**NCTU\_Ragnarok codebook**

INDEX:

1. Default P1
2. Prime P1
3. Binary Search P1-P2
4. LIS P2
5. Computational Geometry P2-P3
6. Dijkstra P3
7. Bellmn\_ford P3
8. SPFA P3-P4
9. Disjoin Set P4
10. Segment set P4
11. String P4-P5
12. Number Theory P5-P6
13. BCC P6
14. SCC P6-P7
15. 2-STA P7-P9
16. Sort P9-P11
17. Big Number P11-P13

**Default**

#include<bits/stdc++.h>

#define FI freopen("in.txt", "r", stdin)

#define IOS ios\_base::sync\_with\_stdio(0);cin.tie(0)

#define pb push\_back

#define mp make\_pair

#define mt make\_tuple

#define ff first

#define ss second

#define EPS 1E-7

#define INF 0x7FFFFFFF

#define PI 3.1415926535897932384626433832795

using namespace std;

typedef long long LL;

typedef pair<int,int> PII;

typedef pair<LL,LL> PLL;

typedef vector<int> VII;

typedef vector<LL> VLL;

typedef vector<PII> VPII;

typedef vector<PLL> VPLL;

//int dx[]={1,0,-1,0};              int dy[]={0,1,0,-1}; //4 Direction

//int dx[]={1,1,0,-1,-1,-1,0,1};    int dy[]={0,1,1,1,0,-1,-1,-1};//8 direction

//int dx[]={2,1,-1,-2,-2,-1,1,2};   int dy[]={1,2,2,1,-1,-2,-2,-1};//Knight Direction

//int dx[]={2,1,-1,-2,-1,1};        int dy[]={0,1,1,0,-1,-1}; //Hexagonal Direction

**Prim**

const int MAXN=110;

bool vis[MAXN];

int lowc[MAXN];

int Prim(int cost[][MAXN],int n) //0-based

{

   int ans=0;

   memset(vis,0,sizeof(vis));

   vis[0]=false;

   for(int i=1;i<n;i++)lowc[i]=cost[0][i];

   for(int i=1; i<n;i++)

   {

       int minc=INT\_MAX;

       int p=-1;

       for(int j=0;j<n;j++)

       {

           if(!vis[j] && minc>lowc[j])

           {

               minc=lowc[j];

               p=j;

           }

       }

       if(minc==INT\_MAX) return -1; //failed

       ans+=minc;

       vis[p]=true;

       for(int j=0;j<n;j++)

           if(!vis[j] && lowc[j]>cost[p][j])

               lowc[j]=cost[p][j];

   }

   return ans;

}

**Binary Search**

int BinarySearch(int Data[],int N,int K){

   int L=0,H=N-1,M;

   while(L <= H){

       M=(L+H)>>1;

       if(K < Data[M])H=M-1;

       else if(Data[M] < K)L=M+1;

       else return M;

   }

   return -1;

}

**LIS**

int LIS(vector<int> &s)

{

   if(s.size()==0) return 0;

   vector<int> v;

   v.push\_back(s[0]);

   for(int i=1;i<s.size();++i)

   {

       int n=s[i];

       if(n>v.back()) v.push\_back(n); //strictly increasing >=(increasing)

       else \*lower\_bound(v.begin(),v.end(),n)=n;

   }

   return v.size();

}

**Computational Geometry**

const double eps=1E-7;

struct PT{

   double x,y;

   PT(){}

   PT(double x,double y):x(x),y(y){};

   PT operator+(const PT& p) const {

       return PT(x+p.x,y+p.y);

   }

   PT operator-(const PT& p) const {

       return PT(x-p.x,y-p.y);

   }

   PT operator\*(double c) const {

       return PT(c\*x,c\*y);

   }

   PT operator/(double c) const {

       return PT(c\*x,c\*y);

   }

   double operator\*(const PT& p) const {

       return x\*p.x+y\*p.y;

   }

   double operator^(const PT& p) const {

       return x\*p.y-y\*p.x;

   }

   double len2() const {

       return x\*x+y\*y;

   }

   double len() const {

       return sqrt(x\*x+y\*y);

   }

   PT Unit() const {

       return \*this / len();

   }

   PT Verticle() const {

       return PT(-y,x);

   }

   PT Rotate(double a) const {

       return PT(cos(a)\*x-sin(a)\*y,sin(a)\*x+cos(a)\*y);

