ROLEX SIR 

Data Structures WEEK -9

FOR MORE JOIN TO

https://t.me/rolexsir_2324

ROLEX SIR

Pgno-146:

```
#include <stdio.h>
#include <stdlib.h>

#define SIZE 10

struct node {
   int key;
   struct node* next;
};

struct node* hashTable[SIZE];

int hashFunction(int key) {
   return key % SIZE;
}

void insert(int key) {
   int index = hashFunction(key);
   struct node* newNode = (struct
node*) malloc(sizeof(struct node));
   newNode->key = key;
   newNode->next = NULL;
   if (hashTable[index] == NULL) {
      hashTable[index] = newNode;
   } else {
      struct node* current =
hashTable[index];
      while (current->next != NULL) {
         current = current->next;
      }
      current->next = newNode;
   }
}

void printHashTable() {
   for (int i = 0; i < SIZE; i++) {
      printf("Bucket %d: ", i);
      struct node* current = hashTable[i];
      while (current != NULL) {
         printf("%d -> ", current->key);
         current = current->next;
      }
      printf("NULL\n");
   }
}

int main() {
   for (int i = 0; i < SIZE; i++) {
      hashTable[i] = NULL;
   }
   int input[] = {461, 137, 675, 197, 294,
965, 131};
   int size = sizeof(input) / sizeof(int);
   for (int i = 0; i < size; i++) {
      insert(input[i]);
   }
   printHashTable();
   return 0;
}
```

Pgno-147:

```
#include <stdio.h>
#include <stdlib.h>

#define TABLE_SIZE 11

int hash(int key) {
   return key % TABLE_SIZE;
}

int linear_probe(int key, int i) {
   return (hash(key) + i) %
TABLE_SIZE;
}

void insert(int table[], int key) {
   int i = 0;
   int index = linear_probe(key, i);
   while (table[index] != 0) {
      i++;
      index = linear_probe(key, i);
   }
   table[index] = key;
}

void print_table(int table[]) {
   printf("Hash Table:\n");
   printf("----------------\n");
   for (int i = 0; i < TABLE_SIZE; i++)
{
      printf("%d: %d\n", i, table[i]);
   }
}

int main() {
   int table[TABLE_SIZE] = {0};
   int keys[] = {12, 18, 13, 2, 3, 23, 5,
15};
   int num_keys = sizeof(keys) /
sizeof(keys[0]);
   for (int i = 0; i < num_keys; i++) {
      insert(table, keys[i]);
   }
   print_table(table);
   return 0;
}
```

OUTPUT:

```
Hash Table:
----------------
0: 0
1: 2
2: 12
3: 13
4: 0
5: 5
6: 15
7: 0
8: 18
9: 23
```

10: 3

PGNO-148:

https://www.hackerearth.com/practice/data-structures/hash-tables/basics-of-hash-tables/practice-problems/algorithm/unusual-construction-3ec2e03f/

```
#include <stdio.h>
#include <stdlib.h>

int hash(int a, int b, int M, int i);

int main() {
   int T, N, M, l, r, num, j;
   long long int w, max, *result;

   scanf("%d", &T);
   result = (long long int*)calloc(T,
sizeof(long long int));

   for(int t = 0; t < T; t++) {
      scanf("%d %d", &N, &M);

      int *L = (int*)calloc(M,
sizeof(int));
      int *R = (int*)calloc(M,
sizeof(int));
      long long int *W = (long long
int*)calloc(M, sizeof(long long int));
      long long int *count = (long long
int*)calloc(M, sizeof(long long int));

      for(int i = 0; i < M; i++) {
         scanf("%d %d %lld", &l, &r,
&w);

         num = 0;
         do {
            j = hash(l, r, M, num++);
            if(L[j] == 0) {
               L[j] = l;
               R[j] = r;
            }
            if(l == L[j] && r == R[j]) {
               count[j]++;
               W[j] += w;
               break;
            }
         } while(1);
      }

      max = count[0];
      for(int i = 1; i < M; i++)
         if(count[i] > max)
            max = count[i];

      for(int i = 0; i < M; i++)
         if(count[i] < max)
            result[t] += W[i];

      free(L);
      free(R);
      free(W);
      free(count);
   }
```

```
    for(int t = 0; t < T; t++)
        printf("%lld\n", result[t]);

    free(result);

    return 0;
}

int hash(int a, int b, int M, int i) {
    return (a + b + i) % M;
}
```

