা হিসাব করার করবা। overflow হলে হবে, এসব নিয়ে মাথা ব্যাথা করতে হবে না। কারণ একই জিনিসকে যদি তুমি একই ভাবে hash করো তাহলে overflow হয়ে একই সংখ্যা হবে।

১১.২ Knuth Morris Pratt বা KMP অ্যালগোরিদম

আমরা কিছুক্ষণ আগে একটি string এর ভিতর আরেকটি string, substring আকারে আছে কিনা তা বের করলাম। তবে সমস্যা হল আমরা জানি না আগের মেথড এ কত সময় লাগবে। মানে আমরা expect করতে পারি যে এটা linear সময় নিবে তবে এটা যে সবসময় linear সময় নিবে তার কোন ঠিক নাই। হয়তো তোমার hashing method এর উপর ভিত্তি করে এমন একটি ইনপুট দেয়া সম্ভব যেখানে অনেক সময় নিবে। কিন্তু আমরা এই সমস্যা কে hashing ছাড়া linear সময়ে সমাধান করতে পারি। এ জন্য বহুল প্রচলিত অ্যালগোরিদম হল KMP বা Knuth Morris Pratt অ্যালগোরিদম। এটি একটু কঠিন অ্যালগোরিদম। অ্যালগোরিদমটা খুব ছোট কিন্তু এটা বুঝা বিশেষ করে এটি কেন linear সময়ে কাজ করে তা বুঝা একটু কষ্টকর।

মনে করো আমরা কোন একটি string T (Text) এর মাঝে একটি string P (Pattern) আছে কিনা তা বের করতে চাই। এজন্য আমাদের প্রথমে P এর prefix function বের করতে হবে।[১]

[১]আশা করি তোমাদের মনে আছে যে prefix কি বা suffix কি। Prefix হল কোন string এর শুরুর অংশ আর suffix হল শেষের অংশ। যেমন $xiox$ একটি string হলে এর prefix হবে $\{x, xi, xio, xiox\}$ আর suffix হবে

Prefix function কি? ধরা যাক আমরা prefix function কে π দিয়ে প্রকাশ করব। তাহলে $\pi(i)$ হবে $P[0\dots i]$ এর সবচেয়ে বড় proper prefix এর "দৈর্ঘ্য" (কেন quotation দিলাম তা একটু পরই পরিষ্কার হবে) যেন তা তার suffix ও হয়। যেমন যদি $P[0\dots i]$ হয় $aabcaaab$ তাহলে 3 length এর proper prefix পাওয়া যাবে যা suffix ও এবং এটি হল aab। এখানে proper prefix বলার কারণ হল আমি তো চাইলে পুরো string নিয়ে বলতে পারতাম যে এটা suffix ও prefix ও। কিন্তু এটা আমরা চাই না, এজন্য আমি বলেছি proper prefix. তাহলে আমরা একটা string এর সকল স্থানের জন্য prefix function এর মান বের করি।

সারনী ১১.১: একটি string এর সকল পজিশনে prefix function এর মান

index	0	1	2	3	4	5	6
P	a	b	a	b	a	c	a
π	-1	-1	0	1	2	-1	0

কিছুক্ষণ আগে বলা prefix function এর সংজ্ঞা এর সাথে দেখবে টেবিল ১১.১ এর π এর মানের মিল নেই। কিন্তু খুব বেশি যে অমিল তাও কিন্তু না। এটি সংজ্ঞা মোতাবেক মানের থেকে এক কম। আসলে আমরা এখানে string টিকে 0-index হিসাবে বিবেচনা করেছি। এবং $\pi(i)$ এর মান এমন হবে যেন $P[0\dots\pi(i)] = P[i-\pi(i)+1\dots i]$. অর্থাৎ π কে আমরা ঠিক দৈর্ঘ্য দিয়ে না বরং index দিয়ে সংজ্ঞায়িত করব। এই টেবিল এ $\pi(0) = \pi(1) = -1$ কেন তা একটু বলা দরকার। কারণ হল আমরা মনে করতে পারি যে $P[0\dots -1]$ হল একটি ফাঁকা string এবং যেহেতু a বা ab এর এমন কোন proper prefix নাই যা $suffix$ ও তাই আমরা ধরে নেই যে ফাঁকা string হবে এই prefix বা suffix. আসলে সত্য বলতে এটা বলার জন্য বলা, -1 না দিলে পরবর্তী অংশ কোড করতে ঝামেলা হবে বা যদি আমরা 0-index না মনে করে যদি 1-index নিতাম তাহলে পুরো জিনিস অনেক সহজ হতো এবং এখানে -1 না হয়ে 0 হতো। যাই হোক, তুমি যখন পুরো algorithm টা বুঝে যাবে তখন এসব কেন কি করছি তাও বুঝতে পারবে তাই এসব নিয়ে এখন ওত বেশি চিন্তা করার কিছু নেই। এখন তাহলে টেবিল ১১.১ এর π এর মানে একবার চোখ বুলিয়ে নাও। দেখো বাদ বাকি সব মান ঠিক আছে কিনা। এখন আমাদের প্রশ্ন হল এই π এর সব মান কেমনে আমরা linear সময়ে বের করতে পারি।

প্রথমত $\pi(0) = -1$. এখন তুমি সর্বশেষ π এর মানকে একটি variable ধরা যাক now এর ভিতরে নাও এবং 1 হতে $|P|-1$ পর্যন্ত i এর একটি লুপ চালাও। আমরা $\pi(i)$ বের করতে চাই। এজন্য আমাদের যা করতে হবে তাহলো $P[now+1]$ এবং $P[i]$ সমান কিনা তা দেখতে হবে। যদি না হয় তাহলে $now = \pi(now)$ করব। আর যদি সমান হয় বা $now = -1$ হয় তাহলে এসব না করে এই লুপ থেকে বের হতে হবে। তবে আমাদের আবারো $P[now+1]$ এবং $P[i]$ তুলনা করব এবং যদি সমান হয় তাহলে now কে এক বাড়াবো এবং $\pi(i)$ এ এই মান রাখব। আর যদি সমান নাহয় তাহলে $now = \pi(i) = -1$ করব। এতক্ষণ যা বললাম তা এক রকম অ্যালগোরিদম এর বর্ণনা। কিন্তু আমাদের বুঝতে হবে কেন আমরা এরকম করছি।

আমরা প্রথমে terminal case $now = -1$ নিয়ে চিন্তা না করে একটা general case নিয়ে চিন্তা করে দেখি। মনে করো কোন এক i এর জন্য $\pi(i)$ জানি, এখন আমরা বের করতে চাই $\pi(i+1)$ এর মান। $\pi(i)$ মানে কি? এর মানে হল $P[0\dots\pi(i)] = P[i-\pi(i)+1\dots i]$ এবং এরকম সবগুলির মাঝে $\pi(i)$ সর্বোচ্চ (অবশ্যই i এর থেকে ছোট)। এখন এরকম আমরা সবচেয়ে বড় $\pi(i+1)$ বের করতে চাই। খেয়াল করো যদি $P[\pi(i)+1] = P[i+1]$ হতো তাহলে আমরা খুব সহজেই বলতে পারতাম যে $\pi(i+1) = \pi(i)+1$ কারণ এর থেকে বড় কিন্তু হওয়া সম্ভব না, হলে $\pi(i)$ আরও বড় হওয়া সম্ভব হতো তাই না? বা অন্য ভাবে বলতে হলে বলতে হয় $\pi(i+1)-1$ কিন্তু $\pi(i)$ এর একটি candidate. সুতরাং আমরা যদি দেখি $P[\pi(i)+1] = P[i+1]$ তাহলে $\pi(i+1) = \pi(i)+1$. এখন প্রশ্ন হল যদি না হয়? আমাদের লক্ষ্য $P[0\dots i+1]$ এর একটি suffix বের করা যা prefix ও বা অন্য ভাবে বলতে হলে বলা যায় $P[0\dots i]$ এর একটি suffix বের করা যা prefix ও এবং সেই prefix এর পরের character $P[i+1]$ এর সমান। আমরা একটি prefix ইতোমধ্যেই চেষ্টা

$\{xiox, iox, ox, x\}$. Proper prefix হল ঐ string বাদে ঐ string এর অন্যান্য prefix. যেমন এই string এর জন্য proper prefix হল $\{x, xi, xio\}$.

করেছি আর তাহলো $P[0\ldots\pi(i)]$. এর পরের বড় candidate suffix কোনটি হবে? সেটি কিন্তু ইতোমধ্যেই বের করে রেখেছ আর তাহলো $\pi(\pi(i))$. কেন? খেয়াল করো আমরা চাই $\pi(i)$ এর থেকে ছোট suffix যা prefix ও। ধরা যাক এরকম কোন একটি suffix বা prefix হল Z. এই Z এর বৈশিষ্ট্য হল এটি একই সাথে $P[0\ldots\pi(i)]$ এর prefix এবং suffix. এবং আসলে এদের মাঝে সবচেয়ে বড়টি আমাদের দরকার। আর সেটিই কিন্তু $\pi(\pi(i))$ তাই না? উদাহরণ দেয়া যাক। মনে করো আমারা $ababa$ এর জন্য $\pi(4)$ জানি আর তাহলো 2. এখন মনে করো আমাদের পরের character হল c যা $P[3]$ এর সমান না। তাই আমাদের aba এর থেকে ছোট এমন একটি string দরকার যা $ababa$ এর একই সাথে suffix ও prefix. এবং সেটি কিন্তু আবার aba এরও prefix ও suffix. তাই আমরা যদি $\pi(2)$ দেখি তাহলে a পাব যা $ababa$ এর prefix ও suffix. তোমরা যদি এটুকু বুঝে থাকো তাহলে আশা করি কোড ১১.১ ও বুঝতে পারবে। একটা জিনিস বলা হয় নাই তাহলো আমরা এতক্ষণ general case নিয়ে চিন্তা করেছি। Terminal case নিয়ে বলা হয় নাই। খেয়াল করো কিছুক্ষণ আগের উদাহরণে যখন আমরা দেখব যে $P[1]$ ও c না তখন আমরা আবার $\pi(1)$ করব আর এক্ষেত্রে আমরা পাব -1. এখন প্রথম কথা হল এই লুপ আজীবন চলতে পারে না, এক সময় আমাদের শেষ করতে হবে আর এছাড়াও তোমরা $\pi(-1)$ কল করতে পারবা না কারণ এটা বলে কিছু নাই, তাই যখন $now = -1$ হয়ে গেছে তখন আমরা লুপ থেকে বের হয়ে গেছি। তবে এই লুপ থেকে বের হওয়ার দুইটি মানে আছে এক $now + 1$ এর সাথে মিলে গেছে দুই মিলে নাই মানে -1 হবে। এই চেক করার জন্যই আমাদের এই লুপ এর শেসে একটি if-else আছে।

কোড ১১.১: prefix function.cpp

```
int pi[100];
char P[100];

int prefixFunction() {
    int now;
    pi[0] = now = -1;
    int len = strlen(P);
    for (int i = 1; i < len; i++) {
        while (now != -1 && P[now + 1] != P[i]) now = ↩
            pi[now];
        if (P[now + 1] == P[i]) pi[i] = ++now;
        else pi[i] = now = -1;
    }
}
```

তবে এখনও আমাদের matching সমস্যার সমাধান হয় নাই। আমরা কেবল মাত্র আমাদের pattern অর্থাৎ P এর prefix function বের করলাম। এখন আমাদের কাজ হল P কি T এর ভিতর আছে কিনা তা বের করা। এজন্য আমরা কিছুটা আগের মতই কাজ করব। প্রথমে $now = -1$ নাও এবং T এর উপর দিয়ে 0 হতে $|T| - 1$ পর্যন্ত একটি লুপ চালাও। i এর লুপে যখন ঢুকবে তখন now নির্দেশ করবে $T[0\ldots i-1]$ এর longest suffix যা P এর prefix. এখন আমাদের বিবেচনা করতে হবে $T[i]$. আগের মত প্রথমে দেখো যে $P[now+1]$ কি $T[i]$ এর সমান কিনা। হলে তো now কে এক বাড়িয়ে দিবে। আর যদি না হয় তাহলে $now = \pi(now)$ করবে এবং আবারো একই চেক করবে। আর যদি $now = -1$ হয়ে যায় তাহলে কি করতে হবে তাতো বুঝতেই পারছ। কোড ১১.২ এ আমরা এটা কোড করে দেখালাম।

কোড ১১.২: kmp.cpp

```
int pi[100];
char P[100], T[100];

```

```
int kmp() {
    int now;
    now = -1;
    int n = strlen(T);
    int m = strlen(P);
    for (int i = 0; i < n; i++) {
        while (now != -1 && P[now + 1] != T[i]) now = pi[now];
        if (P[now + 1] == T[i]) ++now;
        else now = -1;
        if(now == m) return 1;
    }
    return 0;
}
```

এখন কথা হল এর time complexity কত? দুইটি লুপ দেখে যদি ভাব এটি $O(n^2)$ তাহলে ভুল ভাববে। খেয়াল করো for লুপ এর একটি iteration এ now এর মান খুব জোড় এক বারে। আবার while লুপে now এর মান কখনও বাড়ে না, শুধুই কমে। তাই while লুপ আসলে সর্বমোট linear সময় চলে। অর্থাৎ আমাদের prefix function বের করার কোড সময় নেয় $O(|P|)$ আর matching এর কোড সময় নেয় $O(|T|)$.

১১.২.১ KMP সম্পর্কিত কিছু সমস্যা

ধরা যাক P কি T এর মাঝে আছে কিনা শুধু এটাই জিজ্ঞাসা করে নাই বরং কত বার আছে তাও জানতে চেয়েছে। তুমি কি করবে? একটি সহজ বুদ্ধি হল $P\#T$ এর prefix function (একে failure function ও বলে) বের করা। এখানে # হল এমন একটি character যা P বা T কারো ভিতরে নেই। তাহলে prefix function এ যতবার $|P|$ আসবে সেটাই তোমার উত্তর। এছাড়াও আরেকটি উপায় হল উপরে আমরা যখন P কে T এর ভিতরে খুঁজেছিলাম তখন যে $now = |P|$ হলেই 1 return করেছিলাম তা না করে আমরা তখন একটি counter এর মান বাড়াবো এবং $now = \pi(Now)$ কল করব।

আরেকটি সমস্যা এরকম হতে পারে যে আমাদের শুধু P কতবার পাওয়া গেছে তাই জানতে চায় নাই বরং P এর সব prefix কতবার T তে আছে তা জানতে চাওয়া হয়েছে। কেমনে করবে? যা করতে হবে তাহলো প্রতিবার while লুপ শেসে একটি অ্যারে তে now index এর মান এক করে বাড়াতে হবে। অর্থাৎ আমরা now পর্যন্ত prefix পেয়েছি এটা বুঝাতে। কিন্তু শুধু এটা করলে কিন্তু হবে না। উদাহরণ সরূপ $P = aa$ মনে করো আর $T = aaaaa$. এখন এটা তো বুঝছই যে প্রায় সবসময় আমরা $now = 1$ এর মান বাড়াবো কিন্তু $now = 0$ এর মান কিন্তু তেমন বাড়বে না যদিও prefix a বহুবার T তে দেখা যায়। সুতরাং আমাদের যা করতে হবে তাহলো T এর উপরের লুপ শেষ হলে এবার P এর শেষ থেকে শুরুতে লুপ চালাতে হবে। এবং প্রতি i এর জন্য $count[\pi(i)] += count[i]$ করতে হবে। কেন? কারণ হল i এ শেষ হওয়া সবচেয়ে বড় suffix যা prefix ও তাতে কিন্তু তুমি আরও $count[i]$ বার যেতে পারতে যা আগে হিসাব করো নাই।

ধরো একটি string দিয়ে বলা হল এতে কয়টি distinct substring আছে। কেমনে করা যায়? মনে করো string টি হল S এবং এর prefix function আমরা জানি। এ থেকে আমরা সবচেয়ে বড় মান k বের করলাম। এর মানে কি? এর মানে হল এই string এর $k+1$ হতে $|S|$ দৈর্ঘ্যের যে prefix তা এই string এ আর কথাও আসে নাই। কিন্তু $S[1]$ থেকে যেসব unique substring আছে সেসব কেমনে বের করব? সহজ, $S[1\ldots]$ এর জন্য আবার prefix function বের করো। এভাবে S এর প্রতিটি suffix এর জন্য unique prefix এর সংখ্যা যোগ করলে আমরা মোট distinct substring এর সংখ্যা পেয়ে যাব। এজন্য আমাদের সময় লাগছে $O(n^2)$.

একটি string S দেয়া আছে বলতে হবে এমন কোন ছোট string P আছে কিনা যাকে বার বার repeat করলে S পাওয়া যায়। যেমন $S = ababab$ এখানে $P = ab$ কে তিনবার পর পর বসালে

S পাওয়া যায়। এর সমাধান হল দেখতে হবে যে $n - \pi(n-1)$ কি n কে ভাগ করে কিনা। করলে সেই দৈর্ঘ্যের prefix ই হবে উত্তর। এর প্রমাণ একটু কঠিন। তোমরা একটু চিন্তা ভাবনা করে proof by contradiction এ এই জিনিস প্রমাণ করতে পারো।

১১.৩ Z algorithm

KMP তে যেমন prefix function ছিল এখানে আছে z function. $z(i)$ এর মানে হল i তম index হতে শুরু করে কত বড় string পাওয়া যায় যা মূল string এর prefix. যেমন $ababc$ এই string এর জন্য z function এর মান গুলি হবে $\{0, 0, 2, 0, 0\}$. এখানে $z(0)$ এর মান 0 করা হয়েছে। কারণ কিছুটা kmp এর $\pi(0)$ এর মত। আর তাছাড়া এটি আমাদের কোড সহজ করবে।

এখন আসা যাক এটা কেমনে বের করা যায় সেই প্রশ্নে। অবশ্যই $O(n^2)$ সময়ে বের করা কোন ব্যাপার না। আমরা চাই linear সময়ে একটি string S এর z function বের করতে। আমাদের এজন্য দুইটি variable এর দরকার একটি হল $left$ এবং আরেকটি হল $right$. প্রথমে আমরা $z(0) = left = right = 0$ সেট করব। এখন আমরা $i = 1$ হতে $|S| - 1$ পর্যন্ত একটি লুপ চালাব। লুপ এর ভিতর কি করব তা জানার আগে তোমাদের $left$ আর $right$ এর মানে জানলে ভাল হয়। i এর লুপ এর i তম অবস্থানে এই দুইটি variable প্রকাশ করে যে $[0, i-1]$ এই রেঞ্জ এর কোন মান x জন্য $x + z(x)$ এর মান সর্বোচ্চ হয়। অর্থাৎ $S[left \dots right]$ S এর একটি prefix হবে এবং $0 < left < i$ হবে আর এরকম সব candidate এর মাঝে $right$ সর্বোচ্চ হয়। অর্থাৎ প্রতিবার লুপ এর ভিতরে $z(i)$ এর মান বের হয়ে গেলে আমাদের দেখতে হবে যে $i + z(i) - 1$ কি $right$ এর থেকে বেশি কিনা। বেশি হলে $left = i, right = i + z(i) - 1$ করতে হবে।

এখন কথা হল এই $z(i)$ এর মান কেমনে বের করা যায়। প্রথমে আমরা দেখব যে $i \le right$ কিনা। হলে আমরা $z(i)$ এর মানকে এক লাফে অনেক দূর বাড়াতে পারব। কি রকম? খেয়াল করো $S[left \dots right]$ হল S এর একটি prefix বা $S[0 \dots (right - left)]$ এর সমান। এটা আশা করি বুঝতে পেরেছ যে $left < i$ এবং $i \le right$? তাহলে এটা বলা যায় যে $S[i \dots right]$ হল $S[(i - left) \dots (right - left)]$ এর সমান। এবং যেহেতু $i - left < i$ তাই আমরা কিন্তু $z(i - left)$ এর মান আগে থেকেই জানি। এবং আমরা বলতে পারি যে $z(i)$ এর মান কিছুটা $z(i - left)$ এর মত হবে। কারণ ঐ যে বললাম $S[i \dots right]$ হল $S[(i - left) \dots (right - left)]$ এর সমান। এখন খেয়াল করো আমরা সরাসরি $z(i) = z(i - left)$ করতে পারব না। কেন? কারণ আর যাই হোক $i + z(i) - 1 > right$ হতে পারবে না কারণ আমরা $S[right]$ এর পরের information জানি না। তাই যা করতে হবে তাহলো $z(i) = min(z(i - left), right - i + 1)$ করতে হবে। এটি $z(i)$ এর একটি lower limit. অর্থাৎ $z(i)$ কমপক্ষে এতো হবেই। এর থেকে বড় হবে কিনা তা sure না। তাই যা করতে হবে তাহলো একটি লুপ চালয়ে আরও বড় করা যায় কিনা তা দেখতে হবে। এভাবে $z(i)$ এর মান বের করতে হবে। এর কোড ১১.৩ এ দেয়া হল।

কোড ১১.৩: zfunction.cpp

```
char S[100];
int z[100];

void zfunction() {
    int left, right;
    z[0] = left = right = 0;
    int len = strlen(S);
    for (int i = 1; i < len; i++) {
        if (i <= right) z[i] = min(z[i - left], z[right - i + 1]);
        while (i + z[i] < len && S[i + z[i]] == S[z[i]])
```

```
            z[i]++;
        if (i + z[i] - 1 > right)
            left = i, right = i + z[i] - 1;
    }
}
```

উদাহরণ দেখা যাক, মনে করো $ababab$ এর z function বের করছি, আমরা $z(2) = 4$ বের করে ফেলার পর $z(4)$ এর মান কিন্তু সেই $z(2)$ এর information থেকে 2 পেয়ে যাব। কেমনে? দেখো 4 হল $left = 2$ আর $right = 2 + 4 - 1 = 5$ এর ভিতরে। সুতরাং আমরা $z(4) = min(z(2) = 4, 2) = 2$ করব। এর পর আর পরের while লুপ চলবে না কারণ $4 + z(4) < len$ না। এর মানে $z(4)$ বের করতে আমাদের কোন কাজই করতে হচ্ছে না।

এখন কথা হল এটা কেন linear? খেয়াল করো for লুপ এর ভিতরে যেই if আছে সেখানে যদি $z(i) = right - left + 1$ দিয়ে bound হয় অর্থাৎ i হতে শুরু করা string টা $right$ এ গিয়ে আটকে যায়, তাহলে আমরা while লুপ দিয়ে $right$ এর মান বাড়ানোর চেষ্টা করি। আর যদি $right$ এর আগেই bound হয়ে যায় তাহলে কিন্তু এই while লুপ এর equality condition সত্য হবে না। তাই কোন কাজ না করেই এই লুপ শেষ হয়ে যাবে। অর্থাৎ এই while লুপ কিন্তু সবসময় $right$ এর মান বাড়াবে। আর কতই বা বাড়াবে? $len = |S|$ এর থেকে তো আর বড় না তাই না? তাই এটি linear.

১১.৩.১ Z algorithm সম্পর্কিত কিছু সমস্যা

এখন এর মাধ্যমে কি কি সমস্যা সমাধান করা যায়? KMP দিয়ে সমাধান করা যায় এরকম বেশির ভাগ সমস্যাই সমাধান করা সম্ভব। যেমন P কি T এর ভিতর আছে কিনা? এটা সমাধানের জন্য $S = P + \# + T$ করে দেখো যে কত গুলি $z(i)$ এর মান $|P|$ তাহলেই হয়ে গেল। একই ভাবে একটি string এ কত গুলি distinct substring আছে তাও বের করা সহজ। একটি string নিয়ে তার z function এর মান বের করো। ধরা যাক সর্বোচ্চ মান x. এর মানে $S[1\ldots]$ এ আমরা কখনও এক সময় $S[0\ldots x-1]$ এই substring পাব। সুতরাং এই substring নিয়ে এখন না চিন্তা করে $S[0\ldots(x\ldots|S|-1)]$ এই substring গুলি নিয়ে চিন্তা করব। অর্থাৎ এখন আমাদের distinct substring এর count $(|S|-1)-x+1$ বাড়াবো। আর এর পরে $S[1\ldots]$ নিয়ে চিন্তা করব। এভাবে $|S|$ বার S এর $|S|$ টি suffix নিয়ে কাজ করব।

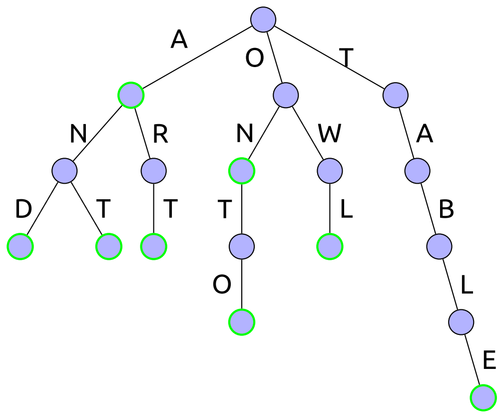
১১.৪ Trie

এটি কথা বলে বুঝানোর থেকে ছবিতে বুঝানো সহজ। চিত্র ১১.১ এ {a, and, ant, art, on, onto, owl, table} শব্দ সমূহের জন্য trie এঁকে দেখানো হল।

চিত্র থেকে খুব সহজেই বুঝা যাবার কথা trie আসলে কি। এটি tree আকারে তোমাকে দেয়া সব word কে represent করে। Common prefix এর জন্য একটিই মাত্র নোড থাকে। যেমন উপরের উদাহরণ এ ant আর art দুইটার জন্যই a নোড দুইটি একই। আমরা বেশি কথা না বলে সরাসরি কোড এ চলে যাব এবং আসলে এই কোড ব্যাখ্যা করারও কিছু নেই। আশা করি তোমরা কোড ১১.৪ দেখে বুঝতে পারবে আমরা কি করছি এবং কেন করছি। আর এও আশা করছি যে তোমাদেরকে যদি বলা হয় আচ্ছা অমুক শব্দ এই trie এ আছে কিনা তাও বের করতে পারবে? Root থেকে traverse শুরু করবে আর দেখবে বর্তমান নোডে বর্তমান character দিয়ে লেবেল করা edge আছে কিনা না থাকলে সেই শব্দ নাই। আর থাকলে এভাবে আগাতে থাকতে হবে। শেষে গিয়ে দেখতে হবে শেষের নোডে isWord এ মার্ক করা আছে কিনা।

কোড ১১.৪: trie.cpp

```
#define MAX_NODE 100000
```

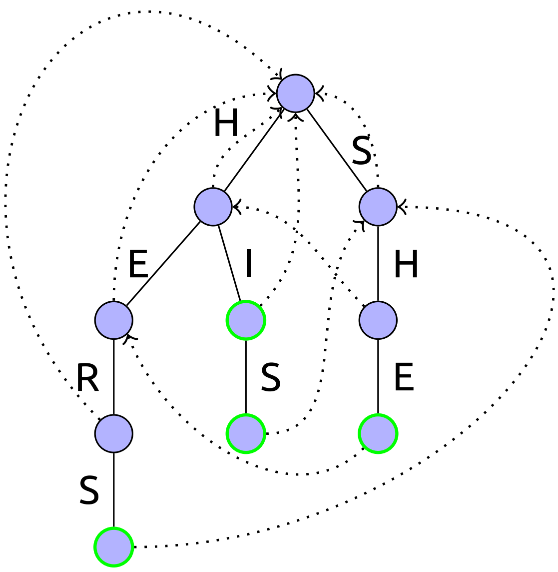


চিত্র ১১.১: {a, and, ant, art, on, onto, owl, table} শব্দ সমূহের জন্য trie

```
#define MAX_LEN 100

char S[MAX_LEN];
int node[MAX_NODE][26]; // assuming words will be consisted of only small letters ['a', 'z']
int root, nnode;
int isWord[MAX_NODE];

// Should be called before inserting any words into trie.
void initialize() {
    root = 0;
    nnode = 0;
    for (int i = 0; i < 26; i++)
        node[root][i] = -1; // -1 means no edge for ('a' + i)th character
}

void insert() {
    scanf("%s", S);
    int len = strlen(S);
    int now = root;
    for (int i = 0; i < len; i++) {
        if (node[now][S[i] - 'a'] == -1) {
            node[now][S[i] - 'a'] = ++nnode;
            for (int j = 0; j < 26; j++)
                node[nnode][j] = -1;
        }
        now = node[now][S[i] - 'a'];
    }
    isWord[now] = 1; // mark that a word ended at this node.
}
```

১১.৫ Aho corasick অ্যালগোরিদম

আমরা KMP বা Z function দিয়ে একটি pattern এবং একটি text এর জন্য linear সময়ে সমাধান করেছিলাম। যদি আমাদের একটি pattern আর অনেকগুলি text থাকত তাহলে বার বার KMP বা Z function ব্যবহার করে আমরা সমাধান করতে পারতাম। খেয়াল করো যেহেতু pattern নির্দিষ্টই থাকছে সেহেতু failure function (বা prefix function) কিন্তু বার বার বের করার দরকার নেই। একই ভাবে z function এর ক্ষেত্রেও একই কথা প্রযোজ্য। তবে যদি pattern একাধিক হয় আর text একটি (আসলে একাধিক text এর জন্যও কাজ করবে) তাহলে linear সময়ে সমাধান করতে আমাদের aho corasick অ্যালগোরিদম লাগবে। মনে করো আমাদের pattern এর string গুলি হল $hers, hi, his, she.$ তাহলে আমাদের চিত্র ১১.২ এর মত একটি ডাটা স্ট্রাকচার বানাতে হবে। এই ছবির ডটেড লাইনগুলি বাদ দিলে এটা আসলে একটা trie মাত্র। এখন বুঝতে হবে ডটেড লাইন এর মাহাত্ম কি! এগুলি হল প্রতিটি নোড এর failure function. অর্থাৎ root হতে কোন নোড পর্যন্ত যেই string টি হয় তার সর্বোচ্চ দৈর্ঘ্যের কোন suffix টি (অবশ্যই proper suffix) trie এ আছে? failure function টি সেই নোডকে point করবে। একটা উদাহরণ দেখা যাক। মনে করো $she.$ এর proper suffix গুলি হল he এবং $e.$ এদের মাঝে e শব্দটি trie এ নাই। মানে তুমি যদি root থেকে traverse শুরু করো তাহলে এই শব্দ পাবে না। এখানে শব্দ পাবে না মানে traverse করে খুঁজার সময়ে শেষে গিয়ে isWord দেখার প্রয়োজন নেই। যাই হোক, তাহলে e বলে কিছু নেই এই trie এ। এখন he দেখা যাক, এবং হ্যা he আছে। এবং এটিই আসলে সর্বোচ্চ length এর proper suffix যার জন্য trie এ নোড আছে। সুতরাং she নোড এর জন্য failure function he কে point করবে। এভাবে প্রতিটি নোড এর জন্য failure function বের করতে হবে। আর হ্যা, যদি কোন suffix না পাও, এর মানে মনে করবা empty string ও একটি suffix, আর তাই failure function কে empty string বা root এ point করাবে। যেমন h এর নোড থেকে আমরা root এ point করেছি।



চিত্র ১১.২: {hers, hi, his, she} শব্দ সমূহের জন্য aho corasick trie

এখন একটি text দিয়ে যদি বলে কোন কোন pattern এই text এর মাঝে আছে বা কতগুলি

pattern আছে তাহলে কেমনে সমাধান করবে? খুব সহজ, একদম kmp এর মত। Root হতে শুরু করো। text এর character দেখো আর সেই অনুসারে সামনে আগাতে থাকো। যদি দেখো তুমি এখন যেই নোড এ আছো সেখানে সেই character এর কোন edge নেই তাহলে failure function এর edge দিয়ে পিছিয়ে গিয়ে সেখানে দেখবে। সেখানে না থাকলে আবার failure function এর edge দিয়ে পিছিয়ে গিয়ে দেখবে। এভাবে করতে থাকবে যতক্ষণ না root এ যাও। যদি root এ আসো তাহলে কি করতে হবে তাতো আমরা kmp তেই দেখেছি তাই না? এটা আসলে kmp এরই একটা extended রূপ। এটা বুঝতে হলে তোমাদের kmp ভালমতো বুঝতে হবে। যাই হোক, এখন কথা হল কেমনে বুঝবে যে কোন কোন pattern এই text এ আছে? তোমরা ভাবতে পারো যে এতো এতো important জিনিস না বলে skip করলাম আর বললাম kmp থেকে দেখে নিও আর এই সহজ জিনিস নিয়ে আমি গুঁতাগুঁতি করছি! "এটা তো অনেক সহজ, text দিয়ে trie এ traverse করার সময়ে যেসব word নোড দিয়ে traverse করব সেসব pattern এই text এ আছে!" না এটা ঠিক না। মনে করো আমাদের উদাহরণে s ও একটি pattern. আর তোমাকে দেয়া text হল his. তাহলে traverse এর সময় কেবল মাত্র h, hi আর his এই তিনটি নোড traverse হবে। অর্থাৎ তুমি মাত্র his pattern টা পেয়েছ। কিন্তু s শব্দের নোড কিন্তু traverse করো নাই। তাহলে কি করতে হবে? যা করতে হবে তাহলো, পুরো text টি traverse হয়ে গেলে তোমাকে দেখতে হবে কোন কোন pattern এর নোড visited হয়েছে। তাদেরকে একটি bfs এর জন্য queue তে রাখতে হবে। এবার একে একে নোড এই queue থেকে তুল আর এর failure function এর নোডকে queue তে ঢুকাও যদি না সেই নোডকে আগেই queue তে ঢুকানো হয়, অর্থাৎ যদি ঐ নোড আগেই visited না হয় আরকি। আমরা আসলে একরকমের bfs করছি যেখানে edge হল failure function এর edge. আর source নোড হল যেসব pattern কে আমরা text এর traverse এর সময় visit করেছি সেসব। এই bfs শেষে দেখতে হবে কোন কোন pattern এর নোড আমরা visit করছি। সেসবই হল আমাদের উত্তর। কেন? কারণটা বেশ সহজ আসলে, আমরা kmp তেও এরকম একটা জিনিস দেখে এসেছিলাম যখন আমরা কোন prefix কত বার আছে- এরকম একটা সমস্যার কথা বলেছিলাম। সুতরাং এটুকু তোমরা নিজেরাই ভেবে বের করতে পারবে। এর থেকে চল আমরা দেখি কেমনে trie এ linear সময়ে failure function বের করা যায়।

এই অংশটা একটু tricky কিন্তু সেই kmp এর মতই। কোন একটি নোড এর failure function বের করতে হলে তোমার parent নোডের failure function এ যেতে হবে এবং সেখান থেকে তোমার নোড এ আসার character দিয়ে সামনে আগানোর চেষ্টা করতে হবে নাহলে আবার failure function. অর্থাৎ কিছুক্ষণ আগে আমরা যেভাবে traverse করেছি ঠিক সেই রকম। এতো সহজই যদি হবে তাহলে আর বর্ণনা করতাম না। আমি যখন এই অ্যালগোরিদমটা শিখি তখন এই জায়গায় একটা ভুল করেছিলাম। তাহলো আমি ভেবেছিলাম এই জিনিস dfs দিয়ে কোড করা তো অনেক সহজ তাই dfs দিয়ে করি। কিন্তু তা হবে না। আগের text এর traversal টা dfs দিয়ে হবে কিন্তু failure function বের করার কাজটা dfs দিয়ে হবে না। বরং তোমাকে bfs করতে হবে। কেন? কারণ dfs দিয়ে যখন নামবে তখন মাত্র একটি branch ধরে নামতে থাকবে। এই branch এর বিভিন্ন নোড এর failure function বের করার সময় তোমার অন্য নোড এরও failure function জানা দরকার তাই না? যেমন ধরো আমাদের উদাহরণের চিত্র ১১.২ এ ধরো $shed$ নামে একটি শব্দও আছে, এখন তুমি যখন d তে আসবে তখন তো তুমি she এর failure function ব্যবহার করবে তাই না? অর্থাৎ he তে থাকবে। যেহেতু he নোড থেকেও d নামে কোন edge বের হয় নাই তাই তুমি এর failure function ব্যবহার করতে চাইবে। কিন্তু এর failure function তো এখনও বের করো নাই। তাই dfs দিয়ে failure function বের করা যাবে না। পরবর্তীতে নাসা ভাইয়া আমাকে এই রকম একটি case দিলে আমি বুঝতে পারি যে আমাদের আসলে bfs করতে হবে। BFS করলে হবে কারণ failure function সবসময় উপরের নোড অর্থাৎ কম depth এর নোড এ point করে, আর bfs নিশ্চিত করে যে বেশি depth এর নোড এর failure function বের করার আগে কম depth এর নোডগুলির failure function বের হয়ে যাবে। এভাবে আমরা সম্পূর্ণ প্রসেস linear সময়ে করতে পারি।

১১.৬ Suffix Array

একটি string S দেয়া থাকবে, তোমাকে এর suffix array A বের করতে হবে যেন $S[i \ldots |S|-1]$ এই suffix টি সকল suffix কে sort করলে তাদের মাঝে $A[i]$ তম suffix হয়। খেয়াল করো এখানে sort বলতে dictionary order বা lexicographical order বুঝানো হচ্ছে আর যেহেতু সকল suffix আলাদা তাই sort করতে কষ্ট হবার কথা না। একটা উদাহরণ দেখা যাক। ধরা যাক $S = xyxxyzz$. তাহলে এর suffix গুলি হচ্ছে $\{z, zz, yzz, xyzz, xxyzz, yxxyzz, xyxxyzz\}$ এবং এদের sort করলে দাঁড়ায় $\{xxyzz, xyxxyzz, xyzz, yxxyzz, yzz, z, zz\}$ বা $S[2], S[0], S[3], S[4], S[6], S[5]\}$. আশা করি এটা বুঝতে পারছ যে সংক্ষেপে প্রকাশের জন্য আমি $S[i]$ বলতে $S[ildots]$ এই suffix কে বুঝিয়েছি। তাহলে আমাদের suffix array হবে $\{1, 3, 0, 2, 4, 6, 5\}$. কেন? $A[0] = 1$ কারণ $S[0]$ আছে 1 এ, বা $A[5] = 6$ কারণ $S[5]$ আছে 5 এ।

এখন কথা হল এই suffix array আমরা কেমনে বানাব? একটি খুব সহজ $\mathrm{O}(n \log^2 n)$ সমাধান আছে। প্রথমে আমরা প্রতিটি index হতে 2^0 length এর substring এর জন্য suffix array এর মত জিনিস বের করব, এরপর 2^1 এবং এভাবে আমরা সব শেষে n দৈর্ঘ্য বা n এর থেকে immediate বড় 2 এর power এর suffix array বের করব। প্রথমে 2^0 অর্থাৎ প্রতিটি index এর character দেখে তাদের sorting এর একটা ordering দিতে হবে। যেমন আমাদের আগের উদাহরণ $xyxxyzz$ টি হবে $\{0, 1, 0, 0, 1, 2, 2\}$. অর্থাৎ x যেহেতু সবচেয়ে ছোট তাই এটি 0, y এর পরে তাই এটি 1 একই কারণে z 2 হয়। এখন আমরা বের করব 2^1 অর্থাৎ দুই দৈর্ঘ্যের জন্য। অর্থাৎ $\{xy, yx, xx, xy, yz, zz, z\}$. এখন এই জিনিস সরাসরি না বের করে আমরা একটি বুদ্ধি খাটাব। মনে করো আমরা 2^j length এর জন্য ordering বের করব। এখন i তম index এর কাহিনী দেখা যাক। আমরা এই 2^j length কে দুইটি 2^{j-1} ভাগে ভাগ করতে পারি। একটি হল $[i, i+2^{j-1}-1]$ আর আরেকটি হল $[i+2^{j-1}, i+2^j-1]$. এখন এই দুইটি অংশের 2^{j-1} length এর ordering কিন্তু আমরা জানি। তাহলে আমরা চাইলে একটি 2^j length কে string দিয়ে প্রকাশ না করে দুইটি নাম্বার দিয়ে প্রকাশ করতে পারি (a, b) যেখানে a হল $[i, i+2^{j-1}-1]$ এর ordering আর b হল $[i+2^{j-1}, i+2^j-1]$ এর ordering. এভাবে আমরা প্রতিটি index এর জন্য দুইটি নাম্বার পাব। তোমরা হয়তো ভাবতে পারো যদি $i + 2^{j-1} \geq |S|$ হয় তাহলে ঐ সংখ্যা কই পাব? সেক্ষেত্রে তুমি -1 ধরে নিতে পারো, কারণ কোন জায়গায় কোন character থাকার থেকে আমরা না থাকাকে বেশি priority দেই। তাহলে আমরা সকল জায়গার জন্য আমরা জোড়া জোড়া নাম্বার পেয়েছি। এখন এদের সর্ট করো এবং এদের ছোট থেকে বড় ক্রমে এদের index কে নাম্বার দিবে। সমান সংখ্যা জোড়া কে অবশ্যই একই নাম্বার দিবা। যেমন আমাদের উদাহরণে 2^1 এর ক্ষেত্রে আমাদের substring গুলি হল $\{xy, yx, xx, xy, yz, zz, z\}$ বা $\{(0,1), (1,0), (0,0), (0,1), (1,2), (2,2), (2,-1)\}$. এখন তোমাকে এদের সর্ট করতে হবে এবং সবচেয়ে ছোট জোড়া এবং তার সমান জোড়াকে তুমি নাম্বার দিবে 0, এর থেকে বড় কে 1 এরকম। এই নাম্বার কিন্তু তাদের index কে দেবে। অর্থাৎ ধরো আমাদের উপরের অ্যারে কে সর্ট করলে সবার আগে $(0,0)$ আসবে। এর মানে 2 index পজিশন 0 পাবে। এর পর আসবে $(0,1)$ যা 0 ও 3 পজিশনে আছে। তাই এই দুই পজিশনে 1 বসবে এভাবে তুমি 2^1 এর জন্য ordering পাবে। তাহলে এই ordering অ্যারে দেখতে এরকম হবেঃ $\{1, 2, 0, 1, 3, 5, 4\}$. এরকম করে তোমরা আশা করি 2^2 আর 2^3 এর জন্য ordering পেয়ে যাবে। যেহেতু $2^3 \geq |S|$ সুতরাং 2^3 এর জন্য পাওয়া ordering ই হল suffix array. যেহেতু আমরা $\log n$ বার সর্টিং করছি তাই আমাদের complexity হল $\mathrm{O}(n \log^2 n)$.