   }

};

int orientation(const PT p1,const PT& p2,const PT& p3){

   double a = (p2-p1)^(p3-p1);

   if(-eps < a && a < eps)return 0;

   return a > 0 ? 1: -1;

}

bool opposite(const PT& p1,const PT& p2,const PT& p3){

   return (p2-p1)\*(p3-p1) < 0;

}

int segmentIntersection(const PT& p1,const PT& p2,const PT& p3,const PT& p4){

   int a123 = orientation(p1,p2,p3);

   int a124 = orientation(p1,p2,p4);

   int a341 = orientation(p3,p4,p1);

   int a342 = orientation(p3,p4,p2);

   if(a123 == 0 && a124 == 0)

       return opposite(p1,p3,p4)||opposite(p2,p3,p4)||opposite(p3,p1,p2)||opposite(p4,p1,p2);

   else if(a123\*a124 <= 0 && a341\*a342 <= 0)

       return true;

   return false;

}

**Dijkstra**

int\* Dijkstra(vector<VPII> E,int N,int S){

   bool \*visit=new bool[N+1];for(int i=1;i<=N;i++)visit[i]=false;

   int \*D=new int[N+1];for(int i=1;i<=N;i++)D[i]=INF;

   priority\_queue<PII,VPII,greater<PII>> P;

   P.push(MP(0,S));D[S]=0;

   while(!P.empty()){

       int weight=P.top().ff,now=P.top().ss;P.pop();

       if(visit[now])continue;

       visit[now]=true;

       for(auto i:E[now]){

           int potential=D[now]+i.ff;

           if(!visit[i.ss] && potential < D[i.ss]){

               P.push(MP(D[i.ss]=potential,i.ss));

           }

       }

   }

   return D;

}

**bellman\_ford**

struct Edge {

   int u,v,cost;

   Edge(int \_u=0,int \_v=0,int \_cost=0):u(\_u),v(\_v),cost(\_cost){}

};

vector<Edge> E;

bool bellman\_ford(int start,int n) //1-based

{

   int \*dist=new int[n+1];

   for(int i=1;i<=n;i++) dist[i]=INT\_MAX;

   dist[start]=0;

   for(int i=1;i<n;i++) //relax

   {

       for(int j=0;j<E.size();j++)

       {

           int u=E[j].u,v=E[j].v,cost=E[j].cost;

           if(dist[u]!=INT\_MAX && dist[v]>dist[u]+cost)

               dist[v]=dist[u]+cost;

       }

   }

   for(int j=0;j<E.size();j++)

       if(dist[E[j].u]!=INT\_MAX && dist[E[j].v]>dist[E[j].u]+E[j].cost)

           return false ; // bad

   return true ; //good

}

**SPFA**

const int MAXN=1010;

struct Edge {

   int v,cost;

   Edge(int \_v=0,int \_cost=0):v(\_v),cost(\_cost){}

};

vector<Edge> E[MAXN]; //MAXN:num of point

bool visited[MAXN];

int cnt[MAXN];

int dist[MAXN];

bool SPFA(int start , int n)

{

   memset(visited,0,sizeof(visited));

   for(int i=1;i<n;i++) dist[i]=INT\_MAX;

   visited[start]=true,dist[start]=0;

   queue<int> que;

   while(!que.empty()) que.pop();

   que.push(start); cnt[start]=1;

   while(!que.empty())

   {

       int u=que.front();

       que.pop();

       visited[u]=false;

       for(int i=0;i<E[u].size();i++)

       {

           int v=E[u][i].v;

           if(dist[u]!=INT\_MAX && dist[v]>dist[u]+E[u][i].cost)

           {

               dist[v]=dist[u]+E[u][i].cost;

               if(!visited[v])

               {

                   visited[v]=true;

                   que.push(v);

                   if(++cnt[v]>n) return false;

               }

           }

       }

   }

   return true;

}

**Disjoin Set**

struct DisjointSet{

   VII p;

   void init(int n){

       for(int i=0;i<=n;i++){p.pb(i);s.pb(1);}

   }

   int Find(int x){

       return x == p[x] ? x : p[x]=Find(p[x]);

   }

   void Union(int x,int y){

       p[Find(x)]=Find(y);

   }

};

**Segment Tree**

struct Node{

   int value;

   Node \*lc,\*rc;

   Node(){value=0;lc=rc=NULL;}

   void pull(){

       value=lc->value+rc->value;