PGNO-149:

```
#include <stdio.h>

int main() {
    int n, p, k, count = 0;
    scanf("%d %d %d", &n, &p, &k);
    for(int i = 1; i <= n; i++) {
        for(int j = i+1; j <= n; j++) {
            int a = i % p;
            int b = j % p;
            if((a+a+a+b) % p == k) {
                count++;
            }
        }
    }
    printf("%d\n", count);
    return 0;
}
```

PGNO-151:
https://www.hackerearth.com/practice/data-structures/hash-tables/basics-of-hash-tables/practice-problems/algorithm/t-rex-and-the-pairs-0a045ce2/

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define ll long long
#define ld long double
#define MOD 1000000007
#define den(a) printf("%lld\n",a)
#define des(a) printf("%lld  ",a)
#define pi = 3.14159265
long long ar1[100009];
long long arg[100009];
long long arf[100009];

int cmfn1(const void *a,const void *b);
int cmfn1(const void *a,const void *b)
{
    if((*(long long *)a - *(long long *)b)
> 0)
    {
        return 1;
    }
    else if((*(long long *)a - *(long long
*)b) == 0)
    {
        return 0;
    }
    else
    {
```

```
        return -1;
    }
}
int main()
{
    long long
a,b,c,d,e,f,g,i,j,k,l,m,n,o,p,q,r,si,t,ans;

    scanf("%lld", &n);
    for(i = 1;i<=n;i++)
    {
        scanf("%lld", &ar1[i]);
    }

    for(i = 1;i<=n;i++)
    {
        arf[i] = i*i + ar1[i];
    }
    for(j = 1;j<=n;j++)
    {
        arg[j] = ar1[j] - j*j;
    }

    qsort(arg+1,n,sizeof(long
long),cmfn1);
    qsort(arf+1,n,sizeof(long
long),cmfn1);

    p = arf[1];
    k = 0;
    j = 1;
    ans = 0;
    si = 0;

    for(i = 1;i<=(n+1);i++)
    {
        //des(arf[i]);
        //den(arg[i]);
        if((p != arf[i]) || (i == n+1))
        {
            l = 0;
            f = 0;
            while((j<=n) && (arg[j] <= p))
            {
                if(p == arg[j])
                {
                    l++;
                    f = 1;
                }

                j++;
            }

            ans = ans + l*k;
            p = arf[i];
            k = 1;
        }
        else
        {
            k++;
        }
    }

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

    return 0;
}
```

PGNO-152:

```
#include <stdio.h>

#define TABLE_SIZE 11

int hash(int key) {
   return key % TABLE_SIZE;
}

int probe(int key, int i) {
   return (hash(key) + i*i) %
TABLE_SIZE;
}

int insert(int table[], int key) {
   int i = 0;
   while (i < TABLE_SIZE) {
      int index = probe(key, i);
      if (table[index] == -1) {
         table[index] = key;
         return index;
      }
      i++;
   }
   return -1;
}

void display(int table[]) {
   for (int i = 0; i < TABLE_SIZE; i++)
{
      printf("%d: %d\n", i, table[i]);
   }
}

int main() {
   int table[TABLE_SIZE];
   for (int i = 0; i < TABLE_SIZE; i++)
{
      table[i] = -1;  // initialize table with
-1
   }

   int keys[] = {43, 92, 123, 101, 36, 81,
52, 28};
   int num_keys = sizeof(keys) /
sizeof(int);

   for (int i = 0; i < num_keys; i++) {
      insert(table, keys[i]);
   }

   display(table);

   return 0;
}
```

PGNO-153:

```
#include <stdio.h>
#include <stdlib.h>

#define TABLE_SIZE 11

int hash(int key)
{
```

```
    return key % TABLE_SIZE;
}

int hash2(int key)
{
    return 7 - (key % 7);
}

void insert(int key, int hash_table[])
{
    int index = hash(key);
    int step = hash2(key);
    int i = 1;

    while (hash_table[index] != -1)
    {
        index = (index + i * step) %
TABLE_SIZE;
        i++;
    }

    hash_table[index] = key;
}

int main()
{
    int input[] = {92, 58, 27, 35, 19, 79,
48, 64};
    int hash_table[TABLE_SIZE];
    int i;

    for (i = 0; i < TABLE_SIZE; i++)
    {
        hash_table[i] = -1;
    }

    for (i = 0; i < sizeof(input) /
sizeof(int); i++)
    {
        insert(input[i], hash_table);
    }

    printf("Hash table:\n");

    for (i = 0; i < TABLE_SIZE; i++)
    {
        printf("%d ", hash_table[i]);
    }

    printf("\n");

    return 0;
}
```

PGNO-153:

```
#include <stdio.h>
#include <stdlib.h>
#define TABLE_SIZE 11
int hash(int key)
{
    return key % TABLE_SIZE;
}
int hash2(int key)
{
    return 7 - (key % 7);
}
```

```
void insert(int key, int hash_table[])
{
   int index = hash(key);
   int step = hash2(key);
   int i = 1;

   while (hash_table[index] != -1)
   {
      index = (index + i * step) %
TABLE_SIZE;
      i++;
   }

   hash_table[index] = key;
}

int main()
{
   int input[] = {92, 58, 27, 35, 19, 79,
48, 64};
   int hash_table[TABLE_SIZE];
   int i;

   for (i = 0; i < TABLE_SIZE; i++)
   {
      hash_table[i] = -1;
   }

   for (i = 0; i < sizeof(input) /
sizeof(int); i++)
   {
      insert(input[i], hash_table);
   }

   printf("Hash table:\n");

   for (i = 0; i < TABLE_SIZE; i++)
   {
      printf("%d ", hash_table[i]);
   }

   printf("\n");

   return 0;
}
```

PGN0-154:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

#define ll long long

typedef struct _Node {
   ll key;
   int value;
   struct _Node* next;
} Node;

Node* newNode(ll key, int value) {
   Node* node = (Node*)
malloc(sizeof(Node));
   node->key = key;
   node->value = value;
   node->next = NULL;
   return node;
}
```

```
void insert(Node** head, ll key, int
value) {
    Node* node = newNode(key, value);
    node->next = *head;
    *head = node;
}

int get(Node* head, ll key) {
    while (head != NULL) {
        if (head->key == key) {
            return head->value;
        }
        head = head->next;
    }
    return 0;
}

void clear(Node* head) {
    while (head != NULL) {
        Node* next = head->next;
        free(head);
        head = next;
    }
}

int main() {
    int T;
    scanf("%d", &T);
    while (T--) {
        Node* map1[100000] = { NULL };
        Node* map2[100000] = { NULL };
        ll a, b, c, d, m;
        scanf("%lld%lld%lld%lld%lld",
&a, &b, &c, &d, &m);
        int n;
        scanf("%d", &n);
        ll A[n];
        for (int i = 0; i < n; i++) {
            scanf("%lld", &A[i]);
            A[i] = (A[i] % m + m) % m;
            ll val1 = (A[i] * A[i]) % m;
            val1 = (val1 + m) % m;
            ll val2 = (A[i] * ((A[i] * ((A[i] *
a + b) % m) + c) % m) + d) % m;
            val2 = (val2 + m) % m;
            insert(&map1[val1], val2, 1);
            insert(&map2[val2], val1, 1);
            // printf("%lld:%lld %lld\n",
A[i], val1, val2);
        }
        ll ans = 0;
        for (int i = 0; i < 100000; i++) {
            Node* head = map1[i];
            while (head != NULL) {
                ans += (ll) head->value *
get(map2[i], head->key);
                head = head->next;
            }
        }
        printf("%lld\n", ans);
        for (int i = 0; i < 100000; i++) {
            clear(map1[i]);
            clear(map2[i]);
        }
    }
    return 0;
}
```

PGNO-155:
https://www.hackerearth.com/practice/data-structures/hash-tables/basics-of-hash-tables/practice-problems/algorithm/notebook-pages-dbad75a5/

```
#include <stdio.h>
#include <stdlib.h>
#define pcx putchar_unlocked
#define gcx getchar_unlocked
typedef long int lint;
#define MaxNum  1000001
lint getli()
{
lint n =0;
register int c = gcx();
while(c<'0' || c>'9') c = gcx();
while(c>='0' && c<='9')
{
        n = n * 10 + c-'0';
        c = gcx();
}
return n;
}
void putli(lint n, char lc)
{
if (n==0)
{
        pcx('0'); if(lc) pcx(lc);
}
char s[24]; lint rdi =-1;
while (n)
{
        s[++rdi] = '0' + n % 10;
        n /= 10;
}
while (rdi>=0) pcx(s[rdi--]);
if(lc) pcx(lc);
}
int main ()
{
   lint *dCnt = (lint *) calloc (MaxNum,
sizeof(lint));
   for (lint ni=1; ni<MaxNum; ++ni)
      for(lint mi=ni; mi<MaxNum;
mi+=ni) ++dCnt[mi];
   lint xCnt[241] ={0};
lint N = getli()+1;
   while(--N)
{
      lint X = getli();
      if (dCnt[X]>1) ++xCnt[dCnt[X]];
   }
   lint ans =0;
   for (lint xi=2; xi<=240; ++xi)
      ans += (xCnt[xi]*(xCnt[xi]-1))/2;
   putli(ans, 0);
return 0;
}
```