এখন যেহেতু কখনও আমাদের ordering এর নাম্বার n কে অতিক্রম করে না, তাই আমরা চাইলে এখানে counting sort করতে পারি। নাম্বার জোড়ার ক্ষেত্রে counting sort একটু ট্রিকি এবং কেমনে করে আমরা সেটা দেখবও না। যাদের ইচ্ছা আছে তোমরা নিজেরা ভেবে বের করতে পারো। খুব একটা কঠিন হওয়া উচিত না। সাধারণত আমি নিজেও এটা হাতে হাতে কোড করি না। হাতে হাতে কোড করা লাগলে আগের মত করি। আর আমার library তে এই counting sort দিয়ে $\mathrm{O}(n \log n)$ এর কোড তুলা আছে। তাই দরকারের সময় সেটা ব্যবহার করি আমি।

তোমাদের যদি ইচ্ছা থাকে তাহলে suffix array বের করার একটি linear algorithm আছে। সেটিও শিখে ফেলতে পারো। যদিও আমি নিজে কখনও ব্যবহার করি নাই, তাই ঠিক বলতে পারছি না সেখানে কেমনে করেছে। তবে এটুকু জানি ওখানে divide and conquer টেকনিক ব্যবহার করে

করা হয়েছে, কিছুটা linear সময়ে selection অ্যালগোরিদম যেভাবে করা হয় সেরকম।

১১.৬.১ Suffix array সম্পর্কিত কিছু সমস্যা

Suffix array এর উপর একটি সুন্দর document আছে যেটি যতদূর সম্ভব দুইজন Romanian তৈরি করেছে। আমি সেখান এর সমস্যা গুলিই একে একে তুলে ধরার চেষ্টা করব। তোমরা যদি পারো তাহলে এই document টা একটু পড়ে দেখো। হয়তো কিছু নতুন জিনিস শিখবে।

আমাদের আলোচনা এর সুবিধার জন্য আমরা ধরে নেই $\lceil \log n \rceil = k$. আমাদের string টি হল S আর তার suffix array থাকবে A তে। অর্থাৎ $A[i]$ বলে $S[i\ldots]$ কত তম suffix. এছাড়াও আমরা আরও একটি অ্যারে বিবেচনা করব $rank$. $rank[A[i]] = i$ হবে অর্থাৎ $S[rank[i]\ldots]$ হল sorted suffix list এ i তম suffix.

আমাদের প্রথম সমস্যা হল একটি string S দেয়া আছে। তোমাকে দুইটি index i এবং j দিয়ে বলা হল এই দুইটি অবস্থান থেকে শুরু করে যেই দুইটি string পাওয়া যায় তাদের longest common prefix (lcp) কত? অর্থাৎ যেমন $xxyyxxyz$ এই উদাহরণে যদি 0 আর 4 এই দুইটি অবস্থান দিয়ে জিজ্ঞাসা করত তাহলে আমাদের উত্তর হবে 3. বুঝতেই পারছ আমরা query খুব দ্রুত করতে চাচ্ছি। প্রথমে আমাদের suffix array বের করতে হবে। আমরা এখানে দুইটি মেথড এর কথা বলব। প্রথম মেথড এর জন্য আমাদের 0 হতে k সকল step এর জন্য ordering এর অ্যারে লাগবে। এই মেথডটি হল কিছুটা ট্রি তে দুইটি নোডের LCA বের করার মত। আমরা এই দুইটি অবস্থানের $k-1$ তম অ্যারে তে ordering দেখব। যদি সমান হয় তাহলে আমরা আমাদের উত্তর এর সাথে 2^{k-1} যোগ করে নিবো আর index দুইটিকে আমরা এই পরিমাণ বাড়িয়ে নিবো। এর পর আমরা $k-2$ নিয়ে চেক করব এবং একই কাজ করব। এভাবে আমরা $k-1$ হতে 0 পর্যন্ত কাজ করলেই $O(\log n)$ সময়ে আমরা lcp বের করে ফেলতে পারব। আরেকটি মেথড হল $Z[0] = lcp(rank[0], rank[1])$, $Z[1] = (rank[1], rank[2])$ এরকম sorted suffix গুলির পাশাপাশি string গুলির lcp বের করা। এখন তোমাদের i আর j নিয়ে যদি query করে তাহলে, $Z[min(A[0], A[1])\ldots max(A[0], A[1]) - 1]$ এই রেঞ্জ এর minimum বের করলেই তোমরা $S[i\ldots]$ আর $S[j\ldots]$ এর lcp পেয়ে যাবে। এখন পাশাপাশি এরকম pair থাকবে $|S|-1$ টি। তাই আমরা $O(n\log n)$ সময়ে আমরা এই পাশাপাশি সব সংখ্যার lcp বের করে রাখতে পারি। আর পরে দুইটি পজিশনের মাঝের minimum এর query তো আমরা $O(\log n)$ সময়ে বা $O(1)$ সময়ে করতেই পারি। তবে মজার ব্যাপার হল এই যে suffix array তে পাশাপাশি suffix গুলির lcp, এটা আসলে linear সময়ে বের করা সম্ভব। এর idea এর সাথে Z function এর idea এর বেশ মিল আছে। এই সমাধানের basic observation হল মনে করো আমরা $lcp(i, j) = 10$ পেয়েছি অর্থাৎ $S[i\ldots]$ আর $S[j\ldots]$ এর lcp হল 10. তাহলে $S[i+1\ldots]$ আর $S[j+1\ldots]$ অবশ্যই 9 হবে তাই না? কিন্তু কাহিনী হল $A[i+1]+1 \neq A[j+1]$ হতেই পারে, অর্থাৎ suffix array তে $S[i\ldots]$ আর $S[j\ldots]$ পাশাপাশি দুইটি suffix মানে $S[i+1\ldots]$ আর $S[j+1\ldots]$ ও যে পাশাপাশি দুইটি suffix হবে তা না। তবে এটা বলাই যায় যে তাদের মাঝে অন্তত 9 টি matching থাকবে। বা mathematical টার্ম এ বললে বলা যায় $lcp(i+1, rank[A[i+1]+1]) \geq lcp(i, rank[A[i]]+1) - 1$. সুতরাং $O(n)$ এ পাশাপাশি সকল lcp বের করার কোড হবে কোড ১১.৫ এর মত।

কোড ১১.৫: saLcp.cpp

```
char S[100];
int lcp[100], A[100], rank[100];

void LCP()
{
    int n=strlen(S);
    int now = 0;

```

```
    for(int i = 0; i < n; i++) rank[A[i]]=i;

    for(int i = 0; i < n; i++)
    {
        now = MIN(now - 1, 0); // lcp will never be -1.
        if(rank[i] == n - 1) {
            now = 0;
            continue;
        }
        // A[i] is the position of S[i ...] in suffix array.
        // A[i] + 1, is the next one for S[i ...].
        // rank[A[i] + 1] is the index of the next suffix in suffix array.
        int j = rank[A[i] + 1];
        while(i + now < n && j + now < n && s[i + now] == s[j + now])
            now++;
        lcp[A[i]] = now;
    }
}
```

একটি string দেয়া আছে। এর minimum lexicographic rotation বের করতে হবে। ধরা যাক একটি string হল $abac$ তাহলে এর rotation গুলি হল $abac$, $baca$, $acab$ আর $caba$. এদের মাঝে lexicographically ছোট string বের করতে হবে। Suffix array দিয়ে এটা সমাধান করা খুবই সহজ। মনে করো string টি S তাহলে এই string টি পরপর দুইবার লিখ SS. এখন দেখো প্রথম $|S|$ ঘরের মাঝে কোন পজিশনের জন্য $A[]$ এর মান সবচেয়ে ছোট। শেষ!

মনে করো তোমাদের একটি string S দেয়া আছে যার length n. এখন S কে S এর সাথে জোড়া লাগিয়ে $2n$ length এর একটি নতুন string T পেলাম। T এর প্রতিটি index i এর জন্য i এ শেষ হয় এবং length n এর বেশি না এরকম lexicographically সবচেয়ে ছোট string ধরা যাক $x(i)$. তোমাকে সেই index টি বের করতে হবে যার জন্য $x(i)$ সবচেয়ে বড়। প্রবলেম এর বর্ণনাটা অনেক বড় হলেও এর সমাধান খুব ছোট। একটু চিন্তা করে দেখো তো এই সমস্যা আর উপরের সমস্যা একই কিনা? অর্থাৎ এর সমাধান minimum lexicographic rotation.

এবারের সমস্যায় তোমাদের একটি string S এর জন্য সবচেয়ে বড় string T এর length বের করতে হবে যেন S এর ভিতরে T অন্তত পক্ষে k বার থাকে। যেমন $ababa$ এর ভিতরে aba মোট 2 বার আছে। এর সমাধানও বেশ সহজ। কোন একটি string k বার থাকা মানে suffix array তে পরপর k টি suffix তুমি পাবেই যাদের prefix হবে সেই k বার থাকা string. তাহলে তোমাকে যা করতে হবে তাহলো suffix array এর প্রত্যেক পরপর k টা suffix এর lcp বের করতে হবে, বা আসলে $[i, i+k-1]$ এই রেঞ্জ এর suffix গুলির lcp বের করার জন্য আসলে i আর $i+k-1$ তম suffix এর lcp বের করলেই চলে। এভাবে প্রতিটি i এর জন্য lcp বের করে তাদের maximum টাই উত্তর।

একটু চিন্তা করে দেখো তো suffix array এর সাহায্যে কোন string এর distinct substring এর count বের করতে পারো কিনা। খুব একটা কঠিন না। মনে করো তুমি suffix array এর ছোট থেকে বড় তে যাচ্ছ। i তম তে এসে দেখবে এর উপরের সাথে তোমার কত lcp. ঠিক তত মনে করবে আগেই নিয়ে ফেলেছি, বাকি length টুকু তোমার উত্তর এর সাথে যোগ করবে। শেষ!

ধরো তোমাদের একটি বড় string S দেয়া আছে আর অনেক গুলি (M টি) ছোট ছোট string দেয়া আছে যাদের দৈর্ঘ্য ধরা যাক 64 এর বেশি হবে না। এখন তোমাকে প্রতিটি ছোট string এর জন্য বলতে হবে সেটি S এর ভিতরে কয়বার আছে। যেহেতু ছোট string গুলি আসলে $64 = 2^6$ এর বেশি length না তাই মাত্র 7 বার suffix array বের করার কাজ (আশা করি তোমাদের মনে আছে

আমরা মোট $\log n$ বার sorting এর কাজ করে suffix array বানিয়েছিলাম?) করলেই হবে। এর পর তুমি প্রতিটি ছোট string এর জন্য দুইটি binary search করবে। একবারে তুমি suffix array এর প্রথম suffix টি বের করবে যার সাথে তোমার string এর lcp বের করে দেখো তা ছোট string এর সমান কিনা, আর আরেকবার একই ভাবে binary search করো তবে তুমি সেই রকম suffix দের সবশেষের টা বের করবা। যেহেতু ছোট string গুলি মাত্র 64 length হতে পারে তাই তুমি চাইলে linear search করতেই পারো। তাহলে মোট complexity দাঁড়ায় $\mathrm{O}(n \log 64 + 64M \log n)$.

তিনটি string দেয়া আছে। তোমাকে সবচেয়ে বড় string এর length বের করতে হবে যেন তা তিনটি string এরই substring হয়। তোমাকে যা করতে হবে তাহলো এই তিনটি string কে অব্যবহৃত character ব্যবহার করে জোড়া লাগাতে হবে। ধরা যাক এরকম দুইটি character হল # আর $?$. তাহলে আমাদের জোড়া লাগা string টা দেখতে হবে $S_1\#S_2?S_3$ এর মত। এখন তোমাকে এর suffix array বের করতে হবে। এখন কিছুটা line sweep বা sliding window এর মত কাজ করতে হবে। প্রথমে $i = 0$ সেট করো আর j কে 0 হতে বাড়াতে থাকো যেন $[i, j]$ রেঞ্জ এর মাঝে ঐ তিনটি string এরই suffix থাকে। এখন তুমি i আর j এর lcp বের করে তোমার উত্তর কে update করো। এবার i কে বাড়াও, এবং এর জন্য j কে "দরকারে" বাড়াতে থাকো যতক্ষণ না আবারো এই রেঞ্জ এর মাঝে তিনটি string এরই substring থাকে। তখন আবার lcp বের করে উত্তর কে update করবে। তিনটি string এরই substring আছে কিনা তা আসলে suffix গুলি জোড়া লাগা string এ কই আছে তা দেখেই বলতে পারবে ($rank$ ব্যবহার করে)।

তোমাকে একটা string দিয়ে বলা হল এতে থাকা সবচেয়ে বড় palindrome এর length বের করো। প্রথমেই বলে নেই, যখনই palindrome এর সমস্যা দেখবে তখনই জোড় আর বিজোড় palindrome কে আলাদা আলাদা করে চিন্তা করে দেখার কথা ভাববে। তাতে সমস্যা কিছুটা সহজ হয়ে যায়। যাই হোক। আমরা আপাতত বিজোড় length এর কথা ভাবি। আমরা যদি কোন ভাবে $S[i\ldots]$ আর $S[i\ldots 0]$ এর lcp বের করতে পারতাম তাহলেই হয়ে যেতো। খেয়াল করো $S[i\ldots]$ হল i তম suffix আর $S[i\ldots 0]$ কে ভাবতে পারো S এর উলটো string এর suffix. আমরা যদি S আর উলটো S কে জোড়া লাগিয়ে তার suffix array বের করি তাহলেই কিন্তু আমাদের সমস্যা সমাধান হয়ে যায়। আমরা প্রতি i আর $2|S| - i - 1$ এর lcp বের করব তাহলেই হবে।

এখন কিছু কঠিন সমস্যা দেখা যাক। মনে করো একটি string S দেয়া আছে। এমন একটি সবচেয়ে ছোট string T বের করতে হবে যেন অনেকগুলি T পরপর জোড়া লাগিয়ে S বানানো যায়। জোড়া লাগানোর সময় T গুলি পরস্পর এর মাঝে overlap করতে পারে। যেমন

```
ababbababbababababbababababbababbaba
ababbaba
      ababbaba
              ababbaba
                      ababbaba
                            ababbaba
```

এর দুই রকম সমাধান বলছি। দুই সমাধানেই আমাদের suffix array লাগবে। প্রথম সমাধান কিছুটা এরকম- খেয়াল করো T কে অবশ্যই S এর prefix হতে হবে। আমরা যা করব তাহলো এক এক করে prefix এর length এক হতে বাড়াতে থাকব আর চেক করব যে এই prefix টা আমাদের কাঙ্ক্ষিত T হিসাবে কাজ করবে কিনা। আমরা কিছুটা sliding window টাইপ এর টেকনিক ব্যবহার করব। প্রথমে এক length এর জন্য আমরা L এবং R দিয়ে suffix array তে সেসব suffix কে bound করব যাদের প্রথম character S এর প্রথম character এর সমান হয়। এর পর যখন আরও এক length বাড়িয়ে 2 length কে ধরব তখন L এবং R যথাক্রমে বাড়িয়ে ও কমিয়ে এমন রেঞ্জ এ ঠিক করতে হবে যেন এদের মাঝের সবার সাথে 2 length মিল থাকে। এরকম করে প্রত্যেক length এ আমাদের এই কাজ করে window ঠিক করতে হবে। এখন বলব প্রতিবার আর কি কাজ করতে হবে। একদম প্রথম বার এই window এর ভিতর থাকা সব suffix এর index গুলি একটি set বা BST তে ঢুকাতে হবে, আর পরে যখন L, R কমিয়ে রেঞ্জ ঠিক করছিলাম তখন আমাদের set বা BST থেকে ঐ index বের করতে হবে। এখন এই যে ঢুকানো বা বের করা এই সময় পাশাপাশি দুইটি index এর মাঝের দূরত্ব আমাদেরকে একটি heap বা priority queue বা আরেকটি set এ রাখতে হবে।

এই কাজটা এর আগেও আমরা করেছি। তাও বলই, যখন তুমি কোন একটি index ঢুকাবে তখন এর দুইপাশের দুইটি index দেখো আর তাদের মাঝের difference কে heap থেকে বাদ দিয়ে নতুন index এর সাথে দুই পাশের দুইটি index এর মাঝের দূরত্ব heap এ রাখতে হবে। আর কোন একটি index বাদ দেয়াড় সময় তার সাথে তার পাশের দুইটি index এর difference কে heap থেকে মুছে ফেলে তাদের মাঝের difference কে heap এ রাখতে হবে। এবার যেই জিনিসটা দেখতে হবে তাহলো heap এর সর্বোচ্চ সংখ্যা কত। যদি সেটা সেই বারের prefix এর length এর সমান বা ছোট হয় এর মানে হল এটিই আমাদের উত্তর। এটুকু যদি বুঝে থাকো তাহলে কেন এটি সঠিক তাও একটু চিন্তা করলে বুঝতে পারবে। এখন আসা যাক দ্বিতীয় সমাধানে। দ্বিতীয় সমাধানের জন্য আমাদের S এর প্রতিটি index এর জন্য মূল string এর সাথে lcp লাগবে। বা অন্য ভাবে বলতে গেলে আমাদের আসলে Z function লাগবে। আর এই প্রবলেমের ক্ষেত্রে আমরা ধরে নিবো $S[0\ldots]$ এই suffix এর জন্য S এর সাথে lcp হল $|S|$. যাই হোক, এখন তুমি এই lcp অনুসারে suffix এর index কে সর্ট করো বড় হতে ছোট ক্রমে। এখন এদের একে একে একটি BST বা set এ যোগ করো। আগের মত একটি heap নিয়ে তাতে BST বা set এ ঢুকানো index গুলির মাঝের দূরত্ব ঢুকাতে থাকো। মনে করো তুমি এখন যেই lcp এর length কে নিয়েছ তার length k আর heap এ থাকা index এর মাঝের সবচেয়ে বড় difference হল h. এখন যদি $h \leq k$ হয় তাহলে h হবে আমাদের একটি candidate উত্তর। এভাবে সকল candidate দের মাঝে থেকে সবচেয়ে কমটি হবে আমাদের উত্তর।

একটি string S দেয়া আছে তোমাকে এর প্রতিটি prefix এর জন্য সর্বোচ্চ period বের করতে হবে। কোন একটি prefix এর জন্য period হবে k যদি এমন একটি string A থাকে যেখানে A কে পরপর k বার জোড়া লাগালে সেই prefix পাওয়া যায়। আমার মনে হয় তোমরা এই সমস্যার সমাধান kmp বা z function ব্যবহার করে কেমনে করতে হবে তা একটু চিন্তা করলে করে ফেলতে পারবা। Suffix array এর সমাধান ও একই রকম। তোমাকে প্রতিটি suffix এর জন্য মূল string এর সাথে lcp বের করতে হবে। ধরো i index এর জন্য lcp হল k. তাহলে যদি $k \geq 2i$ হয় তাহলে প্রতিটি $2i, 3i\ldots$ (এবং এই i এর multiple গুলিকে অবশ্যই k এর থেকে বড় হওয়া যাবে না) prefix এর জন্য তুমি একটি repeatative string $S[0\ldots i]$ পাবে। অর্থাৎ $2i$ এর period 2, $3i$ এর period 3 এরকম আর কি। তোমরা i এর প্রতিটি multiple এ গিয়ে update করে দিয়ে আসতে পারো তার period যাতে সব শেষে তুমি prefix গুলির সর্বোচ্চ period পাও। প্রতিটি multiple এ গিয়ে update করা কিন্তু ওতটা costly না। তুমি 1 length এর জন্য update করবে $n/1$ বার, 2 এর জন্য $n/2$ বার এরকম করে n এর জন্য n/n বার। আর আশা করি তোমরা জানো যে $n/1 + n/2 + \ldots n/n = n\log n$.

তোমাকে একটা string S দেয়া হবে। তোমাকে এর ভিতরে থেকে একটি substring বের করতে হবে যার period সর্বোচ্চ হয়। কোন একটি string এর period k মানে একটি string A আছে যাকে k বার পর পর জোড়া লাগালে মূল string টা পাওয়া যায়। আমরা এই A এর length এর উপর লুপ চালাব। ধরা যাক এই length হল L এবং আমরা 1 হতে $n = |S|$ পর্যন্ত L এর লুপ চালাব। আমরা যা করব মূল string S কে L ভাগে ভাগে ভাগ করব। যেমন $babbabaabaabaabab$ কে আমরা যদি $L = 3$ এর জন্য ভাগ করতে চাই তাহলে হবে $(bab)(bab)(aab)(aab)(aab)(ab)$. শেষ ভাগে আসলে L এর থেকে কম থাকলে সমস্যা নেই। এখন ধরো আমরা প্রতিটি ভাগকে নাম্বার দিচ্ছিঃ 0 তম ভাগ, 1 তম ভাগ এরকম। আমাদের যা করতে হবে তাহলো পাশাপাশি দুইটি ভাগের lcp বের করতে হবে। যেমন উপরের ভাগের জন্য $lcp(0,1) = 3$, $lcp(1,2) = 0\ldots lcp(4,5) = 2$. একই ভাবে আমাদের lcs বের করতে হবে মানে longest common suffix যেমন $lcs(1,2) = 2$ কারণ bab আর aab এর lcs হল 2. lcs আসলে তোমরা lcp এর মত বের করতে পারো, শুধু string টাকে উলটো করে নিতে হবে। এখন তোমাকে এই lcp দেখে দেখে বুঝতে হবে কোন কোন ভাগ সমান। এটা তো বুঝা খুবই সোজা, যদি lcp L এর সমান বা বড় হয় এর মানে সমান। এখন উপরের ভাগের দিকে দেখো। তুমি কি বলবে যে পাশাপাশি aab তিনটা আছে তাই আসলে 3 length এর কোন একটি A কে 3 বারের বেশি repeat করা যাবে না? না খেয়াল করলে দেখবে উপরের উদাহরণে aba কে 4 বার repeat করা যায়। তাহলে তা কেমনে বের করা যায়? প্রথমত আমরা সমান সেগমেন্ট গুলি বের করি, যেমন উপরের উদাহরণে $2, 3, 4$ সমান segment. এবার দেখতে হবে এই segment গুলিকে ডানে ও বামে কত দূর বাড়ানো যায়। যেমন $lcs(1,2) = 2$ আর $lcp(4,5) = 2$. এর মানে এই 3 length করে সেগমেন্ট নেবার কাজ বাম দিকে আরও 2 ঘর সরানো সম্ভব। তাহলে মোট $2 + 3 \times 3 + 2 = 13$

ঘর জুড়ে 3 length এর string এর repeat করার কাজ করা সম্ভব। আর আমরা জানি $\lfloor 13/3 \rfloor = 4$ তার মানে আমরা 3 length এর একটি string পেয়েছি যা 4 বার repeat হয়। শেষ! এই সমাধানের complexity $\mathrm{O}(n \log n)$.

Geometry Concepts: Basic Concepts

By **lbackstrom** — *topcoder member*

Discuss this article in the forums

Introduction

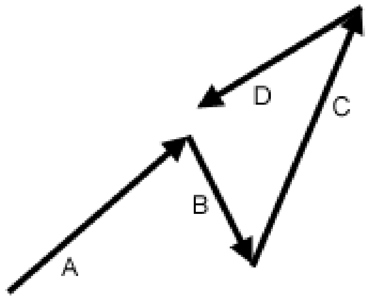
Many topcoders seem to be mortally afraid of geometry problems. I think it's safe to say that the majority of them would be in favor of a ban on topcoder geometry problems. However, geometry is a very important part of most graphics programs, especially computer games, and geometry problems are here to stay. In this article, I'll try to take a bit of the edge off of them, and introduce some concepts that should make geometry problems a little less frightening.

Vectors

Vectors are the basis of a lot of methods for solving geometry problems. Formally, a vector is defined by a direction and a magnitude. In the case of two-dimension geometry, a vector can be represented as pair of numbers, x and y, which gives both a direction and a magnitude. For example, the line segment from (1,3) to (5,1) can be represented by the vector (4,-2). It's important to understand, however, that the vector defines only the direction and magnitude of the segment in this case, and does not define the starting or ending locations of the vector.

Vector Addition

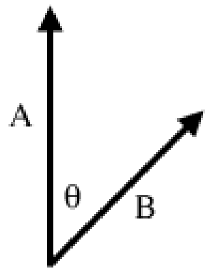
There are a number of mathematical operations that can be performed on vectors. The simplest of these is addition: you can add two vectors together and the result is a new vector. If you have two vectors (x_1, y_1) and (x_2, y_2), then the sum of the two vectors is simply (x_1+x_2, y_1+y_2). The image below shows the sum of four vectors. Note that it doesn't matter which order you add them up in – just like regular addition. Throughout these articles, we will use plus and minus signs to denote vector addition and subtraction, where each is simply the piecewise addition or subtraction of the components of the vector.



The sum of vectors A+B+C+D

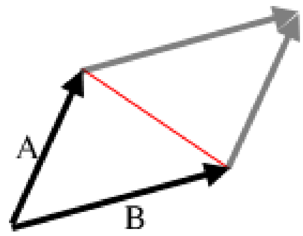
Dot Product

The addition of vectors is relatively intuitive; a couple of less obvious vector operations are dot and cross products. The dot product of two vectors is simply the sum of the products of the corresponding elements. For example, the dot product of (x_1, y_1) and (x_2, y_2) is $x_1*x_2 + y_1*y_2$. Note that this is not a vector, but is simply a single number (called a scalar). The reason this is useful is that the dot product, $A \cdot B = |A||B|Cos(\theta)$, where θ is the angle between the A and B. $|A|$ is called the norm of the vector, and in a 2-D geometry problem is simply the length of the vector, $sqrt(x^2+y^2)$. Therefore, we can calculate $Cos(\theta) = (A \cdot B)/(|A||B|)$. By using the acos function, we can then find θ. It is useful to recall that $Cos(90) = 0$ and $Cos(0) = 1$, as this tells you that a dot product of 0 indicates two perpendicular lines, and that the dot product is greatest when the lines are parallel. A final note about dot products is that they are not limited to 2-D geometry. We can take dot products of vectors with any number of elements, and the above equality still holds.



Cross Product

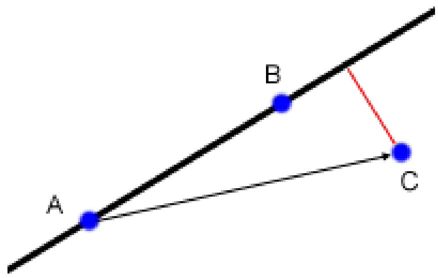
An even more useful operation is the cross product. The cross product of two 2-D vectors is $x_1*y_2 - y_1*x_2$ Technically, the cross product is actually a vector, and has the magnitude given above, and is directed in the +z direction. Since we're only working with 2-D geometry for now, we'll ignore this fact, and use it like a scalar. Similar to the dot product, $A \times B = |A||B|Sin(\theta)$. However, θ has a slightly different meaning in this case: $|\theta|$ is the angle between the two vectors, but θ is negative or positive based on the right-hand rule. In 2-D geometry this means that if A is less than 180 degrees clockwise from B, the value is positive. Another useful fact related to the cross product is that the absolute value of $|A||B|Sin(\theta)$ is equal to the area of the parallelogram with two of its sides formed by A and B. Furthermore, the triangle formed by A, B and the red line in the diagram has half of the area of the parallelogram, so we can calculate its area from the cross product also.



Parallelogram from A and B

Line-Point Distance

Finding the distance from a point to a line is something that comes up often in geometry problems. Lets say that you are given 3 points, A, B, and C, and you want to find the distance from the point C to the line defined by A and B (recall that a line extends infinitely in either direction). The first step is to find the two vectors from A to B (AB) and from A to C (AC). Now, take the cross product AB x AC, and divide by |AB|. This gives you the distance (denoted by the red line) as (AB x AC)/|AB|. The reason this works comes from some basic high school level geometry. The area of a triangle is found as base*height/2. Now, the area of the triangle formed by A, B and C is given by (AB x AC)/2. The base of the triangle is formed by AB, and the height of the triangle is the distance from the line to C. Therefore, what we have done is to find twice the area of the triangle using the cross product, and then divided by the length of the base. As always with cross products, the value may be negative, in which case the distance is the absolute value.



Things get a little bit trickier if we want to find the distance from a line segment to a point. In this case, the nearest point might be one of the endpoints of the segment, rather than the closest point on the line. In the diagram above, for example, the closest point to C on the line defined by A and B is not on the segment AB, so the point closest to C is B. While there are a few different ways to check for this special case, one way is to apply the dot product. First, check to see if the nearest point on the line AB is beyond B (as in the example above) by taking AB · BC. If this value is greater than 0, it means that the angle between AB and BC is between -90 and 90, exclusive, and therefore the nearest point on the segment AB will be B. Similarly, if BA · AC is greater than 0, the nearest point is A. If both dot products are negative, then the nearest point to C is somewhere along the segment. (There is another way to do this, which I'll discuss here).

```
//Compute the dot product AB · BC
    int dot(int[] A, int[] B, int[] C){
        AB = new int[2];
        BC = new int[2];
```

```
        AB[0] = B[0]-A[0];
        AB[1] = B[1]-A[1];
        BC[0] = C[0]-B[0];
        BC[1] = C[1]-B[1];
        int dot = AB[0] * BC[0] + AB[1] * BC[1];
        return dot;
    }
    //Compute the cross product AB x AC
    int cross(int[] A, int[] B, int[] C){
        AB = new int[2];
        AC = new int[2];
        AB[0] = B[0]-A[0];
        AB[1] = B[1]-A[1];
        AC[0] = C[0]-A[0];
        AC[1] = C[1]-A[1];
        int cross = AB[0] * AC[1] - AB[1] * AC[0];
        return cross;
    }
    //Compute the distance from A to B
    double distance(int[] A, int[] B){
        int d1 = A[0] - B[0];
        int d2 = A[1] - B[1];
        return sqrt(d1*d1+d2*d2);
    }
    //Compute the distance from AB to C
    //if isSegment is true, AB is a segment, not a line.
    double linePointDist(int[] A, int[] B, int[] C, boolean isSegment){
        double dist = cross(A,B,C) / distance(A,B);
        if(isSegment){
            int dot1 = dot(A,B,C);
            if(dot1 > 0)return distance(B,C);
            int dot2 = dot(B,A,C);
            if(dot2 > 0)return distance(A,C);
        }
        return abs(dist);
    }
```

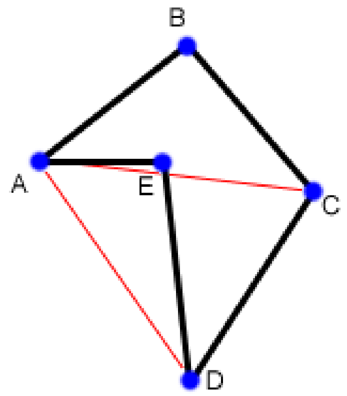
That probably seems like a lot of code, but lets see the same thing with a point class and some operator overloading in C++ or C#. The * operator is the dot product, while ^ is cross product, while + and – do what you would expect.

```
//Compute the distance from AB to C
    //if isSegment is true, AB is a segment, not a line.
    double linePointDist(point A, point B, point C, bool isSegment){
        double dist = ((B-A)^(C-A)) / sqrt((B-A)*(B-A));
        if(isSegment){
            int dot1 = (C-B)*(B-A);
            if(dot1 > 0)return sqrt((B-C)*(B-C));
            int dot2 = (C-A)*(A-B);
            if(dot2 > 0)return sqrt((A-C)*(A-C));
        }
        return abs(dist);
    }
```

Operator overloading is beyond the scope of this article, but I suggest that you look up how to do it if you are a C# or C++ coder, and write your own 2-D point class with some handy operator overloading. It will make a lot of geometry problems a lot simpler.

Polygon Area

Another common task is to find the area of a polygon, given the points around its perimeter. Consider the non-convex polygon below, with 5 points. To find its area we are going to start by triangulating it. That is, we are going to divide it up into a number of triangles. In this polygon, the triangles are ABC, ACD, and ADE. But wait, you protest, not all of those triangles are part of the polygon! We are going to take advantage of the signed area given by the cross product, which will make everything work out nicely. First, we'll take the cross product of AB x AC to find the area of ABC. This will give us a negative value, because of the way in which A, B and C are oriented. However, we're still going to add this to our sum, as a negative number. Similarly, we will take the cross product AC x AD to find the area of triangle ACD, and we will again get a negative number. Finally, we will take the cross product AD x AE and since these three points are oriented in the opposite direction, we will get a positive number. Adding these three numbers (two negatives and a positive) we will end up with a negative number, so will take the absolute value, and that will be area of the polygon.