   }

};

int v[N];

Node\* build(int L,int R){

   Node \*node=new Node();

   if(L == R){

       node->value=v[L];

       return node;

   }

   int mid=(L+R)>>1;

   node->lc=build(L,mid);

   node->rc=build(mid+1,R);

   node->pull();

   return node;

}

void modify(Node \*node,int L,int R,int i,int d){

   if(L == R){

       node->value+=d;

       return;

   }

   int mid=(L+R)>>1;

   if(i<=mid)modify(node->lc,L,mid,i,d);

   else modify(node->rc,mid+1,R,i,d);

   node->pull();

}

int query(Node\* node,int L,int R,int ql,int qr){

   if(ql > R || qr < L)return 0;

   if(ql <= L && R <= qr)return node->value;

   int mid=(L+R)>>1;

   return query(node->lc,L,mid,ql,qr)+query(node->rc,mid+1,R,ql,qr);

}

**String**

VII predo(string pattern){

   VII dp;

   dp.resize(pattern.size());dp[0]=0;

   for(int i=1;i<pattern.size();i++){

       dp[i]=dp[i-1];

       while(dp[i] > 0 && pattern[dp[i]] != pattern[i])dp[i]=dp[dp[i]-1];

       if(pattern[dp[i]] == pattern[i])dp[i]++;

   }

   return dp;

}

void KMP(string word,string pattern,VII dp){

   predo(pattern,dp);

   for(int i=0,match=0;i<word.size();i++){

       while(match > 0 && pattern[match] != word[i])match=dp[match-1];

       if(pattern[dp[i]] == word[i])match++;

       if(match == pattern.size()){

           cout << i-pattern.size()1+1 << endl;

           match=dp[match-1];

       }

   }

}

void ZAlgorithm(string word,string pattern){

   int best=0;

   string S=pattern+word;

   VII Z;

   Z.resize(word.size()+pattern.size());Z[0]=0;

   for(int i=1;i<S.size();i++){

       if(best+Z[best] <= i)Z[i]=0;

       else Z[i]=min(Z[i-best],best+Z[best]-i);

       while(S[i+Z[i]] == S[Z[i]])Z[i]++;

       if(i+Z[i] > best+Z[best])best=i;

   }

   for(int i=pattern.size();i<S.size();i++){

       if(Z[i] >= pattern.size())cout << i-pattern.size() << " ";

   }

}

**Number Theory**

LL GCD(LL A,LL B){

   if(B == 0)return A;

   return GCD(B,A%B);

}

LL ExGCD(LL A,LL B,LL X=1,LL Y=0){

   if(A%B == 0)return Y;

   X-=Y\*(A/B);

   return ExGCD(B,A%B,Y,X);

}

LL Power(LL N,LL M){

   if(M == 0)return 1;

   LL k=power(N,M/2);

   if(M%2)return k\*k\*N;

   else return k\*k;

}

LL ExPower(LL N,LL M,LL P){

   if(M == 0)return 1;

   LL k=ExPower(N,M/2,P);

   if(M%2)return k\*k%P\*N%P;

   else return k\*k%P;

}

void BuildPrime(bool prime[],int N){

   for(int i=2;i<N;i++)prime[i]=true;

   for(int i=2;i<N;i++){

       if(prime[i]){for(int j=i\*i;j<N;j+=i)prime[j]=false;}

   }

}

void ExBuildPrime(LL first[],bool prime[],int N){

   for(int i=2;i<N;i++){

       prime[i]=true;

       first[i]=1;

   }

   for(int i=2;i<N;i++){

       if(prime[i]){for(int j=i\*i;j<N;j+=i){

           prime[j]=false;

           if(first[j] == 1)first[j]=i;

       }}

   }

}

VPII FactorDecomposition(int x){

   VPII Ans;

   while(x > 1){

       LL p,e=0;

       if(prime[x] == true)p=x;else p=first[x];

       while(x%p == 0){x/=p;e++;}

       Ans.push\_back(make\_pair(p,e));

   }

   return Ans;

}

LL Euler(LL x){

   VPLL FD=factor\_decomposition(x);

   LL Ans=1;

   for(auto i:FD){

       Ans\*=i.first-1;

       Ans\*=power(i.first,i.second-1);

   }

   return Ans;

}

bool MillerRabin(LL a,LL n){

   if(a == n)return true;

   if(\_\_gcd(a,n) != 1)return false;

   LL s=0,d=n-1;