PGNO-156:
https://www.hackerearth.com/practice/data-structures/hash-tables/basics-of-hash-tables/practice-problems/algorithm/maximum-subarray-sum-of-subarrays-7f33aEfa/

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#define pcx putchar_unlocked
#define gcx getchar_unlocked
#define hBktSz (1<<17)
typedef long int lint;
typedef struct sasNode {
   lint sum;
   struct sasNode* nxt;
} sas_t;

sas_t *hBkt[hBktSz], *gSasPool;
lint gSasPid =0;

void insertSAS (lint X) {
lint bid = X & (hBktSz -1);
sas_t *cNode = hBkt[bid];
for (; cNode; cNode = cNode->nxt )
   if (cNode->sum == X) return;
   cNode = gSasPool + gSasPid++;
   cNode->sum = X;
   cNode->nxt = hBkt[bid];
   hBkt[bid] = cNode;
}

lint getnl() { //Negative
lint n =0; auto neg =0;
register int c = gcx();
if ('-' == c) { neg =1; c = gcx(); }
while(c<'0' || c>'9') c = gcx();
while(c>='0' && c<='9') {
            n = n * 10 + c-'0';
            c = gcx();
}
if(neg) n *= -1;
return n;
}
void putnl (lint li, char lc) { //Negative
if (0 == li) {
            pcx('0'); if(lc) pcx(lc); return;
} else if (li < 0) {
            pcx ('-'); li *= -1;
}
char s[24]; auto idx =-1;
while (li) {
            s[++idx] = '0' + li % 10;
            li /= 10;
}
for (auto jdx=idx; jdx>=0; --jdx)
pcx(s[jdx]);
if(lc) pcx(lc);
}
int main () {
lint N = getnl();
   int NA[N];
   lint ssaSz = N*(N+1)/2;
   gSasPool = (sas_t*) malloc
((ssaSz)*sizeof(sas_t));

   for (lint ni=0; ni<N;) NA[ni++] =
getnl();
   for (lint ai=0, sasi=-1; ai<N; ) {
      lint sasMax =LONG_MIN, sas =0;
      for (lint zi=ai++; zi<N; ) {
         sas += NA[zi++];
         if (sas > sasMax) sasMax = sas;
```

```
            insertSAS(sasMax);
            if (sas < 0) sas =0;
        }
    }
    lint totSAS =0;
    for (lint bid=0; bid<hBktSz; ++bid)
        for (sas_t* cNode=hBkt[bid];
cNode; cNode=cNode->nxt)
            totSAS += cNode->sum;

    putnl(totSAS, 0);
return 0;
}
```

PGNO-157:
https://www.hackerearth.com/practice/data-structures/hash-tables/basics-of-hash-tables/practice-problems/algorithm/suzakus-festivals-14dacd7c/

```
#include<stdio.h>
#include <stdlib.h>
#define pcx putchar_unlocked
#define gcx getchar_unlocked
typedef long int lint;
typedef unsigned int uint;
typedef struct {
char name[12];
lint nhash;
int nl;
uint expd;
} fest_t;

lint getl () { //Positive
lint n =0;
register int c = gcx();
while(c<'0' || c>'9') c = gcx();
while(c>='0' && c<='9') {
n = n * 10 + c-'0';
c = gcx();
}
return n;
}
void putl (lint li, char lc) {//Positive
if (0 == li) {
pcx('0'); if(lc) pcx(lc); return;
}
char s[24]; auto idx =-1;
while (li) {
s[++idx] = '0' + li % 10;
li /= 10;
}
```