The reason this works is that the positive and negative number cancel each other out by exactly the right amount. The area of ABC and ACD ended up contributing positively to the final area, while the area of ADE contributed negatively. Looking at the original polygon, it is obvious that the area of the polygon is the area of ABCD (which is the same as ABC + ABD) minus the area of ADE. One final note, if the total area we end up with is negative, it means that the points in the polygon were given to us in clockwise order. Now, just to make this a little more concrete, lets write a little bit of code to find the area of a polygon, given the coordinates as a 2-D array, p.

```
int area = 0;
int N = lengthof(p);
//We will triangulate the polygon
//into triangles with points p[0],p[i],p[i+1]

for(int i = 1; i+1<N; i++){
    int x1 = p[i][0] - p[0][0];
    int y1 = p[i][1] - p[0][1];
    int x2 = p[i+1][0] - p[0][0];
    int y2 = p[i+1][1] - p[0][1];
    int cross = x1*y2 - x2*y1;
    area += cross;
}
return abs(cross/2.0);
```

Notice that if the coordinates are all integers, then the final area of the polygon is one half of an integer.

…continue to Section 2

More Resources

Member Tutorials

Geometry Concepts: Line Intersection and its Applications

By **lbackstrom** — *topcoder member*

…read Section 1

In the previous section we saw how to use vectors to solve geometry problems. Now we are going to learn how to use some basic linear algebra to do line intersection, and then apply line intersection to a couple of other problems.

Line-Line Intersection

One of the most common tasks you will find in geometry problems is line intersection. Despite the fact that it is so common, a lot of coders still have trouble with it. The first question is, what form are we given our lines in, and what form would we like them in? Ideally, each of our lines will be in the form $Ax+By=C$, where A, B and C are the numbers which define the line. However, we are rarely given lines in this format, but we can easily generate such an equation from two points. Say we are given two different points, (x_1, y_1) and (x_2, y_2), and want to find A, B and C for the equation above. We can do so by setting

$A = y_2-y_1$

$B = x_1-x_2$

$C = A*x_1+B*y_1$

Regardless of how the lines are specified, you should be able to generate two different points along the line, and then generate A, B and C. Now, lets say that you have lines, given by the equations:

$A_1x + B_1y = C_1$

$A_2x + B_2y = C_2$

To find the point at which the two lines intersect, we simply need to solve the two equations for the two unknowns, x and y.

```
double det = A1*B2 - A2*B1
    if(det == 0){
        //Lines are parallel
    }else{
        double x = (B2*C1 - B1*C2)/det
        double y = (A1*C2 - A2*C1)/det
    }
```

To see where this comes from, consider multiplying the top equation by B_2, and the bottom equation by B_1. This gives you

$A_1B_2x + B_1B_2y = B_2C_1$

$A_2B_1x + B_1B_2y = B_1C_2$

Now, subtract the bottom equation from the top equation to get

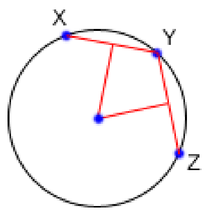
$A_1B_2x - A_2B_1x = B_2C_1 - B_1C_2$

Finally, divide both sides by $A_1B_2 - A_2B_1$, and you get the equation for x. The equation for y can be derived similarly.

This gives you the location of the intersection of two lines, but what if you have line segments, not lines. In this case, you need to make sure that the point you found is on both of the line segments. If your line segment goes from (x_1,y_1) to (x_2,y_2), then to check if (x,y) is on that segment, you just need to check that $\min(x_1,x_2) \le x \le \max(x_1,x_2)$, and do the same thing for y. You must be careful about double precision issues though. If your point is right on the edge of the segment, or if the segment is horizontal or vertical, a simple comparison might be problematic. In these cases, you can either do your comparisons with some tolerance, or else use a fraction class.

Finding a Circle From 3 Points

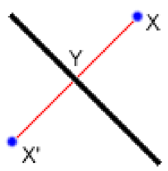
Given 3 points which are not colinear (all on the same line) those three points uniquely define a circle. But, how do you find the center and radius of that circle? This task turns out to be a simple application of line intersection. We want to find the perpendicular bisectors of XY and YZ, and then find the intersection of those two bisectors. This gives us the center of the circle.



To find the perpendicular bisector of XY, find the line from X to Y, in the form $Ax+By=C$. A line perpendicular to this line will be given by the equation $-Bx+Ay=D$, for some D. To find D for the particular line we are interested in, find the midpoint between X and Y by taking the midpoint of the x and y components independently. Then, substitute those values into the equation to find D. If we do the same thing for Y and Z, we end up with two equations for two lines, and we can find their intersections as described above.

Reflection

Reflecting a point across a line requires the same techniques as finding a circle from 3 points. First, notice that the distance from X to the line of reflection is the same as the distance from X' to the line of reflection. Also note that the line between X and X' is perpendicular to the line of reflection. Now, if the line of reflection is given as $Ax+By=C$, then we already know how to find a line perpendicular to it: $-Bx+Ay=D$. To find D, we simply plug in the coordinates for X. Now, we can find the intersection of the two lines at Y, and then find $X' = Y - (X - Y)$.



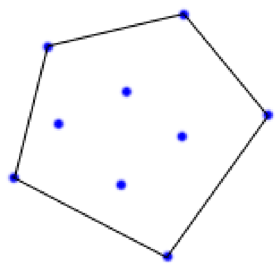
Rotation

Rotation doesn't really fit in with line intersection, but I felt that it would be good to group it with reflection. In fact, another way to find the reflected point is to rotate the original point 180 degrees about Y.

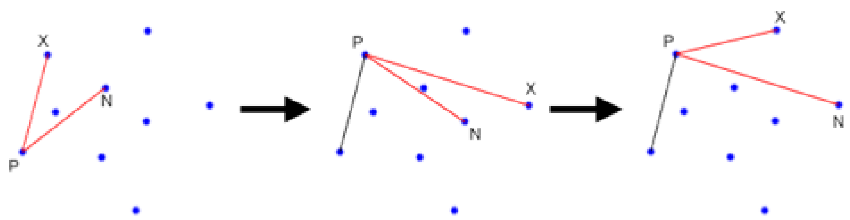
Imagine that we want to rotate one point around another, counterclockwise by θ degrees. For simplicity, lets assume that we are rotating about the origin. In this case, we can find that $x' = x \cos(\theta) - y \sin(\theta)$ and $y' = x \sin(\theta) + y \cos(\theta)$. If we are rotating about a point other than the origin, we can account for this by shifting our coordinate system so that the origin is at the point of rotation, doing the rotation with the above formulas, and then shifting the coordinate system back to where it started.

Convex Hull

A convex hull of a set of points is the smallest convex polygon that contains every one of the points. It is defined by a subset of all the points in the original set. One way to think about a convex hull is to imagine that each of the points is a peg sticking up out of a board. Take a rubber band and stretch it around all of the points. The polygon formed by the rubber band is a convex hull. There are many different algorithms that can be used to find the convex hull of a set of points. In this article, I'm just going to describe one of them, which is fast enough for most purposes, but is quite slow compared to some of the other algorithms.



First, loop through all of your points and find the leftmost point. If there is a tie, pick the highest point. You know for certain that this point will be on the convex hull, so we'll start with it. From here, we are going to move clockwise around the edge of the hull, picking the points on the hull, one at a time. Eventually, we will get back to the start point. In order to find the next point around the hull, we will make use of cross products. First, we will pick an unused point, and set the next point, N, to that point. Next, we will iterate through each unused points, X, and if `(X-P) x (N-P)` (where P is the previous point) is negative, we will set N to X. After we have iterated through each point, we will end up with the next point on the convex hull. See the diagram below for an illustration of how the algorithm works. We start with P as the leftmost point. Now, say that we have N and X as shown in the leftmost frame. In this case the cross product will be negative, so we will set N = X, and there will be no other unused points that make the cross product negative, and hence we will advance, setting P = N. Now, in the next frame, we will end up setting N = X again, since the cross product here will be negative. However, we aren't done yet because there is still another point that will make the cross product negative, as shown in the final frame.



The basic idea here is that we are using the cross product to find the point which is furthest counterclockwise from our current position at P. While this may seem fairly straightforward, it becomes a little bit tricky when dealing with colinear points. If you have no colinear points on the hull, then the code is very straightforward.

```
convexHull(point[] X){
    int N = lengthof(X);
    int p = 0;
    //First find the leftmost point
    for(int i = 1; i<N; i++){
        if(X[i] < X[p])
            p = i;
    }
    int start = p;
    do{
```

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

            //Don't go back to the same point you came from
            if(i == p)continue;

            //If there is no N yet, set it to i
            if(n == -1)n = i;
            int cross = (X[i] - X[p]) x (X[n] - X[p]);

            if(cross < 0){
                //As described above, set N=X
                n = i;
            }
        }
        p = n;
    }while(start!=p);
}
```

Once we start to deal with colinear points, things get trickier. Right away we have to change our method signature to take a boolean specifying whether to include all of the colinear points, or only the necessary ones.

```
//If onEdge is true, use as many points as possible for
    //the convex hull, otherwise as few as possible.
    convexHull(point[] X, boolean onEdge){
        int N = lengthof(X);
        int p = 0;
        boolean[] used = new boolean[N];
        //First find the leftmost point
        for(int i = 1; i<N; i++){
            if(X[i] < X[p])
                p = i;
        }
        int start = p;
        do{
            int n = -1;
            int dist = onEdge?INF:0;
            for(int i = 0; i<N; i++){
                //X[i] is the X in the discussion

                //Don't go back to the same point you came from
                if(i==p)continue;

                //Don't go to a visited point
                if(used[i])continue;
```

```
            //If there is no N yet, set it to X
            if(n == -1)n = i;
            int cross = (X[i] - X[p]) x (X[n] - X[p]);

            //d is the distance from P to X
            int d = (X[i] - X[p]) · (X[i] - X[p]);
            if(cross < 0){
                //As described above, set N=X
                n = i;
                dist = d;
            }else if(cross == 0){
                //In this case, both N and X are in the
                //same direction.  If onEdge is true, pick the
                //closest one, otherwise pick the farthest one.
                if(onEdge && d < dist){
                    dist = d;
                    n = i;
                }else if(!onEdge && d > dist){
                    dist = d;
                    n = i;
                }
            }
        }
        p = n;
        used[p] = true;
    }while(start!=p);
}
```

...continue to Section 3

More Resources

Member Tutorials
Read more than 40 data science tutorials written by topcoder members.

Problem Set Analysis
Read editorials explaining the problem and solution for each Single Round Match (SRM).

Data Science Guide
New to topcoder's data science track? Read this guide for an overview on how to get started in the arena and how competitions work.

Help Center
Need specifics about the process or the rules? Everything you need to know about competing at topcoder can be found in the Help Center.

Member Forums
Join your peers in our member forums and ask questions from the real experts - topcoder members!

Geometry Concepts: Using Geometry in Topcoder Problems

By **lbackstrom** — *topcoder member*

…read Section 2

PointInPolygon (SRM 187)
Requires: Line-Line Intersection, Line-Point Distance

First off, we can use our Line-Point Distance code to test for the "BOUNDARY" case. If the distance from any segment to the test point is 0, then return "BOUNDARY". If you didn't have that code pre-written, however, it would probably be easier to just check and see if the test point is between the minimum and maximum x and y values of the segment. Since all of the segments are vertical or horizontal, this is sufficient, and the more general code is not necessary.

Next we have to check if a point is in the interior or the exterior. Imagine picking a point in the interior and then drawing a ray from that point out to infinity in some direction. Each time the ray crossed the boundary of the polygon, it would cross from the interior to the exterior, or vice versa. Therefore, the test point is on the interior if, and only if, the ray crosses the boundary an odd number of times. In practice, we do not have to draw a raw all the way to infinity. Instead, we can just use a very long line segment from the test point to a point that is sufficiently far away. If you pick the far away point poorly, you will end up having to deal with cases where the long segment touches the boundary of the polygon where two edges meet, or runs parallel to an edge of a polygon — both of which are tricky cases to deal with. The quick and dirty way around this is to pick two large random numbers for the endpoint of the segment. While this might not be the most elegant solution to the problem, it works very well in practice. The chance of this segment intersecting anything but the interior of an edge are so small that you are almost guaranteed to get the right answer. If you are really concerned, you could pick a few different random points, and take the most common answer.

```
String testPoint(verts, x, y){
    int N = lengthof(verts);
    int cnt = 0;
    double x2 = random()*1000+1000;
    double y2 = random()*1000+1000;
    for(int i = 0; i<N; i++){
        if(distPointToSegment(verts[i],verts[(i+1)%N],x,y) == 0)
            return "BOUNDARY";
        if(segmentsIntersect((verts[i],verts[(i+1)%N],{x,y},{x2,y2}))
            cnt++;
    }
    if(cnt%2 == 0)return "EXTERIOR";
    else return "INTERIOR";
}
```

TVTower(SRM 183)
Requires: Finding a Circle From 3 Points

In problems like this, the first thing to figure out is what sort of solutions might work. In this case, we want to know what sort of circles we should consider. If a circle only has two points on it, then, in most cases, we can make a slightly smaller circle, that still has those two points on it. The only exception to this is when the two points are exactly opposite each other on the circle. Three points, on the other hand, uniquely define a circle, so if there are three points on the edge of a circle, we cannot make it slightly smaller, and still have all three of them on the circle. Therefore, we want to consider two different types of circles: those with two points exactly opposite each other, and those with three points on the circle. Finding the center of the first type of circle is trivial — it is simply halfway between the two points. For the other case, we can use the method for Finding a Circle From 3 Points. Once we find the center of a potential circle, it is then trivial to find the minimum radius.

```
int[] x, y;
int N;
double best = 1e9;
void check(double cx, double cy){
    double max = 0;
    for(int i = 0; i< N; i++){
        max = max(max,dist(cx,cy,x[i],y[i]));
    }
    best = min(best,max);
}
double minRadius(int[] x, int[] y){
    this.x = x;
    this.y = y;
    N = lengthof(x);
    if(N==1)return 0;
    for(int i = 0; i<N; i++){
        for(int j = i+1; j<N; j++){
            double cx = (x[i]+x[j])/2.0;
            double cy = (y[i]+y[j])/2.0;
            check(cx,cy);
            for(int k = j+1; k<N; k++){
                //center gives the center of the circle with
                //(x[i],y[i]), (x[j],y[j]), and (x[k],y[k]) on
                //the edge of the circle.
                double[] c = center(i,j,k);
                check(c[0],c[1]);
            }
        }
    }
    return best;
}
```

Satellites (SRM 180)

Requires: Line-Point Distance

This problem actually requires an extension of the Line-Point Distance method discussed previously. It is the same basic principle, but the formula for the cross product is a bit different in three dimensions.

The first step here is to convert from spherical coordinates into (x,y,z) triples, where the center of the earth is at the origin.

```
double x = sin(lng/180*PI)*cos(lat/180*PI)*alt;
    double y = cos(lng/180*PI)*cos(lat/180*PI)*alt;
    double z = sin(lat/180*PI)*alt;
```

Now, we want to take the cross product of two 3-D vectors. As I mentioned earlier, the cross product of two vectors is actually a vector, and this comes into play when working in three dimensions. Given vectors (x_1,y_1,z_1) and (x_2,y_2,z_2) the cross product is defined as the vector (i,j,k) where

$$i = y_1z_2 - y_2z_1;$$
$$j = x_2z_1 - x_1z_2;$$
$$k = x_1y_2 - x_2y_1;$$

Notice that if $z_1 = z_2 = 0$, then i and j are 0, and k is equal to the cross product we used earlier. In three dimensions, the cross product is still related to the area of the parallelogram with two sides from the two vectors. In this case, the area of the parallelogram is the norm of the vector: `sqrt(i*i+j*j+k*k)`.

Hence, as before, we can determine the distance from a point (the center of the earth) to a line (the line from a satellite to a rocket). However, the closest point on the line segment between a satellite and a rocket may be one of the end points of the segment, not the closest point on the line. As before, we can use the dot product to check this. However, there is another way which is somewhat simpler to code. Say that you have two vectors originating at the origin, S and R, going to the satellite and the rocket, and that |X| represents the norm of a vector X.
Then, the closest point to the origin is R if $|R|^2 + |R-S|^2 \leq |S|^2$ and it is S if $|S|^2 + |R-S|^2 \leq |R|^2$
Naturally, this trick works in two dimensions also.

Further Problems
Once you think you've got a handle on the three problems above, you can give these ones a shot. You should be able to solve all of them with the methods I've outlined, and a little bit of cleverness. I've arranged them in what I believe to ascending order of difficulty.
ConvexPolygon (SRM 166)
Requires: Polygon Area

Surveyor (TCCC '04 Qual 1)
Requires: Polygon Area

Travel (TCI '02)
Requires: Dot Product

Parachuter (TCI '01 Round 3)
Requires: Point In Polygon, Line-Line Intersection

PuckShot (SRM 186)
Requires: Point In Polygon, Line-Line Intersection

ElectronicScarecrows (SRM 173)
Requires: Convex Hull, Dynamic Programming

Mirrors (TCI '02 Finals)
Requires: Reflection, Line-Line Intersection

Symmetry (TCI '02 Round 4)
Requires: Reflection, Line-Line Intersection

Warehouse (SRM 177)
Requires: Line-Point Distance, Line-Line Intersection

The following problems all require geometry, and the topics discussed in this article will be useful. However, they all require some additional skills. If you got stuck on them, the editorials are a good place to look for a bit of help. If you are still stuck, there has yet to be a problem related question on the round tables that went unanswered.
DogWoods (SRM 201)
ShortCut (SRM 215)
SquarePoints (SRM 192)
Tether (TCCC '03 W/MW Regional)
TurfRoller (SRM 203)
Watchtower (SRM 176)

More Resources

Basics of Combinatorics

By **x-ray**– *TopCoder Member*
Discuss this article in the forums

Introduction
Counting the objects that satisfy some criteria is a very common task in both TopCoder problems and in real-life situations. The myriad ways of counting the number of elements in a set is one of the main tasks in combinatorics, and I'll try to describe some basic aspects of it in this tutorial. These methods are used in a range of applications, from discrete math and probability theory to statistics, physics, biology, and more.

Combinatorial primitives
Let's begin with a quick overview of the basic rules and objects that we will reference later.

The rule of sum

$$A \cap B = \varnothing \Rightarrow |A| + |B| = |A \cup B|$$

The rule of product

$$|A| \cdot |B| = |A \times B|$$

For example, if we have three towns — A, B and C — and there are 3 roads from A to B and 5 roads from B to C, then we can get from A to C through B in 3*5=15 different ways.

These rules can be used for a finite collections of sets.

Permutation without repetition

When we choose *k* objects from *n*-element set in such a way that the order matters and each object can be chosen only once:

$$(n)_k = P_k^n = \frac{n!}{(n-k)!}$$

For example, suppose we are planning the next 3 challenges and we have a set of 10 easy problems to choose from. We will only use one easy problem in each contest, so we can choose our problems in $(10)_3 = \frac{10!}{(10-3)!}$ different ways.

Permutation (variation) with repetition

The number of possible choices of ***k*** objects from a set of ***n*** objects when order is important and one object can be chosen more than once:

$$n^k$$

For example, if we have 10 different prizes that need to be divided among 5 people, we can do so in 5^{10} ways.

Permutation with repetition

The number of different permutations of ***n*** objects, where there are n_1 indistinguishable objects of type 1, n_2 indistinguishable objects of type 2,…, and n_k indistinguishable objects of type ***k*** ($n_1+n_2+\ldots+n_k=n$), is:

$$\binom{n}{n_1, n_2, \ldots, n_k} = C_n(n_1, n_2, \ldots, n_k) = \frac{n!}{n_1! \cdot n_2! \cdot \ldots \cdot n_k!}$$

For example, if we have 97 coders and want to assign them to 5 rooms (rooms 1-4 have 20 coders in each, while the 5th room has 17), then there are $\binom{97}{20, 20, 20, 20, 17} = \frac{97!}{20! \cdot 20! \cdot 20! \cdot 20! \cdot 17!}$ possible ways to do it.

Combinations without repetition

In combinations we choose a set of elements (rather than an arrangement, as in permutations) so the order doesn't matter. The number of different k-element subsets (when each element can be chosen only once) of n-element set is:

$$\binom{n}{k} = C_k^n = \frac{n!}{k! \cdot (n-k)!}$$

For example, if we have 7 different colored balls, we can choose any 3 of them in $\binom{7}{3} = \frac{7!}{3! \cdot (7-3)!}$ different ways.

Combination with repetition

Let's say we choose k elements from an n-element set, the order doesn't matter and each element can be chosen more than once. In that case, the number of different combinations is:

$$f_k^n = \binom{n+k-1}{k} = \frac{(n+k-1)!}{k! \cdot (n-1)!}$$

For example, let's say we have 11 identical balls and 3 different pockets, and we need to calculate the number of different divisions of these balls to the pockets. There would be $f_{11}^3 = \binom{3+11-1}{11} = \frac{(3+11-1)!}{11! \cdot (3-1)!}$ different combinations.

It is useful to know that f_k^n is also the number of integer solutions to this equation:

$$x_1 + x_2 + \ldots + x_n = k, x_i \geq 0 (i \in \overline{1, n})$$

Why? It's easy to prove. Consider a vector (1, 1, …, 1) consisting of (**n+k-1**) ones, in which we want to substitute **n-1** ones for zeroes in such way that we'll get **n** groups of ones (some of which may be empty) and the number of ones in the i^{th} group will be the value of x_i:

$$(\underbrace{1,\ldots,1}_{x_1},0,\underbrace{1,\ldots,1}_{x_2},0,\ldots,0,\underbrace{1,\ldots,1}_{x_n})$$

The sum of x_i will be **k**, because **k** ones are left after substitution.

The Basics

Binary vectors

Some problems, and challenge problems are no exception, can be reformulated in terms of binary vectors. Accordingly, some knowledge of the basic combinatorial properties of binary vectors is rather important. Let's have a look at some simple things associated with them:

1. Number of binary vectors of length n: 2^n.

2. Number of binary vectors of length n and with k '1' is

$\binom{n}{k}$.

We just choose k positions for our '1's.

3. The number of ordered pairs (a, b) of binary vectors, such that the distance between them (k) can be calculated as follows: $\binom{n}{k} \cdot 2^n$.

The distance between a and b is the number of components that differs in a and b — for example, the distance between (0, 0, 1, 0) and (1, 0, 1, 1) is 2).

Let a = $(a_1, a_2, \ldots a_n)$, b = $(b_1, b_2, \ldots b_n)$ and distance between them is k. Next, let's look at the sequence of pairs $(a_1, b_1), (a_2, b_2), \ldots (a_n, b_n)$. There are exactly k indices i in which $a_i \neq b_i$. They can be (0,1) or (1,0), so there are 2 variants, and n-k can be either (0,0) or (1,1), for another 2 variants. To calculate the answer we can choose k indices in which vectors differs in $\binom{n}{k}$ ways, then we choose components that differs in 2^k ways and components that are equal in 2^{n-k} ways (all of which use the permutation with repetition formula), and in the end we just multiply all these numbers and get $\binom{n}{k} \cdot 2^k \cdot 2^{n-k} = \binom{n}{k} \cdot 2^n$.

Delving deeper

Now let's take a look at a very interesting and useful formula called the inclusion-exclusion principle (also known as the sieve principle):

$$\left|\bigcup_{i=1}^{n} A_i\right| = \sum_{i=1}^{n}(-1)^{i-1}\sum_{\substack{I\subseteq 1,n,\\ |I|=i}}\left|\bigcap_{j\in I} A_j\right|$$

This formula is a generalization of:

$$|A\cup B| = |A| + |B| - |A\cap B|$$

There are many different problems that can be solved using the sieve principle, so let's focus our attention on one of them. This problem is best known as "Derangements". A derangement of the finite set X is a bijection from X into X that doesn't have fixed points. A small example: For set X = {1, 2, 3} bijection {(1,1), (2,3), (3,2)} is not derangement, because of (1,1), but bijection {(1,2), (2,3), (3,1)} is derangement. So let's turn back to the problem, the goal of which is to find the number of derangements of n-element set.

We have X = {1, 2, 3,…, n}. Let:

> A be the set of all bijections from X into X, $|A|=n!$,
>
> A_0 be the set of all derangements of X,
>
> A_i ($i \in X$) be the set of bijections from X into X that have (i,i),
>
> A_I ($I \subseteq X$) be the set of bijections from X into X that have (i,i) $\forall i \subseteq I$, so $A_I = \bigcap_{i\in I} A_i$ and $|A_I|=(n-|A_I|)!$.

In formula we have sums $\sum_{\substack{I\subseteq 1,n,\\ |I|=i}}\left|\bigcap_{j\in I} A_j\right|$, in our case we'll have $\sum_{\substack{I\subseteq 1,n,\\ |I|=i}}\left|\bigcap_{j\in I} A_j\right| = \sum_{\substack{I\subseteq X,\\ |I|=i}}|A_I|$, so let's calculate them:

$$\sum_{\substack{I\subseteq X,\\ |I|=i}}|A_I| = \binom{n}{i}(n-i)! = \frac{n!}{i!}$$

(because there are exactly $\binom{n}{i}$ i-element subsets of X).

Now we just put that result into the sieve principle's formula:

$$\left|\bigcup_{i=1}^{n} A_i\right| = \sum_{i=1}^{n}(-1)^{i-1}\sum_{\substack{I\subseteq X,\\ |I|=i}}|A_I| = \sum_{i=1}^{n}(-1)^{i-1}\frac{n!}{i!} = n!\sum_{i=1}^{n}\frac{(-1)^{i-1}}{i!}$$

And now the last step, from $A_0 = A\setminus\bigcup_{i=1}^{n} A_i$ we'll have the answer:

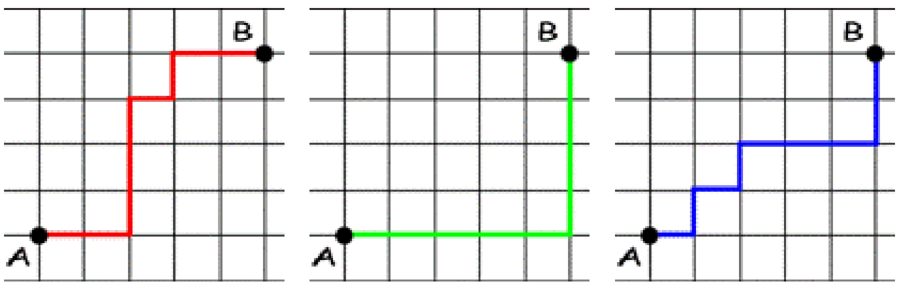
$$|A_0| = |A| - \left|\bigcup_{i=1}^{n} A_i\right| = n! - n!\sum_{i=1}^{n}\frac{(-1)^{i-1}}{i!} = n!\sum_{i=0}^{n}\frac{(-1)^{i}}{i!}.$$

And the last remark:

$$n!\sum_{i=0}^{n}\frac{(-1)^{i}}{i!} = \left[\frac{n!}{e}\right].$$

This problem may not look very practical for use in TopCoder problems, but the thinking behind it is rather important, and these ideas can be widely applied.

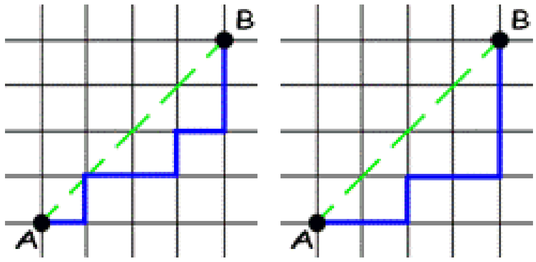
Another interesting method in combinatorics — and one of my favorites, because of its elegance — is called method of paths (or trajectories). The main idea is to find a geometrical interpretation for the problem in which we should calculate the number of paths of a special type. More precisely, if we have two points A, B on a plane with integer coordinates, then we will operate only with the shortest paths between A and B that pass only through the lines of the integer grid and that can be done only in horizontal or vertical movements with length equal to 1. For example:



All paths between A and B have the same length equal to n+m (where n is the difference between x-coordinates and m is the difference between y-coordinates). We can easily calculate the number of all the paths between A and B as follows:

$$\binom{n+m}{m} \text{ or } \binom{n+m}{n}.$$

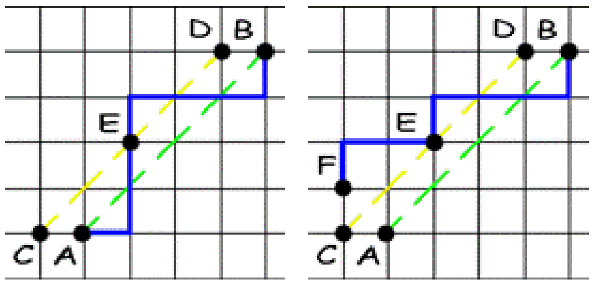
Let's solve a famous problem using this method. The goal is to find the number of Dyck words with a length of 2n. What is a Dyck word? It's a string consisting only of n X's and n Y's, and matching this criteria: each prefix of this string has more X's than Y's. For example, "XXYY" and "XYXY" are Dyck words, but "XYYX" and "YYXX" are not.



Let's start the calculation process. We are going to build a geometrical analog of this problem, so let's consider paths that go from point A(0, 0) to point B(n, n) and do not cross segment AB, but can touch it (see examples for n=4).

It is obvious that these two problems are equivalent; we can just build a bijection in such a way: step right – 'X', step up – 'Y'.

Here's the main idea of the solution: Find the number of paths from A to B that cross segment AB, and call them "incorrect". If path is "incorrect" it has points on the segment CD, where C = (0, 1), D = (n-1, n). Let E be the point nearest to A that belongs to CD (and to the path). Let's symmetrize AE, part of our "incorrect" path with respect to the line CD. After this operation we'll get a path from F = (-1, 1) to B.



It should be easy to see that, for each path from F to B, we can build only one "incorrect" path from A to B, so we've got a bijection. Thus, the number of "incorrect" paths from A to B is $\binom{2n}{n-1}$. Now we can easily get the answer, by subtracting the number of "incorrect" paths from all paths:

$$\binom{2n}{n}-\binom{2n}{n-1}=\frac{(2n)!}{n!\cdot n!}-\frac{(2n)!}{(n-1)!\cdot(n+1)!}=\frac{\binom{2n}{n}}{n+1}$$

This number is also known as n's Catalan number: C_n is the number of Dyck words with length 2n. These numbers also appear in many other problems, for example, C_n counts the number of correct arrangements of n pairs of parentheses in the expression, and C_n is also the number of the possible triangulations of a polygon with (n+2) vertices, and so on.

Using recurrence relations

Recurrence relations probably deserves their own separate article, but I should mention that they play a great role in combinatorics. Particularly with regard to TopCoder, most calculation problems seem to require coders to use some recurrence relation and find the solution for the values of parameters.

If you'd like to learn more, check out these tutorials: An Introduction to Recursion, Recursion, Part 2, and

Dynamic Programming: From novice to advanced. Done reading? Let's take a look at some examples.

ChristmasTree (SRM 331 Division Two – Level Three):

We'll use DP to solve this — it may not be the best way to tackle this problem, but it's the easiest to understand. Let **cnt[lev][r][g][b]** be the number of possible ways to decorate the first **lev** levels of tree using **r** red, **g** green and **b** blue baubles. To make a recurrent step calculating **cnt[lev][r][g][b]** we consider 3 variants:

cnt[lev][r][g][b]=

1) we fill the last level with one color (red, green or blue), so just:

= cnt [lev-1][r-lev][g][b]+ cnt[lev-1][r][g-lev][b]+ cnt[lev-1][r][g][b-lev]+ ;

2) if (lev%2 == 0) we fill the last level with two colors (red+green, green+blue or red+blue), then we calculate the number of possible decorations using the *Permutation with repetition* formula. We'll get

$\frac{lev!}{(lev/2)!(lev/2)!}$ possible variants for each two colors, so just

$\frac{lev!}{(lev/2)!(lev/2)!}$ (cnt[lev-1][r-lev/2][g-lev/2][b]+…+cnt[lev-1][r][g-lev/2][b-lev/2])+;

3) if (lev%3 == 0) we fill the last level with three colors and, again using the *Permutation with repetition* formula, we'll get $\frac{lev!}{(lev/3)!(lev/3)!(lev/3)!}$ possible variants, so we'll get:

$$+\frac{lev!}{(lev/3)!(lev/3)!(lev/3)!}\cdot \mathrm{cnt[lev\text{-}1][r\text{-}lev/3][g\text{-}lev/3][b\text{-}lev/3]}$$

(all cnt[l][i][j][k] with negative indices are 0).

DiceGames (SRM 349 Division One – Level Two):

First we should do some preparation for the main part of the solution, by sorting **sides** array in increasing order and calculating only the formations where the numbers on the dice go in non-decreasing order. This preparation saves us from calculating the same formations several times (see SRM 349 – Problem Set & Analysis for additional explanation). Now we will only need to build the recurrence relation, since the implementation is rather straightforward. Let **ret[i][j]** be the number of different formations of the first *i* dice, with the last dice equal to *j* (so $i \in \overline{0, n-1},\ j \in \overline{0, sides[i]-1}$, where n is the number of elements in **sides**). Now we can simply write the recurrence relation:

$$ret[i][j] = \sum_{k=0}^{j} ret[i-1][k],\ ret[0][i] = 1 (i \in \overline{0, sides[0]-1})$$

The answer will be $\sum_{i=0}^{sides[n-1]-1} ret[n-1][i]$.

Conclusion

As this article was written for novices in combinatorics, it focused mainly on the basic aspects and methods of counting. To learn more, I recommend you check out the reference works listed below, and keep practicing – both in TopCoder SRMs and pure mathematical problems. Good luck!

References:

1. Hall M. “Combinatorial theory”
2. Cameron P.J. “Combinatorics: Topics, Techniques, Algorithms”
3. en.wikipedia.org :-)

More Resources

Member Tutorials

Read more than 40 data science tutorials written by topcoder members.

Problem Set Analysis

Read editorials explaining the problem and solution for each Single Round Match (SRM).

Data Science Guide

New to topcoder's data science track? Read this guide for an overview on how to get started in the arena and how competitions work.

Help Center

Need specifics about the process or the rules? Everything you need to know about competing at topcoder can be found in the Help Center.

Member Forums

Join your peers in our member forums and ask questions from the real experts - topcoder members!

Understanding Probabilities

By **supernova** — *topcoder member*
Discuss this article in the forums

It has been said that life is a school of probability. A major effect of probability theory on everyday life is in risk assessment. Let's suppose you have an exam and you are not so well prepared. There are 20 possible subjects, but you only had time to prepare for 15. If two subjects are given, what chances do you have to be familiar with both? This is an example of a simple question inspired by the world in which we live today. Life is a very complex chain of **events** and almost everything can be imagined in terms of probabilities.

Gambling has become part of our lives and it is an area in which probability theory is obviously involved. Although gambling had existed since time immemorial, it was not until the seventeenth century that the mathematical foundations finally became established. It all started with a simple question directed to Blaise Pascal by Chevalier de Mï¿½rï¿½, a nobleman that gambled frequently to increase his wealth. The question was whether a double six could be obtained on twenty-four rolls of two dice.

As far as topcoder problems are concerned, they're inspired by reality. You are presented with many situations, and you are explained the rules of many games. While it's easy to recognize a problem that deals with probability computations, the solution may not be obvious at all. This is partly because probabilities are often overlooked for not being a common theme in programming challenges. But it is not true and topcoder has plenty of them! Knowing how to approach such problems is a big advantage in topcoder competitions and this article is to help you prepare for this topic.

Before applying the necessary algorithms to solve these problems, you first need some mathematical understanding. The next chapter presents the basic principles of probability. If you already have some experience in this area, you might want to skip this part and go to the following chapter: Step by Step Probability Computation. After that it follows a short discussion on Randomized Algorithms and in the end there is a list with the available problems on topcoder. This last part is probably the most important. Practice is the key!

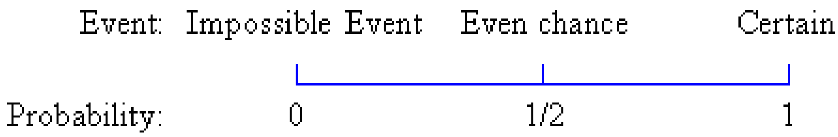
Basics

Working with probabilities is much like conducting an experiment. An **outcome** is the result of an experiment or other situation involving uncertainty. The set of all possible outcomes of a probability experiment is called a **sample space**. Each possible result of such a study is represented by one and only one point in the sample space, which is usually denoted by S. Let's consider the following experiments:

> Rolling a die once
>   Sample space S = {1, 2, 3, 4, 5, 6}
> Tossing two coins
>   Sample space S = {(Heads, Heads), (Heads, Tails), (Tails, Heads), (Tails, Tails)}

We define an **event** as any collection of outcomes of an experiment. Thus, an event is a subset of the sample space S. If we denote an event by **E**, we could say that **E⊆S**. If an event consists of a single outcome in the sample space, it is called a simple event. Events which consist of more than one outcome are called compound events.

What we are actually interested in is the probability of a certain event to occur, or **P(E)**. By definition, **P(E)** is a real number between 0 and 1, where 0 denotes the impossible event and 1 denotes the certain event (or the whole sample space).



As stated earlier, each possible outcome is represented by exactly one point in the sample space. This leads us to the following formula:

$$P(E) = \frac{|E|}{|S|}$$

That is, the probability of an event to occur is calculated by dividing the number of **favorable outcomes** (according to the event E) by the **total number of outcomes** (according to the sample space S). In order to represent the relationships among events, you can apply the known principles of set theory. Consider the experiment of rolling a die once. As we have seen previously, the sample space is S = {1, 2, 3, 4, 5, 6}. Let's now consider the following events:

> Event A = 'score > 3' = {4, 5, 6}
> Event B = 'score is odd' = {1, 3, 5}
> Event C = 'score is 7' = ∅
> A∪B ='the score is > 3 or odd or both' = {1, 3, 4, 5, 6}
> A∩B ='the score is > 3 and odd' = {5}
> A' = 'event A does not occur' = {1, 2, 3}

We have:

P(A∪B) = 5/6
P(A∩B) = 1/6
P(A') = 1 – P(A) = 1 – 1/2 = 1/2
P(C) = 0

The first step when trying to solve a probability problem is to be able to recognize the sample space. After that, you basically have to determine the number of favorable outcomes. This is the classical approach, but the way we implement it may vary from problem to problem. Let's take a look at QuizShow (SRM 223, Div 1 – Easy). The key to solving this problem is to take into account all the possibilities, which are not too many. After a short analysis, we determine the sample space to be the following:

S = { (**wager 1** is wrong, **wager 2** is wrong, **you** are wrong),
(**wager 1** is wrong, **wager 2** is wrong, **you** are right),
(**wager 1** is wrong, **wager 2** is right, **you** are wrong),
(**wager 1** is wrong, **wager 2** is right, **you** are right),
(**wager 1** is right, **wager 2** is wrong, **you** are wrong),
(**wager 1** is right, **wager 2** is wrong, **you** are right),
(**wager 1** is right, **wager 2** is right, **you** are wrong),
(**wager 1** is right, **wager 2** is right, **you** are right) }

The problem asks you to find a wager that maximizes the number of favorable outcomes. In order to compute the number of favorable outcomes for a certain wager, we need to determine how many points the three players end with for each of the 8 possible outcomes. The idea is illustrated in the following program:

```
int wager (vector  scores, int wager1, int wager2)
{
 int best, bet, odds, wage, I, J, K;
 best = 0; bet = 0;

 for (wage = 0; wage ≤ scores[0]; wage++)
 {
  odds = 0;
  //  in 'odds' we keep the number of favorable outcomes
  for (I = -1; I ≤ 1; I = I + 2)
   for (J = -1; J ≤ 1; J = J + 2)
    for (K = -1; K ≤ 1; K = K + 2)
     if (scores[0] + I * wage > scores[1] + J * wager1  &&
      scores[0] + I * wage > scores[2] + K * wager2)   { odds++; }
```

```
  if (odds > best)  { bet = wage ; best = odds; }
  //  a better wager has been found
 }
 return bet;
}
```

Another good problem to start with is PipeCuts (SRM 233, Div 1 – Easy). This can be solved in a similar manner. There is a finite number of outcomes and all you need to do is to consider them one by one.

Let's now consider a series of n independent events: E1, E2, … , En. Two surprisingly common questions that may appear (and many of you have already encountered) are the following:

1. What is the probability that all events will occur?

2. What is the probability that at least one event will occur?

To answer the first question, we relate to the occurrence of the first event (call it E1). If E1 does not occur, the hypothesis can no longer be fulfilled. Thus, it must be inferred that E1 occurs with a probability of P(E1). This means there is a P(E1) chance we need to check for the occurrence of the next event (call it E2). The event E2 occurs with a probability of P(E2) and we can continue this process in the same manner. Because probability is by definition a real number between 0 and 1, we can synthesize the probability that all events will occur in the following formula:

$$P(\text{all events}) = P(E1) * P(E2) * \dots * P(En)$$

The best way to answer the second question is to first determine the probability that no event will occur and then, take the complement. We have:

$$P(\text{at least one event}) = 1 - P(E1') * P(E2') * \dots * P(En')$$

These formulae are very useful and you should try to understand them well before you move.

BirthdayOdds

A good example to illustrate the probability concepts discussed earlier is the classical "Birthday Paradox". It has been shown that if there are at least 23 people in a room, there is a more than 50% chance that at least two of them will share the same birthday. While this is not a paradox in the real sense of the word, it is a mathematical truth that contradicts common intuition. The topcoder problem asks you to find the minimum number of people in order to be more than minOdds% sure that at least two of them have the same birthday. One of the first things to notice about this problem is that it is much easier to solve the complementary problem: "What is the probability that N randomly selected people have all different birthdays?". The strategy is to start with an empty room and put people in the room one by one, comparing their birthdays with those of them already in the room:

```
int minPeople (int minOdds, int days)
{
 int nr;
 double target, p;

 target = 1 - (double) minOdds / 100;
 nr = 1;
 p = 1;

 while (p > target)
 {
  p = p * ( (double) 1 - (double) nr / days);
  nr ++;
 }

 return nr;
}
```

This so called "Birthday Paradox” has many real world applications and one of them is described in the topcoder problem called Collision (SRM 153, Div 1 – Medium). The algorithm is practically the same, but one has to be careful about the events that may alter the sample space.

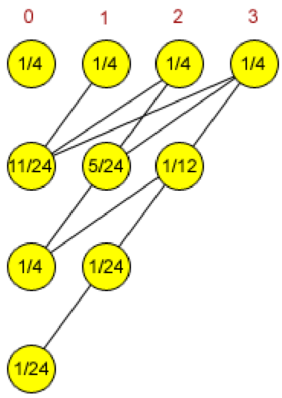
Sometimes a probability problem can be quite tricky. As we have seen before, the ‘Birthday Paradox’ tends to contradict our common sense. But the formulas prove to us that the answer is indeed correct. Formulas can help, but to become a master of probabilities you need one more ingredient: "number sense" . This is partly innate ability and partly learned ability acquired through practice. Take this quiz to assess your number sense and to also become familiar with some of the common probability misconceptions.

Step by Step Probability Computation

In this chapter we will discuss some real topcoder problems in which the occurrence of an event is influenced by occurrences of previous events. We can think of it as a graph in which the nodes are events and the edges are dependencies between them. This is a somewhat forced analogy, but the way we compute the probabilities for different events is similar to the way we traverse the nodes of a graph. We start from the root, which is the initial state and has a probability of 1. Then, as we consider different scenarios, the probability is distributed accordingly.

NestedRandomness

This problem looked daunting to some people, but for those who figured it out, it was just a matter of a few lines. For the first step, it is clear what do we have to do: the function random(N) is called and it returns a random integer uniformly distributed in the range 0 to N-1. Thus, every integer in this interval has a probability of 1/N to occur. If we consider all these outcomes as input for the next step, we can determine all the outcomes of the random(random(N)) call. To understand this better, let's work out the case when N = 4.



NestedRandomness for N = 4

After the **first nesting** all integers have the same probability to occur, which is 1 / 4.

For the **second nesting** there is a 1/4 chance for each of the following functions to be called: random(0), random(1), random(2) and random(3). Random(0) produces an error, random(1) returns 0, random (2) returns 0 or 1 (each with a probability of 1/2) and random(3) returns 0, 1 or 2.

As a result, for the **third nesting**, random(0) has a probability of 1/4 + 1/8 + 1/12 of being called, random(1) has a probability of 1/8 + 1/12 of being called and random(2) has a probability of 1/12 of being called.

Analogously, for the **fourth nesting**, the function random(0) has a probability of 1/4 of being called, while random(1) has a probability of 1/24.

As for the **fifth nesting**, we can only call random(0), which produces an error. The whole process is described in the picture to the right.

The source code for this problem is given below:

```
double probability (int N, int nestings, int target)
{
 int I, J, K;
 double A[1001], B[2001];
```

```
// A[I] represents the probability of number I  to appear

for (I = 0; I < N ; I++)  A[I] = (double) 1 / N;
for (K = 2; K ≤ nestings; K++)
{
 for (I = 0; I < N; I++)  B[I] = 0;
 // for each I between 0 and N-1 we call the function "random(I)"
 // as described in the problem statement
 for (I = 0; I < N; I++)
  for (J = 0; J < I; J++)
   B[J] +=  (double) A[I] / I;
   for (I = 0; I < N; I++)  A[I] = B[I];
}
 return A[target];
}
```

If you got the taste for this problem, here are another five you may want to try:

ChessKnight – assign each square a probability and for every move check the squares one by one to compute the probabilities for the next move.
DiceThrows – determine the probability of each possible outcome for both players and then compare the results.
RockSkipping – the same approach, just make sure you got the lake pattern correctly.
PointSystem – represent the event space as a matrix of possible scores (x, y).
VolleyBall – similar to PointSystem, but the scores may go up pretty high.

Let's now take a look at another topcoder problem, GeneticCrossover, which deals with **conditional probability**. Here, you are asked to predict the quality of an animal, based on the genes it inherits from its parents. Considering the problem description, there are two situations that may occur: a gene does not depend on another gene, or a gene is dependent.

For the first case, consider p the probability that the gene is to be expressed dominantly. There are only 4 cases to consider:

> at least one parent has two dominant genes. (**p** = 1)
>
> each parent has exactly one dominant gene. (**p** = 0.5)
>
> one parent has one dominant gene and the other has only recessive genes (**p** = 0.25)

both parents have two recessive genes (**p** = 0)

Now let's take the case when a gene is dependent on another. This make things a bit trickier as the "parent" gene may also depend on another and so on … To determine the probability that a dependent gene is dominant, we take the events that each gene in the chain (starting with the current gene) is dominant. In order for the current gene to be expressed dominantly, we need all these events to occur. To do this, we take the product of probabilities for each individual event in the chain. The algorithm works recursively. Here is the complete source code for this problem:

```
int n, d[200];
double power[200];

// here we determine the characteristic for each gene (in power[I]
// we keep the probability of gene I to be expressed dominantly)
double detchr (string p1a, string p1b, string p2a, string p2b, int nr)
{
 double p, p1, p2;
 p = p1 = p2 = 1.0;
 if (p1a[nr] ≤ 'Z')  p1 = p1 - 0.5;
 //  is a dominant gene
 if (p1b[nr] ≤ 'Z')  p1 = p1 - 0.5;
 if (p2a[nr] ≤ 'Z')  p2 = p2 - 0.5;
 if (p2b[nr] ≤ 'Z')  p2 = p2 - 0.5;
 p = 1 - p1 * p2;

 if (d[nr] != 1) power[nr] = p * detchr (p1a, p1b, p2a, p2b, d[nr]);
 // gene 'nr' is dependent on gene d[nr]
    else power[nr] = p;
 return power[nr];
}

double cross (string p1a, string p1b, string p2a, string p2b,
 vector  dom, vector  rec, vector  dependencies)
{
  int I;
 double fitness = 0.0;

 n = rec.size();
 for (I = 0; I < n; i++) d[i] = dependencies[i];
 for (I = 0 ;I < n; I++) power[i] = -1.0;
```

```
 for (I = 0; I < n; i++)
  if (power[I] == -1.0) detchr (p1a, p1b, p2a, p2b, i);
  // we check if the dominant character of gene I has
  // not already been computed
 for (I = 0; I ≤ n; I++)
  fitness=fitness+(double) power[i]*dom[i]-(double) (1-power[i])*rec[i];
  // we compute the expected 'quality' of an animal based on the
  // probabilities of each gene to be expressed dominantly

 return fitness;
}
```

See also ProbabilityTree.

Randomized Algorithms

We call randomized algorithms those algorithms that use random numbers to make decisions during their execution. Unlike deterministic algorithms that for a fixed input always give the same output and the same running-time, a randomized algorithm behaves differently from execution to execution. Basically, we distinguish two kind of randomized algorithms:

1. Monte Carlo algorithms: may sometimes produce an incorrect solution – we bound the probability of failure.

2. Las Vegas algorithms: always give the correct solution, the only variation is the running time – we study the distribution of the running time.

Read these lecture notes from the College of Engineering at UIUC for an example of how these algorithms work.

The main goal of randomized algorithms is to build faster, and perhaps simpler solutions. Being able to tackle "harder" problems is also a benefit of randomized algorithms. As a result, these algorithms have become a research topic of major interest and have already been utilized to more easily solve many different problems.

An interesting question is whether such an algorithm may become useful in topcoder competitions. Some problems have many possible solutions, where a number of which are also optimal. The classical approach is to check them one by one, in an established order. But it cannot be guaranteed that the optima are uniformly distributed in the solution domain. Thus, a deterministic algorithm may not find you an optimum quickly enough. The advantage of a randomized algorithm is that there are actually no rules to set about the order in which the solutions are checked and for the cases when the optima are clustered together, it usually performs much better. See QueenInterference for a topcoder example.

Randomized algorithms are particularly useful when faced with malicious attackers who deliberately try to feed a bad input to the algorithm. Such algorithms are widely used in cryptography, but it sometimes makes sense to also use them in topcoder competitions. It may happen that you have an efficient algorithm, but there are a few degenerate cases for which its running time is significantly slower. Assuming the algorithm is correct, it has to run fast enough for all inputs. Otherwise, all the points you earned for submitting that particular problem are lost. This is why here, on topcoder, we are interested in **worst case execution time**.

To challenge or not to challenge?

Another fierce coding challenge is now over and you have 15 minutes to look for other coders' bugs. The random call in a competitor's submission is likely to draw your attention. This will most likely fall into one of two scenarios:

1. the submission was just a desperate attempt and will most likely fail on many inputs.
2. the algorithm was tested rather thoroughly and the probability to fail (or time out) is virtually null.

The first thing you have to do is to ensure it was not already unsuccessfully challenged (check the coder's history). If it wasn't, it may deserve a closer look. Otherwise, you should ensure that you understand what's going on before even considering a challenge. Also take into account other factors such as coder rating, coder submission accuracy, submission time, number of resubmissions or impact on your ranking.

Will "random" really work?

In most optimizing problems, the ratio between the number of optimal solutions and the total number of solutions is not so obvious. An easy, but not so clever solution, is to simply try generating different samples and see how the algorithm behaves. Running such a simulation is usually pretty quick and may also give you some extra clues in how to actually solve the problem.

```
Max = 1000000; attempt = 0;
while (attempt < Max)
{
 answer = solve_random (...);
 if (better (answer, optimum))
 // we found a better solution
 {
  optimum = answer;
  cout << "Solution " << answer << " found on step " << attempt << "\n";
 }
```

```
  attempt ++;
}
```

Practice Problems

Level 1

PipeCuts – SRM 233

BirthdayOdds – SRM 174

BenfordsLaw – SRM 155

QuizShow – SRM 223

Level 2

Collision – SRM 153

ChessKnight – TCCC05 Round 1

ChipRace – SRM 199

DiceThrows – SRM 242

TopFive – SRM 243

ProbabilityTree – SRM 174

OneArmedBandit – SRM 226

RangeGame – SRM 174

YahtzeeRoll – SRM 222

BagOfDevouring – SRM 184

VolleyBall – TCO04 Round 3

RandomFA – SRM 178

PackageShipping – TCCC05 Round 3

QueenInterference – SRM 208

BaseballLineup – TCO '03 Finals

Level 3

GeneticCrossover – TCO04 Qual 3

NestedRandomness – TCCC05 Qual 5

RockSkipping – TCCC '04 Round 1

PointSystem – SRM 174

AntiMatter – SRM 179

TestScores – SRM 226

Hangman42 – SRM 229

KingOfTheCourt – SRM 222

WinningProbability – SRM 218

Disaster – TCCC05 Semi 1

Algorithm Games

By **rasto6sk**– *TopCoder Member*
Discuss this article in the forums

Introduction
The games we will talk about are two-person games with perfect information, no chance moves, and a win-or-lose outcome. In these games, players usually alternate moves until they reach a terminal position. After that, one player is declared the winner and the other the loser. Most card games don't fit this category, for example, because we do not have information about what cards our opponent has.

First we will look at the basic division of positions to winning and losing. After that we will master the most important game — the Game of Nim — and see how understanding it will help us to play composite games. We will not be able to play many of the games without decomposing them to smaller parts (sub-games), pre-computing some values for them, and then obtaining the result by combining these values.

The Basics
A simple example is the following game, played by two players who take turns moving. At the beginning there are n coins. When it is a player's turn he can take away 1, 3 or 4 coins. The player who takes the last one away is declared the winner (in other words, the player who can not make a move is the loser). The question is: For what n will the first player win if they both play optimally?

We can see that n = 1, 3, 4 are winning positions for the first player, because he can simply take all the coins. For n=0 there are no possible moves — the game is finished — so it is the losing position for the first player, because he can not make a move from it. If n=2 the first player has only one option, to remove 1 coin. If n=5 or 6 a player can move to 2 (by removing 3 or 4 coins), and he is in a winning position. If n=7 a player can move only to 3, 4, 6, but from all of them his opponent can winâ€¦

Positions have the following properties:

All terminal positions are losing.

If a player is able to move to a losing position then he is in a winning position.

If a player is able to move only to the winning positions then he is in a losing position.

These properties could be used to create a simple recursive algorithm **WL-Algorithm**:

```
boolean isWinning(position pos) {
    moves[] = possible positions to which I can move from the
position pos;
    for (all x in moves)
        if (!isWinning(x)) return true;

    return false;
}
```

Table 1: Game with 11 coins and subtraction set {1, 3, 4}:

n	0	1	2	3	4	5	6	7	8	9	10	11
position	L	W	L	W	W	W	W	L	W	L	W	W

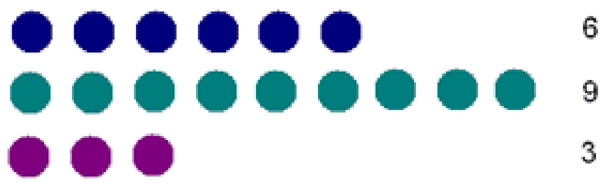
This game could be played also with a rule (usually called the misere play rule) that the player who takes away the last coin is declared the loser. You need to change only the behavior for the terminal positions in WL-Algorithm. Table 1 will change to this:

n	0	1	2	3	4	5	6	7	8	9	10	11
position	W	L	W	L	W	W	W	W	L	W	L	W

It can be seen that whether a position is winning or losing depends only on the last k positions, where k is the maximum number of coins we can take away. While there are only 2^k possible values for the sequences of the length k, our sequence will become periodic. You can try to use this observation to solve the following problem:
SRM 330: LongLongNim

The Game of Nim
The most famous mathematical game is probably the Game of Nim. This is the game that you will probably encounter the most times and there are many variations on it, as well as games that can be solved by using the knowledge of how to play the game. Usually you will meet them as Division I 1000 pointers (though hopefully your next encounter will seem much easier). Although these problems often require a clever idea, they are usually very easy to code.

Rules of the Game of Nim: There are n piles of coins. When it is a player’s turn he chooses one pile and takes at least one coin from it. If someone is unable to move he loses (so the one who removes the last coin is the winner).



Let n1, n2, â€¦ nk, be the sizes of the piles. It is a losing position for the player whose turn it is if and only if **n_1 xor n_2 xor .. xor n_k = 0**.

How is xor being computed?

```
6 = (110)2          1 1 0
9 = (1001)2       1 0 0 1
3 = (11)2             1 1
                 --------
                 1 1 0 0
```

xor of two logic values is true if and only if one of them is true and the second is false

when computing **xor** of integers, first write them as binary numbers and then apply **xor** on columns.

so **xor** of even number of 1s is 0 and **xor** of odd number of 1s is 1

Why does it work?

From the losing positions we can move only to the winning ones:
– if xor of the sizes of the piles is 0 then it will be changed after our move (at least one 1 will be changed to 0, so in that column will be odd number of 1s).

From the winning positions it is possible to move to at least one losing:
– if xor of the sizes of the piles is not 0 we can change it to 0 by finding the left most column where the number of 1s is odd, changing one of them to 0 and then by changing 0s or 1s on the right side of it to gain even number of 1s in every column.

Examples:
Position (1, 2, 3) is losing because 1 xor 2 xor 3 = $(1)_2$ xor $(10)_2$ xor $(11)_2$ = 0
Position (7, 4, 1) is winning because 7 xor 4 xor 1 = $(111)_2$ xor $(10)_2$ xor $(1)_2$ = $(10)_2$ = 2

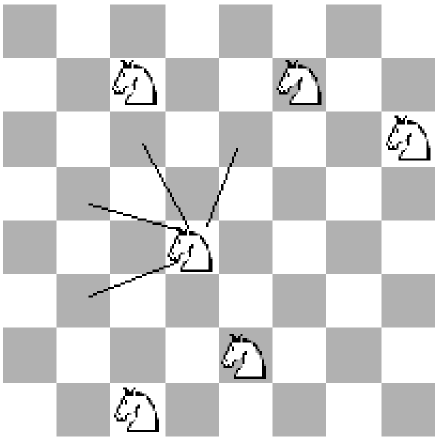
Example problems:
SRM 338: CakeParty

SRM 309: StoneGameStrategist

The last one example problem is harder, because it is not so easy to identify where the sizes of piles are hidden. Small hint: Notice the differences between the sizes of piles. If you would not be able to figure it out you can find the solution in the SRM 309 Problem set & Analysis.

Composite games – Grundy numbers

Example game: N x N chessboard with K knights on it. Unlike a knight in a traditional game of chess, these can move only as shown in the picture below (so the sum of coordinates is decreased in every move). There can be more than one knight on the same square at the same time. Two players take turns moving and, when it is a player's, turn he chooses one of the knights and moves it. A player who is not able to make a move is declared the loser.



This is the same as if we had K chessboards with exactly one knight on every chessboard. This is the ordinary sum of K games and it can be solved by using the grundy numbers. We assign grundy number to every subgame according to which size of the pile in the Game of Nim it is equivalent to. When we know how to play Nim we will be able to play this game as well.