   LL power\_of\_a;

   while(d%2 == 0){d/=2;s++;}

   power\_of\_a=ExPower(a,d,n);

   if(power\_of\_a == 1)return true;

   for(int i=0;i<s;i++){

       if(power\_of\_a == n-1)return true;

       power\_of\_a=power\_of\_a\*power\_of\_a%n;

   }

   return false;

}

bool PrimeMillerRabin(int n){

   LL a\_set[3]={2,7,61};

   //LL a\_set[7]={2,325,9375,28178,450775,9780504,1795265022};

   if(n == 2 || n == 3)return true;

   if(n <= 1 || n%2 == 0 || n%3 == 0)return false;

   for(int i=0;i<3;i++){

       if(!MillerRabin(a\_set[i],n))return false;

   }

   return true;

}

LL C(LL N,LL M,LL\* memory){

   if(M > N/2)M=N-M;

   if(M == 0)return 1;

   if(memory[N][M] == -1)memory[N][M]=C(N-1,M,memory)+C(N-1,M-1,memory);

   return memory[N][M];

}

**BCC**

int adj[9][9];          // 紀錄邊數，支援多重邊

int visit[9], low[9], t = 0;

int stack[9], top = 0;  // 堆疊

int contract[9];        // 每個點收縮到的點

void DFS(int i, int p)

{

   visit[i] = low[i] = ++t;

   stack[top++] = i;   // push i

   for (int j=0; j<9; ++j)

       if (adj[i][j])

       {

           // tree edge

           if (!visit[j])

               DFS(j, i);

           // tree edge + back edge

           if (!(j == p && adj[i][j] == 1))

               low[i] = min(low[i], low[j]);

       }

   // 形成BCC。

   // i點會是BCC裡面，最早拜訪的點。

   if (visit[i] == low[i])

   {

       //      cout << "block:";

       int j;

       do {

           j = stack[--top];   // pop j

           //          cout << j;

           contract[j] = i;

       } while (i != j);

   }

}

void tarjan()

{

   memset(visit, 0, sizeof(visit));

   t = 0;

   for (int i=0; i<9; ++i)

       if (!visit[i])

           DFS(i, i);

}

**SCC**

bool adj[20][20];   // adjancecy matrix

int visit[20];

int sat[20];        // 紀錄一組變數設定方式

int finish[20], scc[20], n = 0;

void DFS1(int i)

{

   visit[i] = true;

   for (int j=0; j<N+N; ++j)

       if (map[i][j] && !visit[j])

           DFS1(j);

   finish[n++] = i;

}

void DFS2(int i)

{

   scc[i] = n;

   visit[i] = true;

   for (int j=0; j<N+N; ++j)

       if (map[j][i] && !visit[j])

           DFS2(j);

}

void two\_satisfiability()

{

   ......

       // Kosaraju's Algorithm：尋找拓撲順序。

       n = 0;  // 時刻

   memset(visit, false, sizeof(visit));

   for (int i=0; i<N+N; ++i)

       if (!visit[i])

           DFS1(i);

   // Kosaraju's Algorithm：找出強連通分量。

   n = 0;  // 強連通分量的編號

   memset(visit, false, sizeof(visit));

   for (int k=N+N-1; k>=0; --k)

   {

       int i = finish[k];

       if (!visit[i])

           DFS2(i), n++;

   }

   // 檢查是否有解，無解則立即結束。

   for (int i=0; i<N; ++i)

       if (scc[i] == scc[inv(i)])

           return;

   // 找出一組解

   memset(sat, 0, sizeof(sat));

   for (int k=0; k<N+N; ++k)

   {

       int i = finish[k];

       if (!sat[scc[i]])

       {

           sat[scc[i]] = 1;

           sat[scc[not(i)]] = 2;

       }

   }

   // 印出一組解

   for (int i=1; i<N; ++i)

       if (sat[scc[i]] == 1)

           cout << i;

       else /\*if (sat[scc[i]] == 2)\*/

           cout << "not" << i;

}

**2-SAT**

**模型一：兩者（A，B）不能同時取  
　　那麼選擇了A就只能選擇B’，選擇了B就只能選擇A’  
　　連邊A→B’，B→A’  
  
模型二：兩者（A，B）不能同時不取  
　　那麼選擇了A’就只能選擇B，選擇了B’就只能選擇A  
　　連邊A’→B，B’→A  
  
模型三：兩者（A，B）要麼都取，要麼都不取  
　　那麼選擇了A，就只能選擇B，選擇了B就只能選擇A，選擇了A’就只能選擇B’，選擇了B’就只能選擇A’  
　　連邊A→B，B→A，A’→B’，B’→A’  
  