```
for (auto jdx=idx; jdx>=0; --jdx)
pcx(s[jdx]);
if(lc) pcx(lc);
}
lint getsx (char *s, int l) { // ISLOWER
register int c = gcx();
lint sl =-1;
while(isalpha(c) && sl<l) {
s[++sl] = (char)c;
c = gcx();
} s[++sl] = '\0';
return sl;
}
inline void putsx (char *s, lint l) {
for (auto ci=0; ci<l; ++ci) pcx(s[ci]);
}
int cmpN (const void *p, const void *q)
{
if ((*(fest_t**)p)->nhash !=
(*(fest_t**)q)->nhash)
return ((*(fest_t**)p)->nhash >
(*(fest_t**)q)->nhash);
else
return ((*(fest_t**)p)->expd <
(*(fest_t**)q)->expd);
}
int cmpE (const void *p, const void *q) {
if ((*(fest_t**)p)->expd !=
(*(fest_t**)q)->expd)
return ((*(fest_t**)p)->expd <
(*(fest_t**)q)->expd);
else
return strcmp((*(fest_t**)p)->name,
(*(fest_t**)q)->name);
}
int main () {
fest_t *FAM = (fest_t*) malloc
(10000*sizeof(fest_t));
fest_t *FAP[10000];
lint T = getl() +1;
while(--T) {
lint N =getl();
for(lint fi=0; fi<N; ++fi) {
FAM[fi].nl = getsx(FAM[fi].name,
sizeof(FAM[fi].name));
FAM[fi].nhash =0;
for (lint ci=0; ci<FAM[fi].nl; ++ci)
```

```
FAM[fi].nhash = FAM[fi].nhash *32 +
FAM[fi].name[ci]-96;
FAM[fi].expd = getl();
FAP[fi] = FAM+fi;
}
qsort (FAP, N, sizeof(fest_t*), cmpN);
lint tail=0;
for(lint head=1, fCnt=1; head<N;
++head) {
if (FAP[tail]->nhash == FAP[head]-
>nhash) {
if (fCnt < 3) {
FAP[tail]->expd += FAP[head]->expd;
++fCnt;
}
} else {
fCnt =1;
strcpy(FAP[++tail]->name, FAP[head]-
>name);
FAP[tail]->nl = FAP[head]->nl;
FAP[tail]->nhash = FAP[head]->nhash;
FAP[tail]->expd = FAP[head]->expd;
}
} N = tail +1;
qsort (FAP, N, sizeof(fest_t*), cmpE);
putsx(FAP[0]->name, FAP[0]->nl);
pcx(' ');
putl(FAP[0]->expd, '\n');
}
return 0;
}
```

PGNO-158:

https://www.hackerearth.com/practice/data-structures/hash-tables/basics-of-hash-tables/practice-problems/algorithm/bob-and-string-easy/

```
#include <stdio.h>

int main()
{
   int t,i,j;
   char h[100001],l[100001];
   scanf("%d",&t);
   for(j=0;j<t;j++)
   {
            int a[27]={0},b[27]={0},c=0;
```

```
        scanf("%s",h);
        scanf("%s",l);
        for(i=0;h[i]!='\0';i++)
        a[h[i]-96]++;
        for(i=0;l[i]!='\0';i++)
        b[l[i]-96]++;
        for(i=1;i<=26;i++)
        {
                if(a[i]!=b[i])
                c=c+abs(a[i]-b[i]);
        }
        printf("%d\n",c);
   }
   return 0;
}
```

PGNO-159:

https://www.hackerearth.com/practice/data-structures/hash-tables/basics-of-hash-tables/practice-problems/algorithm/icpc-team-management/

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
   int cases, N, K, i, j, len, bins[100], flag;
   scanf("%d", &cases);
   int results[cases];
   //printf("cases: %d\n", cases);

   for (i=0; i<cases; i++) {
        flag = 0;
        for (j=0; j<100; j++) {
                bins[j] = 0;
        }

        scanf("%d %d", &N, &K);
   ?/printf("scanned: %d, %d\n", N, K);
                char str[N][100];

                for (j=0; j<N; j++) {

        scanf("%s", str[j]);
        len = strlen(str[j]);
```

```
            //printf("%d\n", len);
                        bins[len] += 1;
                        }

            for (j=0; j<100; j++) {
            if (bins[j] % K != 0) {
            results[i] = 0;
            flag = 1;

            break;
                                    }
                        }
            if (flag == 0) {
            results[i] = 1;
                        }
    }

    for (i=0; i<cases; i++) {
    if (results[i] == 0) {
            printf("Not possible\n");
            }
            else {

            printf("Possible\n");
            }
    }
    return 0;
}
```