```
int grundyNumber(position pos) {
    moves[] = possible positions to which I can move from pos
    set s;
    for (all x in moves) insert into s grundyNumber(x);
    //return the smallest non-negative integer not in the set s;
    int ret=0;
    while (s.contains(ret)) ret++;
    return ret;
}
```

The following table shows grundy numbers for an 8 x 8 board:

0	0	1	1	0	0	1	1
0	0	2	1	0	0	1	1
1	2	2	2	3	2	2	2
1	1	2	1	4	3	2	3
0	0	3	4	0	0	1	1
0	0	2	3	0	0	2	1
1	1	2	2	1	2	2	2
1	1	2	3	1	1	2	0

We could try to solve the original problem with our WL-Algorithm, but it would time out because of the large number of possible positions.

A better approach is to compute grundy numbers for an N x N chessboard in $O(n^2)$ time and then xor these K (one for every horse) values. If their xor is 0 then we are in a losing position, otherwise we are in a winning position.

Why is the pile of Nim equivalent to the subgame if its size is equal to the grundy number of that subgame?

> If we decrease the size of the pile in Nim from A to B, we can move also in the subgame to the position with the grundy number B. (Our current position had grundy number A so it means we could move to positions with all smaller grundy numbers, otherwise the grundy number of our position would not be A.)
>
> If we are in the subgame at a position with a grundy number higher than 0, by moving in it and decreasing its grundy number we can also decrease the size of pile in the Nim.
>
> If we are in the subgame at the position with grundy number 0, by moving from that we will get to a position with a grundy number higher than 0. Because of that, from such a position it is possible to move back to 0. By doing that we can nullify every move from the position from grundy number 0.

Example problems:
SRM 216: Roxor

Other composite games
It doesn't happen often, but you can occasionally encounter games with a slightly different set of rules. For example, you might see the following changes:

1. When it is a player's move he can choose some of the horses (at least one) and move with all the chosen ones.

Solution: You are in a losing position if and only if every horse is in a losing position on his own chessboard (so the grundy number for every square, where the horse is, is 0).

2. When it is a player's move he can choose some of the horses (at least one), but not all of them, and move with all chosen ones.

Solution: You are in a losing position if and only if the grundy numbers of all the positions, where horses are, are the same.

You can verify correctness of both solutions by verifying the basic properties (from a winning position it is possible to move to a losing one and from a losing position it is possible to move only to the winning ones). Of course, everything works for all other composite games with these rules (not only for horse games).

Homework: What would be changed if a player had to move with every horse and would lose if he were not able to do so?

Conclusion
Don't worry if you see a game problem during SRM — it might be similar to one the games described above, or it could be reduced to one of them. If not, just think about it on concrete examples. Once you figure it out the coding part is usually very simple and straightforward. Good luck and have fun.

Other resources:

Winning ways for your mathematical plays by Elwyn R. Berlekamp, John H. Conway, Richard K. Guy

More Resources

Member Tutorials
Read more than 40 data science tutorials written by topcoder members.

Problem Set Analysis
Read editorials explaining the problem and solution for each Single Round Match (SRM).

Data Science Guide
New to topcoder's data science track? Read this guide for an overview on how to get started in the arena and how competitions work.

Help Center
Need specifics about the process or the rules? Everything you need to know about competing at topcoder can be found in the Help Center.

Member Forums
Join your peers in our member forums and ask questions from the real experts - topcoder members!

ZZxq Aa¨vq

R¨wgwZ

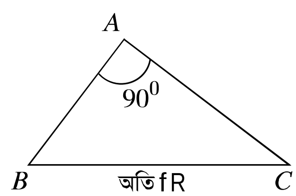
wbggva¨wgK I gva¨wgK chv‡qi R¨wgwZ‡Z cx_v‡Mviv‡mi Dccv`¨ I Gi wecixZ Dccv`¨ wb‡q we¯wiZ Av‡jvPbv Kiv n‡q‡Q| MwYZ wk¶vq cx_v‡Mvivm msµvšÍ welqvejx AZ¨šÍ ¸i"ZcY fwgKv cvjb K‡i| ZvB gva¨wgK D"PZi MwY‡Z cx_v‡Mviv‡mi Dccv‡`¨i Av‡jv‡K AwaKZi Av‡jvPbv Avek¨K| G msµvšÍ Av‡jvPbvi Rb¨ Ôj¤ Awf‡¶cÕ m¤ú‡K m¯úó aviYv _vKv `iKvi| †m j‡¶ GB chv‡q c_g As‡k cx_v‡Mviv‡mi Dccv‡`¨i msw¶ß Av‡jvPbv, wØZxq chv‡q j¤ Awf‡¶‡ci aviYv Ges cx_v‡Mviv‡mi Dccv‡`¨i Abwm×všÍ wb‡q Av‡jvPbv Kiv n‡e| Av‡jvPbvi †kl As‡k cx_v‡Mvivm Ges Gi we¯wZi aviYvi Dci wfwË K‡i hw³gjK Av‡jvPbv I cgv‡Yi Rb¨ wKQ mgm¨v Ašf³ Kiv n‡e|

Aa¨vq †k‡l wk¶v_xiv –

- j¤ Awf‡¶‡ci aviYv e¨vL¨v Ki‡Z cvi‡e|
- cx_v‡Mviv‡mi Dccv‡`¨i Dci wfwË K‡i c`Ë Dccv`¨¸‡jv cgvY I c‡qvM Ki‡Z cvi‡e|
- wÎf‡Ri cwi‡K›`, fi‡K›` I j¤we›` m¤úwKZ Dccv`¨¸‡jv cgvY I c‡qvM Ki‡Z cvi‡e|
- eþv¸‡ßi Dccv`¨ cgvY I c‡qvM Ki‡Z cvi‡e|
- U‡jwgi Dccv`¨ cgvY I c‡qvM Ki‡Z cvi‡e|

3 (K) cx_v‡Mvivm m¤úwKZ Av‡jvPbv

wL‡ói R‡b¥i cvq 600 eQi Av‡M weL¨vZ MxK cwÛZ cx_v‡Mvivm mg‡KvYx wÎf‡Ri †¶‡Î GKwU AZ¨šÍ ¸i"ZcY Dccv`¨ (*Theorem*) eYbv K‡ib| GB Dccv`¨wU Zvi bvgvbmv‡i cx_v‡Mviv‡mi Dccv`¨ e‡j cwiwPZ| Rvbv hvq ZviI cvq 1000 eQi Av‡M wgkixq fwg RwicKvixM‡Yi GB Dccv`¨wU mg‡Ü aviYv wQj| cx_v‡Mviv‡mi Dccv`¨ wewfbfv‡e cgvY Kiv hvq| wbggva¨wgK chv‡q Gi `BwU cgvY †`Iqv Av‡Q| ZvB GLv‡b †Kv‡bv cgvY †`Iqv n‡e bv| wk¶v_xiv Gi cgvY Aek¨B wbggva¨wgK R¨wgwZ‡Z Ki‡e| GLv‡b ïagvÎ Gi eYbv I wKQ Av‡jvPbv _vK‡e|



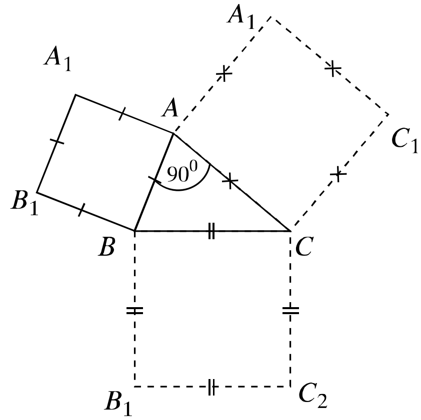
wPÎ 3.1 mg‡KvYx wÎfR

Dccv`¨ 3.1

cx_v‡Mviv‡mi Dccv`¨ :

GKwU mg‡KvYx wÎf‡Ri AwZf‡Ri Dci Aw¼Z eM‡¶‡Îi †¶Îdj Aci `B evûi Dci Aw¼Z eM‡¶ÎØ‡qi †¶Îd‡ji mgwói mgvb|

wPÎ 3·2 Gi ABC wÎfRwU GKwU mg‡KvYx wÎfR| $\angle BAC$ mg‡KvY Ges BC AwZfR| BC AwZf‡Ri Dci †Kv‡bv eM‡¶Î AsKb Ki‡j Zvi †h †¶Îdj n‡e mg‡KvY msjMœ evûØq AB I AC Gi Dci eM‡¶Î A¼b Ki‡j Zv‡`i †¶Î d‡ji †hvMdj Zvi mgvb n‡e|



wPÎ : 3.2

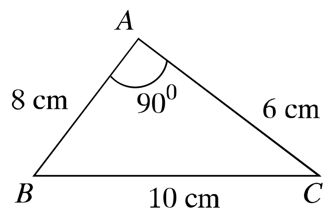
A_vr $BC^2 = AB^2 + AC^2$

GLv‡b $BC^2 = BB_1C_2C$ eM‡¶‡Îi †¶Îdj|

$AB^2 = AA_1B_1B$ Ó Ó

$AC^2 = AA_1C_1C$ Ó Ó

D`vniY ¯^i¦c, GKwU mg‡KvYx wÎf‡Ri (wPÎ : 3.3) mg‡KvY msjMœ evûØ‡qi ``N¨ h_vµ‡g 8 †m.wg I 6 †m.wg. n‡j cx_v‡Mvi‡mi Dccv‡`¨i gva¨g mn‡RB ejv hvq Gi AwZf‡Ri ``N¨ 10 †m.wg n‡e| Abi¦cfv‡e, †h‡Kv‡bv `B evûi ``‡N¨i gva¨‡g ZZxq evûi ``N¨ Rvbv m¤¢e| wb‡gi Dccv‡`¨wU cx_v‡Mvi‡mi Dccv‡`¨i wecixZ cwZÁv wnmv‡e cwiwPZ|

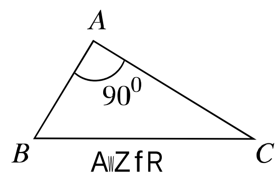


wPÎ : 3·3

Dccv`¨ 3·2

†Kv‡bv wÎf‡Ri GKwU evûi Dci A¼Z eM‡¶‡Îi †¶Îdj Aci `B evûi Dci A¼Z eM‡¶ÎØ‡qi †¶Îd‡ji mgwói mgvb n‡j †k‡lv³ evnØ‡qi Aš—f©³ †KvYwU mg‡KvY n‡e| (wPÎ : 3.4) j¶ Ki|

ΔABC Gi BC evû AwZfR Ges Aci `B evû h_vµ‡g AB I AC.



wPÎ : 3.4

BC evûi Dci A¼Z eM‡¶‡Îi †¶‡Îdj Aci `B evû h_vµ‡g AB I AC evûi Dci A¼Z eM‡¶ÎØ‡qi †¶Îd‡ji mgwói mgvb|

A_vr, $BC^2 = AB^2 + AC^2$

mZivs, $\angle BAC$ GKwU mg‡KvY|

D`vniY¯^i¦c Avgiv ej‡Z cvwi ΔABC Gi AB, BC I CA evûi ``N¨ h_vµ‡g 6 †m.wg 10 †m.wg I 8 †m.wg nq Zvn‡j $\angle BAC$ Aek¨B mg‡KvY n‡e|

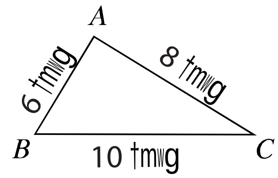
†h‡nZz, $AB^2 = 6^2$ e. †m. wg. = 36 e. †m. wg.

$BC^2 = 10^2$ e. †m. wg. = 100 e. †m. wg.

$AC^2 = 8^2$ e. †m. wg. = 64 e. †m. wg.

$\therefore\ BC^2 = 100 = 36 + 64 = AB^2 + AC^2.$

$\therefore\ \angle BAC = 90^\circ$ = mg‡KvY|

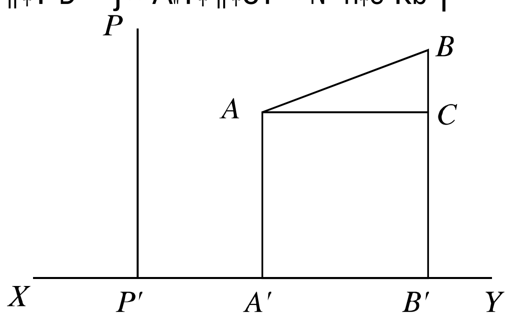


wPÎ 3.5

3 (L) j¤ Awf‡¶c (Orthogonal Projection)

we›`i j¤ Awf‡¶c : †Kv‡bv wbw`ó mij†iLvi Dci †Kv‡bv we›`i j¤ Awf‡¶c ej‡Z †mB we›` †_‡K D³ wbw`ó †iLvi Dci Aw¼Z j‡¤i cv`we›`‡K eSvq|

g‡bKwi, XY GKwU wbw`ó mij‡iLv Ges $P$ †h‡Kv‡bv we›` (wPÎ 3.6)| P we›` †_‡K $XY$ †iLvi Dci Aw¼Z j¤ $PP'$ Ges j¤ $PP'$ Gi cv`we›` P'|
mZivs, P' we›` $XY$ †iLvi Dci $P$ we›`i j¤ Awf‡¶c| A_vr †Kv‡bv wbw`ó †iLvi Dci †Kv‡bv we›`i j¤ Awf‡¶c GKwU we›`| Avgiv G avibv †_‡K ej‡Z cwi †Kv‡bv mij‡iLvi Dci j¤ †h‡Kvb mij‡iLvi j¤ Awf‡¶c GKwU we›`| †m †¶‡Î D³ j¤ Awf‡¶‡ci ˆ`N¨ n‡e kb¨|



wPÎ : 3·6 wbw`ó †iLv $XY$ Gi Dci †Kv‡bv we›` P Ges †iLvsk AB Gi j¤ Awf‡¶c|

†iLvs‡ki j¤ Awf‡¶c :

awi, AB †iLvs‡ki cÖvšÍ we›`Øq $A$ I $B$ (wPÎ : 3.6)| GLb $A$ I $B$ we›` †_‡K XY †iLvi Dci Aw¼Z j¤ h_vK‡g AA' I BB'| AA' j‡¤i cv`we›` A' Ges BB' j‡¤i cv`we›` B'| GB $A'B'$ †iLvskB n‡"Q XY †iLvi Dci AB †iLvs‡ki j¤ Awf‡¶c|

mZivs, †`Lv hv‡"Q j¤ A¼‡bi gva¨‡g Awf‡¶c wbYq Kiv nq| ZvB $A'B'$ †iLvs‡k‡K XY †iLvi Dci AB †iLvs‡ki j¤ Awf‡¶c (*Orthogonal Projection*) ejv nq|

j¶Yxq :
1| †Kv‡bv †iLvi Dci ‡Kvb we›` †_‡K Aw¼Z j‡¤i cv`we›`B H we›`i j¤ Awf‡¶c|

2| †Kv‡bv †iLvi Dci j¤ †iLvi j¤ Awf‡¶c GKwU we›`| d‡j j¤ Awf‡¶‡ci ˆ`N¨ kb¨|

3| †Kv‡bv wbw`ó †iLvi mgvšÍivj †iLvs‡ki j¤ Awf‡¶c H †iLvs‡ki mgvb n‡e|

wPÎ 3.6 G AB †iLvsk XY Gi mgvšÍivj n‡j $AB = A'B'$ n‡e|

KwZcq ¸i“ZcY Dccv`¨

cx_v‡Mviv‡mi Dccv‡`¨i Dci wfwË K‡i Ges j¤ Awf‡¶‡ci aviYvi mnv‡h¨ K‡qKwU ¸i“ZcY Dccv‡`¨i hw³gjK cgvY Dcv¯vcb Kie|

Dccv`¨ 3·3

¯j‡KvYx wÎf‡Ri ¯j‡Kv‡Yi wecixZ evûi Dci Aw¼Z eM‡¶Î H †Kv‡Yi mwbwnZ Ab¨ `B evûi Dci Aw¼Z eM‡¶Î؇qi Ges H `B evûi †h‡Kv‡bv GKwU I Zvi Dci Aci evûi j¤ Awf‡¶‡ci AšMZ AvqZ‡¶‡Îi wØ¸‡Yi mgwói mgvb|

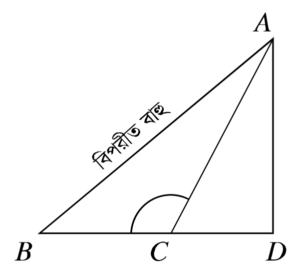
we‡kl wbePb : g‡b Kwi ABC wÎf‡Ri $\angle BCA$ ¯j‡KvY, AB ¯j‡Kv‡Yi wecixZ evû Ges ¯j‡Kv‡Yi mwbwnZ evûØq h_vµ‡g BC I AC

BC evûi ewaZvs‡ki Dci AC evûi j¤ Awf‡¶c CD (wPÎ : 3.7) | cgvY Ki‡Z n‡e †h,
$AB^2 = AC^2 + BC^2 + 2BC \cdot CD.$

cgvY : BC evûi ewaZvs‡ki Dci AC evûi j¤ Awf‡¶c CD nIqvi ΔABD GKwU mg‡KvYx wÎfR Ges $\angle ADB$ mg‡KvY|

mZivs cx_v‡Mviv‡mi Dccv`¨ Abmv‡i

$$AB^2 = AD^2 + BD^2$$
$$= AD^2 + (BC + CD)^2 \qquad [\because BD = BC + CD]$$
$$= AD^2 + BC^2 + CD^2 + 2BC \cdot CD.$$



wPÎ : 3.7

$$\therefore AB^2 = AD^2 + CD^2 + BC^2 + 2BC \cdot CD..........(1)$$

Avevi ΔACD mg‡KvYx wÎfR Ges $\angle ADC$ mg‡KvY|

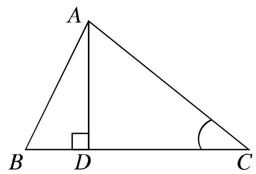
$$\therefore AC^2 = AD^2 + CD^2............(2)$$

(2) bs mgxKiY n‡Z AC^2 Gi gvb (1) bs mgxKi‡Y ewm‡q cvB,
$AB^2 = AC^2 + BC^2 + 2BC \cdot CD$ [cgwYZ]

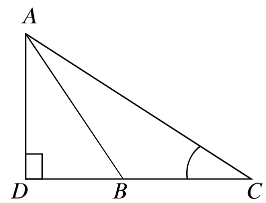
Dccv`¨ 3·4

†h‡Kv‡bv wÎf‡Ri m²‡Kv‡Yi wecixZ evûi Dci Aw¼Z eM‡¶Î Aci `B evûi Dci Aw¼Z eM‡¶Î؇qi mgwó A‡c¶v H `B evûi †h‡Kv‡bv GKwU I Zvi Dci AciwUi j¤ Awf¶‡ci AšMZ AvqZ‡¶‡Îi wØ¸Y cwigvY Kg|

we‡kl wbePb : ΔABC m²‡KvYx wÎf‡Ri $\angle C$ m¶‡KvY Ges m²‡Kv‡Yi wecixZ evû AB | Aci `B evû hv_vK‡g AC I BC | g‡b Kwi, BC evûi Dci (wPÎ: 3·8-L) Ges BC evûi ewaZvs‡ki Dci (wPÎ : 3·8-K) j¤ AD | Zvn‡j Dfq wÎf‡Ri †¶‡Î BC evûi Dci AC evûi j¤ Awf‡¶c CD |



wPÎ : 3·8 (K)



wPÎ: 3·8 (L)

cgvY : ΔABD Gi $\angle ADB$ mg‡KvY|

$\therefore AB^2 = AD^2 + BD^2$ [cx_v‡Mviv‡mi Dccv`¨] .... (1)

c_g wP‡Î $BD = BC - DC$
wØZxq wP‡Î $BD = DC - BC$
$\therefore$ Dfq‡¶‡Î $BD^2 = (BC - DC)^2 = (DC - BC)^2$

$$= BC^2 + DC^2 - 2BC \cdot DC$$

$$= BC^2 + CD^2 - 2BC \cdot CD \quad [\because CD = DC]$$

$\therefore\ BD^2 = BC^2 + CD^2 - 2BC \cdot CD$..........(2)
GLb mgxKiY (1) I (2) n‡Z cvIqv hvq
$AB^2 = AD^2 + BC^2 + CD^2 - 2BC \cdot CD$
ev, $AB^2 = AD^2 + CD^2 + BC^2 - 2BC \cdot CD$ ....(3)
Avevi ΔADC mg‡KvYx wÎfR Ges $\angle D$ mg‡KvY
$\therefore\ AC^2 = AD^2 + CD^2$ [cx_v‡Mviv‡mi Dccv`¨] .... (4)
mgxKiY (3) I (4) n‡Z cvB,
$AB^2 = AC^2 + BC^2 - 2BC \cdot CD.$ [cgwYZ]

we. `. : $C$ we›`y †_‡K AB Gi Dci j¤ A¼‡bi gva¨‡g GKB fv‡e Dccv`¨wU cgvY Kiv hvq|

cx_v‡Mviv‡mi Dccv`¨, j¤ Awf‡¶c Ges cx_v‡Mviv‡mi Dccv‡`¨i Dci wfwË K‡i ewYZ Dccv`¨mgn n‡Z j¶Yxq weIq Ges wm×vš mgn|

j¶Yxq :

1| mg‡KvYx wÎf‡Ri †¶‡Î mg‡Kv‡Yi mwbwnZ evûØq ci¯úi j¤ weavq Zv‡`i c‡Z¨KwUi j¤ Awf‡¶c kb¨| mZivs $BC \cdot CD = 0.$

2| Dccv`¨ 3·3 I Dccv`¨ 3·4, Dccv`¨ 3·1 Gi wfwËi Dci cwZwóZ| ZvB Dccv`¨ 3·3 I Dccv`¨ 3·4 †K Dccv`¨ 3·1 A_vr cx_v‡Mviv‡mi Dccv‡`¨i Abwm×vš ejv hvq|

Dc‡iv³ Av‡jPbv mv‡c‡¶ MwnZ wm×všmgn :

ΔABC Gi †¶‡Î,

1| $\angle C$ ¯’j‡KvY n‡j,

$AB^2 > AC^2 + BC^2$ [Dccv`¨ 3·3]

2| $\angle C$ mg‡KvY n‡j,

$AB^2 = AC^2 + BC^2$ [Dccv`¨ 3·1]

3| $\angle C$ m²‡KvY n‡j,

$AB^2 < AC^2 + BC^2$ [Dccv`¨ 3·4]

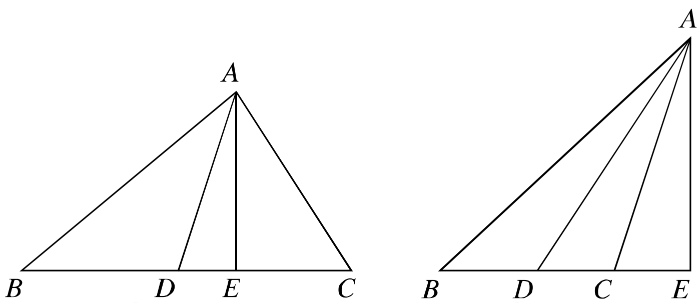
wb‡gv³ Dccv`¨wU cx_v‡Mvivm‡mi Dccv‡`¨i we¯—vi A_vr Dccv`¨ 3.3 I Dccv`¨ 3.4 Gi Dci wfwË K‡i cwZwôZ| GB Dccv`¨wU G¨v‡cv‡jwbqvm KZK ewYZ e‡j GwU G¨v‡cv‡jwbqv‡mi Dccv`¨ bv‡g cwiwPZ|

Dccv`¨ 3·5 (G¨v‡cv‡jwbqv‡mi Dccv`¨)

wÎf‡Ri †h‡Kv‡bv `B evûi Dci Aw¼Z eM‡¶Î Ø‡qi †¶Î d‡ji mgwó, ZZxq evûi A‡a‡Ki Ici Aw¼Z eM‡¶‡Îi †¶Îdj Ges H evûi mgwØLÊK ga¨gvi Ici Aw¼Z eM‡¶‡Îi †¶Îd‡ji mgwói wØ¸Y|

we‡kl wbePb : ΔABC Gi AD ga¨gv BC evû‡K mgwØLwÛZ K‡i‡Q|

cgvY Ki‡Z n‡e †h, $AB^2 + AC^2 = 2(AD^2 + BD^2)$



wPÎ : 3·9 (K) wPÎ : 3·9 (L)

A¼b : BC evûi Dci (wPÎ : 3·9 (K)) Ges BC evûi ewaZvs‡ki (wPÎ 3·9 (L)) AE j¤ A¼b Kwi|

cgvY : ΔABD Gi $\angle ADB$ ¯'j‡KvY Ges BD †iLvi ewaZvs‡ki Dci AD †iLvi j¤ Awf‡¶c DE [Dfq wP‡Î]|

$\therefore$ ¯'j‡Kv‡Yi †¶‡Î cx_v‡Mvivm‡mi Dccv‡`¨i we¯—wZ Abmv‡i [Dccv`¨ 3·3]

Avgiv cvB, $AB^2 = AD^2 + BD^2 + 2\,BD \cdot DE$...........(1)

Avevi, ΔACD Gi $\angle ADC$ m²‡KvY Ges DC †iLvi (wP‡Î 3.9 (K) Ges DC †iLvi ewaZvs‡ki (wP‡Î 3.9 (L)) Dci AD †iLvi j¤ Awf‡¶c DE.

$\therefore$ m²‡Kv‡Yi †¶‡Î cx_v‡Mvivm‡mi Dccv‡`¨i we¯—wZ Abmv‡i (Dccv`¨ 3.4) cvB,

$AC^2 = AD^2 + CD^2 - 2CD \cdot DE$............(2)

GLb mgxKiY (1) I (2) †hvM K‡i cvB,

$$AB^2 + AC^2 = 2AD^2 + BD^2 + CD^2 + 2BD \cdot DE - 2CD \cdot DE$$
$$= 2AD^2 + BD^2 + BD^2 + 2BD \cdot DE - 2BD \cdot DE; \quad [\because BD = CD]$$
$$= 2AD^2 + 2BD^2$$
$$= 2(AD^2 + BD^2). \qquad \text{[cgwYZ]}$$

wm×vš : G¨v‡cv‡jvwbqv‡mi Dccv‡`¨i gva¨‡g wÎfz‡Ri evû I ga¨gvi m¤úK© wbYq |

g‡b Kwi, ΔABC Gi BC, CA I AB evûi %`N¨ h\_vµ‡g $a, b$ I $c$ | $BC, CA$ I $AB$ evûi Dci AwZ ga¨gv $AD, BE$ I $CF$ Gi ˆ`N¨ h_vK‡g d, e I f.

Zvn‡j, G¨v‡cv‡jvwbqv‡mi Dccv`¨ n‡Z cvB,

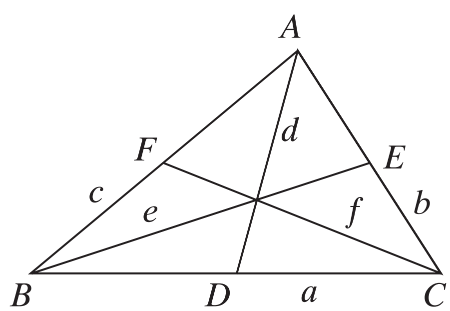
$$AB^2 + AC^2 = 2(AD^2 + BD^2)$$

ev, $c^2 + b^2 = 2\left(d^2 + \left(\frac{1}{2}a\right)^2\right) \quad \left[\because BD = \frac{1}{2}a\right]$

ev, $b^2 + c^2 = 2d^2 + 2 \cdot \frac{1}{4}a^2$

ev, $b^2 + c^2 = 2d^2 + \frac{a^2}{2}$

ev, $d^2 = \frac{2(b^2 + c^2) - a^2}{4}$



Abiƒc fv‡e cvIqv hvq,

$$e^2 = \frac{2(c^2 + a^2) - b^2}{4}$$

Ges $f^2 = \frac{2(a^2 + b^2) - c^2}{4}$

$\therefore$ †Kv‡bv wÎfz‡Ri evûi ˆ`N¨ Rvbv \_vK‡j ga¨gvmg‡ni ˆ`N¨ Rvbv hvq|

Avevi,

$$d^2 + c^2 + f^2 = \frac{2(b^2 + c^2) - a^2}{4} + \frac{2(c^2 + a^2) - b^2}{4} + \frac{2(a^2 + b^2) - c^2}{4}$$
$$= \frac{3}{4}(a^2 + b^2 + c^2)$$

$\therefore 3(a^2 + b^2 + c^2) = 4(d^2 + c^2 + f^2).$

mZivs ejv hvq †Kv‡bv wÎfz‡Ri wZbwU evûi Dci AwZ eM©‡¶Î mg‡ni †¶Îd‡ji mgwói wZb¸Y D³ wÎfz‡Ri ga¨gv Î‡qi Dci AwZ eM©‡¶Îmg‡ni †¶Îd‡ji mgwói Pvi ¸‡Yi mgvb|

wÎfRwU mg‡KvYx A_vr $\angle C$ = mg‡KvY Ges AB AwZfR n‡j

$$c^2 = a^2 + b^2$$

$$\therefore \quad a^2 + b^2 + c^2 = 2c^2$$

ev, $\frac{4}{3}(d^2 + c^2 + f^2) = 2c^2$

ev, $2(d^2 + c^2 + f^2) = 3c^2.$

mZivs, ejv hvq mg‡KvYx wÎf‡Ri ga¨gvÎ‡qi Dci AwZ eM©‡¶Îmg‡ni †¶Î d‡ji mgwói wØ¸Y AwZf‡Ri Dci AwZ eM©‡¶‡Îi †¶Îd‡ji wZY¸‡Yi mgvb|

Abkxjbx 3·1

1| ΔABC Gi $\angle B = 60^\circ$ n‡j cgvY Ki‡h, $AC^2 = AB^2 + BC^2 - AB \cdot BC$

2| ΔABC Gi $\angle B = 120^\circ$ n‡j cgvY Ki‡h, $AC^2 = AB^2 + BC^2 + AB \cdot BC$

3| ΔABC Gi $\angle B = 90^\circ$ Ges BC Gi ga¨we›` D | cgvY Ki ‡h, $AB^2 = AD^2 + 3BD^2$

4| ΔABC G AD, BC evûi Dci j¤ Ges BE, AC Gi Dci j¤ | †`LvI †h,
$BC \cdot CD = AC \cdot CE$

5| ΔABC Gi BC evû P I Q we›`‡Z wZbwU mgvb As‡k wef³ n‡q‡Q| cgvY Ki †h,
$AB^2 + AC^2 = AP^2 + AQ^2 + 4PQ^2$.
[ms‡KZ : $BP = PQ = QC$; ΔABQ Gi ga¨gv AP .
$AB^2 + AQ^2 = 2 \cdot (BP^2 + AP^2) = 2PQ^2 + 2AP^2$
ΔAPC Gi ga¨gv AQ,
$AP^2 + AC^2 = 2PQ^2 + 2AQ^2$]

6| ΔABC Gi $AB = AC$ | fwg BC Gi Dci P †h‡Kv‡bv we›`| cgvY Ki †h,
$AB^2 - AP^2 = BP \cdot PC.$
[ms‡KZ : BC Gi Dci AD j¤ AvK Zvn‡j $AB^2 = BD^2 + AD^2$ Ges $AP^2 = PD^2 + AD^2$]

7 | ΔABC Gi ga¨gvÎq G we›`‡Z wgwjZ n‡j cgvY Ki †h,
$AB^2 + BC^2 + CA^2 = 3(GA^2 + GB^2 + GC^2)$
[ms‡KZ : G¨v‡cv‡jvwbqv‡mi Dccv‡`¨i Av‡jv‡K MnxZ wm×vš— mgn †`L‡Z n‡e A_vr, wÎf‡Ri evûi ˆ`N¨ I ga¨gvi m¤úK †`L‡Z n‡e]

3 (M) wÎfR I eË welqK Dccv`¨

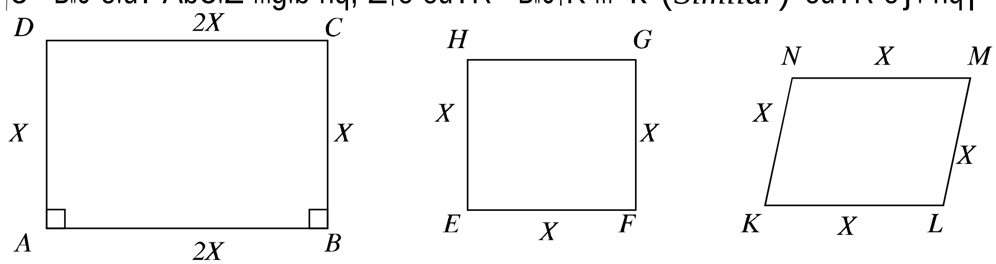
GB As‡k wÎfR I eË welqK K‡qKwU ¸i“ZcY Dccv‡`¨i hw³gjK cgvY Dc¯vcb Kiv n‡e| Dccv`¨mgn cgv‡Yi Rb¨ `BwU wÎf‡Ri m`kZv m¤ú‡K ceÁvb _vKv Avek¨K| gva¨wgK R¨wgwZ‡Z wÎf‡Ri m`kZv m¤ú‡K we¯wiZ Av‡jvPbv Kiv n‡q‡Q| GB Dccv`¨¸‡jv cgv‡Yi c‡e wk¶v_xiv wÎf‡Ri mv`kZv m¤ú‡K †R‡b wb‡e| wk¶v_x‡`i mweav‡_ wÎf‡Ri mv`kZv m¤ú‡K ms‡¶‡c Av‡jvPbv Kiv n‡jv|

†Kv‡Yi †¶‡Î m`kZv : mgvb msL¨K evûwewkó `ywU eû¯f‡Ri GKwUi †KvY¸‡jv hw` avivewnKfv‡e AciwUi †KvY¸‡jvi mgvb nh, Z‡e eû¯fR `ywU‡K m`k‡KvYx eû¯fR ejv nq|

evûi Abcv‡Zi †¶‡Î m`kZv : mgvb msL¨K evûwewkó `ywU eû¯f‡Ri GKwUi kxl© we›`¸‡jv‡K hw` avivewnKfv‡e AciwUi kxl©we›`¸‡jvi m‡½ Ggbfv‡e wgj Kiv hvq †h, eû¯fR `ywUi-

(1) Abyi~c †KvY¸‡jv mgvb nq Ges

(2) Abyi~c `ywU evûi AbcvZ mgvb nq, Z‡e eû¯fR `ywU‡K m`k (*Similar*) eû¯fR ejv nq|



wPÎ 3.10

Dc‡ii wP‡Î j¶ Ki‡j †`Le †h,

(1) AvqZ $ABCD$ I eM© $EFGH$ m`k bq hw`I Zviv m`k‡KvYx|

(2) eM© $EFGH$ I i¤^m $KLMN$ m`k bq hw`I Zv‡`i kxl©we›`¸‡jvi †h‡Kv‡bv avivewnK wgjKi‡Yi d‡j Abyi~c evû `ywUi AbcvZ¸‡jv mgvb nq|

`ywU wÎf‡Ri †ejvq Aek¨ G iKg nq bv| `ywU wÎf‡Ri kxl©we›`¸‡jvi †KvY wgjKi‡Yi d‡j hw` m¤ú³ vi msÁvq D‡jøwLZ kZ© `ywUi GKwU mZ¨ nq, Z‡e AciwUI mZ¨ nq Ges wÎfR `ywU m`k nq|

G cm‡½ D‡jøL¨ †h,

(1) `ywU wÎfR m`k‡KvYx n‡j mgvb †KvY `ywU‡K Abyi~c †KvY Ges Abyi~c †Kv‡Yi wecixZ evû `ywU‡K Abyi~c evû aiv nh|

(2) `ywU wÎf‡Ri GKwUi wZb evû AciwUi wZb evûi mgvbcvwZK n‡j, AvbcvwZK evû `ywU‡K Abyi~c evû Ges Abyi~c evûi wecixZ †KvY `ywU‡K Abyi~c †KvY aiv nq|

(3) Dfq‡¶‡Î Abyi~c †KvY¸‡jvi kxl©we›` wgj K‡i wÎfR `ywU eY©bv Kiv nq| †hgb, ΔABC I ΔDEF Gi Abyi~c †KvY¸‡jv n‡”Q $\angle A$ I $\angle D$, $\angle B$ I $\angle E$, $\angle C$ I $\angle F$ Ges Abyi~c evû¸‡jv n‡”Q AB I DE, AC I DF, BC I EF.|

`ywU wÎf‡Ri m`kZv m¤úwKZ K‡qKwU Dccv‡`¨i mswk¶ß eY©bv †`Iqv n‡jv|

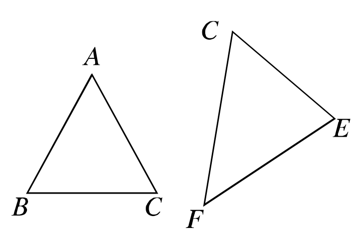
Dccv`¨ 3·6

`ywU wÎfR m`k‡KvYx n‡j Zv‡`i Abyi~c evû¸‡jv mgvbcvwZK n‡e|

cv‡ki wP‡Î ΔABC I ΔDEF m`k‡KvYx wÎfR|

A_©vr, $\angle A=\angle D, \angle B=\angle E$ Ges $\angle C=\angle F$. nIqvq

$\dfrac{AB}{DE}=\dfrac{AC}{DF}=\dfrac{BC}{EF}$ n‡e| A_©vr Abyi~c evû¸‡jv mgvbcvwZK n‡e|

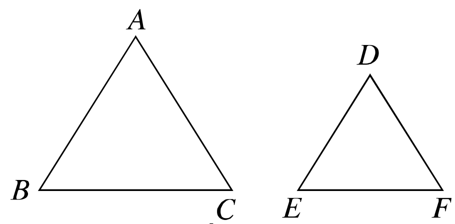


wPÎ : 3·11

Abywm×všÍ : `ywU wÎfyR m`„k‡KvYx n‡j, Zviv m`„k nq|
gšÍe¨ : `ywU wÎf‡Ri GKwUi `y †KvY AciwUi `y †Kv‡Yi mgvb n‡j wÎfyR `ywU m`„k‡KvYx Ges Gi d‡j G¸‡jv m`„k nq| KviY †h‡Kv‡bv wÎf‡Ri wZb †Kv‡Yi mgwó `y mg‡KvY|

Dccv`¨ 3·7

`ywU wÎf‡Ri evû¸‡jvi mgvbcvwZK n‡j Abyi~c evûi wecixZ †KvY¸‡jv ci¯úi mgvb nq|



wPÎ 3·12

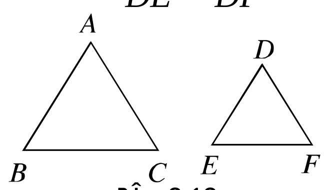
cv‡ki wP‡Î ΔABC I ΔDEF Gi evû¸‡jv mgvbcvwZK A_vr, $\frac{AB}{DE}=\frac{AC}{DF}=\frac{BC}{EF}$ nIqvq wÎfRØ‡qi †KvY¸‡jv ci¯úi mgvb| A_vr, $\angle A=\angle D, \angle B=\angle E$ Ges $\angle C=\angle F.$|

Dccv`¨ 3.7 †K Dccv`¨ 3.6 Gi wecixZ wnmv‡e I ejv †h‡Z cv‡i|

Dccv`¨ 3·8

`ywU wÎf‡Ri GKwUi GK †KvY AciwUi GK †Kv‡Yi mgvb Ges mgvb †KvY msjMœ evû¸‡jv mgvbcvwZK n‡j wÎfyR `ywU m`„k n‡e|

cv‡ki wP‡Îi (wPÎ : 3.13) ΔABC I ΔDEF Gi $\angle A=\angle D$ Ges mgvb †KvY msjMœ evûØq AB, AC Ges DE I EF mgvbcvwZK| A_vr, $\frac{AB}{DE}=\frac{AC}{DF}$ nIqvq ΔABC I ΔDEF m`„k|



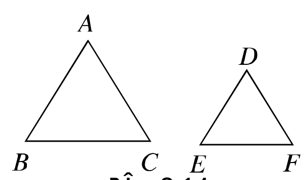
wPÎ : 3.13

Dccv`¨ 3·9

`ywU m`„k wÎfR‡¶‡Îi †¶Îdj؇qi AbycvZ Zv‡`i †h‡Kv‡bv `y Abyi~c evûi Ici Aw¼Z eM©‡¶‡Îi †¶Îdj؇qi Abycv‡Zi mgvb|

cv‡ki wP‡Îi ΔABC I ΔDEF wÎfRØq m`„k| wÎfR `ywUi Abyi~c evû BC I EF| GB Ae¯'vq wÎfRØ‡qi †¶Îd‡ji AbycvZ BC I EF evûØ‡qi Ici Aw¼Z eM©‡¶‡Îi †¶Îdj؇qi Abycv‡Zi mgvb|

A_vr $\frac{\Delta ABC}{\Delta DEF}=\frac{BC^2}{EF^2}$|