模型四：兩者（A，A’）必取A  
　　那麼，那麼，該怎麼說呢？先說連邊吧。  
　　連邊A’→A**

O(VE)

const int N = 10;   // 變數的數量

bool adj[20][20];   // adjancecy matrix

int visit[20];      // DFS visit record

int sat[20];        // 紀錄一組變數設定方式

// 令A為0，not A為10；B為1，not B為11。

int not(int a) {return a<N ? a+N : a-N;}

/\*

// 另外一種方式：令A為0，not A為1；B為2，not B為3。

int not(int a) {return a&1 ? a : a+1;}

int not(int a) {return a^1;}

\*/

bool dfs\_try(int i)

{

   if (visit[i] == 1 || sat[i] == 1) return true;

   if (visit[i] == 2 || sat[i] == 2) return false;

   visit[i] = 1;

   visit[not(i)] = 2;

   for (int j=0; j<N+N; ++j)

       if (adj[i][j] && !dfs\_try(j))

           return false;

   return true;

}

void dfs\_mark(int i)

{

   if (sat[i] == 1) return;

   sat[i] = 1;

   sat[not(i)] = 2;

   for (int j=0; j<N+N; ++j)

       if (adj[i][j])

           dfs\_mark(j);

}

void two\_satisfiability()

{

   // 輸入clause

   memset(adj, false, sizeof(adj));

   int a, b;

   while (cin >> a >> b)

   {

       map[not(a)][b] = true;

       map[not(b)][a] = true;

   }

   // 找出一組解

   for (int i=0; i<N; ++i)

   {

       memset(visit, 0, sizeof(visit));

       if (dfs\_try(i)) {dfs\_mark(i); continue;}

       memset(visit, 0, sizeof(visit));

       if (dfs\_try(not(i))) {dfs\_mark(not(i)); continue;}

       // 無解則立即結束。

       return;

   }

   // 印出一組解。

   for (int i=1; i<N; ++i)

       if (sat[i] == 1)

           cout << i;

       else /\*if (sat[i] == 2)\*/

           cout << "not" << i;

}

O(V+E)

bool adj[20][20];   // adjancecy matrix

int visit[20];      // DFS visit record

int sat[20];        // 紀錄一組變數設定方式

// 令A為0，not A為10；B為1，not B為11。

int not(int a) {return a<N ? a+N : a-N;}

/\*

// 另外一種方式：令A為0，not A為1；B為2，not B為3。

int not(int a) {return a&1 ? a : a+1;}

int not(int a) {return a^1;}

\*/

bool dfs\_try(int i)

{

   if (visit[i] == 1 || sat[i] == 1) return true;

   if (visit[i] == 2 || sat[i] == 2) return false;

   visit[i] = 1;

   visit[not(i)] = 2;

   for (int j=0; j<N+N; ++j)

       if (adj[i][j] && !dfs\_try(j))

           return false;

   return true;

}

void dfs\_mark(int i)

{

   if (sat[i] == 1) return;

   sat[i] = 1;

   sat[not(i)] = 2;

   for (int j=0; j<N+N; ++j)

       if (adj[i][j])

           dfs\_mark(j);

}

void two\_satisfiability()

{

   // 輸入clause

   memset(adj, false, sizeof(adj));

   int a, b;

   while (cin >> a >> b)

   {

       map[not(a)][b] = true;

       map[not(b)][a] = true;

   }

   // 找出一組解

   for (int i=0; i<N; ++i)

   {

       memset(visit, 0, sizeof(visit));

       if (dfs\_try(i)) {dfs\_mark(i); continue;}

       memset(visit, 0, sizeof(visit));

       if (dfs\_try(not(i))) {dfs\_mark(not(i)); continue;}

       // 無解則立即結束。

       return;

   }

   // 印出一組解。

   for (int i=1; i<N; ++i)

       if (sat[i] == 1)

           cout << i;

       else /\*if (sat[i] == 2)\*/

           cout << "not" << i;

}

**Sort**

void BubbleSort(int\* N,int L){

   for(int i=0;i<L-1;i++){

       for(int j=0;j<L-1-i;j++){

           if(N[j] > N[j+1])swap(N[j],N[j+1]);

       }

   }

}

void InsertionSort(int\* N,int L){

   for(int i=1;i<L;i++){

       int key=N[i],j;

       for(j=i-1;j>=0 && N[j] > key;j--)N[j+1]=N[j];

       N[j+1]=key;

   }

}

void Merge(int\* N,int L,int M){

   int tmp[L],p=0;

   int a,b;

   for(a=0,b=M;a<M && b<L;){

       if(N[a] < N[b]){

           tmp[p++]=N[a];

           a++;

       }

       else{

           tmp[p++]=N[b];

           b++;

       }

   }

   if(a == M)for(int i=b;i<L;i++)tmp[p++]=N[i];

   else for(int i=a;i<M;i++)tmp[p++]=N[i];

   for(int i=0;i<L;i++)N[i]=tmp[i];

}

void MergeSort(int\* N,int L){

   int M=L/2;

   if(L == 1)return;

   MergeSort(N,M);

   MergeSort(N+M,L-M);

   Merge(N,L,M);

}

void shell\_sort(int\* ptr, int len)

{

   int gap = len / 2;

   while(gap)

   {

       for(int i = gap; i < len; ++i, gap /= 2)

       {

           for(int j = i; j >= gap; j-=gap)

           {

               if(ptr[j] > ptr[j-gap])

                   swap(ptr, j, j-gap, gap);

               else

                   break;

           }

       }

   }

   return;

}

void heap\_sort(int\* arr, int len)

{

   heapify(arr, len/2-1, len);

   max\_heap(arr, len);

   return;

}

void heapify(int\* ptr, int now, int last)

{

   if(now >= last/2 || now < 0)

       return;

   sub\_heapify(ptr, now, last);

   heapify(ptr, now-1, last);

   return;

}

void sub\_heapify(int\* ptr, int now, int last)

{

   if(now\*2+2 < last && !(ptr[now] >= ptr[now\*2+1] && ptr[now] >= ptr[now\*2+2]))

   {

       int max = (ptr[now\*2+1] > ptr[now\*2+2]) ? now\*2+1 : now\*2+2;

       swap(ptr, now, max, 1);

       if(max < last/2)

           sub\_heapify(ptr, max, last);

   }

   else if(now\*2+1 < last && ptr[now] < ptr[now\*2+1])

   {

       swap(ptr, now, now\*2+1, 1);

       if(now\*2+1 < last/2)

           sub\_heapify(ptr, now\*2+1, last);

   }

   return;

}

void max\_heap(int\* ptr, int len)

{

   if(len <= 1)

       return;

   swap(ptr, 0, len-1, 2);

   sub\_heapify(ptr, 0, len-1);

   max\_heap(ptr, len-1);

   return;

}

int maxbit(int data[], int n) //辅助函数，求数据的最大位数

{

   int maxData = data[0];      ///< 最大数

   /// 先求出最大数，再求其位数，这样有原先依次每个数判断其位数，稍微优化点。

   for (int i = 1; i < n; ++i)

   {

       if (maxData < data[i])

           maxData = data[i];

   }

   int d = 1;

   int p = 10;

   while (maxData >= p)

   {

       p \*= 10;

       ++d;

   }

   return d;

   /\*    int d = 1; //保存最大的位数

         int p = 10;

         for(int i = 0; i < n; ++i)

         {

         while(data[i] >= p)

         {

         p \*= 10;

         ++d;

         }

         }

         return d;\*/

}

void radixsort(int data[], int n) //基数排序

{

   int d = maxbit(data, n);

   int \*tmp = new int[n];

   int \*count = new int[10]; //计数器

   int i, j, k;

   int radix = 1;

   for(i = 1; i <= d; i++) //进行d次排序

   {

       for(j = 0; j < 10; j++)

           count[j] = 0; //每次分配前清空计数器

       for(j = 0; j < n; j++)

       {

           k = (data[j] / radix) % 10; //统计每个桶中的记录数

           count[k]++;

       }

       for(j = 1; j < 10; j++)

           count[j] = count[j - 1] + count[j]; //将tmp中的位置依次分配给每个桶

       for(j = n - 1; j >= 0; j--) //将所有桶中记录依次收集到tmp中

       {

           k = (data[j] / radix) % 10;

           tmp[count[k] - 1] = data[j];

           count[k]--;