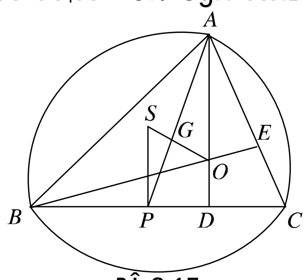
wPÎ : 3.14

wÎf‡Ri †¶‡Î Dc‡iv³ Av‡jvPbv I Dccv`¨mgn‡K wfwË K‡i wb‡gœv³ Dccv`¨mg‡ni hyw³gyjK cgvY Dc¯'vcb Kiv n‡jv|

Dccv`¨ 3.10

wÎf‡Ri cwi‡K›`, fi‡K›` I j¤ we›` mg‡iL|

we‡kl wbePb : g‡b Kwi, ΔABC Gi j¤ we›` $O$ cwi‡K›` S Ges AP GKwU ga¨gv| j¤ we›` $O$ Ges cwi‡K›` S Gi ms‡hvM †iLv AP ga¨gv‡K G we›`‡Z †Q` K‡i‡Q| S, P †hvM Ki‡j SP †iLv BC Gi Dci j¤| Zvn‡j, G we›`wU $\Delta ABC$ Gi fi‡K›` GwU cgvY KivB h‡_ó n‡e|



wPÎ 3.15

cgvY : Avgiv Rwb, †Kv‡bv wÎf‡Ri j¤ we›` †_‡K kx‡li `iZ wÎf‡Ri cwi‡K›` †_‡K H kx‡li wecixZ evûi wØ¸Y| $\Delta ABC$ Gi j¤ we›` O †_‡K A kx‡li `iZ $OA$ Ges cwi‡K›` S †_‡K A kx‡li wecixZ evû BC Gi `iZ SP |

$\therefore\ OA = 2SP$ .... (1)

GLb †h‡nZ AD I SP DfqB BC Gi Ici j¤ †m‡nZ $AD \parallel SP$.

GLb $AD \parallel SP$ Ges AP G‡`i †Q`K|

$\therefore \angle PAD = \angle APS$ [GKvšÍi †KvY]

A_vr, $\angle OAG = \angle SPG$

GLb ΔAGO Ges ΔPGS Gi g‡a¨

$\angle AGO = \angle PGS$ [wecZxc †KvY]

$\angle OAG = \angle SPG$ [GKvšÍi †KvY]

$\therefore$ Aewkó $\angle AOG$ = Aewkó $\angle PSG$

$\therefore$ ΔAGO Ges ΔPGS m`k †KvYx|

mZivs, $\dfrac{AG}{GP} = \dfrac{OA}{SP}$

ev, $\dfrac{AG}{GP} = \dfrac{OA}{SP}$

ev, $\dfrac{AG}{GP} = \dfrac{2SP}{SP}$ [(1) bs mgxKiY n‡Z]

ev, $\dfrac{AG}{GP} = \dfrac{2}{1}$

$\therefore \ AG : GP = 2:1$

A_vr, G we›` AP ga¨gv‡K $2:1$ Abcv‡Z wef³ K‡i‡Q|

$\therefore \ G$ we›` $\Delta ABC$ Gi fi‡K›`| (cgwYZ)

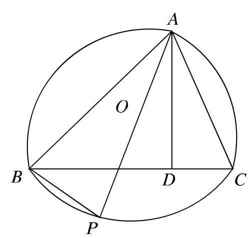
`óe¨ : (1) bewe›`eË (*Nine Point Circle*) : †Kv‡bv wÎf‡Ri evû¸‡jvi ga¨we›`Îq, kxl©we›`¸‡jv †_‡K wecixZ evûÎ‡qi Dci Aw¼Z j¤Î‡qi cv` we›`Îq Ges kxl©we›` I j¤we›`i ms‡hvRK †iLvÎ‡qi ga¨we›`Îq, me‡gvU GB bqwU we›` GKB e‡Ëi Dci Ae¯'vb K‡i| GB eË‡KB bewe›`eË e‡j|

(2) wÎf‡Ri j¤ we›` I cwi‡K›`` ms‡hvRb K‡i Drcb mmxg mij‡iLvi ga¨we›`B bewe›` e‡Ëi †K›`|

(3) bewe›` e‡Ëi e¨vmva wÎf‡Ri cwie¨vmv‡ai A‡a‡Ki mgvb|

Dccv`¨ 3.11 (eþv ¸‡ßi Dccv`¨)

†Kv‡bv wÎf‡R †h‡Kv‡bv `B evûi AšÍM©Z AvqZ‡¶‡Îi †¶Îdj wÎf‡Ri cwie‡Ëi e¨vm Ges H evûØ‡qi mvaviY we›` †_‡K fwgi Dci Aw¼Z j‡¤i AšÍM©Z AvqZ‡¶‡Îi †¶Îd‡ji mgvb|



wPÎ : 3.16

we‡kl wbePb : g‡b Kwi, ABC wÎf‡Ri cwi‡K›` O Ges AP cwie‡Ëi GKwU e¨vm| ΔABC Gi kxl A †_‡K wecixZ evû BC Gi Dci AD j¤|

cgvY Ki‡Z n‡e †h, $AB \cdot AC = AP \cdot AD$.

A¼b : B, P †hvM Kwi|

cgvY : GKB Pvc AB Gi Rb¨ $\angle APB$ I $\angle ACD$ eËvsk¯'Z †KvY| AP e‡Ëi e¨vm e‡j $\angle ABP$ AaeË¯' †KvY Ges BC evûi Dci AD j¤ nIqvq $\angle ADC$ mg‡KvY|

GLb ΔAPB I ΔADC Gi g‡a¨ $\angle APB = \angle ACD$ [GKB eËvsk¯'Z †KvY mgvb|]

$\angle ABP$ = AaeË¯' †KvY = GK mg‡KvY = $\angle ADC$.

$\therefore$ Aewkó $\angle BAP$ = Aewkó $\angle CAD$.

$\therefore \ \Delta ABP$ I ΔADC m`k‡KvYx|

$$\therefore \ \frac{AB}{AD} = \frac{AP}{AC}$$

A_vr, $AB \cdot AC = AP \cdot AD.$ [cgwYZ]

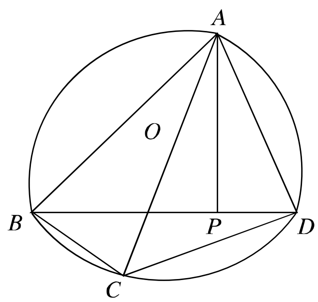
j¶Yxq : ΔABC Gi cwie‡Ëi e¨vmva R n‡j, $R = \frac{1}{2} AP$

A_vr, $AP = 2R.$

$\therefore$ Dc‡ii Dccv`¨ †_‡K cvIqv hvq $AB \cdot AC = 2R \cdot AD.$

Dccv`¨ 3.12 (U‡jwgi Dccv`¨)

e‡Ë AšwjwLZ †Kv‡bv PZf©‡Ri KY©Ø‡qi AšMZ AvqZ‡¶Î H PZf©‡Ri wecixZ evû؇qi AšMZ AvqZ‡¶‡Îi mgwói mgvb|



wPÎ : 3.17

we‡kl wbePb : g‡b Kwi e‡Ë AšwjwLZ $ABCD$ PZf©‡Ri wecixZ evû¸‡jv h_vµ‡g AB I CD Ges BC I AD| AC Ges BD PZf©RwUi `BwU KY| cgvY Ki‡Z n‡e †h,

$AC \cdot BD = AB \cdot CD + BC \cdot AD.$

A¼b : $\angle BAC$ †K $\angle DAC$ Gi †QvU a‡i wb‡q A we›`‡Z $AD$ †iLvs‡ki mv‡_ $\angle BAC$ Gi mgvb K‡i $\angle DAP$ AwK †hb $AP$ †iLv $BD$ KY‡K $P$ we›`‡Z †Q` K‡i|

cgvY : A¼b Abmv‡i $\angle BAC = \angle DAP$

Dfqc‡¶ $\angle CAP$ †hvM K‡i cvB,

$\angle BAC + \angle CAP = \angle DAP + \angle CAP$

A_vr, $\angle BAP = \angle CAD$

GLb ΔABP I ΔACD Gi g‡a¨

$\angle ADP = \angle ACD$

$\angle ABD = \angle ACD$ [GKB eËvsk¯Z †KvY mgvb e‡j]

Ges Aewkó $\angle APB =$ Aewkó $\angle ADC$

$\therefore$ ΔABD I ΔACD m`k‡KvYx|

$\therefore$ $\frac{BP}{CD} = \frac{AB}{AC}$

A_vr, $AC \cdot BP = AB \cdot CD$ .... (1)

Avevi, ΔABC I ΔAPD Gi g‡a¨

$\angle BAC = \angle PAD$ [A¼b Abmv‡i]

$\angle ADP = \angle ACB$ [GKwU eËvsk¯’ †KvY mgvb e‡j]

Ges Aewkó $\angle ABC =$ Aewkó $\angle APD$

$\therefore$ ΔABC I ΔAPD m`„k‡KvYx|

$\therefore \dfrac{AD}{AC} = \dfrac{PD}{BC}$

A_©vr, $AC \cdot PD = BC \cdot AD$ .... (2)

GLb mgxKiY (1) I (2) †hvM K‡i cvB,

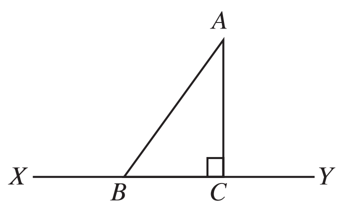
$AC \cdot BP + AC \cdot PD = AB \cdot CD + B \cdot AD$

ev, $AC(BP + PD) = AB \cdot CD + BC \cdot AD$

A_©vr, $AC \cdot BD = AB \cdot CD + BC \cdot AD$ [†h‡nZz $BP + PD = BD$] [cÖgvwYZ]

Abykxjbx 3·2

1|

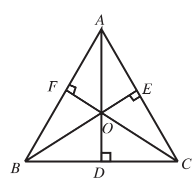


XY †iLvs‡k AB Gi j¤^ Awf‡¶c wb‡Pi †KvbwU?

K. AB L. BC

M. AC N. XY

2|



Ic‡ii wP‡Î †KvbwU j¤^ we›`y?

K. D L. E

M. F N. O

3| i wÎf‡Ri ga¨gvÎ‡qi †Q` we›`y‡K fi †K›`ª e‡j|

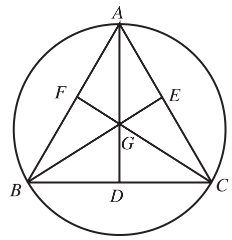
ii fi‡K›`ª †h‡Kv‡bv ga¨gv‡K 3:1 Abcv‡Z wef³ K‡i|

iii m`„k †KvYx wÎf‡Ri Abi~c evû¸‡jv mgvbycvwZK

wb‡Pi †KvbwU mwVK ?

K. i I ii L. ii I iii

M. i I iii N. i, ii I iii



D, E, F h_vµ‡g BC, AC I AB Gi ga¨we›` n‡j Ic‡ii wP‡Îi Av‡jv‡K 4-6 bs c‡ki DËi `vI:

4| G we›`i bvg wK?

K. j¤we›` L. Aš:‡K›`

M. fi‡K›` N. cwi‡K›`

5| ΔABC Gi kxl we›` w`‡q Aw¼Z e‡Ëi bvg wK?

K. cwieË L. Aš:eË

M. ewn:eË N. bewe›` eË

6| ΔABC Gi †¶‡Î wb‡Pi †KvbwU G¨v‡cv‡jvwbqv‡mi Dccv`¨‡K mg_b K‡i?

K. $AB^2 + AC^2 = BC^2$

L. $AB^2 + AC^2 = 2(AD^2 + BD^2)$

M. $AB^2 + AC^2 = 2(AG^2 + GD^2)$

N. $AB^2 + AC^2 = 2(BD^2 + CD^2)$

7| ABC wÎf‡Ri cwieË¯ †h‡Kv‡bv P we›` †_‡K $BC$ I $CA$ Gi Dci $PD$ I $PE$ j¤ A¼b Kiv n‡q‡Q| hw` ED †iLvsk AB †K O we›`‡Z †Q` K‡i, Z‡e cgvY Ki †h, PO †iLv AB Gi Dci j¤| A_vr $PO \perp AB$.

8| ΔABC Gi $\angle C$ mg‡KvY| C †_‡K AwZf‡Ri Dci Aw¼Z j¤ CD n‡j, cgvY Ki †h, $CD^2 = AD \cdot BD$.

9| ΔABC Gi kxlÎq †_‡K wecixZ evû¸‡jvi Ici j¤ AD, BE I CF †iLvÎq O we›`‡Z †Q` K‡i| cgvY Ki †h, $AO \cdot OD = BO \cdot OE = CO \cdot OF$. [ms‡KZ : ΔBOF Ges ΔCOE m`k| $\therefore BO : CO = OF : OE$]

10| AB e¨v‡mi Ici Aw¼Z Aae‡Ëi `BwU R¨v $AC$ I $BD$ ci¯úi $P$ we›`‡Z †Q` K‡i| cgvY Ki †h, $AB^2 = AC \cdot AP + BD \cdot BP$

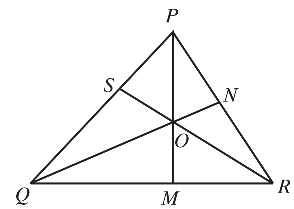
11| †Kv‡bv mgevû wÎf‡Ri cwie‡Ëi e¨vmva $3 \cdot 0$ †m.wg. n‡j H wÎf‡Ri evûi ˆ`N¨ wbYq Ki|

12| ABC mgwØevû wÎf‡Ri kxl©we›` $A$ n‡Z f~wg $BC$ Gi Ici Aw¼Z j¤ $AD$ Ges wÎf‡Ri cwie¨vmva $R$ n‡j cgvY Ki †h, $AB^2 = 2R \cdot AD.$ [e²v¸‡ßi Dccv`¨ $AB = AC$]

13| ABC wÎf‡Ri $\angle A$ Gi mgwØLÛK BC †K D we›`‡Z Ges $ABC$ cwie‡Ë‡K $E$ we›`‡Z †Q` K‡i‡Q| †`LvI †h, $AD^2 = AB \cdot AC - BD \cdot DC.$

14| ABC wÎf‡Ri AC I AB evûi Ici h_vµ‡g BE I CF j¤| †`LvI †h, $\Delta ABC : \Delta AEF = AB^2 : AE^2.$

15| ΔPQR -G PM, QN I RS ga¨gvÎq O we›`‡Z †Q` K‡i‡Q|

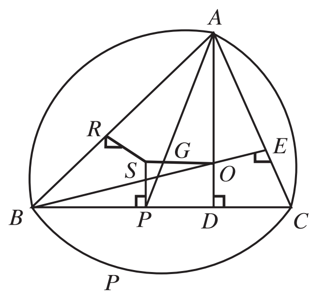


K. O we›`wUi bvg wK? O we›` PM †K wK Abcv‡Z wef³ K‡i?

L. ΔPQR n‡Z $PQ^2 + PR^2 = 2(PM^2+QM^2)$ m¤úKwU cÖZwôZ Ki|

M. †`LvI †h, $\Delta$PQR -Gi evû wZbwUi e‡Mi mgwó O we›` n‡Z kxl©we›` wZbwUi `i‡Zi e‡Mi mgwói wZb¸Y|

16|



Ic‡ii wP‡Î S, O h_vµ‡g cwi‡K›`ª I j¤we›`| AP ga¨gv, BC =a, AC = b Ges AB = c

K. OA Ges SP Gi g‡a¨ m¤úK wbYq Ki|

L. †`LvI †h, S, G, O GKB mij †iLvq Aew¯’Z|

M. $\angle C$ m²‡KvY n‡j a.CD = b.CE mgxKiYwU cÖZwôZ Ki|

Øv`k Aa¨vq

mgZjxq †f±i

Avgiv †¯‹jvi ivwk Ges Zvi Dci wewfbœ cÖKvi MvwYwZK cÖwµqvi c‡qvM wk‡LwQ| wKš' ïa †¯‹jvi ivwk m¤ú‡K avibv _vK‡jB ˆ`bw›`b Rxe‡bi A‡bK Kvhµg e¨vL¨v Kiv hvq bv| G‡¶‡Î Avgv‡`i †f±i ivwki aviYv c‡qvRb nq| GB Aa¨v‡q Avgiv †f±i ivwk m¤ú‡K Av‡jvPbv Ki‡ev|

Aa¨vq †k‡l wk¶v_xiv

- ➢ †¯‹jvi ivwk I †f±i ivwk eYbv Ki‡Z cvi‡e|
- ➢ †¯‹jvi ivwk I †f±i ivwk cZx‡Ki mvnv‡h¨ e¨vL¨v Ki‡Z cvi‡e|
- ➢ mgvb †f±i, wecixZ †f±i I Ae¯'vb †f±i e¨vL¨v Ki‡Z cvi‡e|
- ➢ †f±‡ii †hvM I †hvMwewa e¨vL¨v Ki‡Z cvi‡e|
- ➢ †f±‡ii we‡qvM e¨vL¨v Ki‡Z cvi‡e|
- ➢ †f±‡ii †¯‹jvi ¸wYZK I GKK †f±i e¨vL¨v Ki‡Z cvi‡e|
- ➢ †f±‡ii †¯‹jvi ¸wYZK I eÈbwewa e¨vL¨v Ki‡Z cvi‡e|
- ➢ †f±‡ii mvnv‡h¨ wewfbœ R¨vwgwZK mgm¨vi mgvavb Ki‡Z cvi‡e|

12.1| †¯‹jvi ivwk I †f±i ivwk

ˆ`bw›`b Rxe‡b cvq me‡¶‡ÎB e¯'i cwigv‡ci c‡qvRb nq| 5 †m.wg., 3 wgwbU, 12 UvKv, 5 wjUvi, 6° C BZ¨vw` Øviv h_vµ‡g e¯'i ˆ`N¨, mg‡qi cwigvY, UvKvi cwigvY, AvqZ‡bi cwigvY I Zvcgv‡Îi cwigvY †evSv‡bv nq| Gme cwigv‡ci Rb¨ †Kejgv‡Î GKKmn cwigvY D‡jøL Ki‡jB P‡j| Avevi hw` ejv nq GKwU †jvK GKwe›`y †_‡K hv‡Îv K‡i c_‡g 4 wg. I c‡i 5 wg. †Mj, Zvn‡j hvÎwe›`y †_‡K Zvi `iZ wbYq Ki‡Z †M‡j c_‡g Rvbv `iKvi †jvKwU MwZi w`K Kx? MwZi mwVK w`K bv Rvbv chš— hvÎwe›`y †_‡K †jvKwU KZ`i wM‡q‡Q Zv mwVKfv‡e wbYq m¤¢e bq|

†h ivwk †Kejgv‡Î GKKmn cwigvY Øviv m¤úYi¦‡c †evSv‡bv hvq, Zv‡K †¯‹jvi ev Aw`K ev wbw`K ivwk ejv nq| ˆ`N¨, fi, AvqZb, `"wZ, Zvcgv‡Îv BZ¨vw` c‡Z¨‡KB †¯‹jvi ivwk|

†h ivwk‡K m¤úYi¦‡c cÖKvk Kivi Rb¨ Zvi cwigvY I w`K Df‡qi c‡qvRb nq, Zv‡K †f±i ev mw`K ivwk ejv nq| miY, †eM, ZiY, IRb, ej BZ¨vw` c‡Z¨‡KB †f±i ivwk|

12.2| †f±i ivwki R¨vwgwZK cwZi¦c: w`K wb‡`kK †iLvsk

†Kv‡bv †iLvs‡ki GK cvš‡K Aw`we›` (initial point) Ges Aci cvš‡K Ašwe›` (terminal point) wn‡m‡e wPwýZ Ki‡j H †iLvsk‡K GKwU w`K wb‡`kK †iLv (directed line segment) ejv nq| †Kv‡bv w`K wb‡`kK †iLvs‡ki Aw` we›` A Ges Ašwe›` B n‡j H w`K wb‡`kK †iLvsk‡K $\overrightarrow{AB}$ Øviv mwPZ Kiv

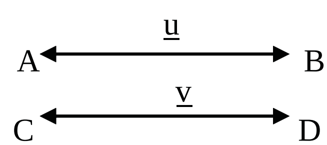
nq| c‡Z¨K w`K wb‡`©kK †iLvsk GKwU †f±i iwk, hvi cwigvY H †iLvs‡ki ˆ`N¨ ($|\overrightarrow{AB}|$ ev ms‡¶‡c $AB$ Øviv mwPZ) Ges hvi w`K A we›`y n‡Z AB †iLv eivei B we›`y wb‡`©kKvix w`K|

wecixZµ‡g †h‡Kv‡bv †f±i iwk‡K GKwU w`K wb‡`©kK †iLvsk Øviv cKvk Kiv hvq, †hLv‡b †iLvskwUi ˆ`N¨ iwkwUi cwigvY Ges †iLvskwUi Aw`we›`y n‡Z Ašwe›`y wb‡`©kKvix w`K c`Ë †f±i iwki w`K|
ZvB, †f±i iwk I w`K wb‡`©kK †iLvsk mgv_©K avi‡Yv| w`K wb‡`©kK‡K R¨wgwZK †f±i e‡j I D‡jL Kiv nq| Avgv‡`i Av‡jvPbv GKB mgZ‡j Aew¯’Z †f±‡ii g‡a¨ mxgve× _vK‡e|
aviK †iLv t †Kvb †f±i (w`K wb‡`©kK †iLvsk) †h Amxg mij‡iLvi Ask we‡kl, Zv‡K H †f±‡ii avi‡K †iLv ev ïa avi‡K (support) ejv nq|
mPivPi GKwU †f±i‡K GKwU A¶i w`‡q mwPZ Kiv nq;

†hgb $\overrightarrow{AB} = \underline{u}$ wKš $\overrightarrow{AB}$ wjL‡j †hgb †evSv hvq †h, †f±iwUi Aw`we›` A I Ašwe›` B, $\underline{u}$ wjL‡j †Zgb †Kv‡bv Z_¨ cvIqv hvq bv|

KvR: 1| †Zvgvi ev‡mv n‡Z ¯‹zj †mvRv `w¶‡Y 3 wK. wg. `‡i Aew¯’Z| ev‡mv n‡Z †n‡U ¯‹‡j †h‡Z GK N›Uv mgq jv‡M| ‡Zvgvi MwZ‡eM KZ?
2| ¯‹zj QzwUi ci mvB‡K‡j 20 wgwb‡U evmv G‡j G‡¶‡Î ‡Zvgvi MwZ‡eM KZ?

12.3| †f±‡ii mgZv, wecixZ †f±i
mgvb †f±i t GKwU †f±i $\underline{u}$-†K Aci GKwU †f±i $\underline{v}$-Gi mgvb ejv nq hw`



(i) $|\underline{u}| = |\underline{v}|$ ($\underline{u}$ Gi ˆ`N¨ mgvb $\underline{v}$ Gi ˆ`N¨)
(ii) $\underline{u}$-Gi avi‡K, $\underline{v}$-Gi avi‡Ki m‡½ Awfbœ A_ev mgvšÍivj nq,
(iii) $\underline{u}$-Gi w`K $\underline{v}$-Gi w`‡Ki m‡½ GKgyLx nq| mgZvi GB msÁv †h wb‡Pi wbqg¸‡jv †g‡b P‡j, Zv mn‡RB †evSv hvq :

(1) $\underline{u} = \underline{v}$
(2) $\underline{u} = \underline{v}$ n‡j $\underline{v} = \underline{u}$
(3) $\underline{u} = \underline{v}$ Ges $\underline{u} = \underline{w}$ n‡j $\underline{u} = \underline{w}$

$\underline{u}$-Gi avi‡K Ges $\underline{v}$-Gi avi‡K †iLvØq Awfbœ ev mgvšÍivj n‡j, Avgiv ms‡¶‡c eje †h $\underline{u}$ Ges $\underline{v}$ mgvšÍivj †f±i|
`ó‡e¨ t †h †Kvb we›`y †_‡K c`Ë †h †Kvb †f±‡ii mgvb K‡i GKwU †f±i Uvbv hvq|
†Kbbv, we›`y P Ges †f±i $\underline{u}$ †`Iqv _vK‡j, Avgiv P we›`y w`‡q $\underline{u}$ Gi avi‡Ki mgvšÍivj K‡i GKwU mij‡iLv Uvwb, Zvici P we›`y †_‡K $\underline{u}$ Gi w`K eivei ($\underline{u}$) Gi mgvb K‡i PQ †iLvsk †K‡U wbB| Zvn‡j A¼b Abhvqx $\overrightarrow{PQ} = \underline{u}$ nq|
wecixZ †f±i : $\underline{v}$ †K $\underline{u}$-Gi wecixZ †f±i ejv nq, hw`
(i) $|\underline{v}| = |\underline{u}|$

(ii) $\underline{v}$ -Gi avi‡K, $\underline{u}$ -Gi avi‡Ki m‡½ Awfbœ ev mgvšÍivj nq|

(iii) $\underline{v}$ -Gi w`K $\underline{u}$ -Gi w`‡Ki wecixZ nq|

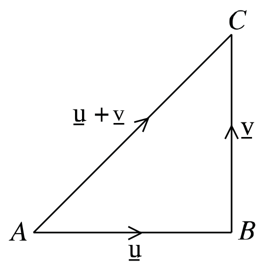
$\underline{v}$ hw` $\underline{u}$ Gi wecixZ †f±i nq, Z‡e $\underline{u}$ n‡e $\underline{v}$ Gi wecixZ †f±i| mgZvi msÁv †_‡K †evSv hvq †h, $\underline{v}$ Ges $\underline{w}$ c‡Z¨‡K $\underline{u}$ Gi wecixZ †f±i †evSv‡Z Ñ$\underline{u}$ nq|

$\underline{u} = \overrightarrow{AB}$ n‡j $-\underline{u} = \overrightarrow{BA}$

12.4| †f±‡ii †hvM I we‡qvM

1| (K) †f±i †hv‡Mi wÎfR wewa

†f±i †hv‡Mi msÁv t †Kvb $\underline{u}$ †f±‡ii cvšÍwe›` †_‡K Aci GKwU †f±i $\underline{v}$ AvKv n‡j $\underline{u}+\underline{v}$ Øviv Giƒc †f±i †evSvq hvi Aw`we›` $\underline{u}$ Gi Aw`we›` Ges hvi cvšÍwe›` $\underline{v}$ Gi cvšÍwe›`|

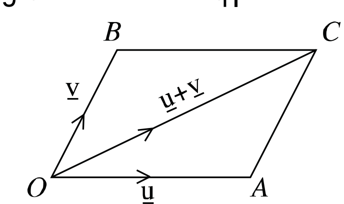


g‡b Kwi, $\overrightarrow{AB}=\underline{u}$, $\overrightarrow{BC}=\underline{v}$ Giƒc `BwU †f±i †h, $\underline{u}$ Gi cvšÍwe›` $\underline{v}$ Gi Aw`we›`| Zvn‡j $\underline{u}$ Gi Aw`we›` Ges $\underline{v}$ Gi cvšÍwe›` ms‡hvRK $\overrightarrow{AC}$ †f±i $\underline{u}$ I $\underline{v}$ †f±iØ‡qi mgwó ejv nq Ges $\underline{u}+\underline{v}$ Øviv m~wPZ nq|

$\underline{u}$ I $\underline{v}$ mgvšÍivj bv n‡j $\underline{u}$, $\underline{v}$ Ges $\underline{u}+\underline{v}$ †f±iÎq Øviv wÎfR Drcb nq e‡j Dc‡iv³ †hvRb c×wZ‡K wÎfR wewa ejv nq|

(L) †f±i †hv‡Mi mvgvšÍwiK wewa

†f±i †hv‡Mi wÎfR wewai Abywm×vš— wn‡m‡e †f±i †hv‡Mi mvgvšÍwiK wewa wbæiƒct †Kvb mvgvšÍwi‡Ki `BwU mwbwnZ evû Øviv `BwU †f±i $\underline{u}$ I $\underline{v}$ Gi gvb I w`K m~wPZ n‡j, H mvgvšÍwi‡Ki †h KY $\underline{u}$ I $\underline{v}$ †f±iØ‡qi mPK †iLvi †Q`we›`Mvgx Zv Øviv $\underline{u}+\underline{v}$ †f±‡ii gvb I w`K m~wPZ nq|



cgvY t g‡b Kwi, †h‡Kv‡bv we›` †_‡K AwâZ $\underline{u}$ Ges $\underline{v}$ †f±iØq $\overrightarrow{OA}$ Ges $\overrightarrow{OB}$ Øviv m~wPZ n‡q‡Q|

OACB mvgvšÍwiK I Zvi $\overrightarrow{OC}$ KY A¼b Kwi| Zvn‡j H mvgvšÍwi‡Ki OC KY Øviv $\underline{u}$ Ges $\underline{v}$ Gi †hvMdj m~wPZ n‡e|

A_©vr $\overrightarrow{OC}=\underline{u}+\underline{v}$ (†f±i ¯'vbvšÍ‡ii gva¨‡g)

OACB mvgvšÍwi‡Ki OB I AC mgvb I mgvšÍivj|

$\therefore \overrightarrow{AC}=\overrightarrow{OB}=\underline{v}$ (†f±i ¯'vbvšÍ‡ii gva¨‡g)

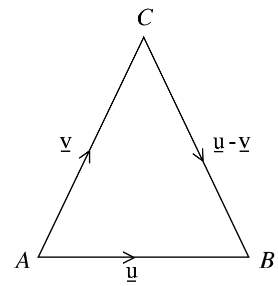
$\therefore\ \underline{u}+\underline{v}=\overrightarrow{OA}+\overrightarrow{OB}=\overrightarrow{OA}+\overrightarrow{AC}=\overrightarrow{OC}$ [wÎfR wewa Abmv‡i]

`óe¨ t (1) `B ev Z‡ZwaK †f±‡ii †hvMdj‡K Zv‡`i jwäI ejv nq| ej ev †e‡Mi jwä wbY‡qi †¶‡Î †f±i †hv‡Mi c×wZ AbmiY Ki‡Z nq|

(2) `BwU †f±i mgvšÍivj n‡j Zv‡`i †hv‡Mi †¶‡Î mvgvšÍwiK wewa c‡hvR¨ bq, wKš wÎfR wewa mKj †¶‡Î c‡hvR¨|

2| †f±‡ii we‡qvM t

$\underline{u}$ Ges $\underline{v}$ †f±iØ‡qi we‡qvMdj $\underline{u}$Ñ$\underline{v}$ ej‡Z $\underline{u}$ Ges (-$\underline{v}$) ($\underline{v}$ Gi wecixZ †f±i) †f±iØ‡qi †hvMdj $\underline{u}$+(Ñ$\underline{v}$) †evSvq|



†f±i we‡qv‡Mi wÎfR wewa

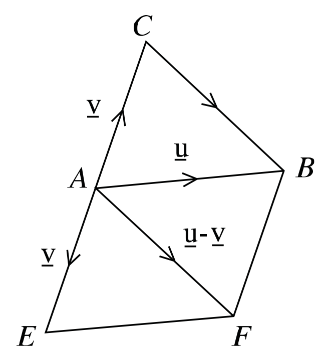
$\underline{u}=\overrightarrow{AB},\ \underline{v}=\overrightarrow{AC}$ n‡j $\underline{u}-\underline{v}=\overrightarrow{CB}$; A_vr $\overrightarrow{AB}-\overrightarrow{AC}=\overrightarrow{CB}$

K_vq t $\underline{u}$ Ges $\underline{v}$ Gi Aw`we›` GKB n‡j $\underline{u}$Ñ$\underline{v}$ †mB †f±i, hvi Aw`we›` n‡”Q $\underline{v}$ Gi Ašwe›` Ges hvi Ašwe›` n‡”Q $\underline{u}$ Gi Ašwe›`|

ms‡¶‡c t GKB Aw`we›`wewkó `BwU †f±‡ii we‡qvMdj n‡”Q Ašwe›`Øq Øviv wecixZµ‡g MwVZ †f±i|

cgvY t CA †iLvsk‡K Ggbfv‡e evwaZ Kwi †hb $AE=CA$ nq| $AEFB$ mvgvšÍwiK MVb Kwi| †f±i †hv‡Mi mvgvšÍwiK wewa Abhvqx, $\overrightarrow{AE}+\overrightarrow{AB}=\overrightarrow{AF}$

Avevi $AFBC$ GKwU mvgvšÍwiK, †Kbbv $BF=AE=CA$ Ges $BF \parallel AE$ e‡j $BF \parallel CA$.



$\therefore\ \overrightarrow{AF}=\overrightarrow{CB}$ †f±i ¯vbvšÍi,

wKš $\overrightarrow{AE}=-\underline{v}$ Ges $AB=\underline{u}$

mZivs $\underline{u}+(-\underline{v})=\overrightarrow{CB}$ cgwYZ nj|

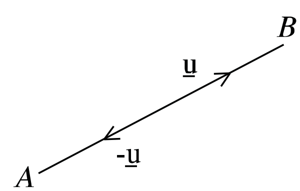
3| kb¨ †f±i : †h †f±‡ii ciggvb kb¨ Ges hvi w`K wbYq Kiv hvq bv Zv‡K kb¨ †f±i e‡j|

$\underline{u}$ †h †Kvb †f±i n‡j $\underline{u}$+(Ñ$\underline{u}$) Kx n‡e?

awi, $\underline{u}=\overrightarrow{AB}$ ZLb Ñ$\underline{u}=\overrightarrow{BA}$ d‡j

$\underline{u}$+(Ñ$\underline{u}$) = $\overrightarrow{AB}+\overrightarrow{BA}$

= $\overrightarrow{AA}$ (wÎfR wewa Abhvqx)



wKš $\overrightarrow{AA}$ Kx ai‡bi †f±i? GwU GKwU we›` †f±i, A_vr Gi Aw`we›` I Ašwe›` GKB we›`; mZivs ^`N¨ kb¨|

A_vr $\overrightarrow{AA}$ Øviv A we›`‡KB eS‡Z n‡e| Gi/c †f±i (hvi ˆ`N¨ kb¨) †K kb¨ †f±i ejv nq Ges $\underline{0}$ cZxK Øviv mwPZ Kiv nq| GB GKgvÎ †f±i hvi †Kvb wbw`ó w`K ev aviK †iLv †bB|

kb¨ †f±‡ii AeZviYvi d‡j Avgiv ej‡Z cwii †h, $\underline{u}+(-\underline{u})=\underline{0}$

Ges $\underline{u}+\underline{0}=\underline{0}+\underline{u}=\underline{u}$

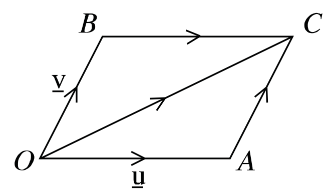
e¯Z kb¨ †f±‡ii m‡½ †k‡lv³ A‡f` wbwnZ i‡q‡Q|

12.5| †f±i †hv‡Mi wewamgn

1| †f±i †hv‡Mi wewbgq wewa (Commutative Law)

†h †Kvb $\underline{u}$, $\underline{v}$ †f±‡ii Rb¨ $\underline{u}+\underline{v}=\underline{v}+\underline{u}$

cgvY t g‡b Kwi, $\overrightarrow{OA}=\underline{u}$ Ges $\overrightarrow{OB}=\underline{v}$, $OACB$ mvgvšwiK I Zvi KY OC A¼b Kwi| OA I BC mgvb I mgvšivj Ges OB I AC mgvb I mgvšivj|



$\therefore \overrightarrow{OC}=\overrightarrow{OA}+\overrightarrow{AC}=\underline{u}+\underline{v}$

Avevi, $\overrightarrow{OC}=\overrightarrow{OB}+\overrightarrow{BC}=\overrightarrow{OB}+\overrightarrow{OA}=\underline{v}+\underline{u}$

$\therefore \underline{u}+\underline{v}=\underline{v}+\underline{u}$

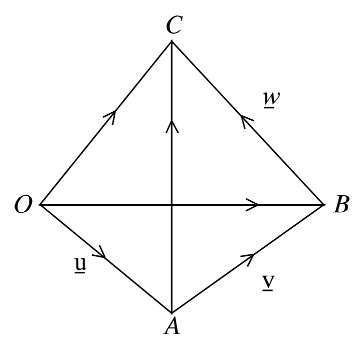
$\therefore$ †f±i †hvRb wewbgq wewa wm× K‡i|

†f±i †hvR‡bi ms‡hvM wewa (**Associative Law**)

†h †Kvb $\underline{u}, \underline{v}, \underline{w}$ Gi Rb¨ $(\underline{u}+\underline{v})+\underline{w}=\underline{u}+(\underline{v}+\underline{w})$

cgvY t g‡b Kwi, $\overrightarrow{OA}=\underline{u}$, $\overrightarrow{AB}=\underline{v}$, $\overrightarrow{BC}=\underline{w}$

A_vr $\underline{u}$ Gi cvšwe›` †_‡K $\underline{v}$ Ges $\underline{v}$ Gi cvšwe›` †_‡K $\underline{w}$ A¼b Kiv n‡q‡Q| O, C Ges A, C †hvM Kwi|



Zvn‡j $(\underline{u}+\underline{v})+\underline{w}=\left(\overrightarrow{OA}+\overrightarrow{AB}\right)+\overrightarrow{BC}$

$=\overrightarrow{OB}+\overrightarrow{BC}=\overrightarrow{OC}$

Avevi, $\underline{u}+(\underline{v}+\underline{w})=\overrightarrow{OA}+\left(\overrightarrow{AB}+\overrightarrow{BC}\right)$

$=\overrightarrow{OA}+\overrightarrow{AC}=\overrightarrow{OC}$

$\therefore (\underline{u}+\underline{v})+\underline{w}=\underline{u}+(\underline{v}+\underline{w})$

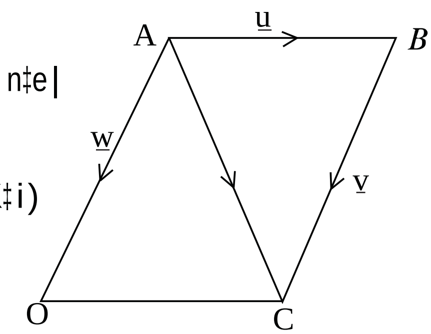
mZivs †f±i †hvRb ms‡hvM wewa wm× K‡i|

Abwm×všt †Kv‡bv wÎf‡Ri wZbwU evûi GKB µg Øviv mwPZ †f±iÎ‡qi †hvMdj kb¨|

Dc‡ii wP‡Î, $\overrightarrow{OB}+\overrightarrow{BA}=\overrightarrow{OA}=\left(-\overrightarrow{AO}\right)$

$\therefore \overrightarrow{OB}+\overrightarrow{BA}+\overrightarrow{AO}=\overrightarrow{OA}+\overrightarrow{AO}=-\overrightarrow{AO}+\overrightarrow{AO}=\vec{0}$

3| †f±i †hv‡Mi eRb wewa (Cancellation Law)



†h‡Kv‡bv $\underline{u}, \underline{v}, \underline{w}$ †f±‡ii Rb¨ $\underline{u}+\underline{v}=\underline{u}+\underline{w}$ n‡j, $\underline{v}=\underline{w}$ n‡e|

cgvY t GLv‡b $\underline{u}+\underline{v}=\underline{u}+\underline{w}$

$\therefore \underline{u}+\underline{v}+(-\underline{u})=\underline{u}+\underline{w}+(-\underline{u})$ (Dfqc‡¶ Ñ$\underline{u}$ †hvM K‡i)

ev, $\underline{u}-\underline{u}+\underline{v}=\underline{u}-\underline{u}+\underline{w}$

$\therefore \underline{v}=\underline{w}$

12.6| †f±‡ii msL¨v ¸wYZK ev †¯‹jvi ¸wYZK (**Scalar multiple of a vector**)

$\underline{u}$ †h‡Kv‡bv †f±i Ges m †h †Kvb ev¯e msL¨v n‡j $m\underline{u}$ Øviv †Kv‡bv †f±i †evSvq, wb‡P Zv e¨vL¨v Kiv nj|