       }

       for(j = 0; j < n; j++) //将临时数组的内容复制到data中

           data[j] = tmp[j];

       radix = radix \* 10;

   }

   delete []tmp;

   delete []count;

}

**Big Number**

const int MAX\_DIGIT = 1000;

const int POSTIONAL = 4;

const int POSTIONAL\_NOTATION = 10000;

struct PositiveBigNum{

   int N[MAX\_DIGIT],L;

   PositiveBigNum():L(0){}

   PositiveBigNum(string S){

       set\_value(S);

   }

   PositiveBigNum(int\* N,int L){

       set\_value(N,L);

   }

   void set\_value(string S){

       L=(S.size()-1)/POSTIONAL+1;

       for(int i=0;i<L;i++)N[i]=0;

       for(int i=0;i\*POSTIONAL<S.size();i++){

           int pow10=1;

           for(int j=i\*POSTIONAL;j<S.size() && j<i\*POSTIONAL+POSTIONAL;j++){

               N[i]+=(S[S.size()-1-j]-48)\*pow10;

               pow10\*=10;

           }

       }

   }

   void set\_value(int\* N,int L){

       this->L=L;

       for(int i=0;i<L;i++)this->N[i]=N[i];

   }

   bool equal\_zero() const {

       if(L == 1 && N[0] == 0)return true;

       return false;

   }

   void kill\_zero(){

       while(L > 1 && N[L-1] == 0)L--;

   }

   int magic(int \*Num,int Length,const PositiveBigNum& A) const {

       PositiveBigNum B(Num,Length);

       B.kill\_zero();

       int Ans=0;

       while(B >= A){

           B=B-A;

           Ans++;

       }

       for(int i=0;i<Length;i++){

           if(i < B.L)Num[i]=B.N[i];

           else Num[i]=0;

       }

       return Ans;

   }

   PositiveBigNum operator+(const PositiveBigNum& A) const {

       const PositiveBigNum &X=(\*this > A ? \*this : A);

       const PositiveBigNum &Y=(\*this <= A ? \*this : A);

       PositiveBigNum tmp=X;

       for(int i=0;i<Y.L;i++)tmp.N[i]+=Y.N[i];

       tmp.N[tmp.L]=0;

       for(int i=0;i<tmp.L;i++){

           tmp.N[i+1]+=tmp.N[i]/POSTIONAL\_NOTATION;

           tmp.N[i]%=POSTIONAL\_NOTATION;

       }

       if(tmp.N[tmp.L] > 0)tmp.L++;

       return tmp;

   }

   PositiveBigNum operator-(const PositiveBigNum& A) const {

       const PositiveBigNum &X=(\*this > A ? \*this : A);

       const PositiveBigNum &Y=(\*this <= A ? \*this : A);

       PositiveBigNum tmp=X;

       for(int i=0;i<Y.L;i++)tmp.N[i]-=Y.N[i];

       for(int i=0;i<tmp.L;i++){

           if(tmp.N[i] < 0){

               tmp.N[i+1]--;

               tmp.N[i]+=POSTIONAL\_NOTATION;

           }

       }

       tmp.kill\_zero();

       return tmp;

   }

   PositiveBigNum operator\*(const PositiveBigNum& A) const {

       PositiveBigNum tmp;

       tmp.L=L+A.L;

       for(int i=0;i<tmp.L;i++)tmp.N[i]=0;

       for(int i=0;i<L;i++){

           for(int j=0;j<A.L;j++)tmp.N[i+j]+=N[i]\*A.N[j];

           for(int j=0;j<tmp.L;j++){

               tmp.N[j+1]+=tmp.N[j]/POSTIONAL\_NOTATION;

               tmp.N[j]%=POSTIONAL\_NOTATION;

           }

       }

       tmp.kill\_zero();

       return tmp;

   }

   PositiveBigNum operator/(const PositiveBigNum& A) const {

       if(\*this < A)return PositiveBigNum(0);

       PositiveBigNum Div,Mod=\*this;

       Div.L=L;

       for(int i=L-1;i>=0;i--)Div.N[i]=magic(Mod.N+i,Mod.L-i,A);

       Div.kill\_zero();

       return Div;

   }

   PositiveBigNum operator%(const PositiveBigNum& A) const {

       if(\*this < A)return \*this;

       PositiveBigNum Mod=\*this;

       for(int i=L-1;i>=0;i--)magic(Mod.N+i,Mod.L-i,A);

       Mod.kill\_zero();

       return Mod;

   }

   bool operator>(const PositiveBigNum& A) const {

       if(L > A.L)return true;

       else if(L < A.L)return false;

       for(int i=L-1;i>=0;i--){

           if(N[i] > A.N[i])return true;

           else if(N[i] < A.N[i])return false;

       }

       return false;

   }

   bool operator>=(const PositiveBigNum& A) const {

       if(L > A.L)return true;

       else if(L < A.L)return false;

       for(int i=L-1;i>=0;i--){

           if(N[i] > A.N[i])return true;

           else if(N[i] < A.N[i])return false;

       }

       return true;

   }

   bool operator<(const PositiveBigNum& A) const {

       return !(\*this >= A);

   }

   bool operator<=(const PositiveBigNum& A) const {

       return !(\*this > A);

   }

   bool operator==(const PositiveBigNum& A) const {

       if(L != A.L)return false;

       for(int i=0;i<L;i++){

           if(N[i] != A.N[i])return false;

       }

       return true;

   }

   PositiveBigNum operator=(const PositiveBigNum& A){

       L=A.L;

       for(int i=0;i<L;i++)N[i]=A.N[i];

       return (\*this);

   }

};

ostream& operator<<(ostream& o,const PositiveBigNum& A){

   o << A.N[A.L-1];

   for(int i=A.L-2;i>=0;i--){

       for(int c=1,tmp=A.N[i];c < POSTIONAL && tmp\*10 < POSTIONAL\_NOTATION;c++,tmp\*=10)o << "0";

       o << A.N[i];

   }

   return o;

}

struct BigNum{

   PositiveBigNum N;

   bool sign;

   BigNum(){}

   BigNum(string S){

       set\_value(S);

   }

   void set\_value(string S){

       if(S[0] == '-'){

           sign=false;

           N.set\_value(S.substr(1,S.size()-1));

       }

       else{

           sign=true;

           N.set\_value(S);

       }

   }

   BigNum operator+(const BigNum& A) const {

       BigNum tmp;

       if(sign^A.sign){

           tmp.N=N-A.N;

           if(N > A.N)tmp.sign=sign;

           else if(N < A.N)tmp.sign=A.sign;

           else tmp.sign=true;

       }

       else{

           tmp.N=N+A.N;

           tmp.sign=sign;

       }

       return tmp;

   }

   BigNum operator-(const BigNum& A) const {

       BigNum tmp=A;

       tmp.sign=!tmp.sign;

       return (\*this + tmp);

   }

   BigNum operator\*(const BigNum& A) const {

       BigNum tmp;

       if(N.equal\_zero() || A.N.equal\_zero())tmp.sign=true;

       else tmp.sign=!(sign^A.sign);

       tmp.N=N\*A.N;

       return tmp;

   }

   BigNum operator/(const BigNum& A) const {

       BigNum tmp;

       if(A.N.equal\_zero())printf("divided by 0 error!!\n");

       else if(N.equal\_zero())tmp.sign=true;

       else tmp.sign=!(sign^A.sign);

       tmp.N=N/A.N;

       return tmp;

   }

   BigNum operator%(const BigNum& A) const {

       BigNum tmp;

       tmp.sign=true;

       tmp.N=N%A.N;

       return tmp;

   }

   bool operator>(const BigNum& A) const {

       if(sign == true && A.sign == true)return (N > A.N);

       if(sign == true && A.sign == false)return true;

       if(sign == false && A.sign == true)return false;

       if(sign == false && A.sign == false)return (N < A.N);

   }

   bool operator>=(const BigNum& A) const {

       if(sign == true && A.sign == true)return (N >= A.N);

       if(sign == true && A.sign == false)return true;

       if(sign == false && A.sign == true)return false;

       if(sign == false && A.sign == false)return (N <= A.N);

   }

   bool operator<(const BigNum& A) const {

       return !(\*this >= A);

   }

   bool operator<=(const BigNum& A) const {

       return !(\*this > A);

   }

   bool operator==(const BigNum& A) const {

       if(sign != A.sign)return false;

       return (N == A.N);

   }

   bool operator!=(const BigNum& A) const {

       return !(\*this == A);

   }

   BigNum operator=(const BigNum &A){

       N=A.N;

       sign=A.sign;

       return (\*this);

   }

};

ostream& operator<<(ostream& o,const BigNum& A){

   if(!A.sign)o << '-';

   return o << A.N;

}