(1) $m=0$ n‡j, $m\underline{u}=0$,

(2) $m\neq 0$ n‡j, $m\underline{u}$ Gi avviK $\underline{u}$ Gi avi‡Ki mv‡_ Awfb;

$m\underline{u}$ Gi ^`N¨ $\underline{u}$ Gi ^`‡N¨i (m) ¸Y Ges

(K) $m>0$ n‡j $m\underline{u}$ Gi w`K $\underline{u}$ Gi w`‡Ki ms‡M GKgLx

(L) $m<0$ n‡j $m\underline{u}$ Gi w`K $\underline{u}$ Gi w`‡Ki wecixZ|

`óe¨ t (1) $m=0$ A_ev $\underline{u}=\underline{0}$ n‡j

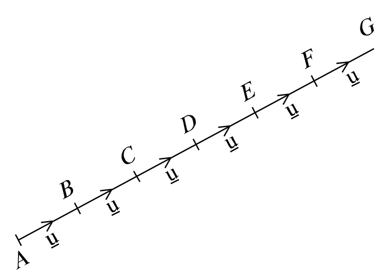
$m\underline{u}=\underline{0}$

(2) $1\underline{u}=\underline{u}$, $(-1)\underline{u}=-\underline{u}$

DcwiD³ msÁv n‡Z †`Lv hvq, $m(n\underline{u})=n(m\underline{u})=mn(\underline{u})$

mn Df‡q >0, Df‡q <0, GKwU >0 AciwU <0, GKwU ev Dfq 0, G mKj †¶Î I c_K c_K fv‡e we‡ePbv K‡i mn‡RB mÎwUi ev¯eZv m¤ú‡K wbwðZ nIqv hvq| wb‡P Gi GKwU D`vniY †`qv njt

g‡b Kwi $\overrightarrow{AB}=\overrightarrow{BC}=\underline{u}$



AC †K G chš Gi|‡c ewaZ Kwi †hb

$CD=DE=EF=FG=AB$ nq|

ZLb $\overrightarrow{AG}=\overrightarrow{AB}+\overrightarrow{BC}+\overrightarrow{CD}+\overrightarrow{DE}+\overrightarrow{EF}+\overrightarrow{FG}$

$=\underline{u}+\underline{u}+\underline{u}+\underline{u}+\underline{u}+\underline{u}=6\underline{u}$

Ab¨w`‡K $\overrightarrow{AG}=\overrightarrow{AC}+\overrightarrow{CE}+\overrightarrow{EG}$

$=2\underline{u}+2\underline{u}+2\underline{u}$

$=3(2\underline{u})$

Ges $\overrightarrow{AG}=\overrightarrow{AD}+\overrightarrow{DG}=3\underline{u}+3\underline{u}=2(3\underline{u})$

$\therefore 2(3\underline{u})=3(2\underline{u})=6\underline{u}$

`óe¨ t `BwU †f±‡ii avivK †iLv Awfb ev mgvšÍivj n‡j, G‡`i GKwU‡K AciwUi mvsL¨¸wYZK AvKv‡i cKvk Kiv hvq|

ev¯Í‡e $AB \| CD$ n‡j,

$$\overrightarrow{AB} = m\overrightarrow{CD}, \text{ †hLv‡b, } |m| = \frac{|\overrightarrow{AB}|}{|\overrightarrow{CD}|} = \frac{AB}{CD}$$

$m > 0$ n‡j, $\overrightarrow{AB}$ I $\overrightarrow{CD}$ mggLx nq,

$m < 0$ n‡j, $\overrightarrow{AB}$ I $\overrightarrow{CD}$ wecixZgLx nq|

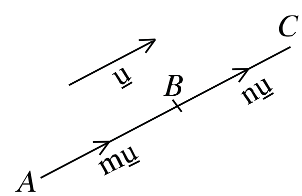
12.7| †f±‡ii mvsL¨¸wYZK msµvšÍ eÈb mÎ

(Distributive laws concerning scalar multiples of vectors)

m, n `BwU †¯‹jvi Ges $\underline{u}$, $\underline{v}$ `BwU †f±i n‡j,

(1) $(m + n)\,\underline{u} = m\,\underline{u} + n\,\underline{u}$

(2) $m(\underline{u} + \underline{v}) = m\,\underline{u} + m\,\underline{v}$



cgvY t (1) m ev n kb¨ n‡j mÎwU Aek¨B Lv‡U|

g‡b Kwi, m, n Df‡q abvZ¥K Ges $\overrightarrow{AB} = m\underline{u}$

$\therefore |\overrightarrow{AB}| = m\,|\underline{u}|$

AB †K C chš— ewaZ Kwi †hb $|\overrightarrow{BC}| = n|\underline{u}|$ nq|

$\therefore \overrightarrow{BC} = n\underline{u}$ Ges

$$|\overrightarrow{AC}| = |\overrightarrow{AB}| + |\overrightarrow{BC}| = m|\underline{u}| + n|\underline{u}| = (m+n)|\underline{u}|$$

$\therefore \overrightarrow{AC} = (m+n)\underline{u}$

wKš' $\overrightarrow{AC} = \overrightarrow{AB} + \overrightarrow{BC}$

$\therefore m\underline{u} + n\underline{u} = (m+n)\,\underline{u}$

m, n Df‡q FYvZ¥K n‡j $(m+n)\underline{u}$ Gi ˆ`N¨ n‡e

$|m+n||\underline{u}|$ Ges w`K n‡e $\underline{u}$ Gi w`‡Ki wecixZ w`K, ZLb $m\underline{u} + n\underline{u}$ †f±iwUi ˆ`N¨ n‡e

$|m||\underline{u}| + |n||\underline{u}| = (|m| + |n|)\,|\underline{u}|$ [$\because m\underline{u}, n\underline{u}$ †f±iØq GKB w`‡K KvR K‡i ] Ges w`K n‡e $\underline{u}$ Gi wecixZ w`K| wKš' $m < 0$ Ges $n < 0$ nIqvq $|m| + |n| = |m+n|$, †m‡nZ G‡¶‡Î $(m+n)\underline{u} = m\underline{u} + n\underline{u}$ cvIqv †Mj|

me‡k‡l m Ges n Gi g‡a¨ c_gwU > 0, AciwU < 0 n‡j $(m+n)\underline{u}$ Gi ˆ`N¨ n‡e $|m+n||\underline{u}|$ Ges w`K n‡e

(K) $\underline{u}$ Gi w`‡Ki mv‡_ GKgLx hLb $|m| > |n|$

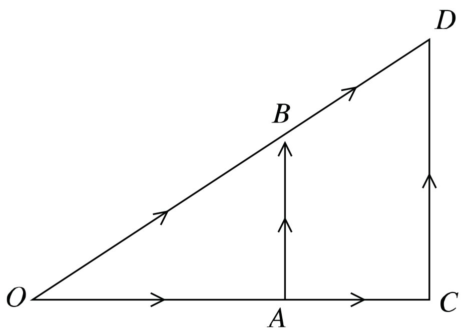
(L) $\underline{u}$ Gi wecixZ w`K hLb $|m| < |n|$

ZLb $m\underline{u} + n\underline{u}$ †f±iwU I ˆ`‡N¨ I w`‡K $(m+n)\underline{u}$ Gi mv‡_ GKgLx n‡e|

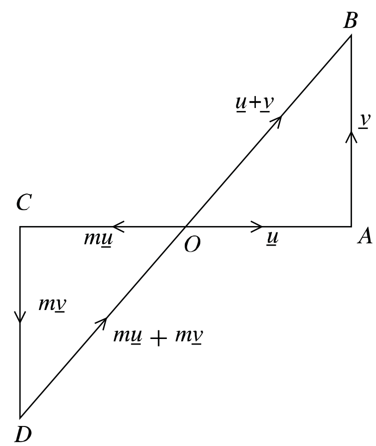
`óe¨ t wZbwU we›` A, B, C mg‡iL n‡e hw` Ges †Kej hw` $\overrightarrow{AC}, \overrightarrow{AB}$ Gi mvsL¨¸wYZK nq|

gšše¨t (1) `BwU †f±‡ii aviK †iLv Awfb A_ev mgvšÍivj n‡j Ges Zv‡`i w`K GKB n‡j, Zv‡`i m`k (similar) †f±i ejv nq|

(2) †h †f±‡ii ˆ`N¨ 1 GKK, Zv‡K (w`K wb‡`kK) GKK †f±i ejv nq|



wPÎ-1



wPÎ-2

g‡b Kwi, $\overrightarrow{OA} = \underline{u}$, $\overrightarrow{AB} = \underline{v}$

Zvn‡j $\overrightarrow{OB} = \overrightarrow{OA} + \overrightarrow{AB} = \underline{u} + \underline{v}$

OA †K C chš— ewaZ Kwi †hb $OC = m.OA$ nq| C we›` w`‡q Aw¼Z AB Gi mgvšÍivj CD †iLv OB Gi ewaZvsk‡K D we›`‡Z †Q` K‡i| †h‡nZ OAB Ges OCD wÎfRØq m`k,

†m‡nZ $\dfrac{|\overrightarrow{OC}|}{|\overrightarrow{OA}|} = \dfrac{|\overrightarrow{CD}|}{|\overrightarrow{AB}|} = \dfrac{|\overrightarrow{OD}|}{|\overrightarrow{OB}|} = m$

$\therefore \overrightarrow{CD} = m\overrightarrow{AB} = m\underline{v}$

wPÎÑ1 G m abvZ¥K, wPÎ-2 G m FYvZ¥K

$\therefore$ $OC = m.\,OA,\ CD = m.\,AB,\ OD = m.OB$

G¶‡Y $\overrightarrow{OC} + \overrightarrow{CD} = \overrightarrow{OD}$ ev, $m\left(\overrightarrow{OA}\right) + m\left(\overrightarrow{AB}\right) = m\left(\overrightarrow{OB}\right)$

$\therefore m\underline{u} + m\underline{v} = m(\underline{u} + \underline{v})$

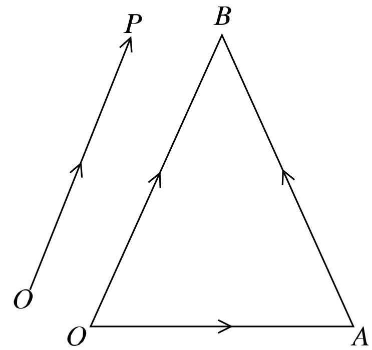
`óe¨ t m Gi mKj gv‡bi Rb¨ Dc‡iv³ mÎ mZ¨|

KvR: m I n Gi wewfbœ cKvi mvswL¨K gvb wb‡q $\underline{u}$ †f±‡ii Rb¨ $(m+n)\underline{u} = m\underline{u} + n\underline{u}$ mÎwU hvPvB Ki|

12.8| Ae¯’vb †f±i (Position Vector)

mgZj¯’ †Kv‡bv wbw`©ó O we›`yi mv‡c‡¶ H mgZ‡ji †h‡Kv‡bv P we›`yi Ae¯’vb $\overrightarrow{OP}$ Øviv wbw`©ó Kiv hvq| $\overrightarrow{OP}$ †K O we›`yi mv‡c‡¶ P we›`yi Ae¯’vb †f±i ejv nq Ges O we›`y‡K †f±‡ii gyjwe›`y (origin) ejv nq|

g‡b Kwi, †Kv‡bv mgZ‡j O GKwU wbw`©ó we›`y Ges GKB mgZ‡j A Aci GKwU we›`y| O, A †hvM Ki‡j Drcb $\overrightarrow{OA}$ †f±i O we›`yi cwi‡cw¶‡Z A we›`yi Ae¯’vb †f±i ejv nq| Abiƒcfv‡e, GKB O we›`yi †cw¶‡Z GKB mgZ‡j Aci B we›`yi Ae¯’vb †f±i $\overrightarrow{OB}$. A, B †hvM Kwi|



g‡b Kwi, $\overrightarrow{OA} = \underline{a}$, $\overrightarrow{OB} = \underline{b}$

Zvn‡j $\overrightarrow{OA} + \overrightarrow{AB} = \overrightarrow{OB}$ A_©vr $\underline{a} + \overrightarrow{AB} = \underline{b}$

$\therefore \overrightarrow{AB} = \underline{b} - \underline{a}$

mZivs, `ywU we›`yi Ae¯’vb †f±i Rvbv _vK‡j Zv‡`i ms‡hvRK †iLv Øviv m~wPZ †f±i H †f±‡ii cÖvšÍwe›`yi Ae¯’vb †f±i †_‡K Avw`we›`yi Ae¯’vb †f±i we‡qvM K‡i cvIqv hv‡e|

`óe¨ : gyjwe›`y wfbœ wfbœ Ae¯’v‡b _vK‡j GKB we›`yi Ae¯’vb †f±i wfbœ wfbœ n‡Z cv‡i| †Kvb wbw`©ó cwZcv`¨ welq‡i mgvav‡b G welq‡i we‡ePbvaxb mKj we›`yi Ae¯’vb †f±i GKB gyjwe›`yi mv‡c‡¶ aiv nq|

KvR : ‡Zvgvi LvZvq GKwU we›`y‡K gyjwe›`y O a‡i wewfbœ Ae¯’v‡b Avil cvPwU we›`y wb‡q O we›`yi mv‡c‡¶ G¸‡jvi Ae¯’vb †f±i wPwýZ Ki|

12.9| KwZcq D`vniY

D`vniY 1| †`LvI †h, (K) $-\left(-\underline{a}\right)=\underline{a}$

(L) $-m\left(\underline{a}\right)=m\left(-\underline{a}\right)=-m\underline{a}$, m GKwU †¯‹jvi|

(M) $\dfrac{\text{a}}{|\underline{a}|}$ GKwU GKK †f±i, hLb $\underline{a} \neq \underline{0}$

mgvavb t (K) wecixZ †f±‡ii ag Abhvqx $\underline{a} + \left(-\underline{a}\right)= 0$

Avevi $\left(-\underline{a}\right)+\left(-\left(-\underline{a}\right)\right)=\underline{0}$

$\therefore -\left(-\underline{a}\right)+\left(-\underline{a}\right)=\underline{a}+\left(-\underline{a}\right)$

$\therefore -\left(-\underline{a}\right)=\underline{a}$ [†f±i †hv‡Mi eRbwewa]

(L) $m\underline{a}+\left(-m\right)\underline{a}=\left\{m+\left(-m\right)\right\}\underline{a}=0\underline{a}=\underline{0}$

$\therefore \left(-m\right)\underline{a}=-m\underline{a} \quad (1)$

Avevi $m\underline{a}+m(-\underline{a})=m\,[\underline{a}+(-\underline{a})]=m\underline{0}=\underline{0}$

$\therefore m(-\underline{a})=-m\underline{a} \qquad (2)$

(1) Ges (2) †_‡K $(-m)\underline{a}=m(-\underline{a})=-m\underline{a}$

(M) g‡b Kwi $\underline{a}$ Akb¨ $\underline{\hat{a}}$ nq| †f±‡ii w`K eivei $\underline{\hat{a}}$ GKwU GKK †f±i Ges $\underline{a}$ †f±‡ii ˆ`N¨ a A_vr

$|\underline{a}|=a$

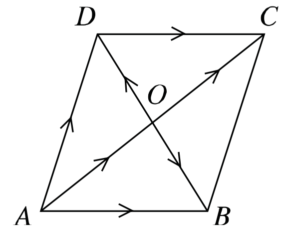
Zvn‡j $\underline{a}=(a)\ \underline{\hat{a}}=|\underline{a}|\ \underline{\hat{a}}$; GLv‡b $|\underline{a}|=a$ GKwU †¯‹jvi hv Akb¨ KviY $\underline{a}\neq\underline{0}$

$\therefore \frac{\underline{a}}{|\underline{a}|}=\frac{|\underline{a}|\underline{\hat{a}}}{|\underline{a}|}=\underline{\hat{a}}$ GKwU GKK †f±i|

D`vniY 2| $ABCD$ GKwU mvgvšwiK hvi KYØq AC I BD|

(K) $\overrightarrow{AC},\ \overrightarrow{BD}$ †f±iØq‡K $\overrightarrow{AB}$ Ges $\overrightarrow{AD}$ †f±iØ‡qi gva¨‡g cKvk Ki|

(L) $\overrightarrow{AB}$ Ges $\overrightarrow{AD}$ †f±iØq‡K $\overrightarrow{AC}$ I $\overrightarrow{BD}$ †f±iØ‡qi gva¨‡g cKvk Ki|



mgvavb t (K) $\overrightarrow{AC}=\overrightarrow{AD}+\overrightarrow{DC}=\overrightarrow{AD}+\overrightarrow{AB}$

Avevi, $\overrightarrow{AB}+\overrightarrow{BD}=\overrightarrow{AD}$ ev $\overrightarrow{BD}=\overrightarrow{AD}-\overrightarrow{AB}$

(L) †h‡nZ mvgvšwi‡Ki KYØq ci¯úi mgwØLwÛZ nq|

$$\overrightarrow{AB}=\overrightarrow{AO}+\overrightarrow{OB}=\frac{1}{2}\overrightarrow{AC}+\frac{1}{2}\overrightarrow{DB}=\frac{1}{2}\overrightarrow{AC}-\frac{1}{2}\overrightarrow{BD}$$

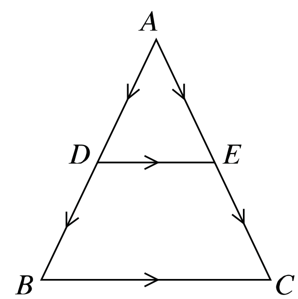
$$\overrightarrow{AD}=\overrightarrow{AO}+\overrightarrow{OD}=\frac{1}{2}\overrightarrow{AC}+\frac{1}{2}\overrightarrow{BD}.$$

D`vniY 3| †f±‡ii mvnv‡h¨ cgvY Ki †h, wÎf‡Ri †h‡Kv‡bv `B evûi ga¨we›`Ø‡qi ms‡hvRK †iLvsk H wÎf‡Ri ZZxq evûi mgvšivj I Zvi A‡aK|

mgvavb t g‡b Kwi, ABC wÎf‡Ri AB I AC evûØ‡qi ga¨we›` h_vµ‡g D I E. D, E †hvM Kwi|

cgvY Ki‡Z n‡e †h, $DE \parallel BC$ Ges $DE=\frac{1}{2}BC$

†f±i we‡qv‡Mi wÎfRwewa Abmv‡i,



$\overrightarrow{AE}-\overrightarrow{AD}=\overrightarrow{DE}$................(1)

Ges $\overrightarrow{AC}-\overrightarrow{AB}=\overrightarrow{BC}$

wKš $\overrightarrow{AC}=2\overrightarrow{AE},\ \overrightarrow{AB}=2\overrightarrow{AD}$

[$\because D$ I E h_vµ‡g AB I AC evûi ga¨we›`]

$\therefore \overrightarrow{AC}-\overrightarrow{AB}=\overrightarrow{BC}$ †_‡K cvB

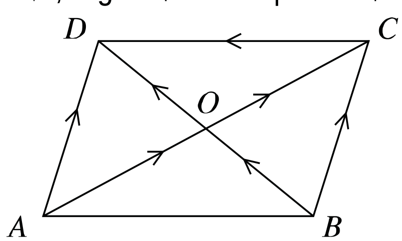
$2\overrightarrow{AE}-2\overrightarrow{AD}=\overrightarrow{BC}$ A_vr $2\left(\overrightarrow{AE}-\overrightarrow{AD}\right)=\overrightarrow{BC}$

ev, $2\overrightarrow{DE} = \overrightarrow{BC}$, [(1) n‡Z]

$\therefore \overrightarrow{DE} = \frac{1}{2}\overrightarrow{BC}$

Aevi $\left|\overrightarrow{DE}\right| = \frac{1}{2}\left|\overrightarrow{BC}\right|$ ev $DE = \frac{1}{2}BC$ mZivs $\overrightarrow{DE}$ I $\overrightarrow{BC}$ †f±iØ‡qi avi K †iLv GKB ev mgvšÍivj| wKš‘ GLv‡b avi K †iLv GK bq| mZivs $\overrightarrow{DE}$ I $\overrightarrow{BC}$ †f±iØ‡qi avi K †iLvØq A_vr DE Ges BC mgvšÍivj|

D`vniY 4| †f±i c×wZ‡Z cgvY Ki †h, mvgvšÍwi‡Ki KYØq ci¯úi‡K mgwØLwÛZ K‡i|



mgvavb t g‡b Kwi, $ABCD$ mvgvšÍwi‡Ki AC I BD KYØq ci¯úi‡K O we›`‡Z †Q` K‡i‡Q|

g‡b Kwi, $\overrightarrow{AO} = \underline{a}, \overrightarrow{BO} = \underline{b}, \overrightarrow{OC} = \underline{c}, \overrightarrow{OD} = \underline{d}$

cgvY Ki‡Z n‡e †h, $|\underline{a}| = |\underline{c}|, |\underline{b}| = |\underline{d}|$

cgvY t $\overrightarrow{AO} + \overrightarrow{OD} = \overrightarrow{AD}$ Ges $\overrightarrow{BO} + \overrightarrow{OC} = \overrightarrow{BC}$

mvgvšÍwi‡Ki wecixZ evûØq ci¯úi mgvb I mgvšÍivj| $\therefore \overrightarrow{AD} = \overrightarrow{BC}$

A_vr $\overrightarrow{AO} + \overrightarrow{OD} = \overrightarrow{BO} + \overrightarrow{OC}$

ev, $\underline{a} + \underline{d} = \underline{b} + \underline{c}$

A_vr $\underline{a} - \underline{c} = \underline{b} - \underline{d}$ [Dfq c‡¶ $-\underline{c} - \underline{d}$ ‡hvM K‡i]

GLv‡b $\underline{a}$ I $\underline{c}$ Gi avi K AC, $\therefore \underline{a} - \underline{c}$ Gi avi K AC.

$\underline{b}$ I $\underline{d}$ Gi avi K BD, $\therefore \underline{b} - \underline{d}$ Gi avi K BD.

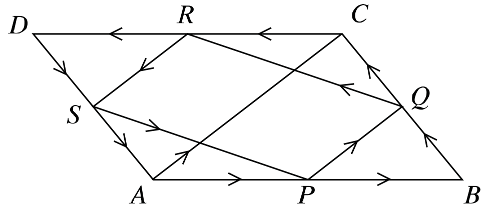
$\underline{a} - \underline{c}$ I $\underline{b} - \underline{d}$ `BwU mgvb mgvb Akb¨ †f±i n‡j Zv‡`i avi K †iLv GKB A_ev mgvšÍivj n‡e| wKš‘ AC I BD `BwU ci¯úi‡Q`x Amgvšivj mij‡iLv| mZivs $\underline{a} - \underline{c}$ I $\underline{b} - \underline{d}$ †f±iØq Akb¨ n‡Z cv‡i bv weavq G‡`i gvb kb¨ n‡e|

$\therefore \underline{a} - \underline{c} = \underline{0}$ ev $\underline{a} = \underline{c}$ Ges $\underline{b} - \underline{d} = \underline{0}$ ev $\underline{b} = \underline{d}$

$\therefore |\underline{a}| = |\underline{c}|$ Ges $|\underline{b}| = |\underline{d}|$

A_vr mvgvšÍwi‡Ki KYØq ci¯úi‡K mgwØLwÛZ K‡i|

D`vniY 5| †f±i c×wZ‡Z cgvY Ki †h, †Kv‡bv PZf©‡Ri mwbwnZ evû¸‡jvi ga¨we›`y mg‡hvRK †iLvmgn GKwU mvgvšÍwiK Drcb K‡i|



mgvavb t g‡b Kwi, ABCD PZf‡Ri evû¸‡jvi ga¨we›` P, Q, R, S| P I Q, Q I R, R I S Ges S I P †hvM Kwi| cgvY Ki‡Z n‡e †h, PQRS GKwU mvgvšÍwiK|

cgvY t g‡b Kwi, $\overrightarrow{AB} = \underline{a}$, $\overrightarrow{BC} = \underline{b}$, $\overrightarrow{CD} = \underline{c}$, $\overrightarrow{DA} = \underline{d}$

Zvn‡j, $\overrightarrow{PQ} = \overrightarrow{PB} + \overrightarrow{BQ} = \frac{1}{2}\overrightarrow{AB} + \frac{1}{2}\overrightarrow{BC} = \frac{1}{2}(\underline{a} + \underline{b})$

Abi|cfv‡e, $\overrightarrow{QR} = \frac{1}{2}(\underline{b} + \underline{c})$, $\overrightarrow{RS} = \frac{1}{2}(\underline{c} + \underline{d})$ Ges $\overrightarrow{SP} = \frac{1}{2}(\underline{d} + \underline{a})$

wKš‘ $(\underline{a} + \underline{b}) + (\underline{c} + \underline{d}) = \overrightarrow{AC} + \overrightarrow{CA} = \overrightarrow{AC} - \overrightarrow{AC} = \vec{0}$

A_©vr $\underline{a} + \underline{b} = -(\underline{c} + \underline{d})$

$\overrightarrow{PQ} = \frac{1}{2}(\underline{a} + \underline{b}) = -\frac{1}{2}(\underline{c} + \underline{d}) = -\overrightarrow{RS} = \overrightarrow{SR}$

$\therefore$ PQ Ges SR mgvb I mgvšÍivj|

Abi|cfv‡e, QR Ges PS mgvb I mgvšÍivj|

$\therefore$ PQRS GKwU mvgvšÍwiK|

AbkxjbxÑ12

1| AB || DC n‡j

i $\overrightarrow{AB} = m.\overrightarrow{DC}$, †hLv‡b m GKwU †¯‹jvi iwk

ii $\overrightarrow{AB} = \overrightarrow{DC}$

iii $\overrightarrow{AB} = \overrightarrow{CD}$

Ic‡ii evK¨¸‡jvi g‡a¨ †KvbwU mwVK?

K. i L. ii

M. i I ii N. i, ii I iii

2| `wU †f±i mgvšÍivj n‡j-

i G‡`i †hv‡Mi †¶‡Î mvgvšÍwiK wewa c‡hvR¨

ii G‡`i †hv‡Mi †¶‡Î wÎfR wewa c‡hvR¨

iii G‡`i ^`N¨ me`v mgvb

Ic‡ii evK¨¸‡jvi g‡a¨ †KvbwU mwVK?

K. i L. ii

M. i I ii N. i, ii I iii

3| AB = CD Ges AB || CD n‡j †KvbwU mwVK?

K. $\overrightarrow{AB} = \overrightarrow{CD}$ L. $\overrightarrow{AB} = m.\overrightarrow{CD}$ †hLv‡b m>1

M. $\overrightarrow{AB} + \overrightarrow{DC}$ <O N. $\overrightarrow{AB} + m.\overrightarrow{CD}$ = O †hLv‡b m>1 wb‡Pi Z‡_¨i Av‡jv‡K 4 I 5 b¤i c‡ki DËi `vI:

AB †iLvs‡ki Dci ‡h‡Kv‡bv we›` C Ges †Kv‡bv †f±i gjwe›`i mv‡c‡¶ A, B I C we›`i Ae¯vb †f±i h_vµ‡g $\underline{a}$, $\underline{b}$, I $\underline{c}$ |

4| C we›`wU AB †iLvsk‡K 2:3 Abcv‡Z Ašwef³ Ki‡j wb‡Pi †KvbwU mwVK?

K. $\underline{c}=\dfrac{\underline{a}+2\underline{b}}{5}$ L. $\underline{c}=\dfrac{2\underline{a}+\underline{b}}{5}$

M. $\underline{c}=\dfrac{3\underline{a}+2\underline{b}}{5}$ N. $\underline{c}=\dfrac{2\underline{a}+3\underline{b}}{5}$

5| †f±i gjwe›`wU O n‡j wb‡Pi †KvbwU mwVK?

K. $\overrightarrow{OA}=\underline{a}-\underline{b}$ L. $\overrightarrow{OA}+\overrightarrow{OC}=\overrightarrow{AC}$

M. $\overrightarrow{AB}=\underline{b}-\underline{a}$ N. $\overrightarrow{OC}=\underline{c}-\underline{b}$

6| ABC wÎf‡Ri BC, CA, AB evûÎ‡qi ga¨we›` h_vµ‡g D, E, F n‡j,

(K) $\overrightarrow{BC}, \overrightarrow{AD}, \overrightarrow{BE}, \overrightarrow{CF}$ †f±i¸‡jv‡K $\overrightarrow{AB}$ Ges $\overrightarrow{AC}$ †f±‡ii gva¨‡g cKvk Ki|

(L) $\overrightarrow{BC}, \overrightarrow{CA}, \overrightarrow{AD}$ †f±i¸‡jv‡K $\overrightarrow{BE}$ Ges $\overrightarrow{CF}$ †f±‡ii gva¨‡g cKvk Ki|

7| ABCD mvgvšwi‡Ki KYØq $\overrightarrow{AC}$ I $\overrightarrow{BD}$ n‡j $\overrightarrow{AB}$ I $\overrightarrow{AC}$ †f±iØq‡K $\overrightarrow{AD}$ I $\overrightarrow{BD}$ †f±iØ‡qi gva¨‡g cKvk Ki Ges †`LvI †h, $\overrightarrow{AC}+\overrightarrow{BD}=2\overrightarrow{BC}$ Ges $\overrightarrow{AC}-\overrightarrow{BD}=2\overrightarrow{AB}$

8| †`LvI †h, (K) $-(\underline{a}+\underline{b})=-\underline{a}-\underline{b}$

(L) $\underline{a}+\underline{b}=\underline{c}$ n‡j $\underline{a}=\underline{c}-\underline{b}$

9| †`LvI †h (K) $\underline{a}+\underline{a}=2\underline{a}$ (L) $(m-n)\underline{a}=m\underline{a}-n\underline{a}$

(M) $m(\underline{a}-\underline{b})=m\underline{a}-m\underline{b}$

10|(K) $\underline{a}, \underline{b}$ c‡Z¨‡K Akb¨ †f±i n‡j †`LvI †h, $\underline{a}=m\underline{b}$ n‡Z cv‡i †KejgvÎ hw` a,b Gi mgvšivj nq|

(L) $\underline{a}, \underline{b}$ Akb¨ Amgvšivj †f±i Ges $m\underline{a}+n\underline{b}=0$ n‡j †`LvI †h, $m=n=0$

11| A,B,C,D we›`¸‡jvi Ae¯vb †f±i h_vµ‡g $\underline{a}, \underline{b}, \underline{c}, \underline{d}$ n‡j †`LvI †h, ABCD mvgvšwiK n‡e hw` Ges †KejgvÎ hw` $\underline{b}-\underline{a}=\underline{c}-\underline{d}$ nq|

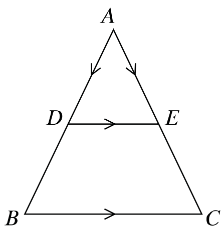
12|†f±‡ii mvnv‡h¨ cgvY Ki †h, wÎf‡Ri GK evûi ga¨we›` †_‡K Aw¼Z Aci evûi mgvšivj †iLv ZZxq evûi ga¨we›`Mvgx|

13|cgvY Ki †h, †Kv‡bv PZf‡Ri KYØq ci¯úi‡K mgwØLwÛZ Ki‡j Zv GKwU mvgvšwiK nq|

14|†f±‡ii mvnv‡h¨ cgvY Ki †h, UwcwRqv‡gi Amgvšivj evûØ‡qi ga¨we›`i ms‡hvRK mij‡iLv mgvšivj evûØ‡qi mgvšivj I Zv‡`i †hvMd‡ji A‡aK|

15| †f±‡ii mvnv‡h¨ cgvY Ki †h, UwcwRqv‡gi KY©Ø‡qi ga¨we›`i ms‡hvRK mij‡iLv mgvšÍivj evûØ‡qi mgvšÍivj Ges Zv‡`i we‡qvMd‡ji A‡aK|

16|



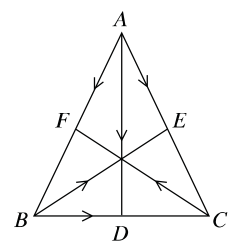
ΔABC Gi AB I AC evûi ga¨we›` h_vµ‡g D I E

K. $(\overrightarrow{AD} + \overrightarrow{DE})$ †K $\overrightarrow{AC}$ †f±‡ii gva¨‡g cKvk Ki|

L. ‡f±‡ii mvnv‡h¨ cgvY Ki †h, AB| |DC Ges $DE = \frac{1}{2}BC$

M. ABCD UwcwRqv‡gi KY©Ø‡qi ga¨we›` h_vµ‡g M I N n‡j †f±‡ii mvnv‡h¨ cgvY Ki †h, MN| |DE| |BC Ges $MN = \frac{1}{2}(BC\text{-}DC)$

17| ΔABC Gi BC, CA I AB evûi ga¨we›` h_vµ‡g D, E I F



K. $\overrightarrow{AB}$ †f±i‡K $\overrightarrow{BE}$ I $\overrightarrow{CF}$ †f±‡ii gva¨‡g cKvk Ki|

L. cgvY Ki †h, $\overrightarrow{AD} + \overrightarrow{BE} + \overrightarrow{CF} = \underline{O}$

M. †f±‡ii mvnv‡h¨ cgvY Ki †h, F we›` w`‡q AwN¼Z BC Gi mgvšÍivj †iLv Aek¨B E we›`Mvgx n‡e|

Î‡qv`k Aa¨vq

Nb R¨vwgwZ

Avgv‡`i ev¯—e Rxe‡b wewfbœ AvKv‡ii Nbe¯—i c‡qvRb I Zvi e¨enviI me`vB n‡q Zv‡K| Gi g‡a¨ mIg I welg AvKv‡ii Nbe¯— Av‡Q| mIg AvKv‡ii Nbe¯— Ges `BwU mIg Nbe¯—i mgš^‡q MwVZ †hŠwMK Nbe¯—i AvqZb I côZ‡ji ‡¶Îdj wbYq c×wZ GB Aa¨v‡q Av‡jvPbv Kiv n‡e|

Aa¨vq †k‡l wk¶v_xiv

- Nbe¯—i cZxKxq wPÎ A¼b Ki‡Z cvi‡e|
- wcRg, wcivwgW AvKwZi e¯—, †MvjK I mgeËfwgK †KvY‡Ki AvqZb Ges côZ‡ji †¶Îdj wbYq Ki‡Z cvi‡e|
- Nb R¨vwgwZi avi Yv c‡qvM K‡i mgm¨v mgvavb Ki‡Z cvi‡e|
- ‡hŠwMK Nbe¯—i AvqZb I côZ‡ji ‡¶Îdj cwigvc Ki‡Z cvi‡e|
- Nb R¨vwgwZi avi Yv e¨enwiK †¶‡Î c‡qvM Ki‡Z cvi‡e|

13.1 †gŠwjK avi Yv

gva¨wgK R¨vwgwZ‡Z we›`, †iLv I Z‡ji †gŠwjK avi Yv Av‡jwPZ n‡q‡Q| Nb R¨vwgwZ‡ZI we›`, †iLv I Zj †gŠwjK avi Yv wn‡m‡e MnY Kiv nq|

1| e¯—i ˆ`N¨, c¯— I D"PZv c‡Z¨KwU‡K H e¯—i gvÎv (dimension) ejv nq|

2| we›`i ˆ`N¨, c¯— I D"PZv †bB| GwU GKwU avi Yv| ev¯—‡e †evSvi R‡b¨ Avgiv GKwU WU (.) e¨envi Kwi| G‡K Ae¯—v‡bi cwZiƒc ejv †h‡Z cv‡i| mZivs we›`i †Kvb gvÎv †bB| ZvB we›` kb¨ gwÎK|

3| †iLvi †Kejgvî ˆ`N¨ Av‡Q, c¯— I D"PZv †bB| ZvB †iLv GKgwÎK|

4| Z‡ji ˆ`N¨ I c¯— Av‡Q, D"PZv †bB| ZvB Zj wØgwÎK|

5| †h e¯—i ˆ`N¨, c¯— I D"PZv Av‡Q, Zv‡K Nbe¯— ejv nq| mZivs Nbe¯— wÎgwÎK|

13.2 KwZcq cv_wgK msÁv

1| mgZj (Plane surface) t †Kv‡bv Z‡ji Dci¯— †h †Kv‡bv `BwU we›`i ms‡hvRK mij‡iLv m¤úYi{‡c H Z‡ji Dci Aew¯—Z n‡j, H Zj‡K mgZj ejv nq| cK‡ii cwb ¯—i _vK‡j H cwbi DcwifvM GKwU mgZj| wm‡g›U w`‡q wbwgZ ev †gvRvBKZ N‡ii †g‡S‡K Avgiv mgZj e‡j _wK| wKš' R¨vwgwZKfv‡e Zv mgZj bq, KviY N‡ii †g‡S‡Z wKQ DP-wbP _v‡KB|

`óe¨ t Ab¨ wKQ D‡jL bv _vK‡j Nb R¨vwgwZ‡Z †iLv ev ˆ`N¨ Ges Z‡ji we¯—vi Amxg (infinite) ev Awbw`ó g‡b Kiv nq| mZivs Z‡ji msÁv †_‡K Abgvb Kiv hvq †h, †Kv‡bv mij †iLvi GKwU Ask †Kv‡bv Z‡ji Dci _vK‡j Aci †Kv‡bv Ask H Z‡ji evB‡i _vK‡Z cv‡i bv|

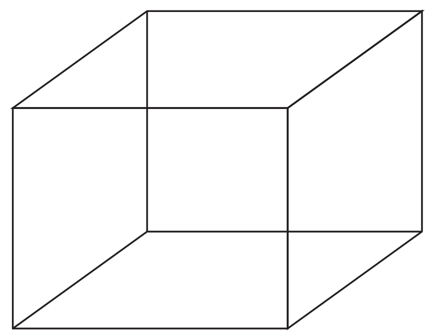
2| eµZj (Curved surface) t †Kv‡bv Z‡ji Dci Aew¯—Z †h‡Kv‡bv `BwU we›`i ms‡hvRK mij‡iLv m¤úYi{‡c H Z‡ji Dci Aew¯—Z bv n‡j, H Zj‡K eµZj ejv nq| †Mvj‡Ki côZj GKwU eµZj|

3| Nb R¨wgwZ (Solid geometry) t MwYZ kv‡¯¿i †h kvLvi mvnv‡h¨ Nbe¯‘ Ges Zj, †iLv I we›`yi ag Rvbv hvq, Zv‡K Nb R¨wgwZ ejv nq| KLbI KLbI G‡K RvMwZK R¨wgwZ (Geometry of space) ev wÎgvwÎK R¨wgwZI (Geometry of three dimensions) ejv nq|

4| GKZjxq †iLv (Coplanar straight lines) t GKvwaK mij‡iLv GKB mgZ‡j Aew¯'Z n‡j, ev Zv‡`i mK‡ji ga¨ w`‡q GKwU mgZj A¼b m¤¢e n‡j H mij‡iLv¸‡iv‡K GKZjxq ejv nq|

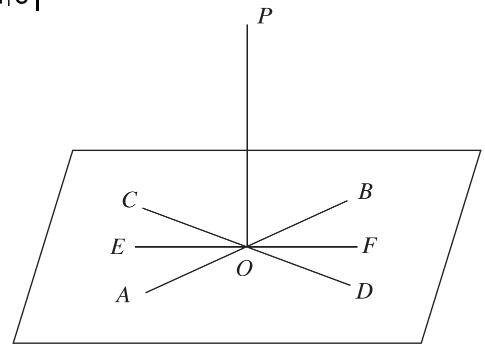
5| ˆbKZjxq †iLv (Skew or non coplanar lines) t GKvwaK mij‡iLv GKB mgZ‡j Aew¯'Z bv n‡j ev Zv‡`i ga¨ w`‡q GKwU mgZj A¼b Kiv m¤¢e bv n‡j G¸‡jv‡K ˆbKZjxq mij‡iLv ejv nq| `BwU †cwÝj‡K GKwUi Dci Avi GKwU w`‡q †hvM ev ¸YwPý AvKwZi GKwU e¯' %Zwi Ki‡jB `BwU ˆbKZjxq mij‡iLv Drcb n‡e|

6| mgvšÍivj mij‡iLv (Parallel line) t `BwU GKZjxq mij‡iLv hw` ci¯úi †Q` bv K‡i A_vr hw` Zv‡`i †Kv‡bv mvavi Y we›` bv _v‡K, Z‡e Zv‡`i mgvšÍivj mij‡iLv ejv nq|



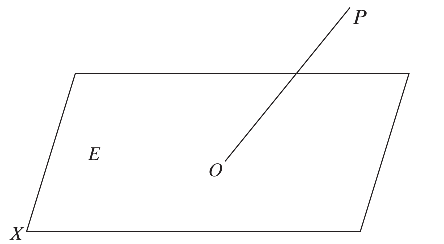
7| mgvšÍivj Zj (Parllel planes) t `BwU mgZj hw` ci¯úi †Q` bv K‡i A_vr hw` Zv‡`i †Kv‡bv mvaviY †iLv bv _v‡K Z‡e H ZjØq‡K mgvšÍivj Zj ejv nq|

8| mgZ‡ji mgvšÍivj †iLv t GKwU mij‡iLv I GKwU mgZj‡K Awbw`©ófv‡e ewaZ Ki‡jI hw` Zviv ci¯úi †Q` bv K‡i, Z‡e H mij‡iLv‡K D³ Z‡ji mgvšÍivj †iLv ejv nq|
`óe¨ t mvaviY wÎgvwÎK e¯'i Qwe wØgvwÎK KvMR ev †ev‡W A¼b wKQUv RwUj| ZvB †kwYK‡¶ cvV`vbKv‡j c‡Z¨KwU msÁvi e¨vL¨vi m‡½ Zvi GKwU wPÎ A¼b K‡i †`wL‡q w`‡j welqwU wk¶v_x‡`i c‡¶ †evSv I g‡b ivLv mnRZi n‡e|

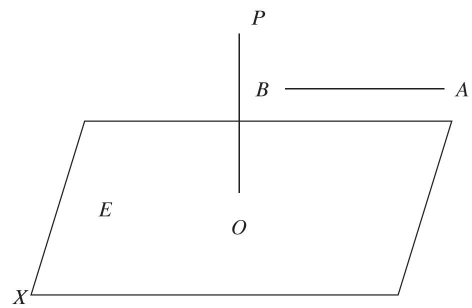


9| Z‡ji j¤^ †iLv (Normal or perpendicular to a plane) t †Kv‡bv mij‡iLv GKwU mgZ‡ji Dci¯' †Kv‡bv we›` †_‡K H mgZ‡ji Dci Aw¼Z †Kv‡bv we›` †_‡K H mgZ‡ji Dci Aw¼Z †h †Kv‡bv †iLvi Dci j¤^ n‡j, D³ mij‡iLv‡K H mgZ‡ji Dci j¤^ ejv nq|

10| wZhK (Oblique) †iLv t †Kvb mij‡iLv GKwU mgZ‡ji mv‡_ mgvšivj ev j¤ bv n‡j, H mij‡iLv‡K mgZ‡ji wZhK †iLv ejv nq|



11| Dj¤ (Vertical) †iLv ev Zj t w¯i Ae¯vq Sjš Ij‡bi mZvi m‡½ mgvšivj †K‡bv †iLv ev Zj‡K Lvov ev Dj¤ Zj e‡j|

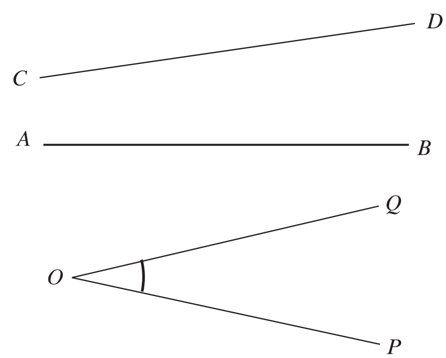


12| AbfwZK (Horizontal) Zj I †iLv t †K‡bv mgZj GKwU Lvov mij‡iLvi mv‡_ j¤ n‡j, Zv‡K kqvb ev AbfwgK Zj ejv nq| Avevi †K‡bv AbfwgK Z‡j Aew¯Z †h †Kvb mij‡iLv‡K AbfwgK mij‡iLv ejv nq|

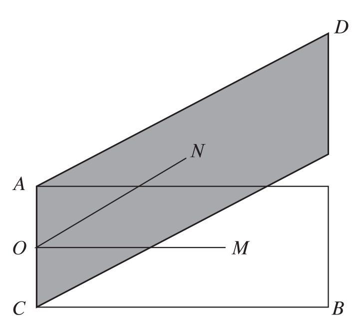
13| mgZj I ˆbKZjxq PZfR t †K‡bv PZf‡Ri evû¸‡jv mK‡j GB Z‡j Aew¯Z n‡j, Zv‡K mgZj PZfR ejv nq| Avevi †K‡bv PZf‡Ri evû¸‡jv mK‡j GKB Z‡j Aew¯Z bv n‡j, H PZfR‡K ˆbKZjxq PZfR ejv nq| ˆbKZjxq PZf‡Ri `BwU mwbwnZ evû GKZ‡j Ges Aci `BwU Ab¨ Z‡j Aew¯Z| mZivs †K‡bv ˆbKZjxq PZf‡Ri wecixZ evûØq ˆbKZjxq|

14| ˆbKZjxq †iLvi AšMZ †KvY t `BwU ˆKZjxq †iLvi AšMZ †KvY Zv‡`i †h‡Kv‡bv GKwU I Zvi Dci¯ †h‡Kv‡bv we›` †_‡K Aw¼Z AciwUi mgvšivj †iLvi AšMZ †Kv‡Yi mgvb| Avevi `BwU ˆbKZjxq †iLvi c‡Z¨‡Ki mgvšivj `BwU †iLv †K‡bv we›`‡Z A¼b Ki‡j H we›`‡Z Drcb †Kv‡Yi cwigvY I ˆbKZjxq †iLvØ‡qi AšMZ †Kv‡Yi mgvb|

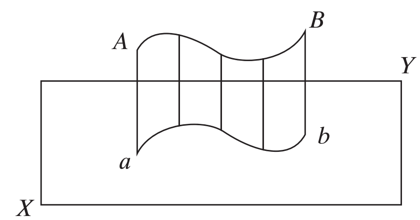
g‡b Kwi, AB I CD `BwU ˆbKZjxq †iLv| †h †Kv‡bv O we›`‡Z AB I CD Gi mgvšivj h_vµ‡g OP Ges OQ †iLvØq A¼b Ki‡j $\angle POQ$ B AB I CD Gi AšMZ †KvY wb‡`k Ki‡e|

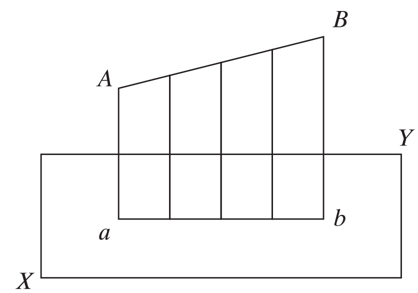


15| wØZj †KvY (Dihedral angle) t `BwU mgZj mij‡iLvq †Q` Ki‡j Zv‡`i †Q` †iLv¯ †h †Kv‡bv we›` †_‡K H mgZjØ‡qi c‡Z¨‡Ki Dci H †Q` †iLvi mv‡_ j¤ Gi/c GKwU K‡i †iLv A¼b Ki‡j Drcb †KvYB H mgZjØ‡qi AšMZ wØZj †KvY|



AB I CD mgZjØq AC †iLvq ci¯úi †Q` K‡i‡Q| AC †iLv¯ O we›`‡Z AB mgZ‡j OM Ges CD mgZ‡j ON Gi/c `BwU mij‡iLv A¼b Kiv n‡jv †hb Zviv DfqB AC Gi m‡½ O we›`‡Z j¤ nq| Zvn‡j ∠MON B AB I CD mgZjØ‡qi AšMZ wØZj †KvY mwPZ K‡i| `BwU ci¯ú‡”Q`x mgZ‡ji AšMZ wØZj †Kv‡Yi cwigvY GK mg‡KvY n‡j, H mgZjØq ci¯úi j¤|





16| Awf‡¶c t †Kv‡bv we›` †_‡K GKwU wbw`ó mij‡Lvi Dci ev †Kv‡bv mgZ‡ji Dci Aw¼Z j¤‡iLvi cv`we›`‡K H †iLv ev mgZ‡ji Dci D³ we›`i cvZb ev Awf‡¶c (Projection) ejv nq| †Kv‡bv mij‡iLv ev eµ‡iLvi mKj we›` †_‡K †Kv‡bv wbw`ó mgZ‡ji Dci Aw¼Z j¤¸‡jvi cv`we›`mg‡ni †mU‡K H mgZ‡ji Dci D³ mij‡iLv ev eµ‡iLvi Awf‡¶c ejv nq| GB Awf‡¶c‡K j¤ Awf‡¶cI (Orthogonal Projection) ejv nq|

wP‡Î XY mgZ‡ji Dci GKwU eµ‡iLv I GKwU mij‡iLvi Awf‡¶c †`Lv‡bv n‡q‡Q|

13.3 `BwU mij‡iLvi g‡a¨ m¤úK

(K) `BwU mij‡iLv GKZjxq n‡Z cv‡i, †m‡¶‡Î Zviv Aek¨B mgvšivj n‡e ev †Kv‡bv GK we›`‡Z ci¯úi †Q` Ki‡e|

(L) `BwU mij‡iLv ˆbKZjxh n‡Z cv‡i, †m‡¶‡Î Zviv mgvšivj I n‡e bv wKsev †Kv‡bv we›`‡Z †Q`I Ki‡e bv|

13.4 ¯Ztwm×

(K) †Kv‡bv mgZ‡ji Dci¯ `BwU we›`i ms‡hvRK mij‡iLv‡K Awbw`ófv‡e ewaZ Ki‡j I Zv m¤úYfv‡e H mgZ‡j Aew¯Z _vK‡e| mZivs GKwU mij‡iLv I GKwU mgZ‡ji g‡a¨ `BwU mvaviY we›` _vK‡j, H mij‡iLv eivei Zv‡`i g‡a¨ AmsL¨ mvaviY we›` _vK‡e|

(L) `BwU wbw`ó we›` ev GKwU mij‡iLvi ga¨ w`‡q AmsL¨ mgZj A¼b Kiv hvq|

13.5 mij‡iLv I mgZ‡ji g‡a¨ m¤úK

(K) GKwU mij‡iLv GKwU mgZ‡ji m‡½ mgvšivj n‡j Zv‡`i g‡a¨ †Kv‡bv mvaviY we›` _vK‡e bv|

(L) GKwU mij‡iLv †Kv‡bv mgZj‡K †Q` Ki‡j Zv‡`i g‡a¨ gvÎ GKwU mvaviY we›` _vK‡e|

(M) hw` †Kv‡bv mij‡iLv I mgZ‡ji `BwU mvaviY we›` _v‡K, Zvn‡j m¤úY mij‡iLwU H mgZ‡j Aew¯Z n‡e|

13.6 `BwU mgZ‡ji g‡a¨ m¤úK

(K) `BwU mgZj ci¯úi mgvšivj n‡j Zv‡`i g‡a¨ †Kv‡bv mvaviY we›` _vK‡e bv|

(L) `BwU mgZj ci¯úi‡Q`x n‡j Zviv GKwU mij‡iLvq †Q` Ki‡e Ges Zv‡`i AmsL¨ mvaviY we›` _vK‡e|

13.7 Nbe¯

 Avgiv Rwb, GKLvbv eB ev GKLvbv BU ev GKwU ev· ev GKwU †MvjvKvi ej meB Nbe¯ Ges Zviv c‡Z¨‡KB wKQ cwigvb ¯vb (Space) `Lj K‡i _v‡K| Avevi GKLÛ cv_i ev KvV, B‡Ui GKwU LÛ, Kqjvi UKiv, G‡Uj gwUi ïKbv LÛ BZ¨w` I Nbe¯i D`vniY| Z‡e G¸‡jv welg Nbe¯|

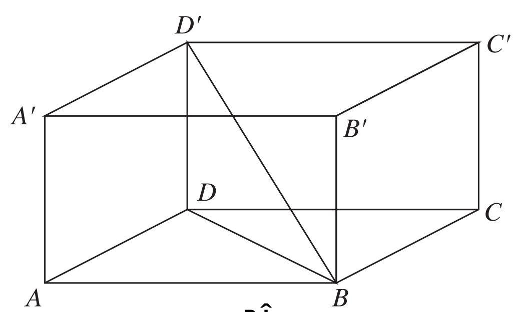
mgZj A_ev eµZj Øviv †ewóZ k‡b¨i wKQUv ¯vb `Lj K‡i _v‡K Gic e¯‡K Nbe¯ (Solid) ejv nq| mgZj¯ †Kv‡bv ¯vb‡K †eób Ki‡Z n‡j †hgb,Kgc‡¶ wZbwU mij ‡iLv `iKvi †Zgwb RvMwZK †Kv‡bv ¯vb‡K †eób Ki‡Z n‡j AšZ PviwU mgZj `iKvi| GB Zj¸‡jv Nbe¯i Zj ev côZj (Surface) Ges G‡`i `wU mgZj †h ‡iLvq †Q` K‡i, Zv‡K H Nbe¯i avi (Edge) ejv nq|

GKwU ev‡·i ev GKLvbv B‡Ui QqwU côZj Av‡Q Ges eviwU avi Av‡Q| GKwU wµ‡KU ej gvÎ GKwU eµZj Øviv Ave×|

KvR: 1| ‡Zvgiv c‡Z¨‡K GKwU K‡i mlg Nbe¯ I welg Nbe¯i bvg wjL|
    2| ‡Zvgvi D‡jwLZ Nbe¯¸‡jvi K‡qKwU e¨envi wjL|

13.8 mlg Nbe¯i AvqZb I Z‡ji †¶Îdj

1| AvqwZK Nb ev AvqZvKvi Nbe¯ (Rectangular Parallelopiped)



wPÎ

wZb‡Rvov mgvšivj mgZj Øviv Ave× Nbe¯‡K mvgvšwiK Nbe¯ ejv nq| GB QqwU mgZ‡ji c‡Z¨KwU GKwU mvgvšwiK Ges wecixZ cô¸‡jv me‡Zvfv‡e mgvb| mvgvšwiK Nbe¯i wZbwU `‡j wef³ eviwU avi Av‡Q|

†h mvgvšwiK Nbe¯i côZj¸‡jv AvqZ‡¶Î, Zv‡K AvqZvKvi Nbe¯ ejv nq| †h AvqZvKvi Nbe¯i côZj¸‡jv eM‡¶Î, Zv‡K NbK (Cube) ejv nq| Dc‡iv³ wP‡Î AvqZvKvi Nbe¯i Ges Nb‡Ki cô¸‡jv ABCD, A′B′C′D′, BCC′B′, ADD′A′, ABB′A′, DCC′D′ Ges avi¸‡jv AB, A′B′, CD, C′D′, BC, B′C′, AD, A′D′, AA′, BB′, CC′, DD′ Ges GKwU KY BD′.

g‡b Kwi, AvqZvKvi Nbe¯i ^`N¨, c¯ I D”PZv h_vµ‡g AB = *a* GKK AD = *b* GKK Ges AA′ = *c* GKK|

(K) AvqZvKvi Nbe¯i mgMZ‡ji †¶Îdj (Area of the whole surface)
= QqwU c‡ôi ‡¶Îd‡ji mgwó

= 2(ABCD Z‡ji †¶Îdj + ABB′A′ Z‡ji †¶Îdj + ADD′A′ Z‡ji †¶Îdj)
= $2(ab + ac + bc)$ eMGKK
= $2(ab + bc + ca)$ eMGKK

(L) AvqZb (Volume) = AB × AD × AA′ NbGKK = abc NbGKK
(M) KY $BD' = \sqrt{BD^2 + DD'^2} = \sqrt{AB^2 + AD^2 + DD'^2} = \sqrt{a^2 + b^2 + c^2}$ GKK

2| Nb‡Ki ‡¶‡Î, a = b = c. AZGe
(K) mgMZ‡ji ‡¶Îdj = $2(a^2+a^2+a^2) = 6a^2$ eMGKK
(L) AvqZb = a. a. a = a^3 NbGKK
(M) KY= $\sqrt{a^2 + a^2 + a^2} = \sqrt{3}a$ GKK|

D`vniY 1| GKwU AvqZvKvi Nbe¯‘i ˆ`N©¨, cÖ¯’ I D”PZvi AbycvZ 4: 3: 2 Ges Zvi mgMZ‡ji †¶Îdj 468 eMwgUvi n‡j, Zvi KY I AvqZb wbY©q Ki|

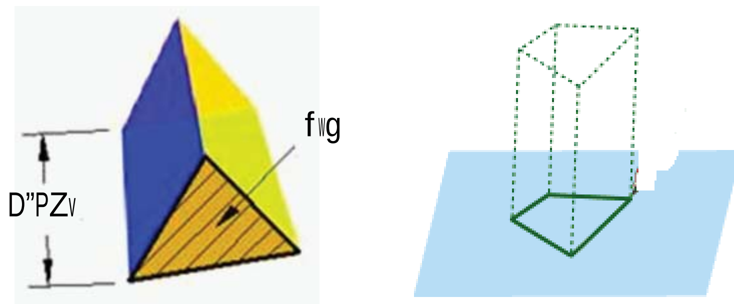
mgvavb : g‡b Kwi, ˆ`N©¨, cÖ¯’ I D”PZv h_vµ‡g $4x, 3x, 2x$ wgUvi|
Zvn‡j, $2(4x.3x + 3x.2x + 2x.4x) = 468$
ev, $52x^2 = 468$ ev, $x^2 = 9 \therefore x = 3$
$\therefore$ Nbe¯‘i ˆ`N©¨ 12 wg., cÖ¯’ 9 wg. Ges D”PZv 6 wg.
Bnvi K‡Yi ˆ`N©¨ = $\sqrt{12^2 + 9^2 + 6^2} = \sqrt{144 + 81 + 36} = \sqrt{261}$ wgUvi =16.16 wgUvi (cÖvq)
Ges AvqZb = 12×9×6 = 648 NbwgUvi|

KvR : 1| wcR‡ev‡Wi GKwU †QvU ev· (KvUb A_ev Jl‡ai †evZ‡ji c¨v‡KU) Gi ˆ`N©¨, cÖ¯’, I D”PZv †g‡c Zvi AvqZb, QqwU Z‡ji †¶Îdj I K‡Yi ˆ`N©¨ wbY©q Ki|

3| wcRg (Prism)

‡h Nbe¯‘i `B cÖvšÍ memg I mgvšÍivj eûfR Øviv Avey× Ges Ab¨vb¨ Zj¸‡jv mvgvšÍwiK Zv‡K wcRg e‡j| wcR‡gi `B cÖvš‡K Bnvi f‚wg Ges Ab¨vb¨ Zj¸‡jv‡K cvk¦Zj e‡j| me¸‡jv cvk¦Zj AvqZvKvi n‡j wcRgwU‡K Lvov wcRg Ges Ab¨‡¶‡Î wcRgwU‡K ZxhK wcRg ejv nq| ev¯‘e †¶‡Î Lvov wcRgB AwaK e¨eüZ nq| f‚wg Z‡ji bv‡gi Dci wbfi K‡i wcR‡gi bvgKiY Kiv nq| †hgb, wÎfRvKvi wcRg, PZzfRvKvi wcRg, cÂfRvKvi wcRg BZ¨vw`|

f‚wg mg eûfR n‡j wcRg‡K mg wcRg (Regular prism) e‡j| f‚wg mg bv n‡j Bnv‡K welg wcRg (Irregular prism) ejv nq| msÁvbmv‡i AvqZvKvi Nbe¯‘ I NbK Dfq‡KB wcRg ejv nq| Kv‡Pi ˆZwi Lvov wÎfRvKvi wcRg Av‡jvKiwk¥i we”Qi‡Yi Rb¨ e¨eüZ nq|



`B ai‡bi wcRg

K) wcR‡gi mgMZ‡ji †¶Îdj
= 2 (fv‡gi †¶Îdj) + cvkZj¸‡jvi †¶Îdj
= 2 (fv‡gi †¶Îdj) + fv‡gi cwimxgv × D”PZv
L) AvqZb = fv‡gi †¶Îdj × D”PZv

D`vniY 2| GKwU wÎfRvKvi wcR‡gi fv‡gi evû¸‡jvi ˆ`N©¨ h_vµ‡g 3, 4 I 5 †m. wg. Ges D”PZv 8 †m. wg.| Bnvi mgMZ‡ji †¶Îdj I AvqZb wbYq Ki|

mgvavb : wcR‡gi fv‡gi evû¸‡jvi ˆ`N©¨ h_vµ‡g 3, 4 I 5 †m. wg.|

‡h‡nZz $3^2 + 4^2 = 5^2$, Bnvi fvg GKwU mg‡KvYx wÎfR hvi †¶Îdj $= \frac{1}{2} \times 4 \times 3 = 6$ eM †m. wg.

∴ wcRgwUi mgMZ‡ji †¶Îdj $= 2 \times 6 + \frac{1}{2}(3 + 4 + 5) \times 8 = 12 + 48 = 60$ eM †m. wg.

Ges Bnvi AvqZb = $6 \times 8 = 48$ Nb †m. wg.

AZGe wcRgwUi mgMZ‡ji †¶Îdj 60 eM †m. wg. Ges AvqZb 48 Nb †m. wg.|

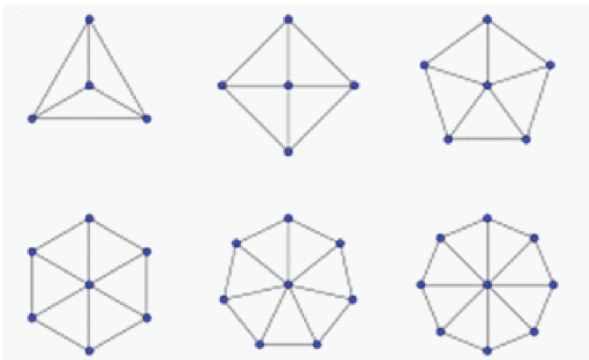
4. wcivwgW (Pyramid)

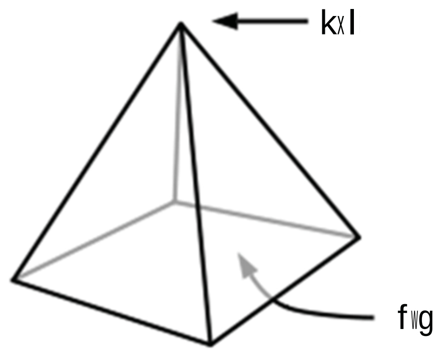
eûf‡Ri Dci Aew¯’Z †h Nbe¯‘i GKwU kxl©we›`y _v‡K Ges hvi cvkZj¸‡jvi c‡Z¨KwU wÎfRvKvi Zv‡K wcivwgW e‡j|

wcivwg‡Wi fvg †h‡Kv‡bv AvKv‡ii eûfR Ges Zvi cvkZj¸‡jvI †h‡Kv‡bv ai‡bi wÎfR n‡Z cv‡i| Z‡e fvg mlg eûfR Ges cvkZj¸‡jv memg wÎfR n‡j Zv‡K mlg wcivwgW ejv nq| mlg wcivwgW¸‡jv LeB `wóbv›`b| kxl©we›`y I fv‡gi †h‡Kv‡bv †KŠwYK we›`yi ms‡hvRK †iLv‡K wcivwg‡Wi avi e‡j| kxl n‡Z fv‡gi Dci AswKZ j¤^`N©¨‡K wcivwg‡Wi D”PZv ejv nq|

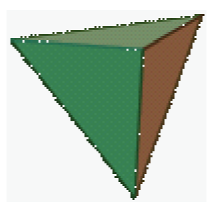
Z‡e Avgiv wcivwgW ej‡Z mPivPi eMvKvi fv‡gi Dci Aew¯’Z PviwU memg wÎfR Øviv †ewóZ Nbe¯‘‡KB ewS| GB ai‡bi wcivwg‡Wi eûj e¨envi Av‡Q|

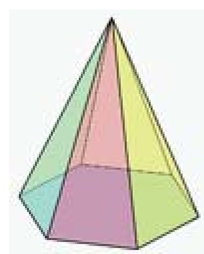
PviwU mgevû wÎfR Øviv †ewóZ Nbe¯‘K mlg PZ¯jK (Regular tetrahedron) e‡j hv GKwU wcivwgW| GB wcivwg‡Wi 3 + 3 = 6 wU avi I 4 wU ˆKŠwYK we›`y Av‡Q| Bnvi kxl n‡Z fwgi Dci Aw¼Z j¤ fwgi fi‡K›`ª cwZZ nq|





wewfbœ ai‡bi wcivwg‡Wi fwgi bKkv





wcivwgW

K) wcivwg‡Wi mgMZ‡ji †¶Îdj

= fwgi †¶Îdj + cvkZj¸‡jvi †¶Îdj

wKš‘ cvkZj¸‡jv memg wÎfR n‡j,

wcivwg‡Wi mgMZ‡ji †¶Îdj = fwgi †¶Îdj + $\frac{1}{2}$ (fwgi cwiwa $\times$ †nj‡bi D”PZv)

wcivwg‡Wi D”PZv h, fwg†¶‡Îi Aš—e„‡Ëi e¨vmva© r Ges †nj‡bi D”PZv l n‡j, $l=\sqrt{h^2+r^2}$

L) AvqZb = $\frac{1}{3}\times$ fwgi †¶Îdj $\times$ D”PZv

D`vniY 3| 10 †m. wg. evûwewkó eMvKvi fwgi Dci Aew¯’Z GKwU wcivwg‡Wi D”PZv 12 †m. wg.| Bnvi mgMZ‡ji †¶Îdj I AvqZb wbY©q Ki|

mgvavb: wcivwg‡Wi fwgi †K›`ª we›`y n‡Z †h‡Kv‡bv evûi j¤ `iZ $r=\frac{10}{2}$ †m. wg. = 5 †m. wg. , wcivwg‡Wi D"PZv 12 †m. wg.| AZGe

Bnvi †h‡Kv‡bv cvkZ‡ji †nj‡bi D"PZv $=\sqrt{h^2+r^2}=\sqrt{12^2+5^2}=\sqrt{144+25}=\sqrt{169}=13$ †m. wg.

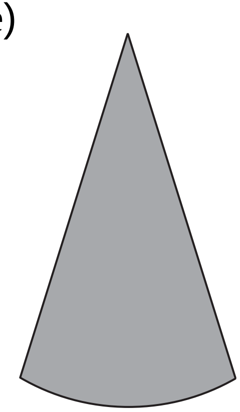
wcivwg‡Wi mgMZ‡ji †¶Îdj $=10\times 10+\frac{1}{2}(4\times 10)\times 13=100+260=360$ eM †m. wg.

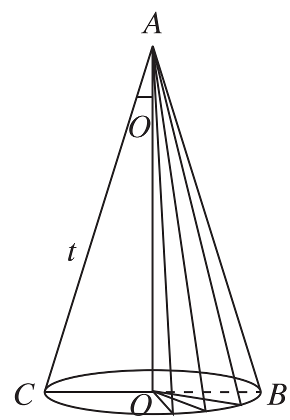
Ges Bnvi AvqZb $=\frac{1}{3}\times(10\times 10)\times 12=10\times 10\times 4=400$ Nb †m. wg.

AZGe wcivwgWwUi mgMZ‡ji †¶Îdj 360 eM †m. wg. Ges AvqZb 400 Nb †m. wg.|

KvR : 1| ‡Zvgvi c‡Z¨‡K GKwU K‡i mlg I GKwU K‡i welg (K) wcRg I (L) wcivwgW AvK|
2| †h‡¶‡Î m¤¢e, †Zvgvi Aw¼Z Nbe¯'wUi mgMZ‡ji †¶Îdj Ges AvqZb wbYq Ki|

4| mgeËfwgK †KvYK (Right circular cone)

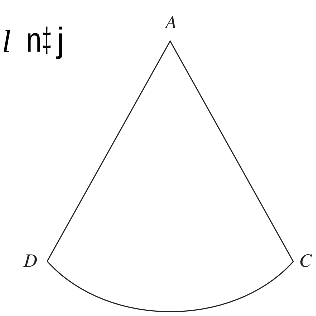




‡Kv‡bv mg‡KvYx wÎf‡Ri mg‡KvY msjMœ GKwU evû‡K A¶ (axis) a‡i Zvi PZw`©‡K wÎfRwU‡K GKevi Nwi‡q Avb‡j †h Nbe¯' Drcb nq, Zv‡K mgeËfwgK †KvYK ejv nq|

wP‡Î, OAC mg‡KvYx wÎfR‡K OA Gi PZw`©‡K †Nviv‡bvi d‡j ABC mgeËfwgK †KvYK Drcb n‡q‡Q|G‡¶‡Î wÎf‡Ri kx©l†KvY θ n‡j, θ †K †KvY‡Ki Aakx©l†KvY (Semi- vertical angle) ejv nq|

†KvY‡Ki D"PZv $OA=h,$ fwgi e¨vmva $OC=r$ Ges †nj‡bi D"PZv $AC=l$ n‡j



(K) eµZ‡ji †¶Îdj $=\frac{1}{2}\times$ fwgi cwiwa $\times$ †nj‡bi D"PZv

$=\frac{1}{2}\times 2\pi r\times l=\pi rl$ eMGKK

(L) mgMZ‡ji †¶Îdj = eµZ‡ji †¶Îdj + fwgZ‡ji †¶Îdj

$=\pi rl+\pi r^2=\pi r(r+l)$ eMGKK

(M) AvqZb = $\frac{1}{3}\times$ fwgi †¶Îdj $\times$ D”PZv = $\frac{1}{3}\pi r^2 h$ NbGKK|

[AvqZ‡bi GB mÎwU wbYq c×wZ D”PZi †kwY‡Z wkLv‡bv n‡e|]

D`vniY 4| GKwU mgeË fwgK †KvY‡Ki D”PZv 12 †m. wg. Ges fwgi e¨vm 10 †m. wg. n‡j Zvi ‡njv‡bv D”PZv, eµZ‡ji †¶Îdj I mgMZ‡ji †¶Îdj Ges AvqZb wbYq Ki|

mgvavb : fwgi e¨vmva $r=\frac{10}{2}$ †m. wg.= 5 †m. wg.

†njv‡bv D”PZv $l=\sqrt{h^2+r^2}=\sqrt{12^2+5^2}=13$ †m. wg.

eµZ‡ji †¶Îdj = $\pi rl=\pi\times 5\times 13=204.2035$ e. †m. wg.

mgMZ‡ji †¶Îdj = $\pi r(l+r)=\pi\times 5(13+5)=282.7433$ e. †m. wg.

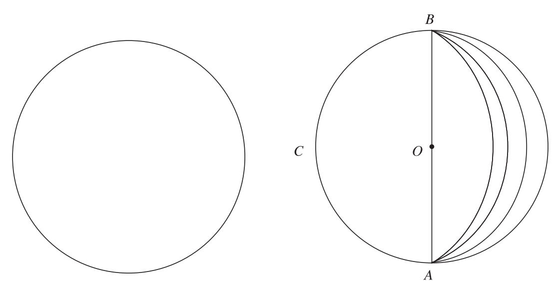
AvqZb = $\frac{1}{3}\pi r^2 h=\frac{1}{3}\times\pi\times 5^2\times 12=314.1593$ N. †m. wg.|

KvR: Rb¨`‡b ev Ab¨vb¨ Avb›` Drm‡e e¨enZ †KvYK AvKwZi GKwU K¨vc msMn K‡i Zvi eµZ‡ji †¶Îdj I AvqZb wbYq Ki|

5| †MvjK (Sphere)

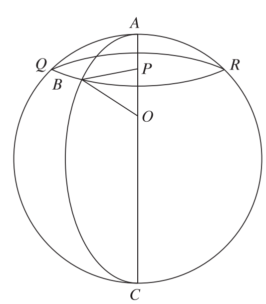
†Kv‡bv AaeË †¶‡Îi e¨vm‡K A¶ a‡i H e¨v‡mi PZw`‡K AaeË †¶Î‡K GKevi Nwi‡q Avb‡j †h Nbe¯‘ Drcb nq Zv‡K †MvjK e‡j| AaeËwUi †K›`ªB †Mvj‡Ki †K›`ª| GB NY‡bi d‡j AaeË †h Zj Drcb K‡i ZvB nj †Mvj‡Ki Zj| †Mvj‡Ki †K›`ª ej‡Z gj e‡Ëi †K›`ª‡KB eSvq|

$CQAR$ †Mvj‡Ki †K›`ª $O$, e¨vmva $OA=OB=OC$ Ges †K›`ª †_‡K h `i‡Z $P$ we›`i ga¨ w`‡q $OA$ †iLvi mv‡_ j¤^ nq Gi¦c GKwU mgZj †MvjKwU‡K †Q` K‡i QBR eËwU Drcb K‡i‡Q| GB e‡Ëi †K›`ª P Ges e¨vmva PB| Zvn‡j PB Ges OP ci¯úi mgvb|



$\therefore OB^2=OP^2+PB^2$

$\therefore PB^2=OB^2-OP^2=r^2-h^2$



†Mvj‡Ki e¨vmva r n‡j,

(K) †Mvj‡Ki côZ‡ji †¶Îdj = $4\pi r^2$ eMGKK|

(L) AvqZb = $\frac{4}{3}\pi r^3$ NbGKK|

(M) h D”PZvq Zj‡PQ‡` Drcb e‡Ëi e¨vmva = $\sqrt{r^2-h^2}$ GKK|

KvR: GKwU †Ljbv ej ev dUej wb‡q Zvi e¨vmva wbYq Ki| AZtci Gi AvqZbI ‡ei Ki|

D`vniY 5| 4 †m. wg. e¨v‡mi GKwU †jŠn †MvjK‡K wcwU‡q $\frac{3}{2}$ †m. wg. ci“ GKwU eËvKvi †jŠncvZ c¯Z Kiv nj| H cv‡Zi e¨vmva KZ?

mgvavb : †jŠn †Mvj‡Ki e¨vmva = $\frac{4}{2} = 2$ †m. wg.| $\therefore$ Zvi AvqZb = $\frac{4}{3}\pi . 2^3 = \frac{32}{3}\pi$ Nb †m. wg.

g‡b Kwi, cv‡Zi e¨vmva = r †m. wg.| cvZwU $\frac{2}{3}$ †m. wg. ci“|

$\therefore$ cv‡Zi AvqZb = $\pi r^2 \times \frac{2}{3}$ N. †m. wg. = $\frac{2}{3}\pi r^2$ N. †m. wg.|

kZvbmv‡i, $\frac{2}{3}\pi r^2 = \frac{32}{3}\pi$ ev, $r^2 = 16$ ev, $r = 4$

$\therefore$ cv‡Zi e¨vmva = 4 †m. wg.

D`vniY 6| mgvb D”PZv wewkó GKwU mgeËfwgK †KvYK, GKwU Aa †MvjK I GKwU wmwjÛvi mgvb mgvb fwgi Dci Aew¯Z| †`LvI †h, Zv‡`i AvqZ‡bi AbcvZ $1 : 2 : 3$

mgvavb : g‡b Kwi, mvaviY D”PZv I fwgi e¨vmva h_vµ‡g h Ges r GKK| †h‡nZ Aa †Mvj‡Ki D”PZv I e¨vmva mgvb| $\therefore h = r$

Zvn‡j †KvY‡Ki AvqZb = $\frac{1}{3}\pi r^2 h = \frac{1}{3}\pi r^3$ NbGKK

Aa †Mvj‡Ki AvqZb = $\frac{1}{2}\left(\frac{4}{3}\pi r^3\right) = \frac{2}{3}\pi r^3$ NbGKK Ges wmwjÛv‡ii AvqZb = $\pi r^2 h = \pi r^3$

$\therefore$ wb‡Yq AbcvZ $= \frac{1}{3}\pi r^3 : \frac{2}{3}\pi r^3 : \pi r^3 = \frac{1}{3} : \frac{2}{3} : 1 = 1 : 2 : 3$

D`vniY 7| GKwU AvqZvKvi †jŠn dj‡Ki ˆ`N¨, c¯ I D”PZv h_vµ‡g 10, 8 I $5\frac{1}{2}$ †m. wg.| GB djKwU‡K Mwj‡q $\frac{1}{2}$ †m. wg. e¨vmva wewkó KZ¸‡jv †MvjvKvi ¸wj c¯Z Kiv hv‡e?

mgvavb : †jŠn dj‡Ki AvqZb = $10 \times 8 \times 5\frac{1}{2}$ N. †m. wg. = 440 N. †m. wg.

g‡b Kwi, ¸wji msL¨v = n

$\therefore$ n msL¨K ¸wji AvqZb = $n \times \frac{4}{3}\pi \left(\frac{1}{2}\right)^3 = \frac{n\pi}{6}$ N. †m. wg.

ckvbmv‡i, $\frac{n\pi}{6} = 440$ $\therefore$ $n = \frac{440 \times 6}{\pi} = 840{\cdot}3$

$\therefore$ wb‡Yq ¸wji msL¨v 840 wU|

D`vniY 8| GKwU mgeË fwgK †KvY‡Ki AvqZb V, eµZ‡ji †¶Îdj S, fwgi e¨vmva r, D”PZv h Ges Aa kxl‡KvY $\alpha$ n‡j †`LvI †h,

$$(i)\ S=\frac{\pi h^2\tan\alpha}{\cos\alpha}=\frac{\pi r^2}{\sin\alpha}$$ eMGKK

$$(ii)\ V=\frac{1}{3}\pi h^3\tan^2\alpha=\frac{\pi r^3}{3\tan\alpha}$$ NbGKK

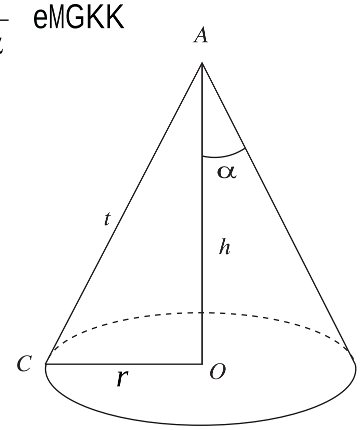
mgvavb : cv‡ki wP‡Î, †KvY‡Ki D”PZv $OA=h$, †nj‡bi D”PZv $AC=l$, fwgi e¨vmva $OC=r$ Ges Aa kxl‡KvY $\angle OAC=\alpha$.

†nj‡bi D”PZv $l=\sqrt{h^2+r^2}$.

wPÎ n‡Z †`Lv hvq †h, $\tan\alpha=\frac{r}{h}$ $\quad\therefore\ r=h\tan\alpha$ ev, $h=\frac{r}{\tan\alpha}=r\cot\alpha$

GLb $(i)\ S=\pi rl=\pi r\sqrt{h^2+h^2\tan^2\alpha}=\pi rh\sqrt{1+\tan^2\alpha}=\pi rh\sqrt{\sec^2\alpha}$

$$=\pi rh\sec\alpha=\frac{\pi r}{\cos\alpha}.r\cot\alpha=\frac{\pi r^2}{\cos\alpha}.\frac{\cos\alpha}{\sin\alpha}=\frac{\pi r^2}{\sin\alpha}$$ eMGKK



$$(ii)\ V=\frac{1}{3}\pi r^2h=\frac{1}{3}\pi(h\tan\alpha)^2h=\frac{1}{3}\pi h^3\tan^2\alpha$$

$$=\frac{1}{3}\pi\left(\frac{r}{\tan\alpha}\right)^3\tan^2\alpha=\frac{\pi r^3}{3\tan\alpha}$$ NbGKK|

5| †hŠwMK Nbe¯ (Compound solid)

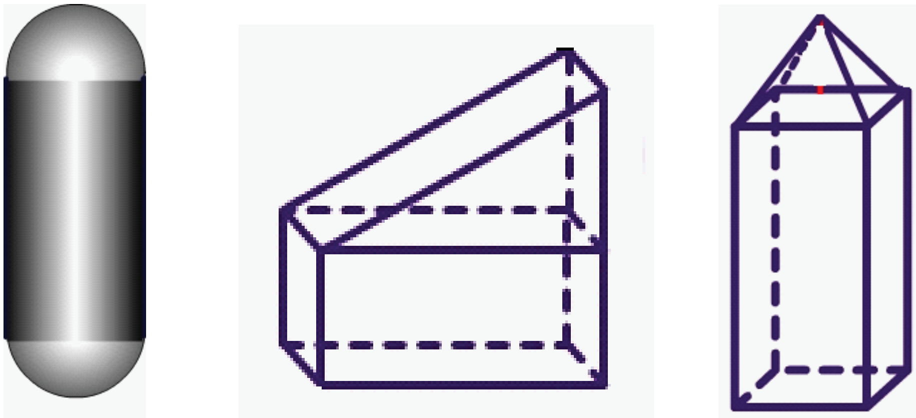
`BwU Nbe¯i mgš‡q MwVZ Nbe¯‡K †hŠwMK Nbe¯ e‡j|
†hŠwMK Nbe¯i K‡qKwU D`vniY t

(1) GKwU AvqZvKvi Nbe¯i Dc‡ii Zj hw` GKwU Lvov wcR‡gi †Kvb I GKwU Z‡ji mgvb nq Z‡e Nbe¯i Dci wgwj‡q wcRwU emv‡j GKwU †hŠwMK Nbe¯ nq|

(2) GKwU wcR‡gi fwg I GKwU PZ¯‹j‡Ki fwg memg n‡j Ges PZ¯‹jKwU‡K wcR‡gi Dci emv‡j GKwU †hŠwMK Nbe¯ nq|

(3) GKwU †Mvj‡Ki e¨vmva I GKwU mgeË fwgK †KvY‡Ki fwgi e¨vmva mgvb n‡j Ges †KvYKwU‡K †Mvj‡Ki Dci emv‡j GKwU bZb Nbe¯ m„wó nq|

(4) `BwU Aa‡MvjK I GKwU mgeË fwgK wmwjÛv‡ii mgš‡q MwVZ †hŠwMK Nbe¯‡K K¨vcmyj ejv †h‡Z cv‡i|



wewfb AvKv‡ii †hŠwMK Nbe¯

Gfv‡e `B ev `B‡qi AwaK Nbe¯i mgš‡q wewfb cKv‡ii †hŠwMK Nbe¯ ˆZwi Kiv hvq| A‡bK `wób›`b ¯vcbvI †hŠwMK Nbe¯| e¨vqvg Kivi A‡bK DcKiYI GKwaK Nbe¯i mgš‡q ‰Zwi Kiv nq|

KvR: †Zvgiv c‡Z¨‡K GKwU K‡i †hŠwMK Nbe¯ A¼b Ki I Bnvi eYbv `vI| m¤e n‡j Bnvi Zjmg‡ni †¶Îdj I AvqZb wbY‡qi mÎ †jL|

D`vniY 9| GKwU K¨vcm‡ji ‰`N¨ 15 †m. wg.| Bnvi wmwjÛvi AvKwZi As‡ki e¨vmva 3 †m. wg. n‡j, mgMZ‡ji †¶Îdj I AvqZb wbYq Ki|

mgvavb: K¨vcm‡ji m¤úY ‰`N¨ 15 †m. wg.| ‡h‡nZ K¨vcm‡ji `B cvš Aa‡MvjKvKwZi, ‡m‡nZ Bnvi wmwjÛvi AvKwZi As‡ki ‰`N¨ $l = 15 - (3 + 3) = 9$ †m. wg.|

mZivs K¨vcm‡ji mgMZ‡ji †¶Îdj

= `B cv‡ši Aa‡MvjvKwZ As‡ki côZ‡ji †¶Îdj + wmwjÛvi AvKwZi As‡ki côZ‡ji †¶Îdj

$= 2\times\frac{1}{2}\times 4\pi r^2 + 2\pi rl = 4\pi(3)^2 + 2\pi\times 3\times 9$ [$\because$ $r = 3$ †m. wg.]

$= 90\pi = 282.74$ eM †m. wg.

Ges K¨vcmjwUi AvqZb = $2\times\frac{1}{2}\times\frac{4}{3}\pi r^3 + \pi r^2 l = \frac{4}{3}\pi(3)^3 + \pi(3)^2\times 9 = 117\pi = 367.57$ Nb †m. wg.|

Abkxjbx 13

1| GKwU AvqZvKvi Nbe¯i ˆ`N¨ 8 †m.wg., c¯ 4 †m.wg Ges D”PZv 3 †m.wg. n‡j Gi KY KZ?

K. $5\sqrt{2}$ †m.wg. L. 25 †m.wg

M. $25\sqrt{2}$ †m.wg N. 50 †m.wg

2| †Kv‡bv mg‡KvYx wÎf‡Ri AwZfR wfb Aci evûØ‡qi ˆ`N¨ 4 †m.wg. Ges 3 †m.wg| wÎfRwU‡K enËi evûi PZw`‡K †Nviv‡j-

i Drcb Nbe¯wU GKwU mgeË fwgK †KvYK n‡e

ii Nbe¯wU GKwU mgeË fwgK wmwjÛvi n‡e

iii Drcb Nbe¯wUi fwgi †¶Îdj n‡e 9π eM †m.wg.

TOTAL PAGES = 356